AUTOMATIC PARALLEL OCTREE GRID GENERATION SOFTWARE WITH AN EXTENSIBLE SOLVER FRAMEWORK AND A FOCUS ON URBAN SIMULATION

By

Ethan Alan Hereth

Kidambi Sreenivas
Research Professor of Mechanical Engineering
(Chair)

Robert S. Webster
Associate Research Professor of Mechanical Engineering
(Committee Member)

Timothy W. Swafford
Professor Emeritus of Computational Engineering
(Committee Member)

D. Stephen Nichols
Assistant Professor of Aerospace Engineering
(Committee Member)

Steve L. Karman, Jr.
Staff Specialist Pointwise, Inc.
(Committee Member)

AUTOMATIC PARALLEL OCTREE GRID GENERATION SOFTWARE WITH AN EXTENSIBLE SOLVER FRAMEWORK AND A FOCUS ON URBAN SIMULATION

By

Ethan Alan Hereth

A Dissertation Submitted to the Faculty of the University of Tennessee at Chattanooga in Partial Fulfillment of the Requirements of the Degree of Doctor of Philosophy in Computational Engineering

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

December 2016

ABSTRACT

The development of an automatic, dynamic, parallel, Cartesian, linear forest-of-octree grid generator and partial differential equation (PDE) solver framework is presented. This research is bundled into an application programmed with `C++` which uses `MPI` for distributed parallelism. The application is named *paros* which stands for PARallel Octree Solver. In its current implementation, the application provides a 'zeroth' order representation of the target geometry, and as such, no cut-cell algorithm, projection method, or immersed boundary condition are implemented. In this case, 'zeroth' order means that no geometry is ever exactly represented in the final computational mesh: an octree element is either completely in the domain or entirely outside of it. Any element that contains or is intersected by a geometry facet is removed from the final mesh which results in a 'blocky' or 'stepped' geometry representation and simplifies boundary computations. The computational octree mesh creation is completely parallel and automated. The algorithm is dynamic in the sense that it is repartitioned dynamically throughout the grid generation process to maintain optimal load balancing during all phases of the mesh genesis. A linear octree data structure is used to store the octree mesh elements and is leveraged for optimal load balancing. An additional hierarchical octree is used to significantly improve algorithms that suffer from this linear storage paradigm. This work focuses on, but is not limited to, applications related to urban simulations and may be applied to plume/contaminant propagation. Within the PDE solution framework a cell-centered, incompressible, unsteady, Navier-Stokes solver with an energy term to account for thermal buoyancy

is implemented and validated using canonical test cases. Turbulence closure is implemented in the form of the Smagorinski large eddy simulation (LES) model. The parallel grid generation and solution process is tested on a large scale cityscape geometry and shown to be robust and efficient. Additionally, an implementation of the compressible Navier-Stokes equations is coded within the framework. The framework is extensible such that adding other types of numerical PDE solvers should not be difficult. Other features including adaptive mesh refinement (AMR) and contaminant transport functionality are included.

DEDICATION

This research and dissertation is dedicated to my amazing wife: Sarah Dawn Hereth, my two handsome sons: Gabriel Alan and Jude Isaiah, and my two beautiful daughters: Lael Sophia and Emery Dawn. Without their love and patience, I could never have completed this endeavor.

I also wish to dedicate this work to my mother, Brenda Louise Hereth, who left us too early; she never doubted my ability and intelligence, and continually encouraged and pushed me to better myself and pursue knowledge. She would be beyond proud if she were with us today. I miss you Mom.

ACKNOWLEDGEMENTS

This dissertation brings to fruition a long journey that began over nine years ago. Over these years I have learned many things, developed many skills, overcome many obstacles, made many friends, and encountered many mentors. Most of the knowledge obtained and obstacles overcome during this time would not have been possible without these friends and mentors. I would like to acknowledge as many of these instrumental people as I can think of, and I hope that those of you whom I neglect to thank will forgive my forgetfulness.

First, I would like to thank the SimCenter at the University of Tennessee at Chattanooga for allowing me to pursue this degree; special thanks to Dr. Timothy W. Swafford for his good-natured demeanor and guidance, which made the transition to the SimCenter welcoming.

Along the way, many SimCenter faculty helped pave the way to where I am today. I would like to thank Dr. Steve L. Karman, Jr. for his encouragement and mentorship; I would not have developed the interests that have led me here without him. Dr. Daniel G. Hyams and Dr. W. Kyle Anderson both also played instrumental roles early on in my studies at the SimCenter. Dr. Lafayette K. Taylor was consistently encouraging and upbeat; this encouragement, and much sage guidance, helped steer me well on my way.

I must profusely thank my final major adviser, Dr. Kidambi Sreenivas. His mentorship, guidance, wisdom, experience, advice, and help was beyond instrumental in the success of this research, as well as much of my professional/career development. His down-to-earth, good-natured

realism has helped keep me grounded, as well as provided many a much-needed laugh along the way. Thank you Sree. Furthermore, I am also indebted to the other members of my committee, in no particular order: Dr. D. Stephen Nichols, Dr. Robert S. Webster, Dr. Timothy W. Swafford, and Dr. Steve L. Karman, Jr. Most of these gentlemen agreed on very short notice to serve on my committee, and they all provided meaningful feedback in ways of which only they were capable.

At this point I wish to mention another special person who played a very significant role early on in the journey to where I am today. Dr. Clarence O. E. Burg was my major adviser when I began the coursework required for my Masters of Sciences in applied mathematics; at that time I had no idea what I wanted to do with my life, nor studies, and I was simply following the winds of fate, unsure of where they would take me. Dr. Burg saw something in me—I am not sure what—and took me under his wing and mentored and guided me until I began to see my own way. He was always positive and uplifting and I am sure that I would not have started my journey at the SimCenter had it not been for him and his friendship and mentoring. Dr. Burg was on my committee until he passed away unexpectedly (October 7, 1966–March 23, 2016); his absence is felt by many. Rest in peace Dr. Burg; you are missed.

There are a couple other individuals who deserve to be mentioned as well: Dr. Daniel J. Arrigo helped me develop an interest in research during my undergraduate studies; he gave me many opportunities and a lot of guidance during this time. My father, Alan Walter Hereth, who, even though he does not understand anything that I do, has always been positive and encouraging. Thank you Dad.

There are many friends, peers, and colleagues who have been a major part of this odyssey, and to whom I must express my deepest appreciation. I will almost certainly forget to mention

some of you; so please know that if you helped me along my way to this achievement, I appreciate you, even if I do not explicitly mention you! Taylor Erwin is a dear friend and mentor, without his friendship and help I would not have obtained the same level of success as I have today. I would be hard pressed to quantify the skills he helped me develop. Wally Edmondson was an endless source of useful—and sometimes not so useful—information, and was always willing to help in any way. He and Ryan Glasby both helped make my induction to the SimCenter as welcoming and easy as possible. Bruce Hilbert is special kind of friend, and we have been through a lot together; he has always been there for me, both professionally and personally. I will always appreciate his honest, sincere, and realistically Godly approach to life. There are many others who have been great sources of conversation, commiseration, laughs, and brain-storming/debugging sessions as well as been willing to put up with the unfortunately all-too-common kvetch/vent/rant. Among these are David Collao, Anshul Mittal, Nick Currier, Weiyang (Ryan) Lin, Faranak Behzadi, Behrouz Shamsaei, Arash Ghasemi, and Lawton Shoemake. Thank you one and all.

Far be it from me to fail to acknowledge the immense role my wife, and to a lesser extent, my children, have played in my success. This has been a very long journey, full of many late nights and long weekends, and, unfortunately all-too-often, an overly stressed and preoccupied husband/father. Without the gracious patience and understanding that they, especially my wife, extended to me throughout this endeavor, I would not be writing this document today. Thank you very much; I hope it has been worth it. I love you all dearly.

Lastly, I must thank God for giving me the grace and strength necessary to complete this undertaking, and for salvation, freely given, that I can do nothing to deserve. ‘I believe, help my unbelief’ (Mark 9:24).

All figures and plots in this research were generated using Paraview[†] and Gnuplot[‡]. Many of the surface meshes used as geometry were generated using Pointwise[§], and the geometry library used in this code was mainly developed by Dr. Steve L. Karman, Jr.

This research was funded in part by the Tennessee Higher Education Commission (THEC) and the Office of Naval Research under grant N00014-14-1-001 with Dr. Judah H. Milgram and Dr. John F. Kinzer as technical monitors

[†]http://www.paraview.org/
[‡]http://gnuplot.sourceforge.net
[§]http://www.pointwise.com/

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Computational Fluid Dynamics (CFD) has become a mature and diverse field with innumerable and far-reaching applications. One important CFD application relates to the propagation of a plume or contaminant at a cityscape scale. As concerns grow about the release of toxic chemicals or biological agents into urban environments, either purposely or accidentally, there is increasing interest in the role CFD can play in optimizing responses to such events [1–5]. It is valuable to be able to predict where and how a potentially toxic contaminant might spread once released into a city environment in order to provide first responders the means to perform focused disaster mitigation. This type of application does not have to be limited to nefarious events; city planners may be interested in air quality control as it relates to pollution [6–12]. For example, Huber et al. [9] use FLUENT to simulate the events of September 11, 2001 in Manhattan in order to study the transport and dispersion of smoke and particles into the surrounding metropolitan area. Additionally, Hanna et al. [10] compare and contrast the simulations from five different CFD software in the Madison Square Garden area of Manhattan, New York. Four of these CFD models were used to plan the Department of Homeland Security's Urban Dispersion Program MSG05 experiment [13].

Numerous models and software exist that specialize in performing urban simulations. Eichhorn created MISKAM [14, 15], a three-dimensional non-hydrostatic flow model with an Eulerian

dispersion model. Michael Brown at Los Alamos National Laboratory, in collaboration with several universities, developed QUIC: the Quick Urban & Industrial Complex Dispersion Modeling System [16]. QUIC is a fast response urban dispersion model that may run on a laptop; it combines a three-dimensional wind field model, a transport and dispersion model, and a pressure solver into an application capable of running neighborhood scale simulations in minutes. It is also able to simulate chemical, biological, and radiological agent dispersion. The Institute of Meteorology and Climatology at the Leibniz Universität in Hannover, Germany has developed PALM: A PArallelized Large-Eddy Simulation Model for Atmospheric and Oceanic Flows [17, 18] and while it is not designed specifically for urban simulations, it has been used to simulate large swaths of Japan by Kanda et al. [19]. Also, Keck et al. [20, 21] used PALM to simulate the city of Macau China using a 6 km by 5 km computational domain with a minimum spacing of one meter. A finite volume application called CFD-Urban, based on CFD-ACE+ [22], was developed by Coirier et al. [23]. Löhner created FEFLO-CFD [24], Chan and Leach at Lawrence Livermore National Laboratory developed FEM3MP [25]; these are two finite element applications for urban simulation. The Defense Science and Technology Laboratory in the UK created the Urban Dispersion Model (UDM) [26–28]. The Defense Threat Reduction Agency in the United States developed the Urban Hazard Prediction Assessment Capability (Urban HPAC) transport and dispersion model [29, 30].

1.1 Literature Review

This research is mainly focused on the generation of urban scale computational meshes. As such, the particulars of the various approaches these software take with regard to the physics simulation will not be emphasized in this review. What follows is an overview of how some

of the aforementioned software models address the difficulties associated with the generation of the computational grid. PALM [18] uses a Cartesian topography based on the mask method described by Briscolini and Santangelo [31]; the spatial discretization is uniform in the horizontal direction but allows for vertical stretching. PALM uses 2.5 dimensional topological input, e.g. digital elevation model (DEM) data, which means that building geometries are simply represented by building outlines or footprints and an associated height. Using this input data the mesh elements are separated into elements either 100% interior to the mesh (a fluid element) or 100% exterior (an obstacle element); thus, the computational domain comprises three unique sub-domains: elements without boundary faces (interior elements), elements with boundary faces (next to an obstacle) within which wall functions are used, and elements within an obstacle that are excluded from any computations. This implies that buildings are only approximated in the mesh and precludes exact discretization of any building, especially those with holes, overhangs, or any significant three-dimensionality. The computational domain is parallelized by a uniform two-dimensional domain decomposition in the horizontal direction with equally sized sub-domains. MISKAM [14, 15] also uses a Cartesian aligned hexahedral mesh to define its domain and, like PALM, buildings and obstructions are blocked out such that elements are either entirely within or entirely outside the computational mesh, resulting in similarly blocky discretizations. However, unlike PALM, MISKAM allows for three dimensionality, e.g. spires, domes, or sloped roofs and holes/flow-through areas in buildings, e.g. arcades or passageways, as well as anisotropy in all three dimensions. The Windows software, WinMISKAM [32], is a wrapper to the core MISKAM model and provides a graphical user interface (GUI) that enables digitizing of building geometry from bitmaps as well as an interactive meshing interface to aid in the creation of the mesh. QUIC [16] also uses

Cartesian grids to discretize its domain; the mesh is uniform in the horizontal directions but may be non-uniform in the vertical direction. QUIC also allows for a coarser outer grid which may be used to extend the domain well outside of the neighborhood of the building geometry without having to maintain the same refinement in the farfield; it is unclear however, exactly how this is implemented. There are two ways to define the building geometry: via a 'City Builder' GUI interface or using an ESRI ArcGis shapefile [33]; there is support for vegetation and buildings with courtyards as well as experimental support for parking garages and bridges. Like many other similar applications, a QUIC mesh element is either completely in or completely out of the computational domain, which results in non-grid-aligned buildings being stair-stepped. QUIC is one of the few applications—out of those that do not use a pre-built mesh—that support importing terrain data instead of assuming a flat ground; implementation details are unclear, however. CFD-Urban [23] provides two different meshing solutions: unstructured prismatic extrusion and prismatic quadtree based approaches. Cityscape geometry is generated directly from geographical information system (GIS) data, and buildings may be either explicitly resolved in the mesh or implicitly modeled by introducing drag source terms into the equations within cells that would contain buildings. The prismatic quadtree/octree mesh approach, explained by Coirier and Kim [34], creates a single root Cartesian element spanning the domain of interest. This root element is then sub-divided in the horizontal directions only in a quadtree manner until a specified element size is reached within a target region of the domain. A 2:1 balance is enforced to keep the mesh 'smooth' by enforcing that no element is ever more than twice the size of any of its neighboring elements. Upon completion of this quadtree refinement, the root octree cell contains only one element in the vertical direction; these cells are then recursively refined in the vertical direction until the desired number of layers in

the z-direction has been obtained. These layers are subsequently mapped to a vertical clustering in order to properly resolve the buildings and atmospheric boundary layer (ABL). This results in what can be most easily thought of as an extruded quadtree mesh. Using this meshing approach, buildings are modeled implicitly by computing element porosity using the provided GIS data; elements with a porosity under a certain threshold are eliminated from the mesh. CFD-Urban provides optional terrain mapping functionality as well, where the lowest vertical plane in the grid may be projected onto the underlying terrain, which may be provided using several standard digital elevation data formats. Two different element mapping approaches are available: a displacement method or a compression method. The displacement method simply displaces the vertical stack of elements a distance prescribed by the terrain data at the current (x, y) location, whereas the compression method keeps the uppermost vertical plane at its constant height and compresses the elements between the terrain and the upper boundary. Coirier and Kim claim that the displacement method is preferred as the compression method reduces mesh skew angle quality [34]. FEFLO-Urban [24] uses unstructured body-fitted tetrahedral meshes or, for large urban simulation, building geometry is modeled using an embedded boundary approach [35] on unstructured tetrahedral meshes. Camelli et al. [36] describe its use in both modes: a body fitting tetrahedral mesh was used to model the flow and dispersion patterns around Tysons Corner in Fairfax County Virginia, while a much larger area around the Madison Square Garden area of Manhattan was simulated using the embedded boundary paradigm to dramatically reduce the time-to-simulation. FEM3MP [25, 37] offers a 'simplified CFD approach' wherein there is an option to explicitly resolve a select few important buildings, or none at all, and virtually model the remaining buildings with drag forces implemented as sink terms in the mean momentum equations [25]. The underlying mesh is a structured mesh

that may be stretched and distorted and is relatively coarse in regions containing the virtual building geometry. The immersed boundary grid generator TOMMIE [38] generates potentially anisotropic Cartesian meshes from geometry in ASCII Stereo-Lithography format (STL) and has been used in urban scale simulation of downtown Oklahoma City [39,40]. Meteodyn [41] offers the commercial software URBAWIND [42, 43] which claims to provide automatic, flow-aligned mesh generation and refinement within a parallel unstructured multigrid solver framework. Technical details are not readily available since this is a commercial code, but it appears that it uses hierarchical meshes and perhaps some sort of immersed boundary conditions for the building geometries. See Fahssis et al. [44] and Kalmikov et al. [45, 46] for examples of the use of UrbaWind in literature.

1.2 Motivation

There are many difficulties that must be overcome to reliably and accurately perform CFD simulations at the urban environment scale; not the least of which is adequately modeling the cityscape geometry. One of the biggest problems one faces when attempting to simulate a CFD problem at a cityscape scale is the generation of the computational mesh. It is a non-trivial task to accurately model a cityscape geometry in a manner amenable to the generation of quality volume meshes. In fact, manual generation of grids on these types of geometries can easily take weeks—even months. To further exacerbate the problem, viscous meshes adequate to accurately model viscous stresses on solid surfaces at a cityscape scale strains the ability of most grid generation and CFD software in terms of memory as well as CPU requirements. This is the case even in today's high performance computing (HPC) environments.

However, in many cityscape scale simulations, the CFD practitioner may only be interested in the macro-behavior of the flow physics. The micro-behavior of the flow, while it does play a significant role in the macro-behavior, may not be the primary interest of the simulation. This may be the case when considering how a contaminant propagates throughout a city landscape and when the primary interest lies in knowing approximately where a contaminant goes, especially when evacuation considerations are of primary concern. In this case, it is less important that the micro-physics be modeled accurately and more important that the macro-physics are approximately correct. When this is the case, certain liberties may be taken, both with regard to the mesh generation, and the CFD implementation.

Regarding mesh generation, one of the easiest aspects that may be leveraged toward this end is the viscous grid spacing; if the accuracy of the micro-physics may be sacrificed to gain a computational edge, the boundary layer may be under-resolved or not resolved at all. In recent simulations involving Chattanooga, TN and Washington, DC, (see Nichols et al. [47, 48]), the computational mesh had nominal viscous spacing of around one meter on all viscous surfaces. Currier [49] ran multi-physics simulations on Washington, DC using similar grids. In fact, 'viscous' spacings akin to these are quite common in current urban simulations [9, 10, 19, 36, 39].

This research is mainly interested in 'zeroth' order geometric accuracy; this does *not* imply that the spatial and temporal accuracy of the solution algorithms are poor. In fact, the spatial and temporal accuracy are at least first order and up to second order in the current implementation of this research. What 'zeroth' order geometric accuracy *does* mean, is that the geometry may only be approximated in the mesh. The primary goal of the application is to approximate the macro-physics in lieu of resolving the micro-physics. With this in mind, the boundary layer mesh

may be exceedingly under resolved or even non-existent as long as appropriate steps are taken to ensure that viscous flow physics are accounted for approximately. Furthermore, one could even imagine a scenario where the accurate representation of the cityscape geometry could be sacrificed for a 'close-enough' representation. Given these considerations, one attractive solution to the geometrically 'zeroth' order meshing problem is non-conformal, octree style meshes.

Octree style meshes are not a new technology and date back to at least 1984 [50, 51]. Other early work, although technically not tree-based, include that of Quirk [52–54] and Berger [55]. Young et al. [56] and Powell [57] were early contributors to tree style meshing. Karman implemented SPLITFLOW [58, 59], HUGG [60], PHUGG [61], and VoxelMesh [62]. Burstedde et al. created *p4est* [63], and Tu et al. at Carnegie Mellon developed Octor [64]. Cambridge Flow Solutions [65] provides a commercial solution in BOXERMesh [66,67], ITASCA Consulting Group sells KUBRIX Geo [68]. NUMECA International [69] provides a grid generation tool called AutoMesh [70] which includes the octree based software HEXPRESS [71]. This list only names a few octree-based technologies/software; it is nowhere near a comprehensive list, nor is it presented as such. Furthermore, numerous successful applications that solve partial differential equations (PDEs) numerically have implemented native octree style meshing: NASA developed Cart3D [72], Sampath et al. at Georgia Institute of Technology developed Dendro [73], Bungartz et al. created a parallel adaptive Cartesian PDE solver using space-filling curves [74]. Most of the aforementioned software, however, are designed to be geometry conforming and must take appropriate steps to accurately represent the specified geometry using unstructured near body meshes, projection methods, or other techniques.

Optimally parallelizing the octree mesh generation process is a separate problem in and of itself, and several approaches have been taken to achieve this goal. One popular method used to obtain optimally parallelized and load balanced octrees is the use of a 'linear' octree storage paradigm. Linear octrees store only leaf elements of the tree in linear arrays. Space filling curves, like Morton or Z order curves and Hilbert curves, are often used to convert the multi dimensional octree data into one dimensional arrays; in order to obtain ideal parallelization, these arrays are split evenly between all available processors. This results in the ability to have functionally perfect load balancing in terms of elements-per-processor; each processor has the same number of elements plus or minus one. Additionally, where appropriate, weighting schemes may be used to give certain elements more or less importance and the sum of the weights may be distributed evenly among the processors in order to achieve load balancing by a different metric. While linear octrees are extremely attractive from the parallelization point of view, they do present a problem in contrast with their hierarchical counterparts: the ability to traverse the tree hierarchically is lost. In this regard, linear octrees are inferior to hierarchical octrees as tree traversal is significantly slower. Whereas, while hierarchical meshes offer fast traversal, parallelization of these types of data structures is much more difficult. Karman and Betro [61] obtained distributed parallelism on hierarchical octree meshes by uniformly refining a single root octree, or several root octrees, on a single processor until there exists at least as many octants as the target number of parallel processes. In the event that there are exactly the same number of octants as parallel processes, each processor is assigned one of the resulting octants. If the ratio of leaf octants to processors is exactly two or four, each group of eight leaf octants having the same parent is divided between four or two processors respectively, otherwise, METIS [75, 76] is used to distribute the available octants. This is an acceptable way to

generate an initial partitioning of a hierarchical octree data structure, but it does not address nor provide for the all but inevitable load balancing issues that will arise upon further mesh refinement.

While the efficient, automatic, and parallel generation of an octree style computational mesh is a significant part of this research, it is not the only aspect that must be considered. The other major aspect of this work is the application of the resulting mesh to the simulation of large urban environments. Thus, the physics of the problem must be identified and analyzed. At the time of this writing, the National Centers for Environmental Information (NCEI) at the National Oceanic and Atmospheric Administration (NOAA) [77] reports that the windiest location in the U.S. over a 30 year average is Mount Washington, NH with a mean wind speed of 35.1 mph. However, the windiest cities reported have much lower 30 year means; the top three cities are Dodge City, KS, Amarillo, TX, and Cheyenne, WY with mean wind speeds of 13.9, 13.5, and 12.9 mph respectively. These data were last updated in 2008 [78]. This type of low speed fluid flow can be considered functionally incompressible, however, buoyancy effects should be considered because pavement, concrete, and buildings can be significantly hotter than the air surrounding them. Turbulence must also be taken into consideration and an appropriate turbulence model chosen that adequately resolves the physics without introducing excessive overhead. Therefore, the goal of this research is to develop a parallel, geometrically zeroth order, incompressible Navier-Stokes (NS) solver using an automatically generated Cartesian octree grid that approximates the urban geometry as discussed throughout this introduction. This application takes advantage of the open source software *p4est* [63], which provides a framework for parallel octree grid generation upon which a parallel Navier-Stokes solver will be built.

1.3 Outline

The remainder of this dissertation is organized as follows. In Chapter 2 the techniques, algorithms, and design choices that were considered and implemented during the development of the automatic, parallel, linear forest-of-octree grid generator are explained and demonstrated. In Chapter 3 the governing equations are presented and the solution methodology explained. In Chapter 4 various validation cases are presented in order to instill confidence in the quality of the implementation of the research. Furthermore, several large-scale urban demonstration cases are shown in order to exhibit the practical applicability of the code. Finally, in Chapter 5, the research is summarized and recommendations for future work are given.

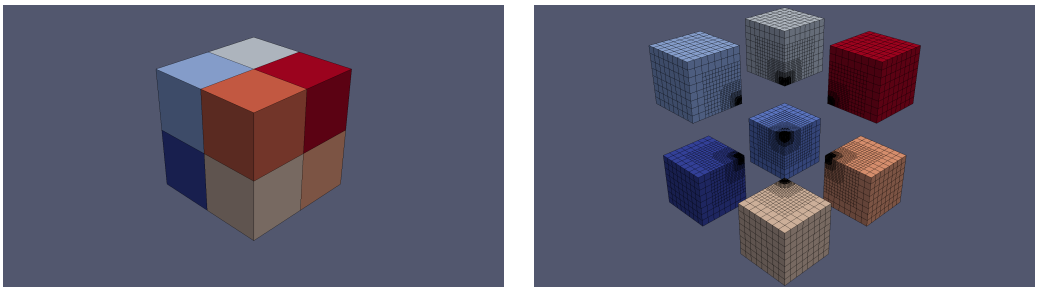
CHAPTER 2

GRID GENERATION METHODOLOGY

This chapter explains the methods used to automatically generate the parallel octree grid and discusses various aspects of the procedure that pose difficulties or present significant benefits in the overall design process.

2.1 Grid Generation With *p4est*

The open source *p4est* library [63] is employed in this project as the framework for the parallel octree grid generation tool. The *p4est* library generates a 'forest-of-octrees' mesh by splitting the computational domain up into multiple single octrees via a provided connectivity deemed the macromesh or macrostructure. Each of the macroelements in the macromesh is itself a single octree, deemed the micromesh or microstructure, and may be arbitrarily refined independently of its neighboring octree elements. Figure 2.1 demonstrates what a simple macromesh comprising eight trees might look like, as well as an example of a corresponding micromesh. The macromesh, or connectivity, is allowed to take on almost any form as long as each macroface and macroedge common between two neighboring macroelements is shared completely or not at all; this implies that any structured mesh may be used as a connectivity for the *p4est*. In this way, *p4est* allows for very general connectivities, providing functionality for periodicities and rotations in the connectivity and even degeneracies, e.g. a hexahedron with a collapsed face. This generality makes it possible



(a) Macromesh (b) Micromesh

Figure 2.1 An example of a macromesh, colored by tree index, and an associated micromesh

to create quadtree representations of, e.g. a Möbius strip or a Klein bottle in two dimensions, and octree representations of a torus or sphere in three dimensions. A *p4est* may also be created out of a tetrahedral mesh by converting each tetrahedron into four octrees by connecting edge midpoints to face midpoints to the centroid of the element. This macrostructure defining the connectivity of the forest is shared among all parallel processes; however, the micromesh is dynamic and completely distributed in parallel. When an octree contains elements at various refinement levels, neighboring elements may vary drastically in size; *p4est* does not restrict size relations between neighboring microelements, but does provide balancing functions to enforce 1:1 and 2:1 size ratios. This balancing feature is important as most PDE solvers capable of operating on octree meshes require a 2:1 balanced mesh. Only leaf octants or microelements are stored in the *p4est* and are given a linear ordering that corresponds to an extension of the space filling z-shaped/Morton curve [79] that spans all trees in the forest [63]. This distinction concerning the octree storage paradigm is important and leads to some very distinct advantages, as well as some disadvantages which will be discussed later. The parallel partitioning of the forest is trivially accomplished by simply dividing the linear array of octants into p segments, where p corresponds to the number of parallel partitions; this results in an

optimal partitioning such that every process owns the exact same number of octants plus or minus one. The partitioning may also be weighted to give certain elements more importance/weight during the partitioning. The *p4est* library is written in the `C` programming language designed with distributed parallelism in mind using `MPI`, and provides the following high level functionality discussed at length by Burstedde et al. [63]:

- ***New***: Create a new forest.
- ***Refine***: Refine the forest based on user defined callbacks.
- ***Coarsen***: Coarsen the forest based on user defined callbacks.
- ***Partition***: Rebalance the parallel partition.
- ***Balance***: Enforce 2:1 size ratios throughout the forest.
- ***Ghost***: Create inter-process ghost elements.
- ***Nodes***: Create unique global node numbers.
- ***Checksum***: Compute a partition-independent forest checksum.

Note that ***New***, ***Refine***, and ***Coarsen*** are processor local and do not require communication, whereas the remaining functions require varying degrees of parallel communication.

2.1.1 Connectivity/Macromesh

For the current project, a connectivity spanning the computational domain must be provided to *p4est* by the user. However, only very simple connectivities are required for this application

and, as such, none of the exceedingly generic connectivities discussed in the previous section are ever required. The only connectivities necessary for this project are very simple structured connectivities which may be provided in a few different ways: by providing a structured 'skeleton' mesh, a bounding box around the geometry, or explicitly specifying the forest dimensions. The bounding box approach is the main *modus operandi* and results in isotropic elements only; however, manually specifying the forest dimensions gives finer control over the number of trees in the forest and additionally allows for anisotropic mesh generation. This manual approach simply takes the bounding box extracted from the geometry and splits it into the desired number of trees in each of the coordinate directions. Lastly, supplying a structured mesh as a 'skeleton' for *p4est* is a useful way to provide the connectivity when the geometry is planar and Cartesian aligned. Using a structured mesh in this way guarantees that the geometry is exactly represented in the final grid and is useful mostly for simple test cases like a flat plate, cavity cases, step cases, etc. when the geometry must not be approximated and anisotropic grids are required. Grids generated in this way take as input a structured ASCII AFLR3 UGRID [80] mesh; each hexahedron in the UGRID mesh becomes a single octree in the forest-of-octrees grid. Internally, the UGRID mesh is converted into the connectivity required by *p4est* and defines the computational domain. These structured UGRID files may be anisotropic but **must** be planar and Cartesian aligned. In this manner, **all** elements in the connectivity will exist in the final octree mesh.

In cases where it is not possible to use a planar, Cartesian aligned skeleton mesh for the connectivity, i.e., for most real cases, bounding box approaches must be used. These methods require that the discretized watertight geometry be provided via an Xpatch [81] facet file. However, any watertight geometry representation could in theory be used if the geometry library was extended

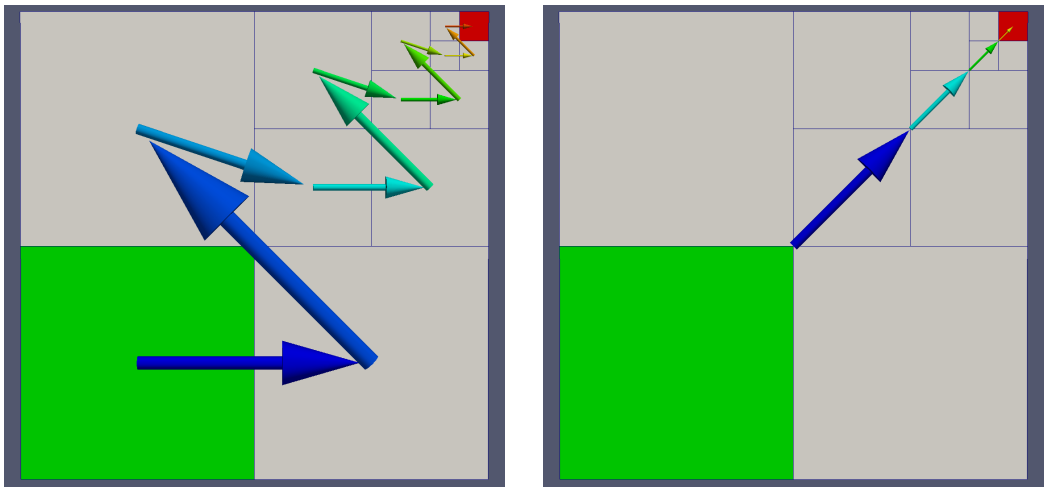
to support it. The bounding box is extracted from the provided geometry representation and is subsequently subdivided into some number of isotropic or anisotropic hexahedra to create a 'brick' connectivity for *p4est*. When isotropic elements are desired, the bounding box is tweaked/stretched in any of the three directions if necessary such that the resulting connectivity will be isotropic. It should be noted, though, that the bounding box is **never** shrunk to guarantee isotropy; it is only stretched. This assures that the provided geometry is always present in the resulting octree grid. However, if isotropic elements are not necessary, the generation of the connectivity is easier; the desired forest dimensions may be specified at runtime, and the bounding box is simply subdivided into said dimensions.

The main disadvantage of these bounding box approaches is that the geometry must be provided as a water-tight discretization; for cityscape geometries, this may pose a significant problem. One way that this could be significantly alleviated is by extracting the geometry from available geographic information system (GIS) data like digital elevation model (DEM), triangulated irregular network (TIN), shapefile, keyhole markup language (KML), or GeoTIFF data, to name just a few. PALM [17, 18], QUIC [16], and MISKAM [14, 15] all provide functionality to generate their grids this way. One disadvantage of this approach, however, is that significant building three-dimensionality may not be properly represented in these formats, so buildings with curved surfaces or complicated architecture may be difficult or impossible to represent or even approximate.

2.1.2 Refinement of the Micromesh

Once the brick connectivity has been created using the bounding box, the forest-of-octrees mesh must be refined to capture the geometry. This refinement is achieved by iterating over the forest

using the refinement function provided by *p4est*, which itself calls a custom geometry refinement callback function. For each element in the forest, the callback function determines if the element contains or is intersected by any of the geometry facets. If so, the element is refined *if* it has not yet reached the maximum allowed refinement level defined by the user. The approach *p4est* employs to achieve this is different from what is normally used to refine an octree since the *p4est* grid is stored linearly, i.e., only octree leaves are stored, and no information about ancestors/descendants is kept. The more traditional approach, which takes advantage of the hierarchical octree data structure, is to pass facets one at a time down the tree hierarchically to determine which element(s) contain them; this hierarchical traversal is optimal for searching an octree structure but is difficult to parallelize. On the other hand, linear octree data structures are trivial to parallelize by dividing the linear array of leaf octants among the available processors. This functionality comes, however, at the cost of a less efficient, more complicated traversal [82]. Examples of this are shown in Figures 2.2 and 2.3, demonstrating the worst case scenario. In this simple example, the red element is the traversal target, and the green element is the first element in the linear storage. Hierarchical traversal takes 5 steps to reach the target element for both the quadtree and octree structures, whereas linear traversal takes 12 steps for the quadtree structure and 28 steps for the octree structure, literally visiting every element in the tree before finding the desired element. Note that this figure also clearly illustrates the origin of the term 'z-curve', often associated with space-filling Morton curves. A consequence of this linear storage is that as the *p4est* refinement function traverses the tree it must search the *entire* geometry facet list *for each* element it finds in the tree in order to determine whether any of the facets intersect the current octree element. A hierarchical octree has only to search decreasing sets of facets for each element it visits as the hierarchical nature of the octree provides means to

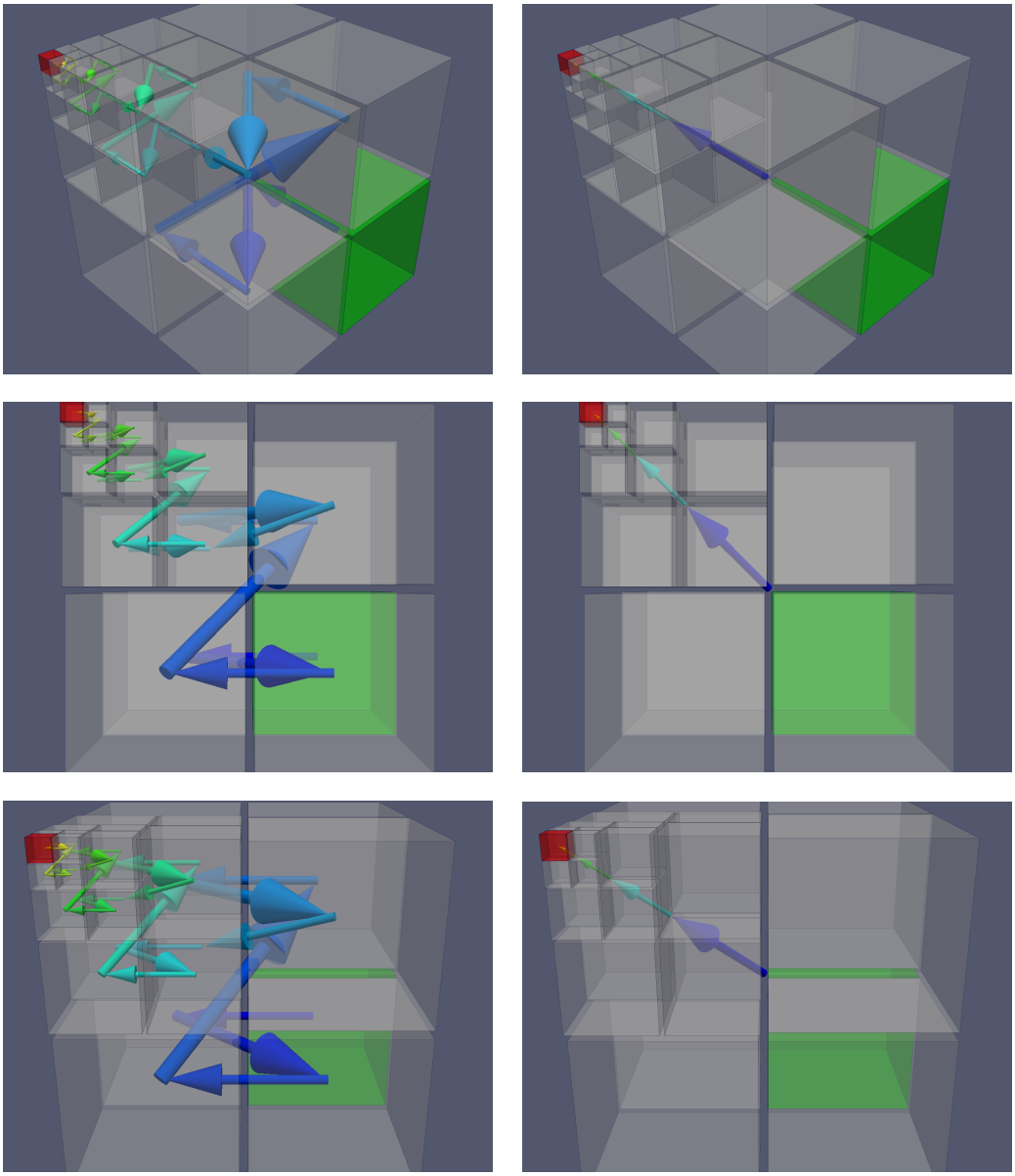


(a) Linear traversal (b) Hierarchical traversal

Figure 2.2 Differences between linear (a) and hierarchical (b) quadtree traversal

eliminate subsets of facets that are not contained by 'parent' elements. The *p4est* approach is clearly sub-optimal and even prohibitive for large geometries. There are, however, a few things that can be done to make this inefficiency less problematic. One involves dynamic repartitioning during the refinement process, and the other uses a temporary hierarchical octree to store geometry facets for rapid retrieval. These optimizations are explained below.

There are two refinement modes in *p4est*: recursive and non-recursive. The recursive mode will refine an element and also consider the eight newly created children elements for refinement recursively, whereas the non-recursive mode will replace an element with its eight children elements but will *not* consider any of the newly created elements for refinement. Each of these approaches have certain advantages, but the non-recursive mode is preferred for geometry refinement for one important reason: it allows for dynamic re-partitioning after each single non-recursive refinement pass. This eliminates potentially large load imbalances caused by certain processors having refined



(a) Linear traversal (b) Hierarchical traversal

Figure 2.3 Differences between linear (a) and hierarchical (b) octree traversal

their portions of the forest more than others. Experience during this research shows that this non-recursive approach can prove to be more than an order of magnitude faster than its recursive alternative during geometry refinement.

This application further alleviates the octree traversal problem by creating a single, relatively coarse, hierarchical octree that stores the entire geometry. This intermediate octree may be used to quickly generate a list of facets contained in a specific spatial area. The *p4est* refinement callback uses this hierarchical geometry octree each time it visits a new element to quickly generate a list of facets spatially local to the current element, thus almost minimizing the number of facets it must query for intersection. This procedure could be improved by creating subsets of processor local facets, so that no processor ever needs to process facets outside of its spatial partition.

Elements containing or intersected by elements of the geometry will be refined provided that they have not already reached the maximum octree level allowed. The software provides another way to specify a refinement stopping criterion for finer control over this refinement process. This approach causes the refinement to be based on the size of the current octree element as it compares to the size of the intersecting geometry elements. It may be undesirable for the octree to refine further in regions where the octree elements are already sized commensurately with the surrounding geometry facets. The user may specify a factor that scales the ratio of the average dimensions of both the octree element and geometry facets to give maximum control over this process. This logic may also be disabled such that any intersected octree element will be refined down to the maximum allowable level regardless of the relative size of the nearby geometry facets.

There are situations when parts of the geometry that are important for water tightness, extent definition, etc., are not necessarily important for use during mesh refinement. One specific example

of this is farfield and sometimes ground boundaries; while these are important for the definition of the geometry/computational domain, it is undesirable to have mesh refinement in these areas, or at least undesirable to have the same level of refinement in these areas as in areas near other solid boundaries. This issue has been addressed herein by the addition of a boundary attribute specifying that certain boundaries are not to be used for mesh refinement. The attribute is called 'norefine' and any boundary marked as a 'norefine' boundary will be ignored during the geometry refinement callback execution, which ensures that mesh refinement will not happen in areas of the domain where it is undesirable.

2.1.3 Quality Control

Once the refinement callback has completed, the forest-of-octrees has been refined down to a maximum allowable level anywhere it is intersected by geometry facets. This refined forest is of low quality as the 2:1 balance requirement is not yet enforced. Figure 2.4 demonstrates the nature of this low quality; notice that near the spherical body, element sizes vary dramatically over short distances, and many elements have neighbors whose level in the octrees differ by more than one. This is an unbalanced octree, and the next step is to balance the refined forest; *p4est* also provides a balancing function, and this function provides all necessary logic required to enforce the specified balance throughout the forest in parallel. The balancing algorithms use non-iterative communication afforded by the concept of the 'insulation layer' to optimally balance the forest across processor boundaries; this approach and associated algorithms are explained in detail by Sundar et al. [83]. When the 2:1 balancing *p4est* function has completed, the *p4est* mesh is now refined and balanced wherever it is near geometry. This mesh is of much higher quality and is

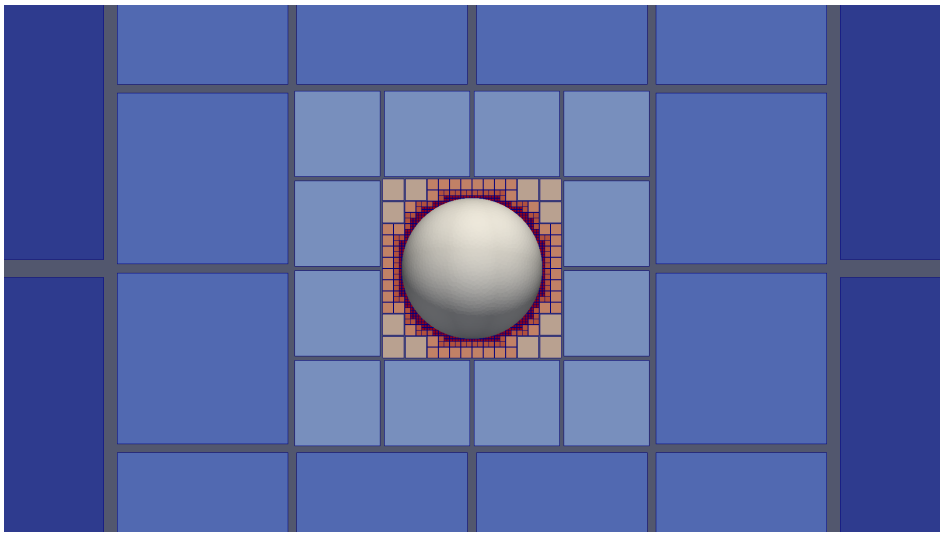


Figure 2.4 Poor quality mesh without 2:1 balance after geometry refinement

now technically a valid mesh for most finite volume Navier-Stokes solvers. Figure 2.5 shows a 2:1 balanced mesh. The main problem at this point is that the elements change size very rapidly as they grow away from the geometry surfaces; every time an element has a neighbor at a different level than its own, their volume ratio is eight. As such, it is desirable to control the number layers of uniformly sized elements that must exist at a certain level before the refinement level may change. The application requires the user to specify this number of layers—deemed buffer layers—which allows for finer control over the gradation of element sizes. These buffer regions make for a higher quality mesh. Note, because of the way octree refinement creates two layers at a time, precise control over these buffer regions is not possible. For example, if there is only one layer of elements at level x and the user specifies a two-layer buffer, the elements adjacent to the elements at level x will be refined causing there to be three layers of elements at level x. This is the nature of octree meshes and cannot be overcome. Quality control is implemented by looping over the balanced

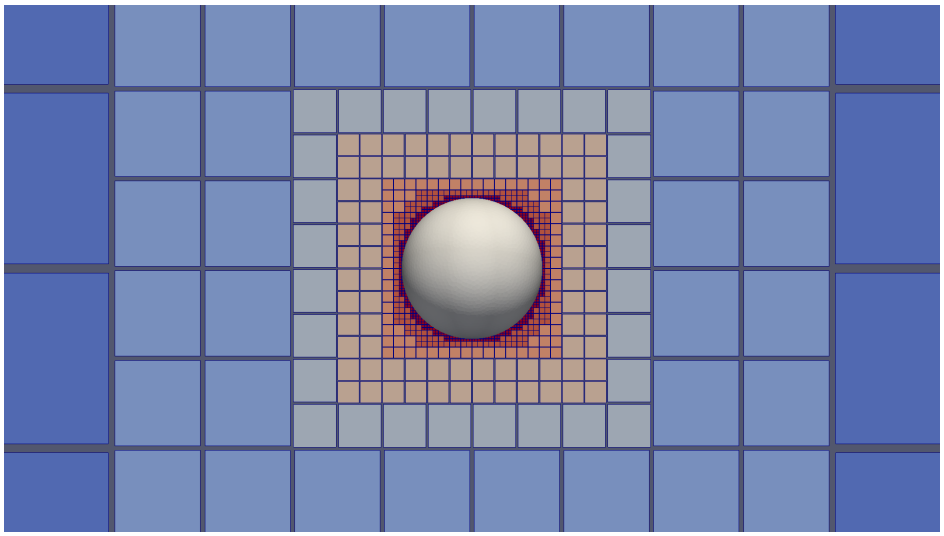
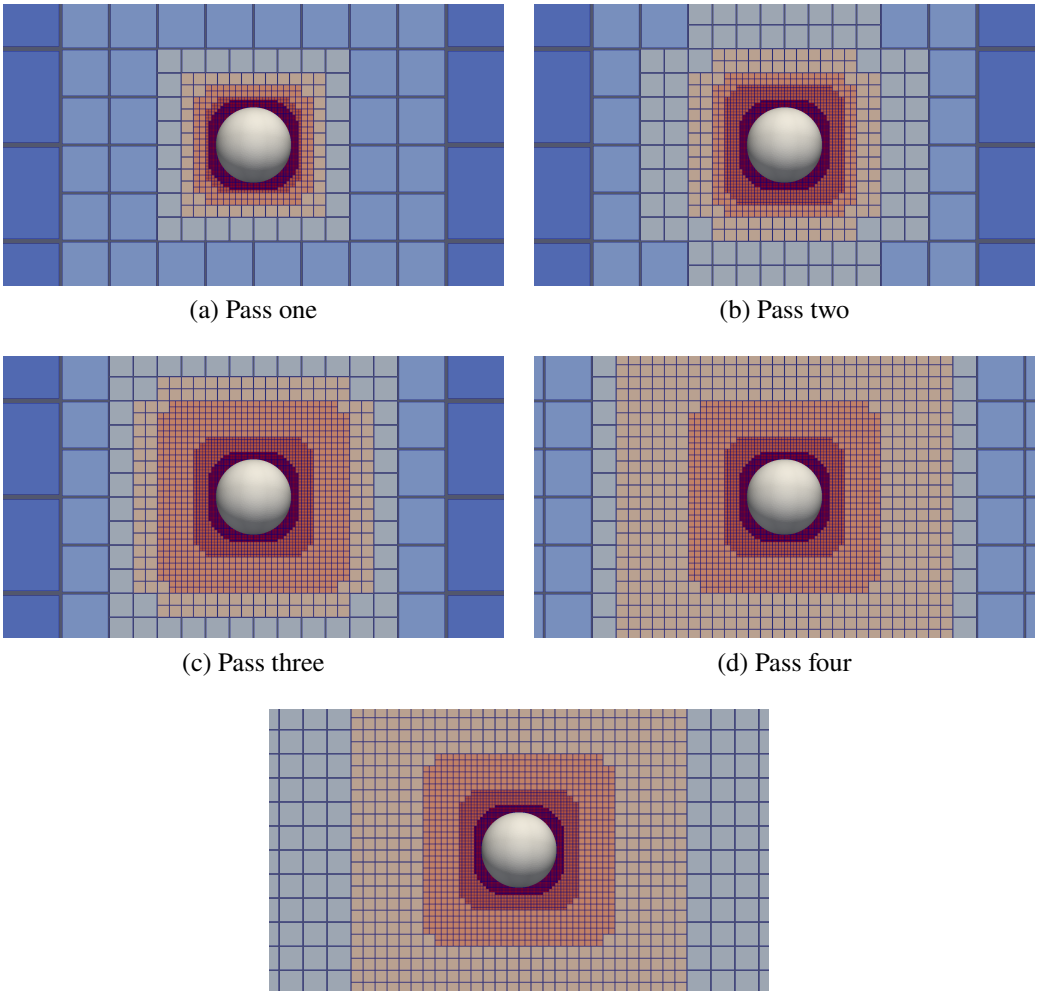


Figure 2.5 A 2:1 balanced mesh without buffer layers

forest from the deepest level up, i.e. from the bottom up[†]. During the loop, bounding boxes are generated around each element at the current level and the dimensions of each bounding box are extended by an appropriate factor to take into account the buffer region. For example, if a buffer region of four layers has been specified, the dimensions of all bounding boxes are scaled by a factor of eight so that they overlap all elements within four body lengths in all directions of each element at the current level. The bounding boxes are then passed to the *p4est* refinement function using a custom callback function, which enforces that each element overlapping any bounding box is refined down to the target level. After each pass through the mesh, the 2:1 balance constraint must be enforced for proper buffer region propagation. This quality refinement continues until the entire mesh has been traversed from the bottom up, at which time the computational mesh is complete.

[†]As a consequence of the linear storage of the mesh elements, this 'bottom up' traversal is implemented as repeated loops over the forest, each subsequent loop targets elements at a higher level.

Figure 2.6 demonstrates how this buffer refinement process modifies the grid at each pass and results in a higher quality final mesh.



(a) Pass one

(b) Pass two

(c) Pass three

(d) Pass four

(e) Pass five

Figure 2.6 An example of buffer control with a six element buffer

Support for areas of custom refinement has also been implemented. Two different types of custom refinement are supported: bounding boxes and baffles. Bounding box custom refinement

allows for a user to specify that the region in space bounded by a given box should be at a user-specified refinement level. Baffle custom refinement has been implemented, such that a user may input any arbitrarily shaped geometry in space—not just a plane—and any elements intersecting said geometry will be refined to a specified level. This allows for finer control over refinement in wake regions or other regions away from geometry.

2.1.4 Mesh Extraction and Boundary Recovery

Now that the grid has been generated, boundaries must be found and marked. This 'marking' process is one of the last phases of the grid generation process. There are two ways the marking may occur depending on which mode is used to generate the *p4est* octree mesh. If a skeleton UGRID mesh is provided, there are no elements cut by geometry. Thus, faces must be marked. However, if a bounding box is used, there will be mesh elements intersected by geometry facets, thus elements must be marked as either *in* or *out* of the geometry, or *cut* by it. Since the latter approach uses a bounding box to define the computational domain, it is highly likely that some resulting elements in the final octree discretization are *outside* of the domain of interest. For this reason, one must be able to query whether a certain element is in or out of the geometry in order to determine which elements in the computational mesh are actually inside the geometry and, thus, are a part of the computational domain. This querying logic is provided by the library handling the geometry via a ray-tracing algorithm[†]. Another *p4est* callback is used to mark either the mesh elements as *cut*, if the geometry is not Cartesian aligned and provided by an Xpatch [81] facet file, or their faces, if a

[†]The geometry library that provides this ray-tracing functionality is an in-house library developed by Dr. Steve L. Karman Jr. at the University of Tennessee at Chattanooga SimCenter.

Cartesian geometry is provided via a UGRID [80] structured skeleton mesh. This callback works much like the geometry refinement callback. It iterates over the *p4est* mesh elements and, using the temporary hierarchical geometry octree, determines which facets, if any, are spatially local to the element. If the method finds that the element contains or is intersected by any geometry facet or facets, that element is marked as *cut*. This marking process is slightly different, and actually easier, when a UGRID skeleton file has been provided. In this mode **all** elements are interior to the mesh, so no element needs to be marked as *cut*. Also, boundary elements are found, and their exposed faces are marked appropriately using tags from the adjacent geometry facet(s).

At this point, the *p4est* forest-of-octrees structure is complete, and a mesh object must be extracted. This mesh provides element-to-element connectivity information, which is not explicitly available from the native *p4est* data. Functionality is provided via the *p4est* API to loop through the mesh while providing necessary information about all neighbors (e.g. the neighbor's level, metrics, etc.). It is important to note that this mesh contains **all** elements in the *p4est* forest-of-octrees, even elements that are *outside* of the geometry. So, this algorithm requires that the elements be given a 'status', i.e., they are marked as either part of the computational mesh or not part of it. There are four valid statuses for elements in the *p4est* mesh: *in*, *out*, *cut*, or *boundary*. The *boundary* status is the only status that might need explaining. It marks elements that contain boundary faces; these elements are also *in*, but they are either next to elements marked *cut*, in the case of a mesh generated using a bounding box, or one or more of their faces have no neighbors, in the case of a mesh generated using a UGRID skeleton connectivity. Element statuses are generated via a flood-fill algorithm. This procedure starts by iterating over the *p4est* mesh until an element with an *unknown* status is located. The actual status of this element is then determined by querying whether

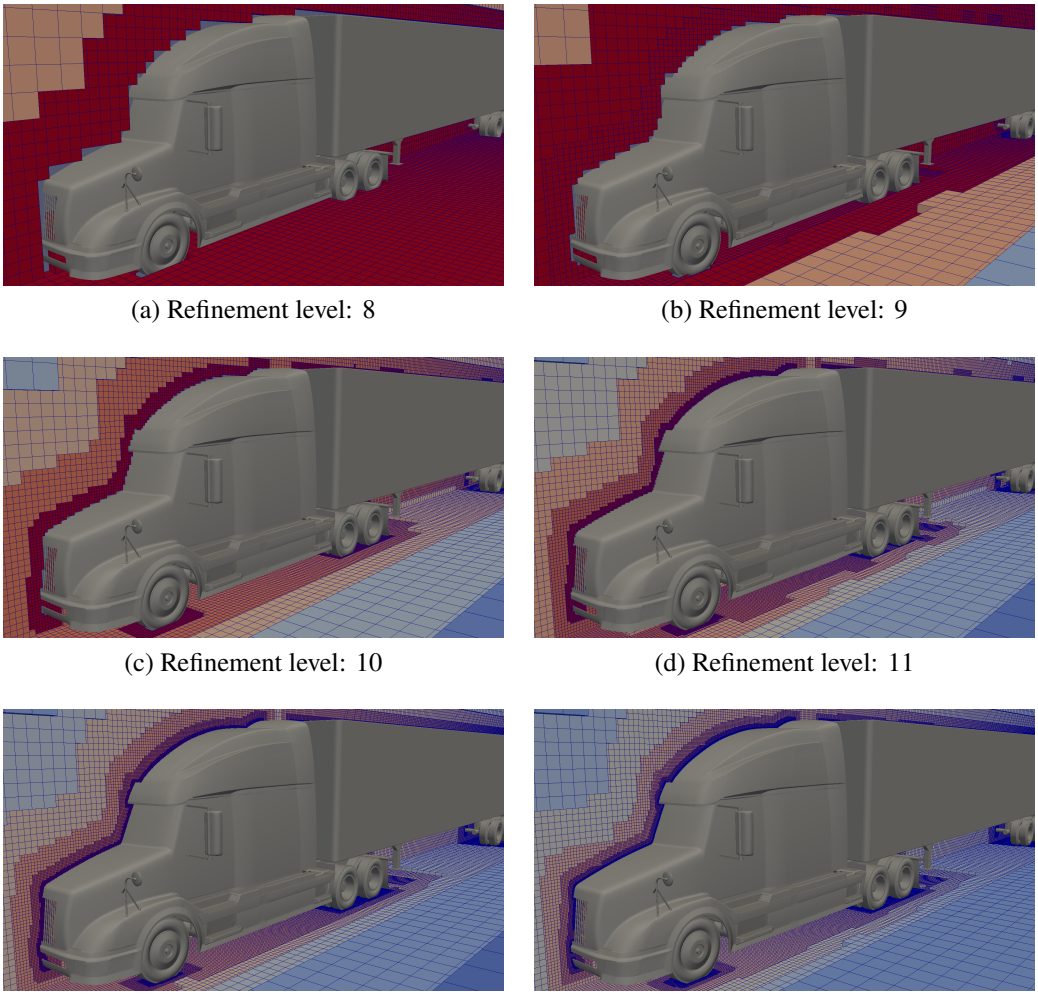
the element is *in* or *out* of the geometry. Note that the only possible statuses in the mesh at this point are either *cut* or *unknown*; all *unknown* elements should be delimited by *cut* elements. Then the algorithm 'floods' through this element's neighbors and its neighbor's neighbors etc., marking their statuses the same as its own and stopping only when it finds elements with *cut* statuses. This flood-fill pass may occur multiple times, depending on the geometry, until all elements have one of the four valid statuses. Also, *boundary* elements are marked during this pass. When an element is adjacent to a *cut* element and it would normally be assigned an *in* status, it is marked instead as a 'boundary' element. At this point, it is important to note that this flood-fill algorithm poses a load-balancing challenge as the amount and locality of work depends highly on the geometry. This is a difficult challenge to overcome and will be explored in more depth later.

Now that the mesh has been generated it is important to note that the manner in which the forest is partitioned needs to change for optimal solver load balancing. Before this point, during the mesh generation, it usually makes sense for each processor to have an equal number of mesh elements, and this uniform partitioning be maintained dynamically as much as possible. This element balancing is especially important during parts of the algorithm when many elements might be added to the mesh. Now, however, load balancing must be obtained by a different metric. At this point, the simulation is beginning, and only elements contained in the computational mesh, i.e. elements with *in* or *boundary* statuses, have any work associated with them. For this reason the partitioning approach now only gives weight to elements with these *active* statuses. With this partitioning scheme, processors may have dramatically different total element counts, but as long as they own equal numbers of *active* elements, load balancing is achieved because *inactive* elements are simply ignored in the solver.

2.2 Approximate Nature of Resulting Computational Mesh

One important distinction that must be made about the computational meshes that are produced during this grid generation process is that the geometry is **never** exactly represented in the final mesh, i.e., this algorithm does **not** implement a cut-cell, nor projection method. Any element in the forest that contains or is intersected by any part of the geometry is **eliminated** from the computational mesh. This procedure is different from grids that use a cut-cell or projection method, which allows for exact representation of any geometry. This accurate geometry representation is accomplished using a cut-cell method by modifying any element that is cut by geometry so that the boundary facet is included in the mesh. This approach obviously creates a mesh that contains more than just hexahedral elements near boundaries and has the potential to create poor quality 'sliver' or 'small' elements as well [84, 85]. Projection methods are different from cut-cell methods, however, they tend to exhibit similar difficulties. Since one of the goals of this project is to quickly generate 'zeroth' order solutions on cityscape geometries, it was decided that exact geometry representation was not necessary nor desirable. Also, even highly curved geometries may be fairly accurately approximated by allowing the forest to refine down to very low levels. Figure 2.7 demonstrates how the accuracy of the representation improves as the maximum level in the octree mesh is allowed to increase.

However, by eliminating elements that are cut by geometry, there is the potential to end up with a highly undesirable 'stair-stepping' feature when increasingly larger elements are cut by geometry and eliminated from the final computational mesh. Figure 2.8a demonstrates this behavior, note that in this figure the red elements are intersected by the geometry and are *not* part of the computational domain. This is exacerbated when combined with the 'norefine' feature



(a) Refinement level: 8

(b) Refinement level: 9

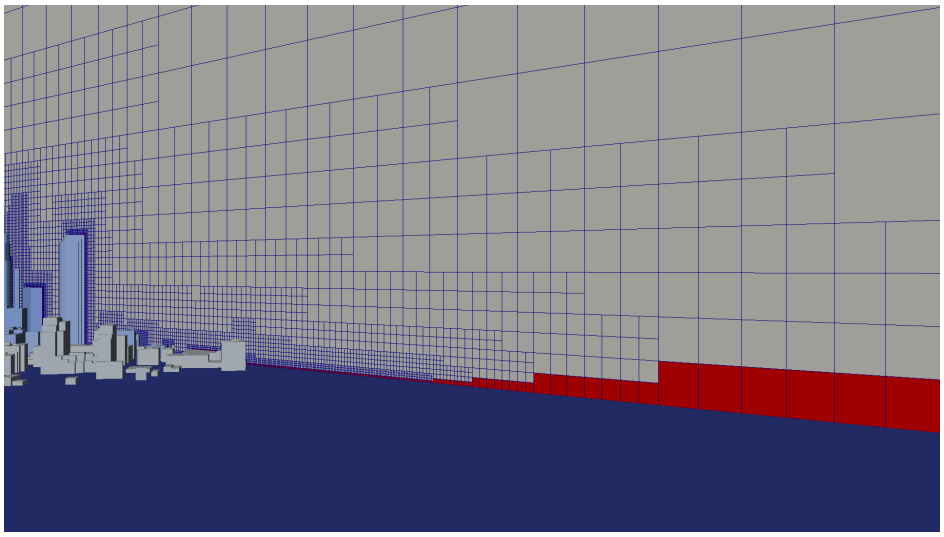
(c) Refinement level: 10

(d) Refinement level: 11

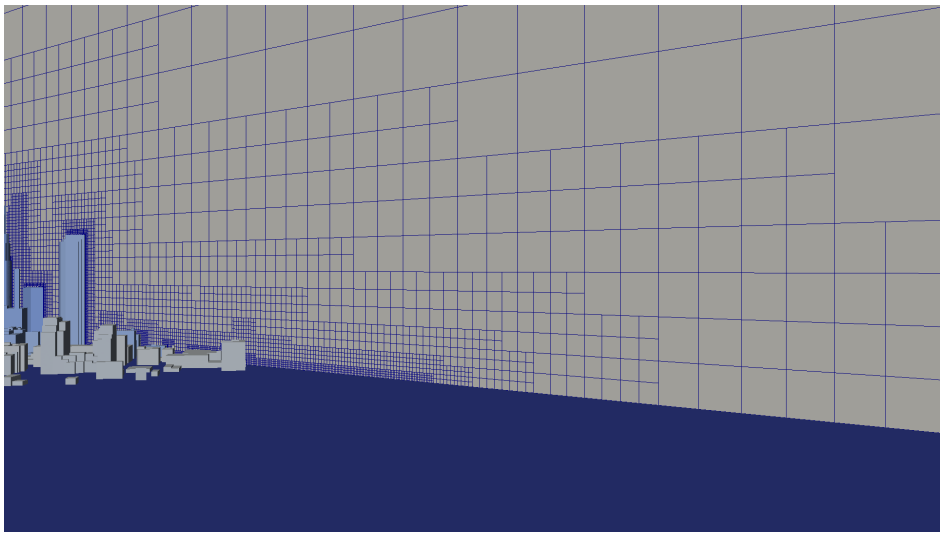
(e) Refinement level: 12

(f) Refinement level: 13

Figure 2.7 Demonstration of improved geometry representation on a tractor trailer model as octree refinement levels increase



(a) Without keep attribute

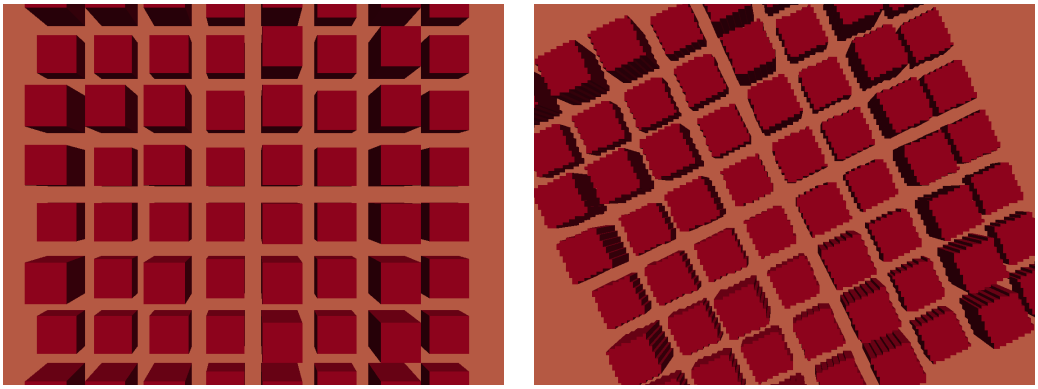


(b) With keep attribute

Figure 2.8 Demonstration of the effect of the ‘keep’ boundary attribute on an urban geometry

and usually happens in 'farfield' regions of the mesh. In order to fix this complication, a 'keep' boundary attribute has been added. Boundaries that do not require refinement, but that are in areas where this 'stair-stepping' character of the grid is undesirable, may be marked with the 'keep' attribute. Any element intersected by a boundary marked with this attribute will be marked directly as a *boundary* element in lieu of a *cut* element, forcing it to be a part of the final computational mesh and eliminating the problematic 'stair-stepping' feature. Figure 2.8b shows how the 'keep' attribute improves the quality of the mesh.

Another place that the geometry-approximating characteristic of this software may produce less than ideal grids is in urban geometries where the street canyons or alleys are not Cartesian aligned. This leads to sub-optimal discretization of buildings in the mesh as seen in Figure 2.9. While it is certainly not possible to completely fix this problem in any real situation, it may be



(a) Cartesian aligned street canyons (b) Non-Cartesian aligned street canyons

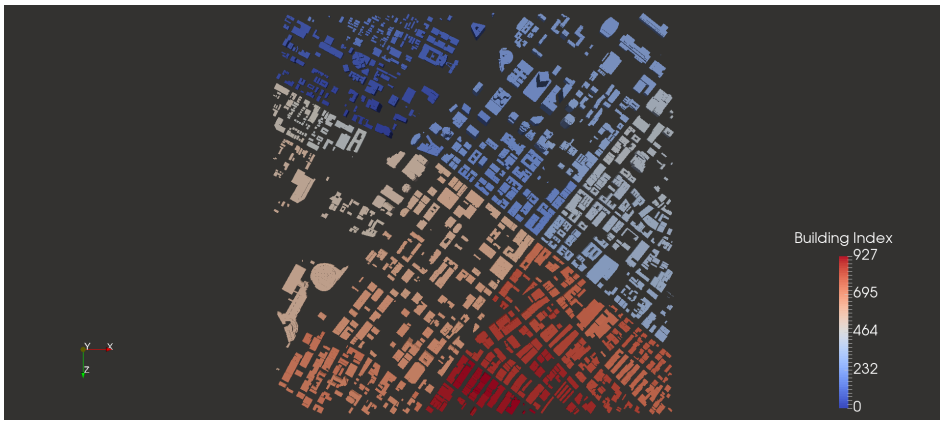
Figure 2.9 Cartesian versus non-Cartesian aligned street canyons

significantly improved by rotating the urban geometry such that the majority of street canyons align with the Cartesian axes.

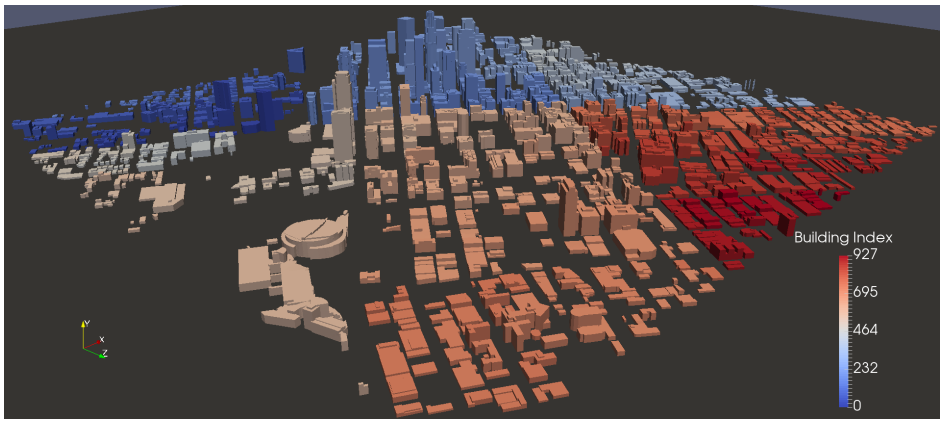
2.3 Parallelism and Parallel Performance

As opposed to hierarchical octree data structures, linear octree storage provides for optimal parallelism. In order to maintain this optimal parallelism however, octant repartitioning must occur at appropriate times during the mesh creation process. As noted in Section 2.1, ***Refine*** is a processor local routine so anytime a refinement callback is executed an octant imbalance is likely. Furthermore, the ***Balance*** function has the potential to create many new octants while ensuring 2:1 balance, and, even though it employs communication during its execution, it does nothing to ensure proper repartitioning. Therefore, calls to the *p4est* ***Partition*** function should be made after every call to the ***Refine*** and ***Balance*** functions. These two *p4est* functions are used liberally, especially during the quality refinement stages. A parallel framework is provided by *p4est* within which the current application must function; thus, for ease of use, all data that may be required during parallel communication should reside within the *p4est* data structures. Custom user data may be created both at the forest level and at the individual octant level within *p4est*. Custom data not unique to each octant may be stored at the forest level, whereas custom data that may vary from octant to octant must be stored in each individual element; e.g., the connectivity/macromesh and geometry are stored at the forest level, whereas the solution state vector, among other things, is stored in each of the elements in the micromesh. Using the ***Ghost*** function, *p4est* provides an interprocess neighbor mapping used to synchronize parallel data across partitions, as well as ways to perform said synchronization.

One great advantage of this approach to parallel grid generation is that the domain is not statically decomposed *a priori*; it is decomposed dynamically throughout the generation process. As mentioned earlier, one place where this dynamic decomposition is particularly advantageous is during the geometry refinement stage. During this phase of the grid generation process, the grid goes through the most dynamic changes and transforms from a simple connectivity of root octants into a nearly final version of the mesh. As this refinement process evolves, areas of the mesh that are near important parts of the geometry can see drastic changes in element counts. For this reason, the refinement process is done non-recursively a single level at a time with repartitioning occurring after each pass. Using this software, a mesh is generated on a large urban geometry, shown in Figure 2.10, which comprises 927 buildings and 4.25 million facets. Figure 2.11 shows the speedup curve attained while generating this mesh using a varying number of processors. This strong scaling speedup curve demonstrates how advantageous this dynamic partitioning can be in practice. While the speedup does taper off as the number of processors gets larger, it is actually super-linear at first. This strong scaling behavior, where the speedup curve drops well below the ideal speedup, is characteristic of many parallel applications and is often explained by a communication imbalance and/or a load imbalance. In this specific case, where the grid contains only approximately 6.2 million elements, there might simply be too few elements for the 256 and 512 processor cases, approximately 24 and 12 thousand elements per processor respectively, which could cause the communication costs to take up a disproportionate amount of the grid generation time. The potential impact of load imbalance on the parallel efficiency of the application is explored in the remainder of this section.



(a) Top view



(b) Perspective view

Figure 2.10 A large urban geometry containing 927 buildings colored by index

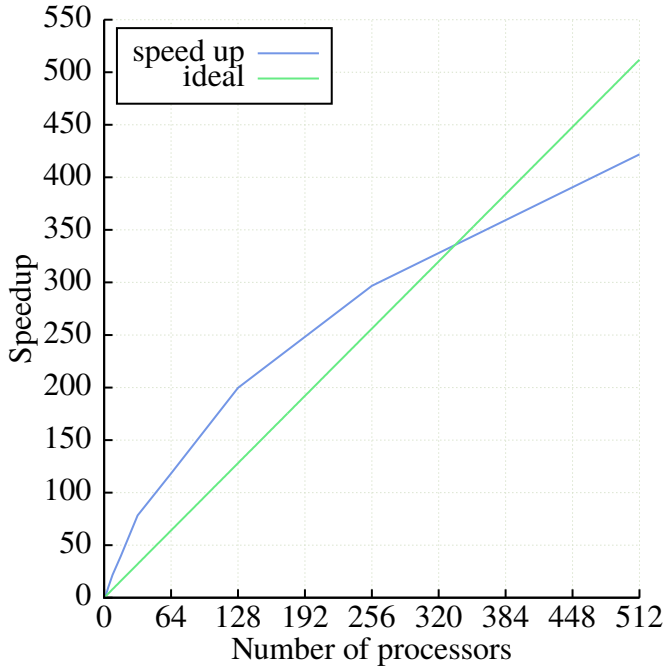


Figure 2.11 Speedup curve obtained creating an urban mesh containing approximately 6.2 million elements

Unfortunately, given the optimal partitioning of the grid afforded by *p4est* in terms of *elements per processor*, there are still algorithms that are difficult to load-balance. The flood-fill algorithm, among other methods that are highly dependent on the geometry, is a good example of this and may play a role in the sub-optimal speedup shown in Figure 2.11. These algorithms are much more sensitive to the geometry when it comes to load-balancing; while all processors have functionally the same number of elements, they might have very different percentages of the geometry which can cause significant load-imbalances. Repartitioning schemes that take proximity to geometry into account as a weighting for the partitions will improve this imbalance. One such scheme that has been explored during this research creates a 'distance' field by looping over the mesh and marking each element with the number of elements or the number of levels that lie

between it and the nearest target boundary; i.e., an element adjacent to a target boundary would receive a level value of one, while all of its neighbors that are not directly adjacent to the boundary would receive a value of two, and all of their neighbors not adjacent to any elements marked with a value of one would get marked three, etc. For an example of this, see Figure 2.12. By choosing

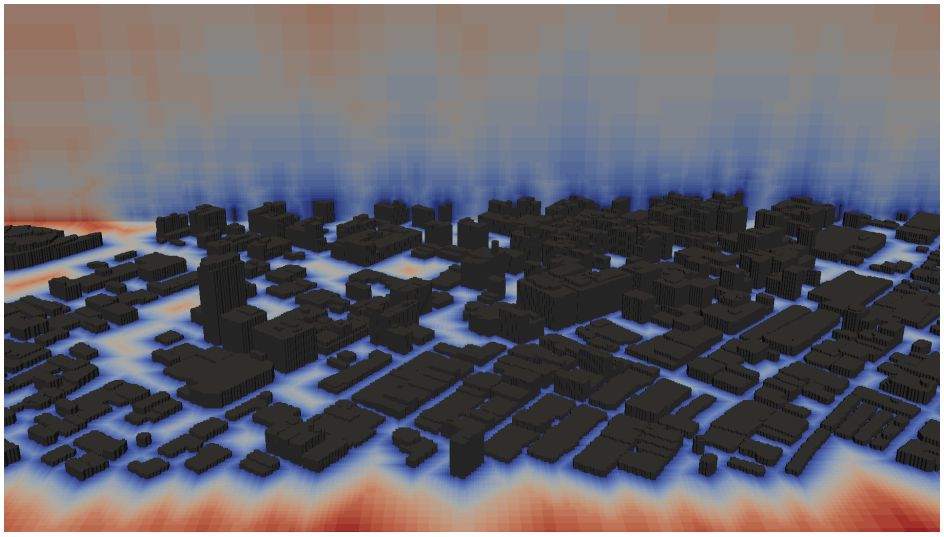


Figure 2.12 Example of the distance map used to repartition the mesh for better load balancing during the flood-fill algorithm

a weighting that favors elements near the target boundaries, i.e., with level values close to one, the resulting partitions can be shown to contain approximately equal numbers of these elements. As seen in Figure 2.13, this approach does produce better load balances for these geometry-centric algorithms, but it does not completely alleviate the problem. In the case of the flood-fill algorithm, it can be shown that the load balancing is directly proportional to the number of individual buildings each process owns. This is because, for each building in a single partition, the owning processor must query the underlying geometry to determine whether an element is inside or outside of the

computational domain at least once per building. The root cause of this sub-optimal load balancing stems from the fact that while each processor can be influenced to own an approximately equal number of elements near target boundaries, it is difficult to enforce that each processor contains approximately equal numbers of individual buildings. Therefore, some processors are required to make many more queries to the geometry library than others.

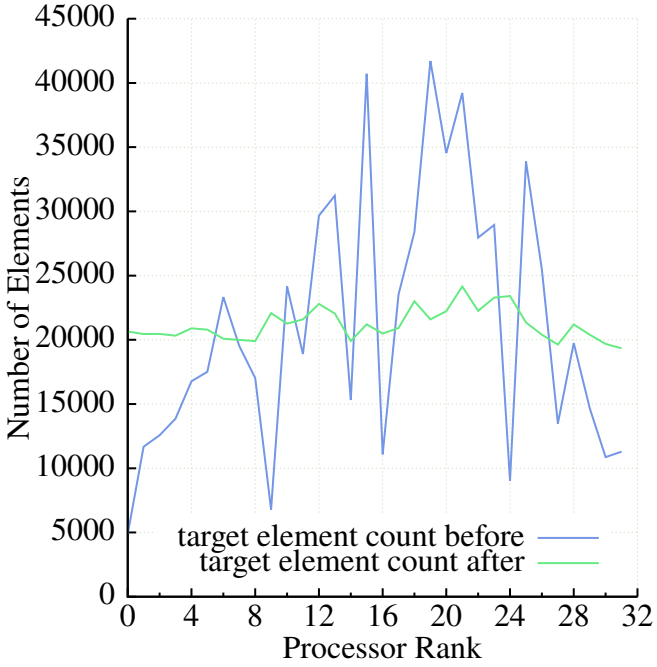


Figure 2.13 Improved distribution of target elements using new geometry-centric repartitioning scheme

This inefficiency was explored further during this research, and a method to significantly improve this load imbalance was discovered. As previously discussed, the source of the imbalance is caused by certain processors owning disproportionately more buildings than others. As such, it is desirable to weight the forest in such a way that all processors can own approximately the same

number of buildings in their partitions. In order to count the number of buildings each processor owns, each building must be identifiable as a single, unique structure in the geometry. Logic has been added to the geometry library that is able to take a single geometry boundary tag that is the same for all buildings and split it into one unique boundary tag for each distinct building contained in the geometry. In the event that the buildings were not tagged separately from the terrain, one could extend this logic to identify where buildings are located based on the angle between adjacent facets. Now that each building has a unique boundary tag, each processor may count the number of buildings it owns and identify which elements contain facets belonging to each building. Since each building should 'weigh' the same, regardless of its size, the following scheme has been implemented. After the forest has been refined and the intersected elements marked as *cut*, the forest is traversed, and the maximum number of elements intersecting any single building is counted. Once this number is known, all elements intersecting this building are assigned a weight of one while all other element-per-building counts are normalized by this number in such a way that the sum total of all element weights per building are approximately equal. For example, if the maximum number of elements-per-building is 1000, each of these 1000 elements will be assigned a weight of one, and if there is a building discretized by 500 elements, each of these elements will get a weight of two, while each element associated with a building discretized by 100 elements will get a weight of ten, etc. Note that *only* intersected elements are assigned weights using this approach. Repartitioning the forest using this new metric results in a much more uniform distribution of buildings per process and this does improve the overall load balancing of the flood-fill algorithm. Figures 2.14 and 2.15 demonstrate the improved building distribution facilitated by this new load-balancing algorithm. Even this, however, does not completely eliminate the imbalance.

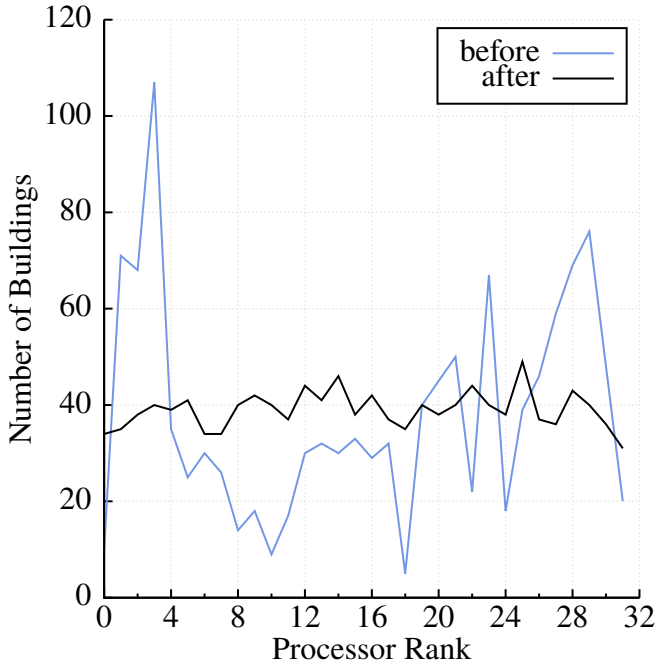


Figure 2.14 Demonstration of how the distribution of buildings changes with geometry-optimized partitioning on 32 processors

The required geometry query used to determine if an element is in or out of the geometry executes in constant time and, even though the new building distribution is much improved, it is not completely uniform, and, therefore, neither is the load balance. Another of the outstanding problems is that a single process may own a single building, but it is not guaranteed to own the *entire* building, and, as such, multiple geometry queries will need to be executed for the building in question instead of only one. Table 2.1 quantifies the load balance improvements achieved for the flood-fill algorithm run on the urban geometry shown in Figure 2.10. Notice how the difference between the minimum and maximum times required by any single processor for this algorithm is relatively much smaller after the repartitioning. While this is a difficult problem to optimally load-balance, the steps taken do result in a significant improvement.

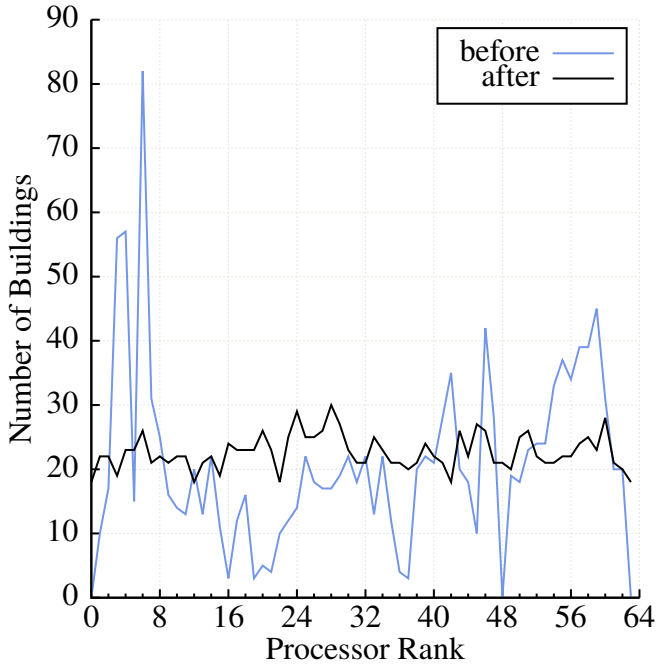


Figure 2.15 Demonstration of how the distribution of buildings changes with geometry-optimized partitioning on 64 processors

To be clear, this load-balancing problem is very geometry-specific; geometries that do not exhibit the same type of complexity as urban geometries, i.e., ones that do not require hundreds or thousands of 'in/out' queries to the geometry library, should not suffer as badly from this load imbalance.

Finally, in Figure 2.16, the improved speedup curve is compared to the original. Notice that the speedup curve is indeed better but still exhibits the same behavior at the higher end of the curve. This seems to point to the fact that communication imbalances are the likely culprit in this case and that the overhead incurred by the load balancing logic may outweigh the benefits as the problem scales.

Table 2.1 Minimum and maximum execution times required by the flood-fill algorithm on any single processor in a parallel partition before and after the improved, geometry-centric partitioning

partition scheme	16		32		64		128		256		512	
	min	max	min	max	min	max	min	max	min	max	min	max
old	17.18	115.73	7.16	76.69	1.17	60.47	0.57	37.85	0.56	24.53	0.55	14.9
new	41.58	56.56	19.22	37.31	11.64	22.76	5.05	16.89	2.75	15.54	1.15	12.31

A final note concerning the surprising 'super-linear' speedup results presented in this section is given here before moving on. Super-linear speedup is a sought-after and not-often seen phenomenon in speedup studies; it is usually caused by improvements in cache hits resulting from optimal cache line sizes produced by a particular parallel partitioning. When the cache contains much, or even all, of the data required for a set of computations, memory access times may decrease dramatically; this reduction in memory access times may result in speedups in addition to those obtained from the actual parallel computations. Furthermore, speedup studies are traditionally performed in a manner different from the way speedup has been studied in this research. Usually, a speedup plot will be generated on a static problem size, and the only variable is the number of partitions used to decompose the problem. For example, in CFD, the scalability of particular parallel solver may be of interest. The scalability of the solver would be studied by using a single, static mesh and exploring the run-times of the parallel solver on this static mesh as the number of processors grows. This approach is quite different from the one used in this research; the parallel grid generation process herein is *not* static in any way, and the decomposition of the mesh may change drastically *within* any single execution of the parallel code. What this means is that, not

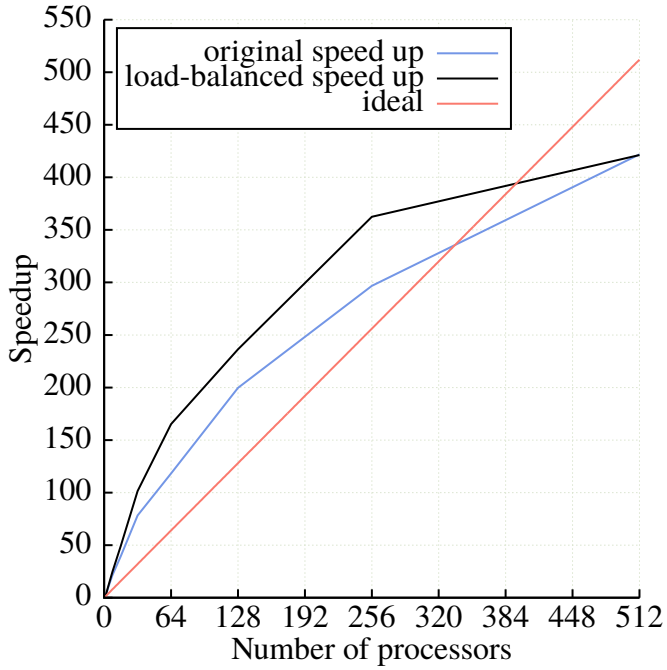


Figure 2.16 Speedup curve obtained after load balance improvements

only does the number of processors vary during the speedup study, the number of elements-per-process also varies dynamically during the execution of the parallel code. This is in stark contrast to the traditional approach described above. So, while it is possible that the optimal partitioning provided by the underlying *p4est* code does yield optimal cache lines, which may contribute to the observed 'super-linear' speedup, it might not be a fair comparison considering that this may not be a conventional application of a speedup study. Nevertheless, the data does not lie: the observed run-times exhibit very good trends regardless of whether this is a proper speedup study.

In order to give credence to a statement made earlier, alluding to the fact that the sub-optimal speedup towards the higher end of the curve (as the processor count approaches its maximum) may be related to low per-process element counts, the same case has been re-run two times on 128,

256, and 512 processors, each time the maximum level allowed in the forest is increased by one. The maximum level allowed in the original mesh is 9, the next larger case allows a maximum of 10 levels, and in the final case 11 levels of refinement is allowed. The original mesh contains ~6.2 million elements, the next mesh ~24.8 million elements, and the final mesh ~57.3 million elements. Table 2.2 shows the approximate element-per-process count for all three cases; the two larger cases both contain significantly more elements on each process than the original, and should exhibit improved communication-to-work ratios (or surface-to-volume ratios). The grid-generation

Table 2.2 Approximate per-process element count distribution

Case	Processor Count		
	128	256	512
Small	49,000	24,000	12,000
Medium	195,000	97,000	48,000
Large	449,000	224,000	112,000

times for each of the three cases are displayed in Figure 2.17; this plot corroborates the theory that poor surface-to-volume ratios are at least partially to blame for the speedup degradation. Note that even though Figure 2.17 is meant to illustrate grid generation speedup characteristics, it plots the actual grid generation *times* in lieu of *speedup*, and thus, it may require some extra explanation as it does not look like either of the speedup figures (Figures 2.11 and 2.16) presented previously. In this plot, the dotted lines represent a perfect speedup for the actual case run, shown in the figure by the solid lines of the same color. Each time the number of processors doubles, the runtime should decrease by a factor of two, which is the same as saying that the slope of lines connecting each

consecutive point should decrease by a factor of four. In this figure, notice that the actual timing curve for the smallest case has all but leveled out by the time it reaches 512 processors and it is not parallel to its associated ideal timing curve. However, as the cases get bigger, the slopes of the final line segments connecting last two time points become much closer to being parallel to their ideal timing curve.

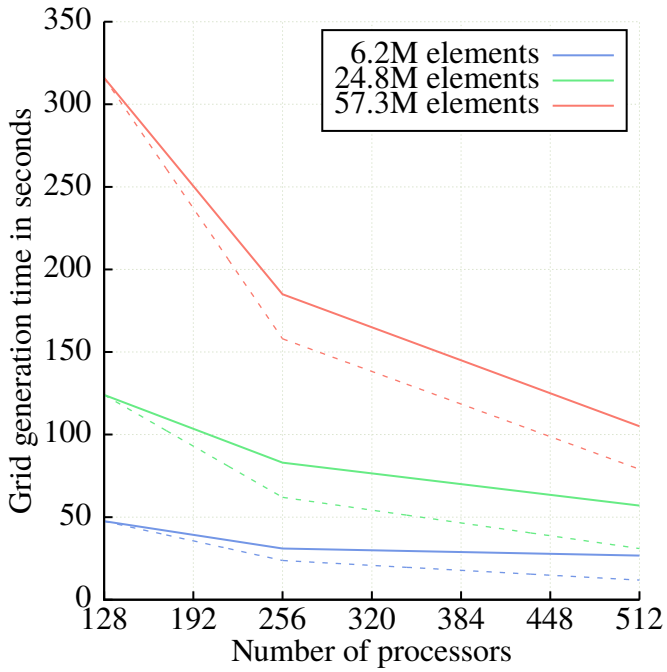


Figure 2.17 Grid generation timing curve demonstrating improved speedup behavior as the mesh size grows

It is also worth noting, that since this process is so dynamic in nature, the end product is not exactly the same as the number of processors varies. This slight variation in the final mesh is likely not significant and can be explained by the fact that the 'path' taken to obtain the final product is indeed different for each processor count. However, it is well known that numerical solutions to

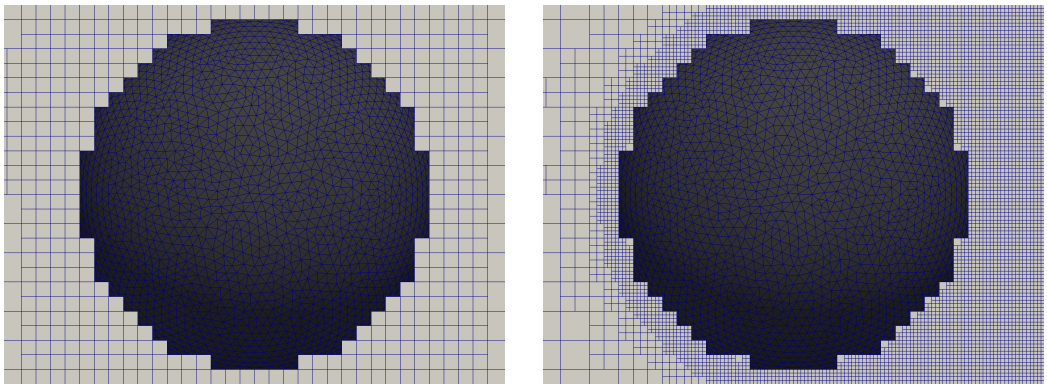
PDEs can be grid-dependent and so any variance in a grid caused simply by changing the number of processors should be quantified to enable a user to gain an understanding about how changing the number processors in a simulation might affect the overall flow solution.

2.4 Adaptive Mesh Refinement

This framework also provides solution-based adaptive mesh refinement (AMR); the mesh may be refined and coarsened based on user specifiable threshold values of a user specified solution feature. This functionality is built into the basic *p4est* framework and all that must be provided are three callback functions: a refinement callback, coarsening callback, and agglomeration callback. The refinement and coarsening callbacks must simply provide the appropriate logic to determine whether the current element should be refined, or the current family of elements should be coarsened, based on the particular solution feature and the input threshold values. The agglomeration callback must also be provided to each of the refinement and coarsening callbacks and is used to either assign solution values to the eight new child elements based on the solution state of the parent element (in the case of refinement) or average/agglomerate the solution data from eight child elements into a parent element (in the case of coarsening). Solution gradient data is used to extrapolate solution data from a parent element onto each of its child elements during refinement if it is available, whereas a simple averaging procedure is used to combine the solution data from all eight child elements into their parent during coarsening. After the refinement and coarsening passes are made through the mesh, the 2:1 balance restriction is almost certainly no longer enforced; thus, the balancing function must be invoked. The same agglomeration callback provided to the

refinement and coarsening functions must also be provided here during the balancing pass to allow the redistribution of solution data while the mesh changes during balancing.

This logic does allow for boundary refinement; however, currently the representation of the geometry in the mesh never actually changes. Existing boundary faces may be subdivided potentially multiple times, but the original planar mesh face will always remain in the mesh. This also means that when boundary elements are refined, none of the new elements are checked for geometry intersection, which means the geometry must be adequately resolved in the *original* mesh as the original representation will never change during AMR. Figure 2.18 demonstrates this issue.



(a) Grid before AMR (b) Grid after AMR

Figure 2.18 A demonstration af AMR and boundary refinement; notice how the modified mesh (b) contains all the boundary faces present in the original mesh (a) but they are more refined, and the modified mesh does not represent the geometry better

Coarsening is not allowed when any child element in the group of eight children is not itself part of the original mesh or is a boundary element; i.e., if any one of the children elements contains a status of *out*, *cut*, or *boundary*, that group of elements is no longer considered for coarsening.

Boundary coarsening is a feature that may be valuable to include, but the logic required for its implementation is significantly more difficult than that required for boundary refinement.

At runtime the user may specify the function used for AMR as well as the threshold values used to determine when refinement or coarsening is required. Currently, only vorticity magnitude is supported for solution mesh adaptation, but adding new functions is a trivial exercise. Also, the frequency at which AMR occurs during the solution iteration is another parameter that the user may specify at runtime. In this manner the AMR method may be called periodically (e.g., once every five or ten time steps) as the user sees fit. Figure 2.19 demonstrates the AMR functionality in action on the revised simple frigate shape (SFS2) geometry. As the grid changes dynamically

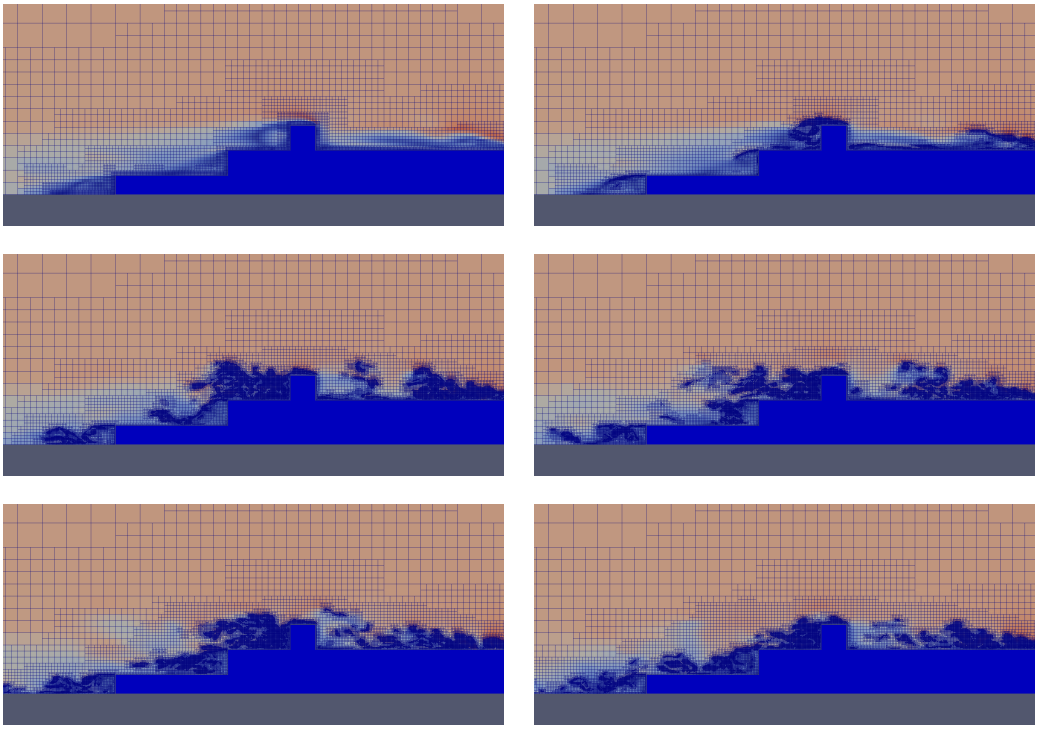
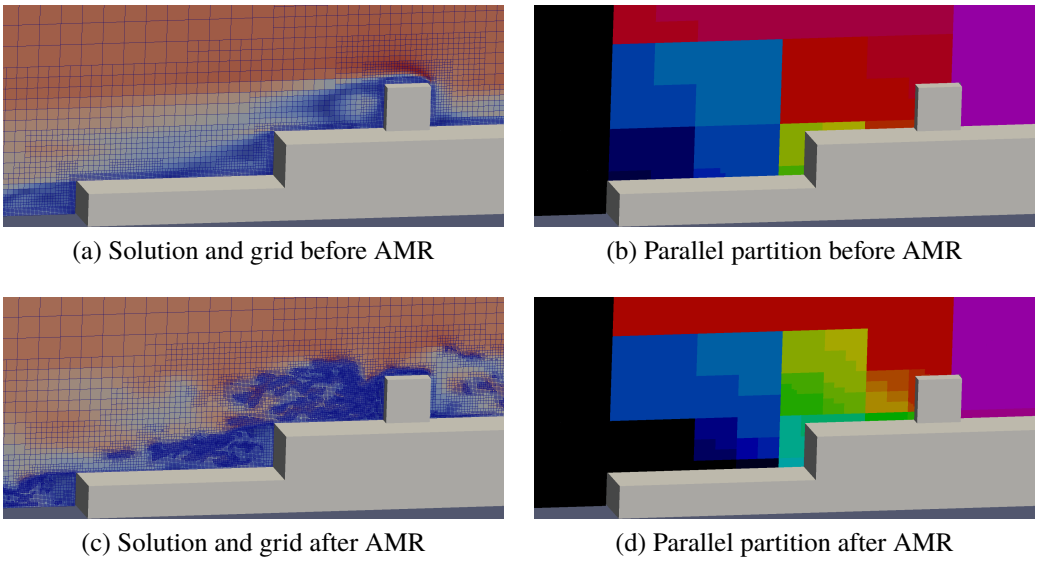


Figure 2.19 A demonstration of unsteady AMR on the SFS2 geometry

during AMR, repartitioning is required to maintain an acceptable load balance. This balance is achieved easily via the previously discussed ***Partition*** function provided by *p4est*. Figure 2.20 demonstrates how the parallel partitioning changes to accommodate the AMR mesh.



(a) Solution and grid before AMR

(b) Parallel partition before AMR

(c) Solution and grid after AMR

(d) Parallel partition after AMR

Figure 2.20 Dynamic AMR and load balancing

CHAPTER 3

SOLUTION METHODOLOGY AND ALGORITHMS

An initial goal of this research is to simulate relatively low speed flows in urban environments where buoyancy may play a significant role. As such, the appropriate physics are modeled by the unsteady, incompressible, Navier-Stokes (NS) equations. However, there is a complication that arises within the context of the incompressible NS equations: the pressure becomes decoupled from the system when incompressibility is enforced, and, in order to maintain the hyperbolicity required by most time-marching CFD schemes, pressure must be recoupled to the other equations. The pseudo-compressibility method is the approach used herein to address the problems related to pressure decoupling; see Chorin [86] and Taylor [87] for details about this method. Furthermore, as heat transfer/buoyancy is important in the simulation of plume propagation, the energy equation cannot be ignored, and requisite steps must be taken to ensure proper coupling with the energy term. This work mimics the work of Kress [88] and Er [89]. This chapter defines the governing equations utilized, as well as the numerical formulation, spatial and temporal discretizations, and source and boundary terms used for their solution.

3.1 Governing Equations

In this section, the Navier-Stokes equations will be given, starting with the most general compressible form of the equations, which will subsequently be simplified and augmented until

the required incompressible NS equations with pseudo-compressibility and heat transfer terms are presented. The resulting equations are a subset of the compressible NS equations.

3.1.1 Compressible Navier-Stokes Equations

The Navier-Stokes (NS) equations can be written in conservative form as

$$\frac{\partial \boldsymbol{Q}(\boldsymbol{x},t)}{\partial t} + \nabla \cdot (\boldsymbol{F} - \boldsymbol{F}_v) = \boldsymbol{S} \qquad \text{in} \quad \Omega \tag{3.1}$$

where Ω is a closed domain. The vector of conservative flow variables, $\boldsymbol{Q}$, and the inviscid and viscous Cartesian flux vectors, $\boldsymbol{F}$ and $\boldsymbol{F}_v$, are defined by

$$\boldsymbol{Q} = \begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{Bmatrix} \qquad \boldsymbol{F}^x = \begin{Bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ (\rho E + p)u \end{Bmatrix} \qquad \boldsymbol{F}^y = \begin{Bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ (\rho E + p)v \end{Bmatrix} \qquad \boldsymbol{F}^z = \begin{Bmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ (\rho E + p)w \end{Bmatrix} \tag{3.2}$$

$$\boldsymbol{F}_v^x = \begin{Bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ u\tau_{xx} + v\tau_{xy} + w\tau_{xz} + \kappa\frac{\partial T}{\partial x} \end{Bmatrix} \quad \boldsymbol{F}_v^y = \begin{Bmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yz} \\ u\tau_{yx} + v\tau_{yy} + w\tau_{yz} + \kappa\frac{\partial T}{\partial y} \end{Bmatrix}$$
$$\boldsymbol{F}_v^z = \begin{Bmatrix} 0 \\ \tau_{zx} \\ \tau_{zy} \\ \tau_{zz} \\ u\tau_{zx} + v\tau_{zy} + w\tau_{zz} + \kappa\frac{\partial T}{\partial z} \end{Bmatrix} \tag{3.3}$$

where ρ, p, and E denote the fluid density, pressure, and specific total energy, respectively. The vector $\boldsymbol{u} = (u, v, w)$ represents the Cartesian velocity vector. The pressure is determined by the definition of total energy and the equation of state for an ideal gas,

$$p = (\gamma - 1)\left(\rho E - \frac{1}{2}\rho(u^2 + v^2 + w^2)\right) \tag{3.4}$$

where γ is defined as the ratio of specific heats, which is 1.4 for air. Temperature and thermal conductivity are represented by T and κ, respectively, and are related to the total energy and velocity as

$$\kappa T = \gamma\left(\frac{\mu}{Pr} + \frac{\mu_T}{Pr_T}\right)\left(E - \frac{1}{2}\left(u^2 + v^2 + w^2\right)\right) \tag{3.5}$$

where Pr and Pr_T are the 'laminar' Prandtl and turbulent Prandtl numbers which are set to 0.72 and 0.9, respectively. The fluid viscous stress tensor, τ, is defined for a Newtonian fluid as

$$\tau_{ij} = (\mu + \mu_T)\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\frac{\partial u_k}{\partial x_k}\delta_{ij}\right) \tag{3.6}$$

where the summation convention has been used. Subscripts i, j, k refer to the Cartesian coordinate components of $\boldsymbol{x} = (x, y, z)$ and δ_{ij} is the Kronecker delta. In addition, μ refers to the fluid dynamic viscosity and is obtained via Sutherland's law, while μ_T denotes a turbulence eddy viscosity, which is obtained from a turbulence model. The source term, $\boldsymbol{S}$, in equation (3.1) may take on many forms and often contains body force terms as well as turbulence model terms.

3.1.2 Incompressible Navier-Stokes Equations

In situations where the characteristic speed of the flow is significantly less than the fluid speed of sound, c, i.e., $u \ll c$, fluid compressibility becomes negligible; i.e. density becomes a very weak function of pressure. This implies that the fluid density ρ remains functionally constant, both spatially and temporally, throughout the flow field. Therefore, the incompressible Navier-Stokes equations may be derived from their compressible counterparts, again given in general by equation (3.1), by treating ρ as a spatial and temporal constant ρ_∞. Dividing each term of equation (3.1) by ρ_∞ yields the final form of these equations. In this case the vector of conservative

flow variables $\boldsymbol{Q}$ and the inviscid Cartesian flux vector $\boldsymbol{F}$ given in equation (3.2) become

$$\boldsymbol{Q} = \begin{Bmatrix} 0 \\ u \\ v \\ w \end{Bmatrix} \qquad \boldsymbol{F}^x = \begin{Bmatrix} u \\ u^2 + \frac{p}{\rho_\infty} \\ uv \\ uw \end{Bmatrix} \qquad \boldsymbol{F}^y = \begin{Bmatrix} v \\ uv \\ v^2 + \frac{p}{\rho_\infty} \\ vw \end{Bmatrix} \qquad \boldsymbol{F}^z = \begin{Bmatrix} w \\ uw \\ vw \\ w^2 + \frac{p}{\rho_\infty} \end{Bmatrix} \tag{3.7}$$

Note that the viscous Cartesian flux vector $\boldsymbol{F}_v$ remains as defined in equation (3.3) except that the terms belonging to the energy equation—the fifth element of the vectors—are eliminated. This decoupling is because, in an incompressible fluid, the molecular viscosity is assumed to be constant in addition to the fluid density. This causes the energy equation to become decoupled from the system as it has no direct dependence on the other fluid properties besides the velocity, which convects the energy along with the flow.

$$\boldsymbol{F}_v^x = \begin{Bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \end{Bmatrix} \quad \boldsymbol{F}_v^y = \begin{Bmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yz} \end{Bmatrix} \quad \boldsymbol{F}_v^z = \begin{Bmatrix} 0 \\ \tau_{zx} \\ \tau_{zy} \\ \tau_{zz} \end{Bmatrix} \tag{3.8}$$

Note, however, that the stress tensor for these incompressible equations, τ_{ij}, is slightly different because the equations have been divided through by ρ_∞. The incompressible form of τ_{ij} is

$$\tau_{ij} = (\nu + \nu_T)\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\frac{\partial u_k}{\partial x_k}\delta_{ij}\right) \tag{3.9}$$

where $\nu = \frac{\mu}{\rho}$ is the kinematic viscosity of the fluid.

3.1.3 Pseudo-Compressible Navier-Stokes Equations

Note that for the incompressible NS equations, equation (3.1) contains no time derivative in the continuity equation; time-marching schemes cannot apply as the entire system is not time dependent. Chorin [86] proposed a method, deemed the pseudo-compressibility or artificial compressibility method, in which a fictitious time derivative of pressure is added to the continuity equation. This method makes time-marching schemes applicable again as the pseudo-compressible form of the incompressible Navier-Stokes equations regain their hyperbolic character. Using this method, the continuity equation takes the form

$$\frac{\partial \tilde{\rho}}{\partial t} + \nabla \cdot \boldsymbol{u} = 0 \tag{3.10}$$

where $\tilde{\rho} = p/\beta$ and β is a constant artificial compressibility parameter which should be as large as possible for accurate simulation of disturbance propagation in incompressible flow [87]. However, in reality, stability, convergence, and accuracy are the limiting factors which put an upper bound on its value. Pan and Chakravarthy [90] report a normal range is from 1 to 10. Hyams [91] reports that a value of 15 is typical. Palacios et al. [92] demonstrate results using values ranging from 1 to 20, as well as quantify the effect of the value of β on convergence and solution accuracy.

So, the pseudo-compressibility formulation of the unsteady, incompressible NS equations can be written in conservative form as equation (3.1), where $\boldsymbol{Q}$ and $\boldsymbol{F}$ are defined by

$$\boldsymbol{Q} = \begin{Bmatrix} p \\ u \\ v \\ w \end{Bmatrix} \qquad \boldsymbol{F}^x = \begin{Bmatrix} \beta u \\ u^2 + \frac{p}{\rho_\infty} \\ uv \\ uw \end{Bmatrix} \qquad \boldsymbol{F}^y = \begin{Bmatrix} \beta v \\ uv \\ v^2 + \frac{p}{\rho_\infty} \\ vw \end{Bmatrix} \qquad \boldsymbol{F}^z = \begin{Bmatrix} \beta w \\ uw \\ vw \\ w^2 + \frac{p}{\rho_\infty} \end{Bmatrix} \tag{3.11}$$

Note that $\boldsymbol{F}_v$ does not change in form from equation (3.8).

3.1.4 Accounting for Buoyancy

Since thermal buoyancy is an important part of this research as it applies to urban simulation, appropriate considerations must be taken to couple an energy equation to the pseudo-compressible NS equations. This coupling is achieved by the addition of a temperature equation. The solution vector is now

$$\boldsymbol{Q} = \begin{Bmatrix} p \\ u \\ v \\ w \\ T \end{Bmatrix} \tag{3.12}$$

where T represents fluid temperature. While the density variation as a result of temperature gradient is small, it is not negligible. Following Er [89] and Kress [88], the Boussinesq approximation is

applied to the momentum equation that acts in the direction of gravity. The Boussinesq approximation [93] treats the fluid density as a constant in every term of this momentum equation except for the source term, where it is multiplied by the acceleration due to gravity g; there, density is treated as a function of temperature. Thus, the source term may be written as

$$\boldsymbol{S} = \frac{1}{\rho_\infty} \begin{Bmatrix} 0 \\ 0 \\ -\rho g \\ 0 \\ 0 \end{Bmatrix} \tag{3.13}$$

if gravity acts in the y direction, or in general

$$\boldsymbol{S} = \frac{1}{\rho_\infty} \begin{Bmatrix} 0 \\ \rho g e_{g_x} \\ \rho g e_{g_y} \\ \rho g e_{g_z} \\ 0 \end{Bmatrix} \tag{3.14}$$

where $\boldsymbol{e}_g = (e_{g_x}, e_{g_y}, e_{g_z})$ is a unit vector pointing in the direction of gravitational acceleration. Using a truncated Taylor series expansion, density may be expressed as a function of temperature

$$\rho(T) = \rho_\infty + \left(\frac{\partial \rho}{\partial T}\right)_p (T - T_\infty) \tag{3.15}$$

where ρ_∞ is the density of the fluid at temperature T_∞. The coefficient of thermal expansion, α_T, in a fluid at constant pressure is

$$\alpha_T = -\frac{1}{\rho_\infty}\left(\frac{\partial \rho}{\partial T}\right)_p \tag{3.16}$$

Substituting equation (3.16) into equation (3.15) yields

$$\rho(T) = \rho_\infty[1 - \alpha_T(T - T_\infty)] \tag{3.17}$$

which may be rewritten as

$$\rho(T) = \rho_\infty[1 + \rho'] \tag{3.18}$$

where $\rho' = -\alpha_T(T - T_\infty)$. By the Boussinesq approximation, equation (3.18) is substituted into equation (3.13) or equation (3.14), while the other density terms are left unmodified. After some rearranging, this final form of the pseudo-compressible buoyant NS equations is obtained

$$\boldsymbol{Q} = \begin{Bmatrix} p^* \\ u \\ v \\ w \\ T \end{Bmatrix} \qquad \boldsymbol{F}^x = \begin{Bmatrix} \beta u \\ u^2 + p^* \\ uv \\ uw \\ Tu \end{Bmatrix} \qquad \boldsymbol{F}^y = \begin{Bmatrix} \beta v \\ uv \\ v^2 + p^* \\ vw \\ Tv \end{Bmatrix} \qquad \boldsymbol{F}^z = \begin{Bmatrix} \beta w \\ uw \\ vw \\ w^2 + p^* \\ Tw \end{Bmatrix} \tag{3.19}$$

$$\boldsymbol{F}_v^x = \begin{Bmatrix} 0 \\ \tau_{xx} \\ \tau_{xy} \\ \tau_{xz} \\ -\frac{\kappa_\infty}{\rho_\infty c_{p\infty}} \frac{\partial T}{\partial x} \end{Bmatrix} \quad \boldsymbol{F}_v^y = \begin{Bmatrix} 0 \\ \tau_{yx} \\ \tau_{yy} \\ \tau_{yz} \\ -\frac{\kappa_\infty}{\rho_\infty c_{p\infty}} \frac{\partial T}{\partial y} \end{Bmatrix} \quad \boldsymbol{F}_v^z = \begin{Bmatrix} 0 \\ \tau_{zx} \\ \tau_{zy} \\ \tau_{zz} \\ -\frac{\kappa_\infty}{\rho_\infty c_{p\infty}} \frac{\partial T}{\partial z} \end{Bmatrix} \quad \boldsymbol{S} = \begin{Bmatrix} 0 \\ \rho' g e_{g_x} \\ \rho' g e_{g_y} \\ \rho' g e_{g_z} \\ 0 \end{Bmatrix} \tag{3.20}$$

where p^* absorbs the hydrostatic pressure into the definition of pressure and is defined as $p^* = \frac{p}{\rho_\infty} + g\,y$ if gravity acts in the y direction, or $p^* = \frac{p}{\rho_\infty} - gB$, where $B = \boldsymbol{x} \cdot \boldsymbol{e_g}$, for an arbitrary gravitational direction.

3.2 Non-Dimensionalization

The NS equations, in the pseudo-incompressible buoyant form defined by equation (3.1) and equations (3.19) and (3.20), will now be non-dimensionalized. To this end, the following definitions for the non-dimensional forms of the constituent variables will be used.

$$\begin{aligned} \hat{\boldsymbol{x}} &= \frac{\boldsymbol{x}}{L} \qquad \hat{t} = \frac{t u_\infty}{L} \quad \hat{\boldsymbol{u}} = \frac{\boldsymbol{u}}{u_\infty} \quad \hat{p} = \frac{p - p_\infty}{\rho_\infty u_\infty^2} \\ \hat{\rho} &= \frac{\rho}{\rho_\infty} \qquad \hat{\mu} = \frac{\mu}{\mu_\infty} \quad \hat{\kappa} = \frac{\kappa}{\kappa_\infty} \quad \hat{T} = \frac{T - T_\infty}{\Delta T} \\ \hat{\tau} &= \frac{\tau L}{\mu_\infty u_\infty} \end{aligned} \tag{3.21}$$

where the subscript ∞ signifies a reference value and the circumflex, or hat ($\hat{}$), represents a non-dimensional variable. L is a representative characteristic length, $\Delta T = T_{\text{wall}} - T_\infty$, and T_{wall} is a reference wall temperature. The divergence and gradient operators non-dimensionalize with a direct application of the chain rule. Substituting the values from equations (3.21) into equation (3.1),

rearranging, and dropping the circumflex, yields the non-dimensionalized form of the compressible NS equations. The general form of the equations do not change from equation (3.1); however, the specific forms of the flux functions and source terms will be different. For the buoyant NS equation the solution vector $\boldsymbol{Q}$ and the inviscid flux vectors remain unchanged by the non-dimensionalization. However, the modified pressure p^*, viscous flux vectors, and source term have the following non-dimensional forms

$$p^* = p + \frac{y}{Fr^2} \tag{3.22}$$

if gravity acts in the y direction or

$$p^* = p - \frac{\hat{B}}{Fr^2} \tag{3.23}$$

for an arbitrary gravitational vector and $\hat{B} = \frac{B}{L} = \frac{\boldsymbol{x}}{L} \cdot \boldsymbol{e_g}$.

$$\boldsymbol{F}_v^x = \begin{Bmatrix} 0 \\ \frac{1}{Re}\tau_{xx} \\ \frac{1}{Re}\tau_{xy} \\ \frac{1}{Re}\tau_{xz} \\ -\frac{\kappa}{Pe}\frac{\partial T}{\partial x} \end{Bmatrix} \quad \boldsymbol{F}_v^y = \begin{Bmatrix} 0 \\ \frac{1}{Re}\tau_{yx} \\ \frac{1}{Re}\tau_{yy} \\ \frac{1}{Re}\tau_{yz} \\ -\frac{\kappa}{Pe}\frac{\partial T}{\partial y} \end{Bmatrix} \quad \boldsymbol{F}_v^z = \begin{Bmatrix} 0 \\ \frac{1}{Re}\tau_{zx} \\ \frac{1}{Re}\tau_{zy} \\ \frac{1}{Re}\tau_{zz} \\ -\frac{\kappa}{Pe}\frac{\partial T}{\partial z} \end{Bmatrix} \quad \boldsymbol{S} = \frac{\alpha_T T \Delta T}{Fr^2} \begin{Bmatrix} 0 \\ e_{g_x} \\ e_{g_y} \\ e_{g_z} \\ 0 \end{Bmatrix} \tag{3.24}$$

In these equations, the non-dimensional parameters are the Reynolds number: $Re = \frac{\rho_\infty u_\infty L}{\mu_\infty}$, the Froude number: $Fr = \frac{u_\infty}{\sqrt{gL}}$, the Prandtl number: $Pr = \frac{\mu_\infty c_p}{\kappa_\infty}$, the Peclet number: $Pe = RePr = \frac{\rho_\infty c_p u_\infty L}{\kappa_\infty}$, and the Grashoff number: $Gr = \frac{g\alpha_T \Delta T L^3}{\nu^2}$. An additional non-dimensional parameter, the Rayleigh number, will be used later and is defined as the product of the Grashoff and Prandtl

numbers: $Ra = GrPr$. By a simple calculation, the coefficient on the source term vector may be written as

$$\frac{\alpha_T T(T_{\text{wall}} - T_\infty)}{Fr^2} = \frac{\alpha_T T \Delta T}{Fr^2} = T\frac{Gr}{Re^2} \tag{3.25}$$

Although not included here, any other form of the NS equations may also be non-dimensionalized following a similar approach.

3.3 Turbulence Modeling

The motivation of this research is to quickly generate flow field solutions in an urban setting where viscous spacing can be very large and drag forces are not important nor desired; thus certain liberties may be taken with the choice of turbulence models. Since the overall effect of blockages, e.g., buildings and other features, is of main interest, and it is intractable to properly resolve the boundary layer physics, the use of typical turbulence models that are designed to accurately simulate surface phenomenon may be overkill. As such, simple large eddy simulation (LES) models seem a better fit and, consequently, the Smagorinski LES turbulence model has been chosen. In 1963 Joseph Smagorinski proposed a turbulence model, one which now bears his name, to simulate atmospheric circulation [94]. This model is algebraic, and its simplicity makes it easy to implement and also makes it faster than other standard turbulence models. Furthermore, Nichols [95] lauds the effectiveness of this model as compared to other standard models like the SAS, SST, and Wilcox stress-ω models. Nichols [95] concluded that the Smagorinski model performed as well as the SST and Wilcox stress-ω models on the surface mounted cube case and better than the SAS model. The

Smagorinski model directly computes the eddy viscosity as

$$\mu_T = (C_s \Delta)^2 \left| \tilde{S} \right| \tag{3.26}$$

where C_s is the 'Smagorinski constant' and is set to 0.17 for this research. Δ is a length scale or filter width related to the size of the grid and is defined to be $\sqrt[3]{V}$, where V is the volume of the grid element. Lastly, $|\tilde{S}| = \sqrt{2\tilde{S}_{ij}\tilde{S}_{ij}}$ represents the magnitude of the strain rate tensor: $\tilde{S}_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$.

3.4 Spatial Discretization on a *p4est* Grid

The integral form of the partial differential equations given in equation (3.1) is discretized over the computational domain, Ω, using a cell-centered, finite-volume method on a Cartesian octree style grid. Octree meshes contain hanging nodes in all but the unusual case of a uniformly refined mesh; hanging nodes occur when an element in the mesh has neighbors at levels different from its own. Figure 3.1 demonstrates the concept of a hanging node. Octree meshes used for finite volume discretizations are generally required to be 2:1 balanced in order to maintain reasonable volume/area ratios. This 2:1 balancing requires that neighboring elements differ by at most one level in their octree. This requirement implies that neighboring elements never have volumes that differ by more than a factor of eight. Extra care must be taken when calculating fluxes through a face of an element when its neighbor across the face is at a different level: the flux must be calculated through the smaller of the two faces. Higher order reconstruction must also be done with care as the extrapolation occurs in different directions in each cell instead of along a single vector.

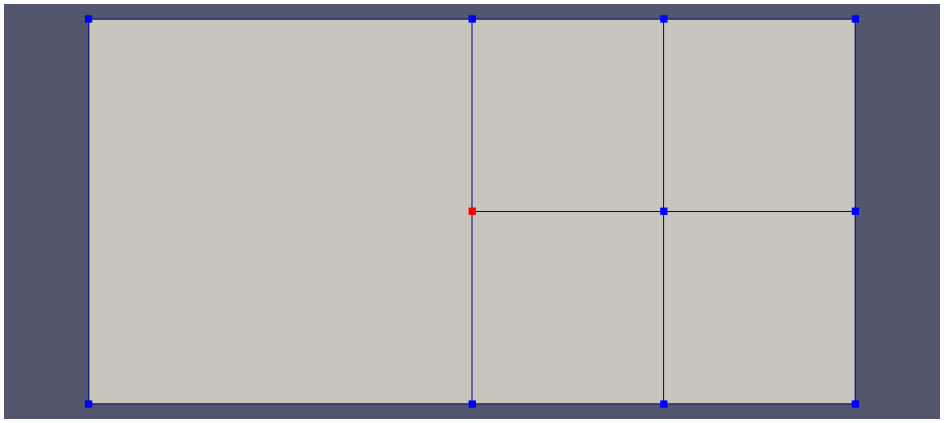


Figure 3.1 An example of a hanging node; the red node is a hanging node that exists because of the difference in refinement levels of the adjacent elements

Figure 3.2 shows the difference between extrapolation vectors for interfaces between two elements at the same level and two elements at different levels.

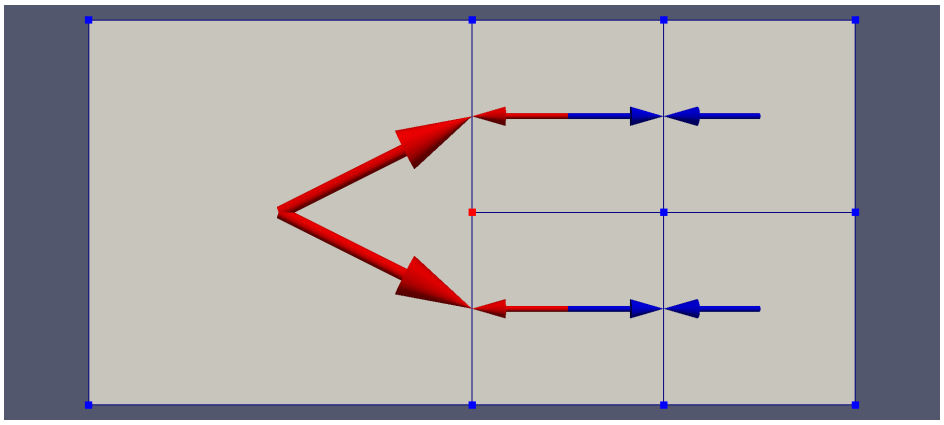


Figure 3.2 An example of the difference between extrapolation vectors for elements at equal levels and elements at different levels of refinement

3.5 Solution Methodology

This section explains the solution methodology used in the current research as well as specifics about certain algorithms implemented therein. The flow solver is written in C++ and is implicit, time-accurate, and designed with a 'plug-and-play' style framework, which means that any PDE may be solved as long as key methods (inviscid and viscous fluxes, Jacobians, etc.) are derived that describe the appropriate physics. The resulting application is named *paros* (PARallel Octree Solver).

3.5.1 Spatial Discretization

Spatial discretization is obtained herein using a cell-centered, finite-volume method on the *p4est* octree meshes described in Section 2.1. This approach is much like the node-centered approach described by Hyams [91], except for the cases where neighboring elements are at different levels in the octree, which implies that there will be hanging nodes. These cases must be approached with more care as was discussed in Section 3.4. If it is assumed that the solution state $\boldsymbol{Q}$ is constant within each control volume, the volume-averaged state vector is

$$\boldsymbol{q} = \frac{\int_V \boldsymbol{Q} dV}{V} \tag{3.27}$$

Thus the spatial terms discretized over each control volume, i, may be expressed as

$$\frac{\partial V_i \boldsymbol{q}_i}{\partial t} + \boldsymbol{R}_i = 0 \tag{3.28}$$

where V_i represents the volume of the i^{th} control volume, $\frac{\partial q_i}{\partial t}$ the change in the conservative variables in the i^{th} cell with respect to time, and $\boldsymbol{R}$ the spatial residual containing all numerical fluxes and, where applicable, any source terms. In general for this research $\boldsymbol{R} = \boldsymbol{R}_{\text{inviscid}} + \boldsymbol{R}_{\text{viscous}} + \boldsymbol{S}$.

3.5.2 Fluxes

The inviscid term in equation (3.1), $\nabla \cdot \boldsymbol{F}$, is integrated over the control volume and converted into a surface integral by Gauss' divergence theorem:

$$\int_{\Omega} \left(\nabla \cdot \boldsymbol{F} \right) d\Omega = \int_{\partial\Omega} \left(\boldsymbol{F} \cdot \hat{\boldsymbol{n}} \right) dS \tag{3.29}$$

The surface integral is approximated discretely as a sum over the faces defining the cell-centered control volume

$$\boldsymbol{R}_{\text{inviscid}} = \sum_{f \in \mathcal{F}} \boldsymbol{\Phi}_f \cdot \boldsymbol{n}_f \tag{3.30}$$

where $\boldsymbol{n}_f$ is the unnormalized face area vector, and $\mathcal{F}$ represents all the faces that make up the surface of the control volume. The inviscid fluxes, $\boldsymbol{\Phi}_f$, are approximated at each interface using the HLLC flux [96] formulated for the incompressible NS equations [97]. The solution variables can be reconstructed with first or second order accuracy; consequently, the fluxes may be first or second order in space as well.

Using the same technique, the viscous term, $\nabla \cdot \boldsymbol{F}_v$, is discretized via the following summation

$$\boldsymbol{R}_{\text{viscous}} = \frac{1}{Re} \sum_{f \in \mathcal{F}} \boldsymbol{F}_{vf} \cdot \boldsymbol{n}_f \tag{3.31}$$

where $\boldsymbol{F}_{vf}$ is the viscous flux evaluated using solution data at face f between two adjacent control volumes. For the incompressible NS equations, the only solution data required for the viscous flux are solution gradients. Also, it is typically considered more accurate to use weighted gradients for the viscous fluxes when they are available [91, 98]. The default approach in this research is to use the weighted least-squares gradients for these calculations if they are available. The gradients are obtained at the face by using either a simple weighted average of the gradient data of the current element and neighboring element or by using a directional derivative approach. See Hyams [91] for a thorough definition of the directional derivative. Using a weighted average is vital in this octree cell-centered paradigm because, when a face neighbor is at a different level in the octree, the interface location is twice as far from the centroid of the larger element as it is from the centroid of the smaller element; refer to Figures 3.1 and 3.2 for an illustration of this problem in two dimensions. Thus, a simple average is inferior to a weighted average because the interface is not halfway between both centroids.

3.5.3 Higher Order Spatial Accuracy

A spatially second order accurate reconstruction is implemented in this work by using solution gradient data to extrapolate cell-centered solution data to the flux interface. The extrapolation is performed using a truncated Taylor series expansion as follows

$$\phi_f = \phi_c + \boldsymbol{\nabla}\phi_c \cdot \boldsymbol{s} \tag{3.32}$$

where ϕ_f is any one of the components of $\boldsymbol{Q}$ extrapolated to the face, ϕ_c is a component of $\boldsymbol{Q}$ at the cell centroid, and $\boldsymbol{s}$ is a vector from the cell centroid to the flux interface. That is, $\boldsymbol{s} = \boldsymbol{x}_f - \boldsymbol{x}_c$ where $\boldsymbol{x}_f$ is the midpoint of the flux interface and $\boldsymbol{x}_c$ is the cell centroid. Green's theorem and the least-squares method are used in this work to obtain cell-centered solution gradients.

3.5.3.1 Green's Theorem

Green's theorem is defined as

$$\int_V \boldsymbol{\nabla}\phi dV = \int_S \phi_f \hat{\boldsymbol{n}} dA \tag{3.33}$$

where again, ϕ_f represents any component of the solution vector $\boldsymbol{Q}$ on the surface of the volume V and $\hat{\boldsymbol{n}}$ is a unit surface normal. Assuming that the components of $\boldsymbol{Q}$ vary linearly (or $\nabla\boldsymbol{Q}$ is constant) over V and replacing the surface integral with a summation over the faces of V, the discrete form of equation (3.33) is

$$\boldsymbol{\nabla}\phi = \frac{1}{V}\sum_{f\in F} \phi_f \boldsymbol{n} \tag{3.34}$$

where $\boldsymbol{n}$ is the area vector belonging to face f. Also, ϕ_f is obtained by a weighted average of solution values from either side of the face f; the weighting procedure used here is a distance weighted average which is essentially a linear interpolation of the solution data from each cell centroid to the face midpoint.

3.5.3.2 *Least-Squares Gradient*

The least-squares method for computing solution gradients is a much more complicated approach. It may be derived generally using the following truncated Taylor series to express a solution value at a neighboring cell center, e_j, in terms of the solution data in a central cell, e_i, as follows

$$\phi_j = \phi_i + (\boldsymbol{x}_j - \boldsymbol{x}_i) \cdot \boldsymbol{\nabla}\phi_i \tag{3.35}$$

Writing this equation once for every neighboring cell, e_i, usually leads to an over determined system of equations, represented by equation (3.36), that must be solved in order to get an approximation of the cell-centered solution gradient $\boldsymbol{\nabla}\phi_i$.

$$\begin{bmatrix} \Delta x_1 & \Delta y_1 & \Delta z_1 \\ \Delta x_2 & \Delta y_2 & \Delta z_2 \\ \vdots & \vdots & \vdots \\ \Delta x_n & \Delta y_n & \Delta z_n \end{bmatrix} \begin{bmatrix} \phi_{xi} \\ \phi_{yi} \\ \phi_{zi} \end{bmatrix} = \begin{bmatrix} \Delta\phi_1 \\ \Delta\phi_2 \\ \vdots \\ \Delta\phi_n \end{bmatrix} \tag{3.36}$$

Note that $\Delta()_j = ()_j - ()_i$ where $()$ is any of x, y, z, or ϕ. Values in the central element are represented by an i subscript, $()_i$, while an n subscript, $()_n$, represents data in a neighbor element, e_n. Arbitrary weights may be assigned to each equation in equation (3.36) individually to obtain weighted least-squares gradients. A common approach is to weight each equation by the inverse of the distance between the two adjacent cell centroids. Notice that since the higher order terms of the Taylor series expansion have been dropped, the least-squares method described here is only able to exactly calculate gradients of a linear function.

Since the system in equation (3.36) is over determined, a least-squares solution procedure must be utilized. If the system is represented by $\mathcal{A}\boldsymbol{x} = \boldsymbol{b}$, a least-squares solution may be obtained by solving the system attained by pre-multiplying each side of the system by the transpose of $\mathcal{A}$, e.g.

$$\mathcal{A}^T\mathcal{A}\boldsymbol{x} = \mathcal{A}^T\boldsymbol{b} \tag{3.37}$$

In order to accomplish this, the matrix $\mathcal{A}$ is factorized using a QR algorithm, which employs the Gram-Schmidt process for orthogonalization. This process is quite complicated to derive but, in the end, turns out to be relatively simple to implement. For a thorough derivation of this method, as well as some important implementation issues, see Appendix A of Hyams' dissertation [91].

Gradient calculations must be carried out carefully in the context of an octree grid with hanging nodes. An element with neighbors at different levels in the octree, i.e. with neighbors that are significantly larger or smaller than itself, may have difficulty obtaining accurate gradients in the direction of the larger neighbor. Green's theorem and both weighted and unweighted least-squares approaches have been examined for gradient computations in this research. The least-squares approach has been shown to be superior to Green's theorem in the context of a grid with hanging nodes. The naïve implementation of Green's theorem tends to introduce spurious solution gradients in the case where a neighboring element is larger than the central element in which gradient data are required. What follows is a specific example of this problem. It is well known that the least-squares method exactly calculates the gradient of a linear function; in fact, even the Green-Gauss method does the same in the case of a Cartesian aligned mesh. Furthermore, the least-squares method does not care what the grid looks like nor about the relative size of neighboring elements. It still exactly

represents a linear gradient even if there are hanging nodes. The Green-Gauss method, on the other hand, breaks down in this situation; it produces significant errors in the gradients calculated within any element that has neighbors that are *bigger* than the element itself. This phenomenon is demonstrated by a very simple and standard test case which is set up as follows. The grid is a unit cube with anisotropic spacing in all three directions, (see Figure 3.3), and the solution variables are each initialized to a different linear function as follows:

$$\begin{aligned} p &= 1.0x + 2.0y + 3.0z \\ u &= 4.0x + 5.0y + 6.0z \\ v &= 7.0x + 8.0y + 9.0z \\ w &= 10.0x + 11.0y + 12.0z \\ T &= 13.0x + 14.0y + 15.0z \end{aligned} \tag{3.38}$$

Gradients of this solution state are calculated using each of the available gradient methods on both

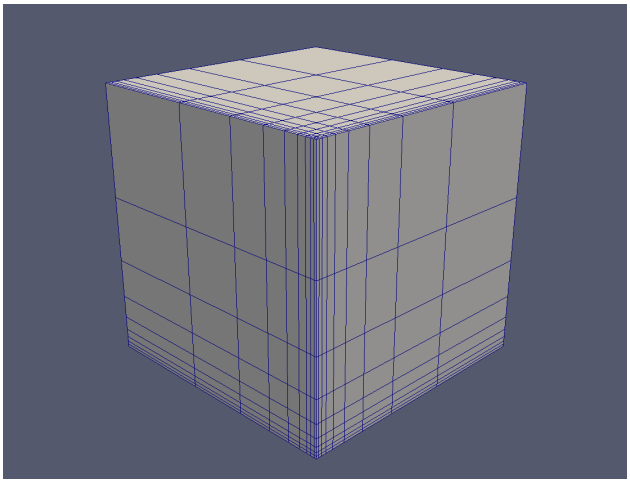


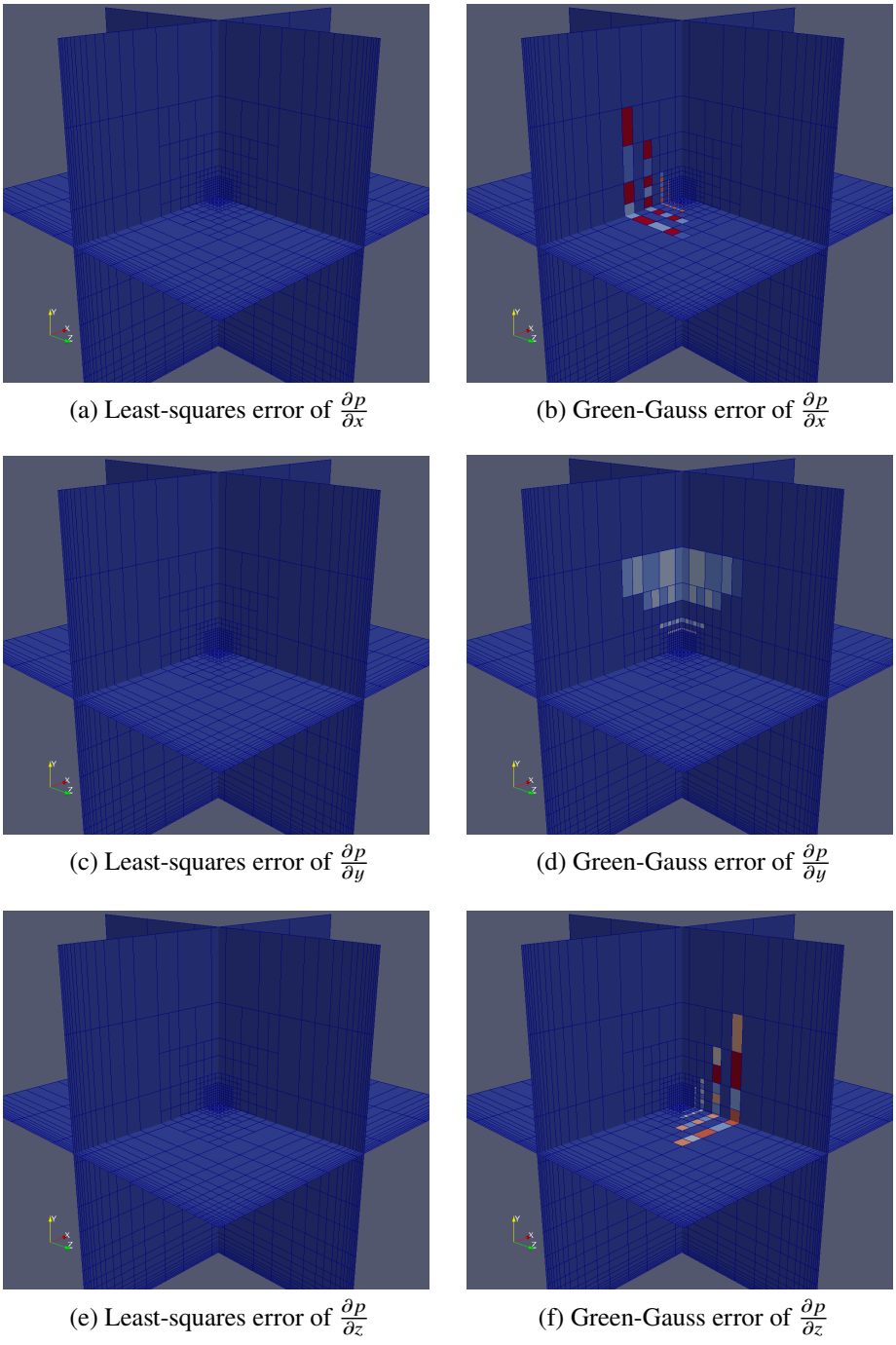
Figure 3.3 Unit cube with anisotropy in all three dimensions used for gradient testing

the uniformly and non-uniformly refined unit cube mesh. As stated before, both the Green-Gauss and weighted/un-weighted least-squares methods have no difficulty recovering the exact gradients of these linear solution states when the mesh is uniform. However, if non-uniform refinement is forced in the center of the unit cube, the Green-Gauss method begins to break down. Figure 3.4 explicitly demonstrates this behavior; there is no noticeable error in any of the least-squares gradients (the actual error is around machine zero). Notice carefully, however, that the Green-Gauss gradients demonstrate significant errors in the particular manner mentioned above. Any x gradient computed using the Green-Gauss method on a non-uniform grid in an element that has a *larger* neighbor across one of its x faces is markedly inaccurate. The same applies to y gradients and z gradients in elements with larger neighbors across their y and z faces, respectively. Careful examination of Figures 3.4a, 3.4c and 3.4e corroborates this description.

Tables 3.1 to 3.3 give a more quantitative overview of this shortcoming of the Green-Gauss gradient method. As expected, the maximum percent error in both the least-squares and weighted least-squares methods are very small. However, in at least one case, the maximum percent error in the Green-Gauss method is over 200%, and the minimum percent error is around 50%, for this test case.

There may be ways to alleviate or improve this shortcoming of the Green-Gauss method by exploring different ways to 'weight' cell-centered values when averaging them to the face; currently, a distance-weighted average is computed, but a volume-weighted approach could be tested, and there are likely other more sophisticated methods that could be used as well.

Finally, it should be noted that it is commonly accepted that unweighted gradients are best used for reconstruction of non-linear data, i.e., for higher order inviscid fluxes, while it is



(a) Least-squares error of $\frac{\partial p}{\partial x}$

(b) Green-Gauss error of $\frac{\partial p}{\partial x}$

(c) Least-squares error of $\frac{\partial p}{\partial y}$

(d) Green-Gauss error of $\frac{\partial p}{\partial y}$

(e) Least-squares error of $\frac{\partial p}{\partial z}$

(f) Green-Gauss error of $\frac{\partial p}{\partial z}$

Figure 3.4 Comparison of the errors produced by the least-squares and Green-Gauss gradient methods on a non-uniform mesh; the error scale ranges from 0.0 to 1.0

Table 3.1 Comparison of the maximum percent errors obtained using the weighted and unweighted least-squares and Green-Gauss gradient methods on the p and u solution variables in equation (3.38) on the unit cube shown in Figure 3.3

Gradient method	$\frac{\partial p}{\partial x}$	$\frac{\partial p}{\partial y}$	$\frac{\partial p}{\partial z}$	$\frac{\partial u}{\partial x}$	$\frac{\partial u}{\partial y}$	$\frac{\partial u}{\partial z}$
least-squares	1.4×10^{-11}	5.4×10^{-12}	6.5×10^{-12}	8.5×10^{-12}	4.3×10^{-12}	6.4×10^{-12}
weighted least-squared	1.3×10^{-11}	5.6×10^{-12}	5.7×10^{-12}	9.9×10^{-12}	5.7×10^{-12}	6.1×10^{-12}
Green-Gauss	236.2	94.5	47.2	129.9	94.5	70.8

Table 3.2 Comparison of the maximum percent errors obtained using the weighted and unweighted least-squares and Green-Gauss gradient methods on the v and w solution variables in equation (3.38) on the unit cube shown in Figure 3.3

Gradient method	$\frac{\partial v}{\partial x}$	$\frac{\partial v}{\partial y}$	$\frac{\partial v}{\partial z}$	$\frac{\partial w}{\partial x}$	$\frac{\partial w}{\partial y}$	$\frac{\partial w}{\partial z}$
least-squares	6.5×10^{-12}	3.9×10^{-12}	5.9×10^{-12}	5.7×10^{-12}	5.0×10^{-12}	6.2×10^{-12}
weighted least-squared	6.9×10^{-12}	6.4×10^{-12}	7.0×10^{-12}	6.3×10^{-12}	8.2×10^{-12}	5.4×10^{-12}
Green-Gauss	114.7	94.5	78.7	108.6	94.5	82.7

Table 3.3 Comparison of the maximum percent errors obtained using the weighted and unweighted least-squares and Green-Gauss gradient methods on the T solution variable in equation (3.38) on the unit cube shown in Figure 3.3

Gradient method	$\frac{\partial T}{\partial x}$	$\frac{\partial T}{\partial y}$	$\frac{\partial T}{\partial z}$
least-squares	4.2×10^{-12}	4.1×10^{-12}	6.5×10^{-12}
weighted least-squared	4.9×10^{-12}	5.5×10^{-12}	6.8×10^{-12}
Green-Gauss	105.4	94.5	85.0

recommended to use their weighted counterparts when gradient data are being extrapolated, i.e. for viscous fluxes [91, 98].

3.5.3.3 Boundary Gradients

Boundary gradients are handled differently than usual in this research; the boundary gradients are not calculated in the same way as the interior gradients. The blocky and non-uniform nature of the boundaries in the meshes produced by this application makes boundary gradients challenging to implement. Thus, in the spirit of 'zeroth' order approximation, this issue has been addressed by taking advantage of cell-centered gradient data to approximate boundary gradients. This means that cell-centered gradient data from the element adjacent to the boundary face are used, either directly or with modification, when boundary gradients are needed. When the boundary face is part of a flow-through or farfield type boundary, the cell-centered gradients are used directly. However, when the boundary face is part of a wall or symmetry boundary condition, the cell-centered gradient data are copied and modified to take into account any Neumann type gradient conditions. This includes zero-gradient temperature (adiabatic) or pressure conditions as well the buoyant condition on the pressure gradient. Symmetry boundary conditions simply require that components of the gradient normal to the boundary are eliminated. Most importantly, normal velocity gradients are calculated on wall boundaries using a simple one-sided, two-point finite difference using both the solution data at the wall and at the adjacent cell center. Since the normal wall velocity gradients play such an important role in viscous flows, it is an unacceptably poor approximation to simply use the values of the cell-centered velocity gradients at the wall. Transverse velocity gradients are explicitly set to zero at the wall for no-slip walls or, in the case of inviscid walls, allowed to retain

their cell-centered values. These approximations have proven to be acceptably accurate, and they help keep the implementation simple and efficient.

3.5.4 Temporal Discretization

The temporal term in equation (3.1) remains to be approximated after the spatial terms have been discretized. Following Beam and Warming [99], and Hyams [91], the following generalized difference expression is used for the temporal discretization

$$\Delta(V\boldsymbol{q})^n = \frac{\theta\Delta t}{1+\xi}\frac{\partial(\Delta(V\boldsymbol{q})^n)}{\partial t} + \frac{\Delta t}{1+\xi}\frac{\partial((V\boldsymbol{q})^n)}{\partial t} + \frac{\xi}{1+\xi}\Delta(V\boldsymbol{q})^{n-1} \tag{3.39}$$

where $\Delta(V\boldsymbol{q})^n = (V\boldsymbol{q})^{n+1} - (V\boldsymbol{q})^n$. Various schemes with various orders of accuracy may be chosen based on the values chosen for the parameters θ and ξ; e.g., $\theta = 1$, $\xi = 0$ is a first-order Euler implicit scheme, whereas $\theta = 1$, $\xi = 0.5$ is an Euler implicit scheme with second-order accuracy (the BDF2 scheme). Other combinations are presented by Beam and Warming [99]. For this research only the two aforementioned Euler implicit schemes have been implemented, both of which have $\theta = 1$, thus equation (3.39) may be simplified into

$$\Delta(V\boldsymbol{q})^n = \frac{\Delta t}{1+\xi}\frac{\partial(V\boldsymbol{q})^{n+1}}{\partial t} + \frac{\xi}{1+\xi}\Delta(V\boldsymbol{q})^{n-1} \tag{3.40}$$

Using equation (3.28) to eliminate the time derivative term gives

$$\frac{\Delta(V\boldsymbol{q})^n - \frac{\xi}{1+\xi}\Delta(V\boldsymbol{q})^{n-1}}{\Delta t} = -\frac{\boldsymbol{R}^{n+1}}{1+\xi} \tag{3.41}$$

Since in this research the grid is static, V does not vary in time and thus the Geometric Conservation Law (GCL) does not need to be applied. Thus, equation (3.41) simplifies to

$$\frac{V}{\Delta t}\left[(1+\xi)\Delta \boldsymbol{q}^{n} - \xi \Delta \boldsymbol{q}^{n-1}\right] = -\boldsymbol{R}^{n+1} \tag{3.42}$$

For a good explanation of how the GCL terms would be added, see Hyams [91]. In order to maintain a divergence free velocity field for incompressible flow regimes, the contribution from the time derivative term (and potentially the GCL term) are explicitly left out of the continuity equation.

3.5.5 Time Evolution

Equation (3.42) is the final form of the temporal and spatial discretization; the implicit nature of the temporal discretization requires that the spatial residual, $\boldsymbol{R}$, be evaluated at time level $n+1$. Since the solution at this time level is unknown, Newton's method will be used to linearize the equations about the known solution state $\boldsymbol{Q}^{n}$. Note that the following derivation follows closely with that of Hyams [91]. From equation (3.42) the following equation may be derived

$$\mathbb{F}^{n+1}(\boldsymbol{Q}^{n+1}) = \frac{V}{\Delta t}\left[(1+\xi)\Delta \boldsymbol{q}^{n} - \xi \Delta \boldsymbol{q}^{n-1}\right] + \boldsymbol{R}^{n+1} \tag{3.43}$$

where $\mathbb{F}^{n+1}$ is the function that should be driven to zero by Newton's method. $\mathbb{F}^{n+1}$ is now expanded using a Taylor series about a known level $(n+1, m)$

$$\mathbb{F}^{(n+1,m+1)} \approx \mathbb{F}^{(n+1,m)} + \frac{\partial \mathbb{F}^{(n+1,m)}}{\partial \boldsymbol{Q}} \Delta \boldsymbol{Q}^{(n+1,m)} + \mathcal{O}(\Delta \boldsymbol{Q}^{2}) \tag{3.44}$$

Lastly, since the left hand side of the equation above is zero upon convergence of the Newton's method, and after dropping the $O(\Delta \boldsymbol{Q}^2)$ term, equation (3.44) may be written as

$$-\mathbb{F}^{(n+1,m)} = \frac{\partial \mathbb{F}^{(n+1,m)}}{\partial \boldsymbol{Q}} \Delta \boldsymbol{Q}^{(n+1,m)} \tag{3.45}$$

Expanding these terms and differentiating $\mathbb{F}$ yields the final, full expression for Newton's method

$$-\left[\frac{V}{\Delta t}\left[(1+\xi)\Delta \boldsymbol{q}^n - \xi \Delta \boldsymbol{q}^{n-1}\right] + \boldsymbol{R}^{n+1}\right] = \frac{\partial \boldsymbol{R}^{(n+1,m)}}{\partial \boldsymbol{Q}} \Delta \boldsymbol{Q}^{(n+1,m)} \tag{3.46}$$

where $\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{Q}}$ is the Jacobian matrix.

The sparse matrix system $\mathcal{A}\boldsymbol{x} = \boldsymbol{b}$, represented by equation (3.46), is solved iteratively using an 'improved' Symmetric Gauss-Seidel method, which is explained in detail by Hyams [91] in Section 4.4.2.

3.5.6 Boundary Conditions

This section gives an overview of the boundary condition implementation. Finite volume schemes use the concept of 'ghost' elements—ghost nodes for node-centered schemes and ghost cells for cell-centered schemes—to facilitate the implementation of boundary conditions. These ghost elements store appropriate solution data, which are used to reconstruct solution values required at the boundary sufficient to enforce the specified boundary condition. Since this research implements a cell-centered scheme, the boundary values specified by each particular boundary condition are reflected into the ghost element in such a way as to enforce that the average of the

ghost and interior elements equals the boundary value unless otherwise noted, e.g.,

$$\frac{q_{\text{interior}} + q_{\text{ghost}}}{2} = \begin{Bmatrix} p_{\text{wall}} \\ u_{\text{wall}} \\ v_{\text{wall}} \\ w_{\text{wall}} \\ T_{\text{wall}} \end{Bmatrix} \tag{3.47}$$

3.5.6.1 Flow-Through/Far-Field Boundaries

Far-field boundaries are implemented using the standard Characteristic Variable Boundary Conditions (CVBCs), wherein the governing equations are rewritten in the direction normal to the boundary, and standard characteristic variable analysis is used to decouple the system. For a detailed explanation of this method see Section 4.5.2 of Hyams' dissertation [91].

3.5.6.2 Inviscid Solid Wall Boundaries

Inviscid solid wall boundary conditions are imposed by forcing tangential boundary flow; this is done during the evaluation of the solid wall flux by enforcing $\boldsymbol{u} \cdot \boldsymbol{n} = un_x + vn_y + wn_z = 0$, where $\boldsymbol{u}$ is the fluid velocity vector and $\boldsymbol{n}$ is the boundary face area vector. This implies that the only contributions to the inviscid flux at these boundaries come from the pressure terms in the momentum equations. Neumann boundary conditions are used for the pressure and temperature terms on slip boundaries; the condition on pressure is usually a zero pressure gradient, however, in buoyant cases, the Neumann condition for pressure, presented in Section 3.5.6.4, must be used instead.

Two conditions for the wall temperature are supported: adiabatic or isothermal walls. For adiabatic walls the normal temperature gradient is simply enforced to be zero by setting the boundary temperature to the same as the interior temperature, while for isothermal walls the temperature on the wall is set to the specified boundary temperature in a Dirichlet manner. Solid wall inviscid fluxes are evaluated in one of two ways, either by using the HLLC flux with the solution state in the ghost element defined such that the average of the interior and ghost state results in the appropriate boundary values for the inviscid wall, as specified in equation (3.47), or the inviscid flux vector $\boldsymbol{F}$, as defined in equation (3.2), is evaluated directly with boundary values appropriate for the boundary. This last approach is necessary because in certain cases, notably the mixed convection cavity case presented in Section 4.2.3, simulations do not fully converge, nor do they yield accurate solutions using the HLLC flux. However, when evaluating the flux vector directly, the solution is accurate and the convergence behavior is improved. In other cases this approach has been demonstrated to make very little difference compared to using a left and right state to evaluate the HLLC flux, and, for this reason, it has been adopted as the default method used to evaluate inviscid fluxes on solid wall.

3.5.6.3 No-Slip Solid Wall Boundaries

No-slip walls have a straight forward implementation; wall velocities are imposed directly, and this implementation supports imposing moving boundaries. The same Neumann boundary conditions for the pressure and temperature terms used in the inviscid/slip wall boundary condition defined above are used for no-slip boundaries as well. Since boundary data are not explicitly solved

for in this cell-centered paradigm, nothing special must be done to the Jacobian for this boundary condition.

3.5.6.4 Buoyant Solid Wall Boundary Condition

In cases where buoyancy is important, an additional Neumann boundary condition for pressure on wall boundaries must be implemented. This boundary condition is outlined by Kress [88]. In this case, in lieu of using a zero pressure gradient, the buoyant pressure gradient is set equal to the buoyant source term explained in Section 3.5.7. Namely,

$$\frac{\partial p}{\partial \eta} = \frac{Gr}{Re^2} T \tag{3.48}$$

where η represents the direction normal to the wall. This boundary condition must be used on both slip and no-slip walls in simulations where buoyancy is important.

3.5.6.5 Percent-of-Slip Wall Boundaries

In certain situations neither the no-slip nor slip wall boundary conditions are fully applicable. Specifically, in the types of urban simulations for which this application is designed, the elements adjacent to the solid walls (buildings, blockages, ground, etc.) often have an off-the-wall spacing of approximately 1 m to 10 m. At this resolution it does not make sense to impose a full no-slip boundary condition at the solid wall boundary, as the solution data at the cell centroid is relatively far from the boundary. In this scenario, it has been observed that the no-slip boundary condition can overly influence the flow near the walls and retards the flow and turbulent mixing, especially in street canyons. However, the viscous effect of the wall on the flow cannot be completely

neglected as vorticity it generates is needed to model and generate turbulence. Thus, a full slip boundary condition is arguably also inappropriate. For this reason, a parameterized 'percent-of-slip' boundary condition, developed by Nichols [95], has been implemented. The intent of this boundary condition is to provide viscous effects near viscous walls without overly damping the flow. This boundary condition simply allows the user to decelerate the flow at the solid wall by allowing only a specified percentage of the computed slip velocity at the wall to be used to compute the final wall velocity. The implementation of this boundary condition is simple, yet effective. The boundary velocity is calculated as with a slip boundary, but with the resulting velocity components scaled by a user-specified percentage. This is functionally a parameterized wall boundary condition that exists somewhere between the normal slip and no-slip wall boundary conditions.

In practice, this boundary condition proves to be very useful. With a full no-slip boundary condition the velocity field does not mix very well; i.e., cross flows are inhibited and alleyways often do not 'feel' the effect of the bulk flow. However, when this percent-of-slip boundary condition is used, much better mixing is observed. This will be demonstrated in Chapter 4. However, it should be noted that using this approximation is obviously a 'modeling' issue and more research is needed to thoroughly 'validate' its use in urban environments that deviate substantially from those explored in the present work.

3.5.6.6 Custom Inflow Boundary Conditions

This application supports two custom inflow boundary conditions: a parabolic profile (for internal flow situations) and a power law profile (for atmospheric boundary layer type applications).

All parameters for both inflow profiles are specifiable at runtime, and the interface is general enough to allow for easy addition of new custom inflow boundary conditions.

3.5.7 Source Terms

If the source terms, $\boldsymbol{S}$, in equation (3.1) are non-zero, as is the case for the buoyant NS equations among others, it must be discretized. The source term in the buoyant NS equations, described by equations (3.1) and (3.24), is applied as a volume weighted source term. In this case, the residual associated with the momentum equations are augmented by the addition of the following term

$$\frac{VGrT}{Re^2}\boldsymbol{e}_g \tag{3.49}$$

where V is the volume of the current element, T is its temperature, and $\boldsymbol{e}_g$ is the normalized vector pointing in the direction of gravitational acceleration.

CHAPTER 4

NUMERICAL RESULTS

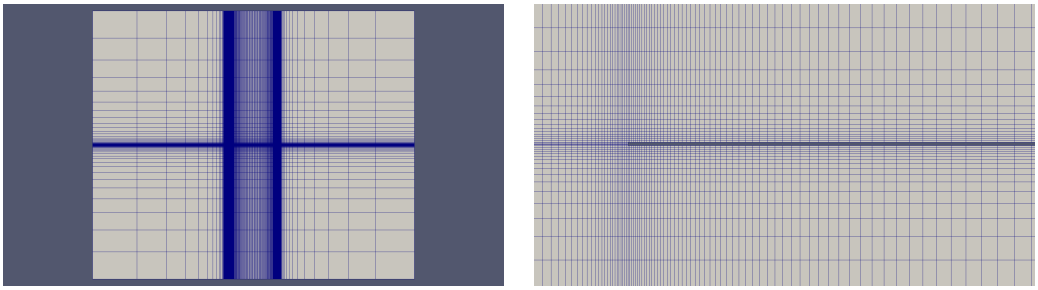
This chapter will present and discuss various test cases used to validate the flow solver implemented for this project.

4.1 Laminar Flat Plate

The flat plate is one of the fundamental test cases used to validate any implementation of a Navier-Stokes solver. The theoretical solution to laminar flat plate flow was derived by Paul Richard Heinrich Blasius [100] in 1908. The Blasius solution is the *de facto* laminar Navier-Stokes test case and will be used here to verify the current implementation.

A structured UGRID mesh is provided to *p4est* as a skeleton for this test case and is run without any extra refinement. The flat plate is two-dimensional, two units long and two-sided; it is floating in free space with a farfield extending five units away from the plate in the x and y directions. In the z direction there are four planes—corresponding to three elements—in order to ensure that there is one slice of elements completely interior to the mesh. Since it is a two-sided flat plate and *p4est* only supports Cartesian aligned elements, the leading and trailing edges are planar with a thickness of 0.002 units; four elements span this distance. There are 200 elements in the streamwise direction; the normal/off-the-wall spacing is 0.001 units, and points are clustered toward the leading and trailing edge of the plate using a hyperbolic tangent distribution with leading

and trailing initial spacing the same as the normal spacing. There are 34 elements above and below the plate, 44 upstream, and 24 downstream. See Figure 4.1 for details about the grid.



(a) Zoomed out view (b) Leading edge veiw

Figure 4.1 Flat plate grid

The flat plate test case was run with a Reynolds number of 10,000, based on the plate length, until steady state convergence was achieved. Three profiles were extracted and plotted against the Blasius profile; profile one corresponds to $\frac{L}{3}$, profile two $\frac{L}{2}$, and profile three $\frac{3L}{4}$ where $L = 2$ is the length of the plate. As seen in Figure 4.2 the three profiles are self similar and match the Blasius solution well.

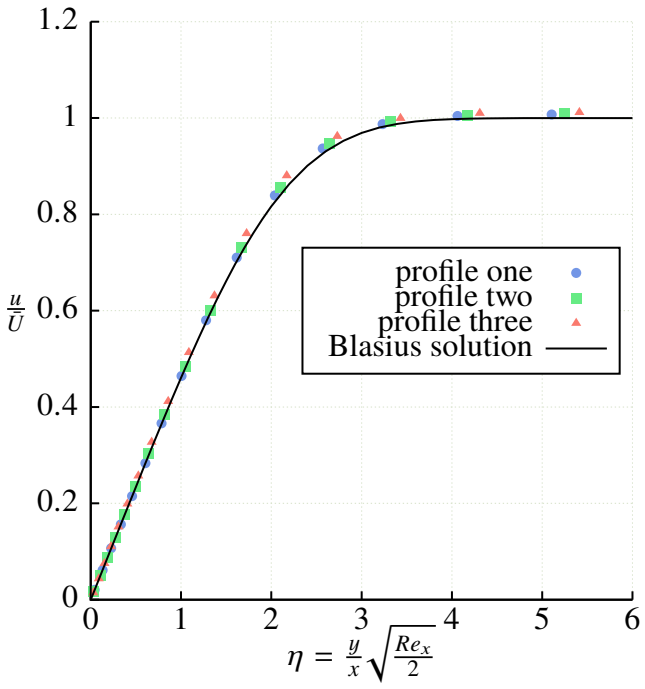
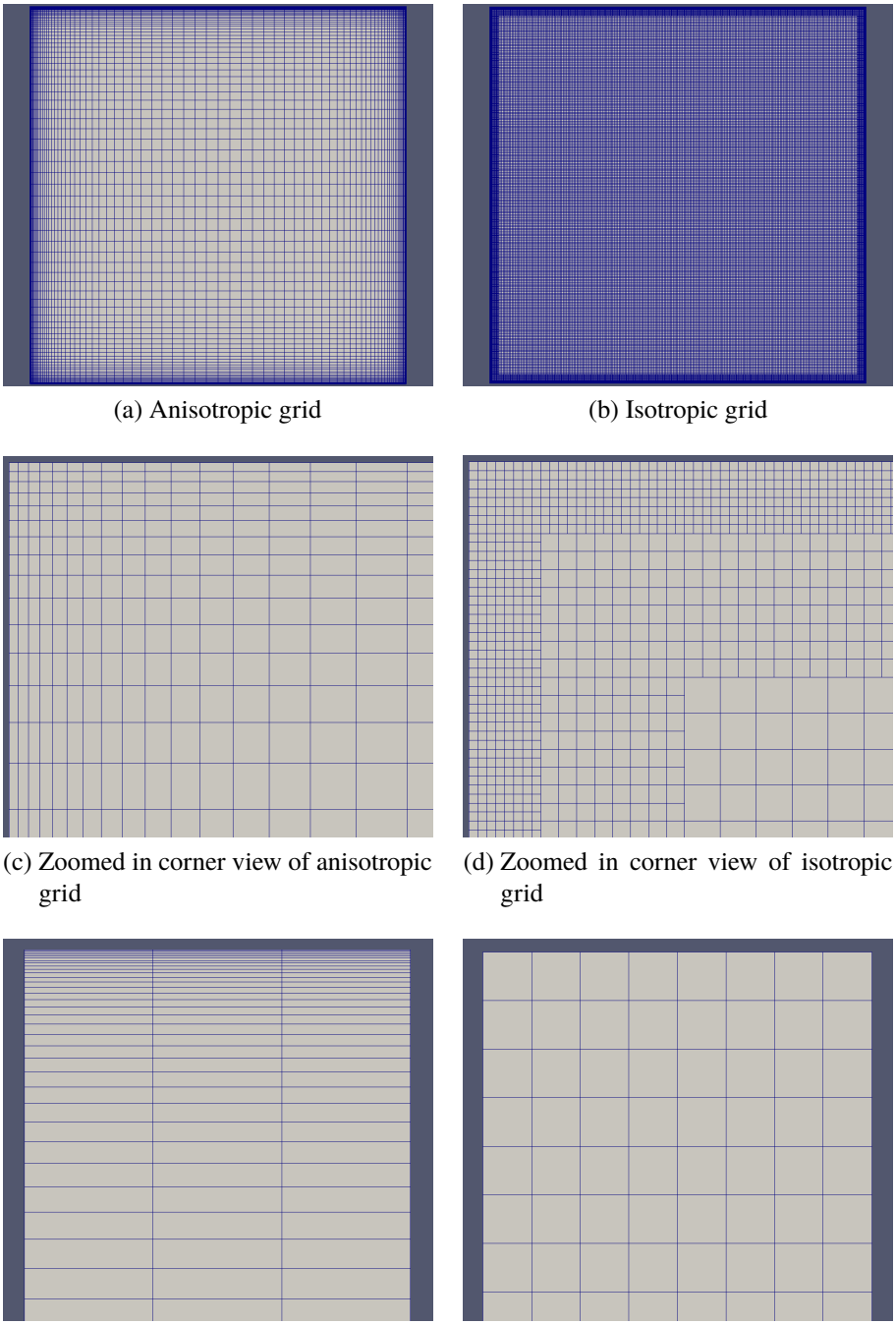


Figure 4.2 Flat plate Blasius comparison solution with Re = 10,000

4.2 Cavity Validation Cases

This section presents various validation cases run on a square cavity geometry. The following test cases are designed to test aspects of the code related to convection and heat transfer. The geometry is a simple unit square extruded a small distance in the z direction so the simulation may be treated as two-dimensional. These cases were run using two different types of grids: a uniform mesh, i.e., all elements were required to be at the same refinement level, and a non-uniform mesh containing elements at different refinement levels. The uniform cases were run using almost the same $81 \times 81 \times 2$ structured mesh that Kress [88] and Er [89] used to obtain their results, except that it contains one more plane in the z direction in order to have a stack of completely interior elements. Both grids have normal/viscous spacing of 1×10^{-3}. This is easy to accomplish with

the uniform grid; the non-uniform grid, however, requires more work. The non-uniform grid was designed to have three levels of refinement such that the elements at level three were isotropic cubes achieving the target normal spacing. As such the non-uniform meshes had a z dimension of 8×10^{-3} with levels three, two, and one each containing eight, four, and two elements in the z direction respectively. Given the way these grids are designed, they exhibit dramatically different cell counts: the uniform grid comprises 19,200 elements while 429,192 elements compose the non-uniform grid. This explicitly demonstrates a potential disadvantage of using isotropic, non-uniform octree meshes as their cell counts can grow very quickly. In these cases, both types of meshes are used to compare the accuracy of *paros* run on two fundamentally different mesh types. The case using the non-uniform mesh exercises parts of the code that the uniform mesh does not: namely the logic that deals with hanging nodes/faces. Refer to Figure 4.3 for a comparison of both cavity grids used for the following simulations.



(a) Anisotropic grid

(b) Isotropic grid

(c) Zoomed in corner view of anisotropic grid

(d) Zoomed in corner view of isotropic grid

(e) Zoomed in side view of anisotropic grid

(f) Zoomed in side view of isotropic grid

Figure 4.3 Comparison of anisotropic and isotropic cavity grids

4.2.1 Natural Convection

The natural convection case is designed to test the heat transfer/buoyancy ability of the code. The case is defined as follows: the flow is initialized at rest, adn the x-min no-slip solid wall is kept at a constant non-dimensional temperature of one, while the x-max wall is kept at a constant non-dimensional temperature of zero. Both the x-min and x-max walls have a zero pressure gradient condition enforced. The y-min and y-max walls are adiabatic and have the non-zero Neumann buoyancy source term condition enforced, $\frac{\partial p}{\partial \eta} = \frac{Gr}{Re^2}T$, as described in Section 3.5.6.4. The z-min and z-max walls have symmetry boundary conditions applied. Figure 4.4 shows a schematic of the geometry, as well as the initial and boundary conditions for this natural convection cavity case.

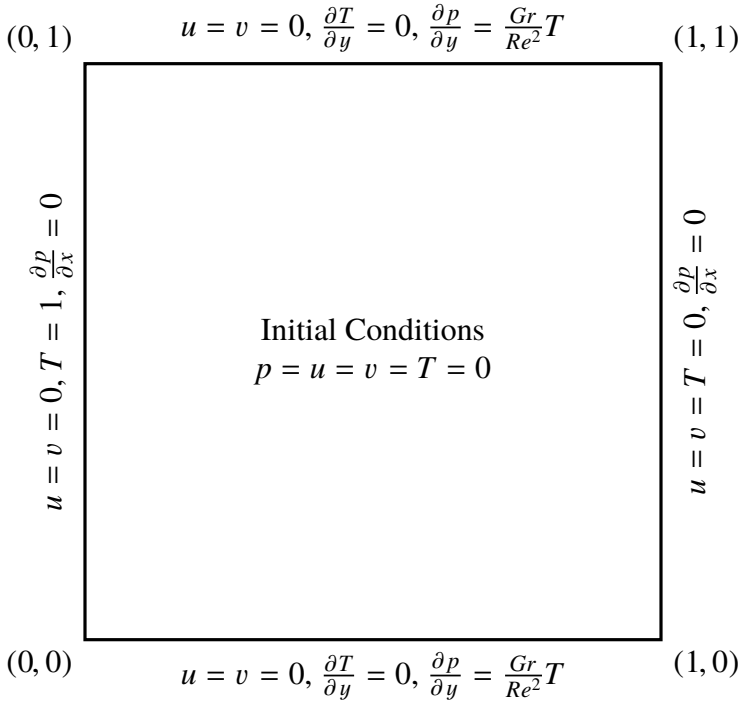


Figure 4.4 Natural convection cavity geometry and boundary condition schematic

The steady state solution for this case is shown in Figure 4.5; note the large recirculation zone in the center of the cavity driven by the heated wall and buoyancy terms added to the equations. The dashed black lines represent the lines along which velocity profiles will be extracted for comparison.

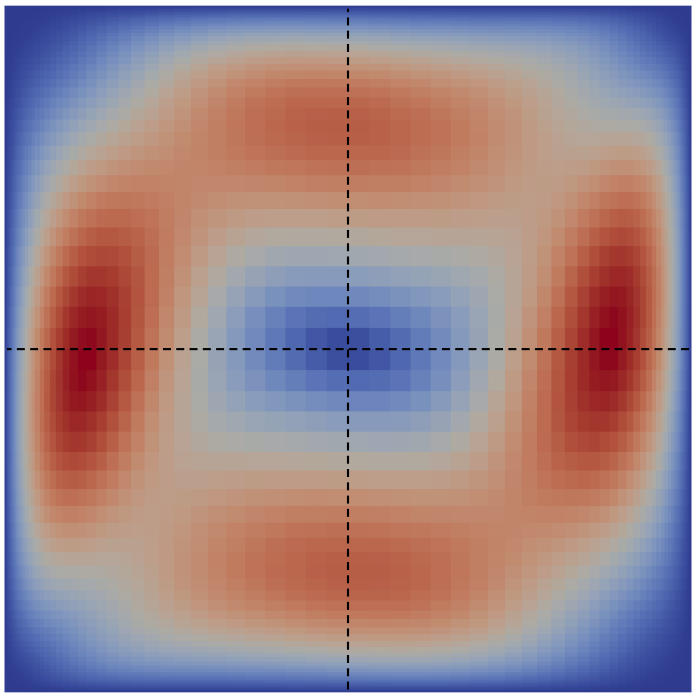


Figure 4.5 Steady state solution showing velocity magnitude in the natural convection cavity case

Davis [101] presents a benchmark numerical solution to this problem, and Kress [88] and Er [89] both present results obtained via the pseudo-compressibility approach. The maximum value of both the x and y velocity values, u and v, respectively, were tabulated by Davis in his benchmark for three different Rayleigh numbers. Table 4.1 shows that the current results match well with these values as well as those reported by Kress [88] and Er [89].

Table 4.1 Comparison of the maximum velocity values along the center lines of the natural convection cavity with $Ra = 1 \times 10^4$

	Davis	Er	Kress	Current ansio	Current iso
maximum u	0.192	0.192	0.191	0.191	0.192
maximum u location	0.823	0.850	0.816	0.825	0.822
maximum v	0.233	0.232	0.232	0.231	0.232
maximum v location	0.119	0.125	0.119	0.126	0.118

Velocity profiles along cavity center lines are shown below, and there is excellent agreement between the results using both the anisotropic and isotropic meshes as well as the results presented by Kress [88].

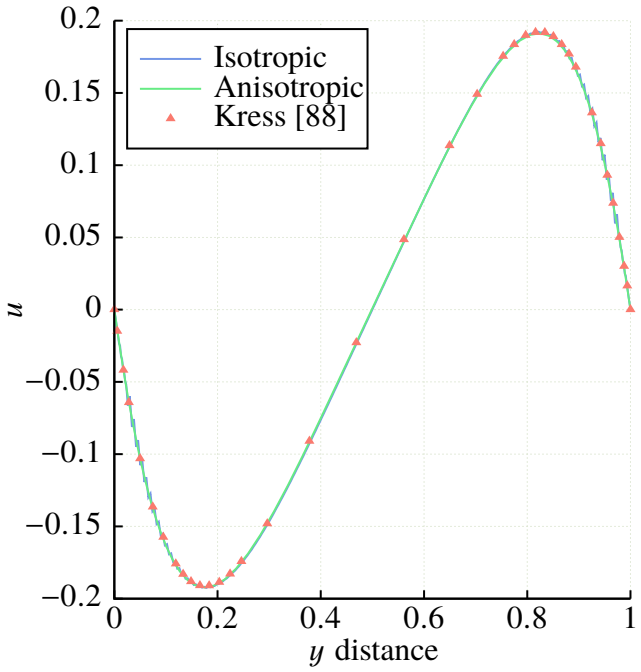


Figure 4.6 Comparison of the vertical velocity profile for natural convection with $Re = 1000$

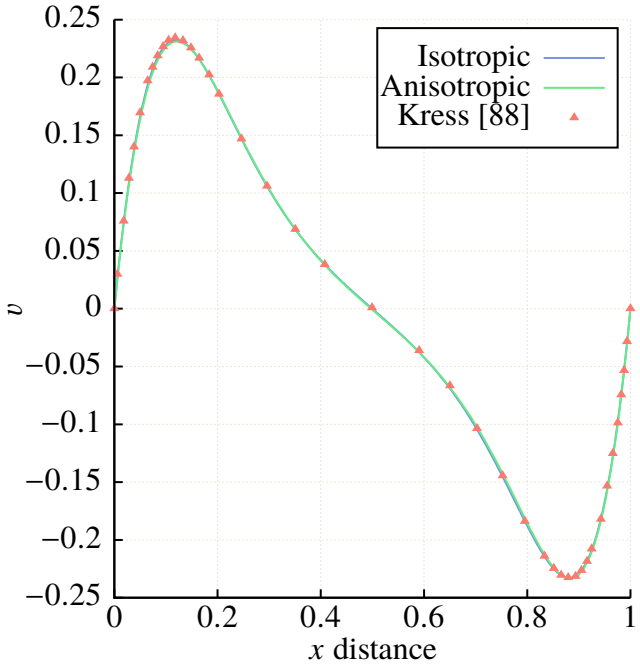


Figure 4.7 Comparison of the horizontal velocity profile for natural convection with $Re = 1000$

4.2.2 Forced Convection

Presented here is the canonical lid driven cavity test case used to validate the convective terms of the code. The x-min, x-max, y-min, and y-max boundaries are no-slip solid walls and have zero pressure and temperature gradient conditions enforced. The y-max boundary has an imposed non-dimensional velocity of $u = 1.0$ and $w = v = 0.0$, which drives the flow in the cavity. Again, the z-min and z-max walls have symmetry boundary conditions applied. The velocity is initialized to zero everywhere else. The geometry, initial, and boundary condition schematic corresponding to this case is given in Figure 4.8.

The case is run viscous and steady and, depending on the Reynolds number, may have multiple recirculation zones. Ghia et al. [102] and Mandal et al. [103] study this case in depth.

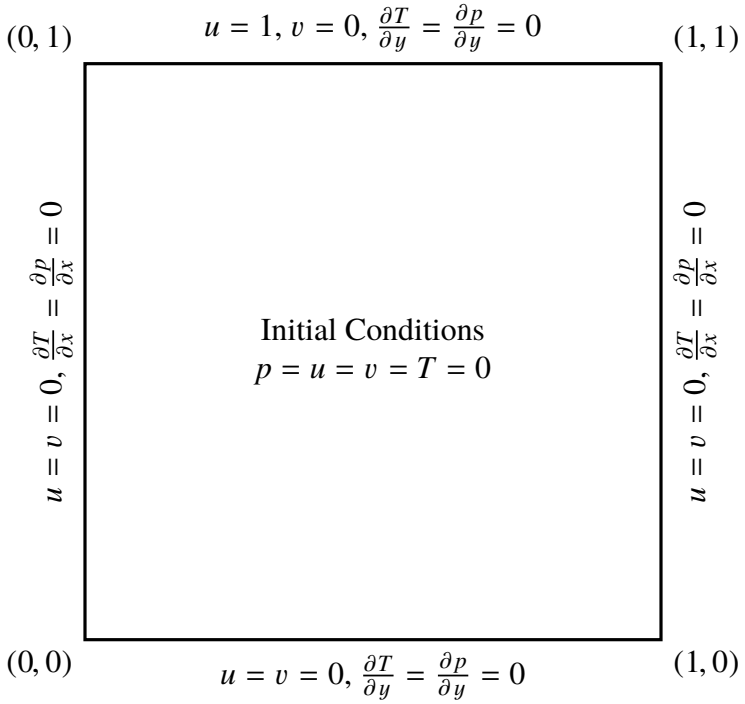


Figure 4.8 Forced convection cavity geometry and boundary condition schematic

Ghia et al. [102] studies this case with Reynolds numbers ranging from one hundred to ten thousand and presents ample solution data, which are used for validation in this case.

With a Reynolds number of one thousand, the steady solution has one large vortex that fills most of the cavity with two secondary, counter rotating vortices in the bottom right corner and a single counter rotating vortex in the bottom left corner. The steady state solution for this case is shown in Figure 4.9; note the large recirculation zone in the center of the cavity which, in this case, is forced by the moving top boundary of the cavity. The dashed black lines represent the lines along which velocity profiles are extracted for comparison. Streamlines and stagnation points agree well

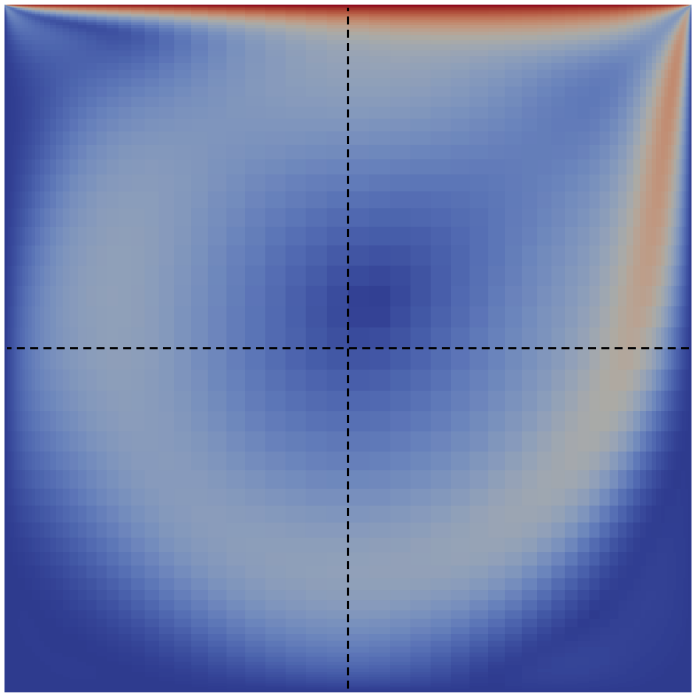


Figure 4.9 Steady state solution showing velocity magnitude in the forced convection cavity case

with Ghia et al. Velocity profiles extracted from the center lines of the cavity are also in excellent

agreement with Ghia et al., as demonstrated in Figures 4.10 and 4.11. The numerical profiles used for comparison were extracted from Tables I and II in the paper by Ghia et al. [102].

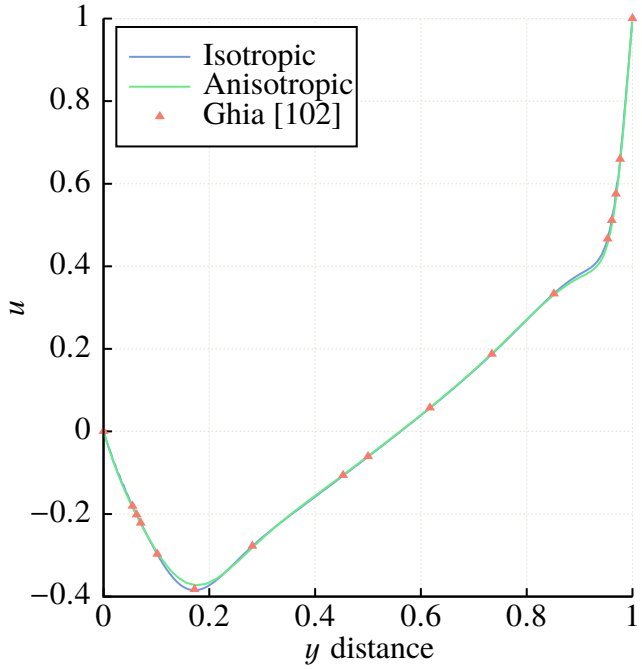


Figure 4.10 Comparison of the vertical velocity profile for the lid driven cavity with $Re = 1000$

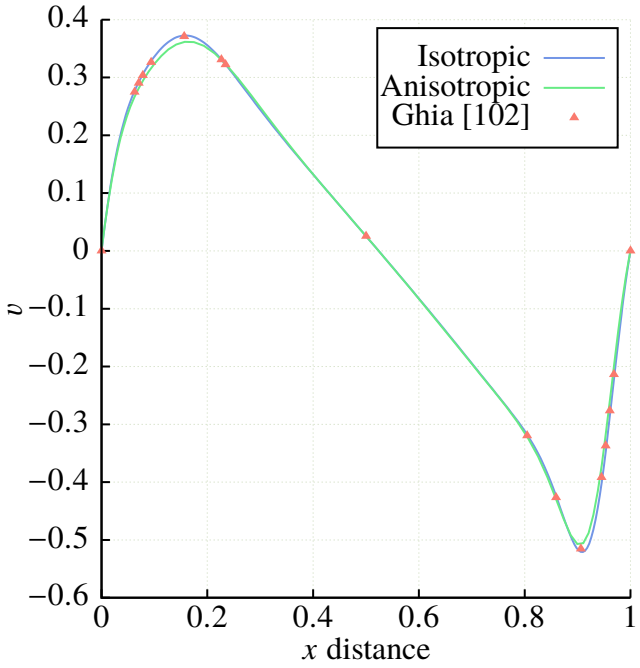


Figure 4.11 Comparison of the horizontal velocity profile for the lid driven cavity with $Re = 1000$

4.2.3 Mixed Convection

The mixed convection test case presented here is designed to simultaneously test both the convective terms and the heat transfer/buoyancy capability of the code. This problem is set up almost exactly the same as the forced convection case, presented in Section 4.2.2, except now the moving top boundary is also maintained at a constant non-dimensional temperature of one, making it an isothermal boundary in lieu of an adiabatic boundary. The Reynolds number is 1000, and since there is heat transfer, a Grashof number is required; for this case the Grashof number is 1×10^6. Furthermore, the top and bottom walls have the Neumann buoyant source term condition applied as described in Section 4.2.1. Once again, the geometry, initial, and boundary condition schematic is shown in Figure 4.12. The steady state solution for this mixed convection case is shown in

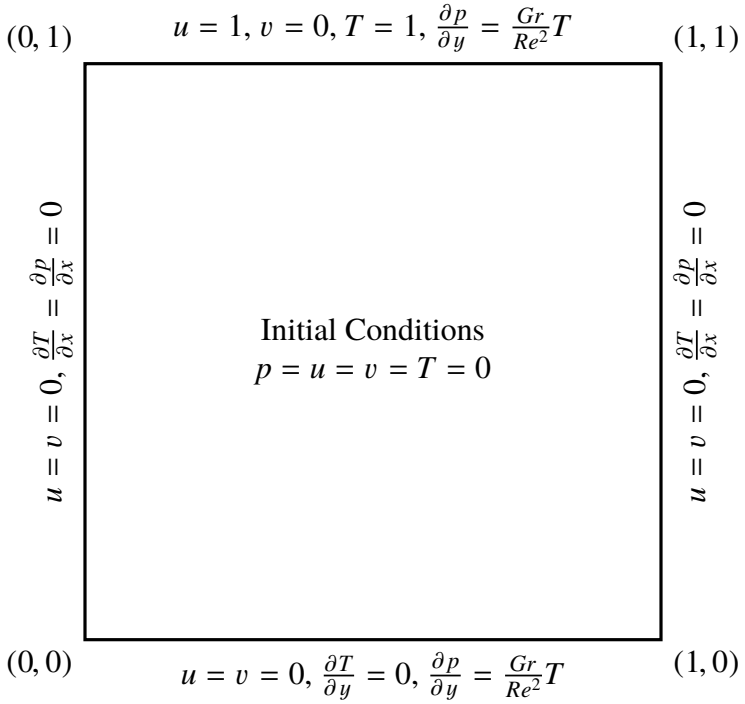


Figure 4.12 Mixed convection cavity geometry and boundary condition schematic

Figure 4.13; note the relatively smaller recirculation zone (compared to both the natural and forced convection cases presented above) towards the top right corner of the cavity. In this particular case, the moving top boundary causes the fluid to circulate, however, since this boundary is also heated, buoyancy effects restrict the recirculation zone from filling up the entire cavity. Once again, the dashed black lines represent the lines along which velocity profiles are extracted for comparison.

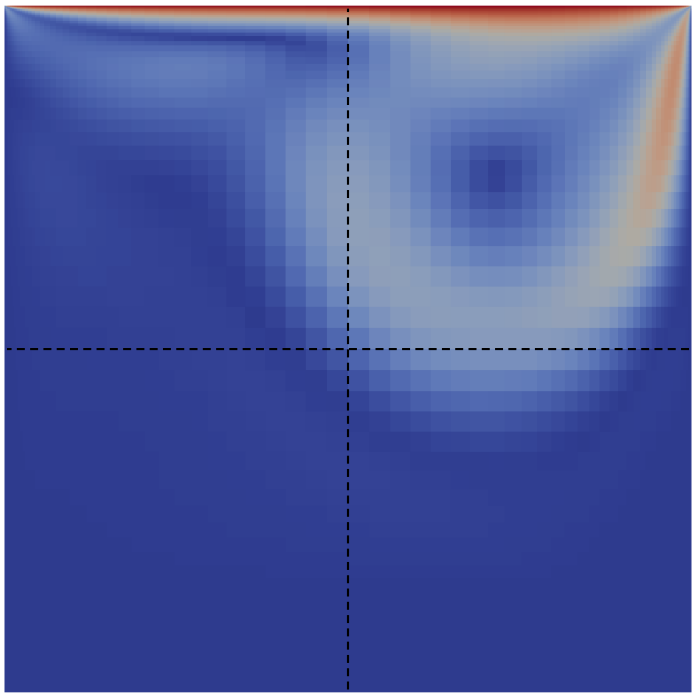


Figure 4.13 Steady state solution showing velocity magnitude in the mixed convection cavity case

Iwatsu [104] and Agrawal et al. [105] have studied mixed convection in a cavity extensively; the current work is compared to their results and is shown in Figures 4.14 and 4.15. While agreement with previous results by Iwatsu [104] and Agrawal et al. [105] is not as good as the

agreement seen in both the natural and forced cavity cases presented earlier in this chapter, similar levels of disagreement are observed in the results presented by Kress [88].

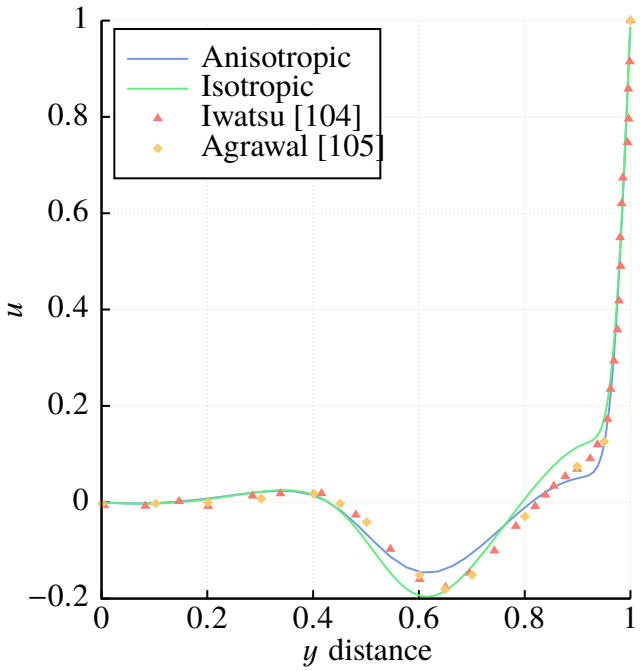


Figure 4.14 Comparison of the vertical velocity profile for mixed convection with $Re = 1000$

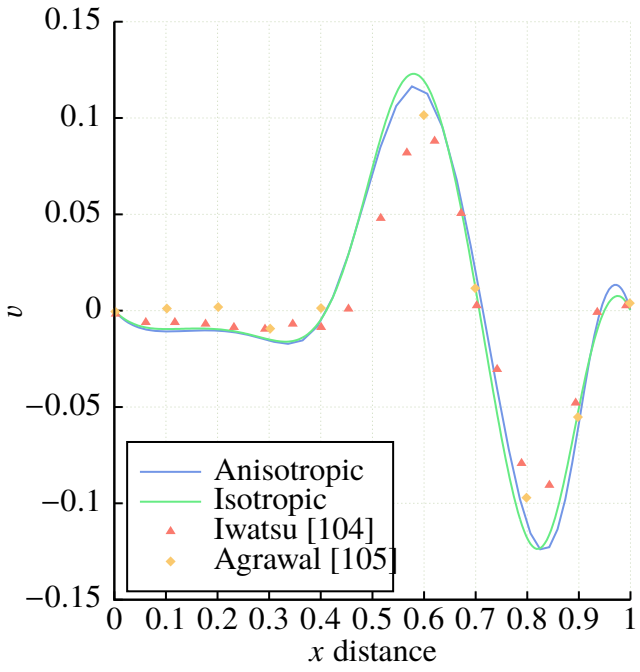


Figure 4.15 Comparison of the horizontal velocity profile for mixed convection with $Re = 1000$

4.3 Square Cylinder

This section presents laminar validation cases—both steady and unsteady—on a square cylinder. Although the square cylinder is not as ubiquitous as its circular counterpart, it is judged to be a very good test case given the Cartesian aligned nature of the research herein. It is recognized that validation of a numerical result is usually performed using information with a 'pedigree;' e.g., measurements with known and/or accepted uncertainties. However, it is difficult to identify laminar test cases—especially unsteady ones—similar to the unit cavity problems presented herein that are available on Cartesian aligned geometries. Consequently, the decision has been made to use computed results from a trusted computational methodology as a standard for comparison (*Tenasi*, see Hyams et al. [106] and Taylor et al. [107]).

The mesh used for these square cylinder validation cases is a structured UGRID mesh. The cylinder is a unit square centered in a square mesh dimensioned 100×100. The normal spacing is 1×10^{-3} units off the wall, with a wake refinement region stretching downstream of the cylinder 10 units. This grid was extruded four planes in the z direction such that there is an entire slice of elements interior to the mesh. The cylinder boundaries have the standard no-slip boundary condition enforced, while the x and y planes are far-field/flow-through boundaries enforced with a characteristic variable boundary condition. The z planes are symmetry boundary conditions.

4.3.1 Steady

For Reynolds numbers that are low enough, flows past circular and square cylinders are steady, and there exists a recirculation zone in the wake region of the blockage. The size of this recirculation zone will vary with the Reynolds number—the larger the Reynolds number, the larger the recirculation zone—until some critical Reynolds number is reached, and the flow transitions to unsteady flow. Figure 4.16 demonstrates the recirculation length, L_r, for a steady flow with $Re = 10$. The contours shown in this figure are lines of constant velocity magnitude and are present to make visualizing the recirculation zone easier.

Zdravkovich [108] shows that for a circular cylinder there exists an empirical relationship between recirculation length, L_r, and Reynolds number

$$\frac{L_r}{D} = 0.05Re : Re \in [4.4, 40] \tag{4.1}$$

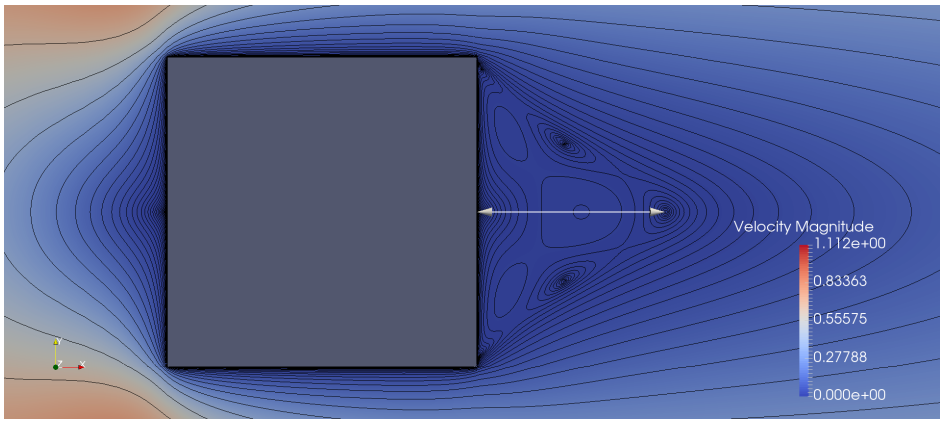


Figure 4.16 An example of steady flow ($Re = 10$) past a square cylinder; the recirculation length, L_r, is shown by the white line

Breuer et al. [109] demonstrated a linear relationship, shown in equation (4.2), between Reynolds number and recirculation length in laminar flow past a square cylinder with a blockage ratio of $\frac{1}{8}$, or 12.5%, where blockage is defined as the cross-sectional area of the cylinder perpendicular to the flow, compared to the cross-sectional area of the test section with no cylinder present.

$$\frac{L_r}{D} = -0.065 + 0.0554Re : Re \in [5, 60] \tag{4.2}$$

Sharma and Eswaran [110] also show a linear relationship for laminar flow past a square cylinder with a blockage ratio of 5%

$$\frac{L_r}{D} = 0.0672Re : Re \in [5, 40] \tag{4.3}$$

Sharma reports that this curve fit has a maximum deviation of 5%.

Various steady square cylinder cases were run, and recirculation lengths were compared against the results above. As seen in Figure 4.17, the results compare well with those of Sharma.

Note that, in this plot, the data corresponding to the results presented by Zdravkovich and Breuer et al. are simply to show a trend; they are not expected to match well with results obtained with *paros* as the simulation by Breuer et al. had a blockage ratio of 12.5% compared to the present computations with no blockage, and Zdravkovich's results were obtained on a circular cylinder.

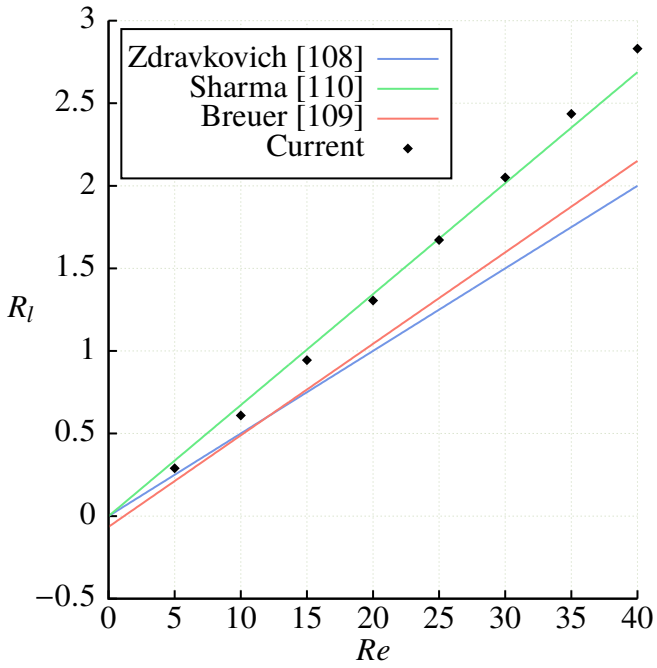


Figure 4.17 Comparison of recirculation length versus Re for the steady square cylinder

4.3.2 Unsteady

Within a certain range of Reynolds numbers, the laminar flow past a square cylinder transitions from steady to unsteady and vortex shedding occurs. Specific values of Reynolds numbers at which shedding begins vary significantly in the literature and depend on factors including blockage ratio, and inflow and outflow boundary placement. Sohankar et al. [111] report the critical

Reynolds number between 50 and 55 for a blockage ratio of 5%; Breuer et al. [109] cite a critical Reynolds number of approximately 60 for a blockage ratio of 12.5%, and Sharma et al. [110] report a critical Reynolds number of 50 with a blockage ratio of 5%. Note that Sohankar et al. [112] imply that if the wall boundaries are sufficiently far away from the cylinder, i.e., a blockage ratio less than 5%, the effect of these boundaries on the flow near the body is minimal. Figure 4.18 and Table 4.2 demonstrate the unsteady results as they compare against those of the University of Tennessee SimCenter at Chattanooga code *Tenasi* [106, 107]; the Strouhal number associated with vortex shedding is compared to assess the time accuracy of the simulation. The Strouhal number is defined as $St = \frac{fL}{U}$; f is the shedding frequency computed using forces exerted on the square cylinder by the steady-periodic flow, L is a characteristic length, and U is the free stream fluid velocity.

Table 4.2 Comparison of *St* versus *Re* data obtained using *paros* and *Tenasi*

		St	
		paros	*Tenasi*
Re	50	0.099	0.104
	60	0.111	0.114
	70	0.120	0.122
	80	0.127	0.128
	90	0.132	0.133
	100	0.137	0.137
	110	0.141	0.141
	120	0.144	0.144
	130	0.147	0.146
	140	0.149	0.148
	150	0.150	0.149
	160	0.151	0.150

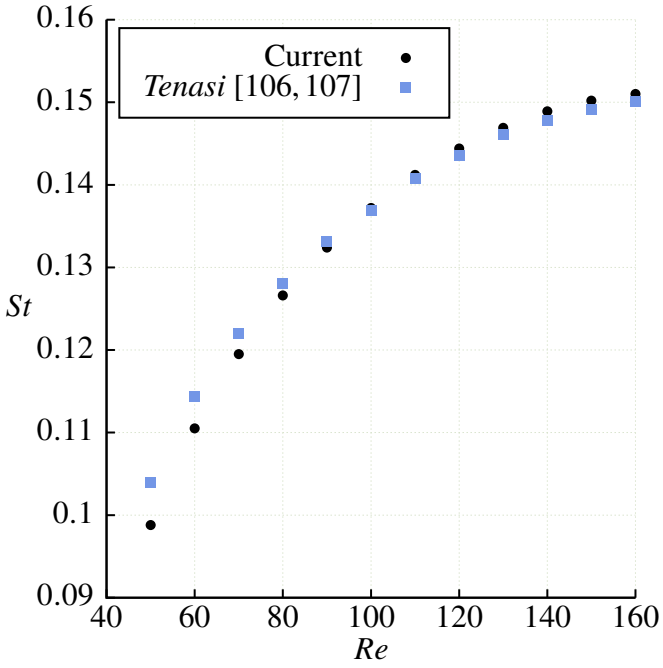


Figure 4.18 Comparison of Strouhal number St versus Re for the unsteady square cylinder

4.4 Turbulent Flow Over a Surface-Mounted Cube in a Tunnel

This final validation case examines the validity of the implemented Smagorinski LES turbulence model by comparing simulation results against the experiment of Martinuzzi and Tropea [113] as well as results obtained using *Tenasi* [106, 107]. Martinuzzi and Tropea explore the effects of various surface-mounted, prismatic, and fully three-dimensional obstacles in fully developed turbulent channel flow. The specific case used for this validation is that of a square cube of dimension $H = 1$ mounted in a channel with dimensions $l = 24H$, $w = 16H$, and $h = 2H$. The cube is mounted in the center of the channel $5H$ downstream of the inlet. This setup is chosen because it is a widely studied test case with ample experimental data, that exhibits fully three-dimensional, highly turbulent flow. Furthermore, the features of this flow are representative of those that might

occur around buildings in an urban environment. Since the Smagorinski turbulence model is likely insufficient to accurately model the type of turbulent flow exhibited by this test case, good agreement with experimental data is not expected. For this reason, *Tenasi* is also used to simulate this test case using the same Smagorinski turbulence model, boundary, and initial conditions, as well as an analogous grid in order to have baseline information for comparison against the current results.

The grid used for this simulation is completely structured; however two distinct meshes were built: one for *paros* and one for *Tenasi*. The version of the grid designed for *Tenasi* was built in the traditional manner, meaning that the grid exported from the grid generation software was used directly in *Tenasi* with no further refinement or modification. However, the grid for *paros* had to be designed for a target refinement level of two in order to keep the 'tree' count manageable. As such, the element dimensions of the original structured mesh designed for *Tenasi* were reduced by approximately a factor of four and the off-the-wall spacing increased by the same factor to create the skeleton mesh/connectivity used as the initial discretization for *paros*. The target normal spacing is 0.002 non-dimensional units, so the initial off-the-wall spacing for the skeleton mesh is 0.016. This design causes the final mesh used by *paros* to have a slightly less smooth variation of elements as they grow away from the boundaries, because every macroelement is uniformly refined twice to create the final mesh. However, it is anticipated that difference in the final meshes should have minimal effect on the accuracy of the results.

The Reynolds number, based on the height, H, of the cube, is 40,000. A parabolic velocity profile is imposed at the inflow boundary and the exit boundary condition is implemented such that the back/exit pressure is the reference/free-stream pressure, and the fluid is allowed to simply exit the boundary. All other boundaries are no-slip solid walls. This case is run spatially first order for

several hundred iterations; second order spatial accuracy is then activated, and the simulation run for a few hundred more iterations to obtain an initial state for unsteady time stepping. After the unsteady solution evolves several hundred iterations into a physically realistic state, time averaging begins. The period over which to time average the unsteady solution is chosen in order to mimic the approach taken by Hajjawi [114], who time averages over a non-dimensional time period of 50 units by setting $\Delta t = 0.05$ and averaging over 1000 iterations. Unfortunately, for this grid, *paros* proved to be unstable using time steps of this size. Instead, a time step of $\Delta t = 0.01$ is found to be stable and time averaging is performed over 5000 iterations. Experience suggests that dual time stepping [115] could alleviate this stability problem

The data to be extracted for comparison come from two profiles downstream of the cube; the first profile is extracted 0.08 units behind the center of the cube ($\frac{x}{H} = 0.08$), while the second is located 0.5 units ($\frac{x}{H} = 0.5$) in the wake. The streamwise velocity component from both *paros* and *Tenasi* are compared to those recorded by Martinuzzi and Tropea in Figures 4.19 and 4.20.

As seen in these profiles, neither simulation results compare well with experiment, although the overall trends are captured reasonably well. The lack of detailed agreement is due to several reasons, two of which are that neither the grid nor the turbulence model are sufficient to accurately resolve the turbulent physics inherent to this problem. Also, there are distinct differences between the time-averaged profiles produced by *paros* and *Tenasi*; there are numerous differences between the implementation specifics in these two codes that could play a role in the observed discrepancies. Notably, *Tenasi* is a node-centered solver while *paros* is cell-centered. Also, there are significant differences between the two codes regarding how the surface gradients are computed, and *Tenasi* has been validated much more thoroughly than the current implementation of *paros*. However, the

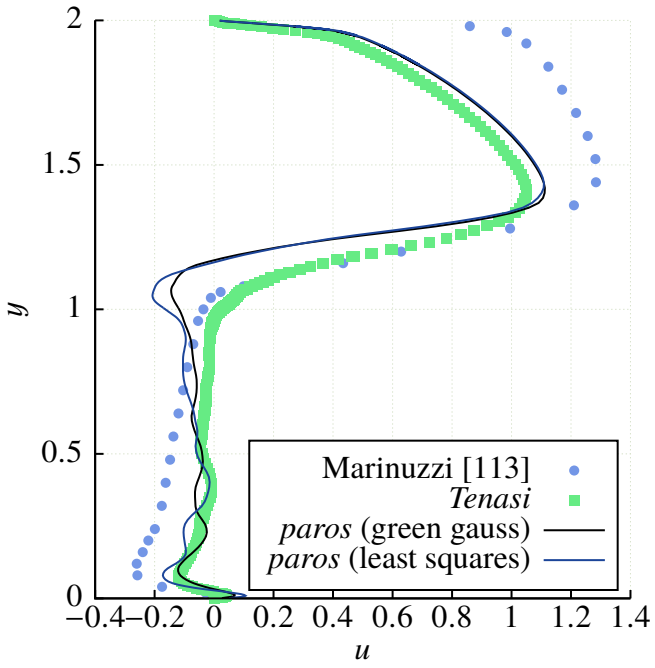


Figure 4.19 Time averaged u velocity profiles compared against experiment and *Tenasi* at $\frac{x}{H} = 0.08$

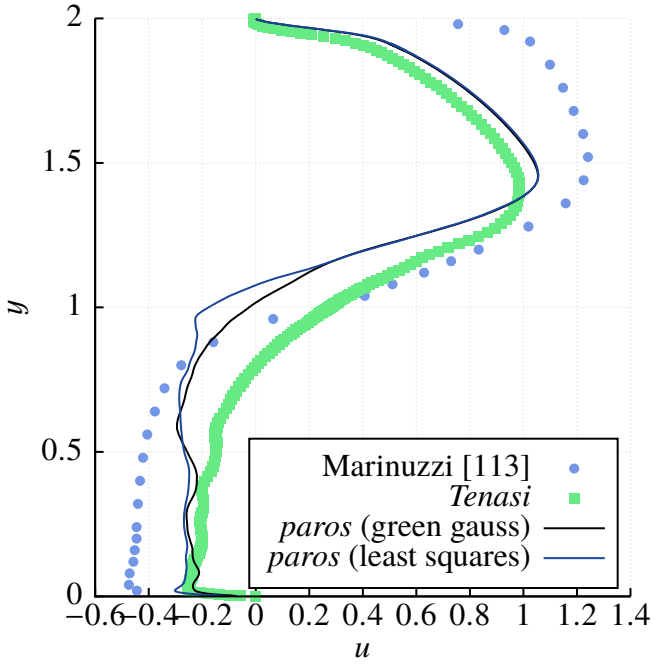


Figure 4.20 Time averaged u velocity profiles compared against experiment and *Tenasi* at $\frac{x}{H} = 0.5$

results are similar enough to inspire confidence in the implementation of the turbulence model. In fact, the profiles produced by *paros* in Figures 4.19 and 4.20 are closer to experiment than those of *Tenasi* in certain places. Recall that a goal of the present work is to provide fast simulations while preserving enough of the physics to be meaningful in a disaster situation, where simulation speedup is tantamount to being an effective mitigation tool. However, although the present tool can provide simulations considerably faster than existing methods, whether the degree of 'captured physics' shown in Figures 4.19 and 4.20 is accurate enough to satisfy this goal, is not known.

4.5 Large Scale Urban Simulation

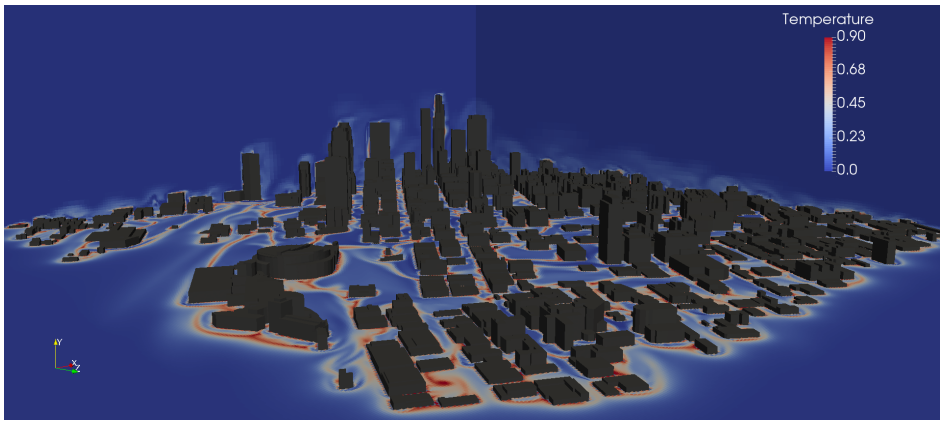
This section will demonstrate the ability of the application as it is applied to a large urban geometry. Multiple cases will be run, compared, and summarized in this section. These cases will be unsteady, with and without buoyancy. In addition, cases exercising the contaminant transport functionality will be shown. The geometry used for these test cases is the same one used for the parallel grid generation speedup analysis previously shown in Section 2.3 and contains 927 buildings and 4.25 million facets. Refer to Figure 2.10 which shows this geometry.

4.5.1 Buoyant Versus Non-Buoyant Cases

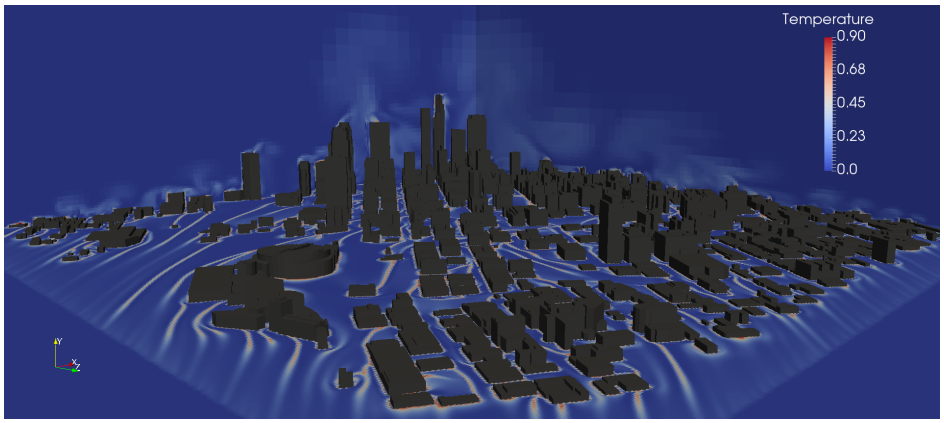
The results in this section will demonstrate the differences in the unsteady solutions obtained with and without heat transfer and buoyancy. Both cases are run with a Reynolds number of 137,014, which corresponds to a free stream velocity of $2\,\mathrm{m\,s^{-1}}$ based on a reference length of 1 m, reference density of $1.22217\,\mathrm{kg\,m^{-3}}$, and a reference viscosity of $1.784\times10^{-5}\,\mathrm{kg\,m^{-1}\,s^{-1}}$. The incoming flow is from the south-west: 46° south of due west, where 'west' corresponds to the negative x-direction. The simulation is run in the following manner: 1000 steady first order steps are

taken, followed by 500 steady second order steps in order to get a developed flow field as an initial condition for the time-accurate simulation, which begins after the prescribed number of steady second order steps have completed. A non-dimensional time step of 7.5 is taken which corresponds to a physical time step of 2.5 seconds per iteration. In the cases where buoyancy is considered, a Grashof number of 1.12×10^9 is used; this corresponds to an ambient temperature of 60 degrees Fahrenheit (15.6 °C), an isothermal wall temperature of 90 degrees Fahrenheit (32.2 °C), and a volumetric thermal expansion coefficient of $1.47 \times 10^{-3}\,\mathrm{K}^{-1}$.

The following figures demonstrate the effectiveness of the buoyant source terms and boundary conditions in the context of this urban simulation. The temperature field shown in Figure 4.21 clearly shows the heat rising in the buoyant simulation, whereas the non-buoyant case shows a higher heat concentration near the ground because the lack of buoyancy does not allow the heat to rise. Figure 4.22 shows how, at this relatively low speed flow and high temperature difference, effects of buoyancy are even visible in the velocity field.

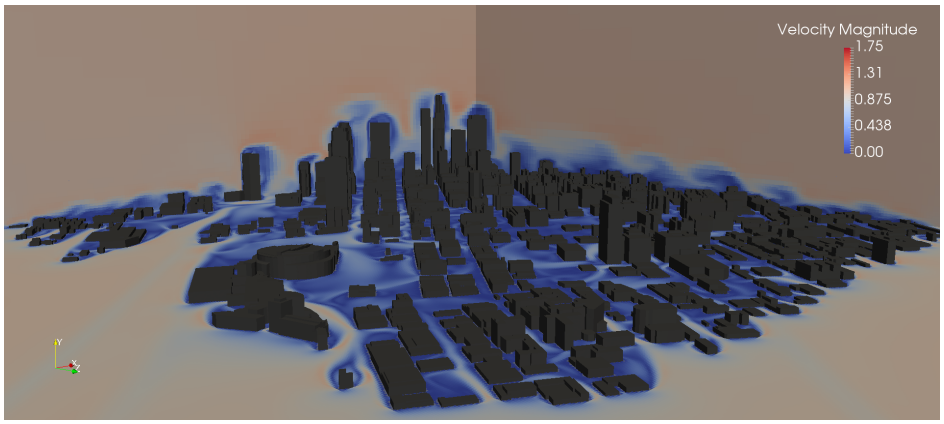


(a) Non-Buoyant Simulation

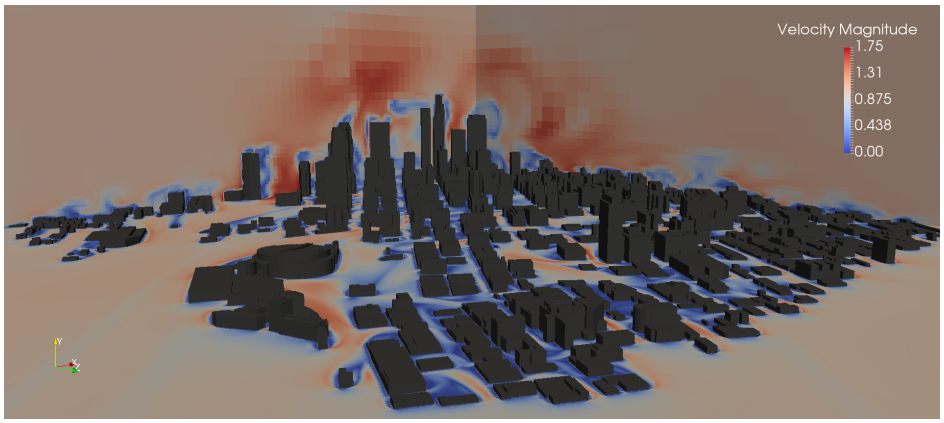


(b) Buoyant Simulation

Figure 4.21 Temperature distribution with and without buoyancy



(a) Non-Buoyant Simulation



(b) Buoyant Simulation

Figure 4.22 Velocity field with and without buoyancy

4.5.2 Buoyant Versus Non-Buoyant Cases Including Contaminant Transport

This section presents a case similar to one given by Nichols et al. [5]. After 500 unsteady steps, or 1250 seconds in real time, a constant release of a neutrally buoyant contaminant is initiated, and the contaminant convects downstream with the prevailing wind which originates from the south-west and propagates toward the north-east. The release point, in the south-west portion of the cityscape, is shown by the red box in the inset of Figure 4.23. This box accurately represents the size of the release location in this simulation; it is 9 m by 9 m wide and 6 m high.

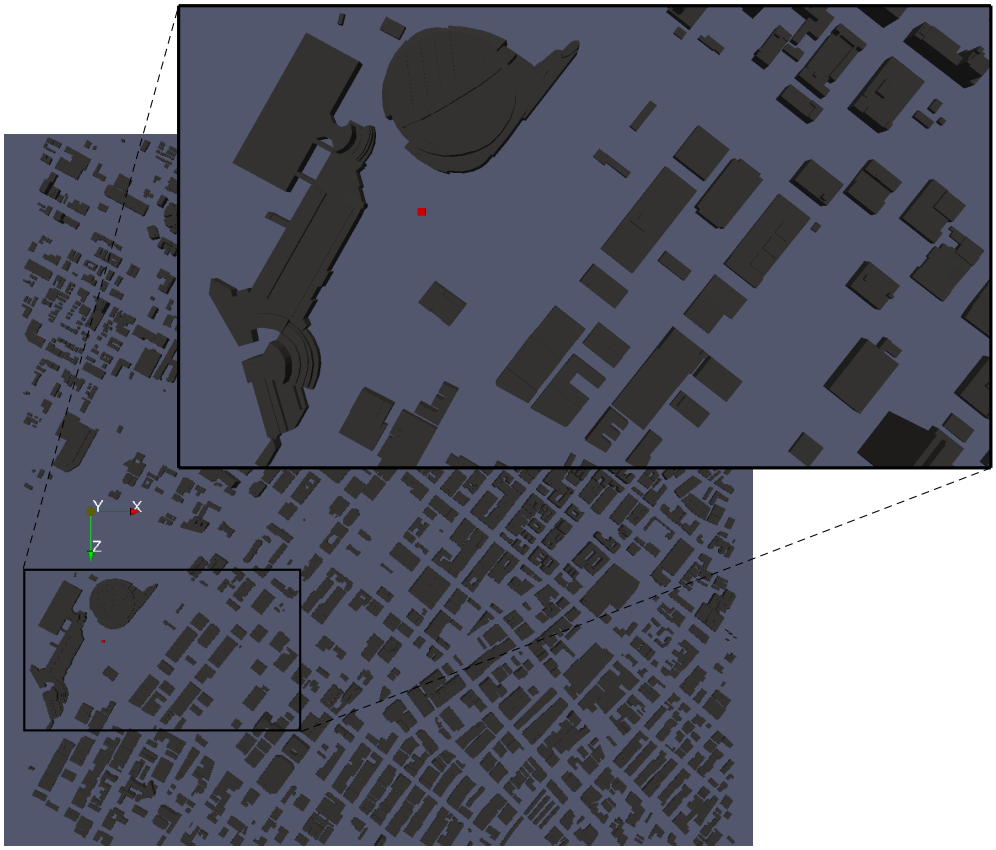
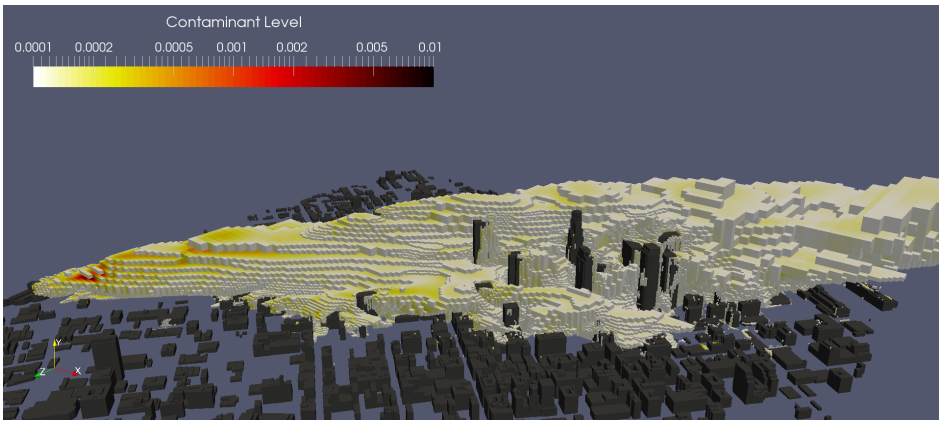
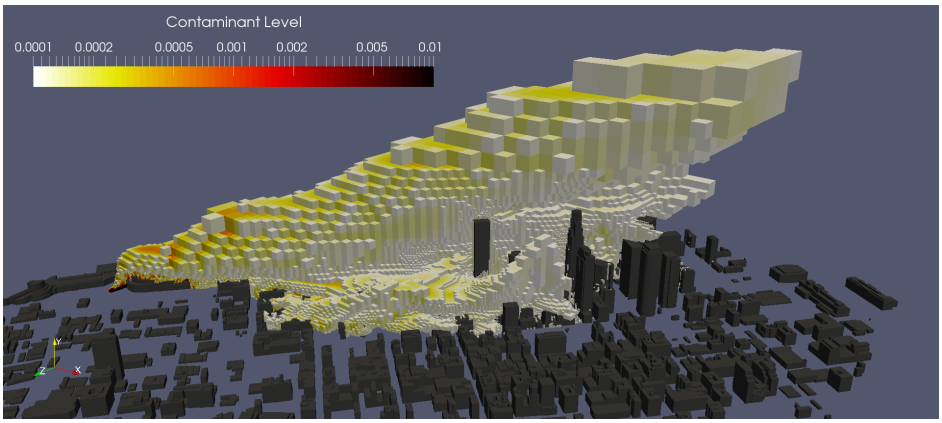


Figure 4.23 Contaminant release location

Figures 4.24 and 4.25 demonstrate how the plume evolves in both simulations after 3500 unsteady time steps (8750 seconds, real time). The plume is represented in these figures by isolating all elements in the solution that contain contaminant levels in the non-dimensional range from 0.0001 to 1.0; the plume is colored using a log scale to accentuate the contaminant levels. Notice how in Figures 4.24b and 4.25b the buoyant effects have clearly caused the plume to rise more vertically, as well as spread outward less into the lower alleyways as compared to Figures 4.24a and 4.25a. Although this is the expected behavior from a qualitative perspective, a dearth of validation-quality measurements makes it very difficult to make quantitative speculations about specifics of the contamination spread (e.g., rates of horizontal and vertical spreading). Figure 4.26 demonstrates contaminant levels of the same simulation at various horizontal slices through the urban geometry. One indication that the buoyancy terms are having the desired effect is that the contaminant levels at street level are much lower for the buoyant case than the levels in the case without heat transfer, especially well downstream of the release point. The effect of buoyancy is more clearly viewed in the vertical slices, shown in Figure 4.28, where the contaminant is obviously convected upwards by buoyantly driven updrafts. Figure 4.27 shows the location of each of the five stations. Station one is located 25 m downstream of the release location in the x direction, while stations two, three, four, and five are located 425 m, 825 m, 1425 m and 1925 m respectively downstream of the release location. Note that even though the contaminant itself is neutrally buoyant, it is being convected by flow that is experiencing buoyant effects because of a temperature gradient present in the simulation.

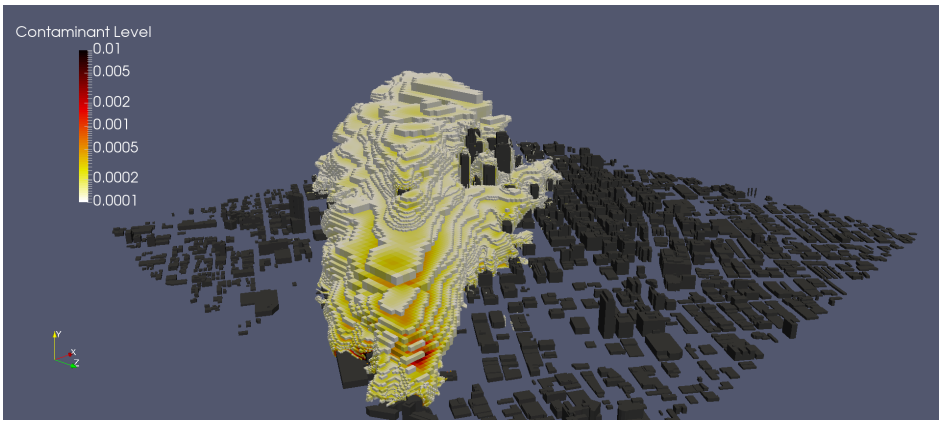


(a) Non-Buoyant Simulation

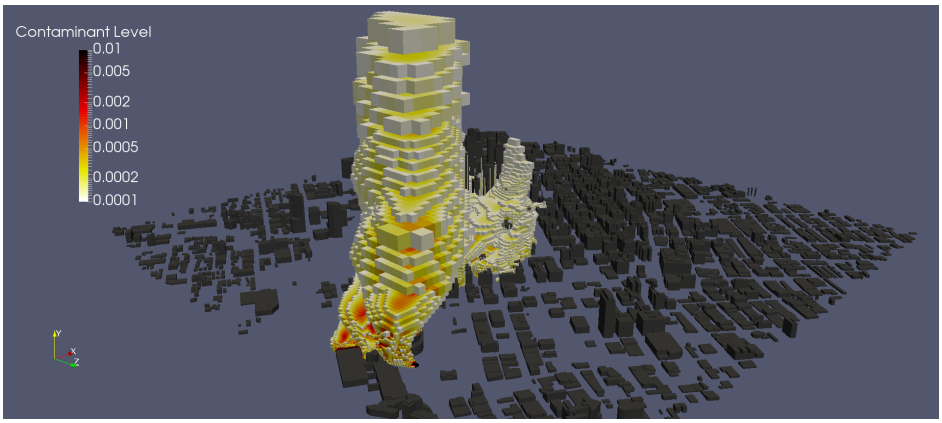


(b) Buoyant Simulation

Figure 4.24 A side view of two unsteady solutions demonstrating contaminant plume propagation (2.43 hours after release, real time)

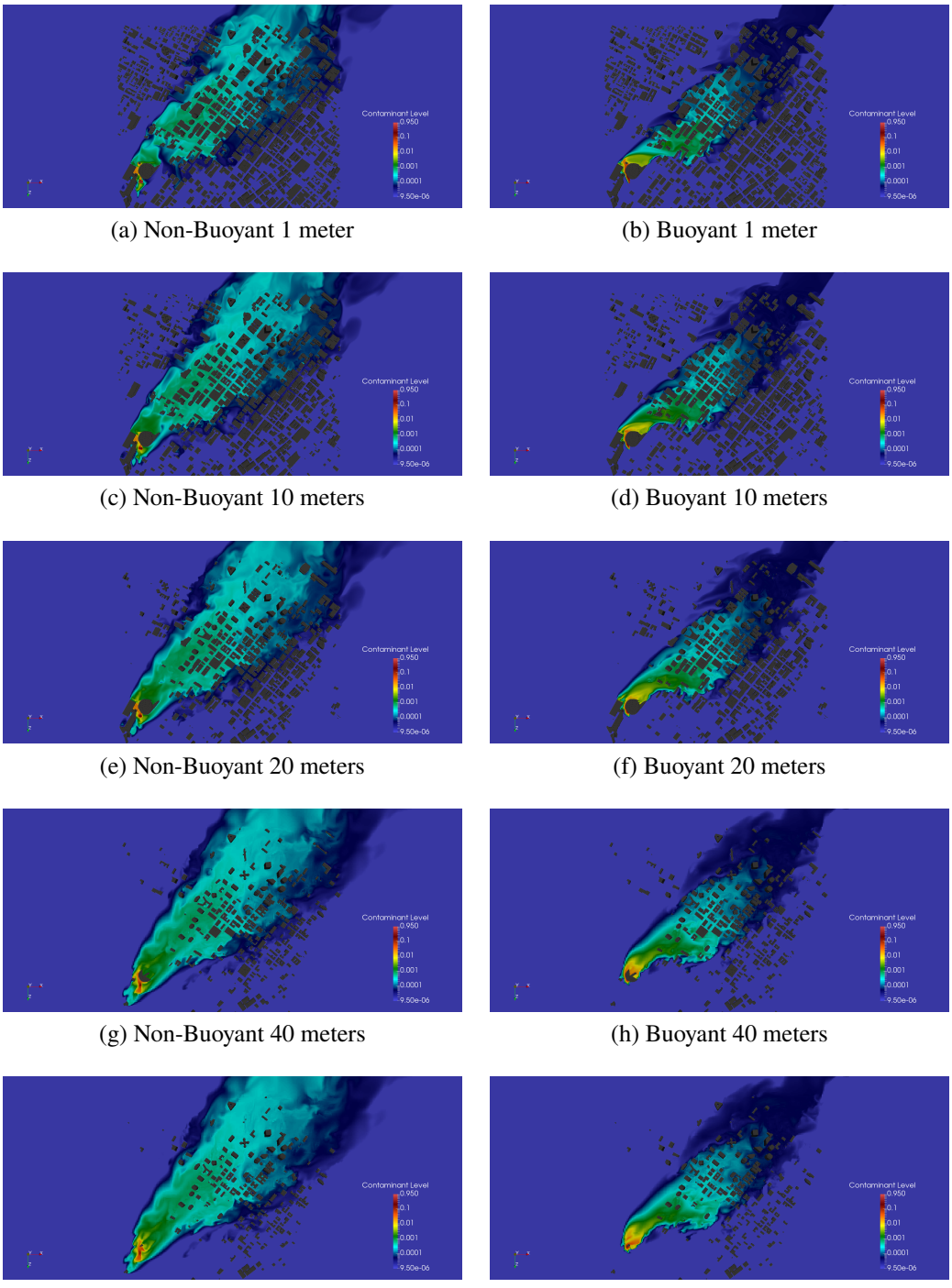


(a) Non-Buoyant Simulation



(b) Buoyant Simulation

Figure 4.25 A view from behind of two unsteady solutions demonstrating contaminant plume propagation (2.43 hours after release, real time)



(a) Non-Buoyant 1 meter

(b) Buoyant 1 meter

(c) Non-Buoyant 10 meters

(d) Buoyant 10 meters

(e) Non-Buoyant 20 meters

(f) Buoyant 20 meters

(g) Non-Buoyant 40 meters

(h) Buoyant 40 meters

(i) Non-Buoyant 50 meters

(j) Buoyant 50 meters

Figure 4.26 Two unsteady solutions showing contaminant concentrations at various elevations (2.43 hours after release, real time)

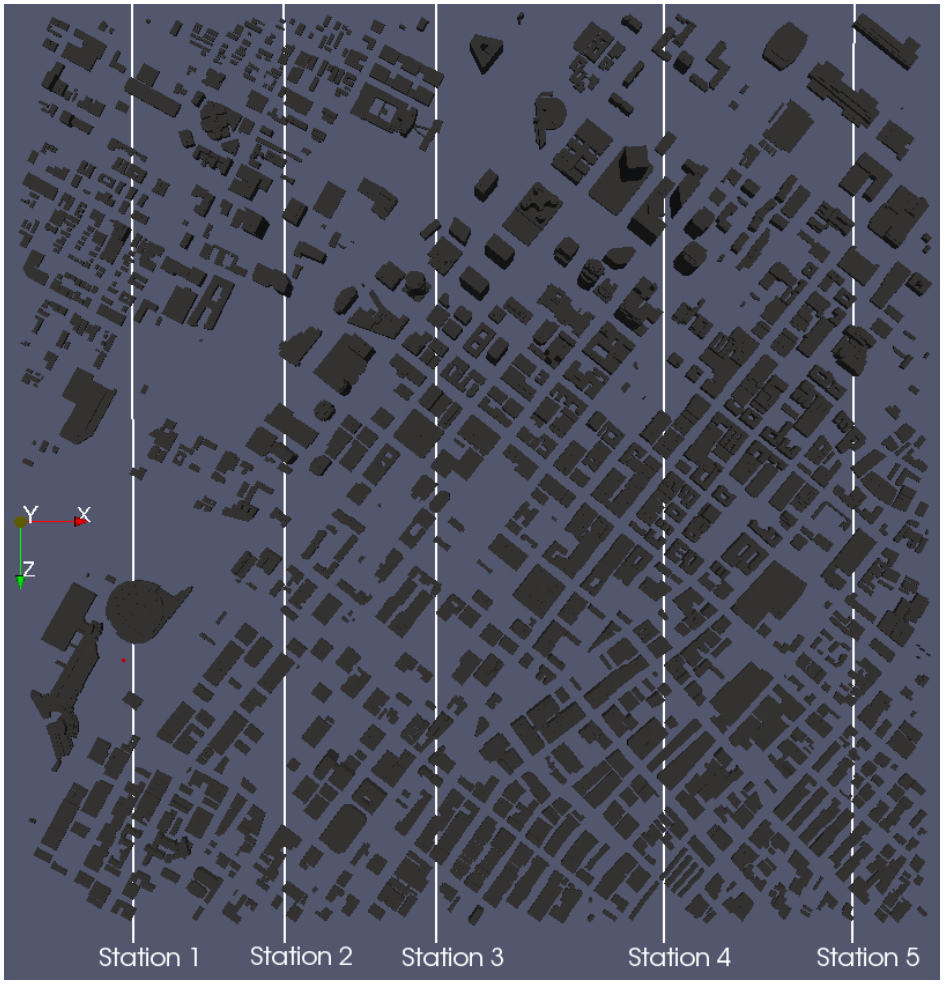
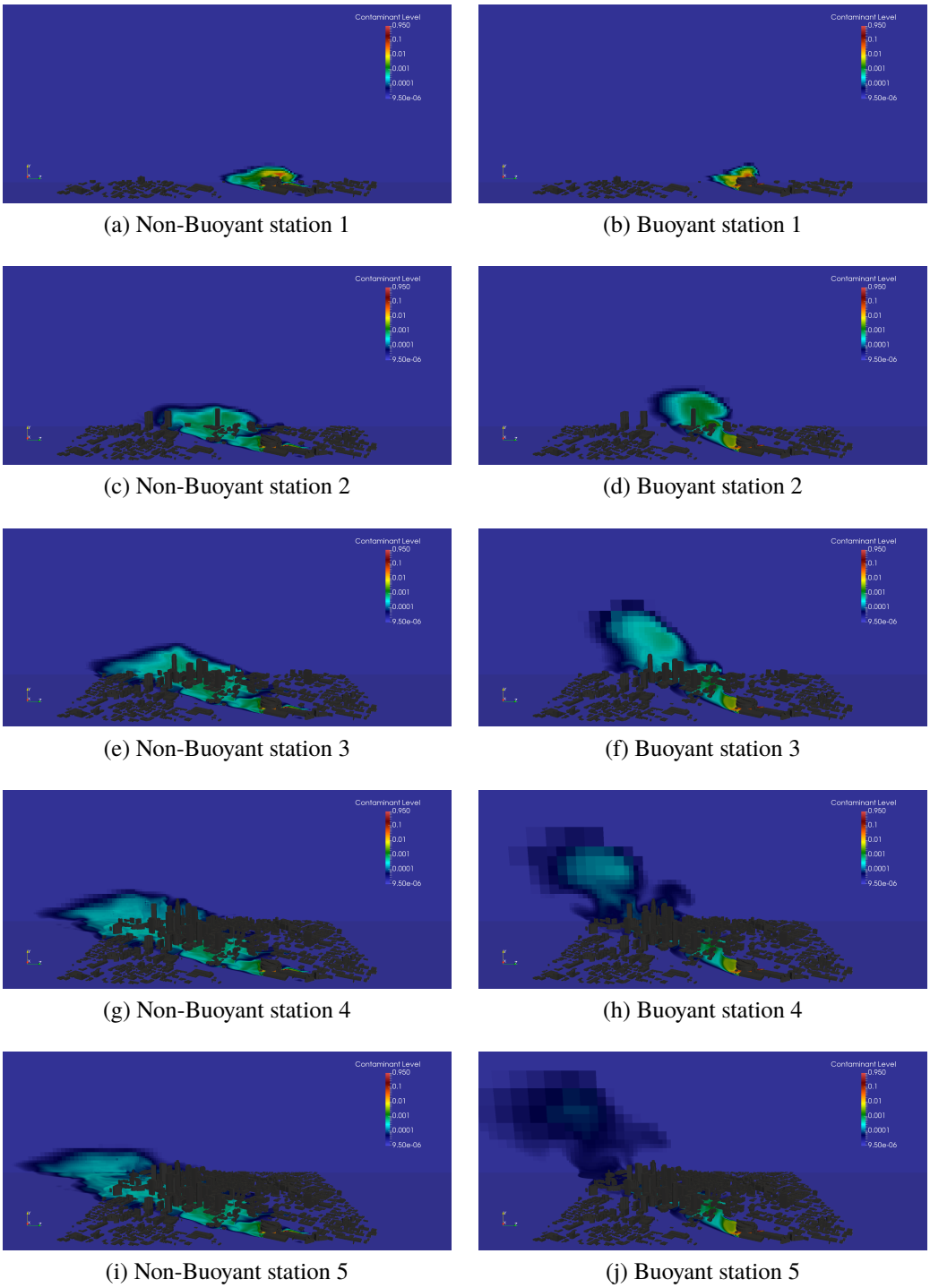


Figure 4.27 Station location schematic



(a) Non-Buoyant station 1

(b) Buoyant station 1

(c) Non-Buoyant station 2

(d) Buoyant station 2

(e) Non-Buoyant station 3

(f) Buoyant station 3

(g) Non-Buoyant station 4

(h) Buoyant station 4

(i) Non-Buoyant station 5

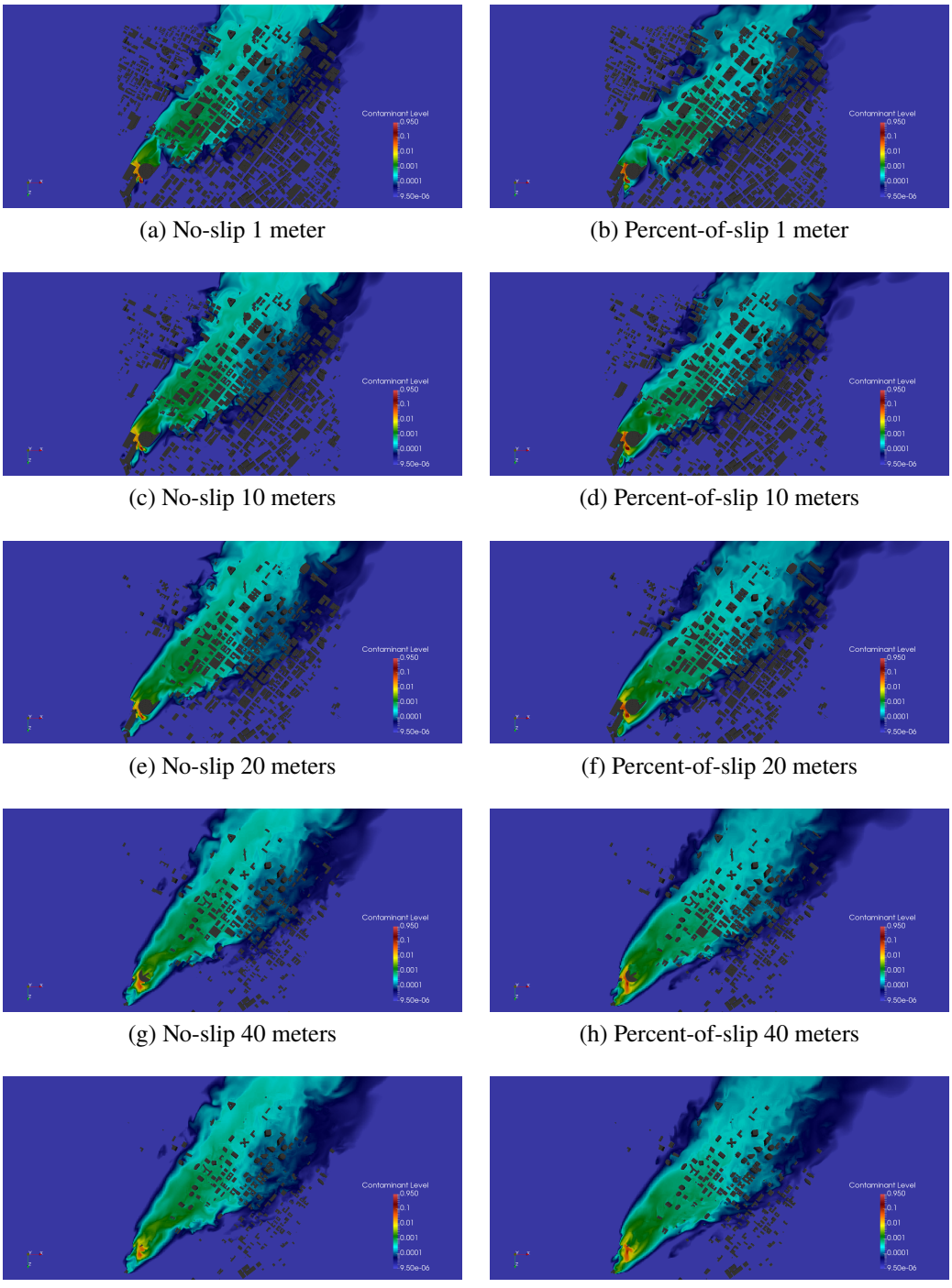
(j) Buoyant station 5

Figure 4.28 Two unsteady solutions showing contaminant concentrations at various stations downstream of the release point (2.43 hours after release, real time)

4.5.3 The Effect of the Percent-of-Slip Boundary Condition on Urban Contaminant Transport

This section demonstrates the effect of the percent-of-slip boundary condition on contaminant transport in an urban setting. The following figures show the results of different simulations, each run with and without the percent-of-slip boundary condition. For all cases a 90% slip condition has been applied, which means that all solid wall boundaries are treated as inviscid walls. However, only 90% of the resulting tangential flow velocity components are used as a boundary condition; see Section 3.5.6.5 for more detail. The results in Figure 4.29 are obtained at a Reynolds number of 2.06×10^5 (for a one meter characteristic length), which corresponds to a free stream flow of $3\,\mathrm{m\,s^{-1}}$; all other parameters are the same as were presented in the beginning of this chapter. The results in Figure 4.39 are the same in every way except that the free stream flow is $5\,\mathrm{m\,s^{-1}}$ which corresponds to a Reynolds number of 3.4×10^5. Once again, after 500 unsteady steps, a constant release of a neutrally buoyant contaminant is released into the flow field. The contaminant levels are presented using a non-dimensional log scale ranging from 9.5×10^{-6} to 9.5×10^{-1}. The simulation using the percent-of-slip boundary condition shows, at least qualitatively, improved mixing and convection of the contaminant throughout the urban environment, especially at the lower elevations. Once again, without measurements or other corroborative information, it is not possible to offer a quantitative assessment about the accuracy of these simulations.

In order to better quantify the effect of the percent-of-slip boundary condition on the contaminate spreading, four different profile locations are set up and contaminate levels are extracted from them for both cases presented in this section. These profiles extend perpendicular to the flow direction and into street canyons. Figure 4.30 shows where the profiles are extracted from within the



(a) No-slip 1 meter

(b) Percent-of-slip 1 meter

(c) No-slip 10 meters

(d) Percent-of-slip 10 meters

(e) No-slip 20 meters

(f) Percent-of-slip 20 meters

(g) No-slip 40 meters

(h) Percent-of-slip 40 meters

(i) No-slip 50 meters

(j) Percent-of-slip 50 meters

Figure 4.29 Comparison of no-slip versus percent-of-slip boundary conditions in an urban setting with a free stream velocity of 3 m s^{-1}

urban environment; the solution data shown in this figure represents an instantaneous snapshot of a contaminant release and is only present to help justify the placement of the extraction lines. Note that data are extracted from these profiles both at ground level (0 m), and at an elevation of 5 m. As seen in some of the plots, especially Figures 4.31, 4.35, 4.40 and 4.44, the percent-of-slip condition does improve the lateral spreading of the contaminate into the street canyons. However, as seen in Figures 4.32, 4.36, 4.41 and 4.46, it does not always make a significant difference. These profiles are extracted from time-accurate solutions instantaneously, i.e., from a single moment in time, and, as such, do not necessarily present an accurate assessment of the effect of this boundary condition on the contaminate spreading. A better way to quantify the effect of this boundary condition would be to extract the profiles from time-averaged solutions. Unfortunately, a lack of time prevented time-averaged profiles from being presented in this document.

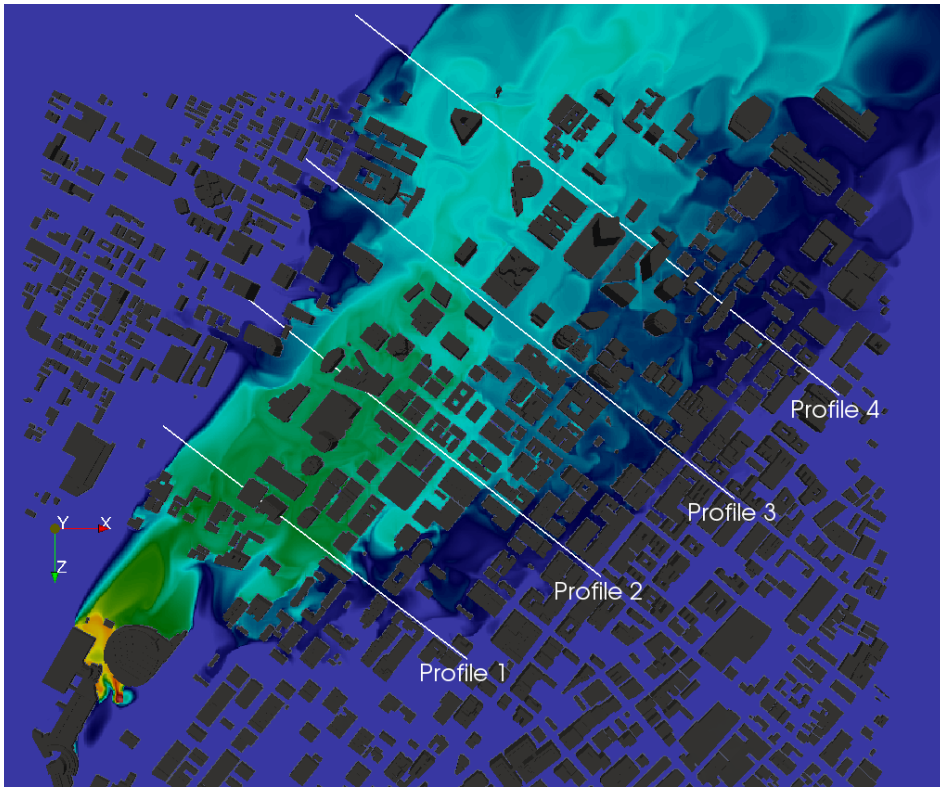


Figure 4.30 Profiles along which contaminate levels are plotted

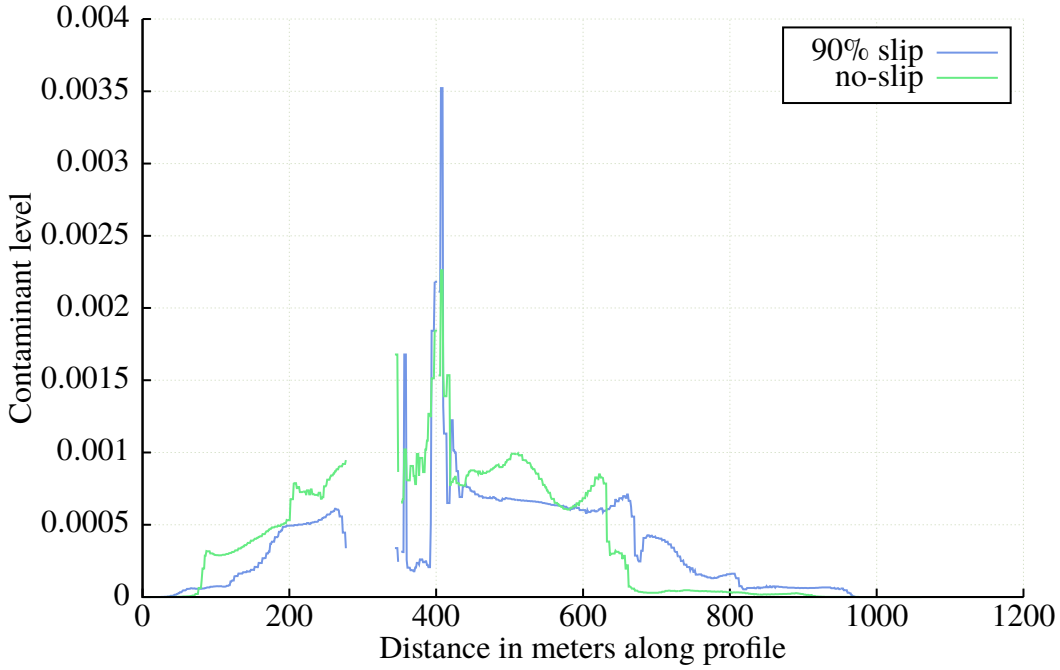


Figure 4.31 Urban contaminate levels for profile one at ground level for $Re = 205522$

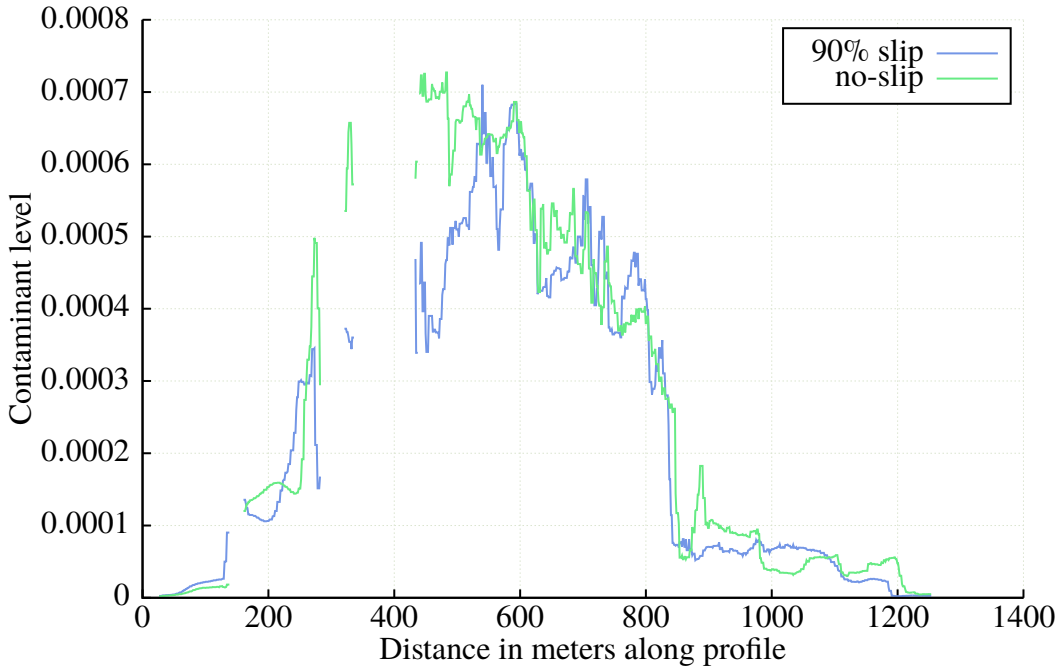


Figure 4.32 Urban contaminate levels for profile two at ground level for $Re = 205522$

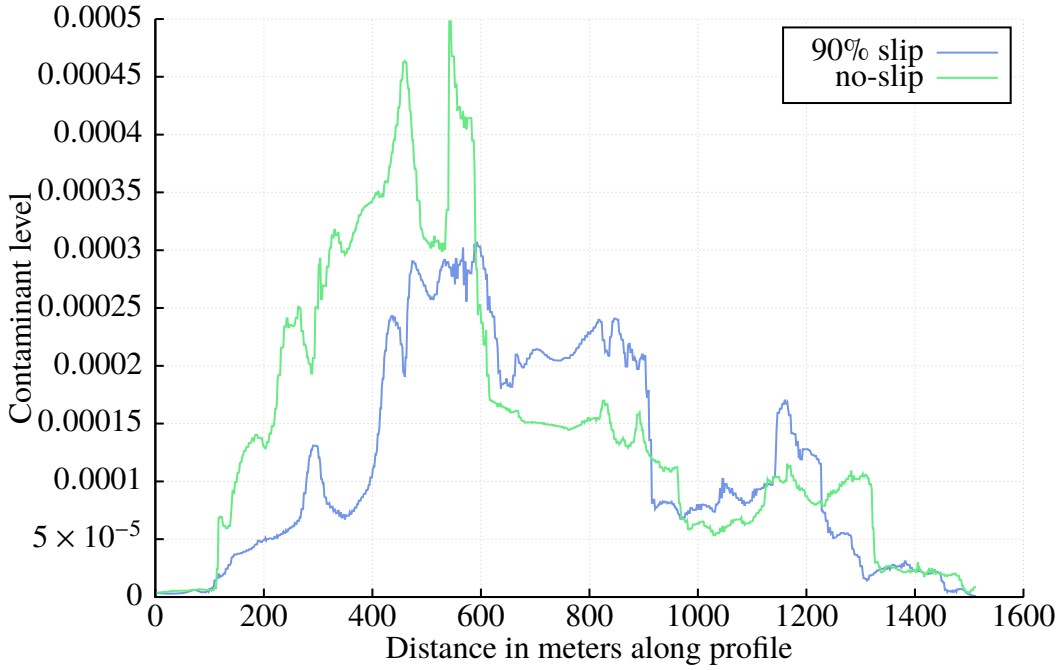


Figure 4.33 Urban contaminate levels for profile three at ground level for $Re = 205522$

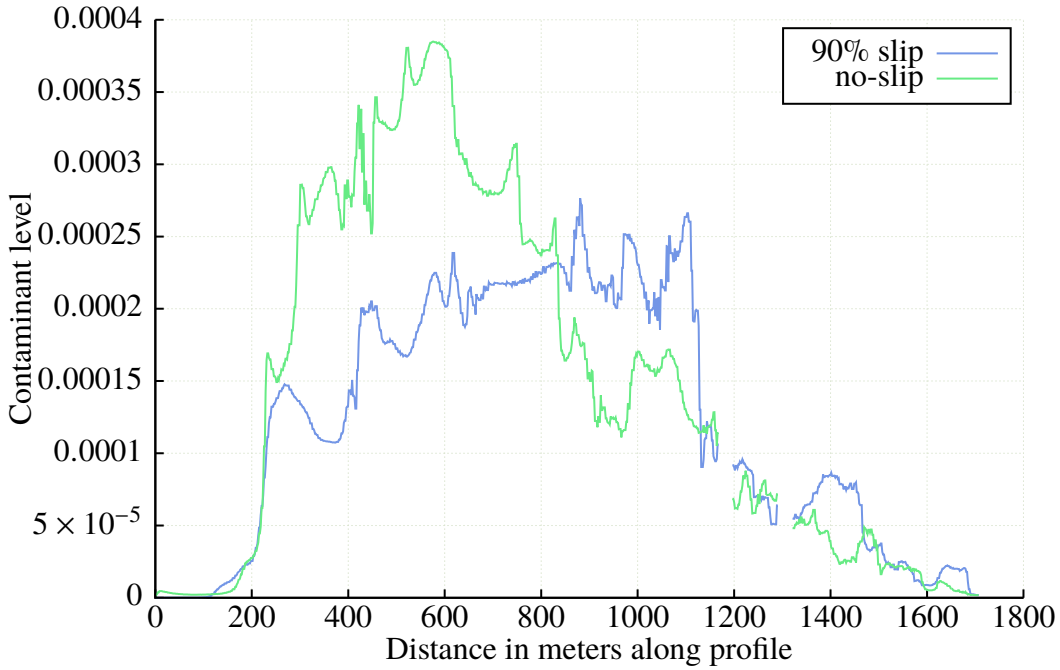


Figure 4.34 Urban contaminate levels for profile four at ground level for $Re = 205522$

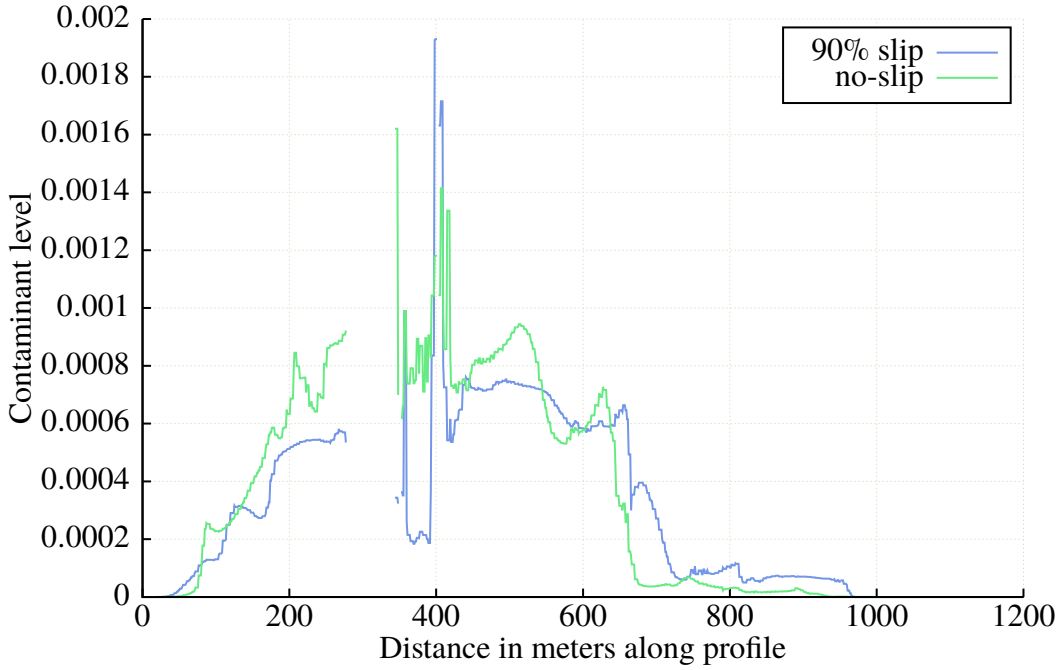


Figure 4.35 Urban contaminate levels for profile one at an altidude of five meters for $Re = 205522$

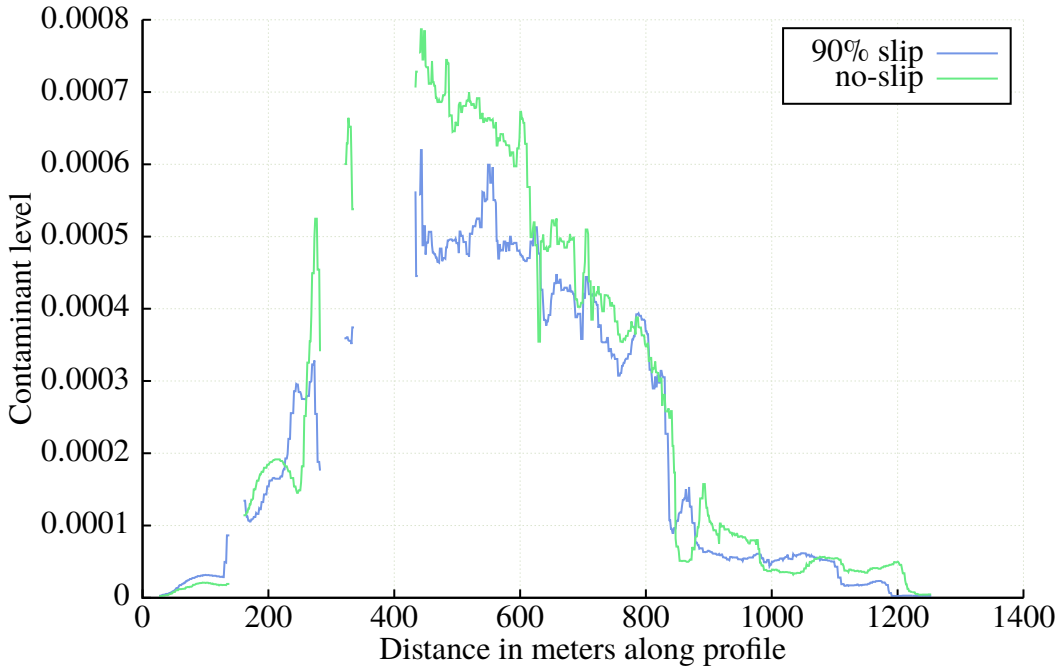


Figure 4.36 Urban contaminate levels for profile two at an altidude of five meters for $Re = 205522$

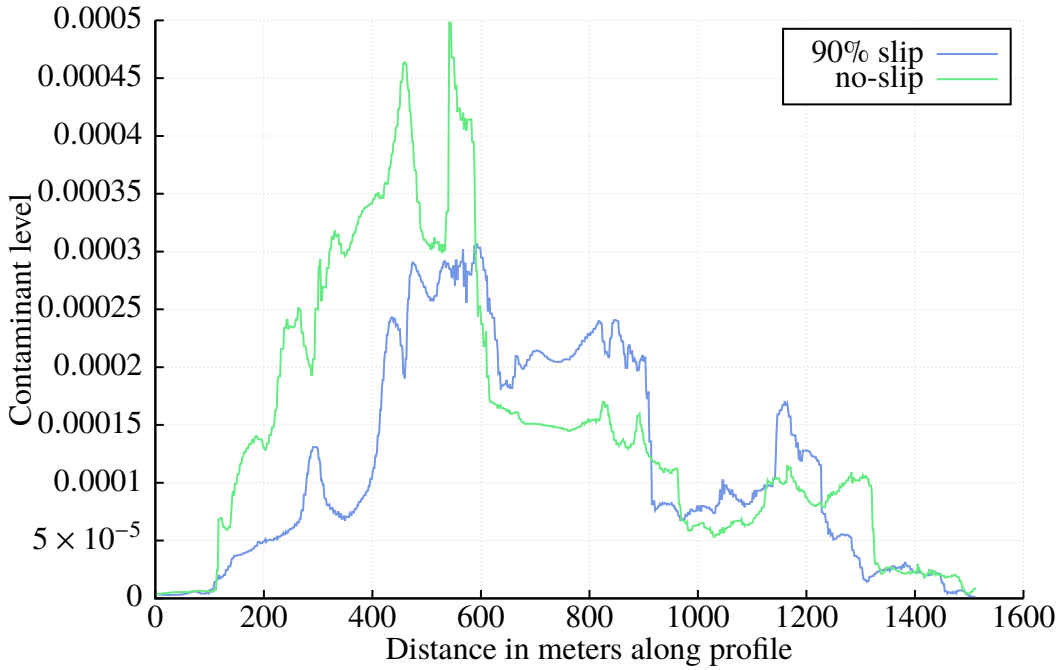


Figure 4.37 Urban contaminate levels for profile three at an altidude of five meters for Re = 205522

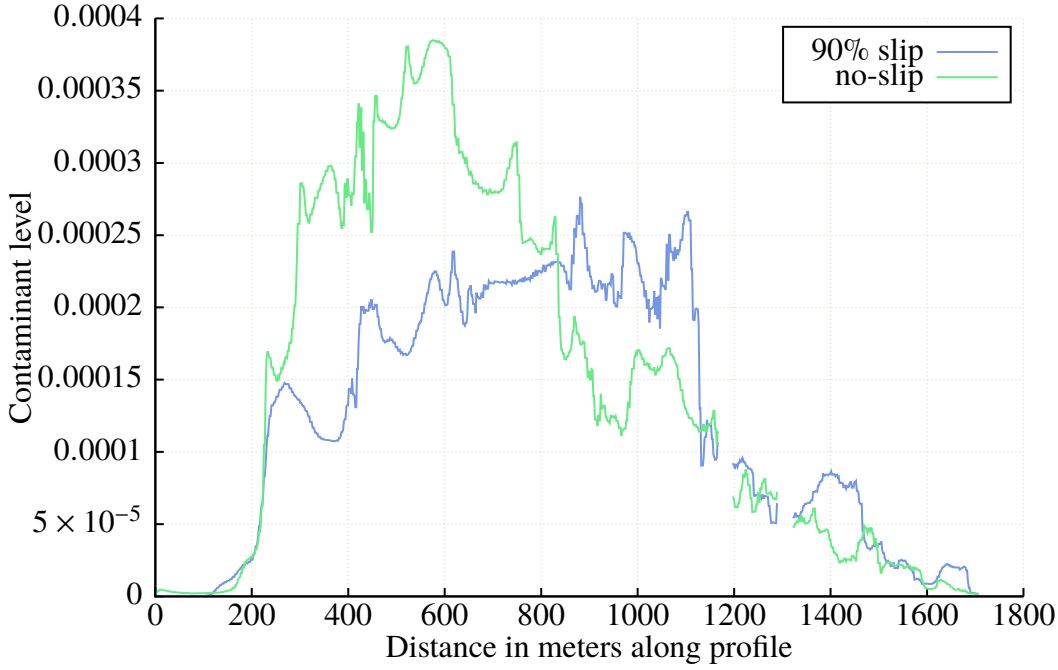
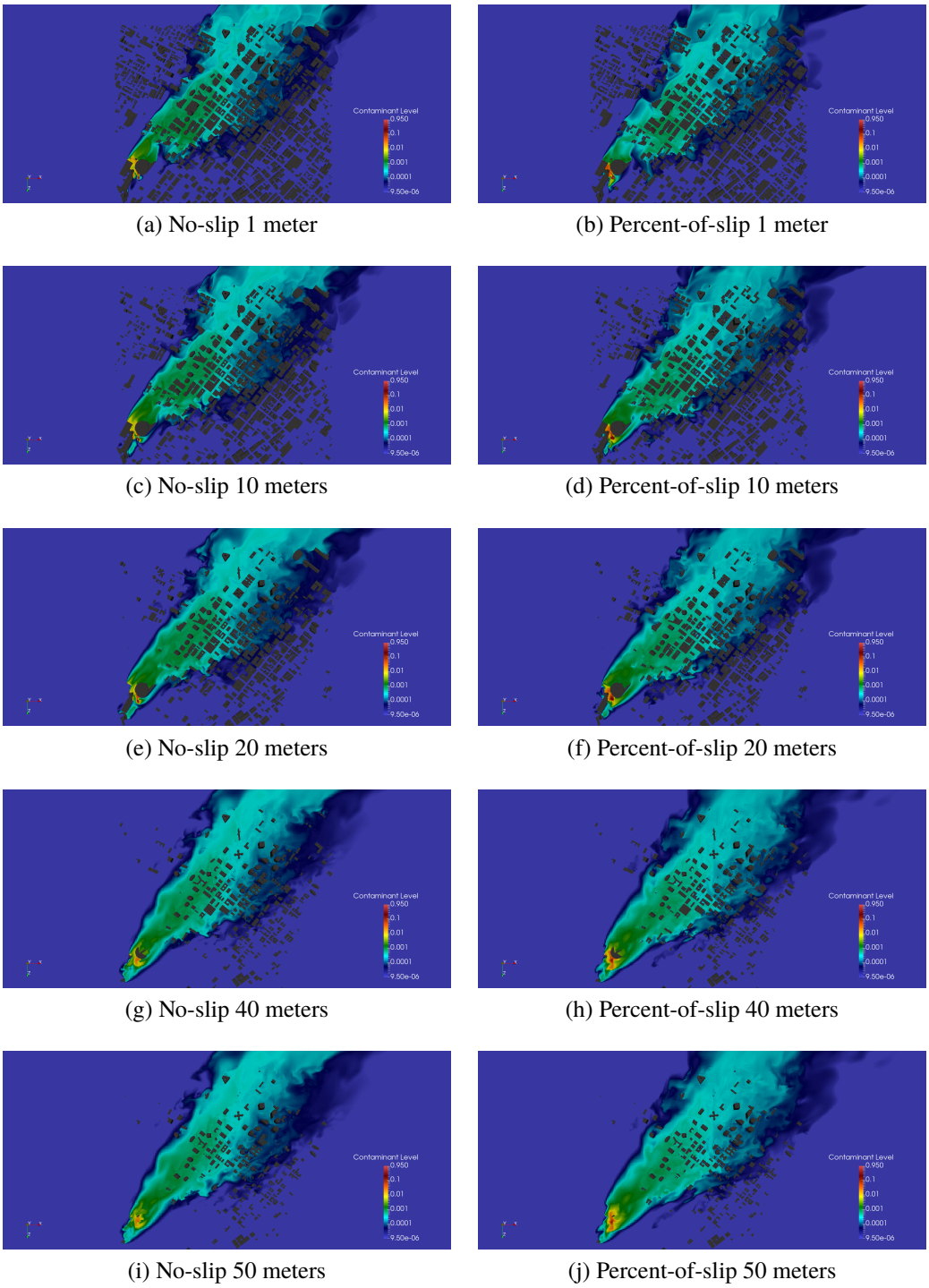


Figure 4.38 Urban contaminate levels for profile four at an altidude of five meters for Re = 205522



(a) No-slip 1 meter

(b) Percent-of-slip 1 meter

(c) No-slip 10 meters

(d) Percent-of-slip 10 meters

(e) No-slip 20 meters

(f) Percent-of-slip 20 meters

(g) No-slip 40 meters

(h) Percent-of-slip 40 meters

(i) No-slip 50 meters

(j) Percent-of-slip 50 meters

Figure 4.39 Comparison of no-slip versus percent-of-slip boundary conditions in an urban setting with a free stream velocity of 5 m s^{-1}

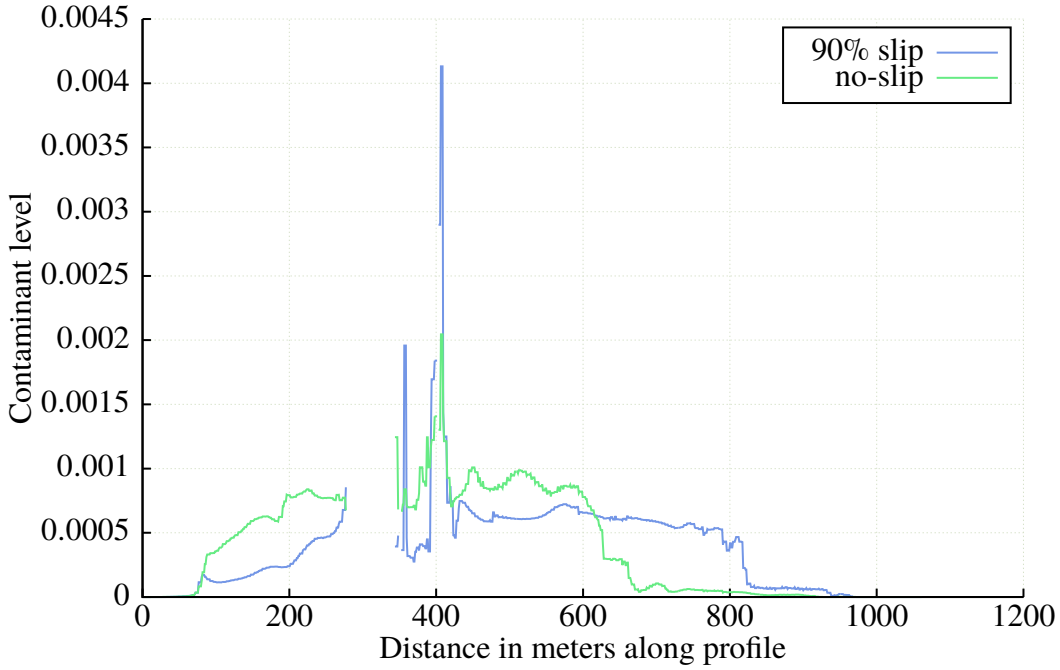


Figure 4.40 Urban contaminate levels for profile one at ground level for $Re = 342189$

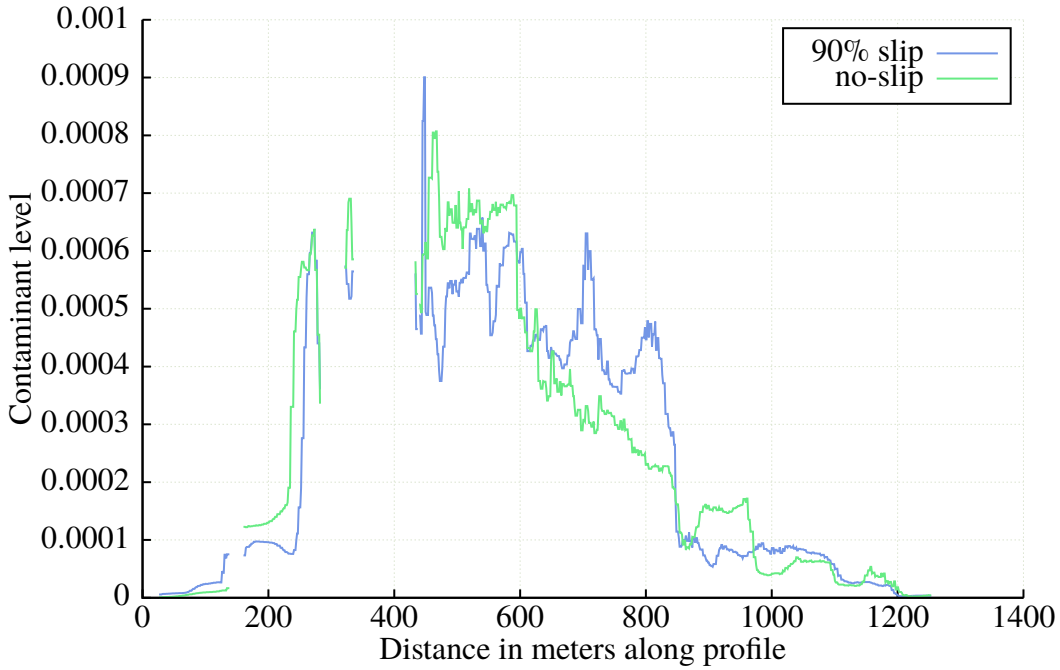


Figure 4.41 Urban contaminate levels for profile two at ground level for $Re = 342189$

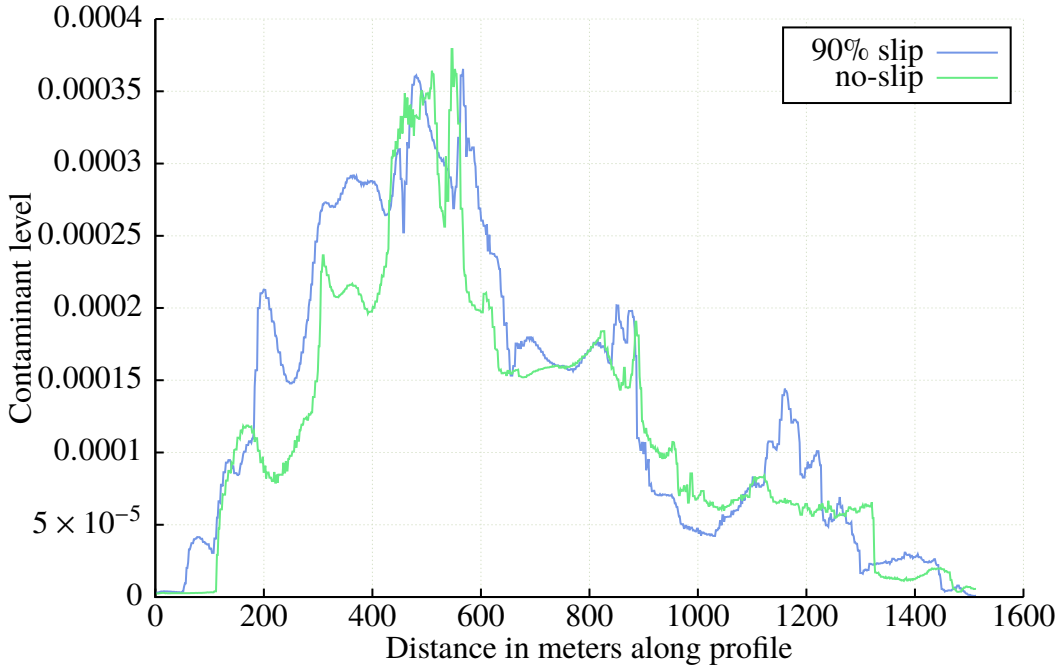


Figure 4.42 Urban contaminate levels for profile three at ground level for $Re = 342189$

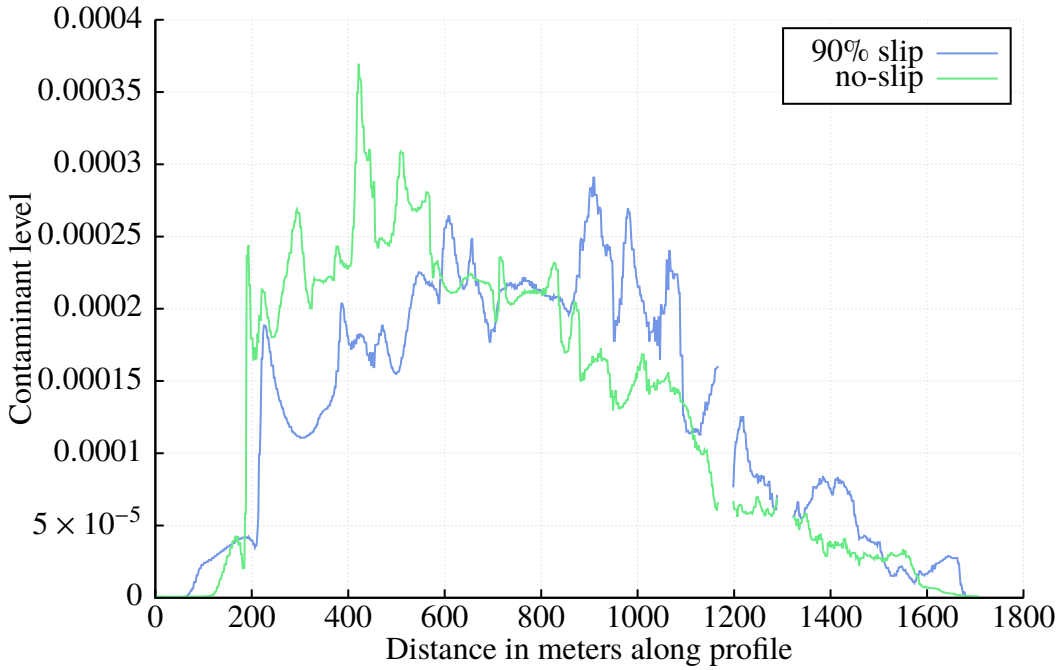


Figure 4.43 Urban contaminate levels for profile four at ground level for $Re = 342189$

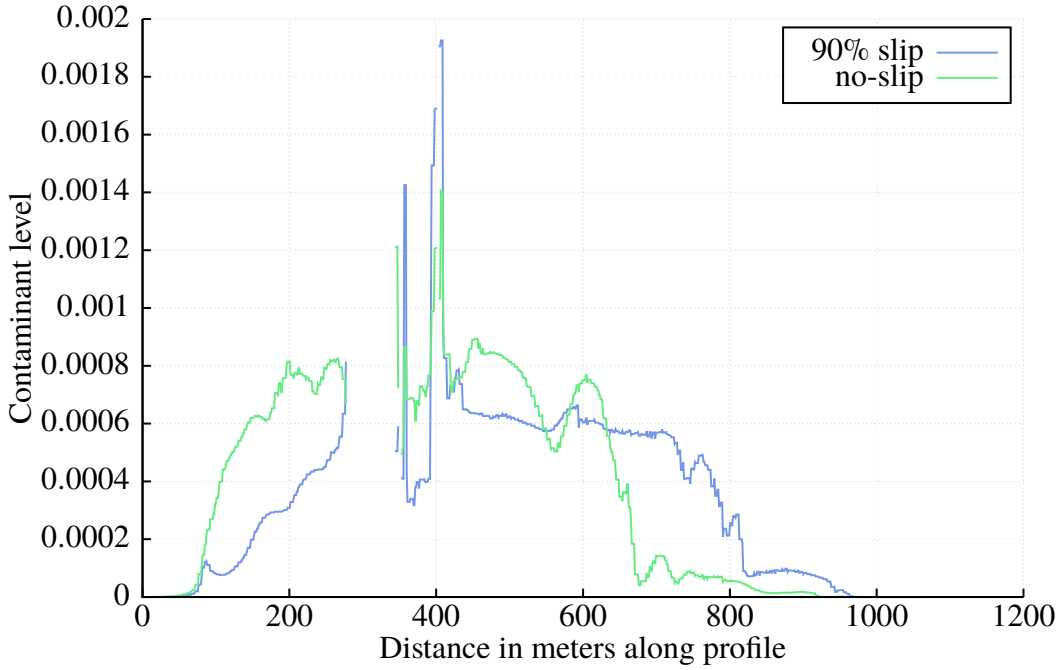


Figure 4.44 Urban contaminate levels for profile one at five meters for Re = 342189

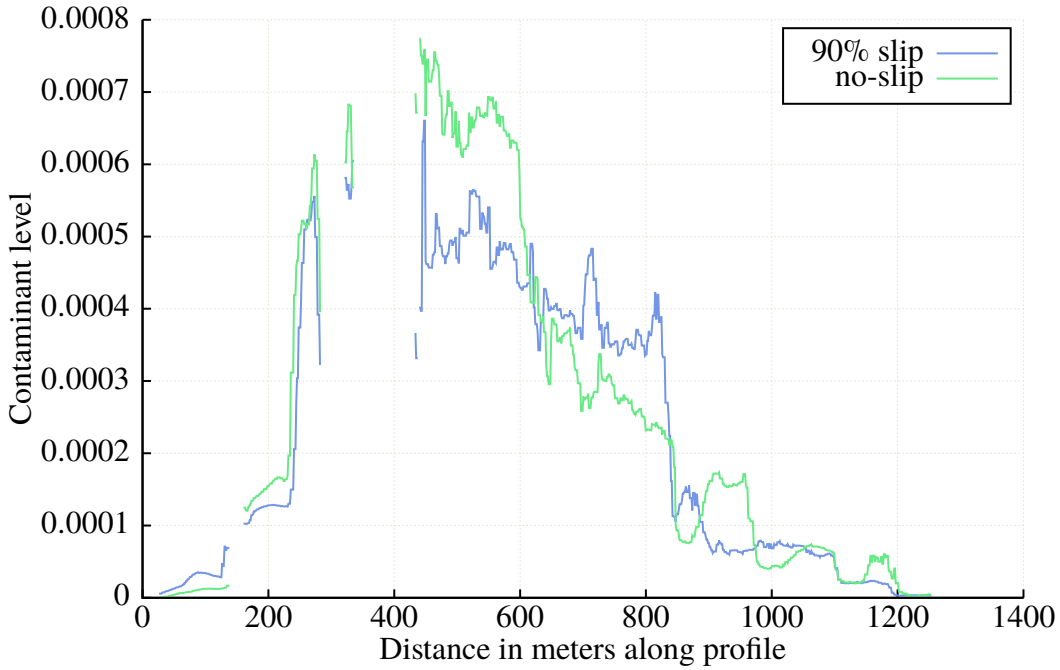


Figure 4.45 Urban contaminate levels for profile two at five meters for Re = 342189

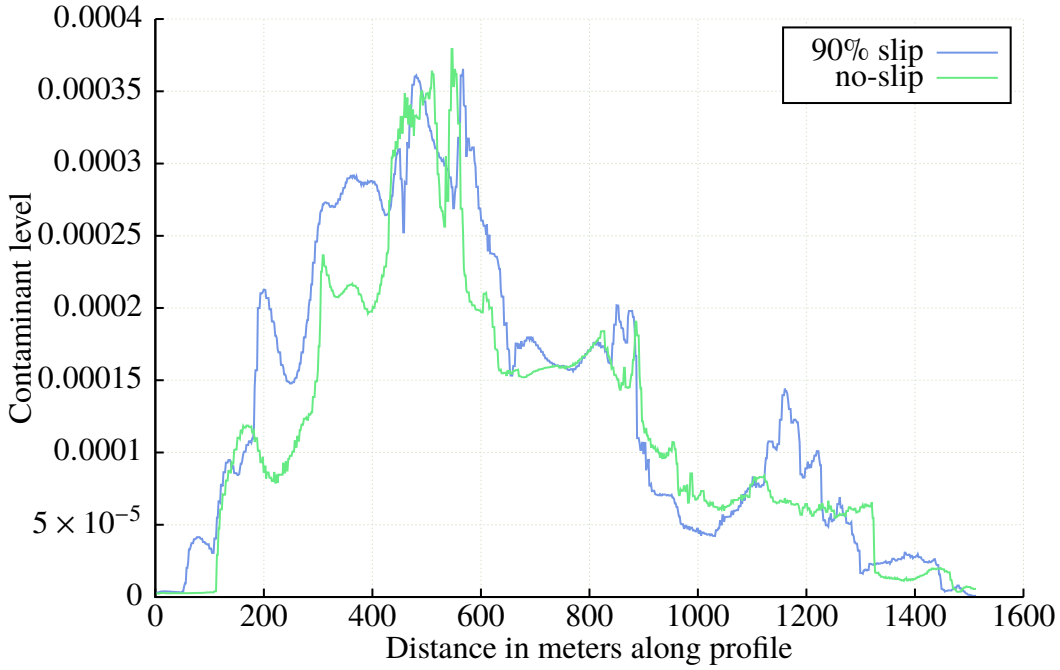


Figure 4.46 Urban contaminate levels for profile three at five meters for Re = 342189

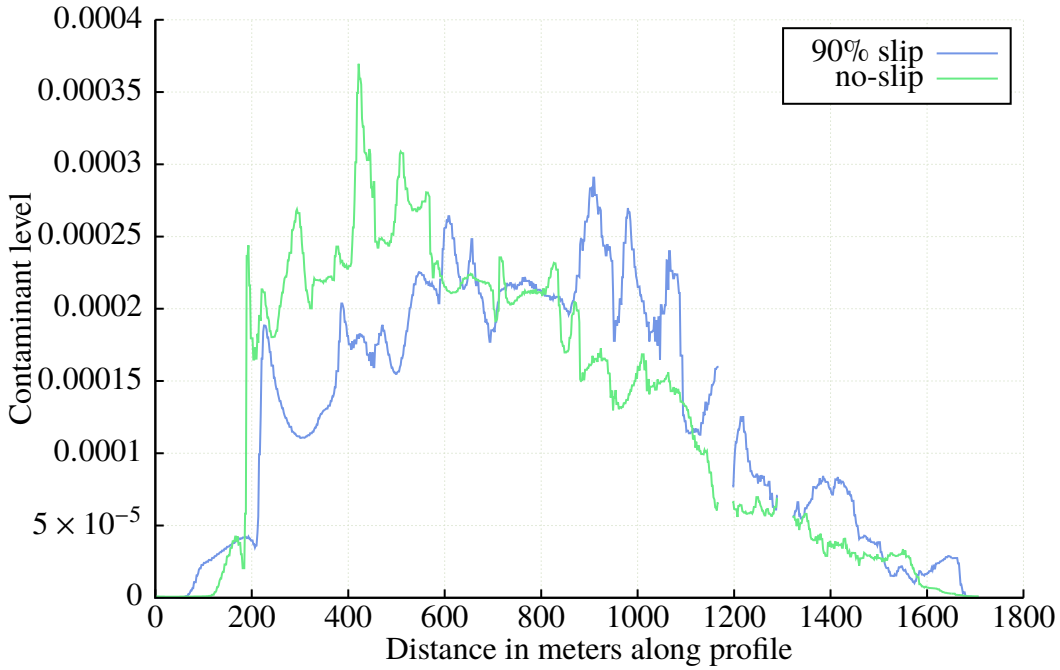


Figure 4.47 Urban contaminate levels for profile four at five meters for Re = 342189

CHAPTER 5

CONCLUSION

During this research a parallel, dynamic, automatic, strictly Cartesian aligned, linear forest-of-octrees grid generation application has been developed. Within the application, a generic and extensible PDE solver framework has been developed and tested using a form of the Navier-Stokes equations capable of addressing flows where buoyancy is non-trivial. This application is developed with distributed parallelism in mind and is realized with the `MPI` library. The main focus of the solver framework is to simulate urban scale problems, viz. plume/contaminant propagation, using the buoyant incompressible NS equations. However, any hyperbolic PDE could in theory be solved within the framework. This software leverages the open source project *p4est* to create an application that demonstrates excellent parallel grid generation speedup behavior and results in a computational mesh that is optimally load balanced. This application does *not* use cut-cell or projection methods, nor does it implement immersed boundary type algorithms; this means that the resulting computational mesh does *not* accurately represent any input geometry and is 'blocky' in nature. However, the software does provide a mechanism whereby certain specific geometries may be represented exactly in the final mesh. This mode is possible *only* when the geometry is itself planar and Cartesian aligned, e.g., a flat plate, square cylinder, forward or backward facing step, etc. A challenge pertaining to load balancing in certain algorithms used during the grid

generation phase, specifically geometry-centric algorithms, has been identified, explained, and improved significantly.

The forest-of-octrees structure is stored using a linear storage paradigm implemented using Morton encoding. Ramifications of this storage paradigm are far reaching—in both positive and negative ways. The main advantage is the functionally perfect load balancing it affords in terms of elements-per-process. One of the major drawbacks of this decision is the loss of hierarchical traversal and the benefits that come with hierarchical data structures. Certain portions of this application suffer significantly from the inability to traverse the octrees structure hierarchically. In order to lessen this impact, a single, coarser hierarchical octree is created in the geometry library and stores the geometry facets. This work leverages both types of octree structures during the creation of the computational mesh: the linear paradigm is used as a data-structure to store mesh elements because of its optimal parallel partitioning properties, while the hierarchical paradigm is leveraged for speed during geometry queries that benefit from spatial 'knowledge'. This mixed approach has proven to be an important feature as algorithms that benefit from the hierarchical storage do so significantly, sometimes gaining speedups greater than an order of magnitude.

Within the PDE solution framework, the buoyant incompressible NS PDEs are implemented using the pseudo-compressibility approach. This implementation is thoroughly tested and validated using various canonical test cases appropriate to the physics in question. The software is then successfully tested on a large scale urban geometry, both with and without heat and contaminant transfer.

To the author's knowledge, this is the first application of its type to use a linear forest-of-octrees data structure for its mesh generation. The forest of trees aspect of the mesh generation

affords greater flexibility with respect to the size and shape of mesh elements; it provides mechanisms to easily and naturally create meshes with isotropic or anisotropic elements without stringent requirements on the shape of the root element. For example, the only way to have isotropic elements in a single tree structure is if the root element itself is isotropic and, conversely, the only way to have anisotropic elements in the mesh using a single tree structure is if the root element is anisotropic. These restrictions go away when using a forest-of-trees structure, i.e. an anisotropic forest can include isotropic mesh elements and vice versa. Furthermore, the octree aspect of the application makes it possible to approximate buildings with significant three dimensionality, e.g. domes, spires, overhangs, or even bridges, whereas many other similar applications only support '2.5' dimensional building geometry.

Also, advances made regarding the load balancing of the flood-fill routine provide tangible benefits. The approach developed provides a method to significantly improve the balance of work during the flood-fill algorithm and leads to faster grid generation times. Although this method yields diminishing returns as the number of processes increases, it is still a valuable contribution.

5.1 Recommendations for Future Work

While this software has the potential to become a valuable tool with many areas of application, it is still very young and immature. There are many aspects of this research that would benefit from further development, refinement, validation, and research.

5.1.1 Grid Generation Considerations

The current adaptive grid refinement (AMR) functionality is sufficiently robust, but the coarsening logic needs to be extended to support the coarsening of boundary elements. The

addition of this feature is encumbered by the fact that the initial discretization of the geometry is also the final discretization; i.e., any boundary face present in the original discretization must always be present in any subsequent form of the grid. These faces may be refined or coarsened as long as the underlying shape of the initial discretization is maintained; unfortunately, this is more difficult during the coarsening phase of AMR. This application does not currently support exact discretization of curved geometry; all geometry is approximated by explicitly removing any mesh element that intersects the geometry resulting in 'blocky' or 'stepped' computational meshes. A cut-cell, immersed boundary, or projection method should be implemented in order to accurately represent any arbitrary geometry. These methods are well established and their implementation within the application should not be exceedingly difficult. Mesh movement would also be a valuable addition to the application. In order to do this however, the robustness of the application will need to be explored to make sure it is able to regenerate the computational mesh reliably every time geometry moves; currently the mesh is static once it has been created, except for when AMR is being used. This should not be a monumental task but one that must be approached with thought and care.

Another limitation of this application within the context of urban simulation is the manner in which the cityscape geometry must be represented. Currently, the building and terrain must be represented with a water-tight tessellation that the software uses to define the computational domain and determine the in-out status of elements therein. Obtaining said water-tight geometry may be a difficult task in and of itself and may prove to be equally prohibitive as the manual generation of the computational domain itself. There are a number of ways this could be assuaged. For example, many similar applications leverage commonly available GIS data directly for the definition of the

geometry without the need to generate a water-tight representation. Furthermore, for significant urban areas, if there does exist a water-tight representation, it will likely be too big for each process to load into memory; a client-server model could be implemented such that a single, larger-memory machine may load the geometry and handle all the related queries. This approach may pose a bit of a bottleneck; but given that the grid generation portion of the run-time is relatively small compared to the simulation time, this drawback should not present a significant problem.

Another aspect of this application that could use refinement pertains to urban simulations where there are significant features in the terrain: e.g., mountainous regions or places where humans have significantly modified the urban landscape during urban planning and development. Given the nature of the grid generation regarding its inability to exactly represent curved geometries, terrain poses an interesting problem, especially in regions near farfield boundaries where large mesh elements are desired. Succinctly, the 'blocky' or 'stepped' nature of the mesh creates large, physically unrealistic features in the mesh near areas of significant terrain curvature. This problem makes urban areas with appreciable variation in terrain more difficult to run and has the potential to adversely affect the solution quality because of non-physical boundaries present in the final computational domain. The implementation of proper boundary resolving mesh generation would clearly eliminate this problem but may add to the computational overhead. It may be possible for elements near terrain boundaries to be treated as porous media in such a way to model the effect of the terrain on the flow without explicitly resolving it.

5.1.2 Computational Considerations

An original goal of this research also entails developing a method for rapid simulation of an urban disaster scenario in order to provide real-time feedback to first responders. While the current state of the application does provide a significant automated advantage in such an event, the time-to-solution still requires concentrated optimization. Part of this goal is realized by the nature of the computational mesh generated by the software wherein geometry is under-resolved and full viscous resolution relaxed in favor of smaller representative meshes that afford an acceptable level of solution fidelity. Effort could be spent to maximize the unsteady time-step while maintaining stability in order to simulate larger physical time periods in a minimum number of time-steps. One way this could be improved is by implementing dual time-stepping. However, the most ground stands to be gained by taking advantage of hardware solutions and leveraging multiple appropriate parallelization paradigms. The state-of-the-art in high performance computing (HPC) is moving toward heterogeneous systems that take advantage of both distributed and shared memory paradigms, as well as accelerators, to attain maximum parallel performance. Significant effort must be put forth to take advantage of shared memory and accelerator paradigms within the context of a distributed memory application; non-trivial algorithmic and programming changes must be thought out and implemented. However, in order to achieve this level of parallel efficiency, these options must be explored thoroughly.

There are outstanding load-balancing issues that do not pose significant bottlenecks, but could benefit from some focused attention. The main difficulty is in achieving optimal load balancing when considering parts of the application that deal directly with geometry and geometry-centric functionality, e.g., boundary condition implementation, surface force calculations, etc.

This application provides nearly perfect load-balancing when it comes to volume-to-volume type computations, but it is more difficult to assure that all processors have been assigned an equal amount of boundary work. However, this problem is not specific to this application, it is a challenge common to many applications that perform the same types of computations.

The contaminant transport functionality of the code seems to work well, but for more realistic plume simulations the application should be modified to support non-neutrally buoyant contaminants. Many contaminants of interest are either heavier or lighter than air, and this characteristic of the particular contaminant will significantly affect the way that it propagates. That is, heavier plumes will tend to sink and spread outwardly more, while lighter plumes will rise more quickly and spread outwardly less. Integrating these effects more accurately into the solver will result in more accurate plume propagation simulations.

REFERENCES

[1] Blocken, B., "Computational Fluid Dynamics for Urban Physics: Importance, Scales, Possibilities, Limitations and Ten Tips and Tricks Towards Accurate and Reliable Simulations," *Building and Environment*, Vol. 91, 2015, pp. 219 – 245, Fifty Year Anniversary for Building and Environment.

[2] Leitl, B., Castelli, S. T., Baumann-Stanzer, K., Reisin, T. G., Barmpas, P., Balczó, M., Andronopoulos, S., Armand, P., Jurčáková, K., and Milliez, M., *Air Pollution Modeling and its Application XXIII*, chap. Evaluation of Air Pollution Models for Their Use in Emergency Response Tools in Built Environments: The 'Michelstadt' Case Study in COST ES1006 ACTION, Springer International Publishing, 2014, pp. 395–399.

[3] Epstein, J. M., Pankajakshan, R., and Hammond, R. A., "Combining Computational Fluid Dynamics and Agent-Based Modeling: A New Approach to Evacuation Planning," *PLoS ONE*, Vol. 6, No. 5, 2011, e20139.

[4] Cybyk, B., Boris, J., Theodore Young, J., Emery, M., and Cheatham, S., "Simulation of Fluid Dynamics Around Complex Urban Geometries," *39th AAIA Aerospace Sciences Meeting and Exhibit*, Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, Reno, Nevada, January 2001, AIAA-2001-803.

[5] Nichols, D. S., Mitchell, B., Sreenivas, K., Taylor, L., Briley, R., and Whitfield, D., "Aerosol Propagation in an Urban Environment," *36th AIAA Fluid Dynamics Conference and Exhibit*, Fluid Dynamics and Co-located Conferences, American Institute of Aeronautics and Astronautics, San Francisco, California, June 2006, AIAA-2006-3726.

[6] Balczó, M. and Lajos, T., "Flow and Dispersion Phenomena in a Simplified Urban Square," *Periodica Polytechnica Civil Engineering*, Vol. 59, No. 3, 2015, pp. 347–360.

[7] Britter, R. E. and Hanna, S. R., "Flow and Dispersion in Urban Areas," *Annual Review of Fluid Mechanics*, Vol. 35, No. 1, 2003, pp. 469–496.

[8] Ahmad, K., Khare, M., and Chaudhry, K., "Wind Tunnel Simulation Studies on Dispersion at Urban Street Canyons and Intersections–a Review," *Journal of Wind Engineering and Industrial Aerodynamics*, Vol. 93, No. 9, 2005, pp. 697 – 717.

[9] Huber, A. H., Freeman, M., Spencer, R., Schwarz, W., Bell, B., and Kuehlert, K., "Development and Applications of CFD Simulations Supporting Urban Air Quality and Homeland

Security," 2006, Presented at Symposium on the Urban Environment; AMS Annual Meeting, Atlanta, GA, January 29 - February 02, 2006.

[10] Hanna, S. R., Brown, M. J., Camelli, F. E., Chan, S. T., Coirier, W. J., Hansen, O. R., Huber, A. H., Kim, S., and Reynolds, R. M., "Detailed Simulations of Atmospheric Flow and Dispersion in Downtown Manhattan: An Application of Five Computational Fluid Dynamics Models," *Bulletin of the American Meteorological Society*, Vol. 87, No. 12, 2006, pp. 1713–1726.

[11] Walton, A., Cheng, A. Y. S., and Yeung, W. C., "Large-Eddy Simulation of Pollution Dispersion in an Urban Street Canyon – Part I: Comparison with Field Data," *Atmospheric Environment*, Vol. 36, No. 22, 2002, pp. 3601 – 3613.

[12] Walton, A. and Cheng, A. Y. S., "Large-Eddy Simulation of Pollution Dispersion in an Urban Street Canyon – Part II: Idealised Canyon Simulation," *Atmospheric Environment*, Vol. 36, No. 22, 2002, pp. 3615 – 3627.

[13] Reynolds, R. M., "The Madison Square Garden Dispersion Study (MSG05) Meteorological Data Description," Tech. Rep. BNL-77144-2006-IR, Brookhaven National Laboratory, Upton, New York, October 2006.

[14] Eichhorn, J., *Entwicklung und Anwendung eines Dreidimensionalen Mikroskaligen Stadtklima-Modells*, Ph.D. thesis, Universität Mainz, Germany, 1989.

[15] Eichhorn, J. and Kniffka, A., "The Numerical Flow Model MISKAM: State of Development and Evaluation of the Basic Version," *Meteorologische Zeitschrift*, Vol. 19, No. 1, February 2010, pp. 81–90.

[16] Nelson, M. and Brown, M., *The QUIC Start Guide (v 6.01): The Quick Urban and Industrial Complex (QUIC) Dispersion Modeling System*, Los Alamos National Laboratory, LA-UR-13-27291.

[17] "PALM - A large-eddy simulation model performing on massively parallel computers," *Meteorologische Zeitschrift*, Vol. 10, No. 5, September 2001, pp. 363–372.

[18] Maronga, B., Gryschka, M., Heinze, R., Hoffmann, F., Kanani-Sühring, F., Keck, M., Ketelsen, K., Letzel, M. O., Sühring, M., and Raasch, S., "The Parallelized Large-Eddy Simulation Model (PALM) Version 4.0 for Atmospheric and Oceanic Flows: Model Formulation, Recent Developments, and Future Perspectives," *Geoscientific Model Development*, Vol. 8, No. 8, 2015, pp. 2515–2551.

[19] Kanda, M., Inagaki, A., Miyamoto, T., Gryschka, M., and Raasch, S., "A New Aerodynamic Parametrization for Real Urban Surfaces," *Boundary-Layer Meteorology*, Vol. 148, No. 2, August 2013, pp. 357–377.

[20] Keck, M., Raasch, S., Letzel, M. O., Ng, E., and Ren, C., "High Resolution Large-Eddy Simulations of the Urban Canopy Flow in Macau," 2012, First International Education Forum on Energy and Environment, Hawai'i Island, Hawaii.

[21] Knoop, H., Keck, M., and Raasch, S., "Urban Large Eddy Simulation – Influence of a Densely Build-up Artificial Island on the Turbulent Flow in the City of Macau," `https://av.tib.eu/media/14368` [Accessed: May 17, 2016], 2014, Computer Animation.

[22] ESI Group, *CFD-ACE+ V2008.2 User Manual*, 2008.

[23] Coirier, W. J., Fricker, D. M., Furmanczyk, M., and Kim, S., "A Computational Fluid Dynamics Approach for Urban Area Transport and Dispersion Modeling," *Environmental Fluid Mechanics*, Vol. 5, No. 5, October 2005, pp. 443–479.

[24] Löhner, R., "A Fast Finite Element Solver for Incompressible Flows," *28th Aerospace Sciences Meeting*, AIAA Aerospace Sciences Meeting, American Institute of Aeronautics and Astronautics, Reno, Nevada, 1990, AIAA-1990-0398.

[25] Chan, S. T. and Leach, M. J., "A Validation of FEM3MP with Joint Urban 2003 Data," *Applied Meteorology and Climatology*, Vol. 46, No. 12, December 2007, pp. 2127–2146.

[26] Brook, D. R., Felton, N. V., Clem, C. M., Strickland, D. C. H., Griffiths, I. H., Kingdon, R. D., Hall, D. J., and Hargrave, J. M., "Validation of the Urban Dispersion Model (UDM)," *International Journal of Environment and Pollution (IJEP)*, Vol. 20, No. 1/2/3/4/5/6, 2003, pp. 11–21.

[27] Hall, D. J., Spanton, A., Griffiths, I., Hargrave, M., , and Walker, S., "The UDM: A model for estimating dispersion in urban areas," Tech. rep., 2000, Tech Report No. 03/00 (DERA-PTN-DOWN).

[28] Hall, D. J., Spanton, A., Griffiths, I. H., Hargrave, M., , and Walker, S., *The Urban Dispersion Model (UDM): Version 2.2 Technical Documentation*, 2003, DSTL/TR04774.

[29] Defense Threat Reduction Agency, *The Hazard Prediction Assessment Capability (HPAC) user's guide version 4.0.3*, 2005, Available from the Defense Threat Reduction Agency Consequence Assessment Branch (TDOC), 8725 John J. Kingman Road, Stop 6201, Fort Belvoir, VA 22060-6201 Telephone: 1-703-325-6106 Fax: 1-703-325-0398 Internet: http://www.dtra.mil.

[30] Chang, J. C., Hanna, S. R., Boybeyi, Z., and Franzese, P., "Use of Salt Lake City URBAN 2000 Field Data to Evaluate the Urban Hazard Prediction Assessment Capability (HPAC) Dispersion Model," *Applied Meteorology*, Vol. 44, No. 4, April 2005, pp. 485âĂŞ501.

[31] Briscolini, M. and Santangelo, P., "Development of the mask method for incompressible unsteady flows," *Journal of Computational Physics*, Vol. 84, No. 1, September 1989, pp. 57–75.

[32] Lohmeyer, A., Eichhorn, J., Flassak, T., and Kunz, W., "WinMISKAM 4.2, Microscale Flow and Dispersion Model for Built Up Areas, Recent Developments," *11th International Symposium Transport and Air Pollution, Proceedings Volume II*, VKM-THD Mitteilungen: Heft 81/VKM-THD Releases: Volume 81, University of Graz, Austria, June 2002.

[33] Environmental Systems Research Institute, Inc. (ESRI), "ESRI Shapefile Technical Description," Tech. rep., Environmental Systems Research Institute, Inc. (ESRI), 380 New York Street, Redlands, CA 92373-8100, July 1998.

[34] Coirier, W. J. and Kim, S., "CFD Modeling for Urban Area Contaminant Transport and Dispersion Model Description and Data Requirements," 2006, Paper JP2.11 at the *Sixth Symposium on the Urban Environment*, American Meteorological Society. Available on-line at www.ametsoc.org and on CD from AMS, 45 Beacon St., Boston, MA 02215.

[35] Löhner, R., Baum, J. D., Mestreau, E., Sharov, D., Charman, C., and Pelessone, D., "Adaptive Embedded Unstructured Grid Methods," *International Journal for Numerical Methods in Engineering*, Vol. 60, No. 3, 2004, pp. 641–660.

[36] Camelli, F., Lohner, R., and Hanna, S., "VLES Study of Flow and Dispersion Patterns in Heterogeneous Urban Areas," *44th AAIA Aerospace Sciences Meeting and Exhibit*, Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, Reno, Nevada, January 2006, AIAA-2006-1419.

[37] Chan, S. T., Humphreys, T. D., and Lee, R. L., "A Simplified CFD Approach for Modeling Urban Dispersion," *2004 AMS Annual Meeting*, 2004, pp. 11–15, On Tuesday 13 of January 2004 in Session 6: Emergency Response, Presentation 6.4.

[38] Khalighi, B., Jindal, S., Johnson, J. P., Chen, K. H., and Iaccarino, G., *The Aerodynamics of Heavy Vehicles II: Trucks, Buses, and Trains*, chap. Validation of the Immersed Boundary CFD Approach for Complex Aerodynamic Flows, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 21–38.

[39] García-Sánchez, C., Philips, D. A., and Gorlé, C., "Quantifying Inflow Uncertainties for CFD Simulations of the Flow in Downtown Oklahoma City," *Building and Environment*, Vol. 78, 2014, pp. 118–129.

[40] Gorlé, C., García-Sánchez, C., and Iaccarino, G., "Quantifying Inflow and RANS Turbulence Model Form Uncertainties for Wind Engineering Flows," *Journal of Wind Engineering and Industrial Aerodynamics*, Vol. 144, 2015, pp. 202–212, Selected papers from the 6th International Symposium on Computational Wind Engineering CWE 2014.

[41] MeteoDyn, "MeteoDyn Meteorology & Dynamics," Online, `http://meteodyn.com/en` [Accessed: May 26, 2016].

[42] MeteoDyn, "URBAWIND," Online, `http://meteodyn.com/en/logiciels/cfd-wind-pedestrian-comfort-safety-urbawind-software/` [Accessed: May 26, 2016].

[43] MeteoDyn, "UrbaWind Software: The CFD Software for Wind Simulation in Built Environment," Brochure.

[44] Fahssis, K., Dupont, G., and Leyronnas, P., "UrbaWind, a Computational Fluid Dynamics Tool to Predict Wind Resource in Urban Area," 2011, MeteoDyn online documentation.

[45] Kalmikov, A., "Uncovering MIT Wind Myths Through Micro-Climatological CFD Analysis," 2013, Paper published online.

[46] Kalmikov, A., Dupont, G., Dykes, K., and Chan, C., "Wind Resource Assessment in Complex Urban Environments: MIT Campus Case-Study using CFD Analysis," Poster, May 2010, Poster presented at the Wind Power Conference & Exhibition 2010, Dallas, Texas, USA.

[47] National Science Foundation (NSF), "EAGER: DMS: Disaster Mitigation System," `http://www.nsf.gov/awardsearch/showAward?AWD_ID=1236706` [Accessed: September 26, 2016].

[48] National Science Foundation (NSF), "Progress Made in Developing Systems for Disaster Mitigation," `http://www.nsf.gov/discoveries/disc_summ.jsp?cntn_id=126485` [Accessed: September 26, 2016].

[49] Currier, N. G., *Reacting Plume Inversion on Urban Geometries Through Gradient Based Design Methodologies*, Ph.D. thesis, The University of Tennessee at Chattanooga, August 2014.

[50] Yerry, M. A. and Shephard, M. S., "Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique," *International Journal for Numerical Methods in Engineering*, Vol. 20, No. 11, 1984, pp. 1965–1990.

[51] Shephard, M. S. and Georges, M. K., "Automatic Three-Dimensional Mesh Generation by the Finite Octree Technique," *International Journal for Numerical Methods in Engineering*, Vol. 32, No. 4, 1991, pp. 709–749.

[52] Quirk, J. J., *An Adaptive Grid Algorithm for Computational Shock Hydrodynamics*, Ph.D. thesis, Cranfield Institute of Technology, 1991.

[53] Quirk, J. J., "An Alternative to Unstructured Grids for Computing Gas Dynamic Flows Around Arbitrariliy Complex Two-Dimensional Bodies," Tech. rep., Institute for Computer Applications in Science and Engineering, NASA Langley Research Center Hampton, Virginia 23665-5225, 1992, ICASE Report 92-7.

[54] Quirk, J. J., "A Cartesian Grid Approach with Hierarchical Refinement for Compressible Flows," Tech. rep., Institute for Computer Applications in Science and Engineering, NASA Langley Research Center Hampton, Virginia 23681-0001, 1994, ICASE Report 94-51.

[55] Berger, M. J., "Adaptive Finite Difference Methods in Fluid Dynamics," Tech. rep., New York University, NY (USA). Courant Mathematics and Computing Laboratory, 1987, Report Number: DOE/ER/03077-277.

[56] Young, D. P., Melvin, R. G., Bieterman, M. B., Johnson, F. T., Samant, S. S., and Bussoletti, J. E., "A Locally Refined Rectangular Grid Finite Element Method: Application to Computational Fluid Dynamics and Computational Physics," *Journal of Computational Physics*, Vol. 92, No. 1, 1991, pp. 1–66.

[57] Powell, K. G., "The Adaptive Cut-Cell Cartesian Approach (Warts and All)," *ICASE/LaRC Workshop on Adaptive Grid Methods*, edited by J. C. South, Jr., J. L. Thomas, and J. Vanrosendale, Hampton, Virginia, November 1995, pp. 59–77, NASA Conference Publication 3316.

[58] Karman, Jr., S. L., "SPLITFLOW: A 3D Unstructured Cartesian/Prismatic Grid CFD Code for Complex Geometries," *33rd AIAA Aerospace Sciences Meeting and Exhibit*, Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, Reno, Nevada, January 1995, AIAA-1995-0434.

[59] Domel, N. and Karman, Jr., S. L., "Splitflow - Progress in 3D CFD with Cartesian Omni-Tree Grids for Complex Geometries," *38th AIAA Aerospace Sciences Meeting and Exhibit*, Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, Reno, Nevada, January 2000, AIAA-2000-1006.

[60] Karman, Jr., S. L., "Hierarchical Unstructured Mesh Generation," *42nd AIAA Aerospace Sciences Meeting and Exhibit*, Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, Reno, Nevada, January 2004, AIAA-2004-613.

[61] Karman, Jr., S. L. and Betro, V. C., "Parallel Hierarchical Unstructured Mesh Generation with General Cutting," *46th AIAA Aerospace Sciences Meeting and Exhibit*, Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, Reno, Nevada, January 2008, AIAA-2008-918.

[62] Karman, Jr., S. L., "Multi-Block Hierarchical Unstructured Grid Generation with Adaptation," *52nd Aerospace Sciences Meeting*, AIAA SciTech, American Institute of Aeronautics and Astronautics, National Harbor, Maryland, January 2014, AIAA-2014-0116.

[63] Burstedde, C., Wilcox, L. C., and Ghattas, O., "`p4est`: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees," *SIAM Journal of Scientific Computing*, Vol. 33, No. 3, 2011, pp. 1103–1133.

[64] Tu, T., O'Hallaron, D. R., and Ghattas, O., "Scalable Parallel Octree Meshing for TeraScale Applications," *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, SC '05, IEEE Computer Society, Washington, DC, USA, November 2005, p. 4.

[65] Cambridge Flow Solutions, "Cambridge Flow Solutions: Proprietary and Customised CFD," 2016, www.cambridgeflowsolutions.com/home/ [Accessed: July 6, 2016].

[66] Dawes, W., "Towards a Fully Parallel Integrated Geometry Kernel, Mesh Generator, Flow Solver & Post-processor," *44th AIAA Aerospace Sciences Meeting and Exhibit*, Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, Reno, Nevada, January 2006, AIAA-2006-0942.

[67] Demargne, A. A., Evans, R., Tiller, P., and Dawes, W. N., "Practical and Reliable Mesh Generation for Complex, Real-World Geometries," *52nd Aerospace Sciences Meeting*, AIAA SciTech, American Institute of Aeronautics and Astronautics, National Harbor, Maryland, January 2014.

[68] ITASCA Consulting Group, Inc., "KUBRIX Geo Version 15.0: Advanced Automatic and Interactive Grid Generation for Engineering Software," 2016, www.itascacg.com/software/kubrix-geo [Accessed: July 6, 2016].

[69] NUMECA International, "NUMECA International: Innovation & Quality," 2012, www.numeca.com/en [Accessed: July 6, 2016].

[70] NUMECA International, "AutoMesh-4G: NUMECA Grid Generation," Brochure, www.numeca.com/sites/numeca/files/automesh-4gtm.pdf [Accessed: July 6, 2016].

[71] NUMECA International, "HEXPRESS: Unstructured Full-Hexahedral Meshing," 2012, www.numeca.com/en/products/automeshtm/hexpresstm [Accessed: July 6, 2016].

[72] NASA, "Cart3D," people.nas.nasa.gove/aftosmis/cart3d/cart3Dhome.html [Accessed: May 16, 2016].

[73] Sampath, R. S., Adavani, S. S., Sundar, H., Lashuk, I., and Biros, G., "Dendro: Parallel Algorithms for Multigrid and AMR Methods on 2:1 Balanced Octrees," *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, IEEE Press, Piscataway, NJ, USA, November 2008, pp. 18:1–18:12.

[74] Bungartz, H., Mehl, M., and Weinzierl, T., *Euro-Par 2006 Parallel Processing: 12th International Euro-Par Conference, Dresden, Germany, August 28 – September 1, 2006. Proceedings*, chap. A Parallel Adaptive Cartesian PDE Solver Using Space–Filling Curves, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 1064–1074.

[75] Karypis, G. and Kumar, V., "MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0," http://www.cs.umn.edu/~metis, 2009.

[76] Karypis, G. and Kumar, V., "A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs," *SIAM Journal of Scientific Computing*, Vol. 20, No. 1, 1999, pp. 359–392.

[77] National Oceanic and Atmospheric Administration (NOAA), "National Centers for Environmental Information," http://www.ncei.noaa.gov [Accessed: January 22, 2016].

[78] National Oceanic and Atmospheric Administration (NOAA), "National Centers for Environmental Information," 2016, http://www.ncdc.noaa.gov/extremes/extreme-us-climates.php [Accessed: January 22, 2016].

[79] Morton, G. M., "A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing," Tech. rep., IBM, Ottawa, Ontario Canada, 1966.

[80] Mississippi State University, "CAVS SimCenter at Mississippi State University," http://www.simcenter.msstate.edu/research/cavs_cfd/aflr.php [Accessed: February 17, 2016], Contact: David L. Marcum at marcum@cavs.msstate.edu.

[81] Leidos, "Xpatch Electromagnetic Simulation Software," https://www.leidos.com/products/software/xpatch/ [Accessed: February 18, 2016].

[82] Frisken, S. F. and Perry, R. N., "Simple and Efficient Traversal Methods for Quadtrees and Octrees," *Journal of Graphics Tools*, Vol. 7, No. 3, 2002, pp. 1–11.

[83] Sundar, H., Sampath, R. S., and Biros, G., "Bottom-Up Construction and 2:1 Balance Refinement of Linear Octrees in Parallel," *SIAM Journal of Scientific Computing*, Vol. 30, No. 5, 2008, pp. 2675–2708.

[84] Hartmann, D., Meinke, M., and Schröder, W., "An Adaptive Multilevel Multigrid Formulation for Cartesian Hierarchical Grid Methods," *Computers & Fluids*, Vol. 37, No. 9, 2008, pp. 1103–1125.

[85] Hartmann, D., Meinke, M., and Schröder, W., "A Strictly Conservative Cartesian Cut-Cell Method for Compressible Viscous Flows on Adaptive Grids," *Computer Methods in Applied Mechanics and Engineering*, Vol. 200, No. 9-12, 2011, pp. 1038–1052.

[86] Chorin, A. J., "A Numerical Method for Solving Incompressible Viscous Flow Problems," *Journal of Computational Physics*, Vol. 2, No. 1, 1967, pp. 12–26.

[87] Taylor, L. K., *Unsteady Three-Dimensional Incompressible Algorithm Based on Artificial Compressibility*, Ph.D. thesis, Mississippi State University, May 1991.

[88] Kress, J. E., *An Unstructured Grid Incompressible Navier-Stokes Algorithm for Convective Heat Transfer Based on Artificial Compressibility*, Master's thesis, The University of Tennessee at Chattanooga, December 2012.

[89] Er, C. S., *Numerical Simulation of Laminar Incompressible Convective Heat Transfer Flow*, Master's thesis, Mississippi State University, December 1995.

[90] Pan, D. and Chakravarthy, S., "Unified Formulation for Incompressible Flows," *27th Aerospace Sciences Meeting*, Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, Reno, Nevada, January 1989, AIAA-1989-122.

[91] Hyams, D. G., *An Investigation of Parallel Implicit Solution Algorithms for Incompressible Flows on Unstructured Topologies*, Ph.D. thesis, Mississippi State University, May 2000.

[92] Palacios, F., Alonso, J. J., and Jameson, A., "Design of Free-Surface Interfaces using RANS Equations," *43rd Fluid Dynamics Conference*, Fluid Dynamics Conferences, American Institute of Aeronautics and Astronautics, San Diego, California, June 2013, AIAA-2013-3210.

[93] Boussinesq, J. V., *Théorie Analytique de la Chaleur*, Vol. II, Gauthier-Villars, Paris, 1903.

[94] Smagorinski, J., "General Circulation Experiments with the Primitive Equations I. The Basic Experiment," *Monthly Weather Review*, Vol. 91, No. 3, March 1963, pp. 99–164.

[95] Nichols, D. S., Private Correspondence, 2016.

[96] Toro, E., Spruce, M., and Spears, W., "Restoration of the contact surface in the HLL-Riemann solver," *Shock Waves*, Vol. 4, No. 1, 1994, pp. 25–34.

[97] Niu, Y.-Y., Chang, C.-H., Tseng, W.-Y. I., Peng, H.-H., and Yu, H.-Y., "Numerical Simulation of an Aortic Flow Based on a HLLC Type Incompressible Flow Solver," *Communications In Computational Physics*, Vol. 5, No. 1, 2009, pp. 142–162.

[98] Anderson, W. K. and Bonhaus, D. L., "An Implicit Upwind Algorithm for Computing Turbulent Flows on Unstructured Grids," *Computers and Fluids*, Vol. 23, No. 1, 1994, pp. 1–21.

[99] Beam, R. M. and Warming, R. F., "An Implicit Factored Scheme for the Compressible Navier-Stokes Equations," *American Institute of Aeronautics and Astronautics*, Vol. 16, No. 4, 1978, pp. 393–402.

[100] Blasius, H., "Grenzschichten in Flüssigkeiten mit kleiner Reibung," *Z. Math. Phys.*, Vol. 56, 1908, pp. 1–37.

[101] Davis, G. D. V., "Natural Convection of Air in a Square Cavity: a Bench Mark Numerical Solution," *International Journal for Numerical Methods in Fluids*, Vol. 3, No. 3, 1983, pp. 249–264.

[102] Ghia, U., Ghia, K. N., and Shin, C. T., "High-*Re* Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method," *Journal of Computational Physics*, Vol. 48, No. 3, 1982, pp. 387–411.

[103] Mandal, J. and Iyer, A., "An Upwind Method for Incompressible Flow Computations Using Pseudo-Compressibility Approach," *19th AIAA Computational Fluid Dynamics Conference*, Fluid Dynamics and Co-located Conferences, American Institute of Aeronautics and Astronautics, San Antonio, Texas, June 2009, AIAA-2009-3541.

[104] Iwatsu, R., Hyun, J. M., and Kuwahara, K., "Mixed Convection in a Driven Cavity with a Stable Vertical Temperature Gradient," *International Journal of Heat and Mass Transfer*, Vol. 36, No. 6, 1993, pp. 1601–1609.

[105] Agrawal, L., Mandal, J. C., and Marathe, A. G., "Computations of Laminar and Turbulent Mixed Convection in a Driven Cavity using Pseudo-Compressibility Approach," *Computers and Fluids*, Vol. 30, No. 5, 2001, pp. 607–620.

[106] Hyams, D. G., Sreenivas, K., Pankajakshan, R., Nichols, D. S., Briley, W. R., and Whitfield, D. L., "Computational Simulation of Model and Full Scale Class 8 Trucks with Drag Reduction Devices," *Computers and Fluids*, Vol. 41, No. 1, 2011, pp. 27–40.

[107] Taylor, L. K., Sreenivas, K., Webster, R. S., and Kress, J., "An Artificial Compressibility Algorithm for Convective Heat Transfer," *44th AIAA Thermophysics Conference*, Fluid Dynamics and Co-located Conferences, American Institute of Aeronautics and Astronautics, San Diego, California, June 2013, AIAA-2015-2894.

[108] Zdravkovich, M. M., *Flow Around Circular Cylinder, Volume One: Fundamentals*, Vol. 1, Oxford University Press, New York, July 1997.

[109] Breuer, M., Bernsdorf, J., Zeiser, T., and Durst, F., "Accurate Computations of the Laminar Flow Past a Square Cylinder Based on Two Different Methods: Lattice-Boltzmann and Finite-Volume," *International Journal of Heat and Fluid Flow*, Vol. 21, 2000, pp. 186–196.

[110] Sharma, A. and Eswaran, V., "Heat and Fluid Flow Across a Square Cylinder in the Two-Dimensional Laminar Flow Regime," *Numerical Heat Transfer, Part A: Applications*, Vol. 45, No. 3, 2004, pp. 247–269.

[111] Sohankar, A., Norberg, C., and Davidson, L., "Numerical Simulation of Unsteady Flow Around a Square Two-Dimensional Cylinder," *Proc. 12th Australian Fluid Mechanics Conference*, 1995, pp. 517–520.

[112] Sohankar, A., Norberg, C., and Davidson, L., "Low-Reynolds Number Flow Around a Square Cylinder at Incidence: Study of Blockage, Onset of Vortex Shedding and Outlet Boundary Condition," *International Journal for Numerical Methods in Fluids*, Vol. 26, No. 1, 1998, pp. 39–56.

[113] Martinuzzi, R. and Tropea, C., "The Flow Around Surface-Mounted, Prismatic Obstacles Placed in a Fully Developed Channel Flow (Data Bank Contribution)," *Journal of Fluids Engineering*, Vol. 115, No. 1, March 1993, pp. 85–92.

[114] Hajjawi, M. K., *Advanced Turbulence Closures for a Scalable Parallel Implicit Unstructured Grid Algorithm*, Ph.D. thesis, The University of Tennessee at Chattanooga, August 2007.

[115] Jameson, A., "Time Dependent Calculations Using Multigrid, with Applications to Unsteady Flows past Airfoils and Wings," *10th Computational Fluid Dynamics Conference*, Fluid

Dynamics and Co-located Conferences, American Institute of Aeronautics and Astronautics, Honolulu, HI, June 1991, AIAA-1991-1596.

APPENDIX A

BUOYANT INCOMPRESSIBLE NAVIER-STOKES EQUATIONS

This appendix will give more details about how the buoyant Navier-Stokes equations used during this research are derived. I am indebted to Dr. D. Steven Nichols for his contributions to this derivation; this entire appendix is an almost verbatim replication of notes he provided to me outlining this approach.

To start out, the equations will be derived assuming that gravity acts in the negative y-direction; in this case only the y-momentum equation needs to be considered as it is the only equation that is modified by the addition of buoyancy terms. Starting with the compressible y-momentum equation with a gravitational source term

$$\frac{\partial \rho v}{\partial t} + \nabla \cdot (\rho v \boldsymbol{u}) = -\frac{\partial p}{\partial y} + \nabla \cdot \left[(\mu + \mu_T) \left(\frac{\partial u_i}{\partial y} + \frac{\partial v}{\partial x_i} \right) \right] - \rho g \tag{A.1}$$

and assuming incompressibility and dividing through by the constant reference density ρ_∞ yields the incompressible y-momentum equation

$$\frac{\partial v}{\partial t} + \nabla \cdot (v \boldsymbol{u}) = -\frac{1}{\rho_\infty} \frac{\partial p}{\partial y} + \nabla \cdot \left[(\nu + \nu_T) \left(\frac{\partial u_i}{\partial y} + \frac{\partial v}{\partial x_i} \right) \right] - g \tag{A.2}$$

Thus, the source term vector, S, in equation (3.1) is

$$\boldsymbol{S} = \begin{Bmatrix} 0 \\ 0 \\ -g \\ 0 \\ 0 \end{Bmatrix} \tag{A.3}$$

or, in another way

$$\boldsymbol{S} = \frac{1}{\rho_\infty} \begin{Bmatrix} 0 \\ 0 \\ -\rho g \\ 0 \\ 0 \end{Bmatrix} \tag{A.4}$$

By the Boussinesq approximation [93], the density may be written as a function of temperature as

$$\rho(T) = \rho_\infty (1 + \rho') \tag{A.5}$$

where $\rho' = -\alpha_T(T - T_\infty)$. Substituting equation (A.5) into equation (A.4) yields

$$\boldsymbol{S} = \frac{1}{\rho_\infty} \begin{Bmatrix} 0 \\ 0 \\ -\rho_\infty(1+\rho')g \\ 0 \\ 0 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ -g - \rho' g \\ 0 \\ 0 \end{Bmatrix} \tag{A.6}$$

Now, in order to write the equations in conservation form, the $-g$ term needs to be combined into the pressure without affecting the $-\frac{1}{\rho_\infty}\frac{\partial p}{\partial x}$ and $-\frac{1}{\rho_\infty}\frac{\partial p}{\partial z}$ terms. Recall that for this part of the derivation, gravity acts in the y-direction only, so gravity needs to be accounted for in the y-direction without affecting the x and z directions. The pressure gradients and the gravity term for all momentum equations are

$$\begin{aligned} x\text{-direction: } & -\frac{1}{\rho_\infty}\frac{\partial p}{\partial x} + 0 = -\frac{1}{\rho_\infty}\frac{\partial p}{\partial x} - \frac{\partial \delta}{\partial x} \\ y\text{-direction: } & -\frac{1}{\rho_\infty}\frac{\partial p}{\partial y} - g = -\frac{1}{\rho_\infty}\frac{\partial p}{\partial y} - \frac{\partial \delta}{\partial y} \\ z\text{-direction: } & -\frac{1}{\rho_\infty}\frac{\partial p}{\partial z} + 0 = -\frac{1}{\rho_\infty}\frac{\partial p}{\partial z} - \frac{\partial \delta}{\partial z} \end{aligned} \tag{A.7}$$

where δ needs to be chosen to satisfy the equations. Assuming that g is constant, it is easy to see that $\delta = g\,y$ works, i.e. $\frac{\partial g\,y}{\partial x} = \frac{\partial g\,y}{\partial z} = 0$ and $\frac{\partial g\,y}{\partial y} = g$. Now, the terms may be combined

$$\begin{aligned} x\text{-direction: } & -\frac{\partial}{\partial x}\left(\frac{p}{\rho_\infty} + g\,y\right) = -\frac{\partial p^*}{\partial x} \\ y\text{-direction: } & -\frac{\partial}{\partial y}\left(\frac{p}{\rho_\infty} + g\,y\right) = -\frac{\partial p^*}{\partial y} \\ z\text{-direction: } & -\frac{\partial}{\partial z}\left(\frac{p}{\rho_\infty} + g\,y\right) = -\frac{\partial p^*}{\partial z} \end{aligned} \tag{A.8}$$

where $p^* = \frac{p}{\rho_\infty} + g\,y$ is the modified pressure. Applying the non-dimensionalization discussed in Section 3.2 to p^* and the y-momentum equation using equations (3.21) yields the following non-dimensional equations

$$p^* = p + \frac{y}{Fr^2} \tag{A.9}$$

and

$$\frac{\partial v}{\partial t} + \nabla \cdot (v\boldsymbol{u}) = -\frac{\partial p^*}{\partial y} + \nabla \cdot \left[\frac{(\nu + \nu_T)}{Re} \left(\frac{\partial u_i}{\partial y} + \frac{\partial v}{\partial x_i} \right) \right] - \frac{\rho'}{Fr^2} \tag{A.10}$$

where $Fr = \frac{u_\infty}{\sqrt{gL}}$, is the Froude number. At this point, the source term vector is

$$\boldsymbol{S} = \begin{Bmatrix} 0 \\ 0 \\ -\frac{\rho'}{Fr^2} \\ 0 \\ 0 \end{Bmatrix} \tag{A.11}$$

The source term needs to be rewritten in order to arrive at the final form used in this work. This form is required because it is written in terms of temperature, and temperature drives the buoyancy. Start with the dimensional source term $\rho' g = \alpha_T (T - T_\infty) g$, (although, note that ρ' is already non-dimensional) and recall that during the non-dimensionalization, this term gets multiplied by $\frac{L}{u_\infty}$ and that the non-dimensional temperature is $\hat{T} = \frac{T - T_\infty}{T_w - T_\infty}$. Performing the non-dimensionalization

to the source term yields

$$\begin{aligned}\frac{L}{u_\infty^2}(-\rho' g) &= \frac{L}{u_\infty^2}\left[\alpha_T(\hat{T}\Delta T)g\right] \\ -\rho'\frac{gL}{u_\infty^2} &= \alpha_T\frac{Lg}{u_\infty^2}\hat{T}\Delta T \\ -\frac{\rho'}{Fr^2} &= \frac{\alpha_T\hat{T}\Delta T}{Fr^2}\end{aligned} \tag{A.12}$$

Thus, dropping the hat (ˆ) syntax, the source term vector may be written as

$$\boldsymbol{S} = \begin{Bmatrix} 0 \\ 0 \\ \frac{\alpha_T T\Delta T}{Fr^2} \\ 0 \\ 0 \end{Bmatrix} \tag{A.13}$$

Recall that $\frac{\alpha_T T\Delta T}{Fr^2}$ may be easily written as $T\frac{Gr}{Re^2}$, which is the final form used in this dissertation. This is the derivation for the simple case where gravity is aligned in a coordinate direction, specifically, the negative y-direction.

A generalization is outlined below that allows for gravity to be aligned arbitrarily in lieu of choosing a Cartesian axis. Beginning with the compressible momentum equations written in tensor form with an unaligned gravity vector

$$\frac{\partial \rho u_i}{\partial t} + \nabla\cdot(\rho u_i \boldsymbol{u}) = -\frac{\partial p}{\partial x_i} + [\text{viscous terms}] + \rho g e_i \tag{A.14}$$

The gravity vector is $\boldsymbol{g} = g\boldsymbol{e}$ where $\boldsymbol{e}$ is a unit vector defined as

$$\boldsymbol{e} = \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix} \tag{A.15}$$

and $g \approx 9.81\ \mathrm{m\,s^{-2}}$ or $g \approx 32.19\ \mathrm{ft\,s^{-2}}$. In order to recover the original formulation with gravity aligned in the negative y-direction, $\boldsymbol{e}$ may be written as

$$\boldsymbol{e} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} \tag{A.16}$$

Applying the Boussinesq approximation as before and dividing through by ρ_∞ gives the incompressible momentum equations

$$\begin{aligned} \frac{\partial u_i}{\partial t} + \nabla \cdot (u_i \boldsymbol{u}) &= -\frac{1}{\rho_\infty}\frac{\partial p}{\partial x_i} + [\text{viscous terms}] + (1 + \rho')ge_i \\ &= -\frac{1}{\rho_\infty}\frac{\partial p}{\partial x_i} + [\text{viscous terms}] + ge_i + \rho' ge_i \end{aligned} \tag{A.17}$$

As before, the $-\frac{1}{\rho_\infty}\frac{\partial p}{\partial x_i} + ge_i$ term needs to be combined into the modified pressure term p^*. Expanding the tensor notation gives

$$\begin{aligned} x\text{-direction: } &-\frac{1}{\rho_\infty}\frac{\partial p}{\partial x} + ge_x = \frac{\partial p^*}{\partial x} \\ y\text{-direction: } &-\frac{1}{\rho_\infty}\frac{\partial p}{\partial y} + ge_y = \frac{\partial p^*}{\partial y} \\ z\text{-direction: } &-\frac{1}{\rho_\infty}\frac{\partial p}{\partial z} + ge_z = \frac{\partial p^*}{\partial z} \end{aligned} \tag{A.18}$$

where $p^* = \frac{p}{\rho_\infty} - g\delta$ such that $\frac{\partial \delta}{\partial x} = e_x$, $\frac{\partial \delta}{\partial y} = e_y$, and $\frac{\partial \delta}{\partial z} = e_z$. One simple form for δ that satisfies these constraints is

$$\delta = xe_x + ye_y + ze_z = \boldsymbol{x} \cdot \boldsymbol{e} \tag{A.19}$$

Now equation (A.17) may be written as

$$\frac{\partial u_i}{\partial t} + \nabla \cdot (u_i \boldsymbol{u}) = -\frac{\partial p^*}{\partial x_i} + [\text{viscous terms}] + \rho' g e_i \tag{A.20}$$

Performing the non-dimensionalization on equation (A.20) yields

$$\frac{\partial u_i}{\partial t} + \nabla \cdot (u_i \boldsymbol{u}) = -\frac{\partial p^*}{\partial x_i} + [\text{viscous terms}] + \frac{\rho' e_i}{Fr^2} \tag{A.21}$$

where $\frac{\rho' e_i}{Fr^2} = -\frac{\alpha_T T \Delta T}{Fr^2} e_i$, $p^* = p - \frac{\hat{\delta}}{Fr^2}$, and $\hat{\delta} = \frac{\delta}{L} = \frac{\boldsymbol{x}}{L} \cdot \boldsymbol{e}$. Finally, the final form of the source term vector for an arbitrarily aligned gravitational vector is

$$\boldsymbol{S} = \frac{\rho'}{Fr^2} \begin{Bmatrix} 0 \\ e_x \\ e_y \\ e_z \\ 0 \end{Bmatrix} = -\frac{\alpha_T T \Delta T}{Fr^2} \begin{Bmatrix} 0 \\ e_x \\ e_y \\ e_z \\ 0 \end{Bmatrix} \tag{A.22}$$

Using this approach, δ is the distance between the control volume centroid and a plane perpendicular to $\boldsymbol{e}$ that intersects with the origin. This plane does not have to be specified, only the 'down' direction must be known.

VITA

Ethan Alan Hereth was born in Eureka Springs, Arkansas, on September 26, 1981 to Alan Walter and Brenda Louise Hereth. He was home-schooled throughout his elementary and high school career and graduated in 2001. After taking two years off to explore a career as a musician, he enrolled full time at the University of Central Arkansas in Conway, Arkansas where he developed an interest in mathematics. In 2007 Ethan graduated *Summa Cum Laude* with a Bachelor of Science degree in Applied Mathematics and received the O. L. Hughes *Outstanding Senior Mathematics Major* Award; his minor was Spanish. He then continued his education as a graduate teaching assistant at the University of Central Arkansas where he graduated with a Masters of Science degree in Applied Mathematics in 2009. Ethan subsequently accepted a graduated research assistantship at the University of Tennessee Chattanooga SimCenter and started taking classes in the fall of 2009. In 2011 he took full time position as a Research Associate at the SimCenter as a grid generation professional and continued as a part time graduate student. Ethan graduated with a Ph.D. in Computational Engineering in December of 2016.