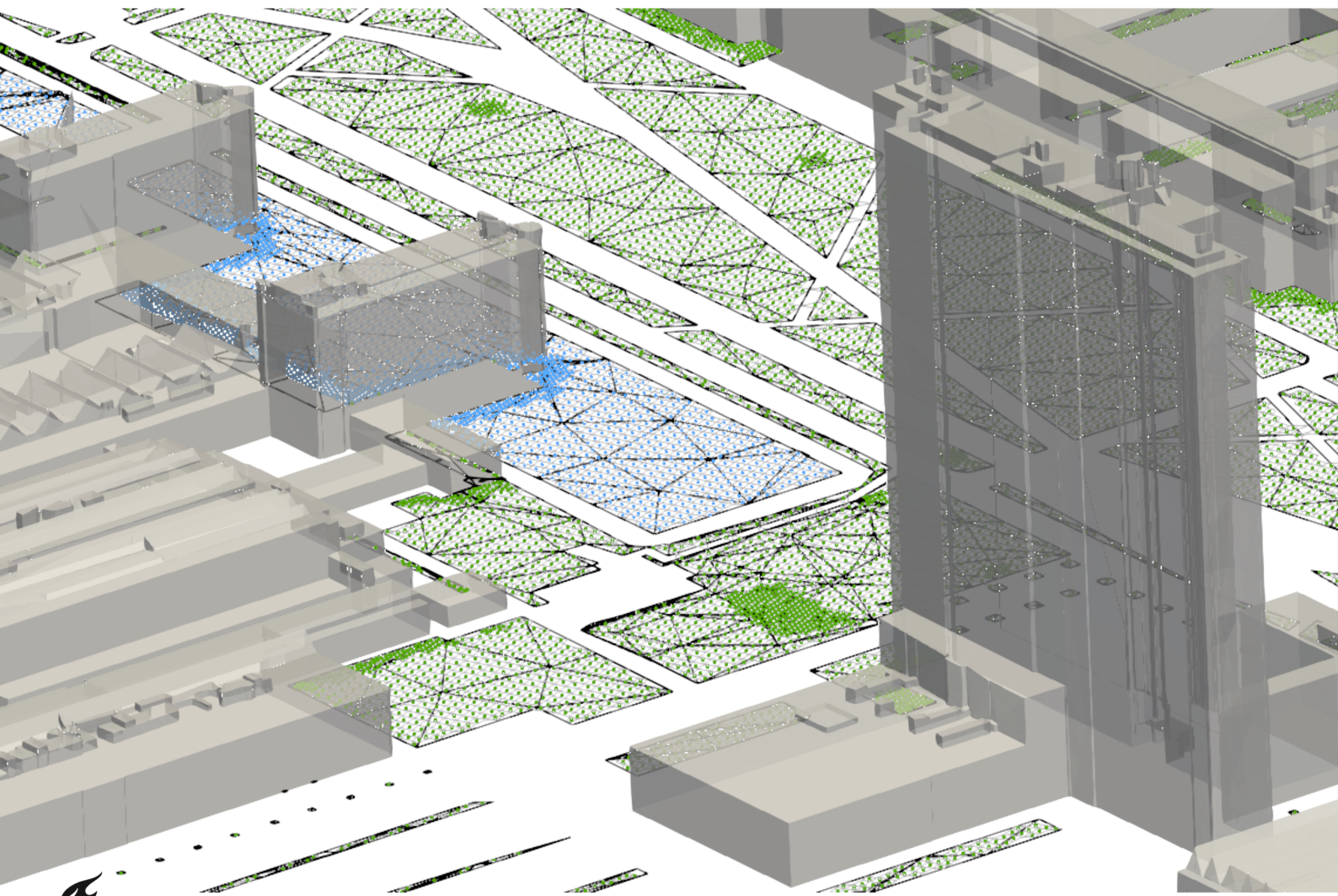


# From GIS to 3D flow simulations in urban environments

Christina Fratzeskou

October 2022





MSc thesis in Geomatics

# **From GIS to 3D flow simulations in urban environments**

Christina Fratzeskou

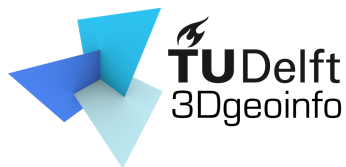
October 2022

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of Master of Science in Geomatics

Christina Fratzeskou: *From GIS to 3D flow simulations in urban environments* (2022)

© This work is licensed under a Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.

The work in this thesis was carried out in the:



3D geoinformation group  
Department of Urbanism  
Faculty of the Built Environment & Architecture  
Delft University of Technology

Supervisors: Dr. Clara García-Sánchez  
Dr. Hugo Ledoux  
Co-reader: Ivan Pađen

# Abstract

Urban physics is a multi-scale and interdisciplinary field, that combines science and engineering for the study of physical processes in urban areas. Computational Fluid Dynamics (CFD) is considered a powerful tool for the study of these processes. In the context of microscale phenomena these processes refer to the transfer of heat and mass in the indoor and outdoor urban environment, and can be observed up to  $\approx 1\text{km}$  above the surface of the Earth. Their development takes place inside the Atmospheric Boundary Layer (ABL), which for the case of urban areas is referred to as the Urban Boundary Layer (UBL), and is formed due to the interactions between the surface obstacles and the wind flows. In CFD simulations the surface of the Earth and the encountered obstacles are represented with the use of 3D models, as well as estimated values that are used to implicitly represent their roughness. In this thesis the aim is to investigate relevant to CFD parameters that could be used as 3D model semantics. For this purpose, a list of parameters was identified. From this list roughness length was selected and a methodology was developed for the assignment of roughness values for the open-source software OpenFOAM. The developed methodology that was built using built-in functions of OpenFOAM, is based on an octree data structure that is used to store the input triangulated model in obj format. The roughness length landuse names are stored in an mtl file that complements the input obj and the roughness length values are specified as a user defined parameter using the landuse names. The process is semi-automated and it entails the assignment of non-uniform roughness at the bottom of the computational domain. Additionally, a methodology to assign non-uniform roughness at the inlet of the domain was developed. The results showed that the assignment of non-uniform roughness at the bottom of the domain was successful for cases of flat terrain, however, under the restriction that the input geometry model abided by certain geometry guidelines, such as no self-intersections, no gaps. For the case of non-uniform roughness at the inlet the process also produced satisfactory results in terms of the assignment process, however the impact of multiple roughness values at the inlet on the calculated flow parameters requires further investigation.



# Acknowledgements

Firstly, I would like to express my gratitude to my supervisor, Clara García-Sánchez, who encouraged and guided me in all matters related to CFD. Our weekly meetings motivated me and helped me keep track of my work. Secondly, I would like to thank Hugo Ledoux, for his valuable insight, while building the methodology presented in this thesis and for his support throughout this process. I would also like to thank Ivan Pađen, for his feedback on this report and Jorge Mejía Hernández for organising the presentation meetings.





# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Objectives and research questions . . . . .	2
1.2	Research scope . . . . .	2
1.3	Thesis Outline . . . . .	3
<b>2</b>	<b>Theoretical background and related work</b>	<b>5</b>
2.1	Concepts from urban physics . . . . .	5
2.1.1	Urban Boundary Layer . . . . .	6
2.1.2	Geometrical properties of obstacles in the UBL . . . . .	7
2.1.3	Material properties of surfaces in the UBL . . . . .	9
2.2	CFD simulations . . . . .	10
2.2.1	Mesh generation . . . . .	10
2.2.2	Flow modelling . . . . .	11
2.3	Coupling GIS and CFD simulations for the urban environment . . . . .	12
2.4	Specification of nonuniform roughness length . . . . .	12
2.5	Octree data structure . . . . .	13
<b>3</b>	<b>Methodology</b>	<b>15</b>
3.1	Application design . . . . .	15
3.1.1	Input/output . . . . .	17
3.1.2	Implementation environment . . . . .	18
3.1.3	Code structure . . . . .	19
3.1.4	Option -setZ0Ground . . . . .	20
3.1.5	Option -setZ0Inlet . . . . .	24
3.1.6	User-defined parameters . . . . .	26
3.2	Limitations and Requirements . . . . .	27
3.3	Quality control . . . . .	28
<b>4</b>	<b>Data, case pre-processing and testing</b>	<b>31</b>
4.1	Datasets . . . . .	31
4.2	Cases pre-processing . . . . .	34
4.2.1	Case configuration . . . . .	34
4.2.2	Simple case grid independence study . . . . .	39
4.3	Testing . . . . .	43
<b>5</b>	<b>Implementation and tools</b>	<b>45</b>
5.1	Case structure . . . . .	45
5.2	Integration in OpenFOAM . . . . .	46
5.3	Classes . . . . .	46
5.4	Functions . . . . .	47
5.5	Tools . . . . .	48
<b>6</b>	<b>Results and analysis</b>	<b>49</b>
6.1	Grid similarity between the use cases . . . . .	49
6.2	Option -setZ0Ground . . . . .	51
6.3	Option -setZ0Inlet . . . . .	60
<b>7</b>	<b>Conclusions and future work</b>	<b>63</b>
7.1	Research Objectives . . . . .	63

*Contents*

7.2 Discussion . . . . .	64
7.3 Future work and recommendations . . . . .	65
<b>A Reproducibility self-assessment</b>	<b>67</b>
A.1 Marks for each of the criteria . . . . .	67
A.2 Self-reflection . . . . .	67

# List of Figures

1.1	Overview of OpenFOAM structure [Greenshields, 2019]. . . . .	2
2.1	Spatial(horizontal) and temporal scales of phenomena in urban physics (as updated by [Schlünzen et al., 2011]) found in [Blocken, 2015]. . . . .	5
2.2	Horizontal spatial scales and respective analysis tools for the study of urban physics [Blocken, 2015]. . . . .	6
2.3	Vertical spatial scales of the Urban Boundary layer (UBL) [Blocken, 2015]. . . . .	7
2.4	Effect of surface roughness on the mean wind speed profile, $z_g$ denotes the depth of the boundary layer above which mean wind speed is constant with height (Davenport,1965 in [Oke, 1978]). . . . .	7
2.5	Example of computational domain representation [Blocken, 2015]. . . . .	8
2.6	Example of computational domain with outlined boundaries [Blocken et al., 2007b]. . . . .	10
2.7	Examples of structured grids [Jayanti, 2018a]. . . . .	11
2.8	Examples of unstructured grids [Jayanti, 2018a]. . . . .	11
2.9	Octree of depth 2 visualisation [Su et al., 2016]. . . . .	13
3.1	Overview of the methodology. . . . .	15
3.2	Examples of the user defined dictionary file for $\epsilon$ . . . . .	16
3.3	Example of combining obj and mtl file. . . . .	18
3.4	Shell specification for setting the roughness length parameter with option <code>-setZ0Ground</code> . . . . .	19
3.5	Plot of the number of missed points for different seed distances for a case of flat terrain. . . . .	21
3.6	Screenshots of the 'complex' case vtk files illustrating the roughness assignment for a seed distance of 0.1 (3.6a) and 1 million (3.6b). Missed points are in red colour. . . . .	22
3.7	Example of misassigned roughness values for an inlet boundary. The different colours indicate two different roughness length values. . . . .	24
3.8	Configuration file for user-defined parameters. . . . .	27
3.9	Comparison between a triangulated surface without overlaps (a) and a surface with overlapping triangles (b) (surface: grey, triangles of surface: blue, overlapping surface triangles: green). . . . .	28
4.1	s_0 case surface model ('brown': terrain, 'green': vegetation, and 'blue': water). . . . .	32
4.2	Study area for c_0 case in the TU Delft Campus. . . . .	32
4.3	c_0 case surface model ('red': buildings, 'green': vegetation, and 'blue': water). . . . .	33
4.4	Overlaps between the geometries for 'Green'(triangles are represented by their centers in red) and 'Water'(triangulated surface in light blue) landcovers for the complex case. . . . .	33
4.5	Unified terrain surface models for the simple cases. . . . .	35
4.6	Unified terrain surface models used for the complex cases. . . . .	35
4.7	Illustration of the refinement boxes used in the complex cases. Boxes 1-3: refined to level 3,2,1 respectively, Box 4: refined to level 4, area of interest is in grey. . . . .	36
4.8	Probe configuration for s_0 and s_1 cases. . . . .	38
4.9	Probe configuration for c_0 and c_1 cases. . . . .	39
4.10	Screenshots of the s_0 case grids included in the mesh independence study. . . . .	40
4.11	Convergence for $U_x$ at four selected probe locations. . . . .	41
4.12	Convergence for $U_z$ at four selected probe locations. . . . .	41
4.13	Vertical profiles(930m-960m) for $U_x$ , $\epsilon$ and $k$ at four locations. . . . .	42
5.1	'complex' case folder structure. . . . .	45
5.2	Structure of application directory. . . . .	46

List of Figures

6.1	Plot of the occurrences of assigned $z_0$ values per landcover for the simple cases. . . . .	51
6.2	Illustration of the assigned roughness length for case s_1. . . . .	51
6.3	$U_x$ at four probe locations. . . . .	52
6.4	$U_z$ at four probe locations. . . . .	52
6.5	Illustration of the assigned roughness length values for the c_0 case. . . . .	53
6.6	Illustrations of the assigned roughness length as generated from option <code>-setZ0Ground</code> . The red boxes signify the areas with observed differences. . . . .	53
6.7	Plot of the occurrences of assigned $z_0$ values per category for the complex case. . . . .	54
6.8	Differences per landcover based on the assigned roughness values between the c_0 and c_1 cases. . . . .	54
6.9	$U_y$ at fifty one probe locations at height 2m. . . . .	55
6.10	$U_z$ at fifty one probe locations at height 2m. . . . .	55
6.11	$U_y$ at ten probe locations with different roughness length at height 2m. . . . .	56
6.12	$U_z$ at ten probe locations with different roughness length at height 2m. . . . .	56
6.13	Differences between the c_0 and c_1 case for $U_{magnitude}$ based on a 2D slice at 2m height above ground. . . . .	57
6.14	Contour map with contours per 0.375 [m/s] for $U_{magnitude}$ for the c_0 case, using a 2D slice at 2m height above ground. . . . .	58
6.15	Contour maps with contours per 0.375 [m/s] for $U_{magnitude}$ , using a 2D slice at 2m height above ground. . . . .	58
6.16	Screenshots of a zoomed area in the contour maps of <a href="#">Figure 6.15</a> with observed differences. . . . .	59
6.17	Contour maps with contours per 0.375 [m/s] for $U_{magnitude}$ , using a 2D slice at 2m height above ground. . . . .	59
6.18	$z_0$ values for the s_2 case with non-uniform roughness at the inlet ( $z_0$ : 0.05 (terrain), 0.03 (vegetation)). . . . .	60
6.19	Screenshots of the s_2 case domain inlet and bottom boundaries. <a href="#">6.19b</a> displays a zoomed view of the separation line between the two geometries. The lines in 'white' are the edges of the triangulated geometries, while in 'blue' the wireframe of the meshed boundaries. . . . .	60
6.20	$z_0$ values for c_2 case with non-uniform inlet ( $z_0$ : 0.03 (vegetation), 0.0002 (water)). . . . .	61
6.21	Screenshots of the c_2 case domain inlet and bottom boundaries. <a href="#">6.21b</a> displays a zoomed view of the separation line between the two geometries. The lines in 'white' are the edges of the triangulated geometries, while in 'blue' the wireframe of the meshed boundaries. . . . .	61
6.22	s_2, s_2.1 cases comparison based on convergence values for $U_x$ and $U_z$ . . . . .	62
6.23	c_2 and c_2.1 cases comparison based on convergence values for $U_x$ , $U_z$ . . . . .	62
A.1	Reproducibility criteria to be assessed [ <a href="#">Nüst et al., 2018</a> ]. . . . .	67

# List of Tables

4.1	Details of the triangulated terrain models used in the original cases. . . . .	31
4.2	Details of all triangulated terrain models used in the test cases. . . . .	34
4.3	Assigned roughness length ( $z_0$ ) values per patch. . . . .	38
4.4	Boundary conditions set-up. The types used for the bottom boundary of the domain refer to all relevant patches that represent ground surfaces, for all other surfaces (i.e. buildings), the <code>epsilonWallFunction</code> is used. . . . .	38
4.5	Details of simulations for grid independence. . . . .	39
4.6	Differences for $U_x$ between the medium, fine and finer grids at selected point locations. . . . .	40
4.7	Correspondence between the application options and the tested cases. . . . .	43
6.1	Mesh characteristics for the simple and complex cases generated from the OpenFOAM utility <code>checkMesh</code> . . . . .	49
6.2	Number of cells per refinement level for the simple and complex cases as generated from the OpenFOAM utility <code>checkMesh</code> . . . . .	50
6.3	Number of faces per boundary patch for the simple and complex cases as generated from the OpenFOAM utility <code>checkMesh</code> . . . . .	50
6.4	Number of points per $U_{\text{magnitude}}$ difference interval. . . . .	57
A.1	The self-assigned grading for the reproducibility criteria presented in <a href="#">Figure A.1</a> . . . . .	67



# List of Algorithms

- 3.1 ROUGHNESS LENGTH ASSIGNMENT . . . . . 20
- 3.2 FIND NEAREST . . . . . 23
- 3.3 OVERLAPS . . . . . 23
- 3.4 CHECKPOINTS . . . . . 25
- 3.5 CHECKPOINTS' COORDS . . . . . 25
- 3.6 PARAMETER ASSIGNMENT . . . . . 25





# Acronyms

UBL	Urban Boundary layer	xi
GIS	Geographical Information System	1
CFD	Computational Fluid Dynamics	1
ABL	Atmospheric Boundary Layer	1
obj	Wavefront object	2
BL	Boundary Layer	6
UCL	Urban Canopy Layer	6
LAI	Leaf Area Index	9
LAD	Leaf Area Density	9
RANS	Reynolds Averaged Navier Stokes equations	9
LES	Large Eddy Simulation	11
DNS	Direct Numerical Simulations	11
GPL	General Public License	17
stl	Stereolithography ASCII	17
stlb	Stereolithography binary	17
vtk	Legacy Visualisation Toolkit	17
frt	OpenFOAM triangulated format	17
ac	Invis AC3D	17
smesh	TenGen surface mesh format	17
tri	Triangle format	17
dx	OpenDX format	17
BGT	Basisregistratie Grootchalige Topografie	17
BAG	Basisregistraties Adressen en Gebouwen	17
mtl	Wavefront Material Template Library	18



# 1 Introduction

The past decades urban physics has become a focus area of increasing importance since its many applications can contribute to a better understanding of global scale challenges that urban areas face. Examples of such challenges, which are closely related to increasing urbanization, are climate change, energy consumption, pollution, transportation etc. [Blocken, 2015]. As a multi-scale and interdisciplinary field, urban physics combine science and engineering for the study of physical processes in urban areas. These physical processes mainly refer to the transfer of heat and mass in the outdoor and indoor urban environment, as well as its interactions with humans, vegetation and materials [Blocken, 2015].

As an applied discipline it requires the collection of data. The type of data required depends on the nature (i.e. objective, scale) of the phenomenon under study. They can be ancillary data used to calculate conditions, represent parts of the urban environment that influence the flow [Jie et al., 2014] or they can be directly related to the calculation of the flow parameters [Back et al., 2021]. The former are either field measurements, reference-based or derived using Geographical Information System (GIS) methodologies [Jie et al., 2014], while the latter are collected in the field or in laboratories (eg wind-tunnel testing) or they can be the product of numerical methods such as Computational Fluid Dynamics (CFD) [Back et al., 2021]. In reality, although meteorological phenomena are observed and studied in different spatial and temporal scales, they are also connected through space and time [Oke, 1978]. Therefore, studies in the field usually employ a mixture of the aforementioned methods [Toparlar et al., 2017].

In the context of urban physics, CFD simulations are usually implemented for the study of phenomena that belong to microscale meteorology, which entails interactions between the wind in the lower part of the Atmospheric Boundary Layer (ABL) and the surface of the Earth [Blocken, 2014]. For urban areas ABL is referred to as UBL. It is formed due to the surface obstacles that wind flows encounter as they move over a surface, such as natural morphology, vegetation and man-made structures (Chapter 1.1 in [Stull, 1988]). In micro-scale CFD simulations terrain and surface mounted obstacles can be represented explicitly or implicitly in the computational domain [Franke and Baklanov, 2007].

When an object is represented explicitly its actual geometry is used and it should be larger than the computational domain mesh cell size. Requirement for the geometry is that it is represented by a watertight 3D model [Blocken, 2015]. Structures in the area of interest are always represented explicitly. On the other hand, an implicit representation refers to objects (buildings, trees, bridges etc.) or ground surfaces located further away from the area of interest or cases of 2D rough surfaces in the area of interest that influence the flow [Blocken, 2015]. An implicit representation is expressed using a surface roughness value. Apart from the morphological characteristics of urban surfaces there are other surface properties that have an important role in the formulation of the flow and influence transport processes that take place in the UBL [Grimmond and Souch, 1994].

Therefore, in this thesis, the focus will be on the coupling of GIS and CFD, and in particular, on identifying relevant surface parameters that could be used as semantics for the input 3D model. For this purpose an application for the assignment of roughness length ( $z_0$ ) values using as input an enhanced 3D model is implemented for the open source software OpenFOAM. Figure 1.1 shows an overview of OpenFOAM structure. Among the identified potential parameters, roughness length was selected since roughness in the lower part of the UBL has a significant influence on the flow [Stull, 1988], facilitating the process of evaluating the results of the application.

For the testing of the application two cases were used, an ideal case and one corresponding to an actual location, part of the TU Delft Campus.

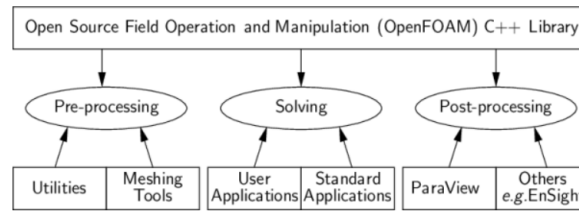


Figure 1.1: Overview of OpenFOAM structure [Greenshields, 2019].

## 1.1 Objectives and research questions

The main objective of this study is to examine the potential benefits of using roughness length as an added semantic to the input 3D model of a CFD simulation. For this purpose, a prototype application for the open-source software OpenFOAM was built. In CFD, roughness length is specified as part of the wall function modelling during the pre-processing step of the simulation. Currently assigning multiple roughness values in OpenFOAM can be a time consuming process, requiring the generation of separate geometries for each surface type. Furthermore, the assignment of multiple roughness values in OpenFOAM is only currently implemented for patches located at the bottom boundary of the domain.

A second objective for this study is to identify potential parameters that could be used as added semantics for input 3D models in CFD simulation for the urban environment.

The main objective of this project can be summarised in the following research question:

*How can non-uniform roughness length be integrated in a CFD software like OpenFOAM through the use of 3D model semantics?*

In order to answer the main question of this project the following relevant sub-questions are formulated:

- To what extent can the integration be automated?
- How does the modified assignment process of roughness length at the bottom of the domain influences the process and results of the simulation?
- How does the modified assignment process for non-uniform roughness length at the inlet of the domain influences the process and results of the simulation?
- Which other relevant to CFD parameters could be used as 3D model semantics with the built application?

## 1.2 Research scope

The scope of this thesis is determined based on the following:

- The parameters presented as potential 3D model semantics were selected based on a non exhaustive literature review.
- The only parameter tested with the application is roughness length. For the incorporation of other parameters further development of the application and research is required.
- The proposed methodology is only tested with flat surface models. However, it is developed to allow for the assignment of semantics to 3D models.
- The implementation is limited to accept surface models in Wavefront object (obj) file format.
- The application was built for cases with rectangular computational domain.

- There is no geometric validation included in the current implementation. 3D models used with the application are considered to be watertight, with no overlapping surfaces.
- Cases of simulations that fall outside the field of urban physics will not be tested, but they could potentially benefit from the open source developments of this study.

## 1.3 Thesis Outline

In this introductory chapter of the thesis the motivation and main objectives of the research were outlined. The content of the consecutive chapters is structured as follows:

**Chapter 2** provides an overview of concepts related to urban physics and CFD simulations relevant to this study. This includes the selected parameters that were considered to have potential as 3D model semantics. In addition, related work regarding the assignment of non-uniform roughness is presented. Lastly, theoretical background required for the understanding of the selected methodology is presented.

**Chapter 3** presents the methodology followed for the development of the application along with explanation on the design choices and evaluation methods.

**Chapter 4** includes detailed presentation of the datasets used and the configuration of the cases selected for the testing of the application.

**Chapter 5** provides implementation details regarding the classes, functions and tools.

**Chapter 6** presents and discusses the application and simulation results.

**Chapter 7** answers the research questions of this study and provides recommendations for future work.



# 2 Theoretical background and related work

In this chapter concepts from the fields of urban physics and geomatics that are relevant to the topic of this thesis are presented, as well as related studies. Firstly, an overview of concepts from urban physics is presented based on the main characteristics of the UBL. In the second part, Section 2.2, key concepts relevant to CFD simulations are presented. In the third part, Section 2.3, the interaction of GIS and CFD is presented through the identification of fields that GIS has the potential to provide support and complement the process of a CFD simulation. The last sections, are dedicated to the analysis of methods and concepts relevant to the application presented in Chapter 3.

## 2.1 Concepts from urban physics

Urban physics mainly refers to the study of physical processes that take place in the lower part of the atmosphere (i.e. troposphere, approximately 10km). The effects and influence of the urban environment, however, can be observed in much larger temporal and spatial scales [Blocken, 2015]. As shown in Figure 2.1 based on their horizontal spatial extent meteorological phenomena can be divided in three main categories: macroscale, mesoscale and microscale phenomena [Blocken, 2015]. The scales as displayed

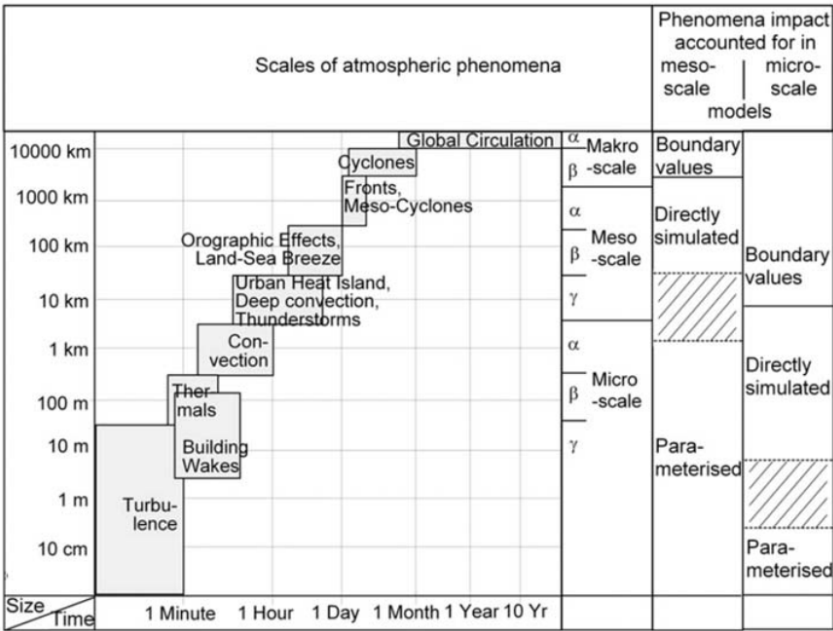


Figure 2.1: Spatial(horizontal) and temporal scales of phenomena in urban physics (as updated by [Schlünzen et al., 2011]) found in [Blocken, 2015].

in Figure 2.1 are indicative, as there is a debate regarding their extents [Oke, 1978]. This is a result of the fact that in reality atmospheric phenomena can not be so clearly distinguished as they are part of a continuum [Oke, 1978].

Common denominator for meteorological phenomena, in the context of urban physics, is the influence of the urban environment, which depending on the studied phenomenon and the employed analysis tool, it can be explicitly or implicitly represented. In the context of this thesis, the focus will be on CFD

## 2 Theoretical background and related work

studies. Relevant to CFD are the spatial scales with horizontal extent that can reach up to 2km (Figure 2.2). This is based on the study of phenomena that in meteorological terms happen in the microscale and in spatial terms are influenced by the building scale [Blocken, 2015]. Furthermore, it should be noted that micro-scale studies require explicit representation of geometries in contrast to meso-scale studies where the morphological aspects are parametrised in the majority of cases, mainly due to the much larger spatial scales (Figure 2.3). In the micro-scale, the physical processes under study entail the transfer of heat and mass in the indoor and outdoor urban environment, as well as the interaction of fluids with humans, nature and man-made structures [Blocken, 2015].





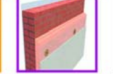

Spatial scale	Global	Mesoscale	Microscale	Building	Component	Material/Human
Distance	< 6500 km	< 200 km	< 2 km	< 100 m	< 10 m	< 1 m
						
Model cat.	NWP	NWP / MMM	CFD	CFD / BES	BC-HAM	MSM / HTM

Figure 2.2: Horizontal spatial scales and respective analysis tools for the study of urban physics [Blocken, 2015].

The domain in which urban physics in the micro-scale are applied and studied is the ABL [Oke, 1978]. The ABL is the part of the atmosphere where the majority of human activities take place. The nature of the ABL is influenced by the morphology and characteristics of the Earth's surface. This means that in the case of urban areas the complexity of their morphology and the diversity of their elements differentiate them from the case of rural areas [Piringer et al., 2002]. For urban areas the ABL can be referred to as the UBL [Lateb et al., 2016].

### 2.1.1 Urban Boundary Layer

Based on the detailed presentation found in *Boundary Layer Meteorology* by Stull [1988], an overview of the ABL is firstly provided. The ABL is the Boundary Layer (BL) that is formed as a response to the interaction of the Earth's surface with the lower part of the atmosphere. This interaction can be better described via transport processes that transfer heat and mass, mainly caused by surface forcings. These surface forcings include heat transfer, transpiration, evaporation, frictional drag, pollutant emissions and terrain obstacles. Catalysts in these processes are the ground's diurnal temperature variations (i.e. due to the absorption of solar radiation), terrain morphology and manmade structures. The determinant factor in the formation of the ABL and one of the main factors that differentiate the ABL from free atmosphere is that alterations in air flows and its physical properties can be observed in a timescale of one hour or less. The height at which these alterations can be observed varies from a few hundred meters up to 3 km, after which free atmosphere begins. Although the surface can influence indirectly the whole of the troposphere, the alterations are much slower [Stull, 1988].

Generally the way transport processes develop can be described by mean wind, waves and turbulence, although in reality wind flows are not so clearly distinguished motions [Stull, 1988]. Inherently the ABL is turbulent, since the occurrence of turbulent air flows is triggered by the surface forcings, which are also an intrinsic part of the ABL [Oke, 1978]. In the case of turbulence its random and complex nature makes turbulence one of the most difficult concepts to define and study. This is evident in the vertical spatial scales shown in Figure 2.3. In particular, highly turbulent flows are observed within the Urban Canopy Layer (UCL), which can be attributed to the proximity of that part of the atmosphere to the surface roughness elements.

Based on the above it could be derived that the nature of the UBL is highly dependent on the morphology and characteristics of the built environment. Due to the different surface properties and morphology, different types of surfaces and obstacles participate in the exchange of energy and transport processes in different ways [Grimmond and Souch, 1994]. These properties are a combination of material (e.g. reflectivity, absorptivity, emissivity, texture etc.) and morphological characteristics (e.g. composition of rough elements, spatial extent, building clusters) [Oke, 1978].



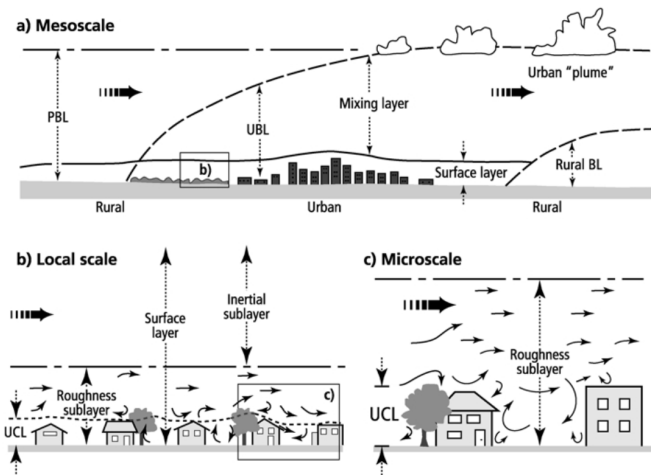


Figure 2.3: Vertical spatial scales of the UBL [Blocken, 2015].

### 2.1.2 Geometrical properties of obstacles in the UBL

The morphological variances encountered in the surface of the ABL are referred to as roughness and they are one of the main sources of three-dimensional alterations (e.g. turbulence) in the UBL [Oke, 1978]. In particular, in the absence of temperature stratification (e.g. neutral ABL, no strong thermal effects) the depth of the surface frictional influence is dependant on the surface roughness (Figure 2.4). It can be derived that the depth of the urban ABL increases with increasing roughness [Oke, 1978].

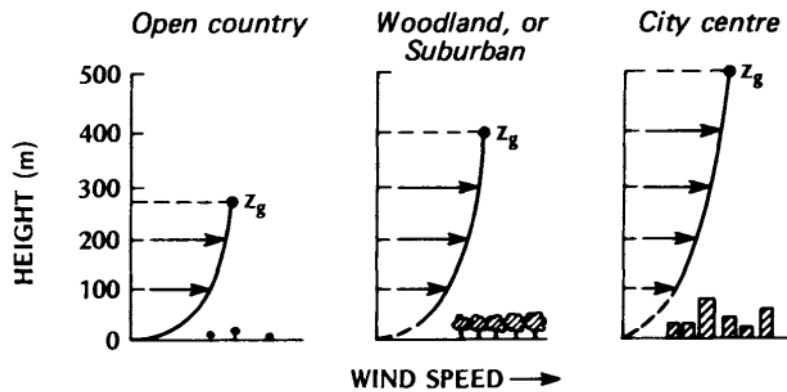


Figure 2.4: Effect of surface roughness on the mean wind speed profile,  $z_g$  denotes the depth of the boundary layer above which mean wind speed is constant with height (Davenport, 1965 in [Oke, 1978]).

The urban fabric is comprised of variable roughness elements, heights, densities, canyon like geometries and types of surfaces, both natural and manmade [Piringer et al., 2007]. In numerical simulations the representation of those features can be done explicitly through the use of actual geometries or implicitly with the use of representative estimated values [Blocken, 2015]. The chosen representation depends on the selected tool which in turn is dictated by the scale of the phenomenon under study. In the list that follows some of the available approaches for the implicit representation of the morphology of the urban environment are presented:

- Roughness length ( $z_0$ ) [m]: It is the height above the surface of the Earth at which the mean logarithmic wind profile extrapolates to zero [Oke, 1978]. It is used to represent the impact of roughness elements and surfaces on the wind [Oke, 1978], with estimated values that can incorporate both the morphological and aerodynamic nature of obstacles and surfaces [Zhou et al., 2006]. The nature of roughness length was investigated by Zhang et al. [2012], who examined the interaction

## 2 Theoretical background and related work

of  $z_0$  of vegetated surfaces and near-surface flows and argued that for the case of flexible roughness elements it should be treated as a dynamic parameter rather than as a solely geometric one. It can be used to represent both homogeneous and inhomogeneous surfaces [Zhou et al., 2006]. Estimations for inhomogeneous surfaces are commonly used in meso-scale studies [Wieringa, 1993], where obstacles and surfaces need to be parametrised due to the large extent of the study areas [Blocken, 2015]. In micro-scale numerical studies, roughness length is used to represent surfaces and obstacles in the upstream and downstream areas of the computational domain, as well as surfaces in the area of interest (central part of the domain as shown in Figure 2.5), for which there is no explicit geometry representation [Blocken, 2015].

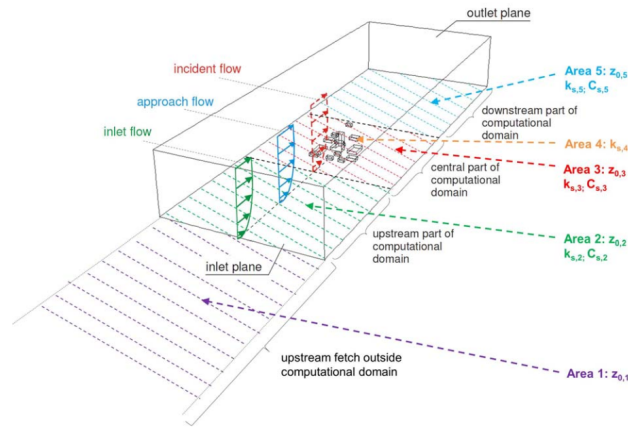


Figure 2.5: Example of computational domain representation [Blocken, 2015].

In Ricci et al. [2020] among other parameters the impact of surface roughness height<sup>1</sup> ( $k_s$ ) on the prediction of wind flows in urban areas was investigated. According to the study, which was conducted as part of a wider research regarding the uncertainties of numerical simulations, it was found that the differences in surface roughness had a more significant impact on the results in the upstream and downstream regions near the bottom of the computational domain. In addition it is suggested that the choice of roughness can have a larger influence in urban environments with open areas, as opposed to high density urban areas where the local forcing effects caused by the presence of the buildings overshadows any other effects.

There are three approaches based on which the roughness length values can be determined:

1. Reference-based: For this approach the values are selected through comparison between data depicting surface cover/use (e.g. site photos, maps), morphology (e.g. height, density of elements) and published tabulated values [Kent et al., 2017]. A well known and commonly used list of roughness length values is the updated Davenport classification for homogeneous surfaces as updated and published by Wieringa [1992] [Blocken, 2015].

These estimated values for roughness length are based on experiments and observations and there are a number of factors that should be considered. The selection of appropriate values is a difficult task, since for inhomogeneous surfaces characteristics such as elevation, spatial density and the shape of the roughness elements are not unique [Kondo and Yamazawa, 1986].

2. Anemometric: This approach entails the use of sensors placed at appropriately selected locations [Kent et al., 2017].
3. Morphometric: This approach includes the calculation of indexes, that represent characteristics of the urban environment that have an impact on the flow regimes [Kent et al., 2017]. Examples of these indexes are the Average roughness element height ( $H_{av}$ ), the Maximum

<sup>1</sup>Equivalent sand-grain roughness given by  $k_s = 9.793z_0/C_s$  based on Blocken et al. [2007b,a]

roughness element height ( $H_{max}$ ), the Plan area index ( $A_p$ ), the Total area under consideration ( $A_f$ ), the Frontal area index ( $\lambda_f$ ) [Kent et al., 2017].

The anemometric and morphometric methods can account for the morphology of the studied area as well as meteorological conditions (e.g. wind direction, speed etc.) [Kent et al., 2017].

In CFD simulations roughness length is specified through wall functions (further explained in ??) applied at the bottom and inlet boundaries of the computational domain [Blocken et al., 2007b]. In particular for the case of the inlet (i.e. the boundary through which the flow enters the domain), roughness length represents the roughness of the area upstream, located outside the computational domain [Blocken et al., 2007b].

- Leaf Area Density (LAD) [ $m^{-1}$ ] and Leaf Area Index (LAI) [-]: Both indicators are used to describe the effect of trees on the wind flow. LAD is leaf area per unit volume of space and LAI is the integration of the LAD over height [Deininger et al., 2020]. In contrast to LAD, LAI can be used more effectively to specify different types of trees [Segersson, 2017].

Deininger et al. [2020] implemented two approaches for the modelling of vegetation in a CFD simulation. In the first approach an explicit representation of trees was used using solid geometries, while in the second vegetation was modelled implicitly with the use of a canopy model. The latter approach included adding source terms to the Reynolds Averaged Navier Stokes equations (RANS) equations that represented the vegetation induced resistance of the flow. For this LAD and LAI were used. The height and locations of the vegetated areas were derived from point cloud and landuse data respectively. In comparison to the first approach, the second provided more realistic results. Furthermore, Segersson [2017] developed a workflow that allows the application of non-uniform LAI to a computational domain bottom patch for a CFD simulation using OpenFOAM. Similarly to the previous study the varying height of the tree canopy is stored in a raster file.

### 2.1.3 Material properties of surfaces in the UBL

Urban areas are characterised by complex morphologies comprised by both artificial and natural surfaces, with the former being the predominant one [Grimmond and Souch, 1994]. The artificial surfaces have a distinctive influence on the local atmosphere due to their mechanical, radiative, thermal [Back et al., 2021] and hydraulic properties [Grimmond and Souch, 1994]. In this section some of the properties of surfaces encountered in the urban environment are listed, along with a short description. The purpose of this section is to provide options of relevant to urban physics parameters, with the potential to be used as 3D model semantics. The potential parameters are listed as follows:

1. Radiative properties: Refer to the radiation emitted, transmitted or absorbed by the surface of a given material, given an incident wavelength and temperature [Oke, 1978].
  - Albedo ( $\alpha$ ): Determines the absorptivity of a surface. 0 corresponds to perfect absorption while 1 indicates a perfect reflector [Oke, 1978]. According to a study implemented by Back et al. [2021] for a city in Austria it was found that surfaces with high-albedo values can contribute to the reduction of the land surface temperature and at the same time have a negative impact for thermal comfort.
  - Emissivity ( $\epsilon$ ): Is the ratio of the radiation emitted by a material to that emitted by a black-body (perfect emitter) at the same temperature [Oke, 1978].
2. Thermal properties:
  - Thermal conductivity: used to measure the ability of a material to conduct heat [Oke, 1978].
  - Thermal admittance: quantifies the ability of a surface to absorb and release heat from/to space over time [Oke, 1978]. It is used to describe building materials, and it is usually used for studies of the indoor environment [Shaik and Babu, 2015].
3. Other descriptive characteristics:

## 2 Theoretical background and related work

- Colour: Darker coloured surface materials can contribute to the absorption of radiation [Back et al., 2021].

## 2.2 CFD simulations

CFD simulations provide a numerical solution to the selected set of, usually, partial-differential equations, according to the selected discretization scheme [Jayanti, 2018a]. The values of the flow variables are computed inside the defined boundaries of the computational domain, at discrete point locations. The determined locations are the product of the mesh generation process [Jayanti, 2018a].

The boundaries of the computational domain define its extents and are namely the inlet, outlet, sides, top and bottom. The wind enters the domain through the inlet and leaves through the outlet boundary. In urban CFD simulations for the outdoor environment, the buildings are represented explicitly with the use of a 3D model. The air around the model is decomposed into smaller units through the mesh generation. An example of a computational domain is shown in Figure 2.6.

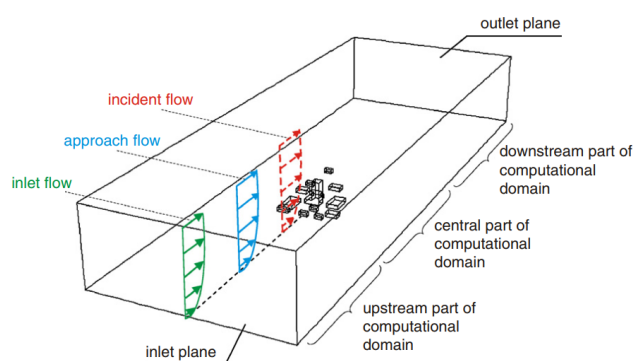


Figure 2.6: Example of computational domain with outlined boundaries [Blocken et al., 2007b].

CFD is widely used for the study of wind flows in the lower part of the in the UBL [Toja-Silva et al., 2018; Toparlar et al., 2017]. Other approaches include full-scale field measurements and wind tunnel testing [Lateb et al., 2016]. Compared to the latter two methods CFD is in most cases considered more efficient in terms of cost and time. Furthermore, field measurements and wind tunnel testing provide point measurements at a limited number of locations, while CFD generates full-scale flow approximations [Montazeri and Blocken, 2013]. In addition, CFD allows the possibility to test different scenarios [Toparlar et al., 2017]. Furthermore, field measurements are performed in real weather conditions and as such can not be replicated, while in reduced-scale modelling testing is done under controlled conditions, however, there are often limitations regarding the similarity of the scaled model [Lateb et al., 2016]. On the other hand, in CFD there is uncertainty regarding the accuracy of the results. The main sources of errors and uncertainties can be attributed to the assumptions and approximations made in the formulation of the selected mathematical models or they are of numerical natures [Franke and Baklanov, 2007]. For this reason, in CFD it is critical for researchers to verify and validate their results [Toparlar et al., 2017]. This is usually done through verification of the selected models, sensitivity analysis and comparison with experimental data [Lateb et al., 2016].

### 2.2.1 Mesh generation

The mesh generation process is a time-consuming process of fundamental importance to the verification and validation process of a CFD simulation [Blocken, 2015]. The added difficulty of the mesh generation does not solely depend on creating a mesh that is independent of the numerical solution but also on the fact that other factors need to be considered. For instance, the requirement of the first cells center to be at a height above the intended roughness length value [Blocken, 2015]. The mesh generation entails

the partition of the computational domain into a finite number of cells [Jayanti, 2018b]. The created grid can be 1D, 2D or 3D, structured or unstructured. A structured grid is comprised by topologically rectangular (2D) (Figure 2.7) or cubical (3D) cells of linear or curvilinear lines, while for an unstructured grid the cells are arbitrarily shaped [Jayanti, 2018b].

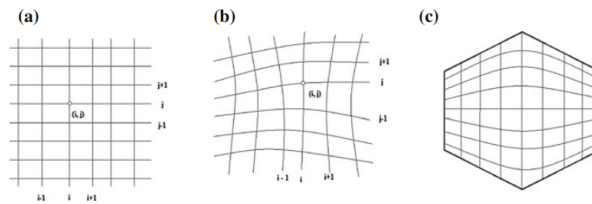


Figure 2.7: Examples of structured grids [Jayanti, 2018a].

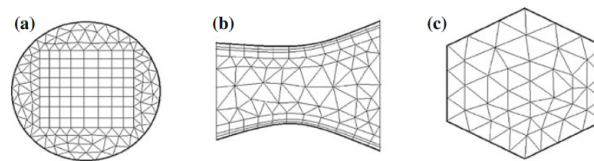


Figure 2.8: Examples of unstructured grids [Jayanti, 2018a].

Unstructured grids are required for the fitting of complex geometries since they can simulate with more detail the physical space than structured ones [Jayanti, 2018a]. Furthermore, the mesh can adapt better to implemented refinements. However, indexing and connectivities between the cells are not as straightforward as for the case of structured grids [Jayanti, 2018a].

### 2.2.2 Flow modelling

Given the complex turbulent nature of the ABL, analytical solutions to initial equations is only possible under restricted and limited conditions. Instead approximations are made, through averaging methods and simplifications. Some of the available numerical approaches/models for the solution of the discretised equations are listed below:

- **RANS**: Solves the Navier-Stokes equations following the Reynolds time-averaged approach. It provides information for the mean wind flow while turbulence is resolved according to a relevant turbulence model [Franke and Baklanov, 2007].
- **Large Eddy Simulation (LES)**: The large scale eddies are resolved while smaller scale turbulence is filtered and modelled separately [Toja-Silva et al., 2018]. In this case, the Navier-Stokes are averaged in space, and a turbulence model is required [Franke and Baklanov, 2007]. According to a review of Toparlar et al. [2017] LES simulations produced higher accuracy results, when compared to field measurements, than RANS simulations. However, it is a more time consuming and computationally expensive approach [Ricci et al., 2020].
- **Direct Numerical Simulations (DNS)**: This provides a direct solution to the selected flow fields. However, due to the very large scales that need to be resolved, it is currently performed only for low-Reynolds numbers [Franke and Baklanov, 2007].

Incompressible steady RANS have been used extensively for the study of urban flows [Blocken, 2015]. Averaging of the Navier-Stokes equations means decomposing the equations into a mean and a fluctuating part. The latter represents the turbulent part of the flow. The way that the turbulent part is modelled in order to arrive to a solution is referred to as the closure problem. Extensively used for the modelling of turbulence is the two equation standard  $k - \epsilon$  model ( $k$ : kinetic energy and  $\epsilon$ : dissipation rate) [Ricci et al., 2020]. It is considered a robust [Ricci et al., 2020], reduced computational cost technique. As such it was selected as for the simulation performed in the present study.

## 2.3 Coupling GIS and CFD simulations for the urban environment

The importance of the role of spatial data and GIS technologies has been acknowledged by many practitioners in the CFD community [Jie et al., 2014; Shouzhi et al., 2018; Wong et al., 2007]. However, there are still steps to be made for advancements in the coupling of the two fields [Deininger et al., 2020]. In this context, several efforts have been made to provide support for CFD through the integration of GIS tools and methodologies in their workflow from researches of both fields [Jie et al., 2014; Shouzhi et al., 2018; Deininger et al., 2020; Back et al., 2021; Wong et al., 2007].

Willenborg et al. [2016] proposed a CityGML-based approach for mesh generation along with the enhancement of 3D models with CFD related parameters. The implemented workflow included the generation of voxels using the PostGIS functionality of the 3DCityDB spatial database. In the same context Deininger et al. [2020] built an interactive Smart City platform with response time measurements along with an optimised workflow for the generation of ready to use models for CFD simulations. In the proposed workflow, CityGML, Digital Terrain Models, pointclouds, landuse and vegetation data were used as input of the preparation of the input 3D model.

Shouzhi et al. [2018] used spatial datasets and GIS operations to extract spatial patterns and classify the urban areas based on selected attributes like building height, and density, with the aim of generating an appropriate input for CFD simulations. The results were used for the improvement of ventilation in the city of Changchun. Grimmond and Souch [1994] proposed the creation of a georeferenced database that would include spatial data of different resolutions based on a GIS methodology for the identification of urban spatial parameters. The data would be used in along with meteorological parameters for the study of phenomena at the regional, neighborhood level as well as microscale phenomena. Based on the above literature review four are the identified areas for the coupling of the two principles: 1) generation of 3D models to be used as input for CFD simulations, 2) Mesh generation for CFD, 3) Creation of spatial databases, 4) Semantic 3D models for CFD.

## 2.4 Specification of nonuniform roughness length

In OpenFOAM the specification of the  $z_0$  is done in the nut (i.e. turbulent viscosity) dictionary file through the `nutkAtmRoughWallFunction` boundary condition. This is required for each patch belonging to the bottom boundary of the computational domain that needs to be represented with a roughness value. In `nutkAtmRoughWallFunction` class  $z_0$  is defined as a `scalarField`. This means that the application of more than one values for  $z_0$  would require a separate dictionary entry for each value of  $z_0$ , along with a specification of the geometry for each entry. However, this process is time consuming, especially for complex geometries, and requires the generation of multiple geometry files.

Segersson [2017], in a proposed tutorial guide, suggests the specification of multiple  $z_0$  values based on the `fvOptions` functionality of OpenFOAM. The `fvOptions` provides a framework which allows for the creation of new source terms, without changing the source code of the software. It is accessed and controlled through the `fvOptionsDict`, where the parameters are set. For his implementation he created a library containing new classes and functions, which were incorporated to OpenFOAM through the `fvOptions`. The  $z_0$  values are either specified through a separate raster file which is imported to OpenFOAM or specified manually for each patch. The patches need to be defined explicitly through the OpenFOAM utilities `topoSet` and `createPatch`. The assignment of the non-uniform roughness length values is done through the implemented `Raster` class. It is performed by transforming/translating the input raster file, where the roughness length values are stored, to fit the generated mesh, and allow for the assignment of the values. Both of the aforementioned implementations require the explicit specification of the patches' geometries, which can be a time consuming and cost inefficient process. The methodology proposed in this study, aims to overcome this issue by using a semantically enhanced surface model. This way the geometries can be combined in one file, which will correspond to one patch in the bottom boundary of the computational domain.

Finally, Azevedo [2013] created an application for the modelling of terrain roughness. The implementation is automated and allows the user to choose among three modelling options for  $z_0$ :

1. Uniform aerodynamic roughness length over all terrain surface,
2. Variable as a function of topography characteristic values. The function is user defined and models roughness as a ground height dependent variable. Ground height is calculated as the average elevation of the four points that define a face in the mesh grid generated by `blockMesh`.
3. Based on real site data in the format of  $xyz_0$ . The input file is manipulated to fit the ground patch of the computational domain. Then  $z_0$  is assigned as the average  $z_0$  of the four points of each face.

Both options 2 and 3 require the generation of a separate file `write_z0`. This file specifies the patches, i.e. the geometries of the ground surface where the roughness will be assigned to. As a final step in both cases the assigned  $z_0$  values are written in the `nut` file. For this options  $z_0$  is treated as a nonuniform `scalarField`. The option of writing the input roughness length values as a non-uniform field, was also incorporated in the proposed methodology (Chapter 3).

## 2.5 Octree data structure

An octree is an hierarchical data structure considered as an extension of quadtree for the representation of 3D space [Samet, 1990]. It is a popular method for the storage of data [Samet, 1990], collision detection, identifying spatial relations between objects, mesh generation, image processing [Madeira et al., 2011]. It is used in areas like robotics, computer graphics, cartography, gaming architectures [Samet, 1990] to name a few. It is built based on recursive decomposition of 3D space [Madeira et al., 2011] and its main components are:

- The root node in an octree, used to represent the entirety of the stored data,
- The leaf nodes, for which no further partition of space is required and,
- The internal nodes, which are non-leaf nodes has eight children or octants nodes [Samet, 1990].

Figure 2.9 shows an example of an octree graph and how it relates to the decomposition of 3D space.

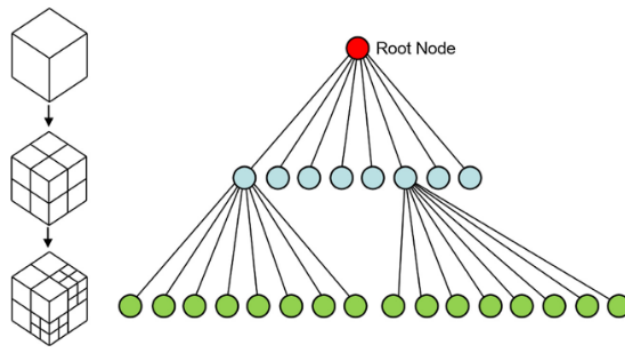


Figure 2.9: Octree of depth 2 visualisation [Su et al., 2016].

One of the advantages of an octree data structure is that it enables fast and efficient searches for identifying spatial relations [Madeira et al., 2011]. The application presented in this thesis is based on this advantage of the octree by using the hierarchical indexed octree implementation of `OpenFOAM`. Drawback of using this implementation is the dependency of the nearest neighbour search upon a user defined distance that is used as a threshold throughout the search.





# 3 Methodology

In this chapter the methodology followed for this thesis is presented. Firstly, a diagram of the workflow is provided. Thereafter the working steps are grouped and addressed in terms of application design, limitations and requirements of the application and the selected assessment methods. Further analysis and implementation details of the proposed methodology are presented in [Chapter 5](#).

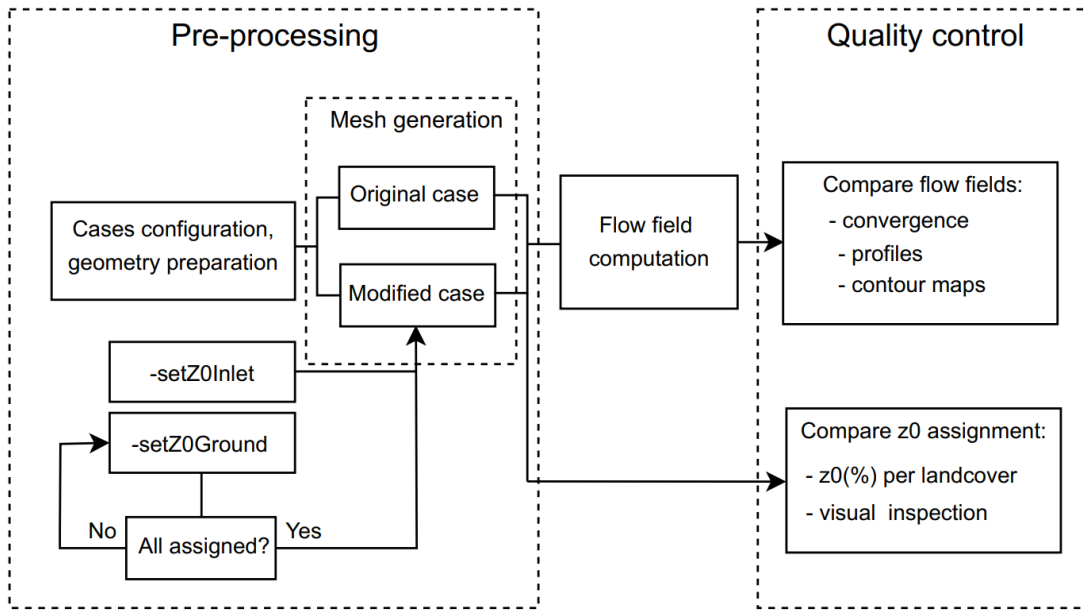


Figure 3.1: Overview of the methodology.

## 3.1 Application design

The purpose of the application is the assignment of roughness length values to the computational domain bottom boundary grid cells based on a ground surface model of varying roughness. The need for such an application was based on the process of assigning roughness values in the CFD software OpenFOAM. Roughness length ( $z_0$ ) is a user defined parameter and it is used to implicitly represent the roughness of a surface ([Chapter 2](#)). The urban environment is comprised of mixed and variant surfaces. Depending on the case, this can translate to a need for multiple surface geometries and each surface should be represented as a watertight model. In [Figure 3.2](#) the specification for roughness length is shown for the configuration file of turbulent  $\epsilon$ , for the case of multiple input geometries ([Figure 3.2a](#)) and for the case of a unified geometry ([Figure 3.2b](#)). The aim of the built application is to investigate the potential benefits of using a surface model that would represent the ground in one file. The use of a unified geometry file for the terrain implies also that any parameter (such as roughness length) corresponding to a particular surface should also correspond to that particular surface in the unified model. Based on this, the design of the application is driven by the need to create a framework with the potential of being further developed to incorporate other CFD parameters, with a similar requirement for correspondence to the input surface model. Additional to the main function of the application was also to investigate the possibility of applying a non-uniform roughness at the inlet of the computational domain (`-setZ0Inlet`), since it is currently not implemented.

```

boundaryField
{
  obj1
  {
    type      epsilonz0WallFunction;
    z0        $z0_obj1;
    value     $internalField;
  }

  obj2
  {
    type      epsilonz0WallFunction;
    z0        $z0_obj2;
    value     $internalField;
  }

  obj3
  {
    type      epsilonz0WallFunction;
    z0        $z0_obj3;
    value     $internalField;
  }

  obj4
  {
    type      epsilonz0WallFunction;
    z0        $z0_obj4;
    value     $internalField;
  }

  outlet
  {
    type      inletOutlet;
    inletValue uniform $turbulentEpsilon;
    value     $internalField;
  }

  inlet
  {
    type      atmBoundaryLayerInletEpsilon;
    #include  "include/ABLConditions";
    value     $internalField;
  }

  ground
  {
    type      zeroGradient;
  }
}

```

(a) Case for multiple input geometries.

```

boundaryField
{
  terrain
  {
    type      epsilonz0WallFunction;
    z0        $z0_terrain;
    value     $internalField;
  }

  outlet
  {
    type      inletOutlet;
    inletValue uniform $turbulentEpsilon;
    value     $internalField;
  }

  inlet
  {
    type      atmBoundaryLayerInletEpsilon;
    #include  "include/ABLConditions";
    value     $internalField;
  }

  ground
  {
    type      zeroGradient;
  }

  #include "include/sideAndTopPatches"
}

```

(b) Case for one input geometry.

Figure 3.2: Examples of the user defined dictionary file for  $\epsilon$ .

The decision to work with OpenFOAM to perform the simulations and by extent for building the application was based on two main criteria. Firstly, it is a free, open source software (as established in 2004) and secondly, it has been widely used for research and engineering from academic and commercial organisations. As a free and open source software it supports the use, reuse, modification and distribution of its code by anyone anywhere for all intended purposes<sup>1</sup> under the General Public License (GPL). The GNU GPL ensures that these permissions will be guaranteed in the distribution and redistribution of any implementation based on OpenFOAM<sup>2</sup> ("copyleft").

The main steps that were made in order to design the application are listed below:

- Selection of the input/output in format.
- Selection of the application environment, i.e. whether it would be implemented inside the environment of *OpenFOAM* or outside and later integrated.
- Selection of the code structure and main algorithm for the assignment of roughness length values to the bottom boundary grid cells.
- Identifying parameters that needed to be user defined.

### 3.1.1 Input/output

The input for the main functionalities of the applications is a terrain surface model and the face centers of the corresponding ground patch. The surface model is a file generated outside OpenFOAM, while the designated patch face centers are a product of the mesh generation process. The choice of the face centers was based on two main criteria. Firstly, it was in accordance with the nature of the roughness length parameter as surface characteristic and secondly, it simplified the assignment process. For the input surface model, the file format was dictated from the available compatible options that are offered by OpenFOAM. Specifically, OpenFOAM allows as input formats<sup>3</sup> the following:

- Stereolithography ASCII (*stl*)
- Stereolithography binary (*stlb*)
- Wavefront object (*obj*)
- Legacy Visualisation Toolkit (*vtk*)
- OpenFOAM triangulated format (*ptr*)
- Invis AC3D (*ac*)
- TenGen surface mesh format (*smesh*)
- Triangle format (*tri*)
- OpenDX format (*dx*)

From the above listed options the *stl* and *obj* formats were considered as input for the surface geometry. They both are widely used open formats allowing for easy exchange between programs. Additionally to the input geometry format, the way the roughness values would be retrieved needed to be specified. The choice of roughness values can be based on visual inspection of landcover data. In most cases these data are available through aerial images [Kent et al., 2017]. Other cases, such as for the Netherlands, they can be derived from available landuse datasets (Basisregistratie Grootchalige Topografie (BGT), Basisregistraties Adressen en Gebouwen (BAG)). Although, these options would be useful additions for further developing the application, for both aforementioned cases, combining the geometry with the added semantics requires intermediate operations, and possible transformations to the input geometry, so that each surface type is accurately represented. The current implementation considered that the input surfaces were already accurately represented so the focus was on building a workflow for the

<sup>1</sup><http://www.gnu.org/philosophy/free-sw.html>

<sup>2</sup><http://www.gnu.org/licenses/gpl-3.0.html>

<sup>3</sup><https://www.openfoam.com/documentation/guides/latest/doc/guide-meshing-snappyhexmesh-geometry.html>

assignment of the roughness length parameter. Based on the above, the idea was to combine the geometries into one file and then find a way to use the roughness values as added semantics to the unified surface model. Between the two accepted formats, currently, there is no available option for `stl` files to carry additional information. On the other hand, `obj` files with the use of Wavefront Material Template Library (`mtl`) files allow for adding semantics to the triangulation. Usually `mtl` files are used for the assignment of metadata regarding the visualisation of the file. However, in our case, instead of colour and illumination information the values of roughness length were used. An example, of the correspondence between an `obj` and an `mtl` file is shown in Figure 3.3.

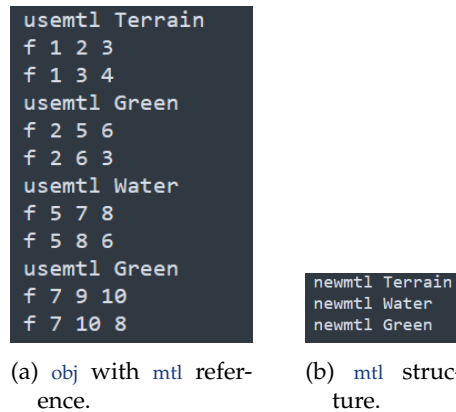


Figure 3.3: Example of combining `obj` and `mtl` file.

The decision regarding the output of the application was based on roughness length being a user defined parameter. As part of the wallFunction modelling roughness length had to be written to a dictionary file in the folder where all the boundary conditions and initial values are located. For the existing process of assigning the roughness, one value per surface type is assigned. However, for our case multiple values for roughness had to be specified for one surface. For this reason, the output of the application generates a list of  $z_0$  values and writes it to the appointed by the user location inside the boundary condition as a nonuniform `List<scalar>`, which is an `OpenFOAM` defined type. Every entry on the list corresponds to a cell in the bottom boundary of the computational domain, and is based on the indexing given to every cell after the mesh generation process is complete. For this reason, the sequence of the entries in the list is unique for every case and dependent on the generated mesh. This is due to complex connectivities in unstructured grids and non-uniformly refined grids, which are essential for `CFD` simulations (Chapter 2). The generation of the mesh is considered to be done using the `OpenFOAM` meshing tools: `blockMesh` for the background mesh and `snappyHexMesh` for the surface refinement and snapping process.

#### 3.1.2 Implementation environment

For the implementation environment of the application two options were considered: 1) a Python implementation (can be found in [https://github.com/cfratz/set\\_nonUniform\\_z0](https://github.com/cfratz/set_nonUniform_z0)), based on an `r-tree` data structure was implemented, using external libraries and 2) a `C++` implementation, using the built-in functions and classes of `OpenFOAM`. Determinant factors in the selection was the possibility for further development and the building of an application that would be user-friendly. The dependence of the assignment process on the generated mesh posed a hurdle in the implementation of the Python application, since mesh data still needed to be exported from `OpenFOAM` and vice versa. This would have meant that in case of multiple parameters (other than roughness length), this process would have to be repeated, adding a strain to the use of the application. Although both options were implemented, the `OpenFOAM C++` implementation was selected to facilitate further development, as well as, the overall use of the application. On the other hand, a difficulty of the chosen implementation was that the source code of `OpenFOAM` is highly templated<sup>4</sup>. This made the assessment of the overall performance of the

<sup>4</sup><https://www.cplusplus.com/doc/oldtutorial/templates/>

application a complicated task, since it required an in-depth understanding of the classes and function calls that were running in the background.

### 3.1.3 Code structure

The main body of the application is built around a list of available options, which are specified through user commands on the Linux shell. It is based on `foamDictionary`<sup>5</sup>, an application of OpenFOAM for the manipulation of dictionaries. This allowed for accessing the user input specifications and also writing the output to the user-specified dictionary file in the boundary conditions. The current available options are listed below:

- `entry`: name of entry in the specified file location.
- `setZ0Ground <patch name>`: assigns roughness length to the specified patch.
- `setZ0Inlet <patch name>`: assigns roughness length to the inlet of the domain.
- `writeZ0`: writes out to a `txt` file the list of roughness length values.
- `writeCoords`: writes out to an `xyz` file the coordinates of the cell centers of the designated for assignment computational domain boundary.
- `exportToVtk`: writes out to a `vtk` file the designated patch mesh information along with their corresponding roughness length as an attribute.
- `setZ0NoGeom <z0 value>`: assigns roughness length, based on the input geometry, to a patch for which the specified geometry is not included in the mesh generation. In this case, the patch is generated with the `blockMesh` utility. The specified `<z0 value >` is taken into account when no correspondence is found between the input geometry and the patch face centers. For the case that the input geometry covers the entire extent of the patch the `<z0 value>` should be specified to 0.
- `setParams`: assigns all related to turbulence model variables that are dependent on the roughness length by assigning the corresponding values to all relevant parts of the mesh.
- `help`: get the list of available options with their description.

Options `-writeZ0`, `-writeCoords` and `exportToVtk` can be specified if one of the `-setZ0Ground` or `-setZ0Inlet` have been selected first. Option `exportToVtk` can be also specified for option `setParams`, for the case of which the corresponding inlet turbulence parameters are written to `vtk`. Pre-requisites for the shell specification is the name of the application, the file location and the name of the roughness length parameter, which may or may not exist as an entry in the specified dictionary file. The dependence of the application on the generated mesh restricts the application to be called within the case study folder. An example of a shell command is shown in [Figure 3.4](#).

```
medium_v0$ setZ0 0/include/ABLConditions -entry z0 -setZ0Ground terrain
```

Figure 3.4: Shell specification for setting the roughness length parameter with option `-setZ0Ground`.

After the application is called there are three main steps that are followed for both `-setZ0Ground` and `-setZ0Inlet` options. Firstly, the correspondence between the input geometry file (`obj`) and the file with the added semantics (`mt1`) is checked. Secondly, based on the `mt1` information included in the `obj` file each triangle is assigned a roughness value and lastly an instance of the mesh is created. The instance corresponds to the last available time step of the mesh generation process. The latter is done in order to allow access to the mesh data.

<sup>5</sup>[https://cpp.openfoam.org/v7/foamDictionary\\_8C.html](https://cpp.openfoam.org/v7/foamDictionary_8C.html)

### 3.1.4 Option -setZ0Ground

The process of the assignment of roughness length is resolved based on an octree radius search (Chapter 2). The main data needed for the search are the face centers of the specified boundary patch, a surface based octree data structure and a search radius. The radius search on the octree is done iteratively over the list of face centers; when an overlap is found between the tested face center and a triangle, the corresponding to the triangle roughness value is written to the list of  $z0$  values. The algorithm described above is shown in pseudocode Algorithm 3.1. The algorithm is implemented with the use of OpenFOAM built-in classes and functions (Chapter 5). The input  $z0val$  (line 11 in Algorithm 3.1) refers to the value that is specified by the user when option `-setZ0NoGeom` is selected. If the option is not selected then  $z0val$  is set to 0.

---

**Algorithm 3.1:** ROUGHNESS LENGTH ASSIGNMENT ( $Cf, tree, triZ0, nearDist, z0val$ )
 

---

**Input:** A patch face centers  $Cf$ , surface based octree  $tree$ , triangles' roughness length values  $triZ0$ , a distance used in the search  $nearDist$ , a roughness length value  $z0val$   
**Output:** Face centers' assigned roughness length values  $z0$ , number of unassigned face centers  $nMissed$

```

1  $nMissed \leftarrow 0$  ;
2  $nearDistSqr \leftarrow 0.25 * magSqr(nearDist, nearDist, nearDist)$  ;
3  $hitPoints \leftarrow$  get nearest point in  $tree$  for all  $Cf$  using  $nearDistSqr$  ;
4 for  $facei \in Cf$  do
5    $pt \leftarrow facei$  ;
6   if a hit is found and  $mag(hitPoints_{facei} - pt) < nearDistSqr$  then
7      $triIndex \leftarrow$  get the index of the overlapping triangle, for which  $pt$  is inside ;
8      $z0_{facei} \leftarrow triZ0_{triIndex}$  ;
9   else
10    if  $z0val \neq 0$  then
11       $z0_{facei} \leftarrow z0val$  ;
12    else
13       $nMissed \leftarrow$  increment by 1 ;
14 return  $nMissed$  ;

```

---

The search, as mentioned above, is based on a user-defined seed distance. The value of the distance that is actually used in the search is given by<sup>6</sup>:

$$nearDistSqr = 0.25 * (nearDist^2 + nearDist^2 + nearDist^2) \quad (3.1)$$

where  $nearDist$  is the user-defined scalar value for the seed distance. A suggested value for the input distance is 0.5, however indicative. As such, it might be the case that the user is required to test several values until all face centers are assigned. The user is notified regarding the number of missed points from the on screen output after running the application. If the number is not 0 then the user is required to change the value of the seed distance. For the case of a flat terrain, the tested cases showed that once an appropriate value is found, then any increase in the seed distance value will not affect the result, neither in terms of the roughness assignment nor in terms of number of missed points. Nevertheless, further testing is needed, both for cases of flat surfaces and for cases of variant height. The plot in Figure 3.5 shows the relation between different values for the seed distance and the number of missed points for the 'complex' case, used in the testing of the application. The result is also illustrated in Figures 3.6a and 3.6b for a distance of 0.1 m and 1,000,000 m respectively.

The search in the octree is implemented by `findNearest`, an OpenFOAM function, part of the `indexedOctree` class. The input octree<sup>7</sup> is built with the restrictions that each tree leaf can not contain more than 10

<sup>6</sup>[https://cpp.openfoam.org/v7/triSurfaceSearch\\_8C\\_source.html](https://cpp.openfoam.org/v7/triSurfaceSearch_8C_source.html) at line 311

<sup>7</sup>[https://cpp.openfoam.org/v7/triSurfaceSearch/\\_8C/\\_source.html#l100198](https://cpp.openfoam.org/v7/triSurfaceSearch/_8C/_source.html#l100198) at lines 232-236

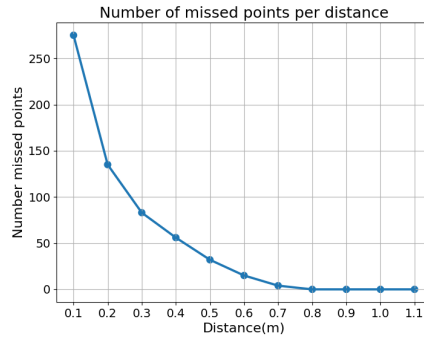


Figure 3.5: Plot of the number of missed points for different seed distances for a case of flat terrain.

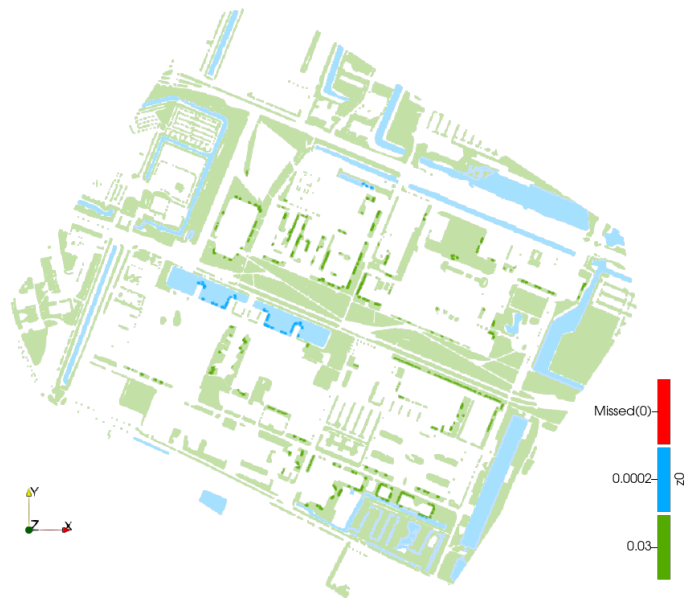
shapes with each triangle allowed to be referenced by maximum three leaves. The latter restriction is applied to avoid duplicate references. The processes that are performed in the background when the `findNearest` function is called are shown in pseudocode in [Algorithm 3.2](#). Starting from the topnode (i.e. node with index 0) of the tree, its sub-nodes are ordered with the first octant being the one containing the specified face center. Then the ordered octants are tested iteratively. If an octant is an internal node and its bounding box overlaps the input face center then function `findNearest` is called again, but this time starting from the found overlapping internal node. This process continues until an octant that is a tree leaf is found. If the octant's bounding box overlaps the face center then the triangles contained in the tested leaf are checked iteratively. The test is successful if the checked triangle contains the input face center and their squared distance (`distSqr`) is smaller than `nearDistSqr` (line 19 in [Algorithm 3.2](#)). If the latter is true then the `nearDistSqr` is updated with the value of `distSqr`. This ensures that only if a triangle is found to be closer than the lastly updated `nearDistSqr` will pass the test. This can also explain why, at least for a flat terrain, after a certain value is found any increase in the seed distance does not alter the resulting assignment of the roughness values.

In order to calculate the distance between the input face center and the tested triangle OpenFOAM firstly identifies the location of the face center in relation to the triangle. Although this part is not incorporated in the pseudocode of [Algorithm 3.2](#), it is important to provide a description of the processes involved in order to better understand and assess the results of the assignment process. The OpenFOAM function called for this, is `nearestPointClassify` and it classifies the orthogonal projection of the input sample point (in our case a face center) based on the Voronoi feature regions of the tested triangle. The output is the orthogonal projection point, a boolean value of the result and the type of the region. The investigated regions are the triangle's vertices, edges and face. The algorithm is adapted from [[Ericson, 2004](#)] (p. 136-142). As mentioned above, only if the point is found inside the triangle's face region the test is considered successful. For the case of a flat terrain this means that the orthogonal projection is the point itself, which is also validated from the results of the application testing.

[Algorithm 3.3](#) demonstrates in pseudocode the function called in lines 10 and 15 of [Algorithm 3.2](#). It is used to test if there is an intersection between a sphere around the input point and the bounding box of the specified node. The square root of `nearDistSqr` is the radius of the sphere. In the algorithm the distance between an octant's bounding box corner points and the input face center is calculated for each direction (i.e.  $x$ ,  $y$  and  $z$ ) and checked against the provided `nearDistSqr`. An octant is filtered out if the sum of the squared distances is outside or intersects the sphere (line 10 in [Algorithm 3.3](#)). Based on the comparison needed for the filtering of octants the choice of a squared vectorised distance instead of the input seed distance can also be justified. The testing is limited to the algorithm as shown in pseudocode in [Algorithm 3.1](#).



(a) Seed distance: 0.1 m



(b) Seed distance: 1,000,000 m

Figure 3.6: Screenshots of the 'complex' case vtk files illustrating the roughness assignment for a seed distance of 0.1 (3.6a) and 1 million (3.6b). Missed points are in red colour.



**Algorithm 3.2: FIND NEAREST** ( $Cfi, tree, startDistSqr$ )

**Input:** A patch face center  $Cfi$ , surface based octree  $tree$ , a start distance used in the search  $startDistSqr$

**Output:** boolean variable  $hit$ , the triangle index for which  $Cfi$  is inside  $triIndex$ , the nearest found point  $hitPoint$

```

1  $hitPoint \leftarrow (0,0,0)$ ;
2  $nearDistSqr \leftarrow startDistSqr$ ;
3  $triIndex \leftarrow -1$ ;
4  $nodeI \leftarrow 0$ , 0 is the topNode of the  $tree$ ;
5  $searchOrder \leftarrow$  sort octants of  $nodeI$  with first being the one containing the  $Cfi$ ;
6 for  $i \leftarrow 0$  to 8 do
7    $octant \leftarrow searchOrder_i$ ;
8   if  $octant$  is a tree node then
9      $subNodeBbox \leftarrow$  get the  $octant$  bounding box;
10    if  $subNodeBbox$  overlaps  $Cfi$  then
11       $nodeI \leftarrow$  get index of internal node that represents  $octant$ ;
12      go back to the beginning of the algorithm and start with updated  $nodeI$  until a  $nodeI$ 
        is the last node of a branch;
13    else if  $octant$  is a tree leaf then
14       $LeafBbox \leftarrow$  get  $octant$  bounding box;
15      if  $LeafBbox$  overlaps  $Cfi$  then
16        for  $triangle \in octant$  do
17           $distSqr \leftarrow$  get distance of nearest point of triangle to  $Cfi$ ;
18           $hit \leftarrow$  True or False;
19          if  $distSqr < nearDistSqr$  then
20             $hitPoint \leftarrow$  get nearest Point coordinates;
21             $nearDistSqr \leftarrow distSqr$ ;
22             $triIndex \leftarrow$  get triangle index;

```

**Algorithm 3.3: OVERLAPS** ( $Bbox, Cfi, startDistSqr$ )

**Input:** A patch face center  $Cfi$ , bounding box of an internal node or leaf  $Bbox$ , a distance used as radius around input point  $nearDistSqr$

**Output:** Boolean value

```

1  $distSqr \leftarrow 0$ ;
2 for  $dir \leftarrow 0$  to 3 do
3    $d0 \leftarrow Bbox_{min_{dir}} - Cfi_{dir}$ ;
4    $d1 \leftarrow Bbox_{max_{dir}} - Cfi_{dir}$ ;
5   if  $d0 > 0 \neq d1 > 0$  then
6     //  $Cfi$  inside both extrema;
7   else if  $mag(d0) < mag(d1)$  then
8      $distSqr \leftarrow +sqr(d0)$ ;
9   else
10     $distSqr \leftarrow +sqr(d1)$ ;
11  if  $distSqr > nearDistSqr$  then
12    return False;
13 return True;

```

### 3.1.5 Option `-setZ0Inlet`

Option `-setZ0Inlet` follows a different process for the assignment of roughness length to the inlet. This differentiation is due to the underlying meaning of the roughness length. For the case of the inlet the roughness value represents the roughness elements that influence the inflow but are located outside of the computational domain. Additionally, it is assumed that the same landuses neighbouring, the inlet, inside the domain are the same as the ones neighbouring the inlet on the outside of the domain. The experiments showed that this option can be implemented, but, depending on the case, not without uncertainty. The reason for this, are the limitations related to the assignment process of values in non-uniformly refined grids. Specifically, different levels of refinement do not allow for a uniformly treated assignment. An example of such a case is provided in [Figure 3.7](#).

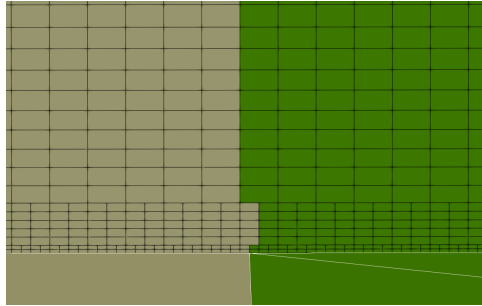


Figure 3.7: Example of misassigned roughness values for an inlet boundary. The different colours indicate two different roughness length values.

A cell that is split in smaller other cells meets neighbour cells with a different level of refinement, lower or higher, and thus imposing the need to make a choice. This implies uncertainty in the creation of a precise representation. However, the accuracy of the simulation results might not be affected even if the assignment process is not precise. This matter will be further addressed later on. For the assignment of the roughness length values to the inlet patch the first step of the process includes finding the shared cells between the inlet patch and the neighbouring ground patches. When a shared cell is found then the ground cell's roughness is assigned to the corresponding inlet cell. This implies that the roughness length should be configured for all ground patches before the `-setZ0Inlet` option is called. The process is concluded when all inlet's first row cells are assigned a value. The second step is to identify changes in the roughness length values and mark them as checkpoints ([Algorithm 3.4](#)). These changes are represented by the corresponding inlet first row faces' indices, although they need to be sorted first according to their span wise coordinate (line 2 in [Algorithm 3.4](#)). This is required because the indexing of the patches' face centers does not follow a spatial sequence. This means that a cell with an index placing it at the end of the first row can actually be located at the beginning of the row. This happens as a result of the refinement process. The spanwise coordinate refers to the direction perpendicular to the inflow (streamwise) direction. It is retrieved through the user defined parameter for the flow direction.

In [Algorithm 3.5](#) the assignment of coordinates to every checkpoint is shown. Although, as for option `-setZ0Ground`, the assignment is based on the face centers of the boundary patches, for the checkpoints' coordinates the maximum and minimum span wise coordinates of the faces' boundaries are used. This is done to create thresholds between the checkpoint coordinates that will minimize the errors in the assignment process. These errors refer, as mentioned above, to the limitations created by the levels of refinement and encountered for face centers with spanwise coordinate close to the thresholds' limits.

The final step is the assignment of roughness length to the rest of the inlet faces based on a comparison of the spanwise coordinate of face centers with the checkpoints' coordinates ([Algorithm 3.6](#)). Lines 6 and 11 in [Algorithm 3.5](#) and [Algorithm 3.6](#) respectively, refer to a case where the first row's last face is also a checkpoint. Option `-setParams` is complementary to option `-setZ0Inlet`. It allows the calculation and assignment of initial values for the turbulence parameters (e.g. kinetic turbulent energy, turbulent kinetic  $\epsilon$ ) to the inlet. The option can only be called if option `-setZ0Inlet` is specified first. It is needed because these parameters are dependent on the values of roughness length at the inlet and due to the fact that every patch required specified treatment.

**Algorithm 3.4:** CHECKPOINTS (*firstRow*, *spanDir*, *z0*)

**Input:** An inlet first row faces *firstRow*, spanwise component face centers *Cfspan*, roughness length values *z0*

**Output:** First row *z0* identified changes as inlet's faces *checkPoints*

```

1 checkPoints0 ← 0 ;
2 firstRow sort ascending according to Cfspan ;
3 for facei ∈ firstRow do
4   if facei ≠ 0 then
5     previous ← get last checkPoint ;
6     if z0previous ≠ z0facei then
7       append facei to checkPoints ;
8 if checkPoints > 1 and last checkPoint ≠ last face of firstRow then
9   append facei to checkPoints ;

```

**Algorithm 3.5:** CHECKPOINTS COORDS (*checkPoints*, *mesh*, *z0*)

**Input:** *checkPoints*, inlet's mesh *mesh*, roughness length values *z0*

**Output:** *checkPoints* coordinates *checkCoords*

```

1 for facei ∈ checkPoints do
2   access mesh and get facei nodes' spanCoords ;
3   maxCoord ← max spanCoords ;
4   minCoord ← min spanCoords ;
5   if facei = last face of checkPoints then
6     if z0facei ≠ z0lastfacei then
7       append minCoord to checkCoords ;
8       append maxCoord to checkCoords ;
9     else
10      append maxCoord to checkCoords
11 else
12   append minCoord to checkCoords

```

**Algorithm 3.6:** PARAMETER ASSIGNMENT (*checkPoints*, *checkCoords*, *CfCoords*, *z0*)

**Input:** *checkPoints*, checkpoints' coordinates *checkCoords*, spanwise component of a patch face centers *Cfspan*, roughness length values *z0*

**Output:** patch parameter *patchParam*

```

1 s ← get number of checkPoints ;
2 s1 ← get number of checkCoords ;
3 for facei ∈ patchFaces do
4   coordi ← Cfspanfacei ;
5   for i ← 1 to s do
6     previousFacei ← checkPointsi-1 ;
7     previousCoord ← checkCoordsi-1 ;
8     nextCoord ← checkCoordsi ;
9     if coordi > previousCoord and coordi ≤ nextCoord then
10      append patchParamfacei ← z0previousFacei ;
11 if facei = last patch facei and s < s1 then
12   append patchParamfacei ← get last checkPoint z0 ;

```

### 3.1.6 User-defined parameters

As mentioned in [Section 3.1.3](#), upon calling the application the user needs to specify the location where the roughness length list will be written, the name of the roughness length entry that will be added to the designated configuration file and the name of the patch for which the assignment will be implemented. Additionally, to those information, the application requires some extra specifications. These specifications already exist in the case configuration files, however scattered. For the purpose of facilitating the implementation, an additional configuration file is required for running the application. The file follows the dictionary structure, similar to most of the configuration files in OpenFOAM and is located in the case's constant folder. The included required parameters are listed below:

Required for both options `-setZ0Ground` and `-setZ0Inlet`:

- `inputFile`: the surface geometry file(`obj`) that corresponds to the specified patch. It is required that the file is the same as the one used during the mesh generation with `snappyHexMesh`.
- `inputMtl`: the `mtl` file that carries the additional semantics of the input surface model.
- `z0_values`: the landcover categories with the corresponding roughness length values. This list is required because the `mtl` file carries only the landover names, the `z0` values are not included in the semantics file.

Required only for option `-setZ0Ground`:

- `nearDist`: the seed distance used in for the search in the surface based octree.

Required only for option `-setZ0Inlet`:

- `flowDir`: It is the direction of the inflow, and it can be symbolised with one of the following ways: (1 0 0): x direction, (0 1 0): y direction and (0 0 1): z direction. It is used for checking if the specified by the user patch is the inlet, and to identify the span wise coordinates.

Required only for option `-setParams`:

- `Params_Inlet`: the parameters needed for the calculation of initial values required by the input turbulence model. The current implementation includes the calculations based on the  $k-\epsilon$  model as adapted for roughness length.

The structure and essential additional user-defined parameters are displayed in [Figure 3.8](#).

```

FoamFile
{
  version      2.0;
  format       ascii;
  class        dictionary;
  location     "constant";
  object       setZ0;
}
// ***** //

inputFile      "constant/triSurface/terrain4nonUniformInlet_translated.obj" ;
inputMtl       "constant/triSurface/terrain.mtl";
flowDir        (1 0 0);
nearDist       0.8;

z0_values
{
  Terrain       0.05;
  Water         0.0002;
  Green         0.03;
}

Params_Inlet
{
  Uref          10;
  Zref          20;
  Cmu           0.09;
  kappa         0.41;
}

// ***** //

```

Figure 3.8: Configuration file for user-defined parameters.

## 3.2 Limitations and Requirements

The main limitations and requirements are derived based on three factors: a) the chosen method for the assignment process, b) the nature of the simulation's grid and c) the chosen models for the simulation of the flow. The limitations and requirements are further explained below based on selected options:

1. `-setZ0Ground`: For this option the radius search in an octree is employed. For this method the triangulated surface model is stored in an octree data structure, which is used to find overlaps between the triangles and the face centers of the user specified ground patch. In order for the method to produce valid results it is required that the input geometry is a watertight model with no overlaps or gaps. Examples of a case where this requirement is evident are shown in [Figure 3.9](#). Furthermore, the surface model should have no duplicate vertices, which is an additional geometry requirement, without which the construction of the octree, as implemented by OpenFOAM, is not possible. In the current implementation there is no overlap check or treatment. Lastly, limitation of this option is the need to test different values for the input search distance until no face centers are left unassigned.
2. Option `-setZ0NoGeom`: This option can only be applied after option `-setZ0Ground` is firstly specified since they both include the same assignment process. The difference between the two options is that for option `-setZ0NoGeom`, the input surface model covers only part of the specified ground patch extent. This implies that the mesh faces are not generated based on the input surface model, and as a result, it can be the case where faces are intersected by the input surface geometries, in contrast to option `-setZ0Ground` where it is ensured that there is no intersection. For cases where intersections occur, there is loss of precision in the assignment process. Depending on the case this could also mean loss of accuracy in the simulation results. Similarly to option `-setZ0Ground` it might be the case that the input search distance needs refining until a roughness value is assigned to all face centers.

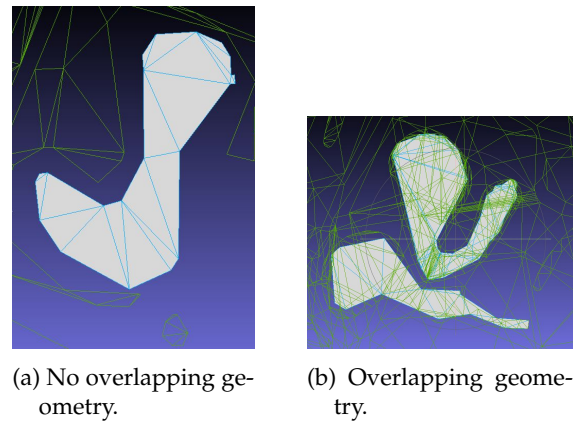


Figure 3.9: Comparison between a triangulated surface without overlaps (a) and a surface with overlapping triangles (b) (surface: grey, triangles of surface: blue, overlapping surface triangles: green).

3. Option `-setZ0Inlet`: The assignment of roughness length is based on identified changes of roughness length values of the inlet patch, in the span wise direction. The changes are identified based on the roughness value of the inlet's first cells' face centers. For this, it is required that all roughness values for the ground patches should already be specified. As mentioned in the previous sections and displayed in [Figure 3.7](#), the process generates uncertain results. This is a limitation imposed by the generated grid, and in order to overcome it, it would require either an adjusted mesh generation process or alterations to surface geometry to match the generated grid. Due to this limitation the `-setZ0Inlet` is tested only on an ideal case.
4. Option `-setParams`: It is implemented for the assignment of calculated parameters related to the selected turbulence model by employing the same assignment process as option `-setZ0Inlet`. In particular, the application is implemented for the  $k-\epsilon$  turbulence model as adapted for roughness length ([Section 4.2.1](#) in [Chapter 4](#)). This limits the use of option `-setParams` for specific flow conditions. Additionally, the results of this option carry similar uncertainty for the same reason as previously mentioned for option `-setZ0Inlet`. The names of the generated variables for the inlet patch are written to file as "parameter name" + "name".

For option `-setParams` the need for generating values for multiple parameters in combination with specifying only one file location upon calling the application restricts the writing out of these parameters to the specified location. This formulates also the way the user will organise the configuration files in the 0 folder. In particular, since all parameters are written out in one dictionary file (user defined location), this file is required to be included in all relevant files, and the overlapping entry values referenced. This is done by using the symbol "\$" and the variable name.

### 3.3 Quality control

The assessment of the application focused on the two main options that dealt with the assignment of non-uniform roughness length. The first step was the selection of the cases that the application would be applied and later evaluated. The testing was intended mainly for options `-setZ0Ground` and `-setZ0Inlet`. For this purpose, as mentioned in the previous section, two cases were selected, a simple ideal case and a more complex one that corresponds to a real location ([Chapter 4](#)). The simple case was created for preliminary testing of the application and the more complex one to allow for further evaluation. The choice was made based on the level of complexity of their underlying geometries. For the assessment of option `-setZ0Ground` for each of these cases two simulations were performed and compared. One simulation with the original geometry files that represented the different landcovers with separate files, which was used as reference, and one with a unified landcover geometry that con-

tained all landcover categories. Similar cases but with slightly modified geometries were used for the assessment of option `-setZ0Inlet`. However, for this option there were no reference data to compare.

The main assessment approach, for option `-setZ0Ground`, included comparing the roughness length values and the simulation results of the original case (separate geometries) and the case for which the application was applied (unified geometry). For this the following factors were examined:

1. The divergence in the simulation results. The desired outcome would be for the two cases to generate results with negligible differences. As indicator to determine the similarity of the flow, the stream wise and vertical velocities values at selected points were used. The accuracy of the assessment depended on the configuration of the point locations. Two were the main criteria used to determine the point locations: selection of sparse enough locations to ensure that the study area was covered to its extents and diversity of the locations in terms of the underlying surface type as well as proximity to buildings. In addition, difference maps were generated between the two cases in terms of the computed  $U_{magnitude}$ , as well as contour maps. This was done for a better inspection of the effect of the assigned roughness length values on the simulation results.
2. The number/percentage of cells per landuse. For this the number of cells per patch was considered, as produced by the mesh generation process, in combination with the number of assigned roughness values per landuse. This comparison was made under a certain level of uncertainty since the number of cells per landuse is also dependent on the mesh generation process. Nevertheless, the differences between the geometries of the two cases were minor, so the generated mesh of the modified case was expected to have negligible deviations from the original case.
3. The effect of using a simplified geometry in the mesh generation process. Although the changes to the geometry used in the modified case were minor, it was considered important to take into account their effect in the mesh generation process. This was examined through a comparative analysis between the two cases in terms of the number of cells per refinement level, the number of faces per wall patch, the max skewness<sup>8</sup>, the number of skew faces and the computation time required for the completion of the mesh generation process.

For both options `-setZ0Ground` and `-setZ0Inlet` the assignment of the roughness length values was evaluated through visual inspection of the generated result with the use of a `vtk` files. This was employed for the former option with the purpose of identifying large scale differences between the two cases, while for the latter it was used as the main assessment method of the assignment result. Finally, for option `-setZ0Inlet` the simulation results were compared against cases for which the option was not applied to verify that the proposed methodology influenced the calculation of the flow parameters. This is done based on the comparison at selected probe locations.

---

<sup>8</sup><https://openfoamwiki.net/index.php/CheckMesh>





## 4 Data, case pre-processing and testing

Two cases were selected for the assessment of the developed methodology. The underlying geometries were modified to allow for the main functionality of the application to be tested. In this chapter details regarding the surface models used and the set up of all used cases are presented. Specifically, in [Section 4.1](#) the surface models used for the original (using separate geometries) cases simple and complex are shown, while in [Section 4.2](#) details regarding the configuration of all cases are presented and explained. In [Section 4.3](#) the testing configuration of the cases, as presented in the aforementioned sections, is presented.

### 4.1 Datasets

The original cases were used as reference for the assessment of the proposed methodology results. The test cases were built based on modifications of the original cases' geometries. The simple case is an ideal case and was generated in the context of this study in Python, whereas the complex case corresponds to a real location, and were used in the published research of [García-Sánchez et al. \[2021\]](#). The complex case depicts part of the TU Delft Campus ([Figure 4.2](#)). The simple case, using the separate geometries as described in this section will be referred to as *s.0*, while the complex case as *c.0*. [Table 4.1](#) shows the details of the aforementioned geometries.

Case	File	Faces	Vertices
s.0	terrain.obj	2	4
	green1.obj	2	4
	water.obj	2	4
	green2.obj	2	4
c.0	green.obj	24468	20050
	water.obj	9495	6127
	buildings.stl	42360	21323

Table 4.1: Details of the triangulated terrain models used in the original cases.

The surface model for the *c.0* case was initially in *stl* format, and later the geometries corresponding to the green and water surfaces were transformed to *obj*, in order to accommodate the requirements of the proposed methodology. The *stl* files were generated with *3dfier*, a 3D reconstruction software, that generates semantically enhanced 3D models [[Ledoux et al., 2021](#)]. The geometries were reconstructed based on a 2D topographic dataset, containing the footprints of the geometries and a point cloud from which the height information were extracted. The surface models used in the simulations of the *s.0* and *c.0* cases are shown in [Figures 4.1](#) and [4.3](#).

4 Data, case pre-processing and testing



Figure 4.1: s.0 case surface model ('brown': terrain, 'green': vegetation, and 'blue': water).

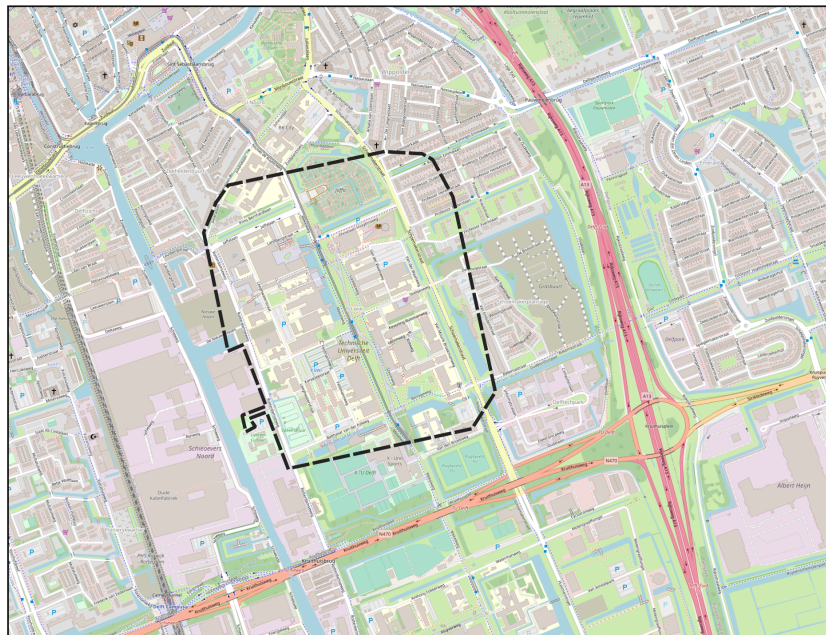


Figure 4.2: Study area for c.0 case in the TU Delft Campus.

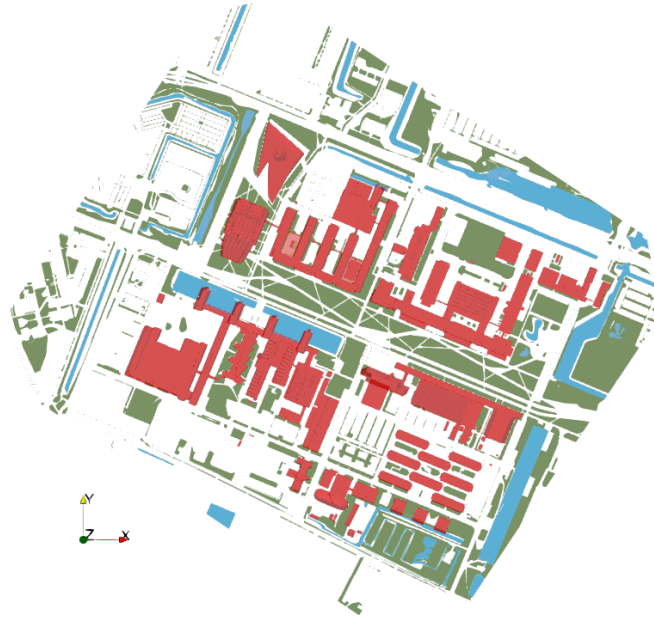


Figure 4.3: c.0 case surface model ('red': buildings, 'green': vegetation, and 'blue': water).

As demonstrated, the geometry of the complex case is comprised by a much larger number of triangles and the surfaces of the different landcover categories are blended together irregularly, whereas for the simple case the surfaces' geometry configuration is regular and comprised of a small number of large triangles. It should be noted that for the complex case the geometry contains areas where the landcovers have overlaps. The found overlapping areas are shown in Figure 4.4.

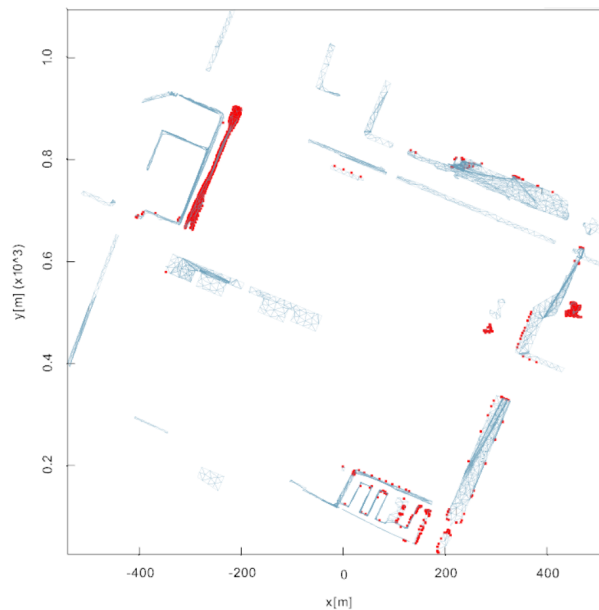


Figure 4.4: Overlaps between the geometries for 'Green'(triangles are represented by their centers in red) and 'Water'(triangulated surface in light blue) landcovers for the complex case.

Although, `3dfier` generates watertight 3D models by accounting for self-intersections and gaps, the cause of the observed overlaps could be the fact that the initial `stl` files were generated separately.

## 4.2 Cases pre-processing

In this section the implemented steps for the preparation of the used cases are presented. For the c\_0 case the mesh generation, solver and turbulence schemes' parameters were based on [García-Sánchez et al. \[2021\]](#). With the exception of the post-processing step, only minor customisations were implemented for the rest of the configuration files, in order to adjust the set up for the application testing. The configuration files for the cases presented in this section can be viewed in [https://github.com/cfratz/set\\_nonUniform\\_z0](https://github.com/cfratz/set_nonUniform_z0).

### 4.2.1 Case configuration

For the configuration of both the simple and complex cases the steps listed below were implemented:

#### Geometry preparation

This included the generation of the s\_0 case geometry and for both the s\_0 and the c\_0 cases, unifying the terrain geometries into one `obj` file, for each case. The cases with the unified model will be referred to as s\_1 and c\_1 respectively. Additionally, slightly modified versions of the aforementioned unified surface models were generated in order to accommodate the case of non-uniform roughness at the inlet (namely s\_2 and c\_2). In particular for the c\_2 case with non-uniform inlet, two surface models were used for the ground. One to represent the water and green landuses, similar to the c\_1 case, and one to represent the terrain neighbouring the inlet of the domain. The aforementioned underlying surface models used for testing the application are shown in Figures 4.5 and 4.6. Details of all the used surface models are shown in Table 4.2, which also includes part of the information displayed in Table 4.1. For the c\_2 case the used surface models cover only part of the extent of the 'Terrain' patch, as shown in Figure 4.6b. An analytical overview of all cases along with their corresponding roughness length values is shown in Table 4.3 .

Case	Faces	Vertices	Patch name	Landcovers
s_0	2	4	terrain	1
	2	4	green1	1
	2	4	water	1
	2	4	green2	1
s_1	8	10	terrain	3
s_2	11	12	terrain	3
c_0	24468	20050	Green	1
	9495	6127	Water	1
	42360	21323	Buildings	1
c_1	33963	26049	WaterGreen	2
	42360	21323	Buildings	1
c_1.1	33963	26049	WaterGreen	2
	42360	21323	Buildings	1
c_2	4	6	Terrain	2
	33963	26049	WaterGreen	2
	42360	21323	Buildings	1

Table 4.2: Details of all triangulated terrain models used in the test cases.

Case c\_1.1 (Table 4.2) is the same in all respects with case c\_1. The only difference is that in the `obj` file used for this case the landuses are flipped and the triangles corresponding to the water surfaces are written first. The reason for creating this case is that it was observed that when this alteration in the `obj` file was implemented, the octree based assignment of option `setZ0Ground` assigned correctly the

roughness length in the areas where overlaps between the vegetation and water surfaces were present (Figure 4.4). This is further illustrated in Section 6.2 of Chapter 6.

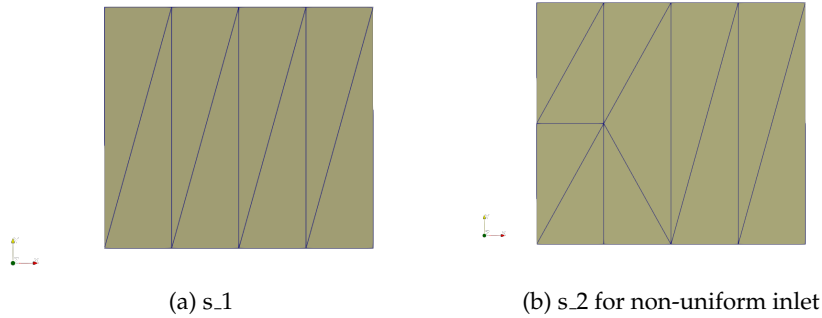


Figure 4.5: Unified terrain surface models for the simple cases.

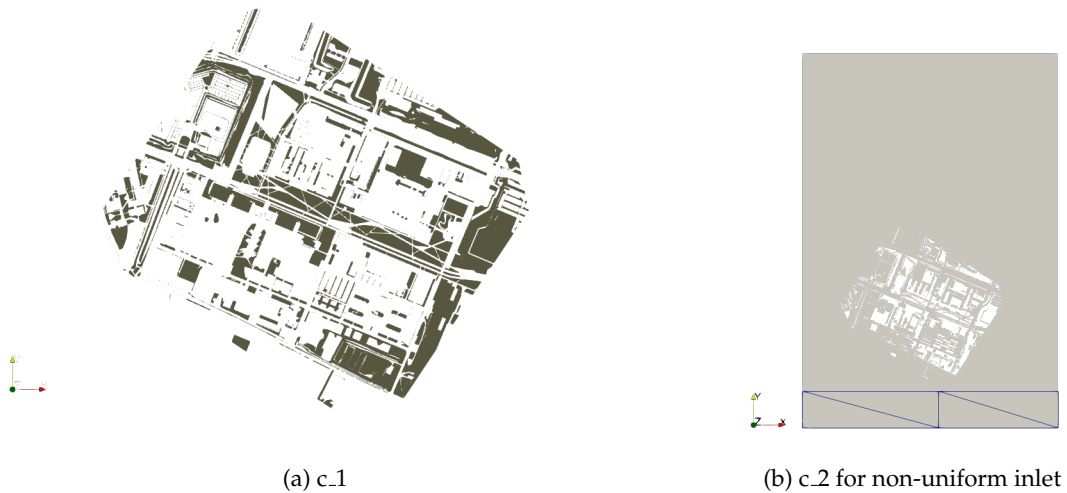


Figure 4.6: Unified terrain surface models used for the complex cases.

### Configuration of the mesh generation parameters

For the generation of the meshes the OpenFOAM `blockMesh` and `snappyHexMesh` utilities were used. The former for the generation of the background mesh and the latter for the generation of the final grid. From here on after we will elaborate more on the details of the mesh of the simple cases, while for the complex cases only necessary details will be provided, since it is a verified case. For the simple cases the used domain is 'empty', it contains no buildings, trees or smaller structures. For this reason the mesh is designed based on a uniform treatment principal in the  $x$  and  $y$  directions. However, in the  $z$  direction higher refinement was imposed close to the bottom of the domain with a gradual transition to lower refined cells towards the top of the domain. This was done to ensure that near walls, where turbulence has a greater effect, changes in the calculated flow parameters are approximated as accurately as possible and that no 'jumps' occur in the calculated profiles.

For the design of the mesh it was ensured that the first cells' center was higher than the maximum roughness length value. The selected dimensions for the computational domain of the 'simple' case are: length( $x$ ) = 969 m, width( $y$ ) = 870 m and height( $z$ ) = 210 m. The background mesh was based on one block refinement with an applied total expansion ratio =  $z_{lastCell} / z_{firstCell}$  of 4 in the  $z$  direction. For the final grid two refinement boxes were used with the same width and length as the computational domain

#### 4 Data, case pre-processing and testing

and with a 5m and 35m height respectively. For the 5m box a refinement of level 2 was implemented, while for the 35m box a refinement level of 1. The patches that intersected the surface models were refined based on a level 3 refinement. It should be noted that this is an ideal case and so the aim of the mesh generation was to ensure that the simulation results converge and are independent of the generated mesh.

In contrast to the simple cases, the complex ones have an area of interest, and it contains buildings and intricate configurations of water, green and terrain surfaces. As such, the computational domain dimensions follow the COST732 guidelines. Based on these guidelines additional space between the model (area of interest) and the domain boundaries is added. Based on the above the domain dimensions are: length ( $y$ )  $\approx 2,705\text{m}$ , width ( $x$ )  $\approx 1,848\text{m}$  and height ( $z$ )  $\approx 586\text{m}$ . For the generation of the background mesh 27 blocks were used with different expansion ratios. For the final grid a top view of the applied refinement boxes for the complex case is graphically presented in [Figure 4.7](#).

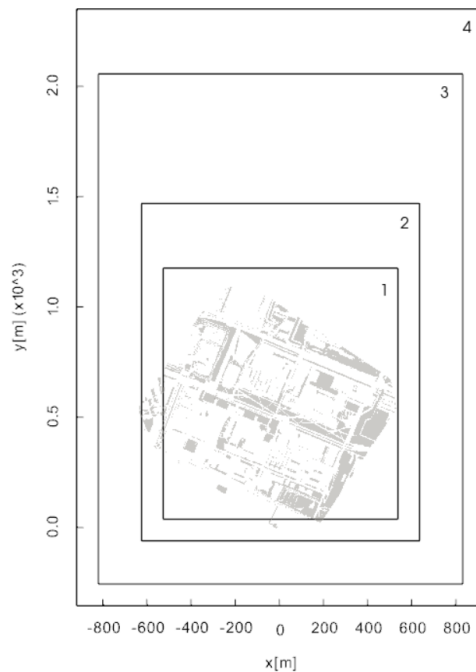


Figure 4.7: Illustration of the refinement boxes used in the complex cases. Boxes 1-3: refined to level 3,2,1 respectively, Box 4: refined to level 4, area of interest is in grey.

#### Choosing an appropriate solver and turbulence model

Both the simple and the complex cases are simulations in the urban [ABL](#) with steady state incompressible turbulent flow. For this type of simulations the steady [RANS](#) set of equations was chosen as the most efficient method for the numerical approximation of the Navier-Stokes equations ([Chapter 2](#)). Based on the above the `OpenFOAM` 'simpleFoam' solver was selected along with the  $k - \epsilon$  turbulence model. For the numerical divergence a bounded `Gauss linearUpwind limited` scheme was selected for velocity and for the turbulence parameters (i.e.  $k$  and  $\epsilon$ ) a bounded `limitedLinear 1` scheme was used, along with a `limited Gauss linear 1` interpolation gradient scheme.

#### Setting up boundary and initial conditions

This part refers to the set up of the initial estimated values for velocity, kinetic energy, dissipation rate, kinematic viscosity and pressure at the boundaries of the computational domain. The values were estimated in accordance to the chosen schemes indicated previously for the simulation of the [ABL](#). At

the inlet the most often used vertical profiles for mean velocity and turbulence properties following the  $k - \epsilon$  model are based on the [Richards and Hoxey \[1993\]](#) set of equations. The used equations for mean velocity,  $k$  and epsilon are shown below:

$$U = \frac{u^*}{\kappa} \ln\left(\frac{z + z_0}{z_0}\right) \quad (4.2)$$

$$k = \frac{u_*^2}{\sqrt{C_\mu}} \quad (4.3)$$

$$\epsilon = \frac{u_*^3}{\kappa(z + z_0)} \quad (4.4)$$

where  $z$  is the height at which the mean wind speed is measured,  $\kappa$  is the von Karman constant: 0.41,  $u^*$  is the friction velocity and  $C_\mu$  a model constant:0.09 of the  $k$ -epsilon turbulence model. Solving [Equation 4.2](#) with mean velocity equal to 10 m/s for the simple cases and equal to 4.97 m/s for the complex cases the friction velocity was derived and used in [Equations 4.3](#) and [4.4](#) to estimate the initial values for the  $k$  and  $\epsilon$  turbulence parameters.

The value for  $z_0$  at the inlet is the roughness of the surface neighbouring the inlet of the domain, assumed equal to the one of the upstream terrain outside the computational domain. For the s.2 and c.2 cases, for which more than one surfaces with different roughness length values are neighbouring the inlet, the values for  $U_*$ ,  $k$  and epsilon are calculated as described in [Section 3.1.5](#) of [Chapter 3](#) based on the aerodynamic roughness ( $z_0$ ). The assigned roughness values per boundary patch for all used cases are shown in [Table 4.3](#). In the table cases s.2.1 and c.2.1 are cases using one roughness length at the inlet, although the neighbouring ground inside the domain has varied roughness. These cases were used for comparison with the cases with non-uniform roughness at the inlet (i.e. s.2 and c.2). This was considered important for the purpose of demonstrating the effect of the proposed methodology for non-uniform inlet on the computed flow parameters.

The roughness length parameter is used, as an input parameter, in all patches at the bottom boundary, for the calculation of the turbulent viscosity (i.e.  $\nu_{\text{t}}$ ), and in the inlet boundary for the estimation of the initial values for velocity,  $k$  and epsilon. Additionally, it is used at the bottom patches for the epsilon, for which the representation of the geometry is provided implicitly [[Blocken et al., 2007b](#)]. This was possible after a modification of OpenFOAM's `epsilonWallFunction`. The modified version, named as `epsilonz0WallFunction`, is based on [Parente et al. \[2011\]](#) adjustment of the  $k$ -epsilon turbulence model for the aerodynamic roughness length( $z_0$ ) parameter. The selected boundary patch types are shown in [Table 4.4](#).

### Setting up the post-processing tools and schemes for the presentation and analysis of the simulation results

For both cases values at selected locations were plotted in order to check and compare the convergence of the simulations between the s.0 - s.1 and c.0 - c.1 cases. In [Figure 4.8](#) and [Figure 4.9](#) the probe configurations for the simple and the complex case are shown respectively. Furthermore slices at 2m height were extracted to illustrate and compare the flow parameters fields. For the s.0 case sample points along a vertical line (930m-965m) at the probe locations ([Figure 4.8](#)) were selected, in order to check the vertical profiles of the flow parameters. The latter was used to check the effect of the generated mesh to the velocity profile and the near wall behaviour of the flow. The profiles are shown in [Figure 4.13](#) in [Section 4.2.2](#).

Case	Patch name	Landcovers	$z_0$
s_0	terrain	1	0.05
	green1	1	0.03
	water	1	0.0002
	green2	1	0.03
	inlet	-	0.05
s_1	terrain	3	[0.05, 0.03, 0.0002]
	inlet	-	0.05
s_2	terrain	3	[0.05, 0.03, 0.0002]
	inlet	-	[0.05, 0.03]
s_2.1	terrain	3	[0.05, 0.03, 0.0002]
	inlet	-	0.05
c_0	Green	1	0.03
	Water	1	0.0002
	Terrain	1	0.5
	y0	-	0.5
c_1	WaterGreen	2	[0.03, 0.0002]
	Terrain	1	0.5
	y0	-	0.5
c_1.1	WaterGreen	2	[0.03, 0.0002]
	Terrain	1	0.5
	y0	-	0.5
c_2	WaterGreen	2	[0.03, 0.0002]
	Terrain	3	[0.5, 0.03, 0.0002]
	y0	-	[0.03, 0.0002]
c_2.1	WaterGreen	2	[0.03, 0.0002]
	Terrain	3	[0.5, 0.03, 0.0002]
	y0	-	0.03

Table 4.3: Assigned roughness length ( $z_0$ ) values per patch.

	inlet	outlet	bottom	top and sides
$U[m/s]$	atmBoundaryLayerInletVelocity	zeroGradient	uniformFixedValue	symmetry
$\epsilon[m^2/s^3]$	atmBoundaryLayerInletEpsilon	zeroGradient	epsilonz0WallFunction	symmetry
$k[m^2/s^2]$	atmBoundaryLayerInletK	zeroGradient	kqRWallFunction	symmetry
$\nu_t[m^2/s]$	calculated	calculated	nutkAtmRoughWallFunction	symmetry
$p[Pa]$	zeroGradient	uniformFixedValue	zeroGradient	symmetry

Table 4.4: Boundary conditions set-up. The types used for the bottom boundary of the domain refer to all relevant patches that represent ground surfaces, for all other surfaces (i.e. buildings), the `epsilonWallFunction` is used.

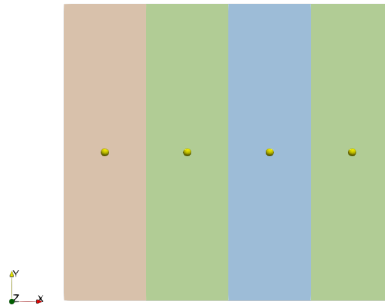


Figure 4.8: Probe configuration for s\_0 and s\_1 cases.



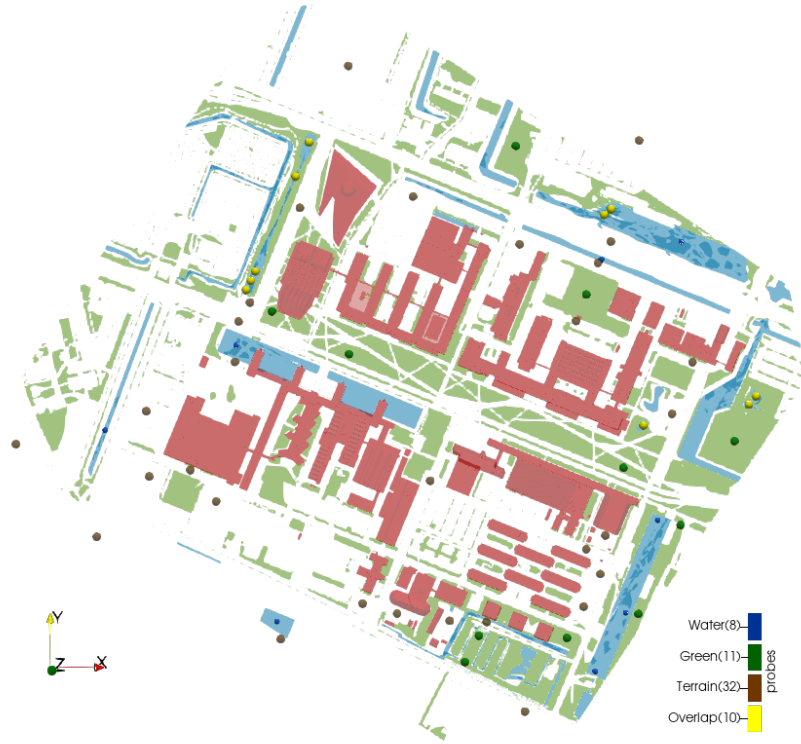


Figure 4.9: Probe configuration for c\_0 and c\_1 cases.

#### 4.2.2 Simple case grid independence study

For the simple case in particular, a grid independence study was also performed. This was based on case s\_0 and the selected grid was used in all simple cases. The grid independence study is a required step in the preparation stage of a CFD simulation for two reasons. Firstly, it ensures that the generated grid will not influence the resulting calculated flow parameters and secondly, it can provide the possibility of a more time efficient grid solution (i.e. a coarser grid) without a significant loss of accuracy in the results. A grid independence study entails the generation of at least three simulations with gradually finer grids. In order for the study to be successful the simulation results should gradually converge at lower values for the finer grids without significant differences. This ensures that the use of an even finer grid would not produce different results. Although the aforementioned steps are necessary for grid independence study, there are additional indicators that need to be computed in order for the study to be complete. For our case four different grids were generated (Table 4.5). The finer grids were generated by multiplying the background mesh resolution by a factor of 1.4.

Grid	XxYxZ(ncells)	Total ncells	Simulation duration(h)
<b>coarse</b>	87x79x20	137.460	2
<b>medium</b>	122x111x28	379.176	3
<b>fine</b>	171x156x39	1.040.364	8
<b>finer</b>	240x219x55	2.890.800	19

Table 4.5: Details of simulations for grid independence.

#### 4 Data, case pre-processing and testing

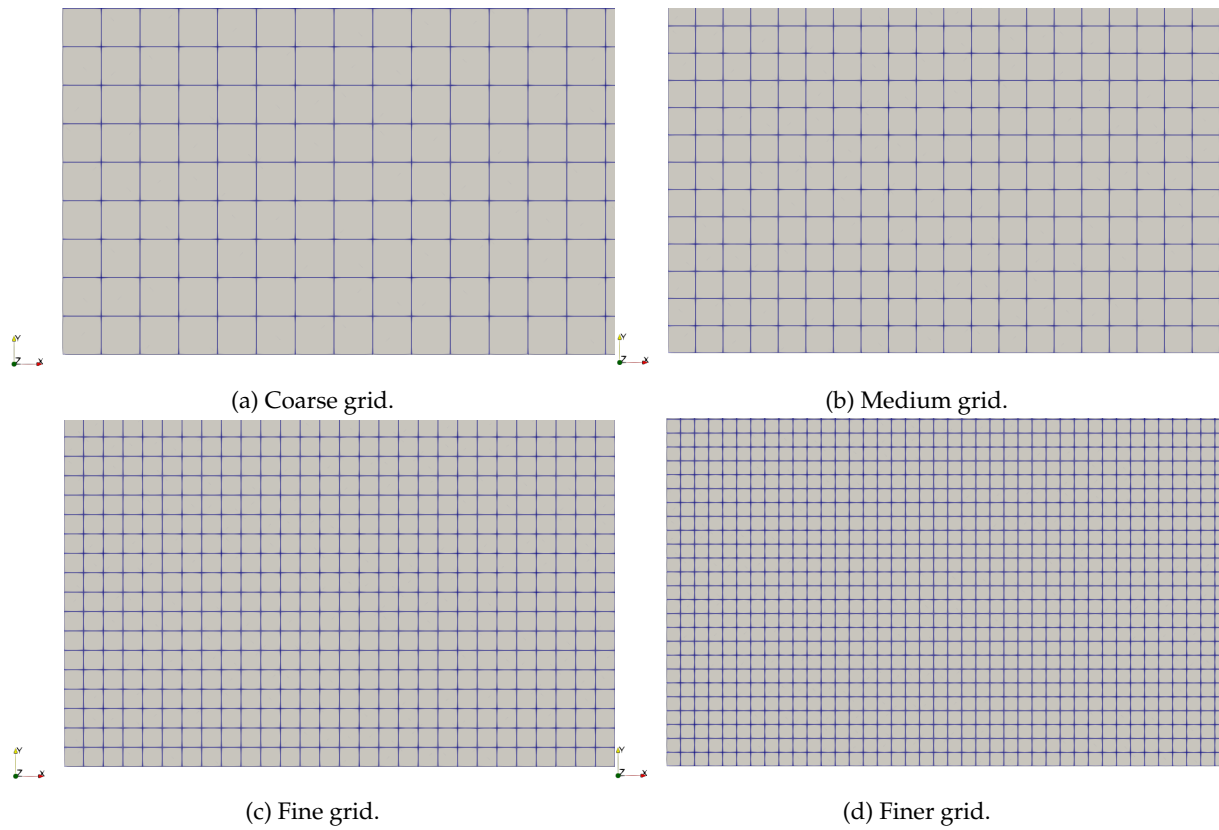


Figure 4.10: Screenshots of the s<sub>0</sub> case grids included in the mesh independence study.

Figures 4.11 and 4.12 show the convergence values for the streamwise velocity ( $U_x$ ) component and the vertical velocity ( $U_z$ ) respectively at four selected probe locations for all grids. The  $U_z$  convergence values are almost identical for all grids at the four probe locations. For the streamwise velocity the convergence values for the medium, fine and finer grids have differences smaller than  $10^{-1}$  m/s (Table 4.6), which is significantly close to conclude that grid independence is achieved. The negative values for  $\text{diff}_{\text{fine}-\text{finer}}$  shown in Table 4.6 are due to the finer simulation needing additional time to fully converge, however still negligible. From the tested grids the medium grid was selected as it fulfilled both the independence and the time efficiency criterion.

Probes	$\text{diff}_{\text{medium}-\text{fine}}$	$\text{diff}_{\text{medium}-\text{finer}}$	$\text{diff}_{\text{fine}-\text{finer}}$
1	0.027	0.032	0.005
2	0.019	0.019	0.000
3	0.013	0.008	-0.005
4	0.007	0.003	-0.003

Table 4.6: Differences for  $U_x$  between the medium, fine and finer grids at selected point locations.

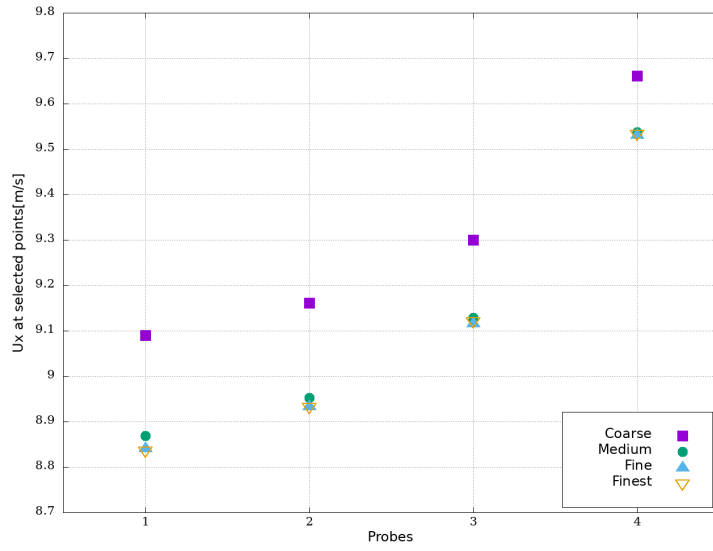


Figure 4.11: Convergence for  $U_x$  at four selected probe locations.

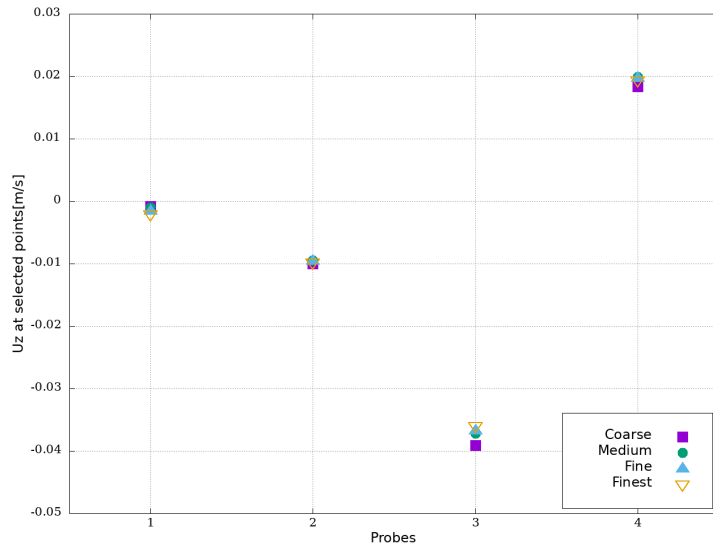


Figure 4.12: Convergence for  $U_z$  at four selected probe locations.

#### 4 Data, case pre-processing and testing

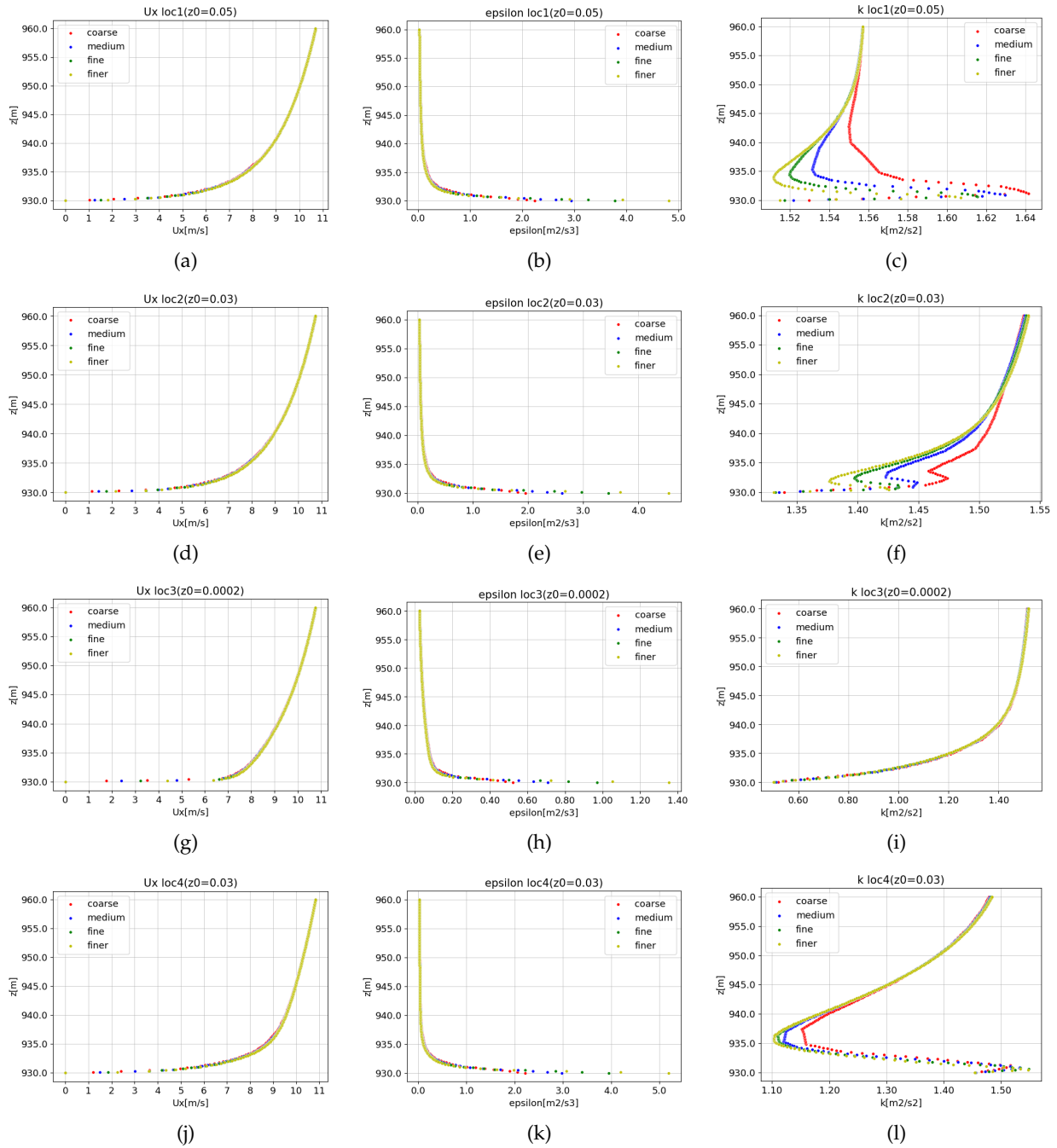


Figure 4.13: Vertical profiles(930m-960m) for  $U_x$ ,  $\epsilon$  and  $k$  at four locations.

## 4.3 Testing

In this section the cases used for the testing of the options included in the proposed methodology are presented. Table 4.7 shows the implemented option/s for every case.

Cases	-setZ0Ground	-setZ0Inlet	-setZ0NoGeom
<b>s_1</b>	x		
<b>s_2</b>	x	x	
<b>s_2.1</b>	x		
<b>c_1</b>	x		
<b>c_1.1</b>	x		
<b>c_2</b>	x	x	x
<b>c_2.1</b>	x		x

Table 4.7: Correspondence between the application options and the tested cases.

As displayed in Table 4.7, c\_2 is the only case used for the testing of option -setZ0NoGeom. The reason for this, is that the additional geometry was only used for the assignment of roughness and it was not included in the mesh generation of the designated patch.



# 5 Implementation and tools

In this chapter details of the implementation are presented. In [Section 5.1](#) a brief description of OpenFOAM's case structure is provided, while the next sections focus more on the selected tools and functionality used in the application. Specifically, in [Section 5.3](#) a description of the selected OpenFOAM classes is given along with their use in the code. Additionally, in [Section 5.4](#) the functions built for application are explained. Lastly, in [Section 5.5](#) the list of used libraries, software and programming languages is presented. The code and application configuration files can be found in [https://github.com/cfratz/set\\_nonUniform\\_z0](https://github.com/cfratz/set_nonUniform_z0).

## 5.1 Case structure

OpenFOAM is a software predominantly created for the use in CFD simulations. It provides functionality that ranges for all steps of a CFD simulation (i.e. preprocessing, solver, postprocessing). It is based on C I/O objects, types, loops, while deriving object-oriented features from C++ as well. In accordance to the functionality it provides its source code utilises a highly templated language to reduce code repeateance and provide the basis for further developmment. Memory manipulation and access OpenFOAM maintains an object registry of entities. The top level of the objectRegistry is Time, and represents the various time steps in a simulation. Time steps are mainly handled by runTime. This also explains the naming of the case folders. A typical case structure consists the following folders:

- 0, for the set-up of the boundary conditions.
- constant, where the surface models, the background mesh information (i.e. polyMesh, it could also the final mesh depending on the case) are stored, as well as the dictionaries for the turbulence model and transport properties are located. For our cases only these dictionaries were required, however, for a different selected physical model more dictionaries might be needed.
- system, where instructions on how the case is ran, the selected source terms' discretisation schemes and how the discretised linear equations are solved are located. Additionally, it contains the set-up for the selected utilities (e.g. related to the mesh, decomposition of the mesh for running in parallel, the postProcessing etc).

In [Figure 5.1](#) the folders generated after the simulation is completed are shown. Folders 1, 2 are the result of the mesh generation process while folder 3002 contains the results of the last time step of the simulation. In the postProcessing folder all the results of the post-processing tools as specified in the system directory are stored.

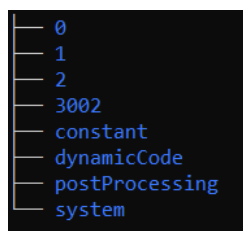


Figure 5.1: 'complex' case folder structure.

## 5.2 Integration in OpenFOAM

The application is easily compiled<sup>1</sup> within OpenFOAM with `wmake` an OpenFOAM compilation script based on `make`. The structure of the application directory is shown in Figure 5.2.

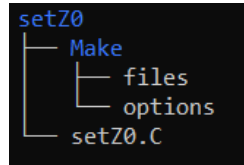


Figure 5.2: Structure of application directory.

Libraries and header files used in the source code of the application are included in the `options` file. All executable(i.e..C) files related to the application that need to be compiled are specified in the `files` file. For personal applications, such as the one created in this thesis, when compiled, the application is written into the specified local directory for custom applications. The compiler can locate this directory from anywhere in the OpenFOAM environment by storing the path under the environment variable `FOAM_USER_APPBIN`.

## 5.3 Classes

The code was based on the use of OpenFOAMv7 classes and their derived functionality. Details and information provided in this section are based on OpenFOAM's C++ Source Code Guide found at <https://cpp.openfoam.org/v7/>. In order to be able to access the selected classes and functions the following header files were included:

`#include "fvCFD.H"`: contains multiple `#include` statements of header files related to finite volume tools. From these, necessary for the application are the definitions related to the `argList` class, the `Time` class, the `fvMesh` class and the `IOdictionary` and `IOobject` classes.

`#include "triSurfaceSearch"`: includes necessary definitions for the use of the `triSurfaceSearch` class. Through this class the octree based search in a surface of OpenFOAM's type `triSurface` is enabled.

`#include "setRootCase.H"`: checks the validity of the user defined options and extracts the command line arguments and stores them under the variable `args` of type `argList`. The command line arguments are passed as `argc`, `argv` to the `main()` function of the application, with `argv[0]` being the name of the program (i.e. `setZ0`).

`#include "createTime.H"`: instantiates the time class and enables access to the available time steps of the case to which the application is applied.

In contrast with the first two definition files, that are declared in the beginning of the code, the `#include` statements for the `"setRootCase.H"` and `"createTime.H"` files are placed inside the main function. The reason for this is that they are directly linked to the arguments provided by the `main()` function. A list of OpenFOAM's selected classes is presented below:

- `argList`: extracts the command line arguments and options. Based on this class the command line arguments are checked, read and stored.

<sup>1</sup><https://cfd.direct/openfoam/user-guide/v7-compiling-applications/#x10-71003.2>



- `fvMesh`: it contains all the geometric and topological information of the generated mesh, that are needed for the finite volume discretisation. For the application an instance of the `fvMesh` is created based on the last available time step (directory 2) as generated after the completion of `snappyHexMesh` utility. Through this class it was possible to access the boundary and internal mesh as well as patch information of the final grid.
- `IODictionary`: it is a class derived from `dictionary` and `IObject` to allow Input/Output functionality through the `objectRegistry`.
- `IObject`: through this class the attributes of an object that is supported with implicit `objectRegistry` management are defined. It is used as input to generate an instance of the `fvMesh` as well as for the instantiation of the `setZODict` dictionary file.
- `triSurface`: It provides a description for triangulated surfaces in `OpenFOAM`. All input surface models provided by the user are converted to `triSurface`. The instances of the class in the code are generated based on the input `.obj` files. A `triSurface` is described by its faces, points and patches (different from the mesh generated patches).
- `pointIndexHit`: describes the relation between a face and a point. In the application it is used to describe the interaction between the face centers and the triangles stored in the octree. The interaction in the code is described based on the following member functions:
  - `hit()`: returns True if a hit is successful,
  - `index()`: returns the index of the nearest found triangle to the tested face center,
  - `hitPoint()`: returns the hit point.
- `triSurfaceSearch`: It is a support class for the `triSurface`. It provides functions for the construction of a `triSurface` based octree as well as octree based operations and searches. From the available functions the void `findNearest(const pointField& samples, scalarField& nearDistSqr, List<pointIndexHit>& info)` is called using as samples the mesh face centers corresponding to the input geometry. This function creates the octree using the `indexedOctree` template class and calls function `indexedOctree<treeDataTriSurface>::findNearest`, based on which overlaps between the sample points and the `triSurface` faces are found. The `treeDataTriSurface`<sup>2</sup> is an alias class defined based on the `treeDataPrimitivePatch` template class of type `triSurface`.
- `indexedOctree`: It is a base class that provides functionality for the generation, manipulation and analysis for octree data structures. It is templated to address different types of input to be converted into octree data structures.

## 5.4 Functions

In this section a description of the functions included in the application code is presented, based on their usage. The first group of functions are functions of the same declaration used in `OpenFOAM v7` utility `foamDictionary`<sup>3</sup>. The available functions are listed below:

*Functions responsible for accessing the user-specified dictionary and checking and extracting the command-line argument from option `entry`:*

- `readDict`: reads the user-specified dictionary file, ensuring that the file is written in the dictionary's specified format(i.e. ASCII or binary).
- `scope`: converts the user-specified entry to a syntax using `'.'`, in case the specification included a semicolon instead.
- `dictAndKeyword`: in case the entry is located inside a subdictionary, the keyword and the sub dictionary are stored separately.

<sup>2</sup>[https://cpp.openfoam.org/v9/treeDataTriSurface\\_8H\\_source.html](https://cpp.openfoam.org/v9/treeDataTriSurface_8H_source.html) at line 48

<sup>3</sup>[https://cpp.openfoam.org/v7/foamDictionary\\_8C\\_source.html](https://cpp.openfoam.org/v7/foamDictionary_8C_source.html) lines 131-237

## 5 Implementation and tools

- `lookupScopedDict`: checks that specified entry exists.

Functions responsible for performing checks in the input `.mtl` and `.obj` files as well as the user-specified target patch/es:

- `checkMtl`: checks the correspondence between the landcovers included in the `.mtl` file and the ones specified in the `setZODict` file.
- `z0ToTriangle`: checks that the included landcovers in the `.obj` file exist in the `setZODict` and assigns the specified  $z_0$  value to the triangles.
- `checkPatch`: checks depending on the selected option whether the specified patch is the inlet or a ground patch.

Functions responsible for the assignment of non-uniform roughness length ( $z_0$ ) to the designated patches:

- `assignZ0`: takes care of the assignment of roughness length for option `-setZ0Ground` to the specified patch face centers based on found overlaps between the face centers and the surface triangles.
- `assignParam`: takes care of the assignment of roughness length for option `-setZ0Inlet` to the inlet face centers.
- `calculateParams`: calculates the turbulence parameters and generates a list for every input patch.

## 5.5 Tools

The application is built using C/C++ programming languages along with the in-built functionality of OpenFOAM v7 on Ubuntu18.04.5 in Windows 10. The same version of OpenFOAM was used for the testing of the application as well as running the selected cases. Handling the running of the cases was done using the provided bash shell. The generation of the `.obj` files for the 'simple' case was done in Python for the original case using the `startinpy`<sup>4</sup> library, a library that implements a robust Delaunay triangulation, mainly suitable for the modelling of terrains. For the 'complex' case the original geometries were, as mentioned in Section 4.1 in Chapter 4, supplied in `.stl` format. The conversion of the `.stl` files to `.obj` format was done in the software MeshLab 2021.10 [Cignoni et al., 2008]. MeshLab was also used for combining the original cases' `.obj` files for both the 'simple' and 'complex' cases. For this the Flatten Visible Layers option was used ensuring that all duplicate vertices unreferenced vertices were deleted.

For the generation of plots and the visualisation of the results the following Python libraries were used:

- `matplotlib`: a 2D visualisation toolkit [Hunter, 2007].
- `seaborn`: a visualisation toolkit providing refined versions of `matplotlib`'s statistical graphics using a simpler interface [Waskom, 2021].
- `pyVista`: a toolkit specialised in the handling and visualisation of `vtk` files [Sullivan and Kaszynski, 2019]. It was used for the generation of the resulting contour maps and `vtk` comparisons and visualisations.

All probe plots were created with `gnuplot`<sup>5</sup>, a command-line program for 2D and 3D plots, providing user-customisation through its built-in scripting language. Although, `gnuplot` is incorporated in the package of OpenFOAM, due to incompatibility issues with Windows, it was accessed directly. Lastly, part of the geometry visualisations presented in Chapters 3 and 4 were generated in the software Paraview<sup>6</sup>.

---

<sup>4</sup><https://github.com/hugoledoux/startinpy/>

<sup>5</sup><http://www.gnuplot.info/>

<sup>6</sup><https://www.paraview.org/>

## 6 Results and analysis

In this chapter the results of the simulations as well as of the application are presented. In [Section 6.1](#) a comparative analysis of the generated grids between cases `s.0 - s.1` and `c.0 - c.1` is presented, in order to investigate the effect of the geometry modifications on the generated meshes. In [Sections 6.2](#) and [6.3](#) the results of options `setZ0Ground` and `setZ0Inlet` are presented.

### 6.1 Grid similarity between the use cases

In this section the results of the mesh generation process for the original (separate) and the modified (unified) geometries are shown for each of the simple (`s.0`, `s.1`) and complex (`c.0`, `c.1`) cases. This is done based on the differences between the original and the modified case in terms of the number of cells per refinement level ([Table 6.2](#)) and the number of faces per boundary patch ([Table 6.3](#)). Furthermore, selected characteristics of the meshes are compared ([Table 6.1](#)).

This step is essential to support the analysis of the results of option `-setZ0Ground`, but also to investigate the effect of the modified geometry in the mesh generation and the simulation results. It is important to mention that the modified geometries were a product of combined `obj` files and only duplicate points were removed during the process ([Section 4.2](#) in [Chapter 4](#)), so only minor changes were made. This makes the use of these indicators relevant, even though they do not provide information regarding the spatial distribution of the differences.

Characteristic	Simple cases		Complex cases		
	<code>s.0</code>	<code>s.1</code>	<code>c.0</code>	<code>c.1</code>	<code>Diff<sub>c.0-c.1</sub></code>
<b>Max skewness</b>	$\approx 0.33$	$\approx 0.33$	$\approx 13.66$	$\approx 13.66$	0
<b>Skew nfaces</b>	0	0	301	306	-5
<b>nCells(type:hexahedra)</b>	3,507,378	3,507,378	15,031,780	15,031,775	5
<b>nCells(type:prisms)</b>	0	0	87,745	87,767	-22
<b>nCells(type:tet wedges)</b>	0	0	251	254	-3
<b>nCells(type:polyhedra)</b>	284,382	284,382	681,315	681,263	52

Table 6.1: Mesh characteristics for the simple and complex cases generated from the OpenFOAM utility `checkMesh`.

In [Table 6.1](#) the selected characteristics of the meshes are presented. The selection was based on the features for which the `c.1` case deviated from the `c.0`. For the simple cases all displayed characteristics of the mesh are the same. On the other hand, for the complex ones differences are observed in the number of cells per used cell type, as well as in the number of found skew faces. The latter deviation is only 5 skew faces and in combination with the fact that 'Max skewness' is the same for both cases, it is considered negligible. Regarding the differences in the types of cells, the majority of the deviations concentrate in the 'prisms', in favour of the 'modified' case (22), and in the 'polyhedra' in favour of the 'original' case(52). For the 'polyhedra' specifically the bulk of the differences is shared between 'polyhedra' of 6, 9 and 12 faces. Based on the aforementioned observations, it is difficult to conclude whether the grid of the `c.1` case is simpler from the one generated for the `c.0` case. For this reason, their effect will be further assessed through the simulation results, which are presented in the next sections.

In [tables 6.2](#) and [6.3](#) the deviations in terms of number of cells and faces are shown respectively. For the simple cases, as for the mesh characteristics, all chosen indicators are the same. This was anticipated

## 6 Results and analysis

since the refinement at the higher level is uniform through out the extent of the domain and the geometries are not blended. On the other hand for the complex cases there are differences, with the c\_0 case being the one with the additional cells and faces. More specifically, the differences in terms of cells are mainly concentrated at the higher refinement level(level 5), as shown in Table 6.2, and in terms of number of faces, in the boundary patches that represent the 'Buildings', 'Green' and 'Water' surfaces. For the 'Green' and 'Water' surfaces this was expected since they are the ones whose geometry was altered. The accounted differences in the 'Buildings' patch, whose used geometry was not altered, are possibly a consequence of this modification. It should be noted that the total difference between the number of cells and faces, respectively, can be explained by the use of different types of cells in the mesh. For both the simple and complex cases execution time for the generation of the mesh was the same regardless of the geometry used.

Ref level \ nCells	Simple cases		Complex cases		
	s_0	s_1	c_0	c_1	Diff <sub>c.0-c.1</sub>
0	270,840	270,840	23,628	23,628	0
1	704,184	704,184	186,624	186,624	0
2	1,083,360	1,083,360	1,539,577	1,539,577	0
3	1,733,376	1,733,376	8,280,903	8,280,904	-1
4	-	-	4,543,981	4,543,976	5
5	-	-	1,226,378	1,226,350	28
<b>Total</b>	<b>3,791,760</b>	<b>3,791,760</b>	<b>15,801,091</b>	<b>15,801,059</b>	<b>32</b>

Table 6.2: Number of cells per refinement level for the simple and complex cases as generated from the OpenFOAM utility checkMesh.

Patch \ nFaces	Simple cases		Complex cases		
	s_0	s_1	c_0	c_1	Diff <sub>c.0-c.1</sub>
Green1	216,672	866,688	-	-	-
Green2	216,672		-	-	-
Terrain	216,672		995,944	995,942	2
Water	216,672		20,926	90,380	19
Green	-	-	69,473		
Buildings	-	-	481,005	480,986	19
Top	13,542	13,542	4,212	4,212	0
Inlet	9,102	9,102	6,480	6,480	0
Outlet	9,102	9,102	6,480	6,480	0
Sides(symmetric)	20,008	20,008	18,720	18,720	0
<b>Total</b>	<b>918,442</b>	<b>918,442</b>	<b>1,603,240</b>	<b>1,603,200</b>	<b>40</b>

Table 6.3: Number of faces per boundary patch for the simple and complex cases as generated from the OpenFOAM utility checkMesh.

In order to be able to have a better understanding of whether these differences are low we calculated their percentage on the total number of their corresponding category for the original case, which in all instances had the larger number of cells and faces. The calculation was done as shown below:

$$diff_{nCells_5} = \frac{diff_5}{nCells_{c.0_5}} \approx 0.002\% \quad (6.5)$$

$$diff_{nFaces_{GWB}} = \frac{diff_{GWB}}{nFaces_{c.0_{GWB}}} \approx 0.007\% \quad (6.6)$$

As shown by the calculated percentages, the numbers are significantly small. This also means that any misassignment in the roughness length resulting from these differences will most likely have a negligible effect in the simulation of the flow. For this reason it is not considered important to find the spatial distributions of the deviations. Instead, we will verify our conclusion based on the comparison of the simulation results as presented in the next section. Based on the above, the differences between the meshes of the s\_0 - s\_1 and the c\_0 - c\_1 cases are not significant and thus the meshes are regarded as similar. In particular for the complex cases, however, if the observed differences are found to be a probable cause for deviations also in the calculation of the flow parameters further investigation into their location would be required.

## 6.2 Option -setZ0Ground

In this section the results of option -setZ0Ground are presented, through comparison of cases s\_0 - s\_1 and c\_0 - c\_1. The comparison is performed in terms of the assigned number of the roughness length values per landcover and the calculated flow parameters. In Figure 6.1 a bar plot of the number of roughness length values per landcover is shown for the simple cases. The assigned number of roughness values is the same for both cases, which can also be observed in the illustration of Figure 6.2.

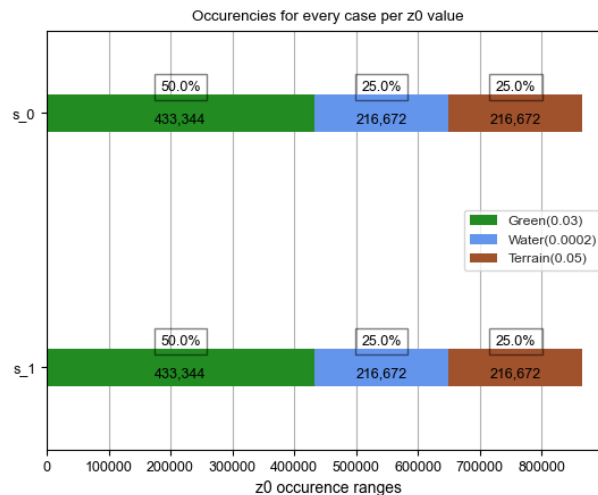


Figure 6.1: Plot of the occurrences of assigned  $z_0$  values per landcover for the simple cases.

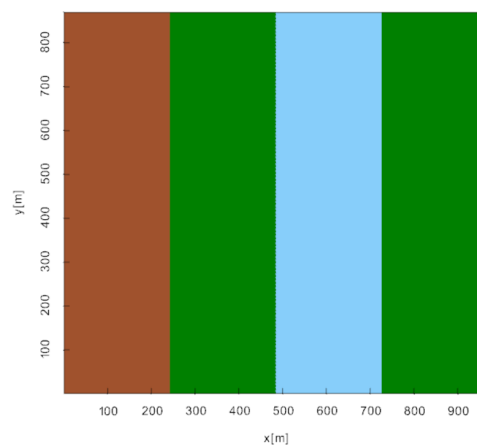


Figure 6.2: Illustration of the assigned roughness length for case s\_1.

## 6 Results and analysis

Additionally, the convergence values for  $U_x$  and  $U_z$  at the four selected probe locations (Figures 6.3 and 6.4) present no deviations between the s\_0 and s\_1 cases.

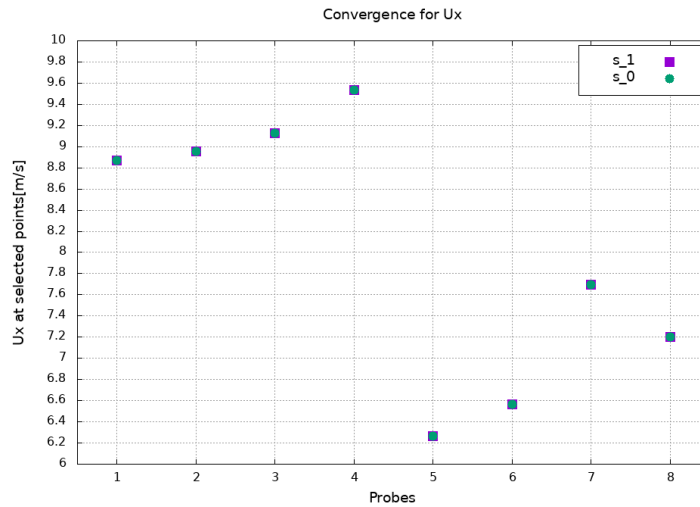


Figure 6.3:  $U_x$  at four probe locations.

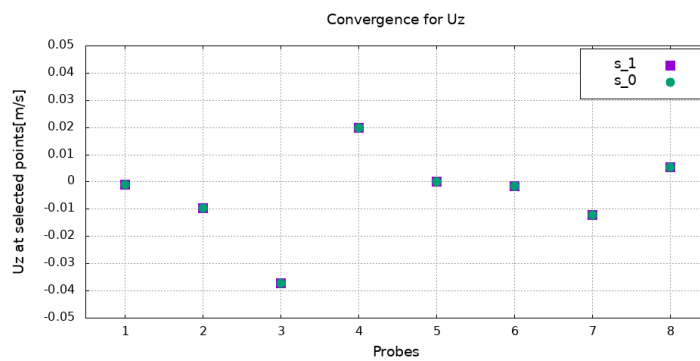


Figure 6.4:  $U_z$  at four probe locations.

The presented results for the simple case show that the process of assigning roughness length did not affect the simulation results, but also that the implemented modification in the geometry, as shown in the previous section, did not have any effect in the mesh generation process. In this sense, the use of the application for a case such as this can only contribute in facilitating the process of setting up the case, since the user is required to provide less parameters, and thus it can improve time efficiency.

As mentioned in [Chapter 3](#) overlaps in the input geometry can compromise the accuracy of the assignment process as implemented in the built application. In particular, this is the case for the geometry used in the complex cases ([Section 4.1](#) in [Chapter 4](#)). For this reason, the assignment in the found regions, where overlaps occur between the different landcovers, is expected to fail. In order to verify this, case c.1.1 is used. For this additional case, the assignment process is not hindered by the accounted overlaps. A description of the details regarding the geometry used for this case are provide in [Section 4.2.1](#). However, it should be noted that the implemented alterations were customised specifically for the complex case, and thus can not be generalised. The results of the three cases are illustrated in [Figures 6.5](#) and [6.6](#) and quantified in the bar plot of [Figure 6.7](#).

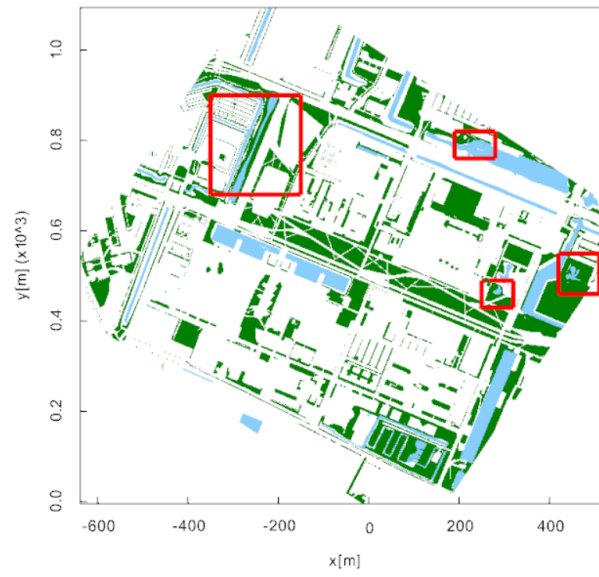


Figure 6.5: Illustration of the assigned roughness length values for the c\_0 case.

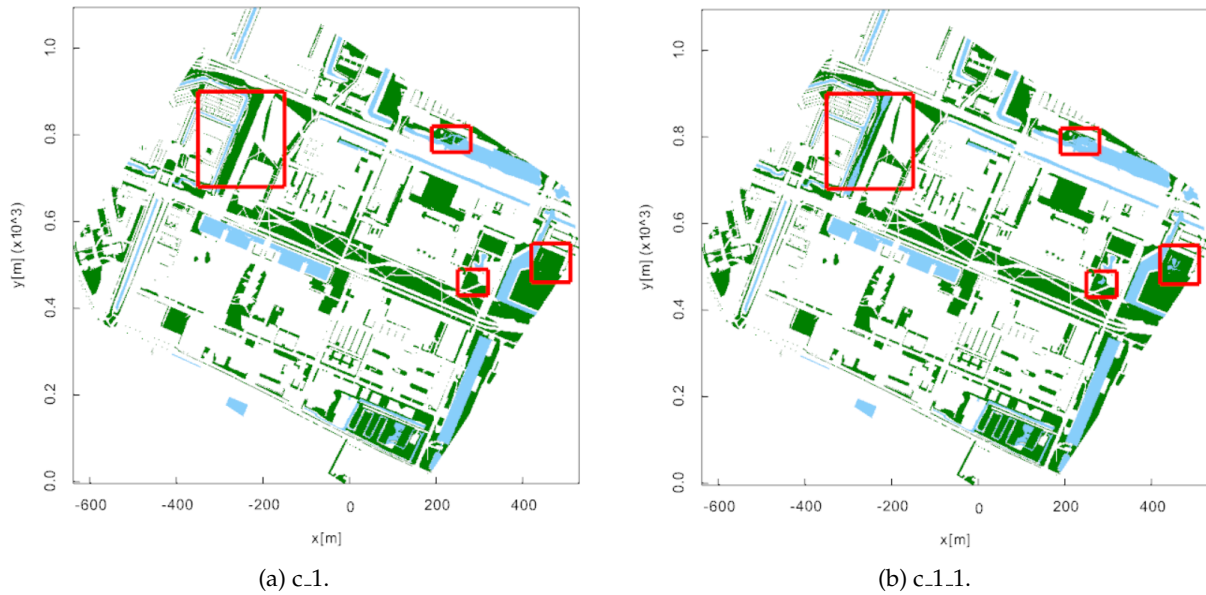


Figure 6.6: Illustrations of the assigned roughness length as generated from option -setZ0Ground. The red boxes signify the areas with observed differences.

## 6 Results and analysis

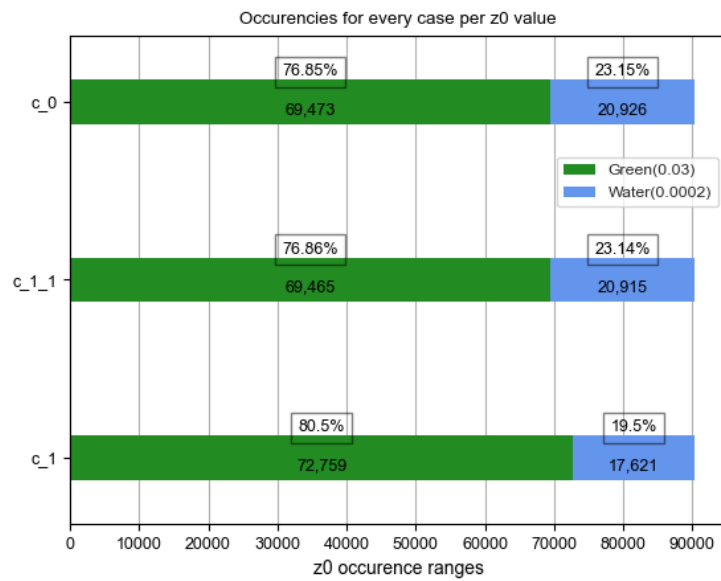


Figure 6.7: Plot of the occurrences of assigned  $z_0$  values per category for the complex case.

As shown in the bar plot although the assigned number of values per category are different between the c.0 and the c.1.1 case, the corresponding percentages are almost identical. On the other hand, for the c.1 case the effect of the overlapping surfaces is evident. Specifically, it displays a much higher percentage of assigned values in the 'Green' landcover and subsequently a much lower one for the 'Water'. The differences correspond to  $\approx 3,286$  more faces in the 'Green' landcover and  $\approx 3,305$  in the 'Water' landcover when compared to the c.0 case assigned values. The approximation is used here since there is a geometry difference of 19 faces (Section 6.1), which is impossible to locate, nevertheless negligible. The correspondence between the differences and the regions with overlaps per landcover can also be observed through visual inspection of Figure 6.8. This validates the original assumption that the regions with overlaps hindered the assignment process of the application.

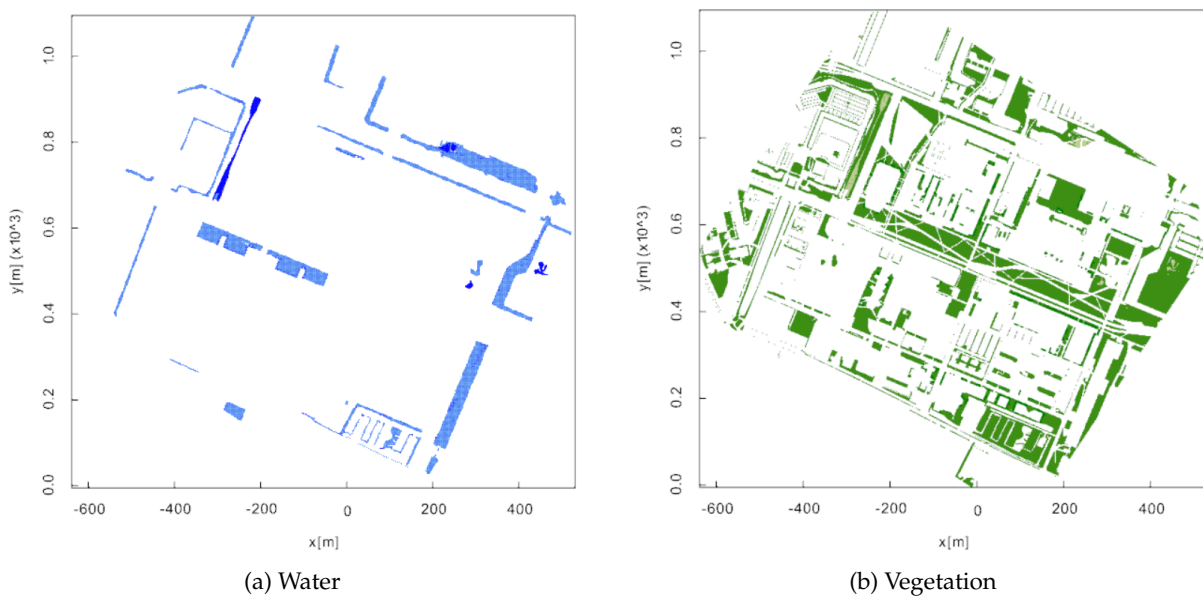


Figure 6.8: Differences per landcover based on the assigned roughness values between the c.0 and c.1 cases.



In Figures 6.9 and 6.10 the convergence values for the three cases are presented, as generated for fifty one probes at 2m height above ground. For both  $U_y$  and  $U_z$  all three cases converge at the same values.

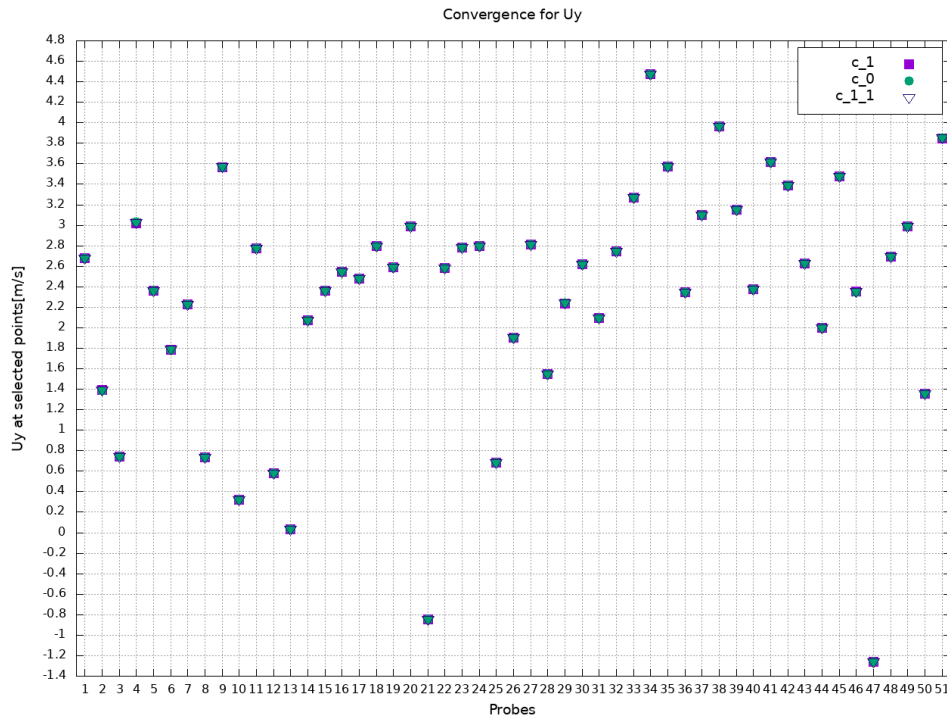


Figure 6.9:  $U_y$  at fifty one probe locations at height 2m.

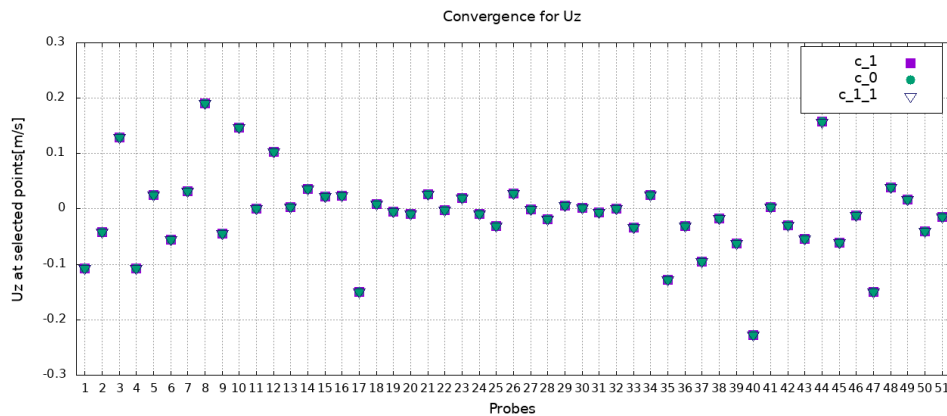


Figure 6.10:  $U_z$  at fifty one probe locations at height 2m.

However, in the presented probe locations the regions with overlapping geometries were not included. The generated values for ten selected locations in the overlapping regions for the two velocities are shown in Figures 6.11 and 6.12 respectively. Although case c\_1.1 converges at the same values as c\_0, the results for c\_1 present deviations up to  $10^{-1}$  for  $U_y$ . Differences are also present in the  $U_z$  velocity component, but are smaller.

## 6 Results and analysis

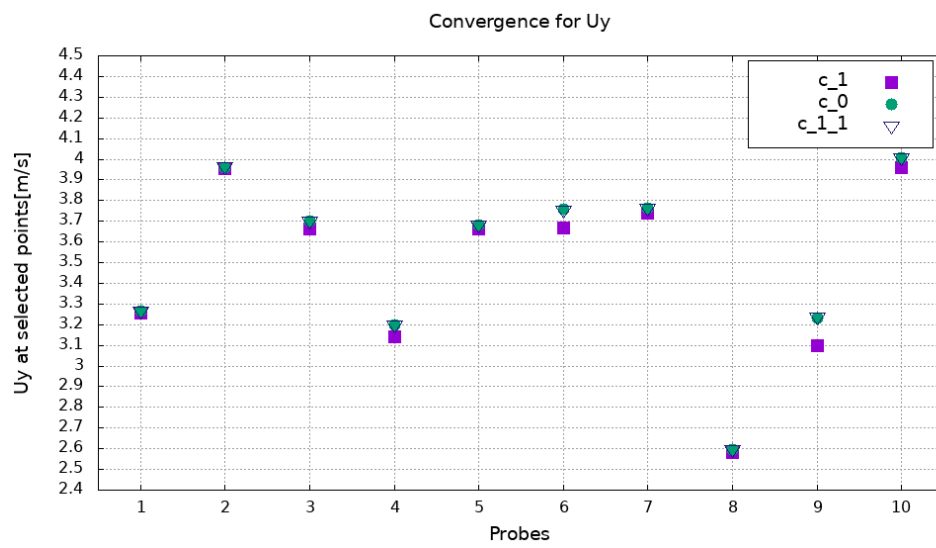


Figure 6.11: Uy at ten probe locations with different roughness length at height 2m.

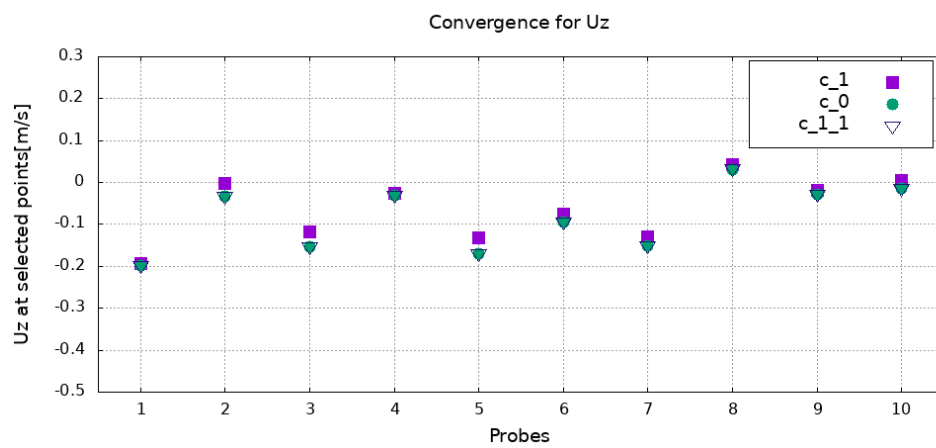


Figure 6.12: Uz at ten probe locations with different roughness length at height 2m.

Illustrations of the differences between the c\_0 and the c\_1, c\_1.1 cases are shown in Figure 6.13, for the c\_1 (6.13a) and the c\_1.1 (6.13b) case respectively. It should be noted that the graphs as well as the numbers depicted in Table 6.4 are derived based on proximity between the grid vertices of the c\_0 and the c\_1, c\_1.1 cases, since the grids are similar but not exactly the same. The vertices of the grids were used, since it was not possible to use the face centers in this situation. The generated graphs confirm that the c\_1 case only deviated from the original in the regions where overlaps between the geometry of the two landcovers were present. The absence of colours corresponding to values larger than ( $\approx 0.45$ ) for the c\_1 case and larger than ( $\approx 0.15$ ) for the c\_1.1 one, is due to the smaller number of points for those intervals. As shown in Table 6.4 the majority of the points for both c\_1, c\_1.1 cases have differences that lie between  $[0, 0.05]$ , which is considered negligible. However, the c\_1 case has a total of 1823 points with differences between 0.05 and 0.45, while the c\_1.1 case has only 80 points with values in the same range. For both cases, differences larger than 0.45 correspond to only three points. This is the reason why some of the colours in the used colourbars are not visible in the difference maps.

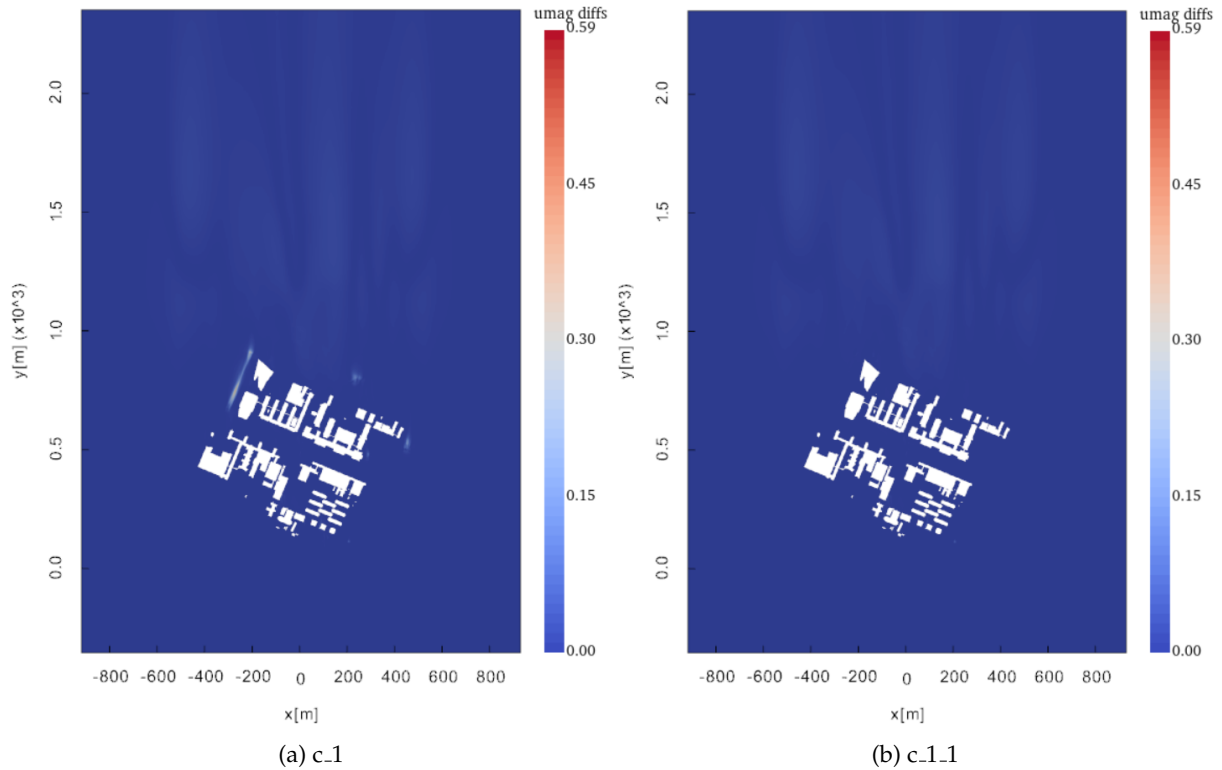


Figure 6.13: Differences between the c\_0 and c\_1 case for  $U_{magnitude}$  based on a 2D slice at 2m height above ground.

Umag diff ranges	c_0-c_1	c_0-c_1.1
$[0, 0.05]$	1,070,253	1,071,993
$(0.05, 0.1]$	892	31
$(0.1, 0.15]$	428	7
$(0.15, 0.2]$	226	5
$(0.2, 0.25]$	129	1
$(0.25, 0.3]$	80	2
$(0.3, 0.45]$	35	4
$(0.45, 0.59]$	2	2

Table 6.4: Number of points per  $U_{magnitude}$  difference interval.

## 6 Results and analysis

The contour maps shown in Figures 6.14, 6.15 and 6.17 display  $U_{magnitude}$  in contours for the three cases using a step of 0.375 [m/s]. From visual inspection the three cases are similar, although deviations can be observed with a closer look in Figures 6.15a and 6.17b in the overlapping regions (Figure 6.16).

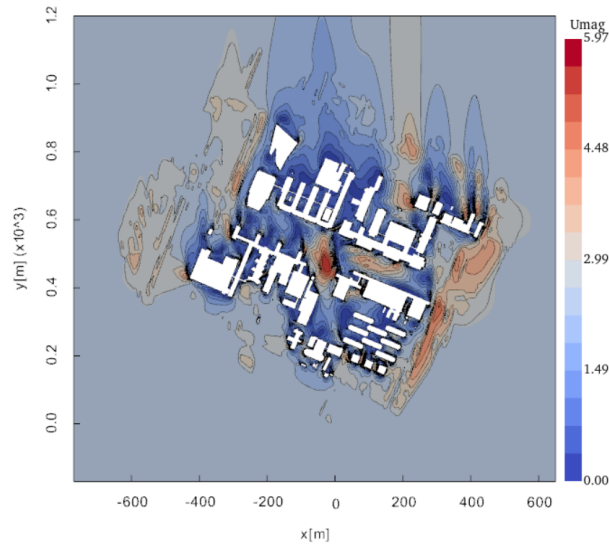


Figure 6.14: Contour map with contours per 0.375 [m/s] for  $U_{magnitude}$  for the c\_0 case, using a 2D slice at 2m height above ground.

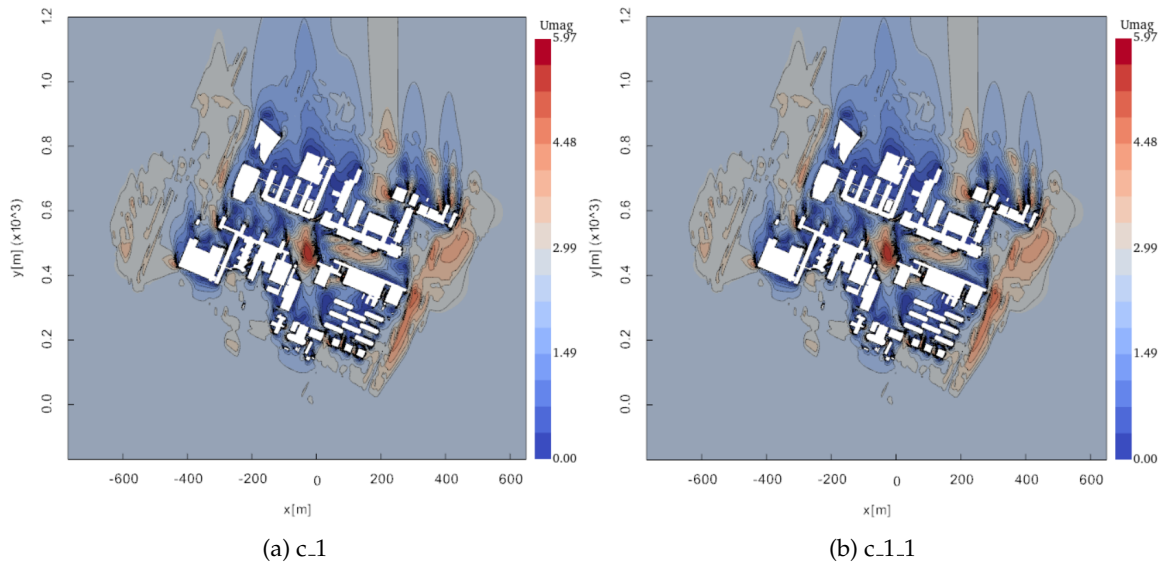


Figure 6.15: Contour maps with contours per 0.375 [m/s] for  $U_{magnitude}$ , using a 2D slice at 2m height above ground.

Based on the above observations we can conclude that the result of the assignment process of `-setZ0Ground` is close enough to the c\_0 case result, under the restriction that there are no overlaps in the geometry. Regarding the effect of the geometry modifications in the simulation results, it can be assumed that the deviations observed in the c\_1.1 case can be attributed to the mesh differences analysed in Section 6.1. Although, as shown in Table 6.4 and illustrated in Figure 6.13b they are smaller than 0.05 in their majority.

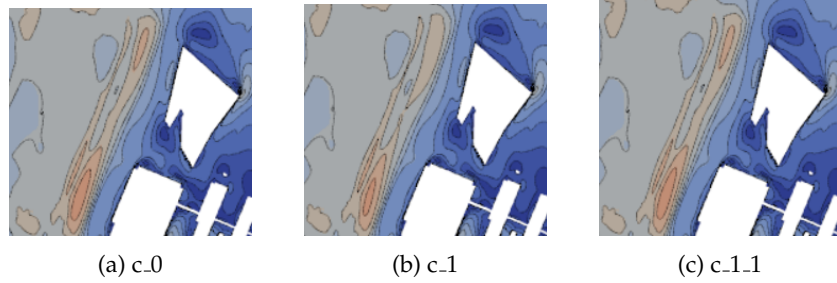


Figure 6.16: Screenshots of a zoomed area in the contour maps of Figure 6.15 with observed differences.

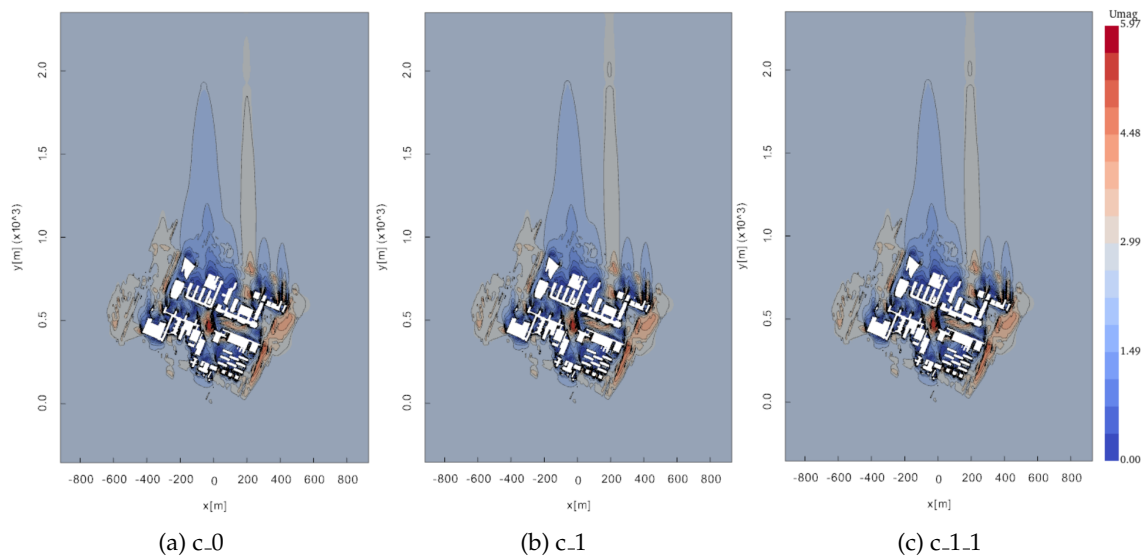


Figure 6.17: Contour maps with contours per 0.375 [m/s] for  $U_{magnitude}$ , using a 2D slice at 2m height above ground.

### 6.3 Option -setZ0Inlet

In this section the results for option `-setZ0Inlet` are presented. Figures 6.18 and 6.20 are illustrations of the assigned roughness length values to the ground and inlet patches for the `s.2` and `c.2` cases respectively. The assessment of the assignment process was based on a comparison of the results with the input `obj` files and the mesh of the domain through visual inspection. In particular, Figures 6.19b and 6.21b display the mesh wireframe of the inlet and bottom boundary along with the edges of the triangulated geometries. Since the assigned inlet values are within the corresponding boundaries of the two geometries the result is considered successful for both cases. It should be noted that in order to avoid having misassigned values, the location of the splitting edge between the two ground geometries neighbouring the inlet boundary was explicitly selected.

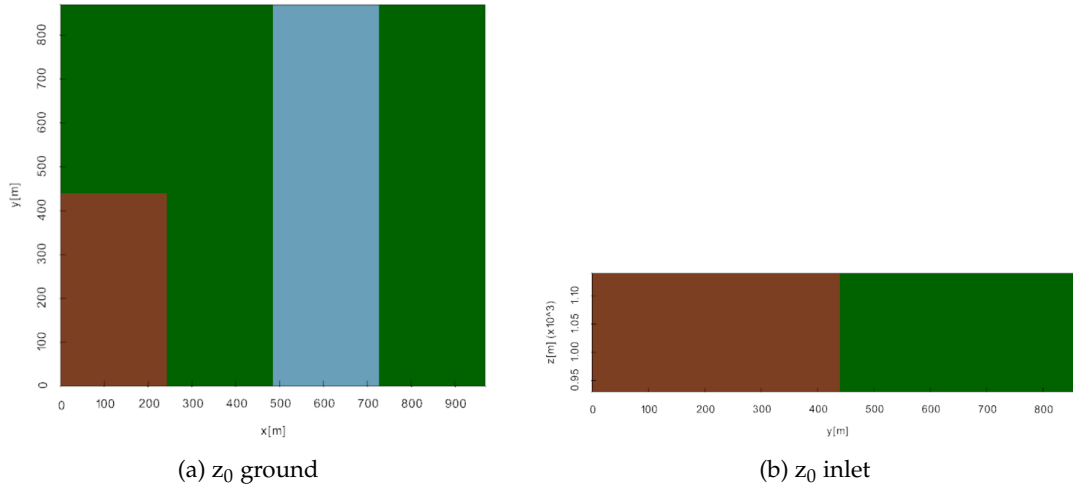


Figure 6.18:  $z_0$  values for the `s.2` case with non-uniform roughness at the inlet ( $z_0$ : 0.05 (terrain), 0.03 (vegetation)).

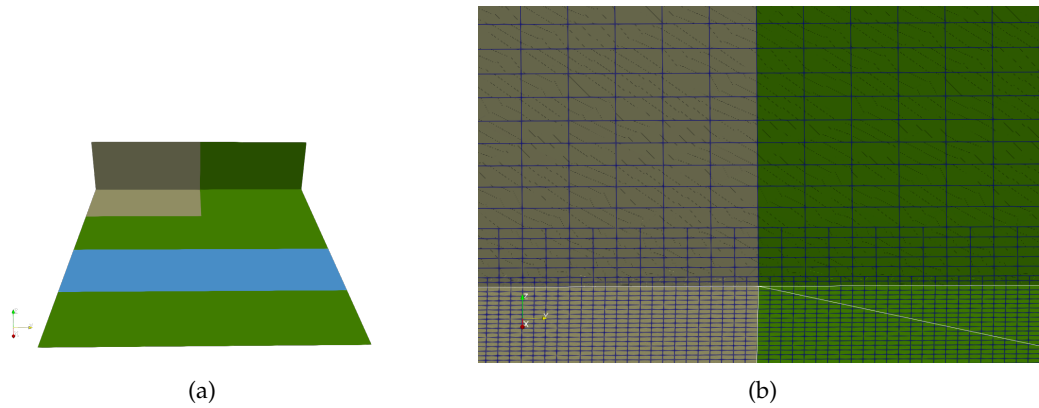


Figure 6.19: Screenshots of the `s.2` case domain inlet and bottom boundaries. 6.19b displays a zoomed view of the separation line between the two geometries. The lines in 'white' are the edges of the triangulated geometries, while in 'blue' the wireframe of the meshed boundaries.

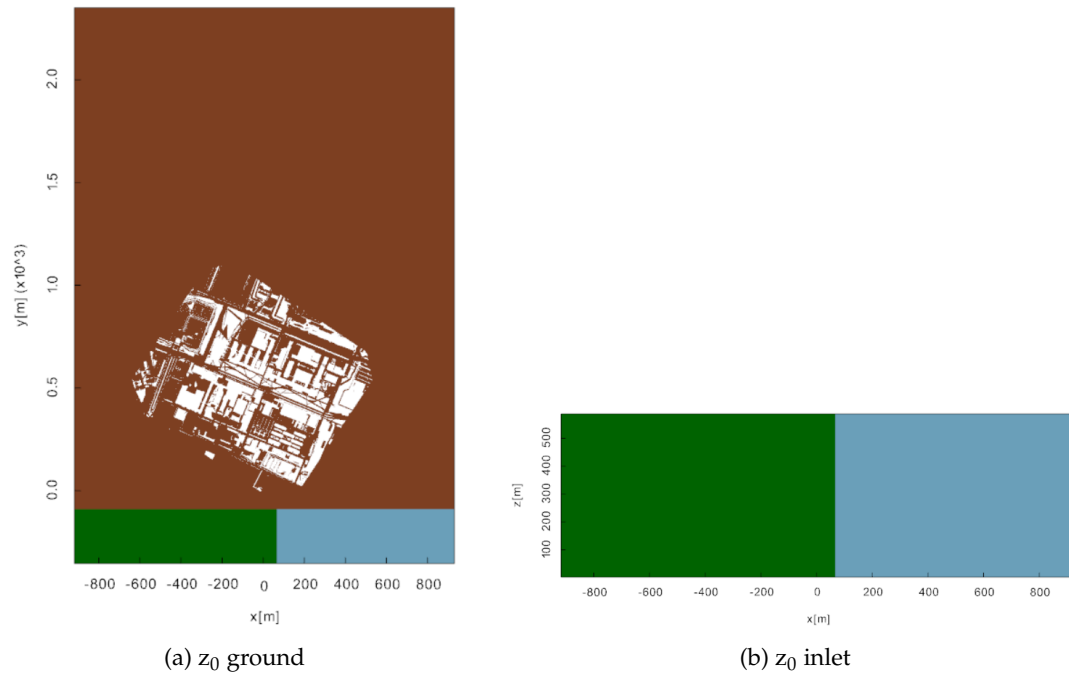


Figure 6.20:  $z_0$  values for c.2 case with non-uniform inlet ( $z_0$ : 0.03 (vegetation), 0.0002 (water)).

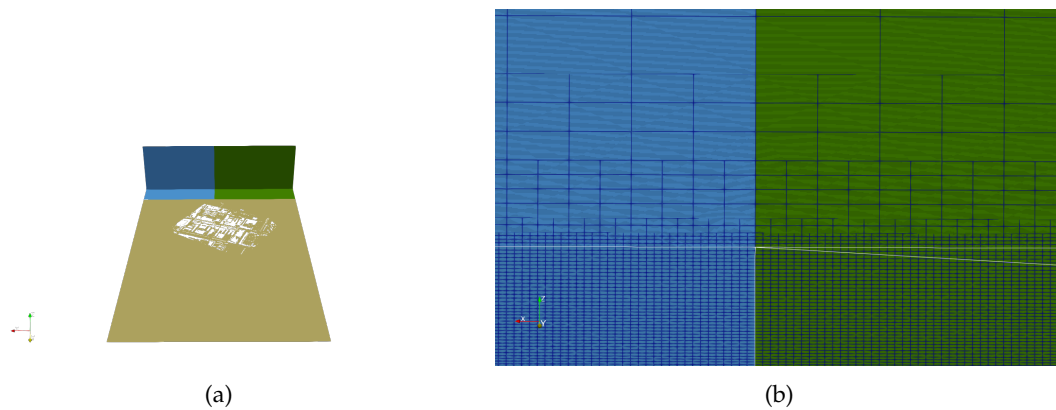


Figure 6.21: Screenshots of the c.2 case domain inlet and bottom boundaries. 6.21b displays a zoomed view of the separation line between the two geometries. The lines in 'white' are the edges of the triangulated geometries, while in 'blue' the wireframe of the meshed boundaries.

## 6 Results and analysis

Since there was no reference case for the testing of option `-setZ0Inlet`, in order to assess the impact of having a non-uniform inlet on the simulation results we plotted the convergence values at selected point locations, five for the `s.2` case and two for the `c.2` case. For this the two cases were compared against cases `s.2.1` and `c.2.1` respectively. These cases have the same patch configuration and roughness length at the ground but use a uniform  $z_0$  at the inlet. Probes 1 and 2 for both cases are located close to the inlet within the boundaries of its ground neighbouring patches.

As shown in Figures 6.22a and 6.22b for the simple cases there are differences in the calculated velocity values with a maximum difference equal to  $\approx 0.1$  at Probe 2 for both  $U_x$  and  $U_z$ , which can not be considered negligible. In addition for cases `c.2` and `c.2.1` as shown in Figure 6.23a there are large differences for both  $U_x$  and  $U_z$  at Probe 1 where the inlet  $z_0$  is different. Probe 2 has the same values since the inlet  $z_0$  of case `c.2` coincides with the one of `c.2.1`.

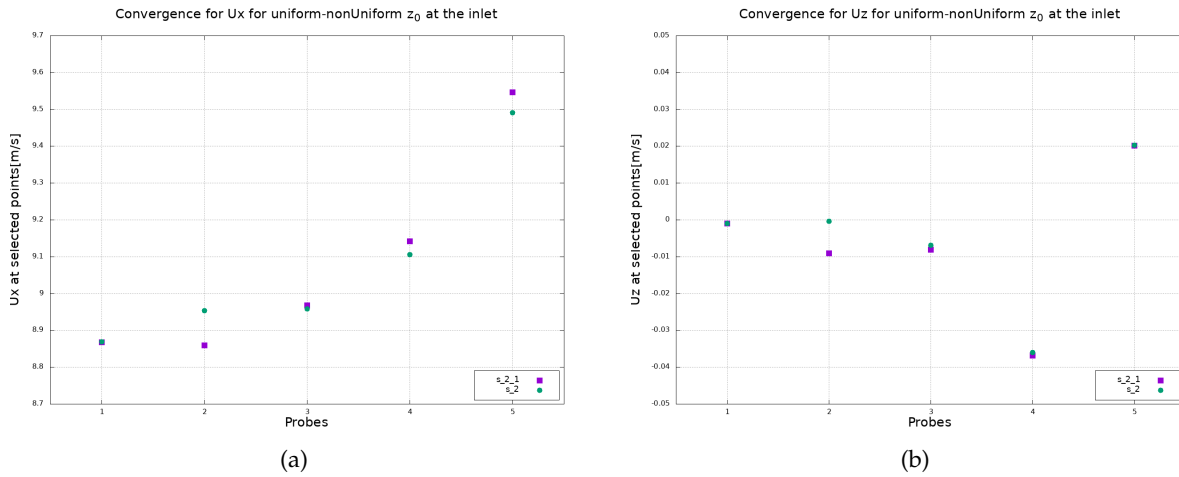


Figure 6.22: `s.2`, `s.2.1` cases comparison based on convergence values for  $U_x$  and  $U_z$ .

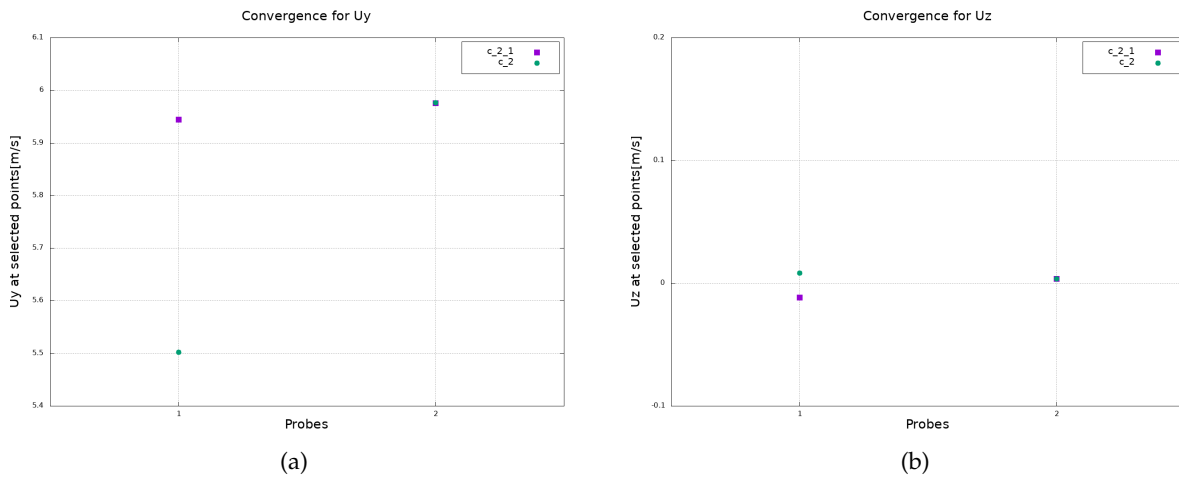


Figure 6.23: `c.2` and `c.2.1` cases comparison based on convergence values for  $U_x$ ,  $U_z$ .



# 7 Conclusions and future work

In this section the research objectives stated in [Chapter 1](#) are addressed. In [Section 7.1](#) answers to the formulated research questions are provided, while in [Section 7.2](#) the limitations and contributions of the study are discussed. Lastly, in [Section 7.3](#) recommendations for future work are provided.

## 7.1 Research Objectives

In order to answer the main research question of this study:

*How can non-uniform roughness length be integrated in a CFD software like OpenFOAM through the use of 3D model semantics?*

A list of sub-questions was formed. Based on the chosen methodology and results of this study we will provide answers to these questions.

Q1: *To what extent can the integration be automated?*

In the current implementation of OpenFOAM roughness length is specified manually, for every generated patch belonging at the bottom of the computational domain. The patches are generated based on the selected OpenFOAM meshing utility. However, due to the diversity of the landuses encountered in the urban environment and depending on the case study it might be required for the user needs to specify multiple patches and thus manually assign the corresponding roughness length values. This implies two issues. Firstly, a geometry for a every patch has to be generated and secondly, separate entries for every patch have to be specified in the boundary conditions. The created methodology aimed at addressing these two issues in an automatic way through the built application. Based on the available data formats for 3D triangulated geometries accepted by the software the `obj` file was selected as the only identified file to allow for the enhancement of spatial data in a straightforward way. In the file all geometries of different landuses were incorporated along with their roughness length values stored in an `mtl` file. The application then assigns the roughness length values as a non-uniform field to the specified patch and writes it to the corresponding boundary conditions folder. However, the process is not fully automated since an external dictionary is still required for the specification of a number of parameters required by the application. Furthermore, in order for the process to assign all the values to the specified patch, there might be a need to test several values for the input parameter `nearDist`. Option `-setZOInlet` is also not fully automated since the user needs to explicitly select the separation point of the geometries so that there are not misassigned cells.

Q2: *How does the modified assignment process of roughness length at the bottom of the domain influences the process and results of the simulation?*

Based on the selected case studies and provided that the input geometry contains no overlaps or gaps the use of a unified geometry file instead of multiple separate files did not seem to have an impact on the mesh generation process, nor have a significant effect on the simulation results. Although deviations were detected both in the mesh composition but also in the calculated flow parameters they were considered negligible.

## 7 Conclusions and future work

Q3: *How does the modified assignment process for non-uniform roughness length at the inlet of the domain influences the process and results of the simulation?*

The option responsible for the assignment of roughness length at the inlet is `-setZOInlet`. The choice of the geometries location can affect the assignment process of the roughness length at the inlet and as a result, the calculation of the flow parameters. For this reason, the geometries should be tailored to the case study. Assigning a non-uniform roughness length at the inlet requires also adjusting the calculation of the turbulence profile at the inlet. The application with option `-setParams` takes care of the calculation of the initial turbulence values based on the standard  $k-\epsilon$  parameters and writes them to the user-specified dictionary file in the boundary conditions folder. Regarding the results of the simulations we observed, based on the two cases used for the testing of the application, deviations between cases with uniform and non-uniform roughness at the inlet. However, this result was only used as an indication that the suggested methodology had an impact on the calculation of the flow parameters, it is not a qualitative assessment of how a non-uniform roughness length assigned at the inlet influences the flow. For this further investigation is required.

Q4: *Which other relevant to CFD parameters could be used as 3D model semantics with the built application?*

Six parameters were selected with the potential to be used as 3D model semantics:

- LAI/LAD: for the representation of trees and the different types of tree species [Deininger et al., 2020].

Parameters related to surface properties as found in Oke [1978]:

- Albedo: to represent the absorptivity of a surface,
- Emissivity: to represent the emitting capability of a surface at a certain temperature,
- Thermal conductivity: used to represent the ability of a material to conduct heat,
- Thermal admittance: used to describe the ability of a surface to absorb and release heat,
- Colour: it is a descriptive characteristic of a surface and it was selected for affecting the radiative properties of a surface [Back et al., 2021].

## 7.2 Discussion

Provided the main objective of this thesis we aimed at creating a methodology that would allow the assignment of non-uniform roughness length automatically, through the use of 3D model semantics. Although, the implementation for the case of option `-setZOGround` is meant to accommodate 3D spatial data, the testing was done based on models that represented flat surfaces. Furthermore, the assignment requires the user to find the optimal value for the search in the octree. Since it was only tested for flat surfaces, the process of defining a suitable value might be a strain for the implementation in the case of input with variable height. In addition, the implementation finds overlaps between the input bottom patch face centers and the triangles of the input geometry based on an octree data structure. This implies that overlapping triangles of different landuses within the input `obj` file will lead to misassigned roughness length values. This was also the case with our results. However despite the fact that the input geometry had overlaps with OpenFOAM's current manual specification of the roughness length, using separate geometries, the same issue was not encountered. This was probably due to the fact that in this case the geometries were meshed separately and any potential overlaps were treated. However, the result was not the same when using the unified geometry.

For option `-setZOInlet` the correct assignment of the values depends on the selection of the boundary edges of the input ground geometries neighbouring the inlet. The need for this arises from the mesh vertical refinement, and it was not possible to overcome. For the tested cases the assignment was successful and there were deviations in the velocity between cases with uniform and non-uniform inlet roughness. However, a more comprehensive assessment regarding the impact of a non-uniform roughness at the inlet on the flow parameters is required.

## 7.3 Future work and recommendations

- For option `-setZ0Ground` further testing is required with 3D surfaces, so that the methodology can be further evaluated and the effect of the `nearDist` parameter better understood.
- The dependency of the current implementation on the `nearDist` parameter could be overcome by using alternative data structures. This would allow for the current implementation to be further automated.
- The option could be further improved by integrating other formats for the input semantics. The methodology presented by [Segersson, 2017] for the storage of non-uniform roughness length in a raster file could be used as reference.
- Further analysis is required regarding the parameters that were proposed to be used as 3D model semantics.
- For option `-setZ0Inlet` further testing is required to assess the impact of possible misassigned values on the simulation results, as well as the impact of non-uniform roughness at the inlet on the calculated flow parameters. The methodology for this option could be further improved by considering a non-uniform roughness at the inlet that is not dependant on the ground roughness inside the domain.



# A Reproducibility self-assessment

## A.1 Marks for each of the criteria

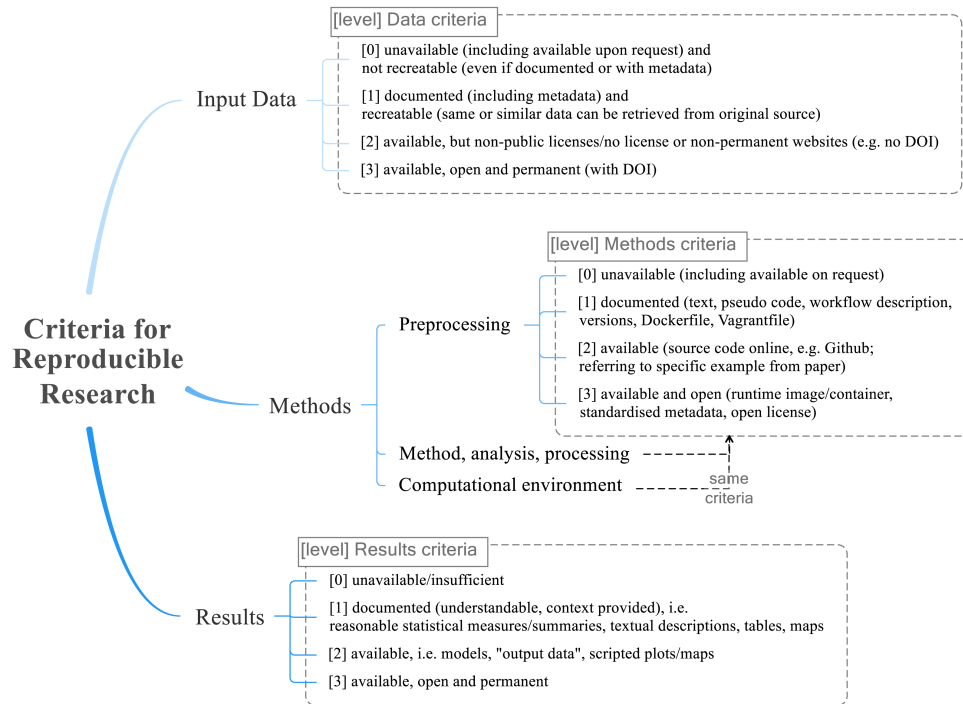


Figure A.1: Reproducibility criteria to be assessed [Nüst et al., 2018].

Criteria	Score
Input data	1
Preprocessing	1
Methods	2
Computational environment	3
Results	1

Table A.1: The self-assigned grading for the reproducibility criteria presented in Figure A.1.

## A.2 Self-reflection

The input data used in this study are available on the GitHub channel. For the TU Delft Campus case only the c.0 was used in the published work of García-Sánchez et al. [2021], all other sub-cases were modified and are available on the GitHub channel. There is a detailed explanation of the preprocessing steps documented in this study. The source code of the employed methodology is available on the GitHub channel. The prototype application was build using OpenFOAM functionality, an open-source

## *A Reproducibility self-assessment*

software, which allows the open use, re-use and redistribution of the code. The results are well documented in the report, however due to their large size, it was not possible to upload them on GitHub. However, they can be reproduced following the steps presented in the current study.

# Bibliography

- Azevedo, J. (2013). Development of procedures for the simulation of atmospheric flows over complex terrain, using OpenFOAM.
- Back, Y., Bach, P. M., Jasper-Tönnies, A., Rauch, W., and Kleidorfer, M. (2021). A rapid fine-scale approach to modelling urban bioclimatic conditions. *Science of The Total Environment*, 756:143732.
- Blocken, B. (2014). 50 years of Computational Wind Engineering: Past, present and future. *Journal of Wind Engineering and Industrial Aerodynamics*, 129:69–102.
- Blocken, B. (2015). Computational fluid dynamics for urban physics: Importance, scales, possibilities, limitations and ten tips and tricks towards accurate and reliable simulations. *Building and Environment*, 91:219–245. Fifty Year Anniversary for Building and Environment.
- Blocken, B., Carmeliet, J., and Stathopoulos, T. (2007a). Cfd evaluation of wind speed conditions in passages between parallel buildings—effect of wall-function roughness modifications for the atmospheric boundary layer flow. *Journal of Wind Engineering and Industrial Aerodynamics*, 95(9):941–962.
- Blocken, B., Stathopoulos, T., and Carmeliet, J. (2007b). Cfd simulation of the atmospheric boundary layer: wall function problems. *Atmospheric Environment*, 41(2):238–252.
- Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., and Ranzuglia, G. (2008). MeshLab: an Open-Source Mesh Processing Tool. In Scarano, V., Chiara, R. D., and Erra, U., editors, *Eurographics Italian Chapter Conference*. The Eurographics Association.
- Deininger, M., von der Grün, M., Pieperreit, R., Schneider, S., Santhanavanich, T., Coors, V., and Voss, U. (2020). A continuous, semi-automated workflow: From 3d city models with geometric optimization and cfd simulations to visualization of wind in an urban environment. *ISPRS International Journal of Geo-Information*, 9:657.
- Ericson, C. (2004). *Real-Time Collision Detection*. CRC Press.
- Franke, J. and Baklanov, A. (2007). *Best Practice Guideline for the CFD Simulation of Flows in the Urban Environment: COST Action 732 Quality Assurance and Improvement of Microscale Meteorological Models*.
- García-Sánchez, C., Vitalis, S., Paden, I., and Stoter, J. (2021). The impact of level of detail in 3d city models for cfd-based wind flow simulations. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLVI-4/W4-2021:67–72.
- Greenshields, C. (2019). OpenFOAM v7 User Guide: 1 Introduction. Library Catalog: cfd.direct Section: User Guide.
- Grimmond, C. S. B. and Souch, C. (1994). Surface description for urban climate studies: A gis based methodology. *Geocarto International*, 9:47–59.
- Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.
- Jayanti, S. (2018a). *Basic Concepts of CFD*, pages 61–130. Springer Netherlands, Dordrecht.
- Jayanti, S. (2018b). *Dealing with Irregular Flow Domains and Complex Physical Phenomena*, pages 271–345. Springer Netherlands, Dordrecht.

## Bibliography

- Jie, Y., Zhan, Q., Xiao, Y., Wang, T., Che, E., Meng, F., and Qian, Y. (2014). Correlation between urban morphology and wind environment in digital city using gis and cfd simulations. *International Journal of Online Engineering (iJOE)*, 10:42–48.
- Kent, C., Grimmond, C., Barlow, J., Gatey, D., Kotthaus, S., Lindberg, F., and Halios, C. (2017). Evaluation of urban local-scale aerodynamic parameters: Implications for the vertical profile of wind speed and for source areas. *Boundary-Layer Meteorology*, 164.
- Kondo, J. and Yamazawa, H. (1986). Aerodynamic roughness over an inhomogeneous ground surface. *Boundary-Layer Meteorology*, 35(4):331–348.
- Lateb, M., Meroney, R., Yataghene, M., Fellouah, H., Saleh, F., and Boufadel, M. (2016). On the use of numerical modelling for near-field pollutant dispersion in urban environments a review. *Environmental pollution (Barking, Essex : 1987)*, 208:271–283.
- Ledoux, H., Biljecki, F., Dukai, B., Kumar, K., Peters, R., Stoter, J., and Commandeur, T. (2021). 3dfier: automatic reconstruction of 3d city models. *Journal of Open Source Software*, 6(57):2866.
- Madeira, D., Montenegro, A. A., Clua, E. W. G., and Lewiner, T. (2011). Gpu octrees and optimized search.
- Montazeri, H. and Blocken, B. (2013). Cfd simulation of wind-induced pressure coefficients on buildings with and without balconies: Validation and sensitivity analysis. *Building and Environment*, 60:137–149.
- Nüst, D., Granell, C., Hofer, B., Konkol, M., Ostermann, F. O., Sileryte, R., and Cerutti, V. (2018). Reproducible research and GIScience: an evaluation using AGILE conference papers. *PeerJ*, 6:e5072.
- Oke, T. R. (1978). *Boundary layer climates*. OCLC: 51200739.
- Pardydjak, E. and Stoll, R. (2017). Improving measurement technology for the design of sustainable cities. *Measurement Science and Technology*, 28.
- Parente, A., Gorié, C., Beeck, J., and Benocci, C. (2011). Improved k- $\epsilon$  model and wall function formulation for the RANS simulation of ABL flows. *Journal of Wind Engineering and Industrial Aerodynamics*, 99:267–278.
- Piringer, M., Grimmond, C. S. B., Joffre, S. M., Mestayer, P., Middleton, D. R., Rotach, M. W., Baklanov, A., De Ridder, K., Ferreira, J., Guilloteau, E., Karppinen, A., Martilli, A., Masson, V., and Tombrou, M. (2002). Investigating the Surface Energy Balance in Urban Areas – Recent Advances and Future Needs. *Water, Air and Soil Pollution: Focus*, 2(5):1–16.
- Piringer, M., Joffre, S., Baklanov, A., Christen, A., Deserti, M., De Ridder, K., Emeis, S., Mestayer, P., Tombrou, M., Middleton, D., Baumann-Stanzer, K., Dandou, A., Karppinen, A., and Burzynski, J. (2007). The surface energy balance and the mixing height in urban areas - activities and recommendations of cost-action 715. *Boundary-Layer Meteorology*, 124:3–24.
- Ricci, A., Kalkman, I., Blocken, B., Burlando, M., and Repetto, M. P. (2020). Impact of turbulence models and roughness height in 3D steady RANS simulations of wind flow in an urban environment. *Building and Environment*, 171:106617.
- Richards, P. J. and Hoxey, R. P. (1993). Appropriate boundary conditions for computational wind engineering models using the k- $\epsilon$  turbulence model. *Journal of Wind Engineering and Industrial Aerodynamics*, 46-47:145–153.
- Samet, H. (1990). *The Design and Analysis of Spatial Data Structures*. Addison-Wesley.
- Schlünzen, K. H., Grawe, D., Bohnenstengel, S. I., Schlüter, I., and Koppmann, R. (2011). Joint modelling of obstacle induced and mesoscale changes—current limits and challenges. *Journal of Wind Engineering and Industrial Aerodynamics*, 99(4):217–225. The Fifth International Symposium on Computational Wind Engineering.



- Segersson, D. (2017). A tutorial to urban wind flow using OpenFOAM. In Proceedings of CFD with OpenSource Software.
- Shaik, S. and Babu, A. (2015). Investigation of unsteady diurnal thermal behaviour of building walls exposed to periodic solar thermal excitation. *Journal of Renewable Energy Science, Technology and Economics*, 1:1–11.
- Shouzhi, C., Qigang, J., and Zhao, Y. (2018). Integrating cfd and gis into the development of urban ventilation corridors: A case study in changchun city, china. *Sustainability*, 10:1814.
- Stull, R. (1988). *An Introduction to Boundary Layer Meteorology*. Atmospheric and Oceanographic Sciences Library. Springer Netherlands.
- Su, Y.-T., Bethel, J., and Hu, S. (2016). Octree-based segmentation for terrestrial lidar point cloud data in industrial applications. *ISPRS Journal of Photogrammetry and Remote Sensing*, 113:59–74.
- Sullivan, B. and Kaszynski, A. (2019). PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK). *Journal of Open Source Software*, 4(37):1450.
- Toja-Silva, F., Kono, T., Peralta, C., Lopez-Garcia, O., and Chen, J. (2018). A review of computational fluid dynamics (cfd) simulations of the wind flow around buildings for urban wind energy exploitation. *Journal of Wind Engineering and Industrial Aerodynamics*, 180:66–87.
- Toparlar, Y., Blocken, B., Maiheu, B., and van Heijst, G. (2017). A review on the cfd analysis of urban microclimate. *Renewable and Sustainable Energy Reviews*, 80:1613–1640.
- Waskom, M. L. (2021). seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021.
- Wieringa, J. (1992). Updating the Davenport roughness classification. *Journal of Wind Engineering and Industrial Aerodynamics*, 41(1):357–368.
- Wiernga, J. (1993). Representative roughness parameters for homogeneous terrain. *Boundary-Layer Meteorol*, 63(4):323–363.
- Willenborg, B., Sindram, M., and Kolbe, T. (2016). Semantic 3d city models serving as information hub for 3d field based simulations.
- Wong, D., Camelli, F., and Sonwalkar, M. (2007). Integrating computational fluid dynamics (cfd) models with gis: An evaluation on data conversion formats. *Proceedings of SPIE - The International Society for Optical Engineering*, 33.
- Zhang, Q., Zeng, J., and Yao, T. (2012). Interaction of aerodynamic roughness length and windflow conditions and its parameterization over vegetation surface. *Chinese Science Bulletin*, 57(13):1559–1567.
- Zhou, Y., Sun, X., Zhu, Z., Zhang, R., Tian, J., Liu, Y., Guan, D., and Yuan, G. (2006). Surface roughness length dynamic over several different surfaces and its effects on modeling fluxes. *Science in China Series D: Earth Sciences*, 49(S2):262–272.

## Colophon

This document was typeset using L<sup>A</sup>T<sub>E</sub>X, using the KOMA-Script class scrbook. The main font is Palatino.

