

AUTOMATIC MESH REPRESENTATION OF URBAN ENVIRONMENTS

By

Babatunde Aliu Atolagbe

Ignatius Fomunung
Professor of Civil Engineering
(Committee Chair)

Arash Ghasemi
Assistant Professor of Civil Engineering
(Committee Co-Chair)

Joseph Owino
Professor of Civil Engineering and
Department Head (Committee Member)

Mbakisya Onyango
Associate Professor of Civil Engineering
(Committee Member)

Weidong Wu
Associate Professor of Computer Science
(Committee Member)

AUTOMATIC MESH REPRESENTATION OF URBAN ENVIRONMENTS

By

Babatunde Aliu Atolagbe

A Thesis Submitted to the Faculty of the University of
Tennessee at Chattanooga in Partial
Fulfillment of the Requirements of the
Degree of Master of Science: Engineering

The University of Tennessee at Chattanooga
Chattanooga, Tennessee

December 2019

Copyright © 2019

Babatunde Aliu Atolagbe

All Rights Reserved

ABSTRACT

A robust watertight mesh generation framework for urban cityscape and waterscape is proposed. The framework, consisting of a set of algorithms implemented in MATLAB, uses geospatial data available from OpenStreetMap and United States Geological Survey repositories, and incorporates Triangle - a popular two-dimensional Delaunay triangulation software - to develop the mesh. For the cityscape component, the facades of the buildings are meshed as structured triangular grids while the roofs and terrains are meshed as unstructured triangular grids using Triangle. For the waterscape component, quadrilateral cells are created based on the requirements of Environmental Fluids Dynamics Code (EFDC) model – a popular modeling platform for environmental fluid flow analysis. The resulting mesh generated is watertight with little human intervention and can serve as a significant preprocessing tool in environmental computational fluid dynamics. Although, there are a few existing methodologies in the literature, most are limited in capacity and are difficult to implement.

Keywords: Grid (or mesh) generation, Cityscape, Waterscape, OpenStreetMap

DEDICATION

This thesis is dedicated to my parents.

ACKNOWLEDGEMENTS

All praises and adorations are due to Allah. I commence my appreciation by acknowledging the guidance of my advisors - Dr Ignatius Fomunung, Dr Arash Ghasemi, Dr Weidong Wu, Dr Joseph Owino and Dr Mbakisya Onyango for their time and indispensable recommendations, as committee members, towards the success of this thesis. Special mention is made of Dr Ghasemi who firstly, accorded me the privilege to conduct this research with him, and secondly, provided the time, guidance and directions to ensure that the objectives are achieved. I also thank Mr Shuvashish Roy for his collaboration on the development of the waterscape mesh representation by providing the needed data as well as advices. The role of the Department of Civil and Chemical Engineering, under the leadership of Dr Joseph Owino, is also greatly acknowledged. Pursuing graduate studies in a department with faculty and staff, like Ms. Karen Lomen, dedicated to catering for the needs of the students could not be better than this.

Very importantly, mention is made of the role of my family throughout this academic sojourn. Their support, financially and emotionally, has been of tremendous influence in the realization of my academic goals. To my friends and colleagues at the University of Tennessee at Chattanooga, including but not limited to Murad Al-Qurishee, Mawazo Fortunatus, Abubakr Zeidan, Kelvin Msechu, Tareq M. Syed, Shuvashish Roy, Maxwell Omwenga, Ammar Elnaiem, Ehsan Ghasemi, and Armel Boutchuen, I thank you all. You have all contributed amazingly to this success.

TABLE OF CONTENTS

ABSTRACT.....	iv
DEDICATION.....	v
ACKNOWLEDGEMENTS.....	vi
LIST OF TABLES.....	ix
LIST OF FIGURES.....	x
CHAPTER	
I. INTRODUCTION.....	1
1.1 Basic Concepts in Mesh Generation.....	3
Grid Cells and Nodes.....	3
Cell Faces and Edges.....	4
Grid Sizes and Cell Sizes.....	5
Grid Consistency with Geometry.....	5
Grid Consistency with Solution.....	6
Grid Organization.....	6
Structured and Unstructured Grids.....	7
Hybrid, Block and Overset Grids.....	8
1.2 Problem Statement.....	9
1.3 Objectives of the Thesis.....	9
1.4 Motivation and Scope of Thesis.....	10
1.5 Thesis Overview.....	10
II. LITERATURE REVIEW.....	12
III. METHODOLOGY.....	25
3.1 Geospatial Data.....	25
3.2 Triangle.....	26
3.3 The Meshing Framework for Cityscape.....	27
Get Geospatial Data.....	28
Extract Building Data and Terrain Elevation.....	29
Mesh Terrain.....	31

Mesh Buildings and Roofs.....	31
Remove Duplicate Grid Nodes	32
Write Grid Output	32
3.4 The Meshing Framework for Waterscape.....	33
Get GIS Data and Extract Coordinates	34
Get Triangular Grids	35
Create Quadrilateral Cells.....	35
Determine Cell Properties and Assign Tags	35
Write Grid Output	37
3.5 Applications of the Program	37
IV. RESULTS	39
4.1 Cityscape Mesh Representation	39
4.2 Waterscape Mesh Representation	46
V. SUMMARY AND CONCLUSION	50
5.1 Summary	50
5.2 Conclusion	52
REFERENCES	53
APPENDIX	
A. CITYSCAPE ALGORITHM.....	56
B. WATERSCAPE ALGORITHM.....	67
VITA.....	74

LIST OF TABLES

1.1 Typical grid cells (Liseikin, 2017).....	4
--	---

LIST OF FIGURES

1.1 A simple bar geometry and its mesh form (drawn using ABAQUS)	2
1.2 Illustration of grid terminology in (a) two-dimension, and (b) three-dimension (Fluent Inc., 2003)	5
1.3 Mesh Types: (a) Structured, (b) Unstructured, and (c) Block-structured (Bern & Plassmann, 1997)	8
2.1 Main steps of the geometry definition and mesh generation process by Gargallo-Peiró et al. (2016); (a) 2D Cadaster mesh (b) topography surface mesh conformal with the cadaster blocks (c) geometry definition with topography and blocks d) final tetrahedral mesh	14
2.2 Typical geometry and mesh representation by Van Hooff and Blocken (2010)	15
2.3 Illustration of selected region and overview of ground plane tessellation by Coirier et al. (2005) with modeled region enclosed within the dashed lines while the solid closed line encloses the resolved regions	17
2.4 Hybrid mesh generated by Kaijima et al. (2013)	18
2.5 Simplification of mesh patches on an irregular roof by Lafarge and Mallet (2012)	19
2.6 A typical object representation using mesh-patches by Lafarge and Mallet (2012)	20
2.7 Meshes based on building-hole (BH) and building-block (BB) approaches by (Schubert et al., 2008)	24
3.1 Flowchart of the cityscape mesh representation	28
3.2 Screenshot of the data export panel on the OpenStreetMap (OSM) website	29
3.3 Typical “.txt” file format of parsed OSM data	30
3.4 Flowchart of the waterscape mesh representation	34
3.5 Description of cell tags foe EFDC model	36
3. 6 Typical mesh of quadrilateral cells with EFDC tags	37

4.1 Unstructured triangle meshes for the terrain (Grid size = 20 nodes; Runtime = 0.1121 secs)	40
4.2 Extrusion of building meshes one step from the footprint - Step1 (Grid size =56 nodes; Run time = 0.1623 secs)	41
4.3 Extrusion of building meshes two steps from the footprint - step=2 (Grid size =73 nodes; Run time = 0.1689 secs)	42
4.4 Building and roof meshes (Grid size = 121 nodes; Run time = 0.1737 secs).....	43
4.5 Watertight mesh for a randomly generated city (Number of nodes = 17,298; Number of cells = 34,477; Run time = 1 minutes, 20 seconds)	44
4.6 Google earth view of layout of 459 buildings in Nashville, Tennessee, USA	45
4.7 Mesh representation of 459 buildings, and terrains, in Nashville, Tennessee, USA (Number of nodes = 48,726; Number of cells = 89,999; Run time = 10 minutes, 53 seconds).....	45
4.8 Google earth view of a part the Tennessee River	46
4.9 Unstructured triangular grids for wet area of waterscape	47
4.10 Quadrilateral cells representing the waterscape.....	47
4.11 Super-imposed mesh representation of the Tennessee River (Grid size = 40,000 quadrilateral cells, 1,553 triangle cells (Total 41,553), Run Time = 7 minutes, 29 seconds).....	48
4.12 A blown-out section of the meshed waterscape.....	49
4.13 An array of EFDC tags for a part of the waterscape mesh	49

CHAPTER I

INTRODUCTION

For decades, numerical simulations have been used to solve complex problems in many different fields including engineering, medicine, physical sciences, earth sciences, life sciences as well as urban physics, by solving equations that represent the physical processes. In urban physics – the science and engineering of physical processes in urban areas, although there has been growing urbanization and modernization which has, in turn, constantly created many complex challenges for scientists and researchers to analyze, it has equally been possible to simulate and understand these complex problems with the use of various computational methods, including numerical simulations. Numerical simulations constitute one and, in fact, most powerful, of three major approaches of tackling problems in urban physics, with the other two approaches including field measurements and wind-tunnel measurements (Blocken, 2015). Numerical simulations or modeling can provide detailed information on the essential physical variables but careful consideration is required in the geometrical implementation of the model, mesh generation, solution techniques and interpretation of the results (Blocken, 2015).

Mesh (or grid) generation is an integral part and, in fact, the first process of numerical simulations. A mesh is a discretization of a geometry into small, and simple, shapes such as triangles and quadrilaterals for two dimensions, or tetrahedra and hexahedra in three dimensions (Bern & Plassmann, 1997). It can also be defined as a preprocessing tool on which different physical continuous quantities are described by discrete functions for approximation of

differential equations using algebraic relations for discrete values that are then analyzed numerically (Liseikin, 2009). As an illustration, Figure 1.1 depicts a physical geometry of a simple bar and the corresponding mesh representation.

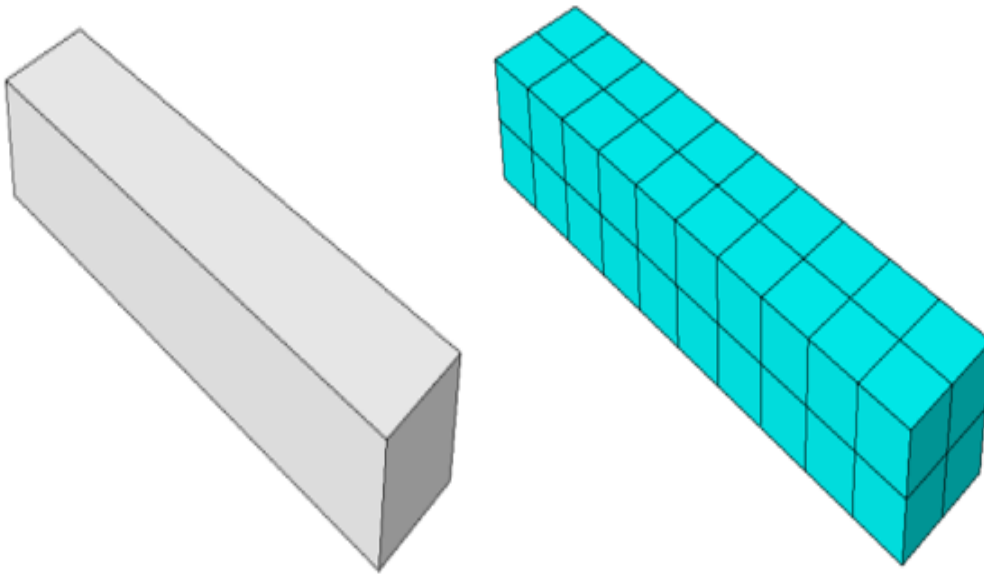


Figure 1.1 A simple bar geometry and its mesh form (drawn using ABAQUS)

Meshes (or grids) are needed because the mathematical models for physical problems such as finite element method, finite volume method and finite difference method actually solve the variables at the discrete nodes and/or cells making up the mesh (LearnCAx, "n.d."). Moreover, the efficiency of any numerical solution depends on the quality and quantity of the mesh (Liseikin, 2009). While the quality of mesh, measured by shape and size of cells, impacts the overall accuracy of the analysis, the quantity of mesh, measured by number of mesh nodes, determines the overall computational cost (i.e. runtime and memory demand) which can easily become extremely high for complex geometries with very fine details (Kaijima, Bouffanais,

Willcox, & Naidu, 2013). Hence, a well-constructed mesh is one that uses appropriate shape in reasonable quantity to idealize the physical geometry.

Furthermore, mesh generation is labor intensive (Kaijima et al., 2013) and can take considerable time of the entire simulation process. According to Kim (2014), the time required to generate efficiently high-resolution mesh for CFD analysis of buildings considered in their study constituted significant part (approximately 80 percent) of the computational time for the entire modeling process. As such, it is desirable that the meshes are generated within considerable time since the runtime is usually a crucial factor for measuring the efficiency of simulations.

1.1 Basic Concepts in Mesh Generation



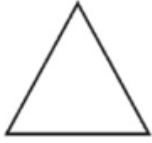



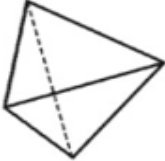
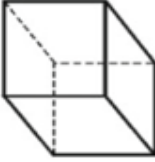
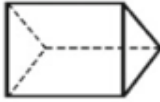
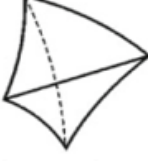
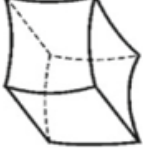

The following discussion of fundamental concepts related to mesh generation are based on the explanations in (Liseikin, 2006, 2009, 2017) and is presented for understanding of this thesis.

Grid Cells and Nodes

Grid nodes are specified points of the meshed domain which connects the grid cells. A group of nodes (grid points) defines cells. Grid cells are collections of small standard dimensional volumes called control volumes that make up the mesh by covering the entire domain or surface without gaps or overlaps. In one dimension, the cell is a closed line or segment with boundaries consisting of two points called vertices. In two-dimension, the cell is a two-dimensional simply connected domain, usually in form of triangles or quadrilaterals, whose boundaries are divided into a finite number of one-dimensional cells called edges (e.g. three segments for triangular cells). In three-dimension, the cell is defined as a simply connected three-

dimensional polyhedron (usually tetrahedrons or hexahedrons), whose boundary is divided into finite number of two-dimensional cells called faces (e.g. four triangular cells for tetrahedron and six for hexahedrons). Table 1.1 presents the cells in one, two and three dimensions.

Table 1.1 Typical grid cells (Liseikin, 2017)

Level	Shape	Straight Form	Curved Form
One-Dimensional	Line		
Two-Dimensional	Triangle		
	Square (Quadrilateral)		
Three-Dimensional	Polyhedrons	  	  

Cell Faces and Edges

Cell faces are boundaries of a three-dimensional cell. Cell edges, on the other hand, are boundaries of two-dimensional cells or of cell faces of three-dimensional cells.

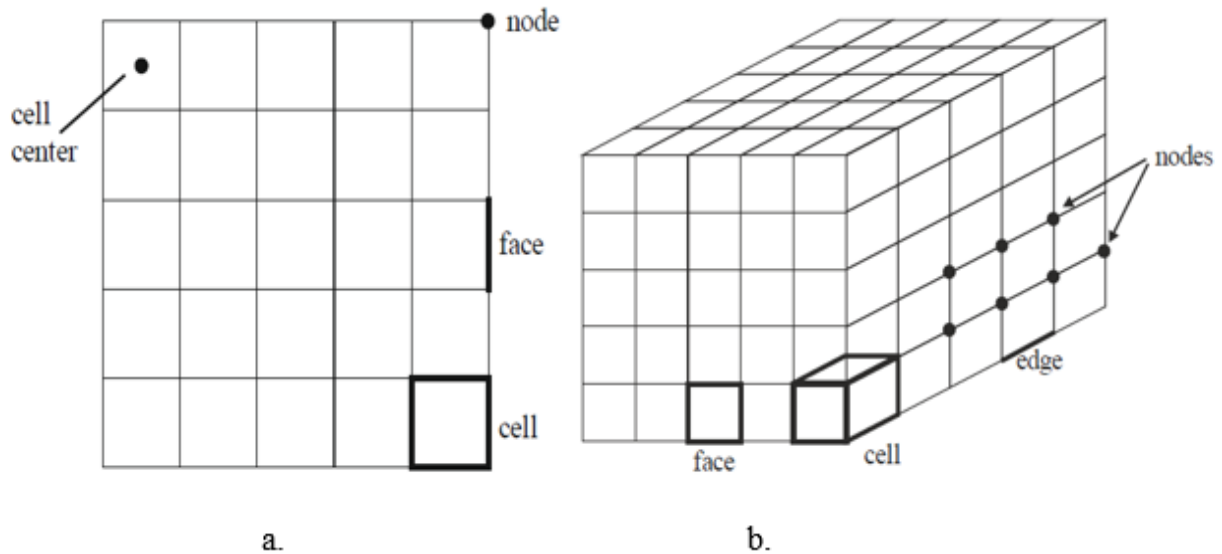


Figure 1.2 Illustration of grid terminology in (a) two-dimension, and (b) three-dimension (Fluent Inc., 2003)

Grid Sizes and Cell Sizes

These are properties of grids that define the accuracy of the mesh. Grid size is indicated by the number of nodes, while cell size is defined as the maximum length value of the cell edges in all directions. It is desirable that the mesh generation process utilizes techniques that increase the number of nodes while reducing the edge lengths in such a way that they approach zero as the number of grids approaches infinity. In fact, small cells are desired to achieve accurate approximation of the equations and to study the convergence of numerical codes.

Grid Consistency with Geometry

This is a measure of the extent of compatibility of the mesh with the geometry of the physical domain. The accuracy of the numerical solution of partial differential equation as well as of the interpolation of a discrete function is greatly affected by the consistency of mesh with

geometry. As such, the nodes must approximate the original geometry appropriately by ensuring that the distance between any point of the domain and the nearest grid node is not too large. Also, the grid should be boundary-fitting (or boundary conforming) by ensuring that there is enough number of nodes along the boundary so that the edges (in two dimensional cells) and faces (in three dimensional cells) formed by the nodes model the boundary of the physical domain efficiently to enable easily and correct application of boundary conditions.

Grid Consistency with Solution

This is a measure of the extent to which the mesh representation of the geometry yields accurate solution. In fact, the arrangement of the nodes and form of the cells should be dependent on the features of the physical solution. Usually, the nodes are arranged in some preferred and consistent directions such as vector fields. In cases where a nonuniform variation of the solution exist, clustering of the nodes in regions of high gradients may be required such that these areas have finer resolution. Furthermore, such clustering may also be required because uniform refinement of the entire domain may have significant computational cost (i.e. runtime and disk space) for complex multidimensional problems. This is the case for problems whose solutions have very rapid variations of localized regions in which without grid clustering in such localized regions, there may be loss of significant features and hence a compromise to the accuracy of the solution.

Grid Organization

Grid organization defines the arrangement of the nodes and cells of grids and it facilitates the procedure for formulating and solving the algebraic equations substituted for the differential

equations. Organization of grids should identify neighboring points and cells. Depending on the local organization of grid nodes, they can be classified fundamentally as structured and unstructured grids. Further subdivisions to these two classes, which combine features of both, are block-structure, overset, and hybrid.

Structured and Unstructured Grids

A grid is considered structured (Figure 1.3a) if the local organization of the grid nodes and the form of the cells do not depend on their position but rather determined by a general rule, and the connectivity of the grid is implicitly taken into consideration. On the other hand, unstructured grid (Figure 1.3b) is one whose neighboring nodes, constituting the grid, have varying connections from point to point and the connectivity of the grid is explicitly defined by a suitable data structuring procedure. Unstructured grids are used in many applications, especially those involving very complex geometries, since they can better approximate the physical geometry than structured grids which lack the flexibility for complicated geometries. However, unstructured grids are relatively disadvantaged algorithmically because of more computational work involved creating them as well as data management problem in organizing and numbering the nodes, edges, faces and cells of the grid which also requires more cost (memory) to store the connectivity.

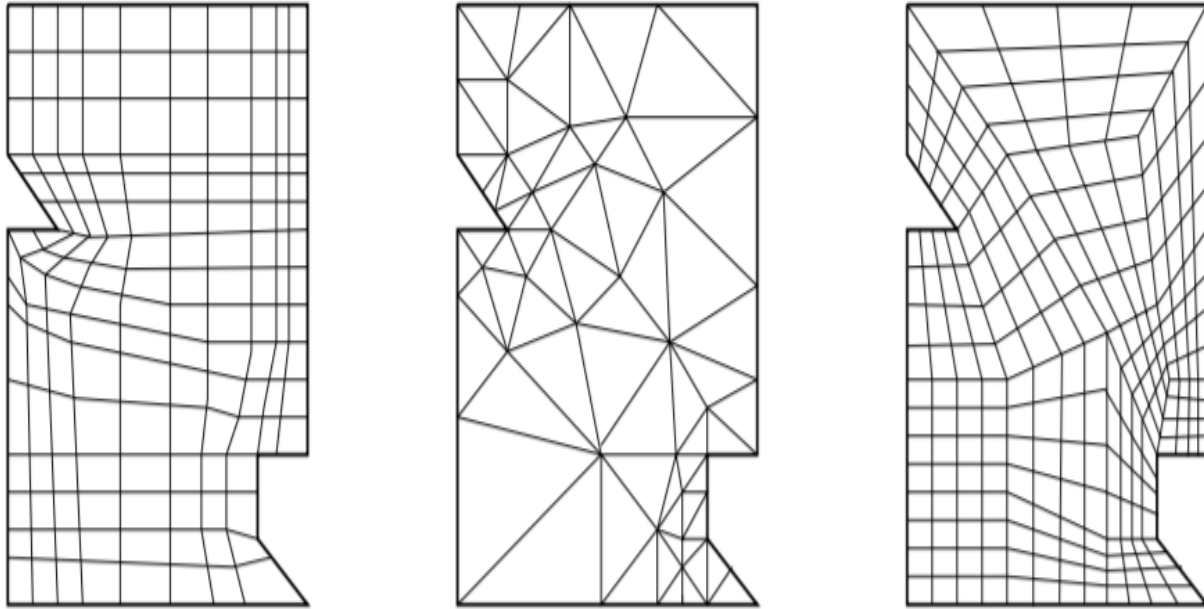


Figure 1.3 Mesh Types: (a) Structured, (b) Unstructured, and (c) Block-structured (Bern & Plassmann, 1997)

Hybrid, Block and Overset Grids

These kinds of grids are formed by combining small structured meshes in varying arrangement thereby yielding unstructured grids. Block-structured or multi-block grids are formed by dividing the physical domain into subdomains, without holes or overlaps, which may be considered as the cells of a coarse unstructured grid but with each subdomain or block meshed as a structured grid. The resulting block-structured grid is regarded as locally structured at the level of each block, but globally unstructured as a collection of all blocks. This concept enables the use of different structured grids in different regions, thus allowing the most suitable grid configuration to be used in each region and providing more flexibility in complex geometries, especially for those involving heterogeneous or non-uniform physical problem, than ordinary structured grids would. Overset grids are like block-structured except that unlike in block-structured, the blocks are made to overlap in overset grids thus simplifying the problem of

selection of the blocks covering the physical domain. Hybrid grids are developed through the combination of properties of both structured and unstructured grids for complex geometry by joining structured and unstructured grids in different regions without overlap.

1.2 Problem Statement

For Computational Fluids Dynamics (CFD) simulations, there are existing grid generation software and programming packages such as MESH2D and GMSH, and modelling software equipped with generation programs such as ANSYS and FLUENT. However, these programs are limited in extent, especially for works requiring multiple complex geometries to be meshed concurrently within limited time. In most cases, researchers are compelled to create mesh generation programs that best represent their simulation domains. For urban environmental simulations, creating meshes for the numerous complex geometries of the cityscape is a major challenge but very few methodologies exist in literature and even these few methodologies are often expensive due to the cost of acquiring needed data, time-consuming because of the time for sourcing the different data components for the geometry, painstaking due to difficulties in extracting details from such data especially because implementing the procedure often require high level of craftsmanship by the user.

1.3 Objectives of the Thesis

The objective of this thesis is to develop programming framework for mesh representation of urban cityscape (i.e. a landscape of the hard features including buildings and land) and waterscape (i.e. a landscape with water as a major feature). The aim is to utilize existing mesh generation techniques and tools to develop algorithms that can be implemented in

programs such as MATLAB for automatically generating mesh representation of any arbitrary city- or water-scape using cheaply available resources (i.e. geospatial data) such that the meshes can then be used for relevant applications like computational environmental fluids dynamics analysis or any other applications where mesh representations of environmental geometric arrangements are desired.

1.4 Motivation and Scope of the Thesis

This thesis is the culminating output of the first part of a research at the Department of Civil Engineering which seeks to develop an optimization framework for energy-aware deployment of drones for civil applications. The underlying motivation for the thesis is the need for a watertight mesh representation of urban cityscape for CFD simulation of wind around buildings to establish regions of wakes such that can be used to develop an energy-optimized path planning framework for flying drones. Additionally, it became necessary to create mesh for waterscape for another research on hydrodynamic flood modeling in the City of Chattanooga. While the motivation for these researches is to develop simulation frameworks to solve urban fluid dynamics problems, the scope of this thesis is limited to the generation of meshes only and does not include details of the computational fluid dynamics analysis.

1.5 Thesis Overview

This thesis focuses on the generation of watertight mesh for urban environment fluid dynamics simulations. Chapter I presents research background, objectives and motivation, as well as brief discussions of basic concepts of mesh generation pertinent to this research. Chapter II presents the literature review of existing methodologies for mesh representation of urban

environments. Chapter III discusses the mesh generation programming framework developed for the thesis. In chapter IV, illustrations of the major procedures in the algorithms are first presented. Thereafter, outputs of the program when tested with OpenStreetMap cityscape data of a part of Nashville, Tennessee and waterscape data of a part of the Tennessee River near Chattanooga are presented. Then a brief discussion is presented on the applications of the program. Finally, in Chapter V, summary, challenges, as well as, conclusions from the thesis are discussed.

CHAPTER II

LITERATURE REVIEW

Representing urban geometries in the form of computational meshes is a very important part of numerical simulations such as computational fluid dynamics modeling for urban environment. However, different researchers tend to adopt mesh generation methodologies that suit their unique applications. For example, Pointwise Inc, a commercial mesh generation software company, asserts that the focus of their research and development of meshing software are customer purpose driven (Karman, Wyman, & Steinbrenner, 2017). Regardless of the motivation and approach adopted, many developers of mesh generation framework for environmental simulations are often concerned with the determination of the most suitable data sources, as well as the features, that significantly affect the solution most. According to Gargallo-Peiró, Folch, and Roca (2016), computational meshes for urban flow simulations, for example, require idealized geometrical description of the domain of interest so as to reach an equilibrium between simplicity and realism. In other words, the geometric model should reflect the major aspects influencing the flow, such as streets, terrain, buildings, while removing smaller features, such as fire hydrants, that cannot be physically modeled (Gargallo-Peiró et al., 2016). Typically, cityscape model generation often depends on Geospatial Information System (GIS) data for building footprints, height and other relevant information as needed. Furthermore, the resulting mesh developed from the model are required to be watertight, well defined around buildings, and composed of well-shaped elements (Gargallo-Peiró et al., 2016). By watertight, it

is inferred that the meshes should not have overlapping nodes such that would cause error in the simulations.

In an attempt to develop a framework for urban flow mesh generation, Gargallo-Peiró et al. (2016) proposed a three-step methodology for representing urban geometries for unstructured mesh generation with data obtained from three sources including the city cadaster (i.e. an information system containing a record of land parcels including geometric description of land parcels and record of interest such as the ownership), a Digital Elevation Model (DEM) of the desired domain within the cadaster, and Light Detection and Ranging (LiDAR) of the domain defined by the cadaster. A method for generating idealized surface geometry of a city landscape for computational meshes was developed. Two main geometrical features were targeted namely: the city blocks and terrain (i.e. the spaces surrounded by streets on which buildings are situated). A two-dimensional mesh of the cadaster is first generated such that all elements are tagged according to street and block regions. Then, the terrain surface mesh is generated, using a Digital Elevation Model (DEM) of the city landscape and lastly, the block facades is extruded to the respective heights retrieved from the LiDAR data of the city. The resulting tetrahedral mesh of the urban geometry representation was used for non-viscous flow or transport simulations of the city of Barcelona in Spain. Figure 2.1 summarizes the procedure. This procedure is limited by the expensive cost of acquiring DEM and LiDAR information.

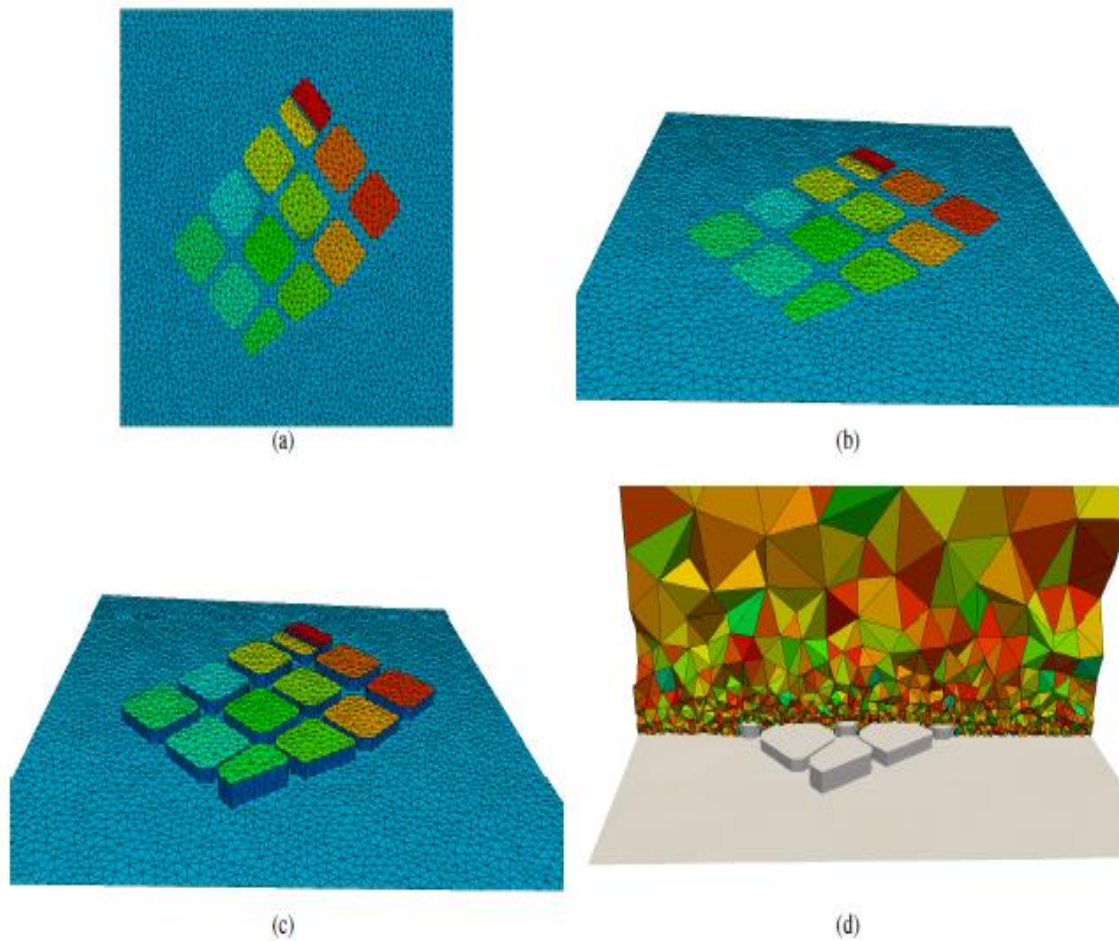


Figure 2.1 Main steps of the geometry definition and mesh generation process by Gargallo-Peiró et al. (2016); (a) 2D Cadaster mesh (b) topography surface mesh conformal with the cadaster blocks (c) geometry definition with topography and blocks d) final tetrahedral mesh

In another work, Van Hooff and Blocken (2010) proposed a CFD modelling approach for urban wind flow and indoor natural ventilation modelling using the Amsterdam Arena Stadium in Netherlands as a case study. The framework involves an automatic modelling of complex geometries with full control over grid quality and resolution while also providing a way to easily implement changes in the model geometry and grid for parametric studies such as the case study. The computational meshing utilized a unique procedure that efficiently generates the geometry

and mesh simultaneously thereby improving the quality of mesh in the vicinity near to the stadium which is considered highly essential for the simulations of urban wind flow and indoor natural ventilation. The mesh generation procedure consists of a series of extrusions which involves creation of the geometry and grid based on geometrical translations and rotations of a pre-meshed 2D cross-sections (Van Hooff & Blocken, 2010). Figure 2.2 shows a mesh representation generated by this procedure. The procedure may be labor intensive and requires high skilled understanding of geometric modeling and meshing techniques.

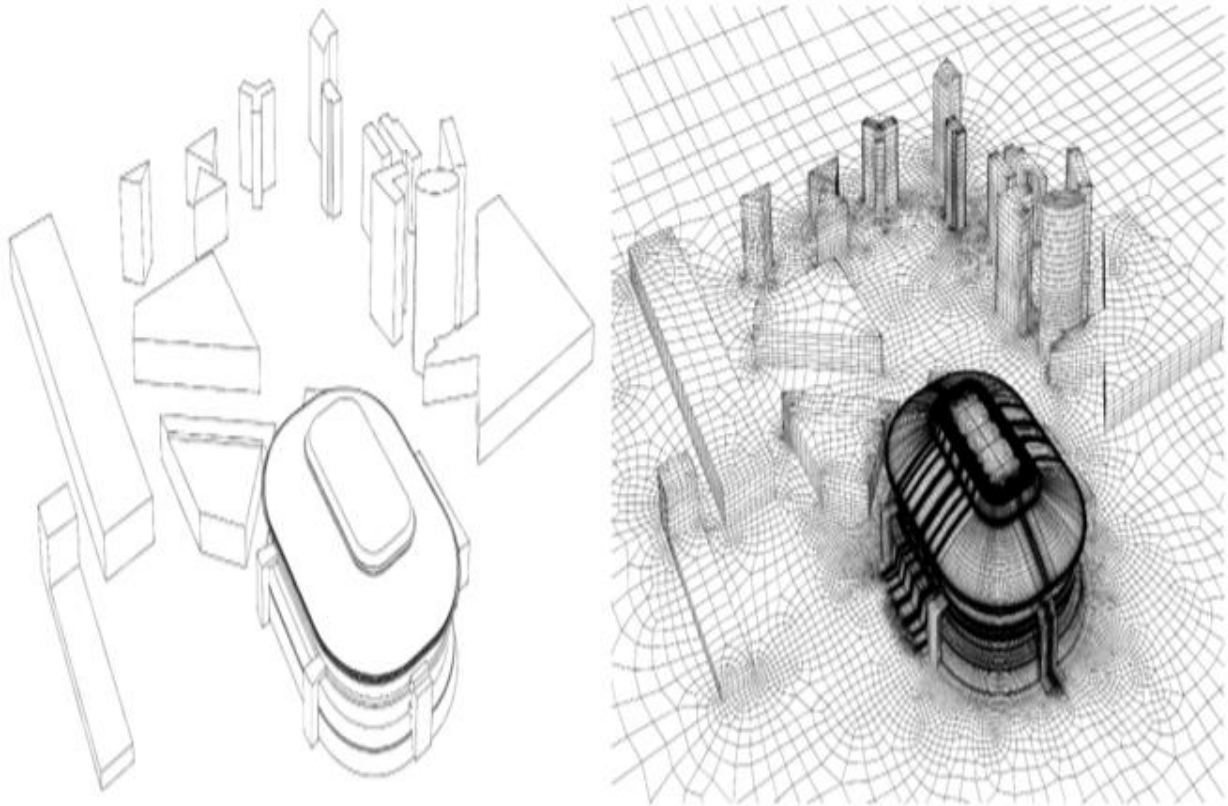


Figure 2.2 Typical geometry and mesh representation by Van Hooff and Blocken (2010)

Coirier, Fricker, Furmanczyk, and Kim (2005) also developed meshes in their work involving the use of a computational fluid dynamics approach to investigate and model urban area transport and dispersion. Their work focused on modeling of wind fields, turbulence fields and dispersion of chemical biological radiological and nuclear substances in urban areas on the building to city blocks scale by solving the Reynolds-Average Navier-Stokes equations. The meshes were developed by adopting a specially crafted cityscape CFD volume mesh framework based on a commercially available mesh software called CFD-Micromesh. According to (Coirier et al., 2005), CFD-Micromesh is an automated program for building three-dimensional geometries and generating mesh from given layouts. By customizing the program and GIS data import, CFD volume meshes of cityscapes were created using a four-step framework. In this framework, the GIS data is first used to construct a voxel solid model whose resolution can be easily controlled by changing the voxel sizes. A voxel is a single data point on a regular grid within a three-dimensional space. Once the solid model is established, the boundaries across which voxel properties changes are projected onto a constant z plane in such a way that the boundaries are vectorized, hence yielding sets of polygons representing the projections. Thereafter, the vectorized model is tessellated (i.e. meshed) into sets of triangular and quadrilateral faces. And finally, the tessellated constant z plane is then extruded along z, yielding volume meshes in forms of prisms and hexahedra. Figure 2.3 presents the illustration of a given region as well as an overview of ground plane tessellation from their approach.

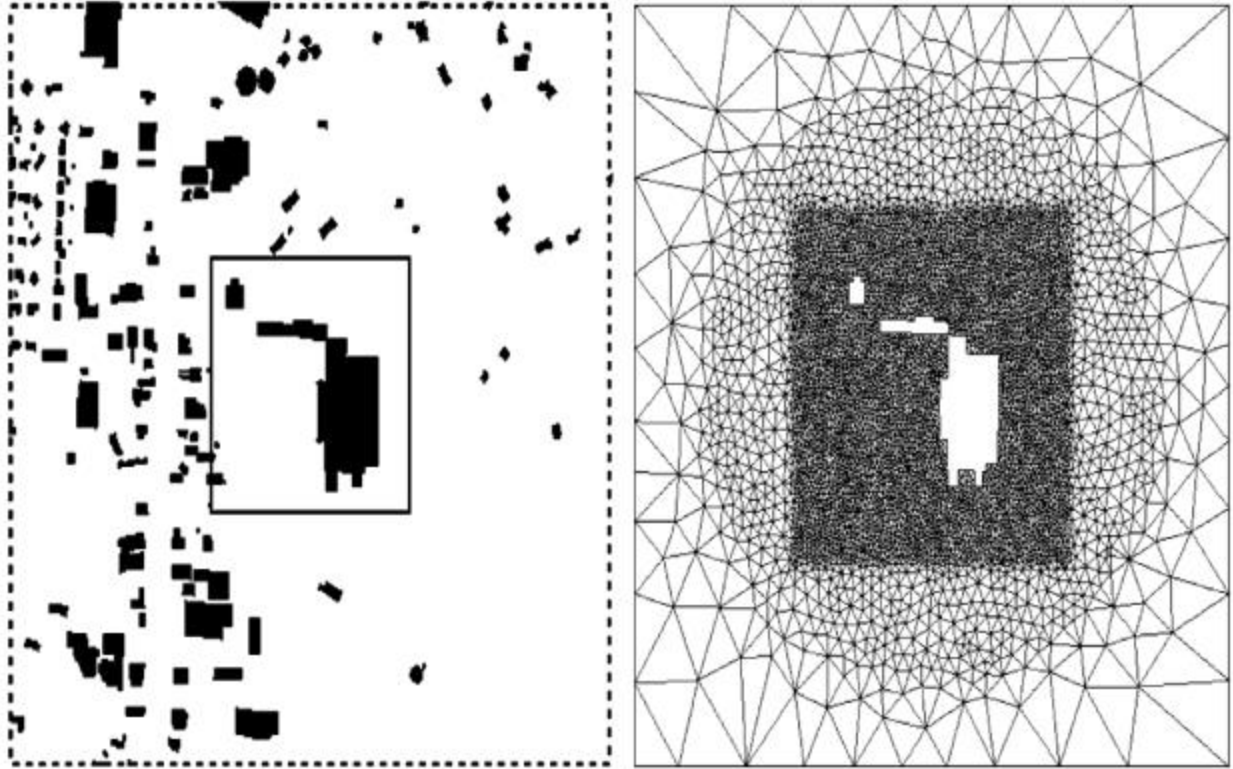


Figure 2.3 Illustration of selected region and overview of ground plane tessellation by Coirier et al. (2005) with modeled region enclosed within the dashed lines while the solid closed line encloses the resolved regions

Exploring computational fluid dynamics for architectural design, Kaijima et al. (2013) identified mesh generation as one of two bottlenecks of their framework for indoor analysis of wind or airflow in buildings during the conceptual design phase. In this framework, a hybrid mesh (i.e. a combination of structured and unstructured mesh) generation technique was adopted to ensure acceptable level of accuracy while enabling the meshing of complicated shapes and fine geometries. The hybrid mesh adopted facilitated simple and reduced number of iterations of a single conceptual design with reduced mesh cells, and hence increasing efficiency in terms of accuracy and cost. The mesh was used for a bus-stop canopy case study and the results were satisfactory. However, it was hinted that, even though the hybrid mesh in the framework reduced

the number of nodes to approximately 4 percent of the original unstructured mesh, the meshing remained time-consuming to some extent. Figure 2.4 depicts the mesh generation procedure for mesh generation of inner and outer surrounding of a building.

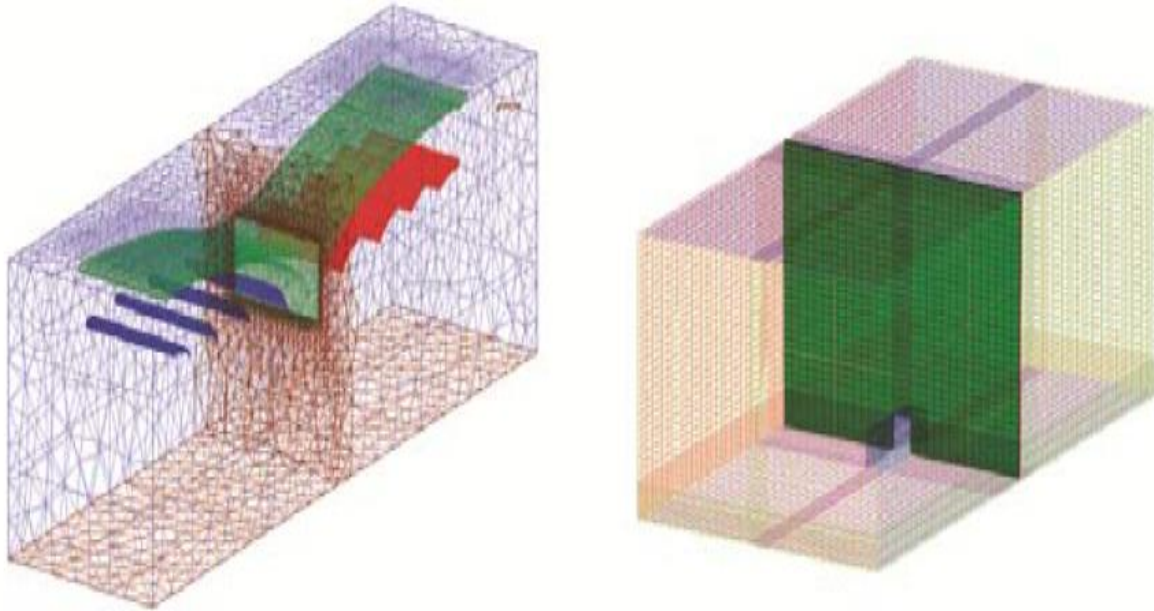


Figure 2.4 Hybrid mesh generated by Kaijima et al. (2013)

In another study, Lafarge and Mallet (2012) developed a robust method for city modelling by reconstructing concurrently the buildings, trees and topologically complicated grounds from three-dimensional point cloud data. For the buildings, the approach involved a hybrid representation combining geometric three-dimensional primitives such as planes, cylinders, spheres or cones for the standard roof sections, and mesh-patches describing the irregular roof components. While the primitive geometries were represented by polyhedral structures obtained from a label map, the mesh-patches were created by triangulating the cells labelled as roof with a Z- component associated to the XY-center of the cells. A standard mesh

simplification algorithm by Garland and Heckbert (1997) was then used to obtain more coarse and compact mesh representations of buildings and the facades were obtained as a vertical projection of the building contours on the ground. Figure 2.5 shows the simplification of the mesh patches of an irregular roof by Lafarge and Mallet (2012).

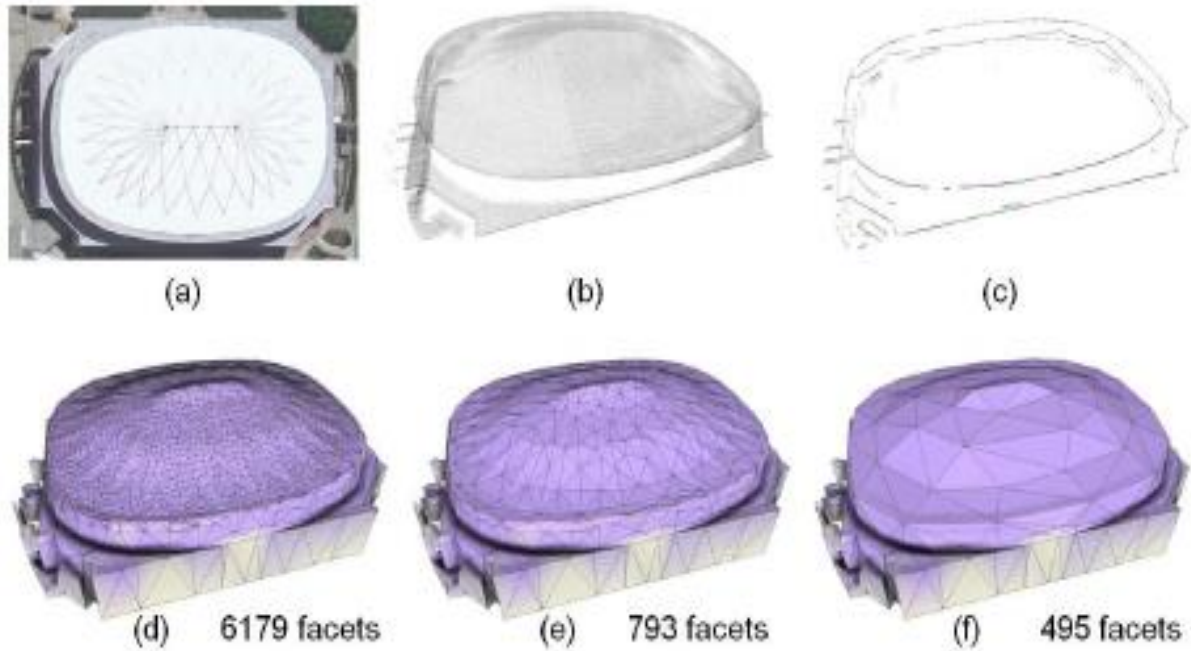


Figure 2.5 Simplification of mesh patches on an irregular roof by Lafarge and Mallet (2012)

For the ground, however, a unique meshing procedure was adopted such that the surface is continuous. A grid of 3D-points were created from spatial sub-sampling of the cells labeled as ground, and the mesh is simplified using the same algorithm (Garland & Heckbert, 1997) used for mesh-plates of the buildings and non-planar primitives (Lafarge & Mallet, 2012). Figure 2.6 shows the complete object representation with ground meshing.

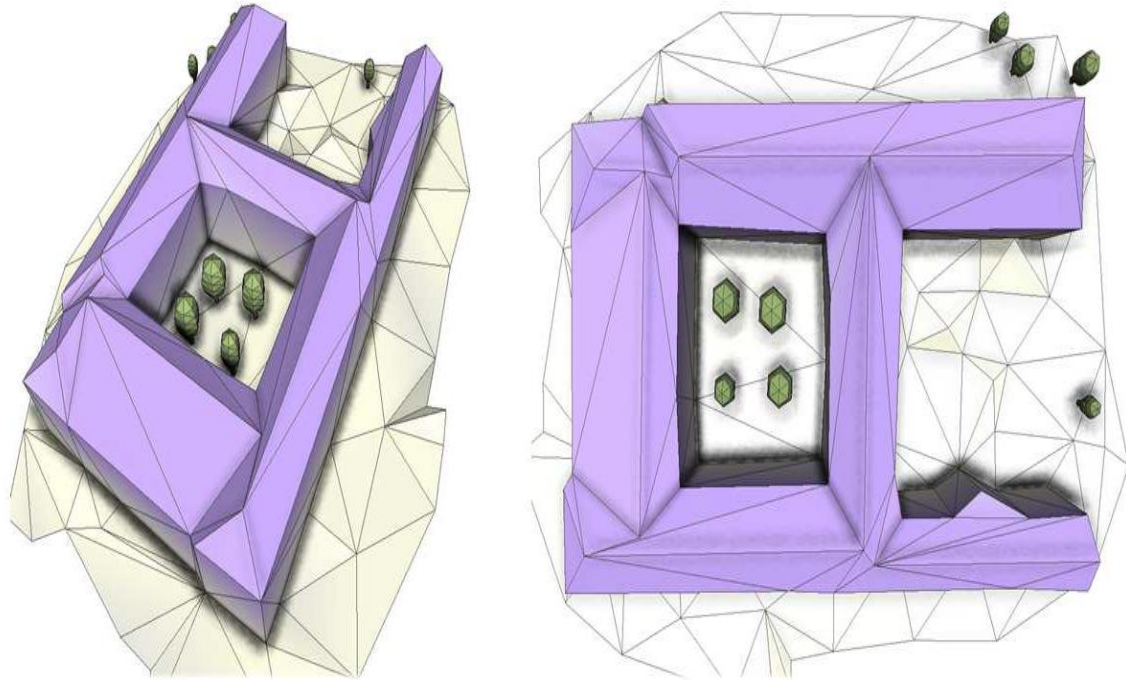


Figure 2.6 A typical object representation using mesh-patches by Lafarge and Mallet (2012).

For hydrodynamic applications, mesh representations of water bodies are also significant for several modeling problems. The use of numerical modeling to solve the intricate hydrodynamic processes in water problem can provide very useful information such as description of circulation, water levels, velocity, temperature variations and stratification processes and their effects on the transport of pollutants and water quality within a water body (Cedillo, 2015). Examples of hydrodynamics models that has been used in recent times are CH3D-z, MIKE 3, Princeton Ocean Model (POM), HEC-RAS and EFDC (Cedillo, 2015).

Environmental Fluid Dynamics Code (EFDC) model has remained popular amongst researchers and has featured in many different applications since its development in 1992 (Liu, 2007). The EFDC model is a comprehensive three-dimensional tool, widely recognized simulation platform and a multi-task, highly integrated modular computational environmental fluid dynamics package which can be used for understanding and predicting the environmental

fluid flows with transportation and mixing associated dissolved or suspended materials, as well as for modeling pollutants and pathogenic organism transport from point and non-point sources (Cunanan & Salvacion, 2016; Wang et al., 2014). The model, originally developed at the Virginia Institute of Marine Science, comprises of an advanced three-dimensional surface water modeling system for hydrodynamic and reactive transport simulations of rivers, lakes, reservoirs, wetland systems, estuaries, and the coastal ocean (Cunanan & Salvacion, 2016; J. M. Hamrick & Mills, 2000). Although the model is an open source code in the public domain (Cedillo, 2015; Cunanan & Salvacion, 2016), it is used widely by universities, governmental agencies and engineering consultants within and outside the USA and is maintained and continuously developed by Tetra Tech Inc. with primary support from the United States Environmental Protection Agency (Cunanan & Salvacion, 2016; J. Hamrick, 2002).

As with any numerical simulation solver, hydrodynamic model involves three stages including the pre-processing phase which includes development and organization of input data such as grids, processing phase where the main simulation occurs, and post-processing phase which basically involves visualization of the outputs of the simulation. Thus, the role of grid generation is also very crucial to the realization of accurate simulation of the problem.

Creating and assessing a suitable grid generation program for EFDC model has been quite challenging. In fact, according to Tetra Tech Inc, the maintainer of EFDC, existing grid generation software generally requires a lots of user experience and artistry to support the mathematical grid generation program (Xiong, 2010). Moreover, existing tools can in certain cases be erroneous or even expensive for personal or non-commercial use.

A popular grid generation software for Environmental Fluid Dynamics Code is GEFDC grid generation program. This program is simply a FORTRAN code which is developed and

capable of producing structured rectangular and curvilinear meshes (Alarcon, McAnally, & Pathak, 2012; Tetra Tech Inc, 2002) and it was originally designed by Tetra Tech Inc as the complementary grid generation tool for EFDC models. However, the GEFDC usually requires modification to suit the model and the success of this depends on the level of skills and experience of the user (Xiong, 2010).

Another existing grid generation, the visual orthogonal grid generation (VOGG), involves a FORTRAN implementation of a novel physical domain grid generation algorithm, a Windows/GIS based interface for creating necessary input files and displaying output results, and several utility programs (Xiong, 2010). It was developed as a component of the United States Environmental Protection Agency (US EPA) Region 4 Total Maximum Daily Load (TMDL) modeling toolbox and specifically supports curvilinear-orthogonal grid generation for the Environmental Fluid Dynamics Code (EFDC) (Xiong, 2010). But, a variety of ASCII text output files are created which readily allow grid information to be processed and reformatted for other hydrodynamic and transport models employing both orthogonal and non-orthogonal curvilinear grid formulations (Tetra Tech Inc, 2002). Although the VOGG is popularly known and adopted, it is considered unstable and errors occur often according to Xiong (2010) .

Another grid generation software that exists, and in fact, has been a more dominantly used is the EFDC Explorer. Created by Dynamic Solutions LLC, the program has a user interface and a grid generation tool (Xiong, 2010) thus making it simple and easier for users to work with. However, this tool is a commercial software and may not always be affordable to non-commercial (i.e. industrial) users.

Apart from developing computational grids for EFDC models, grids can generally be created for other models as necessary since other modeling platforms exist that are different from

EFDC. Schubert, Sanders, Smith, and Wright (2008) developed unstructured mesh generation framework for hydrodynamic modeling for urban flooding. Their methodology focuses on strategies for effective integration of geospatial data for unstructured mesh generation, building representation and flow resistance parameterization. Mesh generation was done using Triangle to generate Delaunay triangle meshes with inputs to Triangle including a polygon defining the outer boundary of the domain being considered, additional polygons defining interior boundaries called mesh holes, polylines that fix break-lines in the mesh, and points that fix vertices. In the development of the meshes, three data sources were utilized. These include Light Detection and Ranging (LiDAR) terrain height surveys, aerial imagery, and vector datasets which includes the building footprint polygons. Three unique unstructured meshing techniques were developed, namely building-hole method (BH), building-block method (BB), and no-building method (NB), all according to the authors can be viewed as three options for urban flooding with different pre-processing demands. In the building-hole method, the first step is the outer boundary definition, from LiDAR data, that follows the course of the river such that is extended beyond the floodplain, to avoid interference between boundary and actual river flow. Thereafter, the interior hole definition is done by extracting location and shape of buildings from Digital Surface Models (DSMs) or aerial photography and converted to vector data structures or polygons using GIS software or platform. With the interior and exterior boundary defined, the data are used as input by Triangle to generate the required meshes. Finally, a Digital Terrain Model (DTM) is used to interpolate terrain heights at mesh nodes or vertices. For building-block (BB) method, only exterior boundary is considered as input to Triangle but the building footprints and heights (or blocks) are burnt into the Digital Terrain Models (DTM) prior to mesh interpolation. Similarly, in no-building (NB) method, only the exterior boundary data is considered for mesh generation

while a bare-earth Digital Terrain Model (DTM) is considered for mesh interpolation. The BH method is mostly used while NB is used least. To complement the capabilities of Triangle for mesh generation, building shape data is output to a CAD Digital Exchange Format (DXF) and a utility is created to convert the created ASCII DXF file into ASCII format required by Triangle. Similarly, in cases where vector datasets of the buildings are not available, Digital Surface Models can be processed to represent building footprints followed by preparation a similar conversion utility for input to Triangle. Figure 2.7 depicts the meshing framework described by the authors.

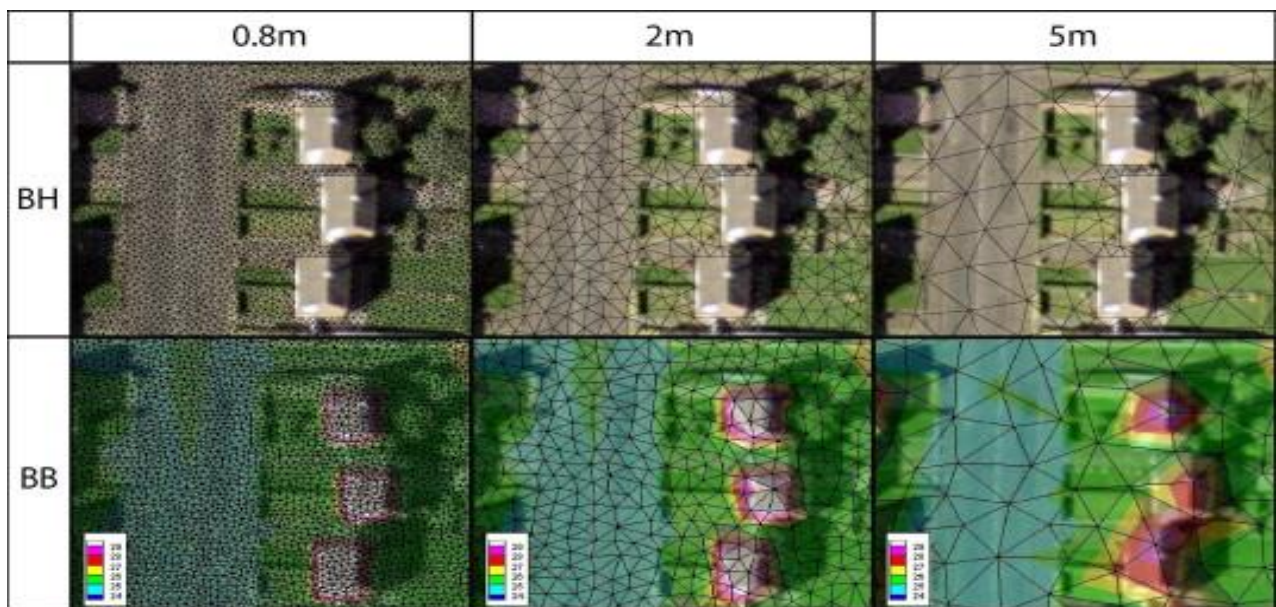


Figure 2.7 Meshes based on building-hole (BH) and building-block (BB) approaches by (Schubert et al., 2008)

CHAPTER III

METHODOLOGY

In this chapter, the programming framework of grid generation for urban environment is discussed. This framework consists of a set of algorithms, presented in Appendix A and B, implemented in MATLAB, that can generate meshes for any given cityscape and waterscape. The cityscape features considered are the buildings and terrain occupying a desired domain. However, for the waterscape, a land area with water as the most obvious feature is inferred.

The procedure involves, first, the acquisition of required geospatial data from OpenStreetMap and United State Geological Survey repository. Then, the data is parsed from its raw form into a “.txt” format which is compatible with the meshing program. Once the data is processed, it is extracted into the program and the mesh is created for the domain defined by the points in the data. This program is designed to be economical using tools including MATLAB, Triangle, and OpenStreetMap, all of which are readily available to users of the program.

3.1 Geospatial Data

The role of geospatial data for any urban study or simulation cannot be over-emphasized. As it can be deduced from chapter two, most existing mesh generation approaches for the urban environment depend on geospatial data (such as latitude and longitude) for accurate geometric representation of the urban features (mostly buildings) considered. Although, there are several sources from which the required geospatial data could be obtained, in this thesis, an online user-

generated open-source repository of geospatial data known as OpenStreetMap (OSM) is adopted as the basic source of data and it is complemented by the United States Geological Survey topographic repository, both of which are free to users. Specifically, the OpenStreetMap, is an open-source repository available online at the official OSM website. Using OSM, the data for any part of a city, state or country can easily be downloaded from the website by either manually selecting the region or entering the coordinates, if known.

OpenStreetMap (OSM) database is used here due to its growing popularity amongst researchers. It is also rapidly evolving and has attained significant feats in the last decade as it has featured in many interesting works in literature. Lastly, using OSM, as in this thesis, can help eliminate difficulties in data accessibility and the expensive cost of LiDAR for individual usage which is a major source of the information for geometry and height of the buildings as it exists in literature. It is worth mentioning that, although OSM is used in this thesis, any other repository can be utilized provided the required information, especially height, can be obtained and formatted accordingly.

3.2 Triangle

Triangle is a C-program developed by Shewchuk (1996). It is used for two-dimensional mesh generation and construction of Delaunay triangulations, constrained Delaunay triangulations, and Voronoi diagrams (Shewchuk, 1996). The program is fast, robust, efficient in memory use and can create quality meshes with exact triangulations. It also affords the user the chance to specify constraints on angles and areas of triangles to improve robustness (Shewchuk, 1996). Just like the OpenStreetMap, Triangle is freely available for downloading by users.

Triangle is used in this project as an embedded program for generating unstructured triangular grids as a part of the meshing algorithm since it offers great meshing advantage for geometries with hollow sections. Specifically, Triangle is used for executing the procedure in algorithm 8, algorithm 9, algorithm 10, algorithm 18 and algorithm 19. Triangle's source code is first downloaded, compiled and run in the program (see algorithm 12). The input file to Triangle is the coordinates and edge connectivity of the plane to be meshed with the centroid of each building or segment as holes. This input file is an "*s.poly*" file format which is created by algorithm 13.

3.3 The Meshing Framework for Cityscape

The mesh generation framework for urban cityscape is depicted in Figure 3.1. The process first involves downloading the geospatial data of the city. The required data, again, include the heights of the buildings, the footprints' coordinates and the corresponding elevations at the terrain level. This raw data is extracted into a file holder in the program. Thereafter, the mesh is obtained as follows. Using Triangle, unstructured triangular grids are developed for the terrain (i.e. the spaces between the buildings) since it is difficult to create structured grids for a plane with several holes such as the case here. Then, the buildings are meshed as structured triangular grids. Lastly, by assuming a flat plane, the roofs are meshed as triangular unstructured grids. Flat roofs are assumed for simplicity in this thesis. Although not very important, an imaginary envelope is created and meshed to enclose the active cells (cells made up of buildings and roofs, and the terrains between them). This envelope is defined in such a way that it is wider than the total area covered by the buildings and higher than all the buildings. Once these grids are developed independently, they are stacked as a single unit and this single grid structure is

inspected for duplicate nodes which are eventually filtered to keep the structure watertight. The resulting watertight structure is output as an input file for the numerical simulation.

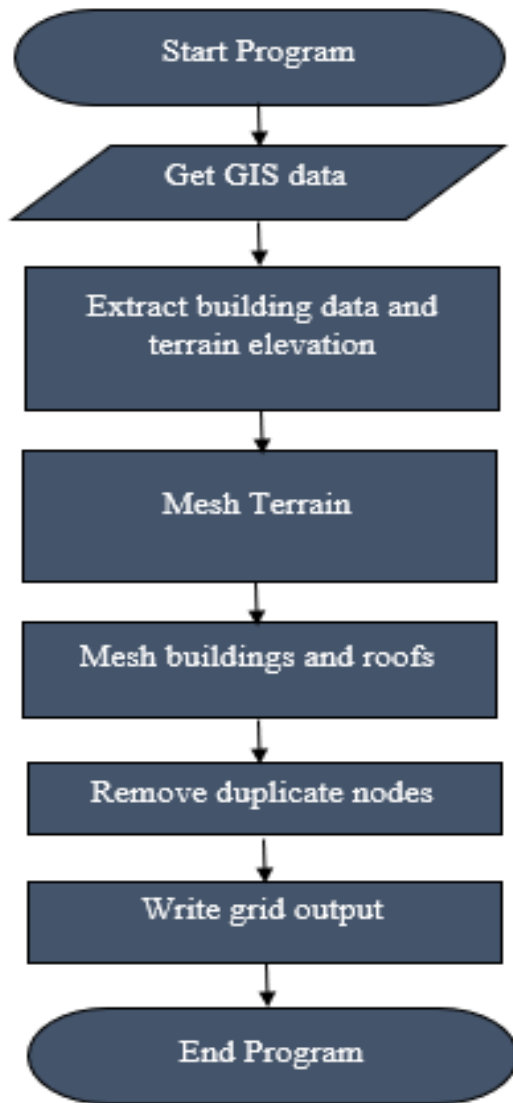


Figure 3.1 Flowchart of the cityscape mesh representation

Get Geospatial Data

In this stage, both the OSM and USGS data are obtained using an existing C-programming algorithm developed by Dr Arash Ghasemi at the University of Tennessee at

Chattanooga. The OSM component of the geospatial data for a domain, such as city, of interest is downloaded from the website by either manually selecting the region or entering the coordinates of the boundary points, if known, using the panel shown in Figure 3.2. The data in its raw “.xml” format is parsed through the C-program and it is converted to a “.txt” format which can then be input into the mesh generation program. The terrain elevation is also downloaded from the USGS website and parsed in a similar way into the C-program.

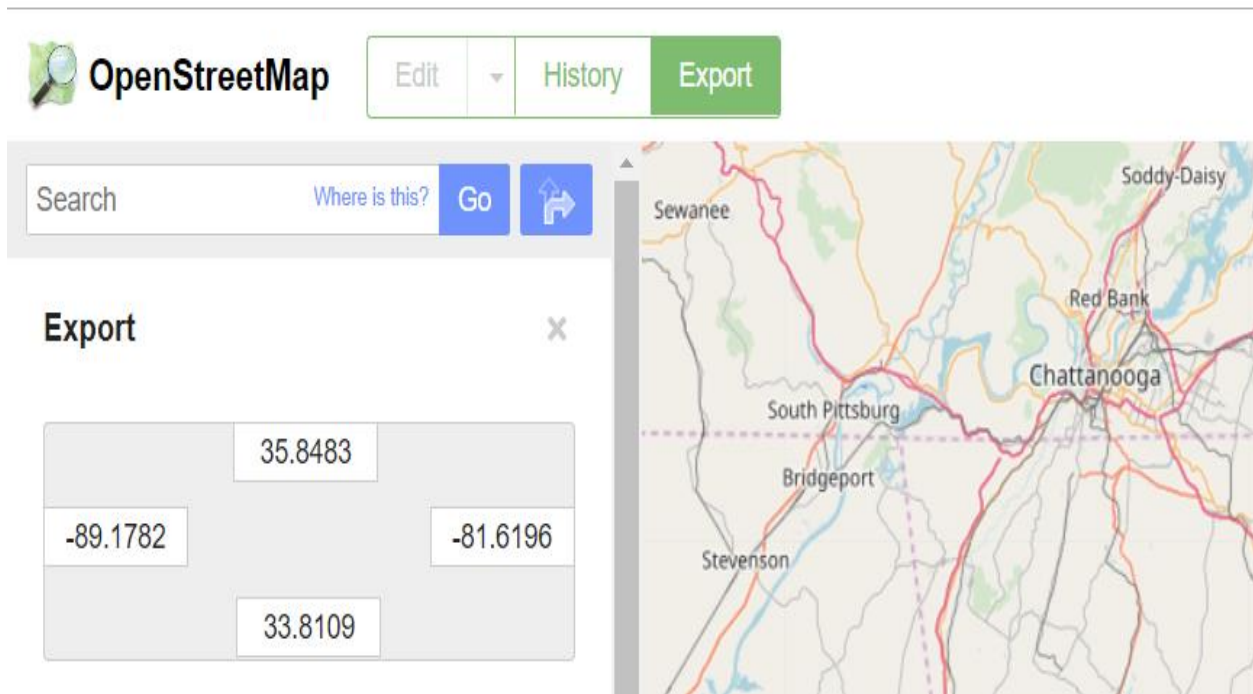


Figure 3.2 Screenshot of the data export panel on the OpenStreetMap (OSM) website.

Extract Building Data and Terrain Elevation

The structure of the new “.txt” format is such that can enable the information to be extracted easily by the mesh generation program. The first line of the “.txt” file obtained in the previous stage contains the total number of buildings, *N_{bld}*, within the domain. By looping over

the number of buildings, the heights, *Hibld*, of each building, the number of points, *Nnodes*, constituting the footprints and the coordinates, *x*, *y* of those points are retrieved in this order. With the *x*, *y* coordinates known, the points are simultaneously searched within the USGS data and the corresponding terrain elevation, *z*, is obtained. Note that, just like there is an iteration over the number of buildings, the number of points constituting footprints are iterated over to obtain the coordinates. This enables the program to determine the beginning of the data for new building.

```

2
3
9
0          0          0
0.5000    0          0
1.0000    0          0
1.0000    0.5000    0
1.0000    1.0000    0
0.5000    1.0000    0
0          1.0000    0
0          0.5000    0
0          0          0
5
9
2.5000    2.5000    0
3.0000    2.5000    0
3.5000    2.5000    0
3.5000    3.0000    0
3.5000    3.5000    0
3.0000    3.5000    0
2.5000    3.5000    0
2.5000    3.0000    0
2.5000    2.5000    0

```

Figure 3.3 Typical “.txt” file format of parsed OSM data

Figure 3.3 typifies a simple input data structure parsed from raw file to text format. In this structure, the number of buildings to be represented in mesh form is two. The first building has a height of 3 units, and its footprint has 8 nodes or points. The coordinates (latitude, longitude, and terrain elevation) of these points are the three-column spacebar-delimiting nine-

line data starting after the line containing the number of nodes (i.e.8) noting that the first node is repeated on the ninth line to form a closed loop for the building. The second building has a height of 5 units, 8 nodes on footprint and coordinates as described for the first building. This procedure is achieved in the program by algorithm 3.

Mesh Terrain

In this process, unstructured triangular grids are developed to fill the spaces between the polygons defined by the building footprints i.e. the terrain. This is achieved as part of algorithm 2 by calling algorithm 9 which has Triangle embedded. In this algorithm, the coordinates and connectivities of the building footprints as well as the centroids of the polygons formed by the footprints (i.e. holes) serve as input to Triangle. The resulting grids from this process, regarded as the active cells, are assigned “*tags=1*” value.

Mesh Buildings and Roofs

In this process, the buildings are first meshed using algorithm 8 as structured triangular grids defined using a regular algebraic method for grid generation. The procedure simply utilizes the buildings’ footprint coordinates and the height of the buildings as obtained from the geospatial database. Using algorithm 5, the footprints are first extruded from base, z , to the height, H , of the building by gradually stepping level-by- level at a size defined by algorithm 6, with the structured triangular grids generated at each step. Algorithm 6 is similar to the extrusion technique adopted by Ghasemi, Taylor, and Newman (2016).

At the top of the meshed building surfaces is the roof surface. For simplicity, flat roof planes are considered in the project. Using the same footprint coordinates for the building, but

with elevation replaced with the height of the building, unstructured triangular grids are developed to fill the roof surface with algorithm 8.

The resulting grids from this process, regarded as the active cells, are assigned “*tags=1*” value and stored with the node coordinates and connectivity defined as fields in the grid structure.

Remove Duplicate Grid Nodes

This phase is required to check the overall grid structure and delete overlapping nodes. Two or more nodes with different numbers are said to be overlapping if they share the same coordinates. These overlapping nodes may not be captured in the visualization phase but would have an impact on the simulation process. When overlapping nodes occur, the mesh is said to be non-watertight. In this program, there is a tendency for duplicate nodes to occur at the intersection of the terrain and the base of the building, as well as the intersection of the roof and the top of the building. To eliminate this scenario, the meshing procedure includes a subroutine that inspects the coordinates and filters overlapping nodes accordingly. This is achieved using algorithm 11.

Write Grid Output

Just like there is an input file to the program, the output file is required to store the generated grid structure which can then be utilized by the numerical simulation solver. In the program, the grid structure is written as a Tecplot file format which has a “.dat” file extension. The Tecplot format is adopted because it can be visualized in VisIt – a data visualization software – which is the preferred choice in our research group.

3.4 The Meshing Framework for Waterscape

The meshing procedure for the waterscape is somewhat like that of the cityscape, except that a few changes are made in the algorithms to accommodate for the structure of the output. The procedure proposed here would provide a cheaper alternative to the available market-scale grid generation programs for Environmental Fluids Dynamics Code (EFDC) discussed in Chapter 2. As such, the program is designed to generate quadrilateral cells, as required for the EFDC model, instead of the triangular cells used for the cityscape mesh. This is the first and basic difference between the two components of the framework.

Just like the cityscape component, the GIS raw data is downloaded from OpenStreetMap or any available GIS repository and the coordinates of the segments (wet and dry areas) within the waterscape are extracted into a file holder in a manner as in the cityscape component. The wet area is first meshed with triangular grids using Triangle. Thereafter, a large square envelope, defined as an imaginary land area over the waterscape being meshed, is created and filled with quadrilateral cells. The nodes making up these cells are checked if they exist inside the triangular grids to establish that the points fall within a wet area, and a tag of “1” is assigned to the nodes, or “0” if otherwise. Depending on the structure of these nodal tags, the cells are assigned values 0, 1, 2, 3, 4, 5 and 9 which are required for the EFDC model. These EFDC tags, and other properties of the cells including cell indices (i, j), coordinates of cell corner points, coordinate of midpoint of the cell, and length of cell in both directions defined as (dx , dy) are output as a “.dat” file ready to use in EFDC model. Figure 3.4 presents a simple flowchart of the procedure.

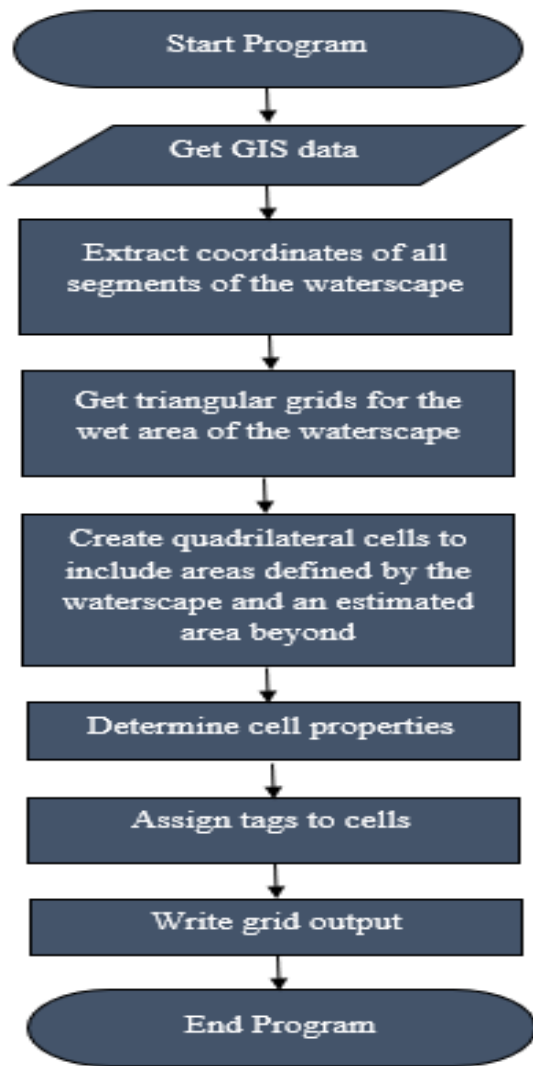


Figure 3.4 Flowchart of the waterscape mesh representation

Get GIS Data and Extract Coordinates

This process is implemented in a similar manner as in the cityscape component. The data downloaded here, however, are simply the coordinates of points defining boundary of selected waterscape. Also, the number of buildings N_{bld} is replaced with the number of segments N_{seg} (which defines the number of wet and dry area divided into different parts within the waterscape domain considered) while the heights of buildings are not applicable here. The procedure for

reading the file remains the same with slight modification in the algorithm to take care of the changes in the data. This procedure is achieved using algorithm 16.

Get Triangular Grids

Obtaining triangular grids here is also the same as with the cityscape procedure. Since the grids needed for EFDC are quadrilateral, the triangle grids here are only needed to serve as basis for establishing cell nodes that fall within the wet area. The procedure is achieved in algorithm 15 using algorithm 19.

Create Quadrilateral Cells

This process defines an imaginary square area enclosing the original area defined by the triangular grid. This big square is divided into a $n_x \times n_y$ array of cells and the procedure is achieved as part of algorithm 15.

Determine Cell Properties and Assign Tags

In this phase, the properties of the $n_x \times n_y$ cells are determined in a structured grid pattern. These values include: cell indices, coordinates of cell vertices, coordinate of centroids and cell sizes (dx, dy). Once the points are determined, the points are checked whether they fall inside the wet area, now represented with unstructured triangular grid, or dry area with a value of 1 or 0 assigned respectively to the nodes using algorithm 17. The values are then used to determine the tags of the cell. With algorithm 20, tags are assigned to the cells according to the definitions for EFDC model as shown in Figure 3.5 and Figure 3.6.

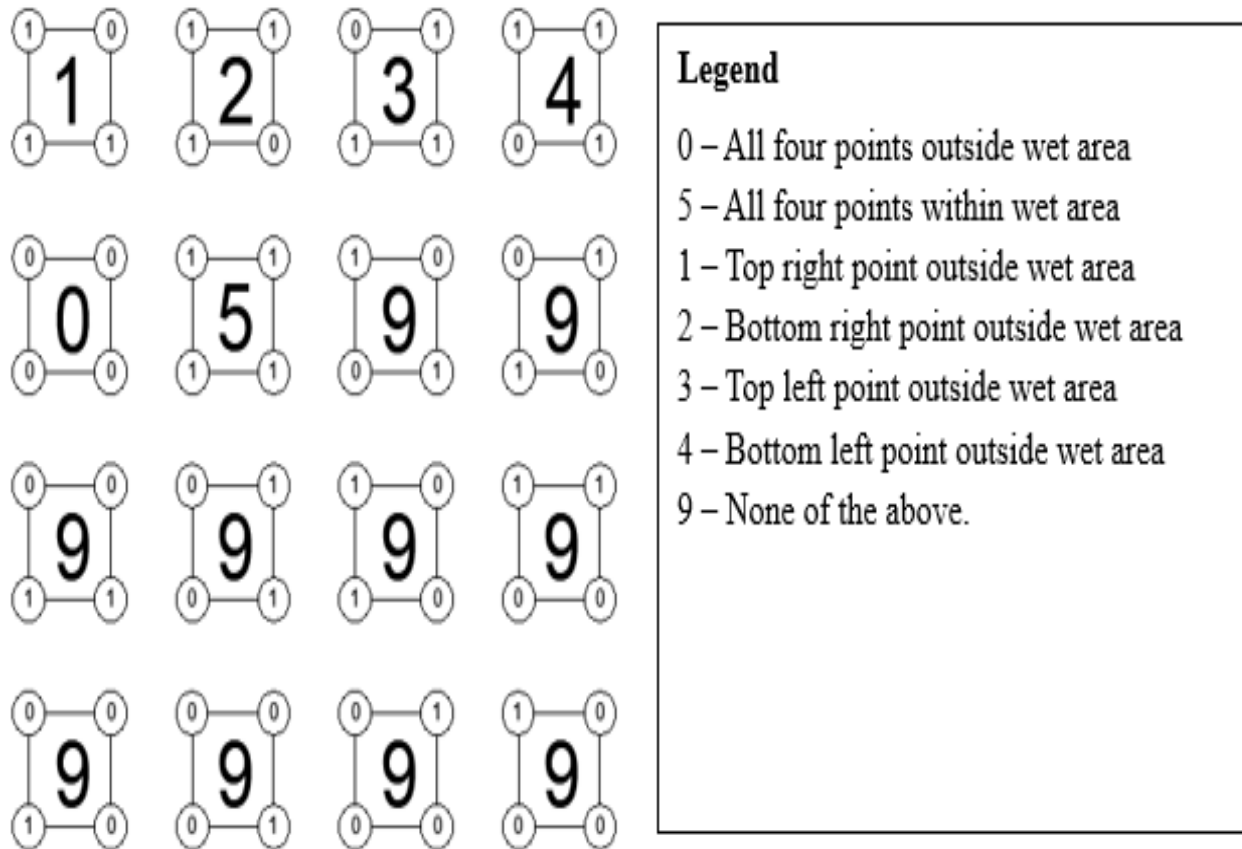


Figure 3.5 Description of cell tags for EFDC model

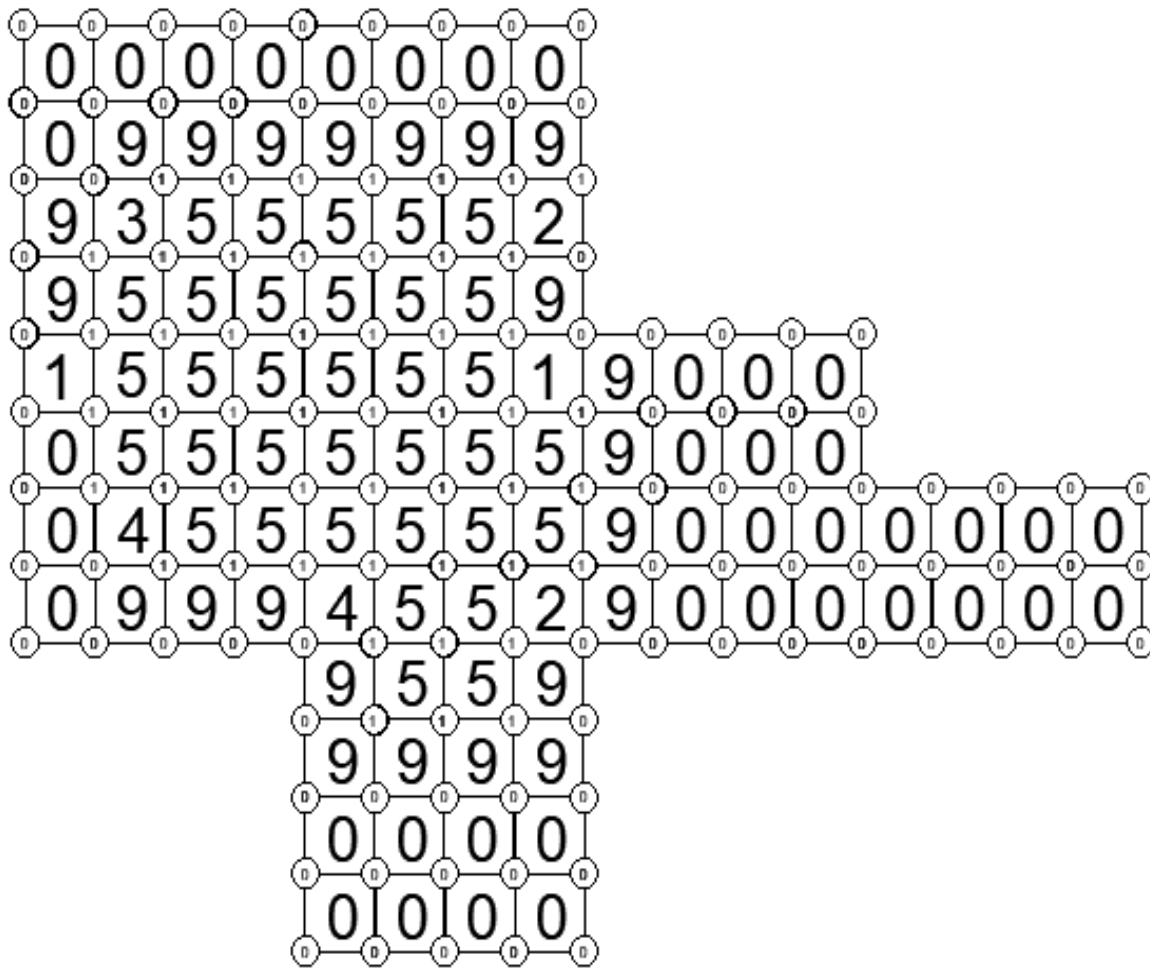


Figure 3.6 Typical mesh of quadrilateral cells with EFDC tags

Write Grid Output

Once the cell values have been determined, they are output in a “.dat” file. This process is the same as that for cityscape meshing procedure.

3.5 Applications of the Program

In order to demonstrate the significance, and efficiency, of this mesh generator, the grid structures generated from both cityscape and waterscape mesh generation programs can be used as input to computational fluid dynamics solvers. Current research exercises within the Civil

Engineering Department at the University of Tennessee - Chattanooga are utilizing this program. The cityscape meshing procedure has been used to mesh a part of Nashville, Tennessee as input to a computational fluid dynamic solver to establish the wind flow pattern around a flying Unmanned Aerial Vehicle (i.e. drone) within the buildings in the cityscape. Similarly, the waterscape meshing component has been used to obtain grids for a part of the Tennessee River near Chattanooga to study the flow and distribution of Nano-particles in the river. While the meshes have been created for these two studies using the program, simulation outputs could not be obtained because of time limitation and, as such, it is difficult to ascertain the acceptability of the meshes beyond the confirmation through visualization. However, once completed, the results from these simulations, while establishing the basis for which they were conducted would equally validate the efficiency of the mesh generation program for subsequent applications such as in disaster and extreme-event modeling, urban air pollution distribution studies, flood modeling, and wind modeling.

CHAPTER IV

RESULTS

In this section, the outputs from the program, visualized using a software called Visit (Childs, 2012), are presented. The MATLAB program was run on a 64-bit GPU-based computer with Linux operating system having processor speed of 4 x AMD FX Quad-Core, memory of 8 GB RAM and solid-state disk capacity of 85GB.

4.1 Cityscape Mesh Representation

In order to illustrate the process of execution of the program, consider a file, named `input_file1.txt`, containing city data formatted as described in Chapter 3, Figure 3.3. The number of buildings (which is 2), as well as, coordinates of points on the building footprints and heights of buildings can be extracted from this file with all other required input parameters set in the program. Figure 4.1 is an illustration of a meshed terrain of the spaces between the building. Notice how the terrain elevation changes are captured. The number of nodes in the meshed terrain is 20 and the time required for generation the grid data is 0.1121 seconds.

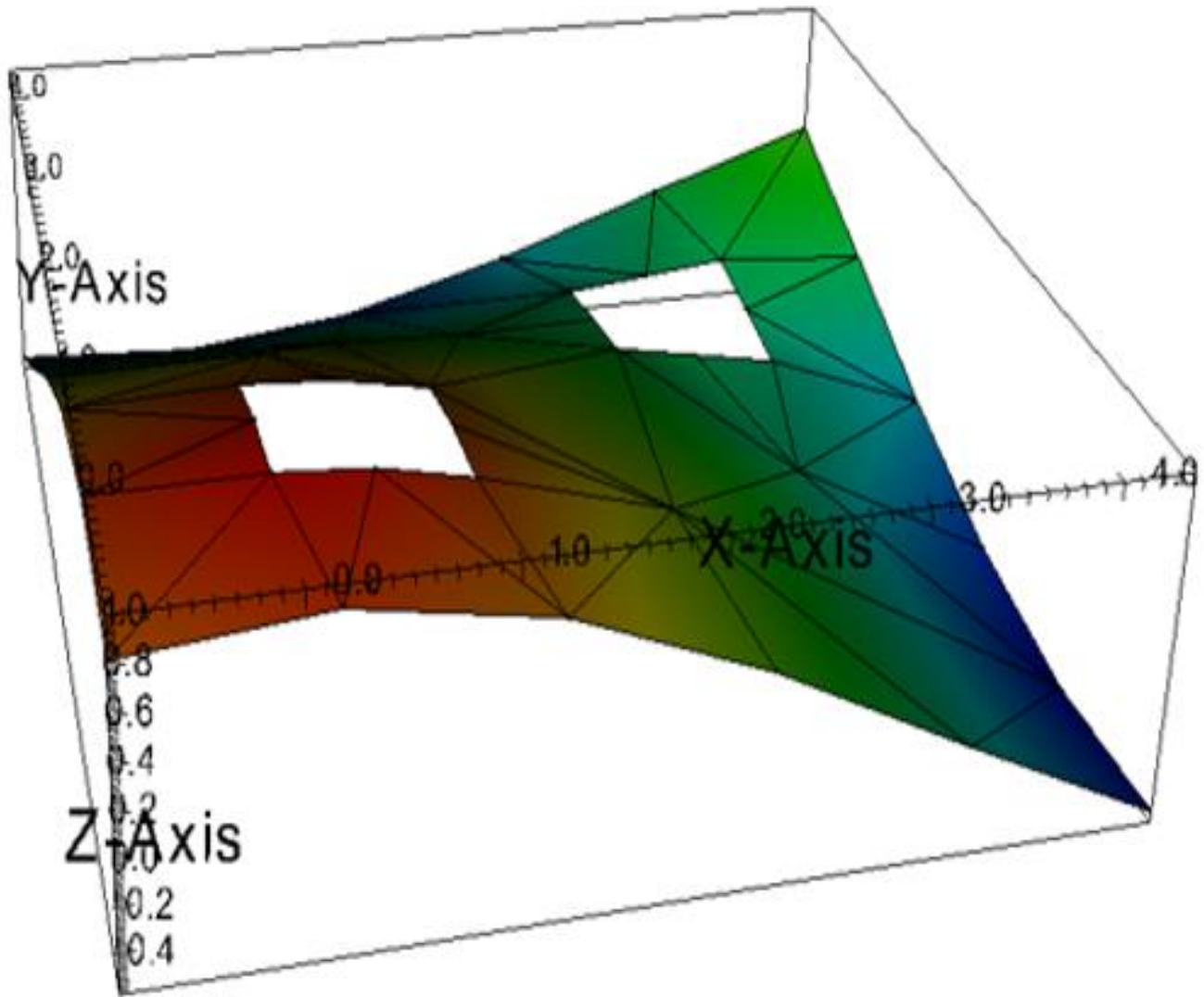


Figure 4.1 Unstructured triangle meshes for the terrain (Grid size = 20 nodes; Runtime = 0.1121 secs)

Figures 4.2 – 4.4 illustrate the stepwise process of generating the building and roof grids. Figure 4.2 depicts the first step of the extrusion with 56 nodes generated in 0.1623 seconds. Figure 4.3 depicts the second step of the extrusion with 73 nodes generated in 0.1689 seconds. This step-wise extrusion continues until the top of the building is reached and the roof plane is added. Figure 4.4 shows that the entire building has been meshed with the meshed roof plane consisting of 121 nodes generated in 0.1737 added.

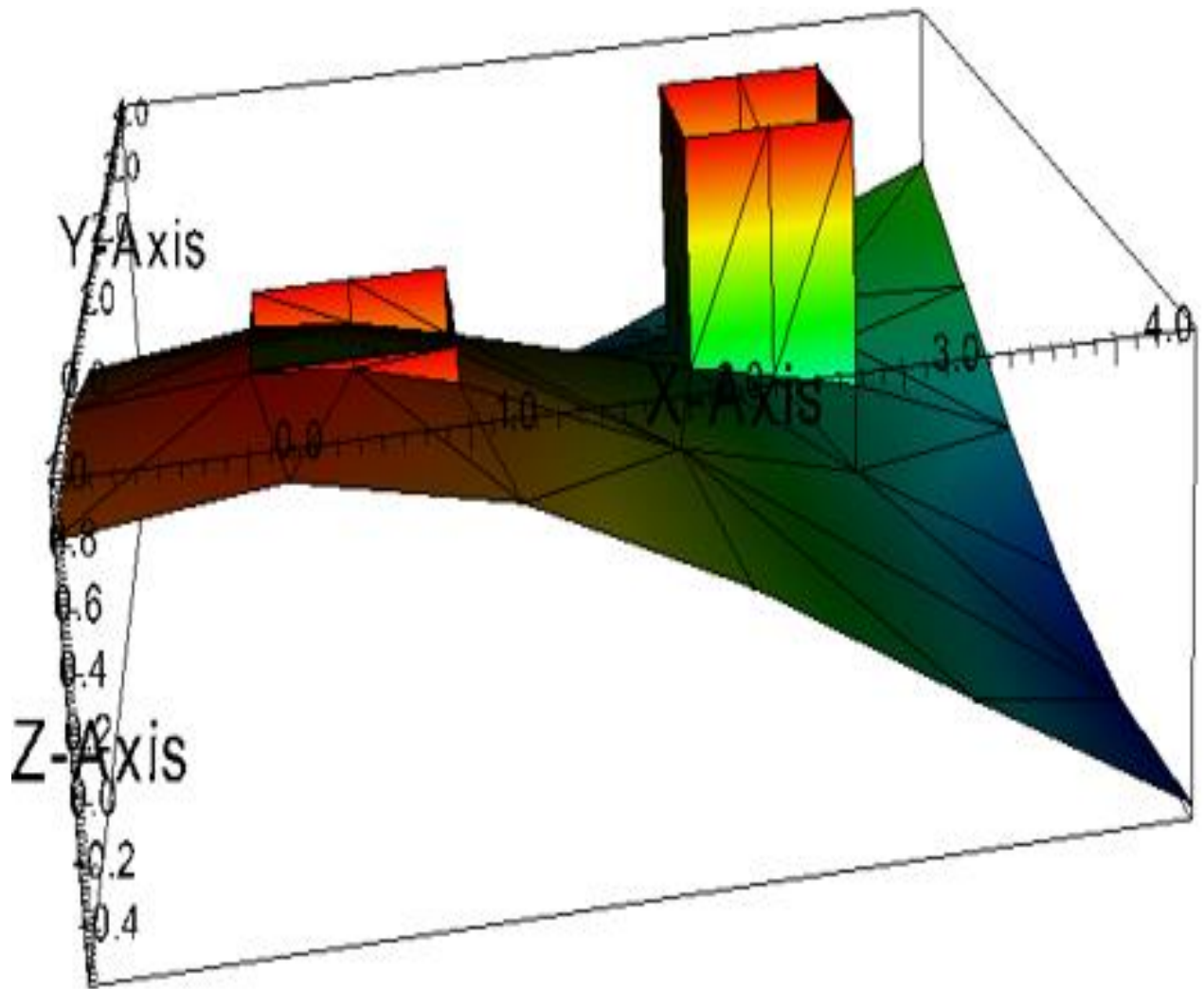


Figure 4.2 Extrusion of building meshes one step from the footprint - Step1 (Grid size =56 nodes; Run time = 0.1623 secs)

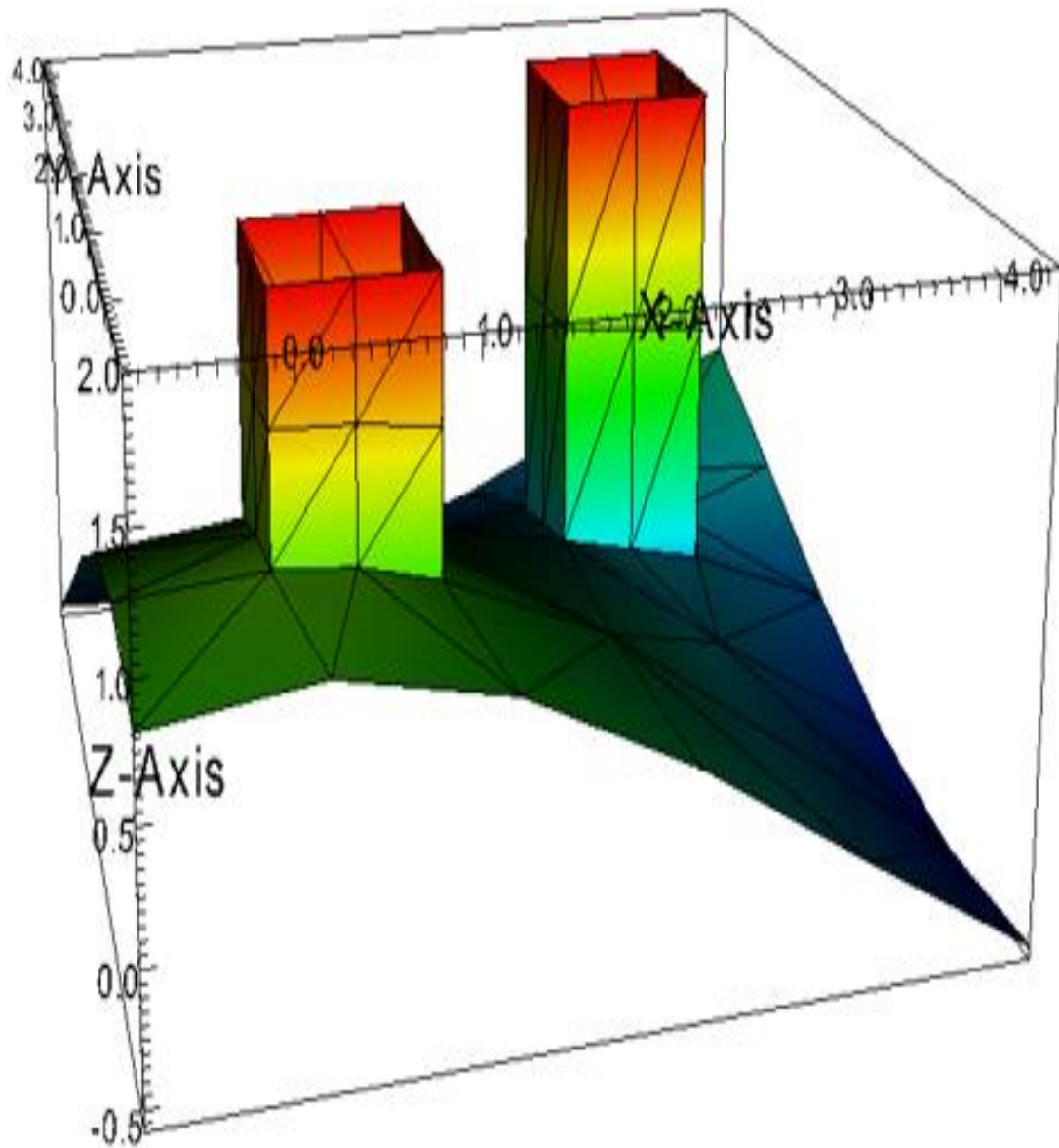


Figure 4.3 Extrusion of building meshes two steps from the footprint - step=2 (Grid size = 73 nodes; Run time = 0.1689 secs)

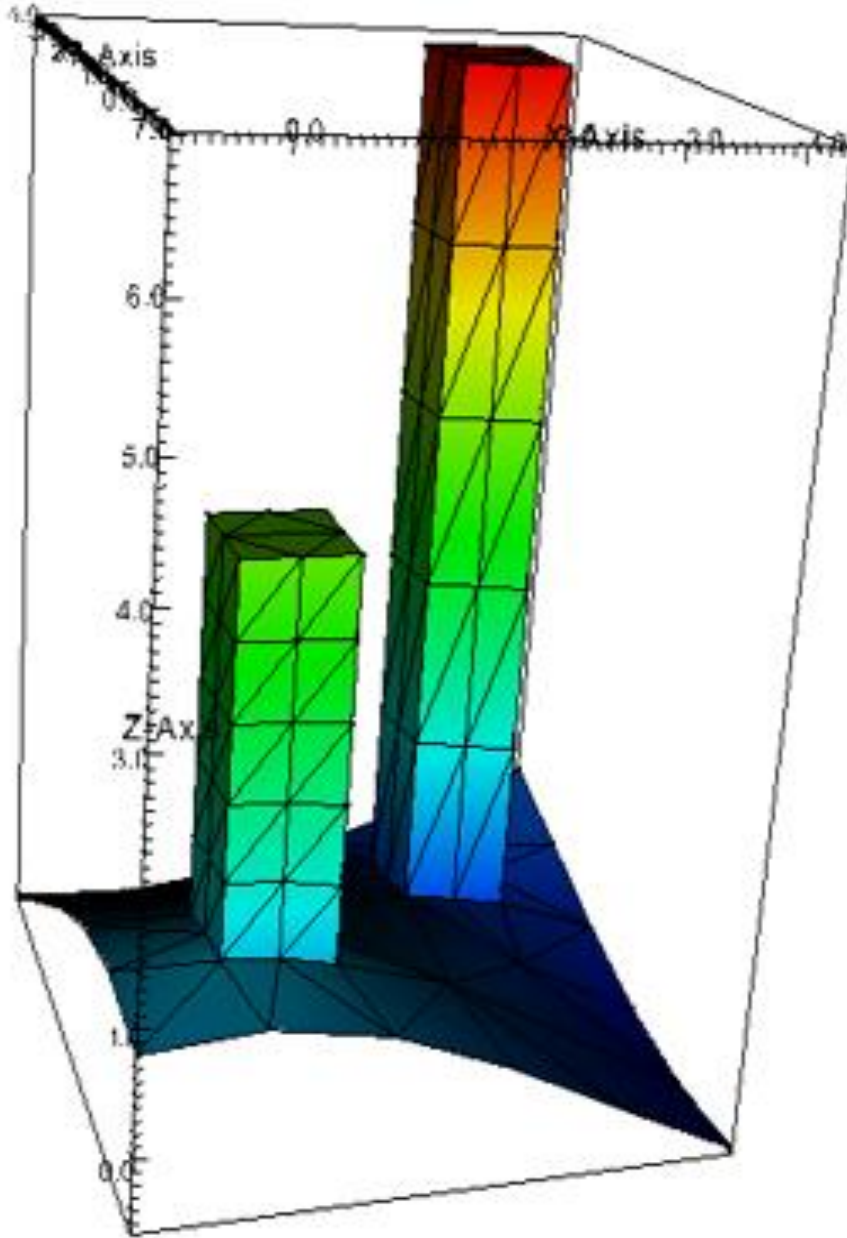


Figure 4.4 Building and roof meshes (Grid size = 121 nodes; Run time = 0.1737 secs)

The cityscape program is used with different combination of inputs to determine its robustness for random geometric arrangement of buildings in any city. A randomly generated city data in MATLAB, formatted as described in Chapter III, containing an array of 10 x 10 buildings (i.e. 100) was supplied as input to the program. The resulting mesh consists of grid size of

17,298 nodes and total cells of 34,477 elements generated in just 1 minute, 20 seconds. Figure 4.5 depicts the corresponding mesh representation of the random city.

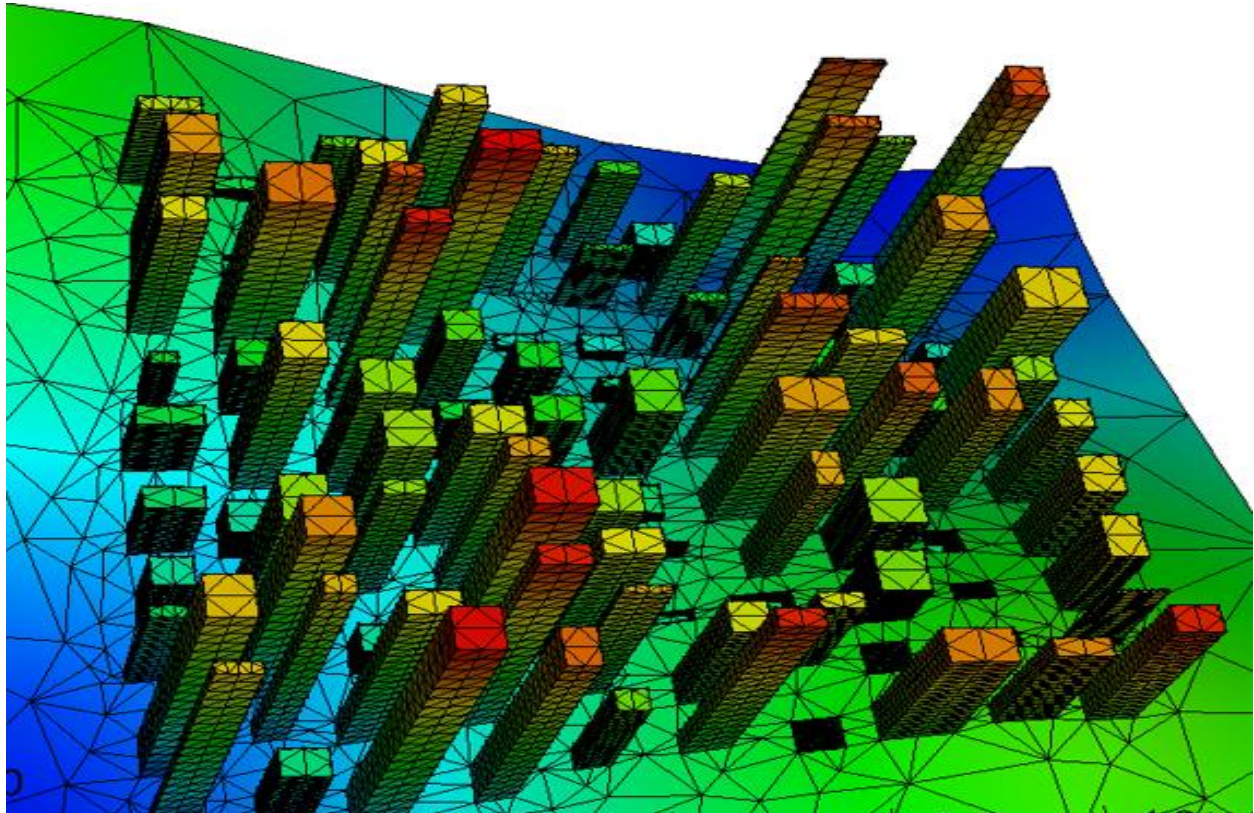


Figure 4.5 Watertight mesh for a randomly generated city (Number of nodes = 17,298; Number of cells = 34,477; Run time = 1 minutes, 20 seconds)

As shown in Figure 4.5, the program appears to be working as designed. For further validity, the program was tested with actual OpenStreetMap and USGS data. Figure 4.6 is a satellite view of a part of Nashville, Tennessee consisting of 459 randomly selected buildings for meshing. The resulting mesh consists of grid size of 48,726 nodes and total cells of 89,999 elements generated in just 10 minutes, 53 seconds. Figure 4.7 depicts the corresponding mesh representation of the random city.



Figure 4.6 Google earth view of layout of 459 buildings in Nashville, Tennessee, USA

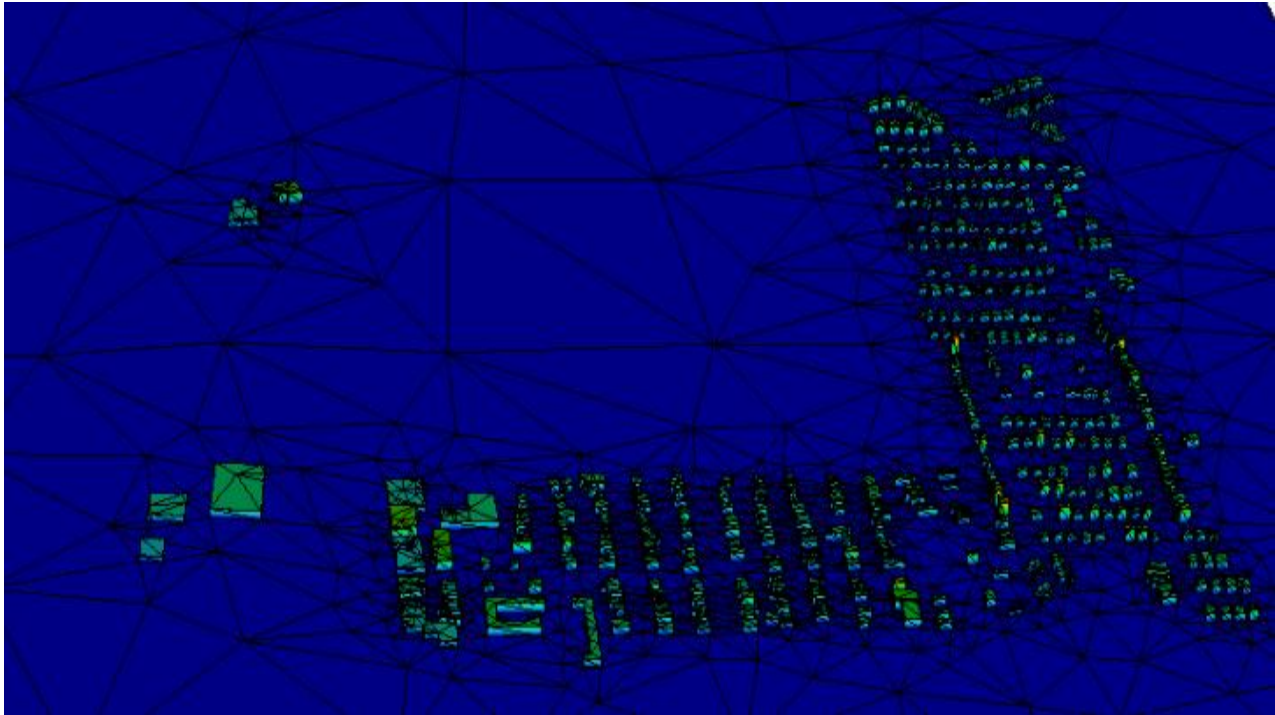


Figure 4.7 Mesh representation of 459 buildings, and terrains, in Nashville, Tennessee, USA
(Number of nodes = 48,726; Number of cells = 89,999; Run time = 10 minutes, 53 seconds)

4.2 Waterscape Mesh Representation

The waterscape meshing procedure is illustrated with a section of the Tennessee River shown in Figure 4.8. The unstructured triangular grids representing the wet area (i.e. river) only is depicted by Figure 4.9 and the quadrilateral (square) cells representing the entire waterscape (i.e. wet and dry areas) is shown in Figure 4.10. A superposition of both grid structures, depicted by Figure 4.11, enables node and cell property assignments. As shown in the Figure 4.12, tags are first assigned to individual nodes. These tags indicate regions that are within or outside the river. Tag 1 (red area) means the points are within the river, while tag 0 (blue area) implies otherwise. With these nodal tag definitions, the EFDC tags for each of the cells are determined depending on the configuration of the corner nodes for each cell as defined in Chapter III.

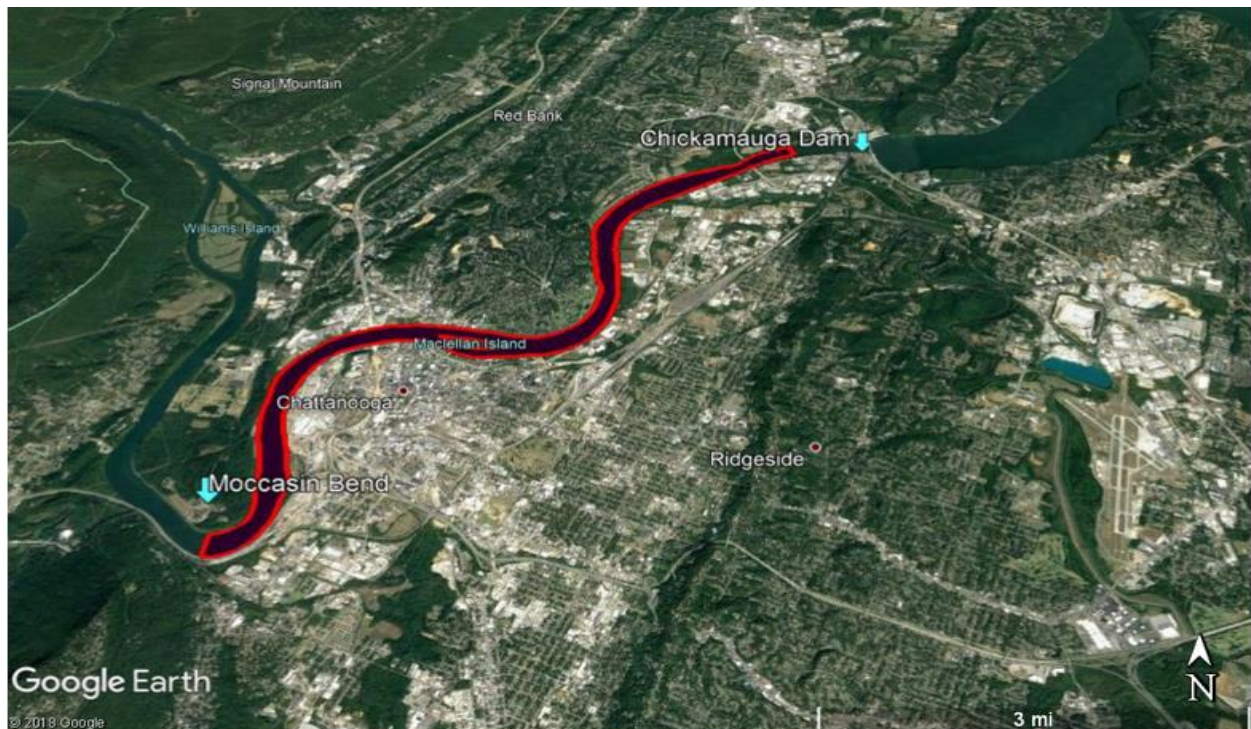


Figure 4.8 Google earth view of a part the Tennessee River

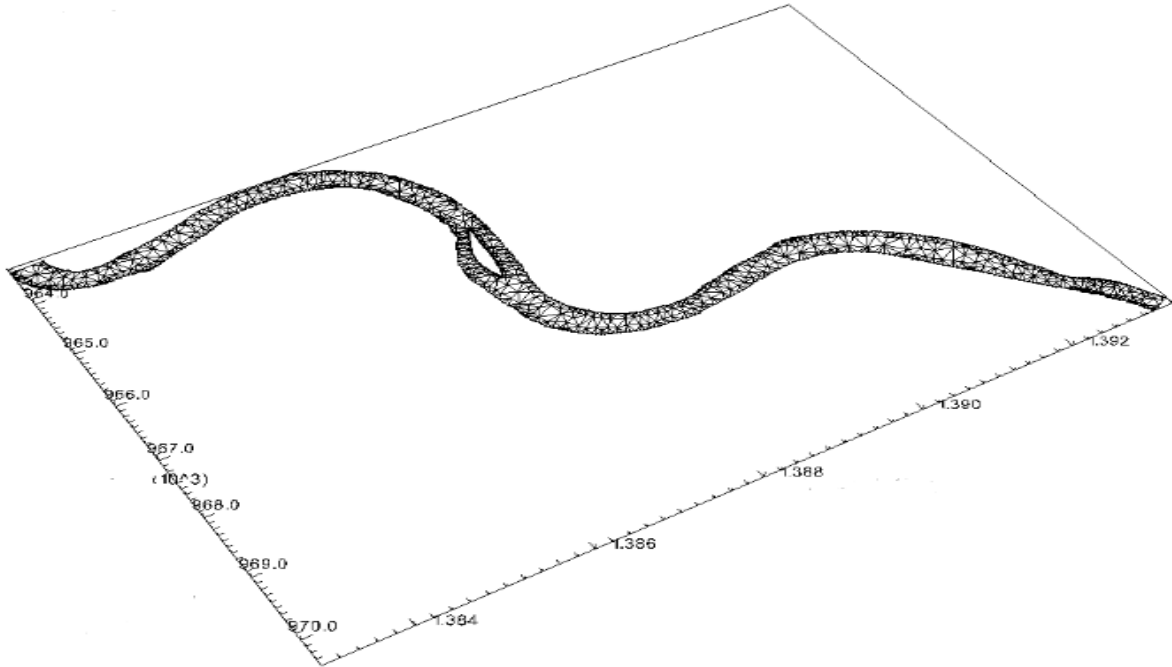


Figure 4.9 Unstructured triangular grids for wet area of waterscape

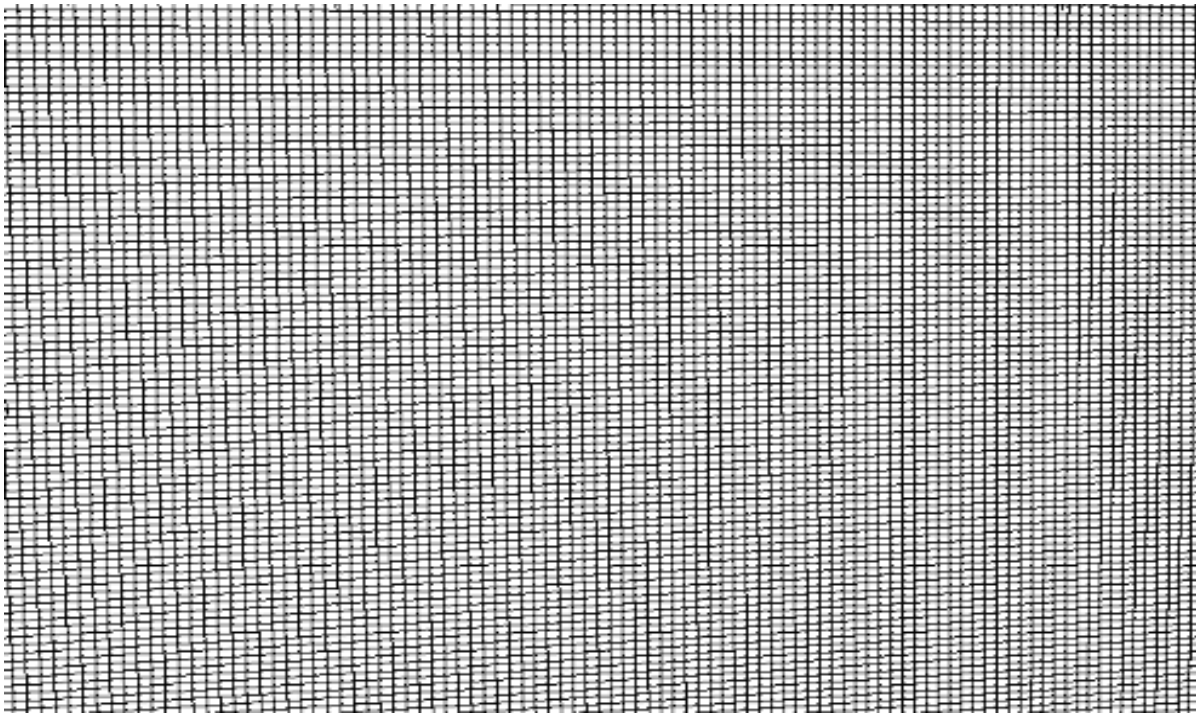


Figure 4.10 Quadrilateral cells representing the waterscape

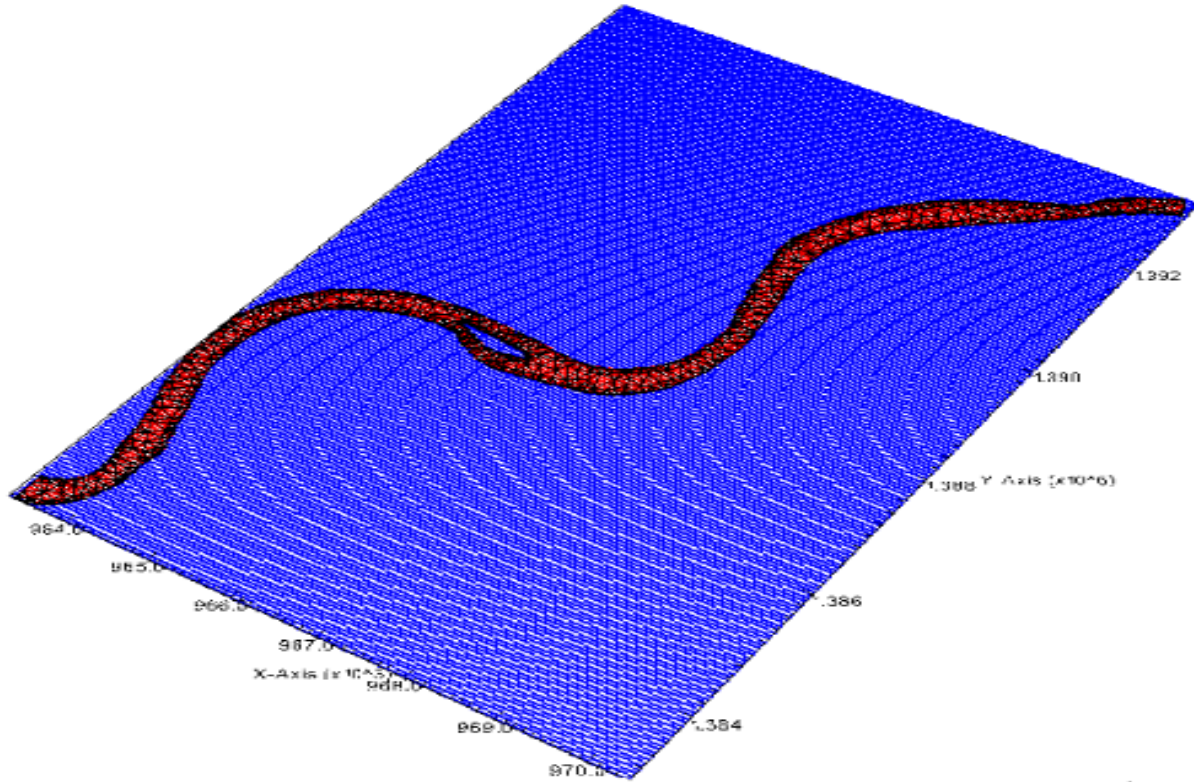


Figure 4.11 Super-imposed mesh representation of the Tennessee River (Grid size = 40,000 quadrilateral cells, 1,553 triangle cells (Total 41,553), Run Time = 7 minutes, 29 seconds)

As shown in Figure 4.11, there are a total of 41,553 cells, out of which 40,000 are active (quadrilateral) cells which are needed for EFDC modeling. The total time taken is 7 minute, 29 seconds. Again, the triangular grids have only been generated for nodal tag assignment and are not actually needed for EFDC simulation. In other words, only the properties of the square cells are exported as output of the procedure. Figure 4.12 is a blown-out view of a section of the meshed waterscape while Figure 4.13 depicts a part of the EFDC tag formation of the meshed waterscape.

CHAPTER V

SUMMARY AND CONCLUSION

5.1 Summary

Solutions to complex urban problems require computational simulations of the physical environments. However, the simulations, because they often utilize numerical approaches such as approximation of partial differential equations representing these physical problems need efficient grid technologies that discretize the physical domain into smaller components called meshes. An efficiently constructed mesh must satisfy several conditions but ultimately must be one that allows a convergence of the numerical equations for the problem with reasonable computational demand.

In this thesis, a programming framework, implemented in MATLAB, is developed for mesh representation of the environment. This framework generates hybrid mesh for any given cityscape and waterscape. The framework, unlike other approaches in literature, is cheap since it utilizes tools including MATLAB, Triangle, and geospatial data from OpenStreetMap and United States Geological Survey repositories, all easily available for free. It does not use data sources such as construction drawings and layouts of existing features which may be difficult to access in some cases, neither does it utilize Digital Elevation Model (DEM) and Light Detection and Ranging (LiDAR) which can be very costly. Although, it may be argued that both OSM and USGS repository are LiDAR data, the process of having to generate LiDAR data for individual

use is by-passed in this program since the user can simply download the data from the aforementioned repositories.

The program consists of two meshing components. The first component generates mesh representation for a cityscape, with buildings and terrains as the main features. The resulting mesh is made watertight using a unique algorithm developed in this thesis. The second component generates mesh representation of a given waterscape such as a river, including any island that may exist. The resulting mesh is such that can be used in Environmental Fluid Dynamics Code, a popular computational platform for environmental fluid modeling and simulations. The framework is validated by creating mesh representation for a part of Nashville, Tennessee, and the Tennessee River.

Although, there remain some challenges with efficiency of the program, especially in terms of computational demand, the work presented here is an alternative to existing urban environments mesh representation methods. As a matter of fact, the issues with computational demand can be mitigated with high-skilled programming techniques, such as effective pre-locations of arrays as well as improved organization of the algorithms, which are currently not being adequately explored. Nevertheless, the time of implementation is still very low compared with existing approaches in literature. Similarly, OpenStreetMap which is the major geospatial repository is still developing and the needed data is currently not available for all locations on the globe. This means that the program is currently limited to areas for which OpenStreetMap data has been generated, though it is envisaged that the OSM repository would become much developed in a few years considering the current rate of development of the database and its growing popularity, especially amongst researchers. This thesis can equally play a role in that regard by serving as a motivation for the OSM developers. Lastly, the framework presented for

cityscape is currently designed to generate flat roof surfaces for the buildings. Since, in reality, the roofs of buildings have varied geometries with varying complexities, it is necessary that the framework be improved to incorporate such complexities in the future. This is not explored now but can constitute future works.

5.2 Conclusion

This thesis has utilized established grid generation techniques to develop an automatic meshing program for important urban problems involving cityscape and riverscape. The program is crucial to environmental computational fluids dynamics and the framework presented in this thesis can help eliminate or reduce the painstaking, time-exhausting and costly acquisition of data such as LiDAR for individual usage as well as existing layouts of the built environment that are used in current trends thus by-passing the need for creating preliminary three-dimensional geometric representation of the building.

Although there is no such work that uses OSM data, as this thesis, for mesh representation of urban environment, no claim is made on whether this is the best approach to urban environments' meshing. It is simply hoped that this work can alleviate the meshing procedure for researchers involved in environmental fluids dynamics modeling.

REFERENCES

- Alarcon, V. J., McAnally, W. H., & Pathak, S. (2012). Comparison of two hydrodynamic models of weeks bay, alabama. Paper presented at the International Conference on Computational Science and Its Applications.
- Bern, M. W., & Plassmann, P. E. (1997). Mesh generation: Pennsylvania State University, Department of Computer Science and Engineering, College of Engineering.
- Blocken, B. (2015). Computational Fluid Dynamics for urban physics: Importance, scales, possibilities, limitations and ten tips and tricks towards accurate and reliable simulations. *Building Environment*, 91, 219-245.
- Cedillo, P. E. (2015). Hydrodynamic Modeling of the Green Bay of Lake Michigan Using the Environmental Fluid Dynamics Code. Theses and Dissertations. 1042. <https://dc.uwm.edu/etd/1042>
- Childs, H. (2012). VisIt: An end-user tool for visualizing and analyzing very large data.
- Coirier, W., Fricker, D., Furmanczyk, M., & Kim, S. (2005). A computational fluid dynamics approach for urban area transport and dispersion modeling. *Environmental Fluid Mechanics*, 5(5), 443-479.
- Cunanan, A. M., & Salvacion, J. W. (2016). Hydrodynamic Modeling of Laguna Lake Using Environmental Fluid Dynamics Code. *Int'l Journal of Research in Chemical, Metallurgical and Civil Engg. (IJRCMCE)*, 3(1).
- Fluent Inc. (2003). Grid Terminology. Retrieved from <http://jullio.pe.kr/fluent6.1/help/html/udf/node10.htm>
- Gargallo-Peiró, A., Folch, A., & Roca, X. (2016). Representing urban geometries for unstructured mesh generation. *Procedia engineering*, 163, 175-185.
- Garland, M., & Heckbert, P. S. (1997). Surface simplification using quadric error metrics. Paper presented at the Proceedings of the 24th annual conference on Computer graphics and interactive techniques.
- Ghasemi, A., Taylor, L. K., & Newman, J. C. (2016). Massively Parallel Curved Spectral/Finite Element Mesh Generation of Industrial CAD Geometries in Two and Three Dimensions. Paper presented at the ASME 2016 Fluids Engineering Division Summer Meeting

collocated with the ASME 2016 Heat Transfer Summer Conference and the ASME 2016 14th International Conference on Nanochannels, Microchannels, and Minichannels.

- Hamrick, J. (2002). Theoretical and Computational Aspects of Sediment and Contaminant Transport in the EFDC Model. Third Draft, Tetra Tech, Inc., Fairfax, VA.
- Hamrick, J. M., & Mills, W. B. (2000). Analysis of water temperatures in Conowingo Pond as influenced by the Peach Bottom atomic power plant thermal discharge. *Environmental Science & Policy*, 3, 197-209.
- Kaijima, S., Bouffanais, R., Willcox, K., & Naidu, S. (2013). Computational fluid dynamics for architectural design. *Architectural Design*, 83(2), 118-123.
- Karman, S. L., Wyman, N., & Steinbrenner, J. P. (2017). Mesh generation challenges: A commercial software perspective. Paper presented at the 23rd AIAA Computational Fluid Dynamics Conference.
- Kim, D. (2014). The Application of CFD to Building analysis and Design: A Combined Approach of An Immersive Case Study and Wind Tunnel Testing. Virginia Tech.
- Lafarge, F., & Mallet, C. (2012). Creating large-scale city models from 3D-point clouds: a robust approach with hybrid representation. *International journal of computer vision*, 99(1), 69-85.
- LearnCAX. ("n.d."). Grid Generation for CFD Simulation: Introduction. Retrieved 6/10/2019 <https://www.learncax.com/knowledge-base/blog/by-category/cfd/grid-generation-for-cfd-simulations-introduction>
- Liseikin, V. D. (2006). A computational differential geometry approach to grid generation Second Edition: Springer Science & Business Media.
- Liseikin, V. D. (2009). Grid generation methods. Second Edition: Springer Science & Business Media.
- Liseikin, V. D. (2017). Grid generation methods. Third Edition: Springer International Publishing.
- Liu, X. (2007). 3D Numerical Modeling of Hydrodynamics and Sediment Transport in Estuaries. Retrieved from http://purl.flvc.org/fsu/fd/FSU_migr_etd-1164
- Schubert, J. E., Sanders, B. F., Smith, M. J., & Wright, N. G. J. A. i. W. R. (2008). Unstructured mesh generation and landcover-based resistance for hydrodynamic modeling of urban flooding. *31(12)*, 1603-1621.

Shewchuk, J. R. (1996). Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In *Applied computational geometry towards geometric engineering* (pp. 203-222): Springer.

Tetra Tech Inc. (2002). VOGG: A Visual Orthogonal Grid Generation Tool for Hydrodynamic and Water Quality Modeling.

Van Hooff, T., & Blocken, B. (2010). Coupled urban wind flow and indoor natural ventilation modelling on a high-resolution grid: A case study for the Amsterdam ArenA stadium. *Environmental Modelling & Software*, 25(1), 51-65.

Wang, Y., Jiang, Y., Liao, W., Gao, P., Huang, X., Wang, H., . . . Lei, X. (2014). 3-D hydro-environmental simulation of Miyun reservoir, Beijing. 8(4), 383-395.

Xiong, Y. (2010). Coupling sediment transport and water quality models: Mississippi State University.

APPENDIX A

CITYSCAPE ALGORITHM

Algorithm 1 CityExtrude

- 1: Define (nx, ny) as number of cells for meshing the city envelope.
 - 2: Define $Nstep$ as number of steps of extrusion along the vertical axis.
 - 3: Set file name for input file for raw city data i.e. $fname1$
 - 4: Set file name for output file of grids generated i.e. $fname2$
 - 5: Set $Tolerance$ for screening duplicate nodes in algorithm 11
 - 6: $[XYZ, ICON, Tags] \leftarrow$ algorithm 2 ▷ Generate mesh for the city
 - 7: $[XYZ, ICON] \leftarrow$ algorithm 11 ▷ Remove duplicate mesh nodes
 - 8: Write mesh data into grid structure. grd with fields: $x, y, z, icon, tags$
 - 9: Store grid data in $fname2$
 - 10: Output: $fname2$
 - 11: End
-

Algorithm 2 CityVolumeMeshing

- 1: This function generates the grid data for the volume mesh
 - 2: Input: $nx, ny, Nstep, fname1$
 - 3: Output: $ICON, XYZ, Tags$

 - 4: $[XYZRaw, EdgesRaw, Height, Nbld, P1, P2, index] \leftarrow$ algorithm 3
 - 5: **Step 1: Mesh the Terrains between building foots**
 - 6: Set $XYZOut, EdgesOut$ and $HolesOut$ as null
 - 7: Initialize $ibld \leftarrow 1$
 - 8: **for** $ibld$ in $Nbld$ **do**
 - 9: $XYZ \leftarrow XYZRaw\{ibld\}$ ▷ Input to algorithm 10
 - 10: $Edges \leftarrow EdgesRaw\{ibld\}$ ▷ Input to algorithm 10
 - 11: $XYZOut = [XYZOut; XYZ]$ ▷ Input to algorithm 9
 - 12: $EdgesOut = [EdgesOut; Edges]$ ▷ Input to algorithm 9
 - 13: Get Centroid for each building as holes
 - 14: $[XYZIbld, ICONIbld] \leftarrow$ algorithm10 ▷ Meshing the building base
 - 15: $row \leftarrow$ random cell (or row) number in $ICON$
 - 16: $RndmCell \leftarrow ICONIbld(row)$ ▷ RndmCell is a 1x3 vector
 - 17: $pt_i = RndmCell(i): i = 1 : 3$
 - 18: $V_j \leftarrow XYZ_{Ibld}(pt_j, 1 : end) : j = 1 : 3$ ▷ V is 1x3 vector of coordinates of the vertices
 pt_i .
 - 19: $x_k = V_j(k, 1): k = 1 : 3$
 - 20: $y_k = V_j(k, 2): k = 1 : 3$
 - 21: $HolesOut(ibld) \leftarrow \frac{\sum_{k=1}^3 x_k, y_k}{3}$ ▷ Input to algorithm 9
 - 22: **end for**
 - 23: $[XYZTerrain, ICONTerrain] \leftarrow$ algorithm9
 - 24: Add tags (tag = 0) to the $tags$ field of grd
-

Algorithm 2 CityVolumeMeshing(continued)

25: **Step 2: Mesh the Buildings and Roofs, and City Envelope**

26: Initialize $ibld \leftarrow 1$

27: **for** $ibld$ in $(Nbld + 1)$ **do**

28: **if** $ibld = (Nbld + 1)$ **then**

29: Generate grid for envelope

30: $[XYZInit, Edges] \leftarrow$ algorithm 4

31: $H \leftarrow$ **Height** \triangleright Height of the imaginary city envelope: $H > Hibld_{max}$

32: **else**

33: Generate grid for buildings and roof

34: $XYZInit = XYZRaw\{ibld\}$

35: $Edges \leftarrow EdgesRaw\{ibld\}$

36: $H \leftarrow Height\{ibld\}$ \triangleright Height of the imaginary city envelope: $Hibld$

37: **end if**

38: $[XYZBuild, ICONBuild] \leftarrow$ algorithm 7

39: $[XYZRoof, ICONRoof] \leftarrow$ algorithm 8

40: **if** $ibld = (Nbld + 1)$ **then**

41: Grid is for Envelope

42: Add tags (tag = 0) to the *tags* field of grid data

43: **else**

44: Grid is for Buildings and roofs

45: Add tags (tag = 1) to the *tags* field of grid data

46: **end if**

47: **end for**

48: Output..

49: $ICON \leftarrow [ICONTerrain, ICONBuild],$

50: $XYZ \leftarrow [XYZTerrain, XYZBuild],$

51: *Tags*

52: End

Algorithm 3 BuildingRawData

```
1: This function reads the input file "fname" containing raw data of buildings
2: Input: filename.dat
3: Output: XYZ, Edges, Height, Nbld, P1, P2, jj

4: Open fname
5: Read Number of buildings Nbld
6: Initialize jj  $\leftarrow$  1, jjinit  $\leftarrow$  1
7: Initialize ibld  $\leftarrow$  1; inode  $\leftarrow$  1
8: for ibld in Nbld do
9:   Read building height: Height
10:  Read number of points in the footprints: Nodes
11:  for inode in Nodes do
12:    Read coordinates of inode: xi, yi, zi
13:    XYZTemp(inode)  $\leftarrow$  xinode, yinode, zinode
14:    Edge1  $\leftarrow$  jj
15:    if inode = Nodes then
16:      Edge2  $\leftarrow$  jjinit
17:    else
18:      Edge2  $\leftarrow$  jj + 1
19:    end if
20:    EdgeTemp(inode)  $\leftarrow$  Edge1 : Edge2
21:    jj = jj + 1
22:  end for
23:  jjinit  $\leftarrow$  jjinit + Nodes
24:  Edges{ibld}  $\leftarrow$  EdgeTemp
25:  XYZ{ibld}  $\leftarrow$  XYZTemp
26: end for
27: Close fname
28: Determine the coordinates of two opposite corners (P1x,y, P2x,y) of the city
29: End
```

Algorithm 4 CityBoundaryData

- 1: This algorithm generates footprints coordinates of imaginary boundary of city envelope such that is larger than, and encloses, the actual domain defined by the buildings.
 - 2: Input: nx, ny, j_{init}
 - 3: Output: $XYZ, Edges$

 - 4: Get the coordinates of two corner points on the actual city data from algorithm 3
 - 5: Define a length of extension D
 - 6: Extend the corner points $P1, P2$ by distance D to imaginary points $X1, X2$.
 - 7: Assign $x1, y1 \leftarrow X1(x, y); x2, y2 \leftarrow X2(x, y)$
 - 8: Calculate cell size: $dx, dy \leftarrow \frac{x2, y2 - x1, y1}{nx, ny}$
 - 9: Initialize $Edges \leftarrow Edges_{ix2}, XY \leftarrow x_i, y_i$
 - 10: Initialize $x_0 = x1 - dx, y_0 = y1 - dy$
 - 11: **for** All nodes **do**
 - 12: $x_i = x_{i-1} + dx; y_i = y_{i-1} + dy;$
 - 13: $z_i \leftarrow$ interpolate z from GIS data using x_i, y_i coordinates
 - 14: **if** last node **then**
 - 15: $Edges(i, 1) = j; Edges(i, 2) = j_{init}$
 - 16: **else**
 - 17: $Edges(i, 1) = j; Edges(i, 2) = j + 1$
 - 18: $j = j + 1$
 - 19: **end if**
 - 20: **end for**
 - 21: Output..
 - 22: $XYZ \leftarrow [x_i, y_i, z_i]$
 - 23: $Edges \leftarrow [Edges1, Edges2]$
 - 24: End
-

Algorithm 5 StepExtrude

- 1: This function extrudes the footprints to the height of the building one step every time it runs by forming structured triangular elements.
- 2: Input: $XYZInit$, dz , $NodeInitGlobe$, $NodeSizeGlobe$
- 3: Output: $XYZStepped$, $ICONStepped$

- 4: $StepNodeSize = NodeSizeGlobe - NodeInitGlobe$
- 5: $NodeInit = NodeInitGlobe$
- 6: $r \leftarrow$ rows in $XYZInit$; $c \leftarrow$ columns in $XYZinit$
- 7: $XYZprime = [\text{zeros}(r, c - 1), dz]$
- 8: $XYZStepped \leftarrow$ Add $XYZprime$ to the $XYZInit$ arrays
- 9: **for** i in $StepNodeSize$ **do**
- 10: $node(i) = i$
- 11: **if** i is not the last node in the step **then**
- 12: $p1 = node(NodeInit)$
- 13: $p2 = p1 + 1$
- 14: $p3 = p1 + StepNodeSize + 1$
- 15: $p4 = p1 + StepNodeSize$
- 16: **else**
- 17: $p1 = node(i)$
- 18: $p2 = p1 - StepNodeSize + 1$
- 19: $p3 = p1 + 1$
- 20: $p4 = p1 + StepNodeSize$
- 21: **end if**
- 22: Formulate triangular elements or Icons
- 23: $tri1 = [p1 \ p2 \ p3]$
- 24: $tri2 = [p1 \ p3 \ p4]$
- 25: $Icon \leftarrow$ Stores $tri1, tri2$
- 26: **end for**
- 27: $ICONStepped = Icon$
- 28: Output..
- 29: $XYZStepped$, $ICONStepped$
- 30: End

Algorithm 6 StepSize

- 1: This function computes the step size dz at every step along the vertical axis (i.e. height of the building)
 - 2: Input: XYZ , $Height$, $Nstep$
 - 3: Output: dz

 - 4: $H \leftarrow$ Height of building
 - 5: $Z \leftarrow$ Elevation of the footprint points
 - 6: $dz \leftarrow \frac{H - Z}{Nstep}$
 - 7: End
-

Algorithm 7 GetBuildingGrid

- 1: This function develops grids on the surfaces of a building
 - 2: Input: $Height, Nstep, XYZinit, NodeInitGlobe, NodeSizeGlobe$
 - 3: Output: $XYZ, ICON$

 - 4: $NodeInitGlobe$: Last node number (of existing building and terrain grid points) plus 1
 - 5: $NodeSizeGlobe$: Total number of existing nodes (of building and terrain grid points)
 - 6: $dZ \leftarrow$ algorithm 6 ▷ Input to algorithm 5
 - 7: Initialize $NodeInit = NodeInitGlobe$, $NodeSize = NodeSizeGlobe$
 - 8: Initialize $XYZ \leftarrow XYZinit$, $ICON \leftarrow null$
 - 9: **for** $step$ in $Nstep$ **do**
 - 10: $[XYZnew, ICONnew] \leftarrow$ algorithm 5
 - 11: Add $XYZnew, ICONnew$ to $XYZ, ICON$ arrays
 - 12: Set $XYZinit = XYZnew$
 - 13: Set $NodeInit = NodeInit +$ (Number of nodes $XYZinit$)
 - 14: Set $NodeSize = NodeSize +$ (Number of nodes $XYZinit$)
 - 15: **end for**
 - 16: End
-

Algorithm 8 GetRoofGrid

```
1: This function develops grids for a flat roof surface
2: Input: XYZRaw, EdgesRaw, Height
3: Output: XYZ, ICON

4: filename ← algorithm 13           ▷ Rewrite edge segment into Triangle's input file for
   algorithm 12
5: Set Holes ← null
6: XY, ICON ← algorithm 12         ▷ Includes path to Triangle.
7: for All XY do
8:   H ← Height{ibld}
9:   XYZ ← Add H to (XY)i
10: end for
11: End
```

Algorithm 9 GetTerrainGrid

```
1: This function develops grids for a flat roof surface
2: Input: XYZRaw, EdgesRaw
3: Output: XYZ, ICON

4: filename ← algorithm 13           ▷ Rewrite edge segment into Triangle input file
5: Set Holes ← HolesOut
6: filename ← Write file for XY, Edges, Holes as input for algorithm 12
7: XY, ICON ← algorithm 12         ▷ Includes path to Triangle.
8: for All XY do
9:   z ← Elevation of building footprints
10:  XYZ ← Add z to (XY)i
11: end for
12: End
```

Algorithm 10 GetBuildingBaseGrid

```
1: This function develops grids for the base of a building
2: Input: XYRaw, EdgesRaw
3: Output: XYZ, ICON

4: filename ← algorithm 13           ▷ Rewrite edge segment into Triangle input file
5: Set Holes ← null
6: XY, ICON ← algorithm 12         ▷ Includes path to Triangle.
7: for All XY do
8:   Set H ← zero
9:   XYZ ← Add H to (XY)i
10: end for
11: End
```

Algorithm 11 RemoveRepeatedNodes

```
1: This algorithm removes duplicate nodes from an array of  $XYZ$  points
2: Input:  $XYZ, ICON, Tolerance$ 
3: Output:  $XYZSorted, ICONSorted$ 

4: Step 1: Find duplicate rows in  $XYZ$  array and assign same node numbers
5:  $Perm \leftarrow 1 : Nnodes$   $\triangleright$  Serves as original node numbers associated to  $XYZ$ 
6:  $Found \leftarrow zeros(Nnodes, 1)$   $\triangleright$  Holds flags 1 or 0 for existing and non-existing node
   numbers, respectively, in  $Perm$ 
7: Initialize  $inode \leftarrow 1$ 
8: for  $inode$  in  $Nnodes$  do
9:    $inode$  iterates over original (raw) nodes  $XYZ$ 
10:  if  $Found(inode) = 1$  then
11:    skip  $\triangleright$  Node is already a duplicate
12:  end if
13:   $XYZraw \leftarrow XYZ(inode, :)$ 
14:  for  $jnode$  in  $Nnodes$  do
15:     $jnode$  iterates over the adjusted nodes  $XYZhold$  (assumed same as  $XYZ$  for
       $inode = 1$ )
16:     $XYZhold \leftarrow XYZ(jnode, :)$ 
17:    if  $inode = jnode$  then
18:      skip
19:    end if
20:    if  $norm(XYZraw - XYZhold) \leq Tolerance$  then  $\triangleright$  Duplicate exists.
21:      Update  $Perm$  ..
22:       $Perm(jnode) \leftarrow inode$ 
23:       $Found(jnode) = 1$  ;
24:    end if
25:  end for
26: end for
```

Algorithm 11 RemoveRepeatedNodes(continued)

27: Step 2: Eliminate gaps in node numbers (that now exists in $Perm$ due to repetition)
28: Assign $LastNode \leftarrow 0$ ▷ Initial node number which is certainly 1.
29: Initialize $inode \leftarrow 1$
30: **for** $inode$ in $Nnodes$ **do**
31: $ThisNode = Perm(inode)$
32: **if** $(ThisNode - LastNode) \leq 1$ **then**
33: skip ▷ Shifting not required
34: **else**
35: $diff = ThisNode - LastNode$
36: $Perm(inode) = ThisNode - diff + 1$
37: **end if**
38: **end for**
39: Step 3: Effect the new node numbering in the $ICON$ array
40: Initialize $inode \leftarrow 1$
41: **for** $inode$ in $Nnodes$ **do**
42: $RawNode = inode$
43: $ICONSorted \leftarrow$ Replace $RawNode$ in $ICON$ with $Perm(inode)$
44: **end for**

45: Step 4: Delete repeating rows in XYZ array (i.e. nodes).
46: First, Assign Negative values to repeating nodes in $Perm$
47: Let $PermHold \leftarrow Perm$
48: Initialize $PrevNodes$ as Null Array
49: Initialize $inode \leftarrow 1$
50: **for** $inode$ in $Nnodes$ **do**
51: $ThisNode = Perm(inode)$
52: **if** $ThisNode$ exists in $PrevNodes$ **then**
53: $Perm(inode) = -Perm(inode)$
54: **else**
55: $Perm(inode) = Perm(inode)$
56: **end if**
57: Update $PrevNodes$
58: **end for**

59: Step 5: Now filter rows in XYZ array corresponding to negative node values.
60: Initialize $inode \leftarrow 1$
61: **for** $inode$ in $Nnodes$ **do**
62: $ThisNode = Perm(inode)$
63: **if** $ThisNode$ is Negative **then**
64: $XYZSorted \leftarrow$ Remove row from XYZ
65: **end if**
66: **end for**
67: End

Algorithm 12 RunTriangle

- 1: This function uses **Triangle** to generate unstructured triangular meshes for a defined 2D plane
 - 2: Input: *s.poly* file
 - 3: Output: *XY*, *ICON*

 - 4: Check if file extension is ".poly"
 - 5: *fname1* \leftarrow Open file..
 - 6: Call **Triangle** by adding its directory to the CityExtrude program path with *fname1* as the input
 - 7: Read output of **Triangle** and store in another file *fname2*
 - 8: close *fname1*
 - 9: Read coordinates and connectivities into *XY* and *ICON* respectively
 - 10: End
-

Algorithm 13 WriteSegmentForTriangle

- 1: This function writes coordinates *XY*, edges and holes as an input file to **Triangle**
 - 2: Input: *fname*, *Edges*, *XYZ*, *holes*
 - 3: Output: *filename.poly*

 - 4: *filename* \leftarrow Convert *fname* to "*s.poly*" format
 - 5: Open *filename.poly*
 - 6: Write *XY* values of the points *XYZ* into *filename.poly*
 - 7: Write the *Edges* as segments into *filename.poly*
 - 8: Write the *holes* into *filename.poly*
 - 9: Close *filename.poly*
 - 10: End
-

APPENDIX B

WATERSCAPE ALGORITHM

Algorithm 14 WaterScapeMesh

- 1: Define (nx, ny) as number of quad cells in both directions.
 - 2: Set file name for input file for raw city data i.e. $fname1$
 - 3: Set file name for output file of grids generated i.e. $fname2$
 - 4: $[grd] \leftarrow$ algorithm 15
 - 5: Store grid data in $fname2$
 - 6: Output: $fname2$
 - 7: End
-

Algorithm 15 RiverGrid

- 1: This function generates the grid data for the Waterscape mesh
 - 2: Input: $nx, ny, N_{seg}, XYZRaw, EdgesRaw$
 - 3: Output: $cells$

 - 4: **Step 1: Get triangular grids for Waterscape**
 - 5: Extract coordinates and connectivity into arrays $XYZ, Edges$ respectively.
 - 6: **for** i in N_{seg} **do**
 - 7: Store $XYZ_{seg}, Edges_{seg}$ in $XYZ, Edges$ ▷ Input to algorithm 19
 - 8: **end for**
 - 9: **if** $N_{seg} = 1$ **then**
 - 10: Island doesn't exist
 - 11: $Holes \leftarrow null$
 - 12: **else**
 - 13: Let $i_{seg} \leftarrow 1$
 - 14: **for** i_{seg} in $(N_{seg} - 1)$ **do**
 - 15: Extract $XYZ_{seg}, Edges_{seg}$ ▷ Input to algorithm 18
 - 16: $XYZ_{island}, ICON_{island} \leftarrow$ algorithm 18
 - 17: Pick any triangle
 - 18: $r \leftarrow$ random cell (or row) number in $ICON_{island}$
 - 19: $RndmCell \leftarrow ICON_{island}(r)$ ▷ RndmCell is a 1x3 vector
 - 20: $pt_i = RndmCell(i)$
 - 21: $V_j \leftarrow XYZ_{island}(pt_j, 1 : end); j = 1 : 3$ ▷ V is 1x3 vector of coordinates of the vertices pt_i .
 - 22: $x_k = V_j(k, 1), k = 1 : 3$
 - 23: $y_k = V_j(k, 2), k = 1 : 3$
 - 24: $Holes(i) \leftarrow \frac{\sum_{k=1}^3 x_k, y_k}{3}$ ▷ Input to algorithm 19
 - 25: **end for**
 - 26: **end if**
 - 27: $XYZ_{tri}, ICON_{tri} \leftarrow$ algorithm 19
-

Algorithm 15 RiverGrid(continued)

28: **Step2: Define quad cells beyond the Waterscape domain**
29: Extend two distant corner points defined by XYZ_{tri} to $P1, P2$ by a desired length
30: Let $x1 = P1_x, y1 = P1_y, x2 = P2_x, y2 = P2_y,$
31: Calculate cell size: $dx, dy \leftarrow \frac{x2, y2 - x1, y1}{nx, ny}$
32: Hold $x1_{init} \leftarrow x1$
33: Hold $y1_{init} \leftarrow y1$
34: **Get coordinates of grid cells**
35: initialize $inode, jnode \leftarrow 1$
36: initialize $jj \leftarrow 1$
37: **for** $jnode$ in $(ny + 1)$ **do**
38: **for** $inode$ in $(nx + 1)$ **do**
39: $x(inode, jnode) = x1 + dx;$
40: $y(inode, jnode) = y1;$
41: Check point x, y in $ICON_{tri}$; Add tags $T = 1$ or 0
42: $T(inode, jnode) \leftarrow$ algorithm 17
43: $index(inode, jnode) = jj$
44: $jj = jj + 1$
45: $x1 = x1 + dx$
46: **end for**
47: $x1 = x1_{init}$
48: $y1 = y1 + dy$
49: **end for**
50: **Calculate cell values**
51: Initialize $icell, jcell \leftarrow 1$
52: **for** $jcell$ in ny **do**
53: **for** $icell$ in nx **do**
54: $cells\{icell, jcell\}.corners.x = x(icell, jcell), x(icell + 1, jcell), x(icell + 1, jcell + 1), x(icell, jcell + 1)$
55: $cells\{icell, jcell\}.corners.y = y(icell, jcell), y(icell + 1, jcell), y(icell + 1, jcell + 1), y(icell, jcell + 1)$
56: $cells\{icell, jcell\}.centroid = \bar{x}, \bar{y}$
57: $cells\{icell, jcell\}.dxdy = dx, dy$
58: $cells\{icell, jcell\}.tags = T(icell, jcell), T(icell + 1, jcell), T(icell + 1, jcell + 1), T(icell, jcell + 1)$
59: **end for**
60: **end for**

61: **Step3: Write EFDC File**
62: $cells \leftarrow$ algorithm 20
63: Output grd
64: End

Algorithm 16 RiverRawData

```
1: This function reads the input file containing raw data of the segments
2: Input: filename.dat
3: Output: XYZ, Edges, Nseg

4: Open filename
5: Read Number of segments Nseg
6: Initialize jj  $\leftarrow$  1, jjinit  $\leftarrow$  1
7: iseg  $\leftarrow$  1; inode  $\leftarrow$  1
8: for iseg in Nseg do
9:   Read number of points in this segment: Nodes
10:  for inode in Nodes do
11:    Read coordinates of inode: xi, yi, zi
12:    XYZTemp(inode)  $\leftarrow$  xinode, yinode, zinode
13:    Edge1  $\leftarrow$  jj
14:    if inode = Nodes then
15:      Edge2  $\leftarrow$  jjinit
16:    else
17:      Edge2  $\leftarrow$  jj + 1
18:    end if
19:    EdgeTemp(inode)  $\leftarrow$  Edge1 : Edge2
20:    jj = jj + 1
21:  end for
22:  jjinit  $\leftarrow$  jjinit + Nodes
23:  Edges{iseg}  $\leftarrow$  EdgeTemp
24:  XYZ{iseg}  $\leftarrow$  XYZTemp
25: end for
26: Close filename
27: End
```

Algorithm 17 CheckPointInCells

- 1: This algorithm checks point $XYpt_{x,y}$ if it falls within any, or none, of the triangular cells of the river mesh and add tags of 1 or 0 respectively..
- 2: Input: $XYZtri, ICONtri, XYpt$
- 3: Output: tag

- 4: Initialize $tag \leftarrow 0, itri \leftarrow 1$
- 5: $N_{elem} \leftarrow$ number of elements (or rows) in $ICONtri$
- 6: **for** $itri$ in N_{elem} **do**
- 7: **if** $tag = 1$ **then**
- 8: Skip
- 9: **end if**
- 10: $Row = ICONtri(itri, :)$
- 11: $pt_i = RndmCell(i)$
- 12: $V_j \leftarrow XYZtri(pt_j, 1 : end)$ $\triangleright V$ is 1×3 vector of coordinates of the three vertices in the random cell.
- 13: $x_k = V_j(k, 1) : k = 1 : 3$
- 14: $y_k = V_j(k, 2) : k = 1 : 3$

$$A = \begin{bmatrix} (x1 - x3) & (x2 - x3) \\ (y1 - y3) & (y2 - y3) \end{bmatrix}$$

$$b = \begin{bmatrix} (x - x3) \\ (y - y3) \end{bmatrix}$$

- 15: $Out = A \setminus b$
 - 16: $Out = A b$
 - 17: $r = Out(1); s = Out(2)$
 - 18: **if** $r \geq 0 \ \& \ s \geq 0 \ \& \ (r + s) \leq 1$ **then**
 - 19: $tag \leftarrow 1$ \triangleright Point falls in one of the triangles
 - 20: **else**
 - 21: $tag \leftarrow 1$ \triangleright Point doesn't fall in any of the triangles.
 - 22: **end if**
 - 23: **end for**
 - 24: End
-

Algorithm 18 GetIslandTri

1: This function generates temporary grids for dry segments (or islands) such that can be used for establishing holes for the island.
2: Input: $XYZ, Edges$
3: Output: $XY, ICON$

4: $filename \leftarrow$ algorithm 13 \triangleright Rewrite edge segment into Triangle input file
5: Set $Holes \leftarrow null$
6: $XY, ICON \leftarrow$ algorithm 12 \triangleright Includes path to Triangle.
7: End

Algorithm 19 WaterscapeTriangulation

1: This function generates temporary triangular meshes for the Waterscape
2: Input: $XYZ, Edges$
3: Output: $XY, ICON$

4: $filename \leftarrow$ algorithm 13 \triangleright Rewrite edge segment into Triangle input file
5: Set $Holes \leftarrow null$
6: $XY, ICON \leftarrow$ algorithm12 \triangleright Includes path to Triangle.
7: Set $H \leftarrow zero$
8: **for** All XY **do**
9: $XYZ \leftarrow$ Add H to $(XY)_i$
10: **end for**
11: End

Algorithm 20 AssignEFDCTags

```
1: This function assigns tags to all cells tags according to the EFDC requirements
2: Input: cells, nx, ny
3: Output: cells

4: Initialize i, j  $\leftarrow$  1
5: for j in ny do
6:   for i in nx do
7:     tags  $\leftarrow$  cells{i, j}.tags
8:     if tags = 1111 then
9:       cells{i, j}.efdctags = 5
10:    else if tags = 1101 then
11:      cells{i, j}.efdctags = 1
12:    else if tags = 1011 then
13:      cells{i, j}.efdctags = 2
14:    else if tags = 1011 then
15:      cells{i, j}.efdctags = 3
16:    else if tags = 1110 then
17:      cells{i, j}.efdctags = 4
18:    else if tags = 0000 then
19:      cells{i, j}.efdctags = 0
20:    else
21:      cells{i, j}.efdctags = 9
22:    end if
23:  end for
24: end for
25: End
```

VITA

Babatunde Atolagbe was born in Lagos Island, Nigeria and acquired elementary and secondary education in Lagos State where he lived, before moving to Kwara State to attend The University of Ilorin for undergraduate studies in Civil Engineering. Following his graduation from the university in 2015, Babatunde proceeded to Oyo State Ministry of Works and Transport where he was deployed for National Youth Service Corps (NYSC) exercise, a one-year compulsory governmental service scheme required of graduates of higher institutions in Nigeria. During this time, he served as a graduate trainee engineer and participated, under supervision, in the inspection of state infrastructure under construction and in-service.

Babatunde left for the United States in the Fall of 2017 to pursue graduate studies. He attended East Tennessee State University, Johnson City, TN and completed a semester of graduate coursework in Engineering Technology before joining The University of Tennessee – Chattanooga (UTC) as a Teaching and Research Assistant in the spring of 2018. During his time at UTC, he participated in different researches including the development of lightweight self-consolidating concrete for precast modular walls, deep-learning based condition assessment of civil infrastructure, development of programming framework for power optimization of unmanned aerial vehicles for civil applications, survey of winter maintenance strategies for open-graded friction course (OGFC) pavement and mesh representation of urban environments. In recognition of his hardwork, Babatunde was presented with the 2019 outstanding master's student award for civil engineering.