

**KARADENİZ TECHNICAL UNIVERSITY
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES**



TRABZON



KARADENİZ TECHNICAL UNIVERSITY
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES



This thesis is accepted to give the degree of

By
The Graduate School of Natural and Applied Sciences at
Karadeniz Technical University

The Date of Submission : / /

The Date of Examination : / /

Thesis Supervisor :

Trabzon

ACKNOWLEDGEMENTS

This thesis has been supported by The Scientific and Technological Research Council of TURKEY 1001 Project with the number 118Y452 and entitled as “Developing a Web Framework for 3D City Model Analyses.

First, I would like to thank my supervisor Prof. Dr. Çetin CÖMERT for his endless support during my PhD. It was a privilege to experience his guidance, suggestions, scientific enthusiasm, and I will always be honored to be his student.

I would like to express my gratitude to Mapisso software company and especially M. Emre YILDIRIM who is the founder of Mapisso. Translation of the developed algorithms to the web services was jointly developed with M. Emre YILDIRIM from Mapisso.

Of course, my special thanks go to my fellows and colleagues Res. Asst. Fatih KADI, Asst. Prof. Dr. Berkant KONAKOĞLU, Asst. Prof. Dr. Volkan YILMAZ, Res. Asst Zeynep AKBULUT and Res. Asst. Alper Tunga AKIN for making the working environment fun.

Lastly, I owe many thanks to Muhammet GÜMRÜKÇÜOĞLU for his support for almost everything.

Ziya USTA
Trabzon 2021

STATEMENT OF ETHICS

I declare that this PhD thesis submitted with the title “The Design and Development of a Web-Based 3D Geographic Information Management Framework” has been completed under the guidance of my supervisor Prof. Dr. Çetin CÖMERT. All referred information used in this thesis that is not original to this work has been indicated in the text and cited in the reference list. I have obeyed all research and ethical rules during my research, and I accept all responsibility if proven otherwise. 15/10/2021.

Ziya USTA

TABLE OF CONTENTS

	<u>Page No</u>
FOREWORD	III
STATEMENT OF ETHICS	IV
TABLE OF CONTENTS	V
SUMMARY	VII
ÖZET.....	VIII
LIST OF FIGURES	IX
LIST OF TABLES	XII
1. CHAPTER 1 GENERAL INFORMATION	1
1.1. Introduction	1
1.2. Problem Definition.....	2
1.3. Research Questions	3
1.4. Contributions.....	3
1.5. Structure and Scope of the Thesis	4
1.5.1. Structure of the Thesis	4
1.5.2. Scope of the Thesis	5
1.6. Background	6
1.6.1. Key Technologies and Concepts For 3D WebGIS Applications	6
2. CHAPTER 2 TILING, STREAMING AND DISPLAY OF 3D GEOSPATIAL DATA ON THE WEB	29
2.1. Introduction	29
2.2. Related Work	30
2.3. Methodology	38
2.3.1. Decomposition of the 3D Geospatial Data.....	38
2.3.2. Updating 3D Tileset	44
2.3.3. Implementation of the 3D Tiles Specification	47
2.3.4. Development of the RESTful Web Services.....	56
2.3.5. Tiling of the Terrain	58
2.4. Findings and Discussion	67
2.4.1. Performance Comparison of R-Tree and Adaptive QuadTree.....	67
2.4.2. Hierarchical Data Structures or Regular Data Structure	68

2.4.3.	Examining the Differences Between the Proposed Methodology and the Related Works	69
2.4.4.	Layer-Based or Object-Based Tiling Scheme	71
2.4.5.	Order of the Operation During Tiling Process	72
2.4.6.	High Precision Rendering of the 3D Tileset	72
2.4.7.	Review of the 3D Tiles Specification	73
2.4.8.	Evaluation of the Terrain Tiling and Quantized Mesh Specification.....	75
2.5.	Conclusion.....	76
3.	CHAPTER 3 WEB-BASED 3D ANALYSIS AND QUERY OF 3D GEOSPATIAL DATA.....	78
3.1.	Introduction	78
3.2.	Related Work	79
3.3.	Methodology	80
3.3.1.	Storage of the 3D Geospatial Data.....	80
3.3.2.	The 3D Intersection Algorithm	82
3.3.3.	Integration of the 3D Intersection Algorithm with Tiling and 3D Tiles	86
3.3.4.	Other Types of 3D Analyses Implemented in the Framework.....	87
3.4.	Findings and Discussion	88
3.4.1.	Performance of the 3D Intersection Algorithm.....	88
3.4.2.	Limitations	88
3.5.	Conclusion.....	91
4.	CHAPTER 4 WEB-BASED PROCEDURAL MODELLING OF 3D MODELS .	92
4.1.	Introduction	92
4.2.	Background and Related Work	93
4.3.	Methodology	95
4.3.1.	Obtaining Roof Type Information	95
4.3.2.	Extrusion and Block Model Generation.....	97
4.3.3.	Construction of the Roof Geometries	100
4.3.4.	Visualization of the 3DCM via Browser.....	105
4.4.	Findings and Discussion	106
4.4.1.	Performance Metrics of the Roof Type Prediction	106
4.4.2.	Special Cases and Floating-Point Issues	108
4.5.	Conclusion.....	109
5.	REFERENCES.....	109
	CURRICULUM VITAE	

PhD. Thesis

SUMMARY

THE DESIGN AND DEVELOPMENT OF A WEB-BASED 3D GEOGRAPHIC
INFORMATION MANAGEMENT FRAMEWORK

Ziya USTA

Karadeniz Technical University
The Graduate School of Natural and Applied Sciences
Geomatics Engineering Graduate Program
Supervisor: Prof. Dr. Çetin Cömert
2021, 115 Pages

Management of 3D geospatial data involves four distinct steps: creation, analysis, query, and visualization. However, the existing works mostly focus only on the visualization step. There are other aspects to be tackled such as interoperability, 3D analysis and query, managing varying levels of detail, and 3D model creation.

Imposed by web environments, the main problem with the management of large-scale geospatial 3D data is the requirement to work via smaller chunks of the data. Browsers impose memory limits and networking protocols impose traffic management restrictions.

Hence in this thesis, for the first time in the literature a web framework has been designed and developed for management of 3D geospatial data using only open-source software components. End-users can create their 3D models, tile 3D data, analyze and query 3D data and can visualize 3D without any software component of plug-in installation. Using modelling component of the framework, 3D models can be generated procedurally using only 2D data for the first time on the web. Using the tiling component, large-scale 3D geospatial data can be decomposed to tiles and the tileset can be visualized via browsers efficiently. The efficiency comes from that developed framework guarantees the rendering performance as 60fps while displaying tiles. Using analyze component, 3D analyzes, and queries can be performed, and result can be visualized on the web. To perform 3D analyzes efficiently a new 3D intersection algorithm has been developed and used in the analyze component of the developed framework. 3D Tiles specification and Quantized Mesh specification have been implemented while developing the framework to ensure the interoperability of the framework with other software components.

Key Words: 3D WebGIS, 3D Geospatial Data, 3D Analyzes, 3D Modelling, 3D Data Structures

Doktora Tezi

ÖZET

WEB TABANLI 3B COĞRAFI BİLGİ YÖNETİMİ İÇİN BİR ÇATI TASARIMI VE
GELİŞTİRİLMESİ

Ziya USTA

Karadeniz Teknik Üniversitesi
Fen Bilimleri Enstitüsü
Harita Mühendisliği Anabilim Dalı
Danışman: Prof. Dr. Çetin CÖMERT
2021, 115 Sayfa

3B Coğrafi verinin yönetimi dört ayrı adımdan oluşmaktadır: Oluşturma, analiz, sorgu ve görselleştirme. Mevcut çalışmaların çoğunluğu sadece görselleştirme adımına odaklanmaktadır.

3B veri yönetimindeki ana problem web ortamları tarafından dayatılan daha küçük boyutlu verilerle çalışma zorunluluğudur. Tarayıcılar hafıza sınırı kısıtı dayatırken ağ protokolleri veri trafiği yönetimi sınırlamaları dayatmaktadır.

Bu tezde, literatürde bir ilk olarak, 3B konumsal verinin web tabanlı yönetilmesi için bir web çatısı tasarlanmış ve sadece açık kaynak kodlu yazılım bileşenleri ile geliştirilmiştir. Son kullanıcılar geliştirilen bu çatıyı kullanarak herhangi bir yazılım bileşeni ya da eklenti kurulumu gerekmeden 3B modellerini oluşturabilir, bölümleyebilir, üzerlerinde analiz ve sorgular gerçekleştirebilirler. Çatının modelleme bileşeni kullanılarak literatürde ilk defa bu bileşen ile sadece 2B veri girdisinden prosedürel modelleme yöntemi ile 3D modeller web tabanlı olarak üretilebilir. Bölümleme bileşeni kullanılarak büyük boyutlu 3B konumsal veri bölümlenebilir ve oluşturulan bölümler etkin bir şekilde tarayıcıda görüntülenebilir. Etkinlik, geliştirilen çatının görüntüleme performansı olarak 60fps değerini garanti edebilmesinden kaynaklanmaktadır. Analiz bileşeni kullanılarak 3B analiz ve sorgular gerçekleştirilebilir ve sonuçlar web tabanlı olarak görüntülenebilir. Analizleri etkin bir şekilde gerçekleştirebilmek için yeni bir 3B kesişim algoritması geliştirilmiştir ve çatının analiz bileşenine entegre edilmiştir. Çatının diğer yazılım bileşenleri ile birlikte çalışabilmesi için çatı geliştirilirken 3D Tiles ve Quantized Mesh standartları kullanılmıştır.

Anahtar Kelimeler: 3B WebCBS, 3B Konumsal Veri, 3B Analizler, 3B Modelleme, 3B Veri Yapıları

LIST OF FIGURES

	<u>Page No</u>
Figure 1. Visualization Pipeline with Hardware and Software Support.	8
Figure 2. Camera parameters that define view frustum	9
Figure 3. View Frustum (top) and frustum culling (bottom) (Om användning, 2010).....	10
Figure 4. An object stored as coordinate sequences in a GeoJSON file.	12
Figure 5. An object stored as vertex and index arrays in an OBJ file.....	12
Figure 6. Scene graph representation of the universe.	13
Figure 7. Distributed Visualization Pipeline (Doyle and Cuthbert, 1998).....	18
Figure 8. Classification of client-based rendering methods (Evans et al. 2014).....	19
Figure 9. A small part of a CityGML file.	22
Figure 10. A small part of a X3D file.	23
Figure 11. A small part of a CityJSON file.....	24
Figure 12. Binary data and its encoding in a glTF file.....	25
Figure 13. glTF file and its contents.	25
Figure 14. Regular tiling of the CityGML dataset (Gesquiere and Manin, 2012)	32
Figure 15. Regular tiles can be seen in the web map client of 3DCityDB.....	33
Figure 16. Rendering Strategy for Progressive Visualization (Gaillard et al., 2015)	34
Figure 17. Streaming algorithm loads additional tiles and removes that are not visible based on camera position (Krämer and Gutbell, 2015).	35
Figure 18. 3D geospatial data decomposed to R-Tree structure	41
Figure 19. Pseudo code for R-Tree construction.....	42
Figure 20. Tile borders overlap and features are always completely inside of a node.	43
Figure 21. QuadTree decomposition of the 3D geospatial data.....	44
Figure 22. Updating tile borders in order to preserve object integrity.....	44
Figure 23. Pseudo code for Adaptive QuadTree construction	45
Figure 24. Pseudo code for adding new feature in R-Tree.....	46
Figure 25. Pseudo code for adding a new feature in Adaptive QuadTree	47
Figure 26. Pseudo code for removing a feature in R-Tree and Adaptive QuadTree.....	47
Figure 27. A part of a tileset.json file that describes 3 levels hierarchy	48
Figure 28. UML Class Diagram for A Tileset in 3D Tiles	49
Figure 29. 3D polygon surfaces (left), triangulated polygons (right)	50

Figure 30. Rendering without vertex normals (left) and with vertex normals (right).....	51
Figure 31. Buildings over the terrain (top), buildings clamped to the terrain (bottom).....	52
Figure 32. Relationship between geometric error and space screen error (URL-19).	53
Figure 33. A tile and its child rendered using additive method	54
Figure 34. View Frustum and tiles (URL-19)	55
Figure 35. View frustum and tiles (URL-19).....	56
Figure 36. Displaying varying LODs according to the distance of the camera	57
Figure 37. Generated tileset.json file for a multi LOD dataset	57
Figure 38. System Overview	59
Figure 39. TileMap Diagram according to TMS (TMS, 2010).....	60
Figure 40. First 3 levels of tiles in the quantized-mesh specification	61
Figure 41. layer.json file for our terrain dataset.....	61
Figure 42. Step by step RDP Line Simplification. Green points are selected as points to keep and red points are removed during the simplification process.....	63
Figure 43. The blue point is invisible to the viewer due to the spherical shape of the earth (URL-21).....	65
Figure 44. Computation of Point P, the HOP (URL-22).....	66
Figure 45. QMesh class used for storing terrain data in quantized mesh format.....	68
Figure 46. Testing the rendering performance	72
Figure 47. Concepts and Technologies Used in 3D Tiles Standard.....	74
Figure 48. Intersection of triangles (Yellow) with tile borders (Green)	76
Figure 49. Cracks on the left and bottom edges of the tile.....	77
Figure 50. An example of a single document in the “tiles” collection.....	82
Figure 51. An example of two documents in the “attributes” collection	83
Figure 52. Sweep Plane and collision detection.....	84
Figure 53. Half-Edge data structure	85
Figure 54. Two objects are about to intersect.	86
Figure 55. Intersection is calculated and rendered as a red polygonal mesh.	86
Figure 56. A sphere and a cylinder constructed as a result of 3D Buffer analysis	88
Figure 57. Non manifold shape. Red edge is incident to four faces	90
Figure 58. A special case none of the vertices of the objects are inside the other object ...	90
Figure 59. Training and Prediction Processes.....	97

Figure 60. Calculation of the surface normal.....	98
Figure 61. 2D building footprints (left) 3D block model (right).....	98
Figure 62. Condominium unit plan (Çağdaş, 2012).....	99
Figure 63. Polygons (left) multiplied and translated (middle) condominium unit in 3D..	100
Figure 64. 3D condominiums stored in the CityJSON file.	101
Figure 65. Input polygon (green), wave fronts (purple lines) and SS (red lines).....	102
Figure 66. Edge Event (Yellow edge shrinks to zero at the orange point)	103
Figure 67. Split Event (yellow edge meets red reflex vertex and this creates blue polygon).	103
Figure 68. Sweep Line Process.	105
Figure 69. General System Architecture.....	106
Figure 70. Red roofs: Flats, Green roofs: Hippeds, Purple roofs: Gables.....	107
Figure 71. Performance metrics of training process.	108
Figure 72. Degeneration. Two Edge Events must meet on the same point (left) but not properly they do not meet (right).....	110

LIST OF TABLES

	<u>Page No</u>
Table 1. WebGL libraries and their major capabilities.	16
Table 2. Summary of the related Works	37
Table 3. Summary of the software components.....	38
Table 4. System Specifications of the Test Machine	39
Table 5. Performance metrics of the data structures	69
Table 6. Example of Half-Edge Data Structure	85
Table 7. Performance of the 3D Difference Based on 3D Intersection Algorithm.....	89
Table 8. Confusion matrix of prediction process	108
Table 9. Performance metrics of prediction process	109

1. CHAPTER 1 GENERAL INFORMATION

1.1. Introduction

Forming the basis for many spatial information processing applications, digital elevation models (DEM) refer to a 2.5D representation. DEM is based on the principle that the topographic surface is defined mathematically by represented with a function (f). Determining the function “f” is a “surface fitting” problem. With a mathematically defined surface, it is now possible to perform many analyses or applications that this definition allows. However, in such a definition, since the z value of any point will be calculated by the formula " $z = f(x, y)$," a single z value can be obtained against the same x and y values. Such a definition or representation is called 2.5D. In a 3D representation, multiple z values can be obtained against the same x and y values. In other words, in contrast to 2.5D representation, 3D representation allows in which not only the surface but above and below the surface can be defined. Therefore, in the analyses which require 3D representation, DEMs will be insufficient to use. Analysis involving objects that need to be defined above and below the surface, such as shadow analysis, various geological analysis, and airflow analysis, will require 3D representation.

Today, the main input data for 3D GIS applications that require 3D representation are 3D City Models (3DCM). A 3DCM is a digital representation of city objects with three-dimensional geometry and attribute information, with buildings as the most prominent feature.

With the advancement in 3D Graphics and computational capacities of computers, the use of 3DCMs is becoming increasingly common. Many cities around the world are adopting their own 3DCMs. Even in Turkey, which has always been a problematic country about data acquisition and sharing, with a nationwide project called "3D City Models and Cadastre", 3DCMs of every province are being produced.

With rapid developments in web technologies such as HTML5 and WebGL, WebGIS applications have replaced desktop applications for 3D GIS. The ability to visualize 3DCMs and interact with 3DCMs via browsers has become an exciting and trending topic (Rodriguez et al., 2013). Access to 3DCMs from internet browsers enables the use of 3DCMs by a large mass of professionals who are not experts in spatial information but who can benefit from

3DCM in their own studies (Prandi et al., 2015). With the dramatic increase in using 3DCMs and advancements in web technologies, a new research topic called 3D WebGIS has emerged.

1.2. Problem Definition

Despite the importance of web-based management of 3DCMs, this topic has not been investigated extensively. Management of 3DCMs involves four distinct steps; creation, analysis, query and visualization. However, the existing 3D WebGIS works mostly focus only on the visualization step. Although 3D visualization is an important aspect of 3DCM management, there are other aspects to be tackled such as interoperability, 3D analysis and query, managing varying levels of detail, and 3D model creation.

Imposed by web environments, the main problem with the management of large-scale geospatial 3D data is the requirement to work via smaller chunks of the data. Browsers impose memory limits and networking protocols impose traffic management restrictions. Thus, algorithms will have to figure out, either during visualisation or analyses, the affected chunks and manage the operation accordingly. Some examples to the management difficulty when working with the smaller chunks, called “tiles”, would be to preserve object integrity due to tile borders and updating the tiles’ contents when their data is changed.

The interoperability problem refers to the neglectance of this issue in most of the related work. The way of developing interoperable applications is to employ standards. One problem here is the immaturity of the 3D geospatial data standards. The immaturity is due to by either their definition or the limited number implementations.

Concerning 3D analyses and query, the issue is whether the analyses are carried out in an “online” or “offline” mode. In the offline mode, the analyses are performed on a desktop 3D software and the result is “packed” and sent to the client for visualisation. The so-called “3D web GIS” implementations generally operate on this mode. This makes the “real time playing with the 3D model” awkward; when a user wants to modify the parameters of an analysis he has to re-perform the analysis on the desktop and repeat the remaining steps. In addition, user needs and works with two different software. This is in no way online with the “true” web based operational mode where all the interactions of the user are over a single web based software. This way, a user can perform his/her analyses using the same web-

based software in an online mode. The needed here are the mechanisms of organizing the tiles and tiling accordingly, which have been prototypically implemented in this thesis.

With respect to managing varying levels of detail the issue revolves around being multiscale or multi-representational. In the multiscale mode, the data is generally organized into different tileset for each different level of detail. Whereas in the multi-representation the different levels of detail can be handled in a single tileset. There are pros and cons of each alternative, which will be discussed later in the thesis. A thorough discussion of the topic is missing in the literature.

Concerning the creation of 3D city models, one part of the issue is the lack of open-source components. To our knowledge there is only one open-source tool from Delft University. Other well-known tool is the CGA (Computer Generated Architecture) component of ESRI CityEngine, which is not web based currently and a commercial product. In this thesis a web-based component has been developed as a REST web service.

1.3. Research Questions

Research questions focus on designing and implementing a 3D SaaS tool, which realizes the management of large-scale 3D geospatial data online by examining today's modern web technologies and 3D GIS capabilities. In this context, research questions have been explained below.

The thesis investigates the below research questions:

- 1- What are the most up-to-date and most suitable web technologies preferably open-source for the management of 3DCMs?
- 2- What are the open standards and formats that can be used for the streaming of the 3DCMs on the web?
- 3- Which tiling scheme must be used for decomposition of the data?
- 4- What would be the “right” implementation of different levels of detail? Should it be multi-scale or multi-representational?
- 5- Are “online” web-based 3D analyses viable?
- 6- Is a web based procedural modelling component viable for 3D model generation from 2D?

1.4. Contributions

- Investigation of modern web technologies and concepts that can be used in 3DCM management in Chapter 1.
- A comprehensive literature review for web based management of 3DCMs in Chapter 1, Chapter 2, Chapter 3, and Chapter 4.
- Investigation of spatial data structures for the tiling of 3DCMs in Chapter 2.
- Design and implementation of an interoperable 3D tiling system for large-scale 3D geospatial data based on OGC 3DTiles specification supports multiple spatial data structures in Chapter 2.
- Developing a novel, fast algorithm 3D intersection for the web-based real-time 3D overlay analyses of 3D polygonal meshes in Chapter 3.
- Prototype implementation of the developed 3D intersection algorithm for the realization of web-based 3D Analysis in Chapter 3.
- Design and implementation of web-based procedural 3D model generation in Chapter 4

1.5. Structure and Scope of the Thesis

1.5.1. Structure of the Thesis

This thesis is based on papers I have published during my PhD and also based on projects that I carried on during my PhD. Most of the parts have been updated and extended to include the latest research and developments. These updates and extensions cover the timeline that begins at the publication dates and deadlines of the projects until the publication of the thesis. Also, some parts of the papers and projects have been distributed across multiple chapters of the thesis.

The thesis has been organized into 4 chapters. Chapter 1 discusses the fundamentals that must be known to understand before diving deeper into the proposed research and works such as key web technologies for web-based management of 3D geospatial data. Chapter 2 deals with streaming and visualization of large-scale 3D geospatial data on the web. Chapter 2 discusses how this challenging task can be realized using modern web technologies and proposes a tiling and a rendering service for implementation of these purposes. Chapter 3

concerns web-based 3D analysis and query capabilities and how they can be implemented in web environments. How the visualization of analysis and query results can be integrated with proposed tiling and rendering services of Chapter 2. Chapter 4 discusses procedural generation of 3D models from 2D datasets and its implementation in a web environment. Also, the web service that was designed and implemented for this purpose is explained.

1.5.2. Scope of the Thesis

This thesis is concerned with the management of the large-scale 3D geospatial data online. For this purpose, a web framework is developed. While developing this framework following objectives have been considered.

Only open-source software components must be used to develop the framework.

Software components should be developed in accordance with 3D standards whenever possible for interoperability.

Mostly, 3D geospatial data is stored and published as 3DCMs by vendors and geometries of the city objects are stored as 3D polygonal meshes in 3DCMs. Although, 3D geospatial data can be in many different representations such as Voxels, Point Clouds, B-Rep (Boundary Representation), Constructive Solid Geometry (CSG) and etc., these representations are not considered in this thesis; only 3D polygonal mesh representation is considered. No doubt, the proposed web framework could be extended to include the other representations as well. However, this thesis focuses on decomposition and management of the large-scale 3D geospatial data on the web rather than its different representations.

Analysis is the analytic process that examines geometric or topological properties of features and extract new information. There are a wide range of different analysis techniques in the GIS domain such as overlay analysis, proximity analysis, network analysis etc. In this thesis only 3D overlay analyses which are 3D intersect, 3D clip, 3D erase, and 3D difference are considered and implemented. 3D topological operations, 3D map algebra are considered out of scope. Additionally, poor data may cause errors in the analysis. Correctness of the data is the responsibility of the end user. None of the developed software components try to detect and repair the errors in the data for you. 3D data quality is not addressed within the thesis since it is a completely different research topic

There are many roof types and modelling each of them requires a different algorithm. In this thesis, only the four most used roof types that are explained in Chapter 4 are

considered. This thesis focuses on modelling of roofs using only widely available 2D data such as building footprints and satellite imagery. Modelling some other roof types requires additional data such as floor plans and architectural drawings which are not widely available.

1.6. Background

1.6.1. Key Technologies and Concepts For 3D WebGIS Applications

1.6.1.1. HTML5 and WebGL

HTML is the World Wide Web's core markup language used to structure and present content on the World Wide Web. HTML is the standard markup language for documents designed to be displayed in a web browser and it is an open standard. (URL-1). HTML5 is the fifth version of this standard. The biggest contribution of HTML5 in terms of 3D WebGIS applications, is the new "canvas" element used to render 3D contents on web pages.

Canvas element provides an API and contexts to draw graphics. For 3D graphics, it provides WebGL contexts. WebGL is a cross-platform, royalty-free web standard for a low-level 3D graphics API based on OpenGL ES, exposed to ECMAScript via the HTML5 Canvas element (Khronos, 2014).

Before the invention of WebGL, in order to visualize 3D content via browsers and developing 3D web applications, additional plug-ins must be used or standalone software had to be installed on the client's device, such as Flash and Silverlight. Plugins for browsers such as Cortona3D (URL-2), FreeWRL (URL-3), or Java applets such as XNavigator (URL-4) have been used for visualizing 3D contents on the web. Nasa WorldWind (URL-5) and Google Earth (URL-6) were able to work web based but had to be downloaded and installed. After being redeveloped using WebGL, Google Earth now can be used without any additional installation. Another example is Unity Web Player which is now deprecated and had to be installed for displaying video games developed using Unity3D in the browsers. Many 3D web applications such as GeoPortail (URL-7) and 3D Macau (URL-8) require the installation of an additional software called Terra Explorer as a 3D viewer on the client's device. Open3DGIS (O3DG) (URL-9) application requires plug-in installation.

One of the major advantages of WebGL is that it is supported by all browsers; hence it does not require a plug-in. Using HTML5 and WebGL together, it is possible to develop 3D web applications without additional software or plug-in installation.

WebGL provides access to developers via JavaScript from the browser to the client's GPU (Parisi, 2014). Hence, it enables GPU-accelerated algorithms to visualize 3D data and perform operations on 3D data (Taraldsvik, 2011) that improves performance.

WebGL is a de facto standard for 3D graphics on the web. There are two versions of WebGL. WebGL 1.0 is based on OpenGL ES 2.0 and WebGL 2.0 is based on OpenGL ES 3.0. WebGL 2.0 is released in 2017 and brings some improvements to WebGL 1.0, such as multiple draw buffers that enables drawing multiple buffers at once from a shader and instanced drawing that enables rendering multiple copies of the same 3D content. At the time of writing this thesis, WebGL 2.0 is not supported by Safari, which means that 3D web applications developed with WebGL 2.0 will not work properly in Safari. In contrast to Safari, most modern browsers and libraries support WebGL 2.0 and WebGL 2.0 will become more common shortly.

WebGL is based on OpenGL, originally developed in 1992 and started to get old in today's technology stack. Today, modern GPUs are more complex and powerful than 29 years ago. To better take advantage of modern GPUs' advanced features, new graphic APIs have been developed after OpenGL. These APIs are often referred to as "modern graphic APIs".

One of the modern graphic APIs is Vulkan (URL-10). Vulkan is developed by Khronos, the group behind OpenGL as a cross-platform graphic API that can work on all systems and released in 2016. While Vulkan is being developed by Khronos, "DirectX12" (URL-11) has been developed by Microsoft and "Metal" (URL-12) has been developed by Apple. In contrast to Vulkan, DirectX12 has been developed to be used in the Microsoft platform and Metal has been developed to be used in macOS and IOS platforms. Compared to OpenGL, they are more low level and performant.

At the time of writing this thesis, there is ongoing work about a new graphic API for browsers based on modern Graphic APIs called WebGPU currently being developed by W3C GPU for the Web Community Group (URL-13). Although specification has not been completed yet, it has already started to be supported by the major browsers such as Safari, Firefox, and Chrome. It will be the successor of WebGL in the near future.

1.6.1.2. Visualization Pipelines in Web Applications

The visualization pipeline consists of three major conceptual steps; Filtering, Mapping, and Rendering (Doyle and Cuthbert 1998, Hildebrandtand 2014, Klimke 2019, Koukofikis et al., 2018). Filtering and mapping steps are executed on CPU by utilizing a data structure called scene graph while the rendering process is executed on GPU utilizing graphic APIs (Figure 1). These steps have been explained in more detail in the following sections.

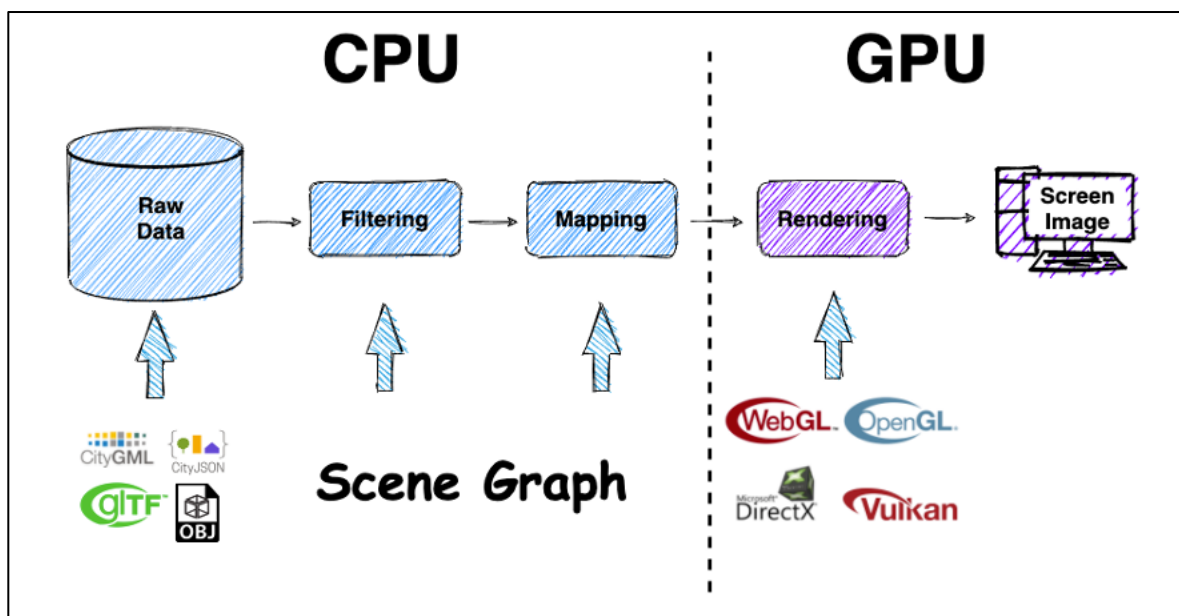


Figure 1. Visualization Pipeline with Hardware and Software Support.

1.6.1.2.1. Filtering Step

To be able to display objects from different angles a “synthetic” camera object is used. Camera parameters define the visible area which is referred to as view frustum or viewing frustum in Computer Graphics (Figure 2). In the filtering step, objects that fall into the view frustum are selected from a larger geospatial dataset. This process is often called “frustum culling” in Computer Graphics.

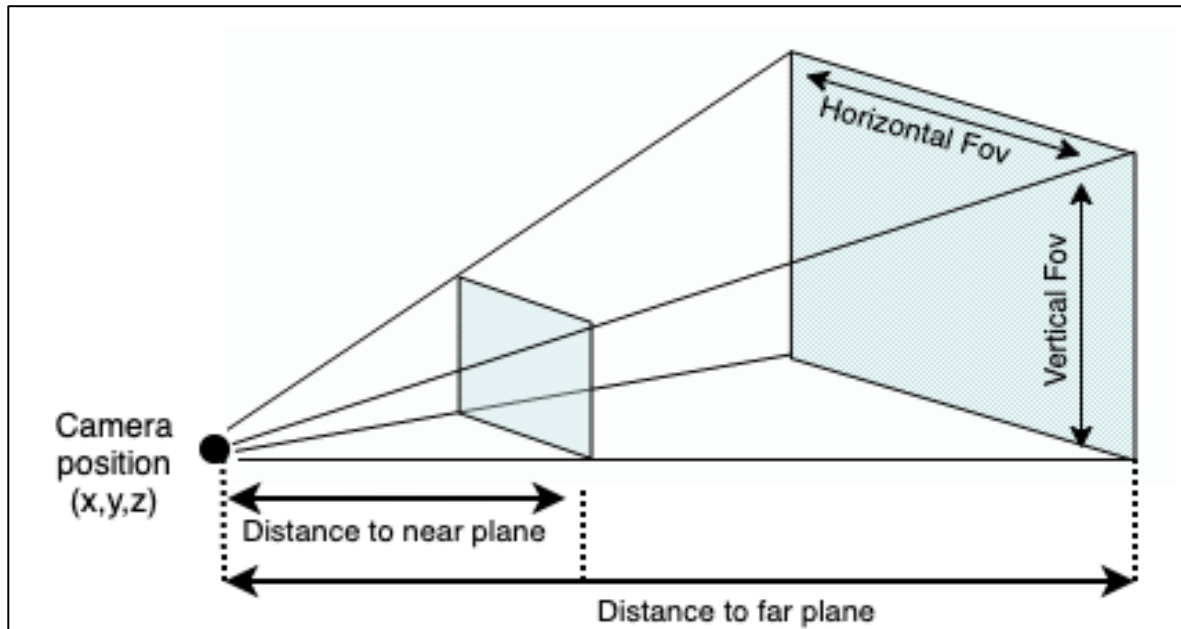


Figure 2. Camera parameters that define view frustum

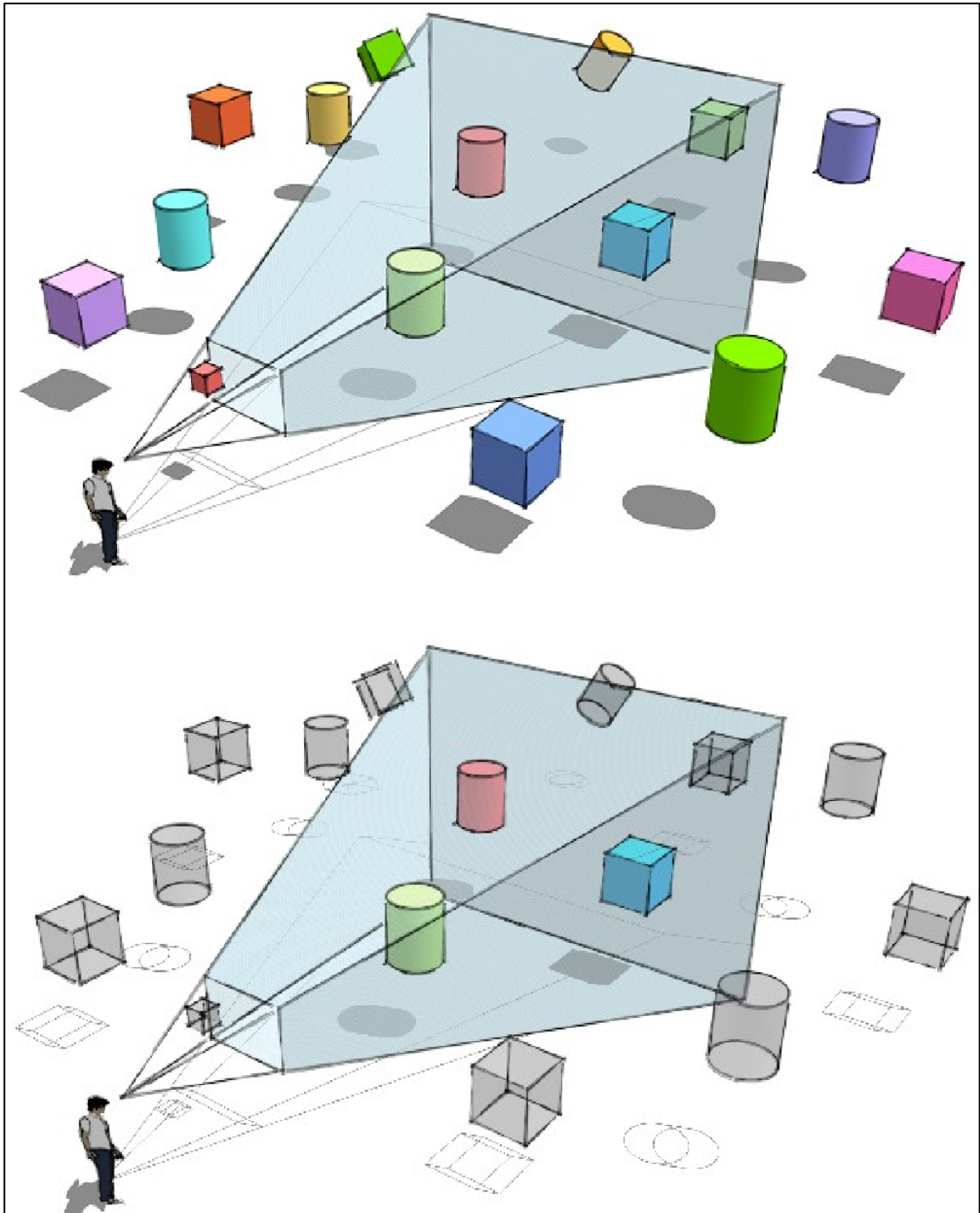


Figure 3. View Frustum (top) and frustum culling (bottom) (Om användning, 2010)

For the frustum culling process each vertex of the data is tested against visible volume. Using camera parameters, the view frustum is constructed as a polyhedron. Then performing ray-surface intersection tests it is determined whether each vertex is inside or outside of the view frustum. These spatial queries are computationally expensive and must be optimized

well especially in a web environment. This is where scene graphs which are explained more detailed in the next sections come to usage.

Additionally, in the geospatial applications filtering is not always made according to camera parameters. Culling process can be done according to the results of an analysis or query. After an analysis and query, in order to show results to the user, only the features that have the desired results are selected.

1.6.1.2.2. Mapping Step

In this step, filtered geometry is mapped to visual representations for rendering. In geospatial formats, geometric data is usually stored as ordered coordinate arrays. These coordinate arrays are stored according to a winding order “clockwise” or “counterclockwise” can be seen in Figure 4. Rendering step is done using graphical APIs on the GPU and in the graphic APIs, geometric data is stored as vertex and index arrays (Figure 5). Hence, in the mapping step, geometric data in the form of coordinate arrays is converted to vertex buffers and index buffers which graphic APIs accept. Thus, filtered geometric data is mapped to data types on the graphic API such as vertex buffer object (VBO) and index buffer object (IBO). Additionally, in this step, all of the other attributes that affect the appearance of the objects are mapped such as colour, opacity, and texture.

```

1  {
2  "type": "FeatureCollection",
3  "name": "3DGEJSON",
4  "crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84" } },
5  "features": [
6  { "type": "Feature",
7  "properties": { "height": 6 },
8  "geometry": { "type": "Polygon",
9  "coordinates": [ [ [ 39.808496626334296, 40.953373485626066, 260.385345458984375 ],
10 [ 39.808358935756708, 40.953409568086698, 260.385345458984375 ],
11 [ 39.808376630714534, 40.953448344424622, 260.385345458984375 ],
12 [ 39.808463781563923, 40.953426661825389, 260.385345458984375 ],
13 [ 39.808482432013733, 40.953465882115381, 260.385345458984375 ],
14 [ 39.808531220868694, 40.953454013851363, 260.385345458984375 ],
15 [ 39.808496626334296, 40.953373485626066, 260.385345458984375 ] ] ] } },
16 { "type": "Feature",
17 "properties": { "height": 9 },
18 "geometry": { "type": "Polygon",
19 "coordinates": [ [ [ 39.82724925660343, 40.95292664698114, 87.627105712890625 ],
20 [ 39.82721372006249, 40.952829561967668, 87.627105712890625 ],
21 [ 39.827199386566754, 40.952832546886953, 87.627105712890625 ],
22 [ 39.827118138110706, 40.952850778411069, 87.627105712890625 ],
23 [ 39.827145268158361, 40.952930996218484, 87.627105712890625 ],
24 [ 39.827151042798135, 40.952946442395508, 87.627105712890625 ],
25 [ 39.82724925660343, 40.95292664698114, 87.627105712890625 ] ] ] } },

```

Figure 4. An object stored as coordinate sequences in a GeoJSON file.

```

o bina1.1
v 3703352.439709047 3089915.6773491837 4159522.7048030565
v 3703353.1598360166 3089908.1934797885 4159527.5901352465
v 3703345.7374877417 3089910.5845264196 4159532.389927235
v 3703344.985621803 3089918.0555725005 4159527.542126134
v 3703355.918484094 3089918.5798875536 4159526.638426727
v 3703356.638611064 3089911.0960181584 4159531.523758917
v 3703349.216262789 3089913.4870647895 4159536.3235509056
v 3703348.46439685 3089920.9581108703 4159531.4757498046

f 1 2 6 5
f 2 3 7 6
f 3 4 8 7
f 4 1 5 8
f 1 2 3 4
f 5 6 7 8

```

Figure 5. An object stored as vertex and index arrays in an OBJ file.

1.6.1.2.3. Scene Graphs

Scene Graph is a data structure, which is mainly used to describe the objects, attributes, and object relationships in a scene (Xu et al., 2020). Scene graph is a directed acyclic graph where each node represents a local space in a 3D scene. Scene graphs are constructed in a

hierarchical manner and contents of a node in a scene graph must be completely inside of a bounding volume of the parent node.

Using the spatial and hierarchical properties of the nodes in a scene graph, filtering and mapping steps of the visualization pipeline can be optimized. To better understand a scene graph, let's think of it with an example. In this example we want to represent and display our universe as a 3D scene. To construct a scene graph, stars, galaxies, and planets are arranged hierarchically and can be seen in Figure 6. The current view frustum intersects only objects which take place in Europe (Figure 6).

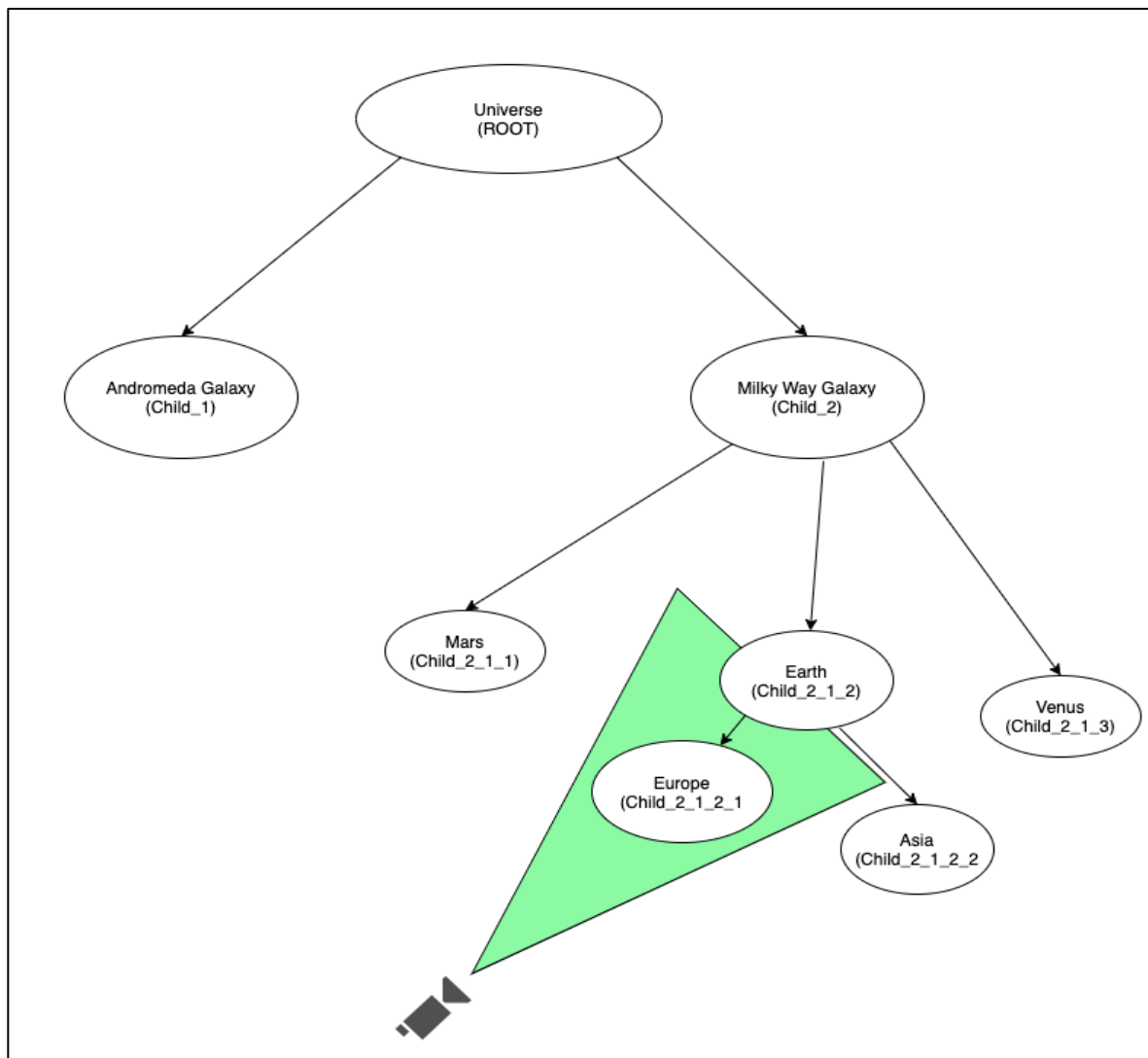


Figure 6. Scene graph representation of the universe.

Instead of doing visibility tests for each object in the scene for frustum culling, the scene graph is traversed and only the contents of the nodes which intersect with view frustum

and its children are tested. These nodes are “Earth” and “Europe” for the given example. There is no need to perform visibility checks for the other nodes. If a parent is outside the frustum this means its children are also outside the frustum. Thus, the number of the visibility tests decrease significantly, and performance improves.

Another optimization technique that can be implemented by imposing scene graphs is “state sorting”. State sorting is grouping objects that have similar graphical properties such as texture and materials into a node while constructing a scene graph. Hence similar objects are batched and drawn together with the least number of state changes such as set colour, set texture, which are expensive in GPU.

In the scene graph, there are three types of nodes as the root node, non-leaf nodes, and leaf nodes. The root node represents the whole collection of objects in the 3D scene. Non-leaf nodes are internal nodes of scene graphs. They can contain any number of children and represent the logical and spatial aggregation of objects. Leaf nodes are the bottom nodes in the scene graph, which have no children.

Scene graphs can be constructed in two ways. One approach is to store geometry only in leaf nodes. At the upper nodes in the hierarchy, aggregation is done using pointers to leaf nodes. In the second approach, every node in the scene graph can store its own geometry. The first approach increases storage efficiency by storing geometry only once but requires traversing the scene graph at the runtime in order to find the relevant data for the upper nodes in the hierarchy. The second approach does not require traversing scene graphs for finding relevant data of the node, because every node stores its own geometry; hence, it is more performant than the first approach. Nevertheless, it decreases storage efficiency because some geometries are stored more than once. The choice between the two approaches is up to the requirements of the application.

1.6.1.2.4. Rendering Step

Rendering is the last step of the visualization pipeline. In Computer Graphics, the term "rendering" refers to the process of producing 2D images from a 3D scene. A 3D scene that is produced after the filtering and mapping steps is sent to the GPU for the render process. Using visual elements such as position, colour, opacity and texture, a 2D projection of a 3D scene is produced according to a viewpoint using graphic APIs that are compiled on the GPU.

For rendering, two small pieces of the program run on the GPU, a vertex shader, and a fragment shader. Vertex shader takes vertex array, index array, normals, colour, texture coordinates and calculates screen coordinates of vertices. A vertex shader runs once for each vertex of a 3D scene. Vertex shader processes three types of data; attributes, uniforms and varyings. Attributes are used to store information that can differ for each vertex such as position, colour, vertex normals, texture coordinates. Uniforms are used to store information that is the same for each vertex such as transformation matrices and lightning positions. Varyings are used to store information that is passed from vertex shader to fragment shader.

Fragment shader runs once for each pixel of the scene. It takes information in the forms of uniforms, varyings, and pixels then calculates colour and depth values for each pixel.

In WebGL, vertex shader and fragment shader are written in a language called GLSL (Graphics Library Shading Language) It is a C type low-level language that compiles and runs on GPUs.

1.6.1.2.5. WebGL Libraries

Not surprisingly, many JavaScript libraries have been developed on top of WebGL that abstract low-level coding and provide high-level APIs while still making it possible to enjoy the benefits of WebGL. These libraries significantly reduce the application development time. They automatically perform filtering, mapping and rendering steps according to data and parameters supplied to them. Using only WebGL without JavaScript libraries, many works that these libraries do under the hood must be implemented by the developer. For instance, lightning must be explicitly defined by implementing an illumination model on your own. Vertex and fragment shaders must be written in GLSL. All of these low-level graphical processes make WebGL very verbose. Because of this nearly all of the web-based 3D applications in the literature use WebGL libraries. To give a complete list of WebGL libraries is out of the scope of this thesis, however, it is worth mentioning some of them which are open source and used in geospatial applications.

There are many WebGL libraries as well. Three.js, Cesium.js, Babylon.js, Deck.gl, Harp.gl, MapboxGL.js and iTowns are worth mentioning among them. Evans et al. (2014) have surveyed browser-based rendering approaches that include some 3D formats and WebGL libraries. Also, Kramer et al. (2015) tested some of the WebGL libraries on real-world geospatial use case scenarios. Since new libraries have been developed after Kramer

et al. (2015) and Evans et al. (2014), a new comparative summary in terms of WebGL libraries has been given in Table 1. For 3D WebGIS applications, there is no one fit for all WebGL libraries; hence selection of the library highly depends on the use case and application requirements. WebGL library should be chosen by considering some key aspects for a 3D geospatial application such as 3D format support variety, low-level access capability to WebGL, WebGPU support, the capability to connect existing OGC services and support for 3D streaming standards.

Table 1. WebGL libraries and their major capabilities.

WebGL Library	Support for glTF	Low-Level Access to WebGL	Support for WebGPU	WFS, WMS Support	3D Tiles or I3S Support
Three.js	Yes	Yes	Partially	No	No
X3DOM	Yes	No	No	No	No
Cesium.js	Yes	No	No	Yes	Yes
Deck.gl	Yes	Yes	No	No	Yes
MapboxGL.js	No	Yes	No	Yes	No
Harp.gl	Yes	Yes	No	No	No
Babylon.js	Yes	Yes	Yes	No	No
iTowns	Yes	Yes	No	Yes	Yes

glTF is the most efficient 3D format for web applications that utilize WebGL today. MapboxGL.js has a big limitation for directly loading 3D models in glTF format. Thanks to its low-level access to WebGL, glTF can be parsed and loaded with MapboxGL.js but this attempt will require a lot of low-level coding which the primary purpose of the libraries is to avoid. In that case, filtering and mapping steps must be done explicitly by the application

developer. Low-level access to WebGL means the ability to create custom shaders for specific use cases. In this context, all of the libraries except X3DOM and Cesium.js provide low-level access to WebGL by allowing creation of custom shaders and give more control to application developers for visualization.

Considering the WebGPU will be the new standard graphic API and supported by all major browsers in the near future, it is important that libraries should support WebGPU in terms of reusability of the previously written rendering code using these libraries.

Another important aspect is the ability to visualize data which is published as WFS and WMS. This makes it possible to render DTMs which are published as WFS or WMS as basemaps underneath the 3DCMs. Hence, available data already published as WFS, or WMS can be visualized easily. This increases interoperability of the applications and makes it possible to use already published data in the applications. In this context, Cesium.js, MapboxGL.js and iTowns, which are developed geo applications in mind, support this feature.

3D streaming standards define organization of the 3D geospatial data on the server and delivery formats for geo data. Hence, the 3D geospatial data organized according to these standards can be rendered by clients which supports these standards easily. When the data changes, the same rendering code will work perfectly as long as data is organized according to these standards. Otherwise, low level code that contains render logic for a hierarchical tileset must be coded explicitly by the application developer.

1.6.1.2.6. Distribution of the Visualization Pipeline

In a web application, the visualization pipeline steps described in previous sections could be implemented in a distributed manner. Depending on the steps that are implemented on the client or server, client-server architectures classified into three types in the context of visualization (Klimke, 2019) (Figure 7):

- 1- Thick client - Thin Server: Client fetches filtered data from server and mapping and rendering is implemented in the client.
- 2- Medium Client - Medium Server: Client fetches mapped data and renders it.
- 3- Thin Client - Thick Server: The client fetches the rendered image from the server. All three steps of the visualization pipeline are implemented on the server.

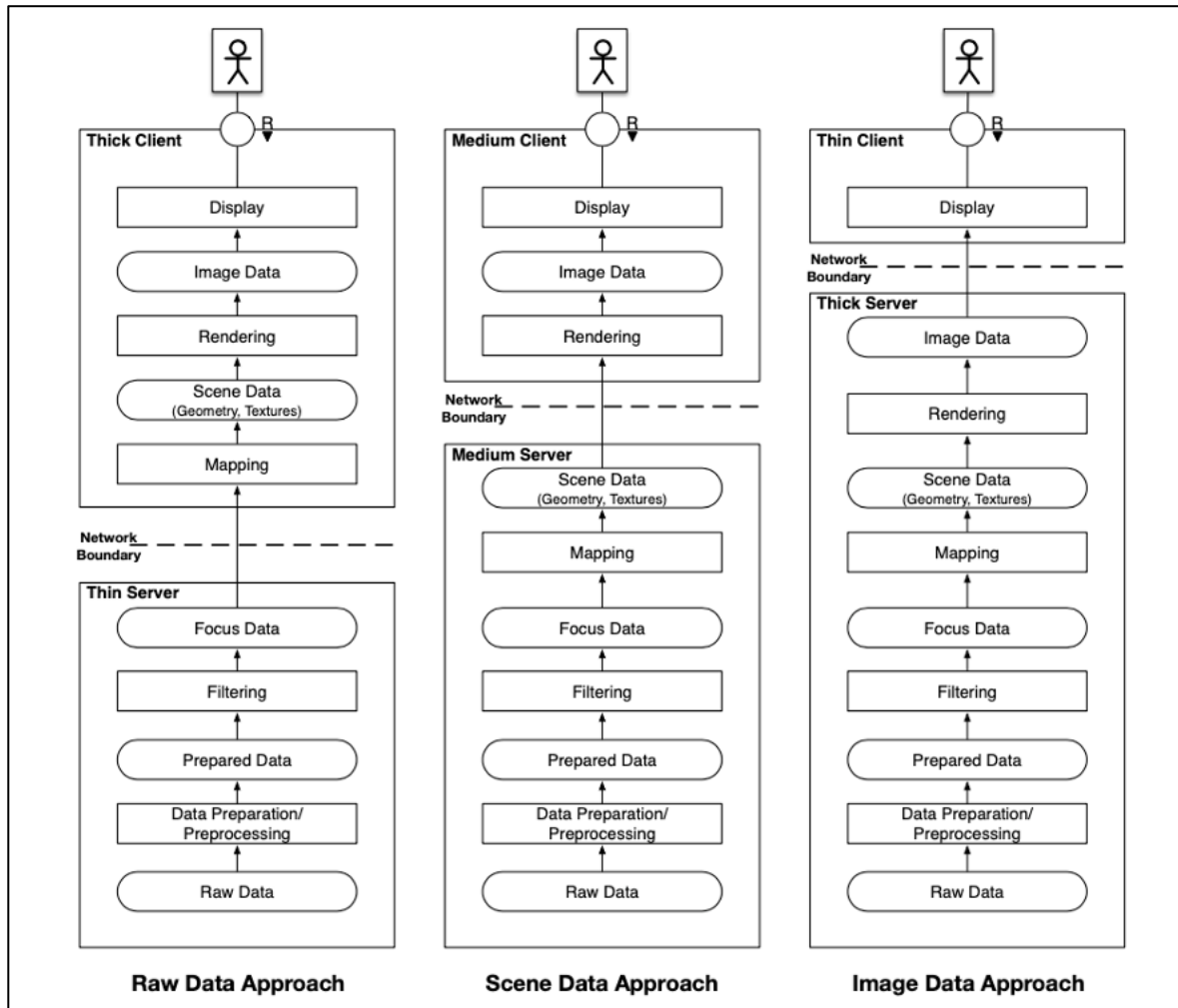


Figure 7. Distributed Visualization Pipeline (Doyle and Cuthbert, 1998).

In the Thin Client - thick server approach, since the rendering process is done in the remote server, the client's hardware capabilities do not affect the render performance and render quality. The 3D scene is rendered on the server, and the final view is sent to the client as an image. This approach's major drawback is that for each user interaction such as zoom in or rotation that modifies the view, a new view must be generated on the server and must be sent to the client; hence, this approach increases network overhead between server and client significantly. To decrease the negative effects of the increased network overhead on the performance, this approach requires high-end hardware on the server-side. Despite the powerful hardware support on the server, latency may occur in real-time applications that depend on heavy user interaction. Examples for server-side rendering are Google Stadia, Google's cloud-based gaming platform, GeForce Now, Nvidia's cloud-based gaming platform and Amazon Luna, Amazon's cloud-based gaming platform.

In the thick client- thin server approach, since the rendering process is done on the client's machine using the client's hardware, the client's hardware capabilities significantly affect the rendering performance and render quality. However, in this approach, the data is fetched only once from the server and cached in the browser's memory. As a result of the user interaction, if modification of the scene does not require fetching new contents from the server, the new view is produced from locally cached data. Hence this approach minimizes the network overhead in the rendering process.

Jankowski et al. (2013) and Evans et al. (2014) classify the visualization approaches into two categories as the declarative approach (retained mode) and the imperative approach (immediate mode) (Figure 8).




	2D	3D
<p><u>Declarative</u> Scene Graph Part of HTML Document DOM Integration CSS/Events</p>		<p>Declarative 3D for the Web Architecture Community Group</p> 
<p><u>Imperative</u> Procedural API Drawing Context Flexible</p>	<p><canvas></p>	

Figure 8. Classification of client-based rendering methods (Evans et al. 2014).

In the declarative approach, only 3D content is provided to the browser using formats such as X3D or glTF then the browser renders the 3D content in the background. In short, only "what to draw" is provided to the browser. 3D content can be rendered without imperative scripting. In the imperative approach, in addition to "what to render", "how to draw" is also provided to the browser via low-level graphic APIs such as WebGL. In this approach, every step of the rendering pipeline is defined explicitly, hence, gives developers more control over the rendering pipeline in exchange for requiring a deeper understanding of 3D graphics. Since every step is defined explicitly, low-level imperative scripting makes

the rendering process more performant; however, it slows down the process of developing 3D applications because its low-level nature requires much more time for coding and debugging.

There are some limitations in the classification of Jankowski et al (2013) and Evans et al (2014). Imperative and declarative approaches are not equivalent to each other and should not be done with a binary classification as declarative and imperative. There are many implementations in between. When we look at Figure 7, X3DOM has been placed as the declarative equivalent of WebGL however X3DOM is a JavaScript library which implements WebGL under the hood. There are many JavaScript libraries for 3D rendering with WebGL which eases development of 3D applications by freeing developers from writing low-level WebGL code and these libraries which makes imperative vs declarative classification is very difficult are not included in the table. Additionally, the scene graph is placed under the declarative headlines and creates the perception that there is no scene graph implementation with WebGL. However, there are many WebGL libraries which implement scene graphs.

1.6.1.3. 3D Formats on the Web

There are different data models that determine in which manner data can be stored, organized, and manipulated. One of the most common data model approaches is the hierarchical data model. In the hierarchical model data is organized into a tree-like structure, where each record has a single parent or root. Each record is stored with its geometry and attributes together. Common data is repetitively stored for each record. In hierarchical data models, since searching for data requires traversing the entire model from top to bottom until the required data is found, accessing the data at the bottom of the hierarchy is slow while accessing the data at the top of the hierarchy is fast.

One of the most used 3D formats which follows the hierarchical data model approach is CityGML. At the time of writing thesis CityGML is the only OGC standard for storage and exchange of the 3DCMS. CityGML is designed as an open data model and XML-based format for the storage and exchange of 3DCMs. CityGML defines the classes and relations for the most relevant topographic objects in cities and regional models with respect to their geometrical, topological, semantical, and appearance properties. "City" is broadly defined to comprise not just built structures but also elevation, vegetation, water bodies, "city

furniture" and more. Included are generalization hierarchies between thematic classes, aggregations, relations between objects, and spatial properties. CityGML is applicable for large areas and small regions and can represent the terrain and 3D objects in different levels of detail simultaneously (OGC, CityGML 2.0).

There are two versions of CityGML, CityGML 1.0 and CityGML 2.0, which have been accepted as OGC Standard. At the time of writing this thesis, there is ongoing work about the next major version, "CityGML 3.0," and it will be released as a new OGC standard. It will bring several improvements and new functionalities. One of the most important functionalities is the new "Dynamizer" module for representing time dynamic data in 3DCMs that is important in the context of smart cities and digital twins. The other important one is the new "Versioning" module that stores and represents changes in the 3DCMs. Detailed information about the other new functionalities and improvements can be found at (Chaturvedi et al., 2017), (Chaturvedi and Kolbe 2017), (Kutzner and Kolbe, 2018) and (Kutzner et al., 2020).

Even though the brand new CityGML 3.0 is considered, CityGML is not an appropriate format for displaying 3DCMs directly in the browser. CityGML has the disadvantages of the hierarchical data model which it follows. It is deeply nested and contains data duplication (Figure 9).

```

<gml:surfaceMember>
  <gml:Polygon>
    <gml:exterior>
      <gml:LinearRing>
        <gml:posList srsDimension="3">-118.42717988940433 34.244934943938965 0.0 -118.42711851176027 34.2448773169205
        </gml:LinearRing>
      </gml:exterior>
    </gml:Polygon>
  </gml:surfaceMember>
  <gml:surfaceMember>
    <gml:Polygon>
      <gml:exterior>
        <gml:LinearRing>
          <gml:posList srsDimension="3">-118.42717988940433 34.244934943938965 0.0 -118.42717988940433 34.24493494393896
          </gml:LinearRing>
        </gml:exterior>
      </gml:Polygon>
    </gml:surfaceMember>
    <gml:surfaceMember>
      <gml:Polygon>
        <gml:exterior>
          <gml:LinearRing>
            <gml:posList srsDimension="3">-118.42711851176027 34.2448773169205 0.0 -118.42711851176027 34.2448773169205 4.
            </gml:LinearRing>
          </gml:exterior>
        </gml:Polygon>
      </gml:surfaceMember>
    </gml:surfaceMember>
  </gml:surfaceMember>

```

Figure 9. A small part of a CityGML file.

Deeply nested structure requires nested loops which cause intensive CPU cycles to parse and extract information from a CityGML file and data duplication increases memory usage unnecessarily.

The other well-known 3D format which follows the hierarchical data model approach is X3D. It was developed and maintained by the Web3D Consortium (URL-14) and became an ISO standard. A small part of a X3D file can be seen in Figure 10.

```

<Scene>
  <Shape>
    <Appearance>
      <Material />
    </Appearance>
    <IndexedFaceSet
      solid="false"
      convex="false"
      coordIndex="0 1 5 4 -1 1 2 6 5 -1 2 3 7 6 -1 3 0 4 7 -1 0 1 2 3 -1 4 5 6 7 -1"
      creaseAngle="0.5">
      <Coordinate DEF="ObjCoordinates"
        point="3703352.5 3089915.75 4159522.75 3703353.25 3089908.25 4159527.5 3703345.75 3089910.5 4159532.5 3703345 3089918 4159527.5 370335
      </Coordinate>
    </IndexedFaceSet>
  </Shape>
  <Shape>
    <Appearance>
      <Material />
    </Appearance>
    <IndexedFaceSet
      solid="false"
      convex="false"
      coordIndex="0 1 5 4 -1 1 2 6 5 -1 2 3 7 6 -1 3 0 4 7 -1 0 1 2 3 -1 4 5 6 7 -1"
      creaseAngle="0.5">
      <Coordinate DEF="ObjCoordinates"
        point="3703352.5 3089915.75 4159522.75 3703353.25 3089908.25 4159527.5 3703345.75 3089910.5 4159532.5 3703345 3089918 4159527.5 370335
      </Coordinate>
    </IndexedFaceSet>
  </Shape>

```

Figure 10. A small part of a X3D file.

X3D is not as deeply nested as CityGML, however it has the same disadvantages such as data duplication. Duplicated vertices can be seen in Figure 10 between two objects which are two “Shapes” in X3D data model.

Another common data model approach for storage and organization of the data is the network data model. In the network data model hierarchy between objects and the data are separated. The data which belongs to all of the objects is stored globally and relevant data for an object is accessed using pointers. Hence, repetitive data is minimized in the network data model. Also, network data model is easier to update than hierarchical data model because a change in the geometry is automatically propagated to objects hence, objects now point to updated geometry now.

CityJSON is the network data model equivalent of the well-known CityGML. It is an OGC 3DCM standard candidate at the time of writing this thesis (Ledoux et al, 2019). In the CityJSON Object hierarchy and geometric data is separated. Vertices are stored as a global array and relevant data for each record is accessed using index numbers of the record (Figure 11). Values of the boundaries point to vertex coordinates of a surface for an object in Figure 11.

```

"CityObjects": {
  "B-201391182746-80FE6BE00A9C": {
    "type": "Building",
    "geometry": [
      {
        "type": "MultiSurface",
        "boundaries": [[[0,1,2]], [[2,3,4]], [[0,2,5]], [[4,1,7]], [[8,9,10]], [[11,12,13]], [[9,14,15]]],
      }
    ]
  },
  "B-201391182746-80FE6BE00A9C": {
    "type": "Building",
    "geometry": [
      {
        "type": "MultiSurface",
        "boundaries": [[[16,17,18]], [[18,19,20]], [[16,18,21]], [[20,17,23]], [[8,9,10]], [[11,12,13]], [[9,14,15]]],
      }
    ]
  }
},
"vertices": [[300762.679,5041624.941,14.274],[300749.896,5041617.688,14.274],[300755.943,5041628.802,14.274],
[300769.697,5041653.169,14.274],[300762.009,5041641.543,14.274],[300760.033,5041650.831,14.274],
[[300779.945,5041646.854,14.274],[300779.553,5041646.725,14.274],[300779.753,5041646.981,14.274]],

```

Figure 11. A small part of a CityJSON file.

Another popular format which follows the network data model approach is glTF (GL Transmission Format). Cross-browser support of WebGL and widespread usage of it has created the need for a new 3D format that is compact, suitable to be used with WebGL and requires minimal processing for rendering. With these needs and goals, glTF, a new 3D file format, has been created by Khronos Group, also the creator of OpenGL and WebGL. glTF is the only 3D format which is created with WebGL in mind after the invention of the WebGL in this section.

In contrast to X3D, which embeds all scene graph structures as XML elements, glTF separates the scene graph and geometry. Geometry is represented as a binary block containing vertices, indices, normals as raw byte arrays. The binary format for the block has been designed according to WebGL specification. Hence, the binary file can be loaded

directly into the client's graphic card and can be rendered using WebGL without additional processing.

In glTF format, the scene graph structure and geometry are separated. Scene graph is described as JSON, and all of the geometry is stored as a single buffer in the binary file with bin extension. Buffer is stored as a little-endian blob. A subset of data in the buffer is represented via bufferView. A 44-byte buffer that represents a single triangle and its encoding in glTF is shown in Figure 12.

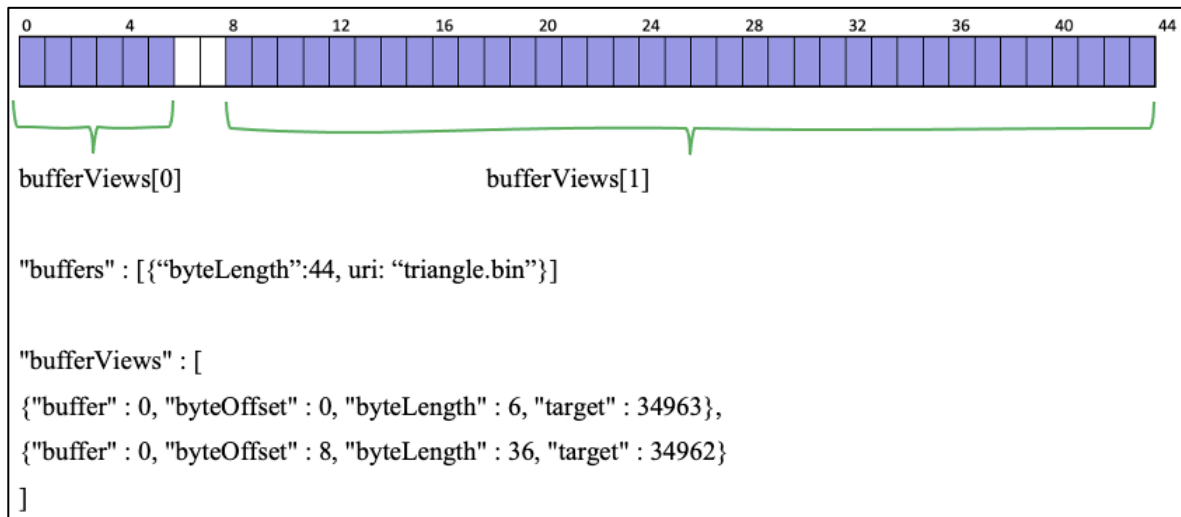


Figure 12. Binary data and its encoding in a glTF file.

Accessors in a glTF file have been used to retrieve data as typed arrays from within a bufferView. Using the information in the accessors and bufferViews, relevant geometric information for an object can be extracted from the blob. Figure 13 shows the complete glTF file for a triangle mesh.

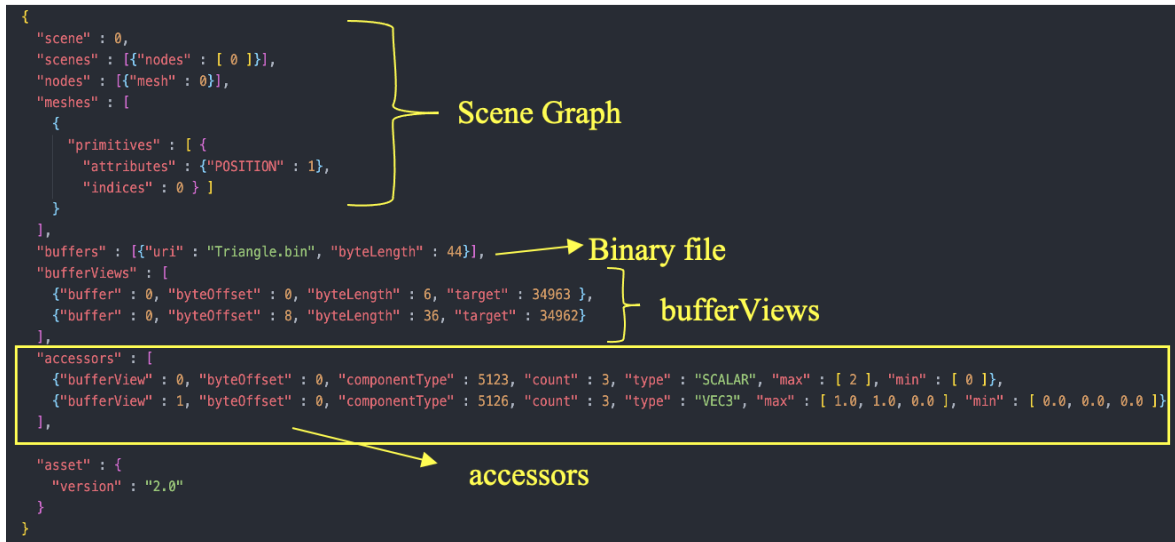


Figure 13. glTF file and its contents.

X3D standard was developed in 2003, CityGML in 2008, glTF in 2017 and CityJSON in 2019. 3D data formats which follow the hierarchical data model such as x3D and CityGML use XML for encoding and can be considered a bit out of data. In contrast, Formats which follow the network data model are respectively new and use JSON as encoding.

JSON is a much more preferred format by developers over XML. The main reason for this is that JSON is easier to parse than XML. It does not require complex marshalling and unmarshalling processes. Most programming languages have built-in data types that correspond to key-value pairs in JSON, such as "Dictionaries" in Python, "Maps" in Java and Golang. Hence, there is no need for external libraries while working with JSON files. These native built-in data types ease the mapping of JSON content in these languages. Also, since its native built-in data type, any JSON file can be parsed easily with JavaScript, which is the most used browser-based script language. Additionally, JSON files are smaller than their XML counterparts.

1.6.1.4. Standardization Efforts Related to Delivery of the 3D Geospatial Data Over the Web.

A number of web service standards have been developed by OGC to deal with visualization of spatial data on the web in an interoperable manner. These services can be categorized into data services and portrayal services. Data services provide access to the

spatial data such as WFS and WCS, and portrayal services such as WMS deliver images of the spatial data not the data itself (Altmer and Kolbe, 2003).

For 3D geospatial data OGC has published Web View Service (WVS) as “Discussion Paper” (Hagedorn, B. 2010) and Web 3D Service Implementation Standard (W3DS) (Schilling, A. and Kolbe T. H., 2010) as a Draft Specification. W3DS is designed as a “WFS like” data service that accesses the 3D geodata itself utilizing “GetScene” operation. For a decomposed dataset, W3DS specification offers “GetTile” operation. Using GetTile request parameters TileLevel, TileRow and TileCow the relevant tile is fetched from the server. For visualization of a specific area, mapping between camera parameters and tile request parameters are missing in W3DS. Tiling related problems such as object integrity during decomposition, definition of the data structure, tile size and update of the tileset are not addressed in W3DS.

Unlike W3DS, WVS is designed as a “WMS like” portrayal service that delivers a 3D rendered image of the data, not the data itself utilizing “GetView” operation. Tiling and tilesets are neglected in WVS. Management of a tileset is proposed as a “future work”.

In 2012, OGC published 3D Portrayal Interoperability Experiment (3DPIE) Final Report (3DPIE final report, OGC, 2012). In this 3DPIE activity, different use case scenarios for service-based streaming of 3D geospatial data were implemented and tested using OGC drafts for the candidate standards W3DS and WVS. Three different tests have been carried out. In the first test, the whole CityGML data was fetched from the server and visualized on a thick client. The purpose was to test the display of the CityGML directly on the browser. Fetching and parsing the entire CityGML file and applying filtering, mapping and rendering steps caused significant latency and performance issues on the client. Therefore, after this test, pre-process and tiling of the CityGML file has been suggested as a result of the test. In the second test, the entire CityGML data pre-processed in the server using JAXB (Java Xml Binding) based parser and tiled into regular tiles. Hence, only the visible tiles have been fetched from the server, progressive visualization has been accomplished based on the camera parameters provided by the client. In the third test CityGML format has been converted to JSON and visualized on the client using Three.js. For fetching the tiled 3DCM, GetTile operation of the W3DS has been used in the tests in 3D PIE. Although CityGML has been decomposed into regular tiles, how object integrity is maintained is not discussed. Other tiling related problems such as choice of the data structure, tile size definition, and how the tileset can be updated are not addressed as well.

After the interoperability tests and engineering reports essential parts of the W3DS and WVS have been combined into one common service as “3D portrayal, hence 3D Portrayal Service Standard (3DPS) published in 2017 (3DPS, 2017). 3DPS defines a standard service interface for web-based 3D geodata portrayal. Two major operations are described in the standard as “GetScene” operation and “GetView” operation. GetScene operation allows a client to request and retrieve a scene graph in a standard data format from a 3DPS service. The fetched data from the server is rendered on the client’s machine. Supported formats are X3D, VRML, Esri I3S and glTF. In contrast to GetScene operation, GetView operation allows a client to retrieve rendered images of a 3D scene based on the parameters defined in the request. The rendering process is done on the server-side, and the rendered image is fetched from the server. Supported image formats are PNG and JPEG. Although tiling is suggested and pointed as future work in previous works, GetTile operation in the W3DS has been deprecated in the 3DPS. The reason was that according to OGC there is no one-size-fits-all solution for tiling 3D geodata (3DPS, 2017). Hence, tiling and tiling related problems have been completely excluded in 3DPS. 3DPS focuses on the way of communication between server and client for streaming of the 3D geospatial data on the web by defining service interfaces, operations, request, and response parameters rather than tiling and handling of the tilesets.

Then for the tiling of the 3D geospatial data and streaming of the tileset, OGC has been published two standards, 3D Tiles and Indexed 3D Scene Specification (I3S) Indexed 3D Scene Specification (URL-15) developed by ESRI and 3D Tiles specification developed by Cesium (URL-16). Both specifications are open and optimized for streaming and rendering 3D geospatial data over the web. The foundation of these specifications is spatial hierarchical data structures. Although I3S is an open standard, it is generally used by ESRI products. There is no open-source software or component that generates a tileset according to I3S. It is only used in a few applications (Koukofikis et al., 2018). On the other hand, 3D Tiles has been started to be used by many commercial and open-source software components. These components have been described in section 2.3 “Related Work”. For its open nature, 3D Tiles has been selected and implemented while developing web components in this thesis. .

3D Tiles is based on a spatial data structure that enables the Hierarchical Level of Detail (HLOD); hence, only visible tiles are streamed for a given 3D view (3D Tiles Specification, 2019).

The specification supports many spatial data structures such as k-d trees, quadtrees, octrees, r-trees, and many others as long as the data structure is a hierarchical tree. To decide which data structure will be used for tiling is the burden of the developer. In 3D Tiles, a tileset is a set of tiles organized in a hierarchical spatial data structure, and a tile is a node in this data structure. The hierarchy and metadata about tiles are described in a file called “tileset.json”.

After 3D Tiles, OGC continues to do a lot of work for standardizing service-based delivery of 3D geodata on the web through a series of engineering reports. OGC published a draft specification for 3D geodata API that organizes access to a variety of 3D datasets according to a nested hierarchy of 3D geo data. (OGC API-Tiles-3D Engineering Report, 2020). In this 3D GeoVolumes also called 3D Container work, OGC tries to standardize API access to tiled 3D geodata resources by standardizing URL definitions for HTTP Get methods of REST APIs.

If the standards mentioned throughout this section are evaluated, it will be seen that 3D Tiles and I3S standardize organization of the 3DCMs on the server by utilizing hierarchical data structures and different delivery formats. 3DPS standardizes the way of communication between the server and the client in the context of 3D geospatial data. Finally, 3D GeoVolumes try to standardize the way of the access to 3D geo data on the web by defining standard URL paths for HTTP methods.

As can be seen, standardization efforts continue and although there are brand new 3D standards, interoperability is not a solved problem for web-based management of the large-scale 3D geospatial data.

2. CHAPTER 2 TILING, STREAMING AND DISPLAY OF 3D GEOSPATIAL DATA ON THE WEB

2.1. Introduction

For streaming and visualization of 3D geospatial data, the data is decomposed into multiple tiles. As needed, relevant tiles are sent over the network and processed by client components. Streaming and displaying only the most relevant data decreases the amount of the data transmitted and improves performance significantly. Nevertheless, tiling 3D geospatial data are respectively new and works on the topic are rare. While tiling improves performance, it creates many other problems revolving around generation and management of the tileset.

While generating 3D tileset, decomposition of the 3D geospatial data is made according to a data structure. Some data structures are better for analyses and spatial queries while others are preferable for data visualization. In this context, the data structure should be selected according to the purpose of the operations such as spatial query or visualization. After the decomposition, all of the operations are performed through the selected data structure by traversing the data structure, finding relevant tiles, and performing operations on the contents of those tiles. Hence, performance of the operations is really dependent on the selection of the data structure. During the generation of the tileset, some city objects may intersect with multiple tiles at the tile borders and unity of city objects may be compromised. This makes data management difficult across the tiles. After the generation of the tileset, tileset may need to be updated. Adding new contents to the tiles may exceed tile size threshold and boundaries of the new contents may intersect multiple tiles. For performant update operations, these operations must be handled by updating only affected tiles without re-generating the whole tileset. Decomposition of the 3D geospatial data into smaller tiles is done based on a tile size threshold. Reducing the tile size increases the number of tiles. Since the increase in the number of tiles increases the number of HTTP requests for fetching tiles from the server and overall network usage thus decreases performance. On the other hand, increasing tile size increases the amount of the data to be streamed and degrades the performance. Hence there is a trade-off while determining the tile size.

All of the problems mentioned so far are tackled and solved in this chapter by designing and developing a web framework using current 3D geospatial standards. The components of the framework developed as a SaaS (Software as a Service) tool. The developed tool consists of two RESTful web services. One service is responsible for the generation of 3D tileset from a 3D geospatial data resource, and the other service is responsible for display of the generated 3D tileset. Both of the services are also support 2.5D vector terrain data as well for tiling and visualizing along with 3DCMs. While developing services, the 3D Tiles standard is also implemented in order to guarantee interoperability of the developed framework with other software components.

2.2. Related Work

In this section, 13 academic works and 6 software components about the tiling of 3D geospatial data have been investigated in the terms of tiling scheme, tile size, object integrity, update of the tileset and implementation of the 3D standards. These works and their limitations are explained in more detail below. Also, summary of the investigated works can be found in Table 2 and summary of the investigated components can be found in Table 3.

Gesquiere and Manin (2012) developed a client-server architecture for visualizing CityGML data on the web. They parsed the CityGML file using C++ library libCityGML and converted CityGML data into several regular tiles (Figure 14). Extracted geometry and semantic information of the city objects stored in the JSON files on the server. Tiles have been fetched and visualized on the client's browser based on camera parameters using WebGL. This work uses regular tiling without a hierarchy. 3DCMs have been converted into rectangular shaped tiles. The major drawback of these methods is that the distribution of buildings in 3DCMs is heterogeneous; hence, in some areas, there are much more city objects than others and the regular tiling approach is lack hierarchical subdivision, hence, does not consider density of objects in the city and creates heterogeneous size of tiles. The lack of the hierarchy obscures taking the advantage of the scene graphs for optimizing filtering and mapping steps of the visualization pipeline. Since in the regular tiling each tile is in the same level, visibility checks must be done for each city object in each tile. Reducing the number of the visibility checks for frustum culling, is not possible in this regular tileset. The other drawback of regular tiling is that objects may intersect with multiple tiles on the

tile borders and since the unity of a building can be compromised, this makes object management difficult between tiles. The tiling process is performed as offline mode on a desktop. After the generation of the tileset, tiles are visualized in online mode. Additionally, any of the web-based 3D standards are not implemented and that makes interoperability of the work rather limited.

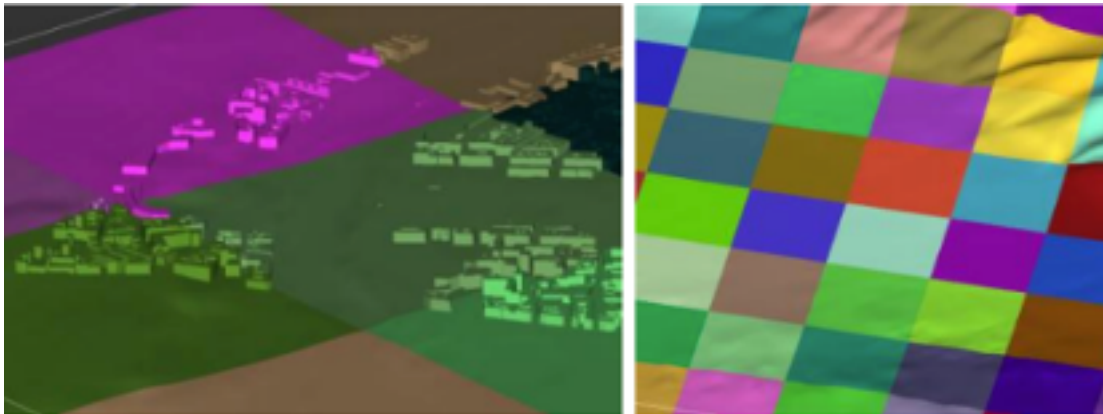


Figure 14. Regular tiling of the CityGML dataset (Gesquiere and Manin, 2012)

Prandi et al. (2013) developed a smart city platform and deployed various smart city services which require tiling. They use regular tiling and the tiling process is not web-based as Gesquiere and Manin (2012). Hence their work has the same limitations as Gesquiere and Manin (2012). Visualization pipeline is not optimized well because of the lack of the hierarchy between tiles and there is no solution about determination of the tile size or preserving the object integrity.

Chaturvedi et al (2015) developed a web-based 3D client for 3DCityDB in order to visualize CityGML data in the browser. In this work, a regular tiling method has been implemented in order to visualize large scale CityGML data progressively (Figure 15). Visualization is done using Cesium.js. 3DCityDB is not a web component and needs to be installed on a desktop. Hence, Chaturvedi et al (2015) has the same limitations with previous works. Buildings intersected with multiple tiles can be seen in Figure 7 and how the integrity of these objects can be preserved is not addressed. Same is true for tile size, updating the tileset. Additionally none of the 3D standards has been implemented.

In his MSc. Thesis Willenborg (2015) converted CityGML geometry to voxel representation and made simulation of explosions in urban space. Simulation results were stored in the PostgreSQL database using 3DCityDB and results visualized in the browser

using Web Map Client of the 3DCityDB. Hence, this work has the exact same limitations as Chaturvedi et al (2015).

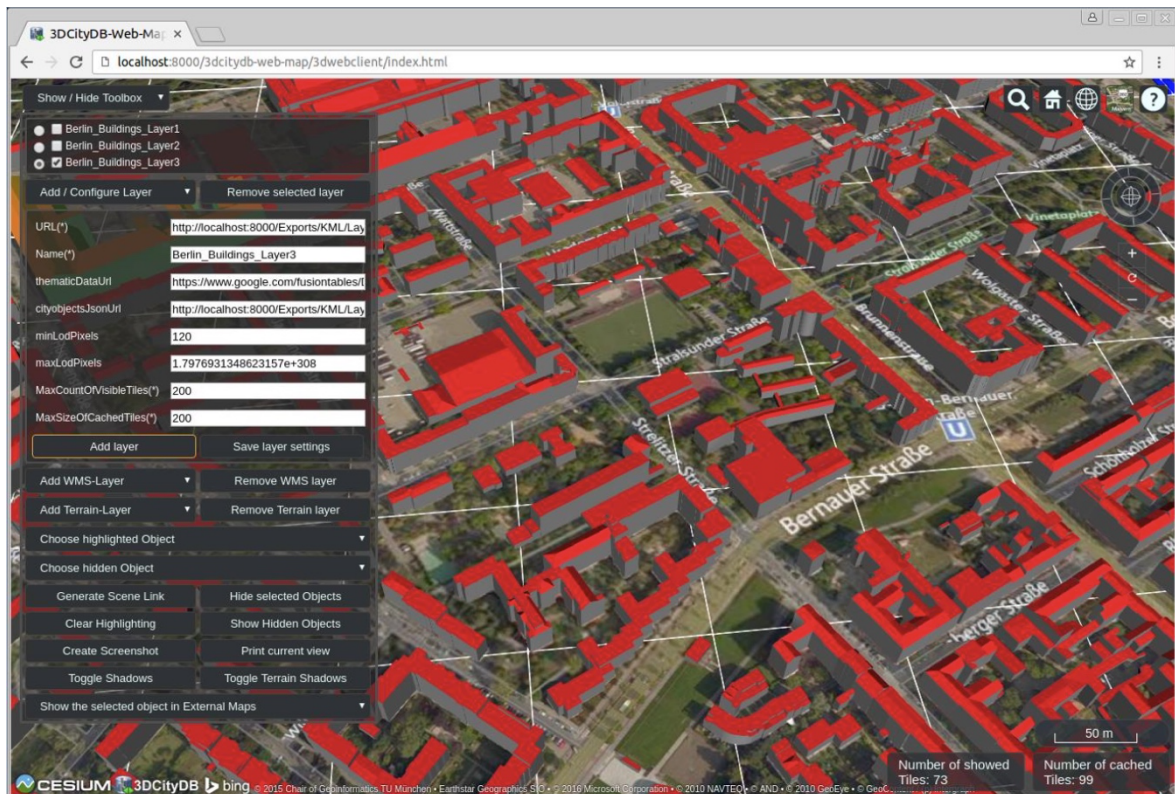


Figure 15. Regular tiles can be seen in the web map client of 3DCityDB

Kilsedar et al (2019) visualized 3DCM on the web using 3DCityDB and the web-map client of 3DCityDB. They developed a component called shp2city in order to convert 3D data in the Esri Shapefile format to CityGML. Then, they processed CityGML data using 3DCityDB and extracted geometry visualized using the web-map client of 3DCityDB.

Gaillard et al. (2015) developed a framework for visualizing 3DCMs on the web. The CityGML file has been converted into JSON and the whole dataset is also decomposed into several fixed sized regular tiles. Then progressive visualization is accomplished using Three.js library. They developed a rendering strategy based on the regular data structure (Figure 8). Based on the camera parameters, tiles have been rendered in different LODs. The tile that consists of point of view (POV) rendered with high detailed buildings and DTM (red tile), the neighbour tiles are rendered with low detailed buildings and DTM (red tiles with green buildings) and neighbour of the neighbour tiles are rendered without buildings and only includes DTM. This work has the limitations of the regular tiling mentioned for

previous works above. Buildings intersected with multiple tiles can be seen in Figure 16. Additionally, the proposed rendering strategy works for only with regular tiling. It is difficult to implement this rendering strategy with hierarchical data structures. Because finding adjacent tiles is more difficult and there may be too many adjacent tiles in hierarchical data structures.

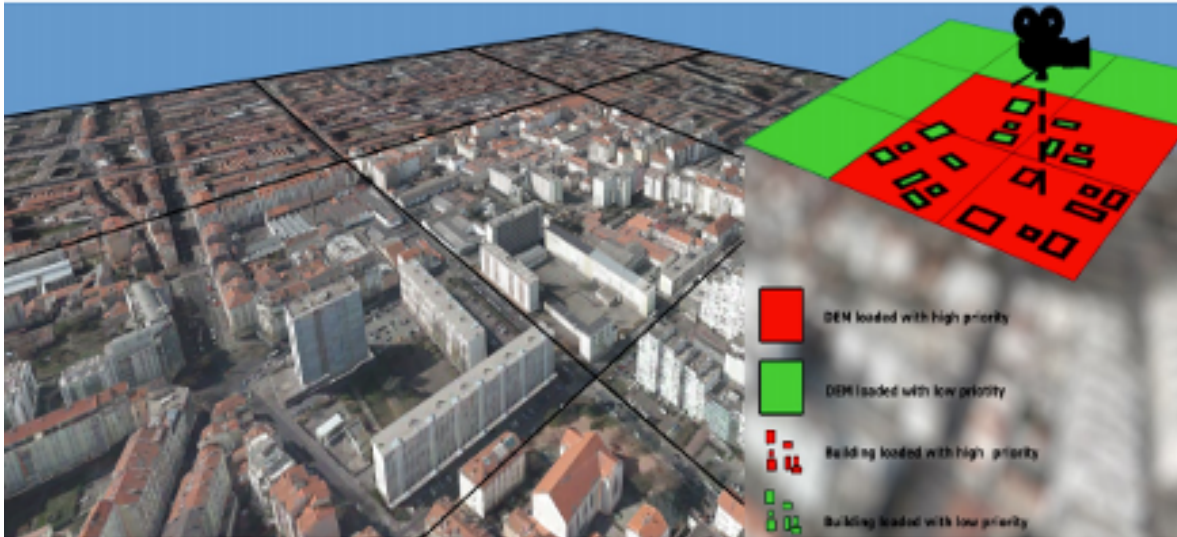


Figure 16. Rendering Strategy for Progressive Visualization (Gaillard et al., 2015)

Krämer and Gutbell (2015) developed 3D geospatial applications and tested WebGL frameworks such as Three.js, Cesium.js and X3DOM. They converted CityGML data into X3D and tiled CityGML into several rectangular tiles. They developed a simple streaming algorithm and loaded tiles into browser memory on demand based on camera parameters (Figure 17). This work also suffers limitations of the regular tiling. The proposed streaming algorithm works only for regular tiling. Hierarchical data structures are difficult to implement with this streaming algorithm. Additionally, any of the 3D standards is not implemented in this work.

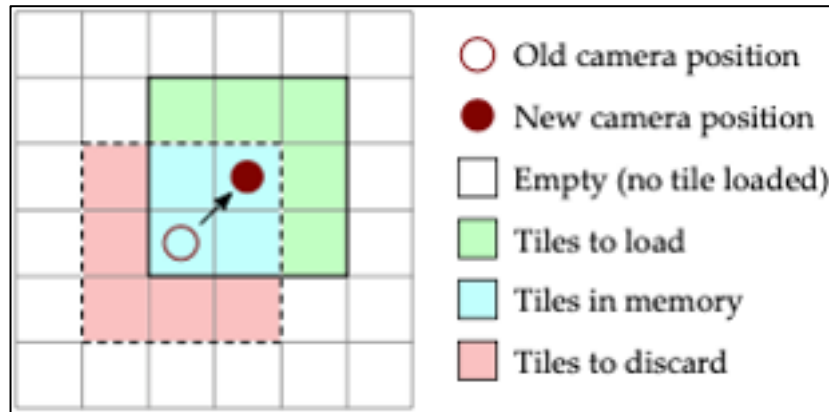


Figure 17. Streaming algorithm loads additional tiles and removes that are not visible based on camera position (Krämer and Gutbell, 2015).

With the arrival of the 3D standards, researchers also tried to solve tiling and displaying large scale 3D geospatial data by implementing 3D standards in order to achieve interoperability.

Gutbell et al (2016) developed a framework for server-based rendering of 3DCMs using 3D Portrayal Service Standard. Unity game engine and Blender 3D modelling software in order to visualize rendered images on the client, both of the software are desktop applications which require additional software installation on desktop. Since this work uses server-based rendering, every interaction on the client requires re-rendering the scene and fetching the new rendered image from the server which increases overall network usage and causes latency on the client.

Klimke (2019) developed a framework for web-based provisioning and application of large-scale virtual 3DCMs. They used 3D Portrayal Service Standard for server-based rendering of the framework. Since these studies use server-based rendering, every interaction on the client requires re-rendering the scene and fetching the new rendered image from the server which increases overall network usage and causes latency on the client.

Koukofikis et al (2018) developed prototype implementations for interoperable visualization of 3DCMs using 3D Portrayal Service Standard. They tested and validated interoperability of the 3D Portrayal Service Standard using 3D Tiles and I3S streaming standards in urban-centric use cases using CityGML data. For tiling 3DCMs they used commercial software components from Cesium ion, ESRI and FME. They used different software components which did not communicate with each other, hence requiring a lot of

user interaction during the workflow. Additionally, software components are commercial and not web based except Cesium ion.

Gaillard et al (2020) presented a new method for visualization and personalization of 3DCMs which supports multi-scale resolution of 3DCMs using 3D Tiles standard. The 3DCM has been tiled in the pre-processing step according to 3D Tiles standard and the generated tileset has been stored in a relational PostgreSQL database. Based on the user defined rules on the client, the scene graph is generated on-the-fly on the server and fetched to the client. The major drawback of the proposed method was that even small updates on the 3DCM or ruleset requires the re-generation of the whole scene graph and re-generation of the tileset. Another drawback of this study was the tiling method. The tiling process and generation of the hierarchical tileset has been done using road-network. The buildings adjacent to the main roads are placed in the higher tiles in the hierarchical tree while buildings adjacent to the side roads are placed in the lower tiles. This tiling method does not represent the size of the data. Hence, this situation can lead to tiles in heterogeneous file sizes. A more precise tiling method which represents data size accurately based on the actual size of the data must be implemented in order to balance tile sizes.

Lu et al (2020) has visualized real-time large-scale weather data using 3D Tiles streaming standard. The point-based weather data has been tiled using octree which is a hierarchical data structure and visualized tiles using Cesium.js. This study only supports tiling and visualization of point clouds, not other types of the 3D geospatial data.

Jaillet et al (2020) has visualized time-dynamic data along with 3DCMs on the web by extending 3D Tiles standard. The 3DCM has been tiled using py3dtiles open-source software component. This component is a python library in order to convert 3DCMs in the CityGML format to 3D Tiles. The drawback is that py3dtiles is not a web-based component. A summary of the related work can be found in Table 2.

Table 2. Summary of the related Works

Related Work	Tiling Scheme	Rendering	Object Integrity	Implementation of Standards
Gesquiere and Manin (2012)	Regular	Client	No	No
Prandi et al. (2013)	Regular	Client	No	No
Chaturvedi et al (2015)	Regular	Client	No	No
Willenborg (2015)	Regular	Client	No	No
Gaillard et al. (2015)	Regular	Client	No	No
Krämer and Gutbell (2015)	Regular	Client	No	No
Gutbell et al (2016)	-	Server	Yes	WVS
Koukofikis et al (2018)	Hierarchical	Client	Yes	I3S, 3D Tiles, 3DPS
Kilsedar et al (2019)	Regular	Client	No	No
Klimke (2019)	Hierarchical	Server	Yes	3DPS
Gaillard et al (2020)	Hierarchical	Client	Yes	3D Tiles
Lu et al (2020)	Hierarchical	Client	Yes	3D Tiles
Jaillot et al (2020)	Hierarchical	Client	Yes	3D Tiles

When we look at the software components for tiling and visualizing 3DCMs on the web, there is no web-based solution as open-source software components. `obj23dtiles` is an open source Node.js module in order to convert 3d data to 3D Tiles. `obj23dtiles` (URL-17) is based on another open source Node.js module `obj2gltf` which has been developed by Cesium. The major drawback of the `obj23dtiles` is that it does not construct a hierarchy from data, hence does not implement a tiling method. It only converts the whole 3DCM in the `obj` format to `b3dm` format. Another open source Node.js component is `citygml-to-3dtiles` (URL-18) which has the same drawback as `obj23dtiles`. It only converts CityGML data to `b3dm` format without implementing a tiling method. The most mature open-source component is

py3dtiles which converts CityGML data to 3D Tiles by tiling data and constructing a hierarchical tileset from it. Py3dtiles is not a web-based solution and must be installed as a standalone software component. At the time of writing this thesis it supports only glTF 1.0 hence it cannot handle glTF 2.0 3D models. FME, VirtualCityPublisher and Cesium ion are commercial software components for tiling 3DCMs according to 3D Tiles. FME has a 3D Tiles writer that tiles 3D geospatial data for generation of the hierarchy, it uses object count as a parameter. Each city object has a different number of vertices and indices hence, each city object differs in size. Tiling, based on object count causes heterogenous tile sizes. A more precise tiling method which represents data size accurately such as vertex count and index count must be implemented in order to decompose data more efficiently. A summary of the capabilities of the software components can be found in Table 3.

Table 3. Summary of the software components.

Software Components	Type	Licence	Hierarchical Tiling	Support for Updating the tileset	Support for 3D Standards
FME	Desktop	Commercial	Yes	No	3D Tiles
Cesium ion	Web-Based	Commercial	Yes	No	3D Tiles
Virtual City Publisher	Web-Based	Commercial	Yes	No	3D Tiles
py3dtiles	Desktop	Open Source	Yes	No	3D Tiles
obj23dtiles	Desktop	Open Source	No	No	3D Tiles
citygmlto3dtiles	Desktop	Open Source	No	No	3D Tiles

When the current studies in the literature and existing software components are examined, some limitations are observed. First, most of the works in the literature uses regular tiling without constructing a hierarchy which makes it impossible to optimize filtering and mapping steps by imposing scene graphs. Additionally decomposing a whole dataset to prefixed tiles causes compromising object integrity at the tile borders. None of the

works has investigated how this problem can be solved. Another important limitation is that none of the works or components so far support update of the generated tileset without re-generation of the tileset. Most of the proposed works require user intervention at some point in the proposed work flows hence, most of the proposed solutions are not fully automated which limits the reusability of the proposed methods. Additionally, there is no fully automated web-based open-source solution for tiling and visualizing 3D geospatial data. These limitations have been attempted to be eliminated by proposing the new methods which are described in the following sections of this chapter.

2.3. Methodology

2.3.1. Decomposition of the 3D Geospatial Data

2.3.1.1. Determination of the Tile Size

Tile size determines the amount of the data to be transmitted from server to client for a tile and directly affects the render quality. A client should be able to render and display the contents of a tile without any lag and without degrading the rendering performance. Rendering performance is measured as frame per second (fps) which is the number of rendered images per second. In the game industry and computer graphics 60fps is accepted as “good” rendering performance and 60 fps value is selected as the minimum acceptable rendering performance while determining the tile size. Since rendering is done on the client in our application, rendering performance is highly dependent on the client’s hardware. In order to establish mapping between rendering performance and the tile size, data threshold tests are performed using a computer that has a low-end GPU. The reason for choosing a computer with a low-end GPU is to guarantee the 60fps even with such computers. Table 3 shows the specs of the test machine.

Table 4. System Specifications of the Test Machine

Intel i5 1.8Mhz Turbo Boost CPU

4GB DDR3 RAM

Intel HD Graphics 4000 GPU

The tests started with a single LOD1 block model and fps is monitored at runtime. Data density is increased by adding new detailed city objects. With addition of the new vertices and indices by adding new objects fps started to drop. When the data reached 180 kb fps started to drop to under 60 fps while rotating objects. Hence, 180 kb is determined as the tile size threshold. Since a float value is 4 bytes, the total size of an object is calculated based on the vertex and index counts by using the following formula 2.1. “SoV” stands for size of the vertices and “v” stands for vertex count.

$$SoV = v * 3 * 4 \quad (2.1)$$

Since an index value is represented as 2 bytes integer value, total size of the indices has been calculated using the following formula 2.2. “SoI” stands for size of the indices and “i” stands for index count.

$$SoI = i * 3 * 2 \quad (2.2)$$

Since vertex normals are stored for each vertex as float x, y, z values, vertex size multiplied by 2 in the formula 2.3 to consider size of the vertex normals. Thus total size of the data “T” is calculated as in the following formula (2.3)

$$T = 2 * SoV + SoI \quad (2.3)$$

2.3.1.2. R-Tree Decomposition

R-Tree is based on recursive decomposition of data into two branches at each level with respect to a “tile size” threshold. That is, the decomposition of the nodes continues until each node complies with the threshold. Spatial coherence (3D Tiles, 2018) the enforcement that the content for child tiles is completely inside the parent's bounding volume must be incorporated into tiling procedure. One way of ensuring spatial coherence would be to use bounding rectangles and bounding volumes for 2D and 3D respectively.

R-Trees can be constructed by either bottom-up or top-down methods. The top-down method starts with decomposing the root into two nodes first. These nodes contain farthest objects to be tiled. A way of computing the farthest objects is explained in Guttman (1984).

By “bounding rectangle enlargement” for 2D data and “bounding volume enlargement” for 3D data, new objects to be included in either of the two tiles are computed. The tile with the minimum area or volume when the new object is enclosed is the right tile to include the new object.

A top-down R-Tree tiling has been implemented in this thesis. And the “tile size threshold” was set to “180kb”. Then nodes are populated with new objects via bounding volume enlargement. After all the remaining objects added to these two nodes, the threshold is checked; if it is exceeded then the node is subdivided into two nodes. This routine repeats itself until all the nodes comply with the threshold. Thus, 3D geospatial data is decomposed into a hierarchical data structure (Figure 18). Pseudo code for the construction of the R-Tree can be found in Figure 19.

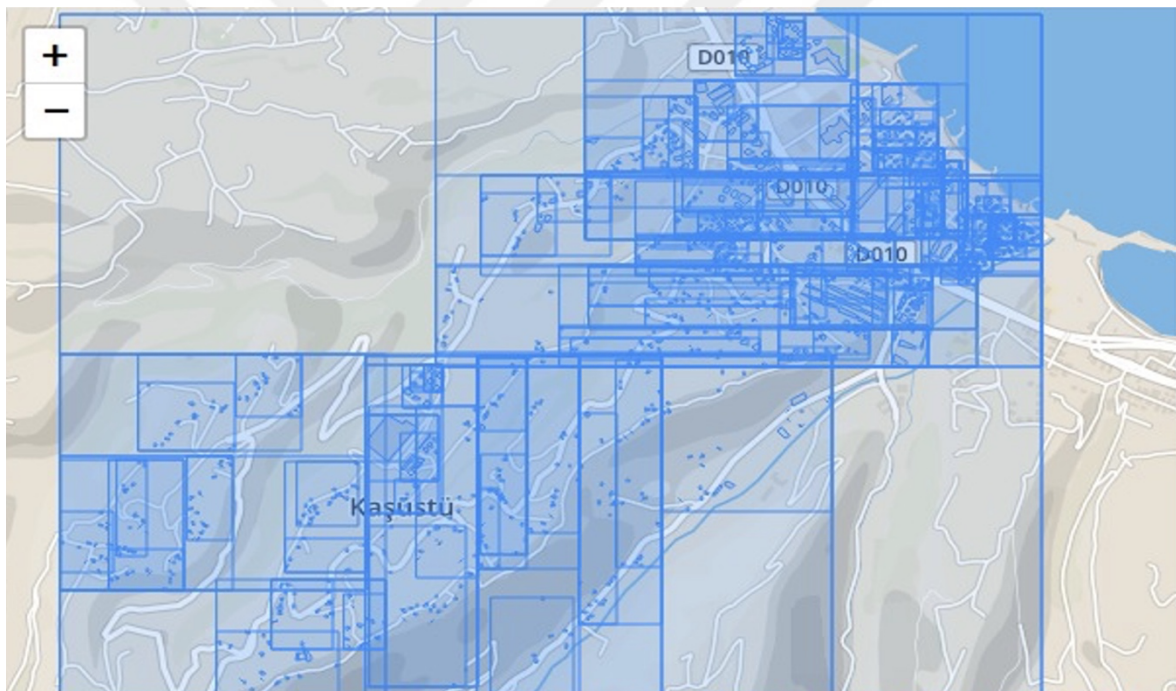


Figure 18. 3D geospatial data decomposed to R-Tree structure

Notations for Pseudo Code

$N \rightarrow$ Any Node

$F \rightarrow$ Feature in features

$S(N) \rightarrow$ size of a node

$B(N) \rightarrow$ Bounding Box of a node

$B(F) \rightarrow$ Bounding Box of a feature

$P(N) \rightarrow$ Parent of a node N

$Si(N) \rightarrow$ Sibling of a node N

MVE \rightarrow Minimum Volume Enlargement

```

1: While  $S(N) >$  threshold do
2:   for  $F$  in features do
3:     Add  $F$  into  $N$ 
4:   end for
5: if  $S(N) >$  threshold
6:   Subdivide  $N$  to  $N_1$  and  $N_2$ 
7:   DISTRIBUTE (features,  $N_1$ ,  $N_2$ )
8: function DISTRIBUTE (features,  $N_1$ ,  $N_2$ )
9: find the pair  $F_1$ ,  $F_2$  that creates maximum MVE ( $F_1, F_2$ )
10:  if  $MVE(N_1, F_1) < (N_2, F_1)$ 
11:     $N_1 \rightarrow N_1 \cup F_1$ 
12:     $B(N_1) \rightarrow B(N_1) \cup B(F_1)$ 
13:     $N_2 \rightarrow N_2 \cup F_2$ 
14:     $B(N_2) \rightarrow B(N_2) \cup B(F_2)$ 
15:  else
16:     $N_1 \rightarrow N_1 \cup F_2$ 
17:     $B(N_1) \rightarrow B(N_1) \cup B(F_2)$ 
18:     $N_2 \rightarrow N_2 \cup F_1$ 
19:     $B(N_2) \rightarrow B(N_2) \cup B(F_1)$ 
20:  end if
21:  Remove  $F_1$ ,  $F_2$  from features
22:  for  $F$  in features do
23:    if  $MVE(N_1, F) < MVE(N_2, F)$ 
24:       $N_1 \rightarrow N_1 \cup F$ 
25:       $B(N_1) \rightarrow B(N_1) \cup B(F)$ 
26:    else
27:       $N_2 \rightarrow N_2 \cup F$ 
28:       $B(N_2) \rightarrow B(N_2) \cup B(F)$ 
29:    end if
30:  end for
31: end function
32: function MVE ( $A$ ,  $B$ )
33:  $B(A) \cup B(B) / B(A)$ 
34: end function

```

Figure 19. Pseudo code for R-Tree construction

During the creation of the R-Tree, bounding boxes of the nodes are updated with the bounding box of the features that were added to these nodes. Since bounding boxes of the nodes can overlap each other, object integrity is preserved (Figure 20).



Figure 20. Tile borders overlap and features are always completely inside of a node.

2.3.1.3. Adaptive QuadTree Decomposition

QuadTree is based on recursive decomposition of data into four equal branches at each level with respect to a “tile size” threshold. That is, the decomposition of the nodes continues until each node complies with the threshold. QuadTree construction starts with decomposing the root node into four equal sized children. Then features are added to the child nodes and threshold is checked for each one of four child nodes. If any of the child nodes exceeds the threshold, it is decomposed to four equal child nodes. This process continues recursively until each node complies with the threshold.

For QuadTree construction “tile size threshold” is set as “180kb” as in the R-Tree decomposition. If the 3D model to be tiled exceeds the 180kb then it is decomposed into four nodes. Then nodes are populated with new objects. After all the remaining objects added to these four nodes, the threshold is checked; if it is exceeded then the node is subdivided into four nodes. Thus, 3D geospatial data is decomposed and converted into hierarchical data structure (Figure 21).

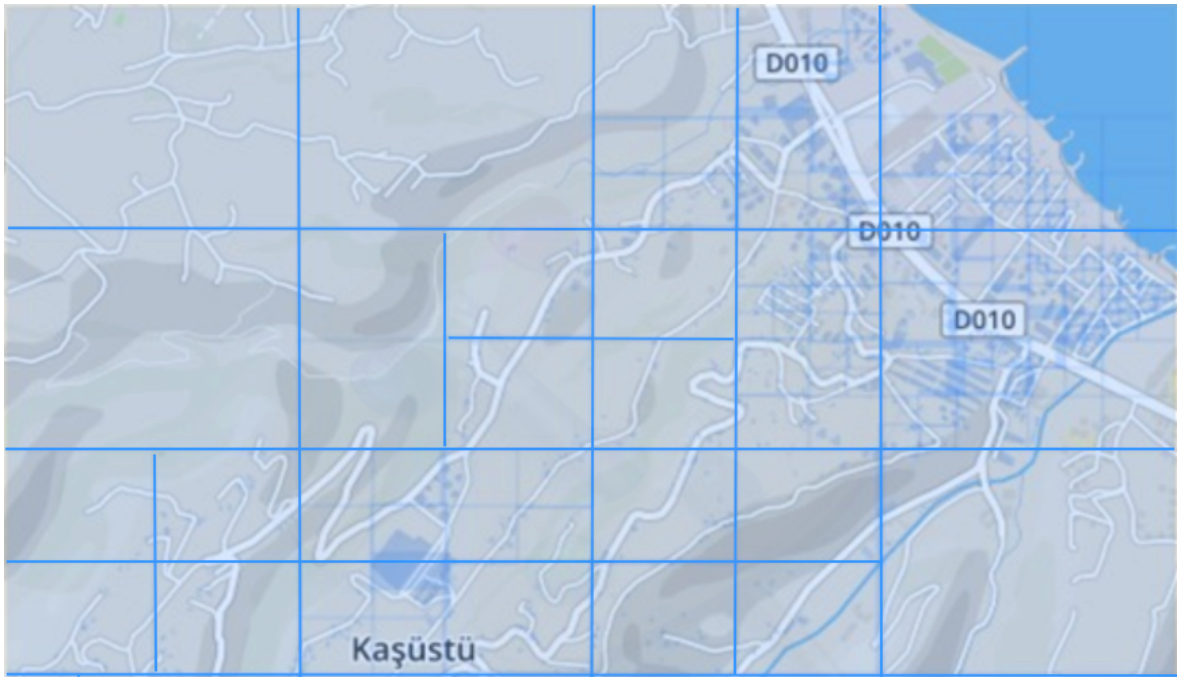


Figure 21. QuadTree decomposition of the 3D geospatial data.

In QuadTree, nodes are decomposed into four pre-fixed rectangular or square tiles. Tile borders are not calculated dynamically as in the R-Trees while adding new objects to the nodes. Hence, some objects may intersect multiple tiles at tile borders. In order to prevent object integrity, unlike as in the traditional quadtrees, intersection volume is calculated for each tile that object intersects. Then the object is added to the tile that has the intersection volume the most and the borders of that tile is updated adaptively to contain the newly added object (Figure 22). Construction of the Adaptive QuadTree is shown in Figure 23.

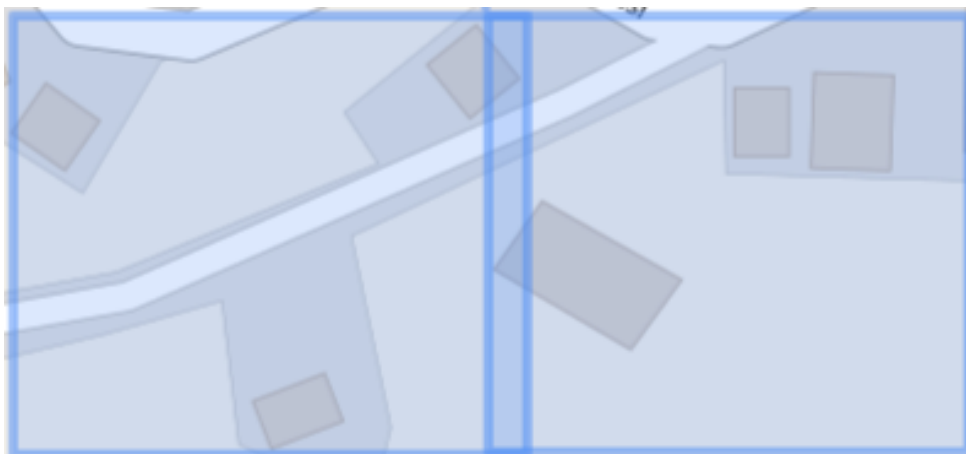


Figure 22. Updating tile borders in order to preserve object integrity.

```

1: While S(N) > threshold do
2:   for F in features do
3:     Add F into N
4:   end for
5:   if S(N) > threshold
6:     Subdivide N to N1,N2,N3 and N4
7:     for F in features do
8:       if F ∈ N1
9:         N1 → N1 ∪ F
10:        B(N1) → B(N1) ∪ B(F)
11:       if F ∈ N2
12:         N2 → N2 ∪ F
13:        B(N2) → B(N2) ∪ B(F)
14:       if F ∈ N3
15:         N3 → N3 ∪ F
16:        B(N3) → B(N3) ∪ B(F)
17:       else
18:         N4 → N4 ∪ F
19:        B(N4) → B(N4) ∪ B(F)
20:     end for
21: end for

```

Figure 23. Pseudo code for Adaptive QuadTree construction

2.3.2. Updating 3D Tileset

Updating tileset means adding new features to relevant nodes or removing existing features from nodes in the data structure. In order to update without re-construction of the tileset, two main operations are performed:

- Traverse data structure and find affected nodes
- Add features to these nodes or remove features from these nodes

While these operations are performed, first, the tile size threshold must be checked and if it exceeds relevant nodes must be decomposed to new child nodes. Then bounding boxes of the affected nodes must be re-calculated. For updating R-Tree and Adaptive Quadree,

although these general steps are the same, there are minor differences between these two data structures in the implementation.

2.3.2.1. Adaptive New Features in R-Tree

For adding a feature to a node, first, data structure is traversed from top to bottom and leaf nodes are found. Then it is decided which leaf node the feature is added based on minimum volume enlargement value. Minimum volume enlargement value is calculated between the feature to be added and the leaf nodes. Then, the feature is added to the node that requires minimum volume enlargement value. Figure 24 shows the pseudo code for adding new features in R-Tree.

```

1:for N in leaf nodes and F in features in do
2:  calculate MVE(N, F)
3:  if MVE (N, F) is the minimum
4:    N → N ∪ F
5:    B(N) → B(N) ∪ B(F)
6:    if S(N) > tile size
7:      Divide N into N1 and N2
8:      DISTRIBUTE (features, N1, N2)
9:    end if
10:  end if
11:end for

```

Figure 24. Pseudo code for adding new feature in R-Tree

2.3.2.2. Adding New Features in Adaptive QuadTree

For adding a feature to a node, first, data structure is traversed from top to bottom and the relevant node is found. Then the feature added to this node and tile size threshold is checked. If it is exceeded the relevant node is decomposed to four child nodes. Then features of the relevant node distributed to child nodes. Figure 25 shows pseudo code for adding the new feature in Adaptive QuadTree.

```

1: for F in features do
2:   for N in leaf nodes do
3:     if B(F) ∈ B(N)
4:       N → N ∪ F
5:       B(N) → B(N) ∪ B(F)
6:       if S(N) > tile size
7:         Divide N into N1,N2,N3,N4
8:         for F in features do
8:           if F ∈ N1
9:             N1 → N1 ∪ F
10:            B(N1) → B(N1) ∪ B(F)
11:          if F ∈ N2
12:            N2 → N2 ∪ F
13:            B(N2) → B(N2) ∪ B(F)
14:          if F ∈ N3
15:            N3 → N3 ∪ F
16:            B(N3) → B(N3) ∪ B(F)
17:          else
18:            N4 → N4 ∪ F
19:            B(N4) → B(N4) ∪ B(F)
20:          end if
21:        end for
22:      end if
23:    end for
24: end for

```

Figure 25. Pseudo code for adding a new feature in Adaptive QuadTree

2.3.2.3. Removing Features in R-Tree and Adaptive QuadTree

Removing a feature is algorithmically the same for both R-Tree and Adaptive QuadTree. To remove a feature from a Node, data structure is traversed and the relevant feature is found in the node hierarchy. Then the feature is removed from that node. After the removing process, the tile size threshold is checked and if possible, the relevant node is merged with its siblings to parent. Figure 26 shows the pseudo code for removing a feature.

```

1: find N in nodes consist of F
2: N → N / F
3: B(N) → B(N) / B(F)
4: if S(N) + P(N) < tile size threshold
5:   P(N) → P(N) ∪ N
   Remove N
6: else break
7: end if

```

Figure 26. Pseudo code for removing a feature in R-Tree and Adaptive QuadTree

2.3.3. Implementation of the 3D Tiles Specification

In 3D Tiles, hierarchical information and metadata about tiles are encoded to a JSON file called “tileset.json”. Thus, the tileset.json file is consumed by the client implementation at the runtime and the scene graph is derived for the visualization pipeline of the application. To generate tileset.json file hierarchical information that is derived from decomposition is encoded into a JSON file (Figure 27).

```

"boundingVolume" : {
  "region" : [ 0.6947392307760392, 0.7146410206129109, 0.695489298613394, 0.7150729689986538, 0.0, 42.0 ]
},
"geometricError" : 129.18604091253025,
"refine" : "ADD",
"content" : {
  "uri" : "0.b3dm",
  "boundingVolume" : {
    "region" : [ 0.6947392307760392, 0.7146410206129109, 0.695489298613394, 0.7150729689986538, 0.0, 42.0 ]
  }
},
"children" : [ {
  "boundingVolume" : {
    "region" : [ 0.6947392307760392, 0.7146410206129109, 0.6951918552458325, 0.7149567392788528, 0.0, 42.0 ]
  },
  "geometricError" : 64.59302045626512,
  "refine" : "ADD",
  "content" : {
    "uri" : "1.b3dm",
    "boundingVolume" : {
      "region" : [ 0.6947392307760392, 0.7146410206129109, 0.6951918552458325, 0.7149567392788528, 0.0, 42.0 ]
    }
  },
  "children" : [ {
    "boundingVolume" : {
      "region" : [ 0.6947392307760392, 0.7146410206129109, 0.6950447087526505, 0.7149133446464471, 0.0, 42.0 ]
    },
    "geometricError" : 32.29651022813256,
    "refine" : "ADD",
    "content" : {
      "uri" : "3.b3dm",
      "boundingVolume" : {
        "region" : [ 0.6947392307760392, 0.7146410206129109, 0.6950447087526505, 0.7149133446464471, 0.0, 42.0 ]
      }
    }
  },

```

Figure 27. A part of a tileset.json file that describes 3 levels hierarchy

According to 3D Tiles specification, tile definition includes a “Bounding Volume”, a “Refinement”, a “Geometric Error” and a “Content” in 3D Tiles. Bounding Volume is a 3D axis aligned minimum bounding box that encloses the tile. Refinement is the type of the refinement method that determines the refinement process. Geometric Error is a metric in meters that is used by the client engine to decide the refinement process. Content is a little-endian binary blob that contains the scene data which is a subset of the scene graph and encoded into a file with extension ”b3dm”. Content, also contains an “uri” that points the

path of the scene graph data file and a bounding volume. Figure 28 shows the UML class diagram for a tileset in 3D Tiles.

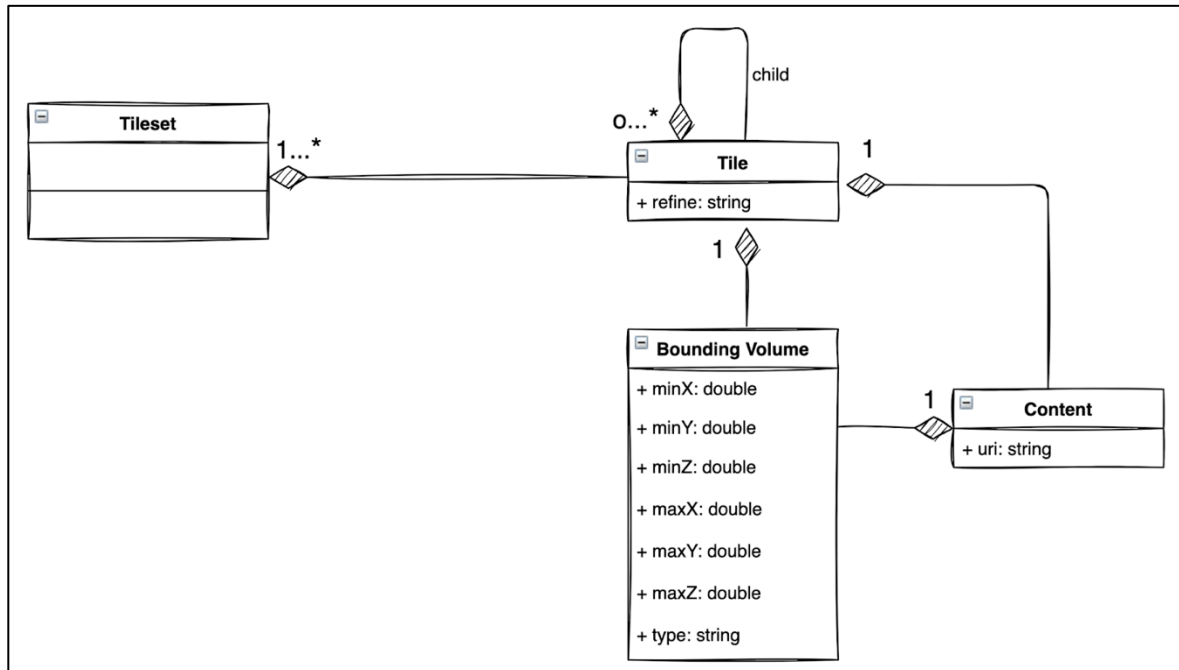


Figure 28. UML Class Diagram for A Tileset in 3D Tiles

2.3.3.1. Generation of the Tile Content

Content of a tile refers to both geometric and attribute data of the features of the tile. In 3D Tiles, contents of a tile are stored in a format called B3DM (Batched 3D Model). In b3dm format, geometry of the features are stored as glTF, non-spatial attributes are stored in “Feature Table” and “Batch Table”. glTF, feature table and batch table together form the b3dm file thus, both spatial and non-spatial properties of the features are represented in the b3dm file.

In the feature table, semantics which are “BATCH LENGTH” and “RTC_CENTER” are stored. Batch length is the number of features in the tile and rtc_center is the coordinates of the center of the tile in the earth centered earth fixed EPSG 4979 coordinate system. In the batch table, non-spatial attributes of the features are stored along with feature ids.

The reason for the additional tables to store the contents of a tile is that 3D formats which have been investigated in Chapter 1 such as glTF and X3D focus on storing graphical elements which are consumed by WebGL; they do not support non-spatial attribute data.

While decomposing 3D geospatial data into tiles, for each tile, contents of a tile have been written into a single b3dm file. By batching multiple 3D models into a single file, multiple 3D models can be transmitted with a single request and in the visualization pipeline, they can be rendered with the least number of WebGL draw calls.

To create b3dm files for tiles, vertices, indices, and vertex normals are encoded as glTF, then, relevant feature tables and batch tables are encoded.

2.3.3.1.1. Triangulation of the 3D Polygon Surfaces

WebGL uses points, lines and triangles as geometric primitives in order to render objects. Thus, 3D polygonal surfaces of the objects in a 3D geospatial data must be triangulated to display them via WebGL. For this purpose, a 3D polygon triangulation algorithm Ear-Clipping is implemented using earcut4j open-source java library (Figure 29).

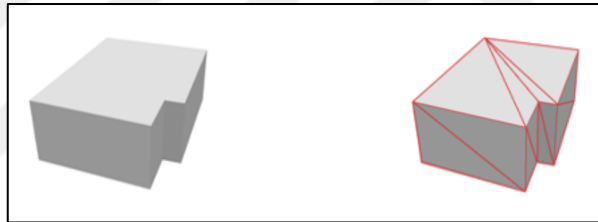


Figure 29. 3D polygon surfaces (left), triangulated polygons (right)

2.3.3.1.2. Calculation of Vertex Normals

Vertex normals are needed in the rendering step for lightning and calculation of the colours of the pixels. Without vertex normals individual surfaces of the 3D models can not be distinguished (Figure 30).

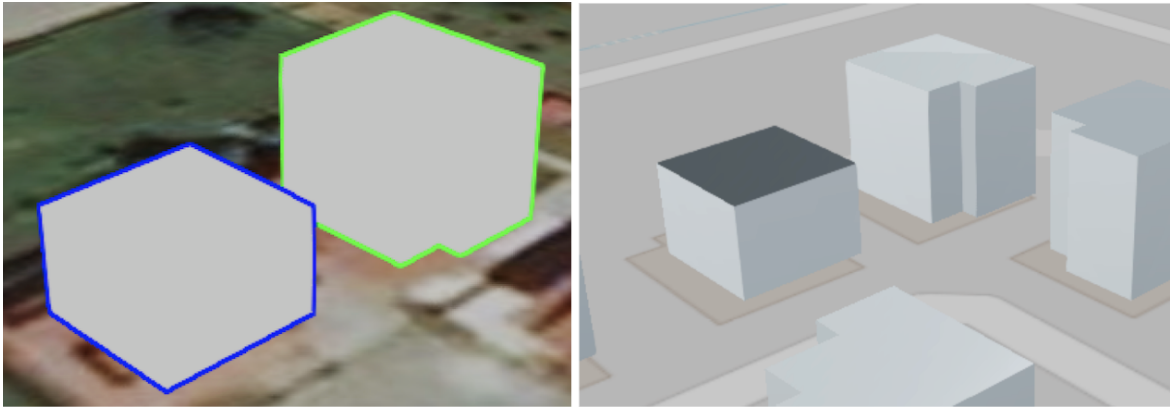


Figure 30. Rendering without vertex normals (left) and with vertex normals (right)

Vertex normals are calculated using surface normals. Surface normal is a unit vector that is perpendicular to the surface. Surface normals are calculated using vertex coordinates and vector math. After calculation of surface normal for each triangle, vertex normals are calculated as the sum of surface normals which the vertices belong to. After calculation of vertex normals, vertex normals have been added to the b3dm file for each tile.

2.3.3.1.3. Clamping to the Terrain

3D geospatial data cannot be considered independent of the terrain, for a realistic visualization, buildings must be placed on the digital terrain models (DTM). Most of the 3D geospatial dataset is not aligned to a digital terrain model or have pre-calculated heights according to a different DTM. In such situations height differences may occur between 3D city models and DTMs hence, heights of the city objects must be aligned according to the DTM (Figure 31).

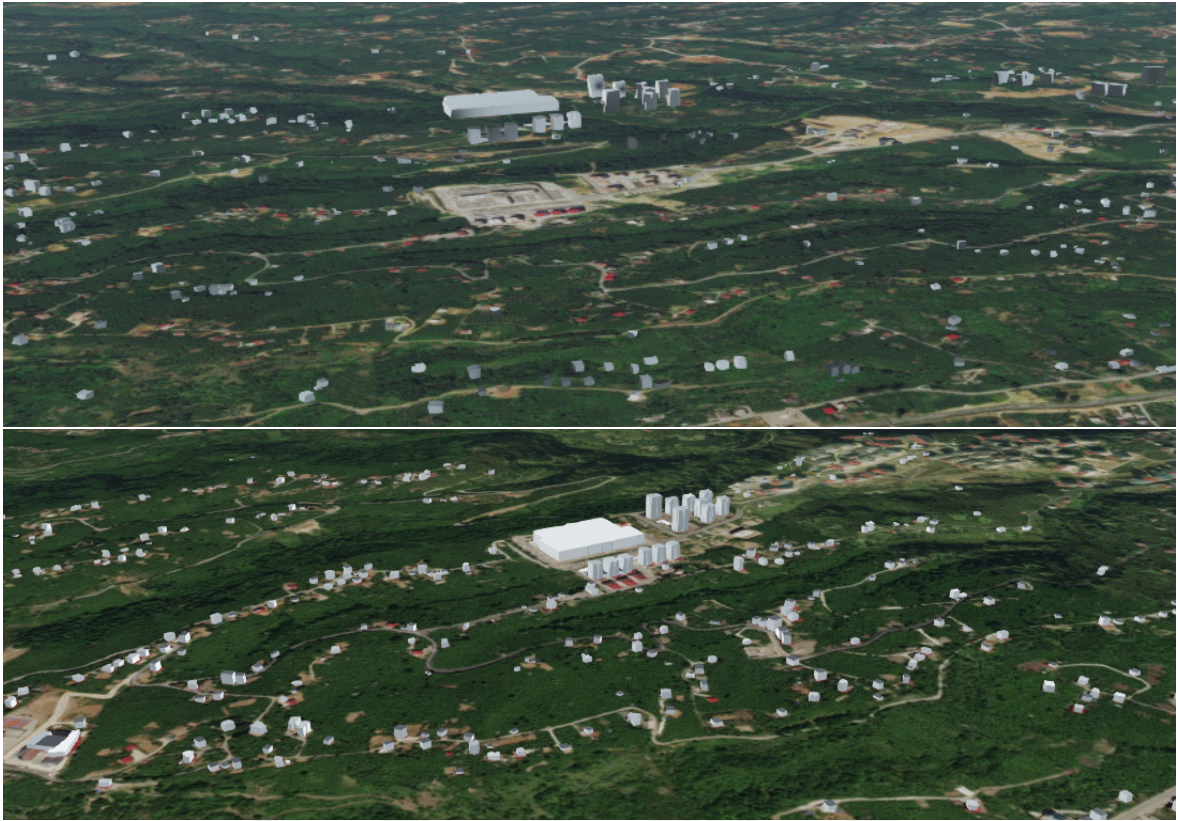


Figure 31. Buildings over the terrain (top), buildings clamped to the terrain (bottom)

In order to clamp buildings to the terrain, first, building footprints intersected with DTM then, an offset along the z axis is applied until the min z value of building footprint matches the min z value of the intersected pixels. Thus, buildings have been clamped to terrain during the tiling.

2.3.3.2. Calculation of the Geometric Error for A Tile

Client implementation will need to determine if a tile is sufficiently detailed for rendering or it must be refined by its children. This decision is made using the geometric error value of the tile. For a tile, the geometric error is used to determine whether the children of the tile should be rendered. At runtime client rendering engines calculate space screen error (SSE) using the geometric error value of the tile according to the following formula (2.4). If the SSE exceeds a pre-defined threshold value, children of the tile are rendered; hence, geometry is refined with a higher level of detail.

There is no formula for calculation of the geometric error value in 3D Tiles specification, hence, a formula has been produced and tested with our datasets. Root tile that

has the most simplified geometry, should have the maximum geometric error value, then this value should gradually decrease in each level of the hierarchy and should be zero at leaf tiles that have the highest level of detail. According to this logic, the following formula has been developed to calculate the geometric error value of the tiles. For a given tile T_i , geometric error of T_i ,

$$G_i = G_{i-1} - G/L \quad (2.4)$$

G_i stands for geometric error of a tile, G stands for geometric error of the root tile and L stands for the level of the tile. Based on the G_i value, at runtime SSE value can be calculated as following formula (2.5).

$$SSE = (G_i \times \text{screenHeight}) / (\text{tileDistance} \times 2 \times \tan(\text{fovy}/2)) \quad (2.5)$$

“screenHeight” stands for the height of the screen in pixels, “tileDistance” stands for the distance of the tile from eye point, “fovy” is the vertical fov angle of the viewing frustum in radians. Relationship between geometric error and space screen error is shown in Figure 32.

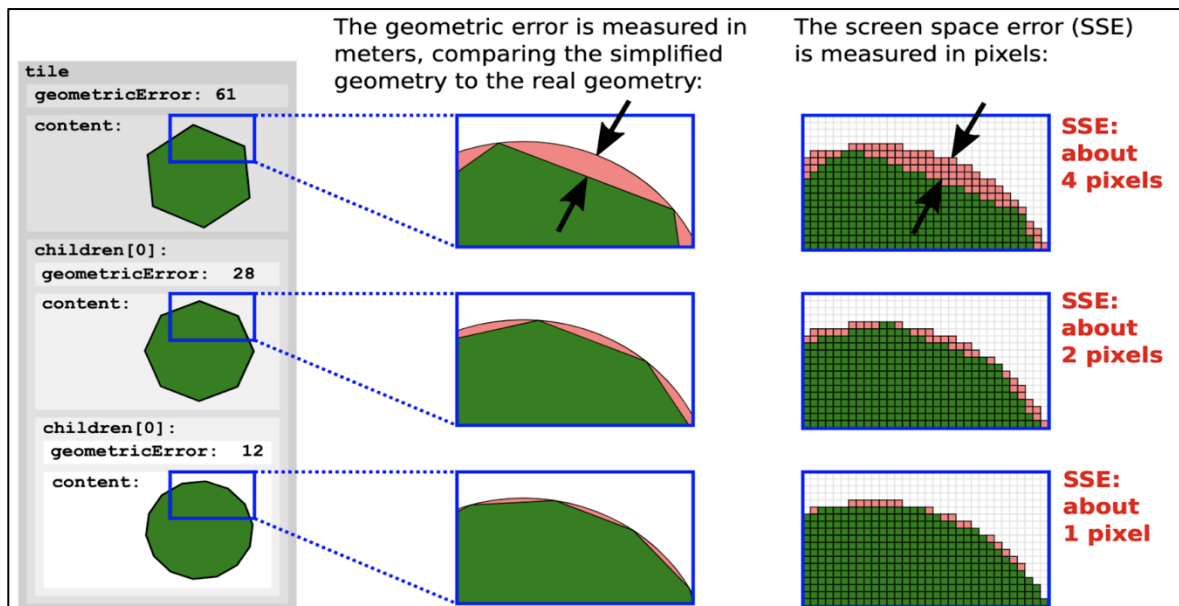


Figure 32. Relationship between geometric error and space screen error (URL-19).

2.3.3.3. Refinement Method

There are two different refinement methods in 3D Tiles. Supported refinement methods are replacement and additive. In the replacement method, child tile is rendered and the parent is no longer rendered. In the additive method, child tile is rendered in addition to the parent. Root tile (blue) and its child (pink) is rendered as “additive” as shown in Figure 33.



Figure 33. A tile and its child rendered using additive method

2.3.3.4. Display of the Tileset on the Browser

For a 3D tileset that is decomposed according to 3D Tiles, at first, the `tileset.json` file is loaded by the client in order to extract the scene graph of the tileset. Utilizing the scene graph, intersection between the bounding volume of the root tile and view frustum is tested. If the bounding volume of the root and view frustum intersects, content of the root is considered for rendering. Then the bounding volumes of the children are tested against the view frustum. Whichever child intersects, their `b3dm` files are fetched and loaded from the server (Figure 34).

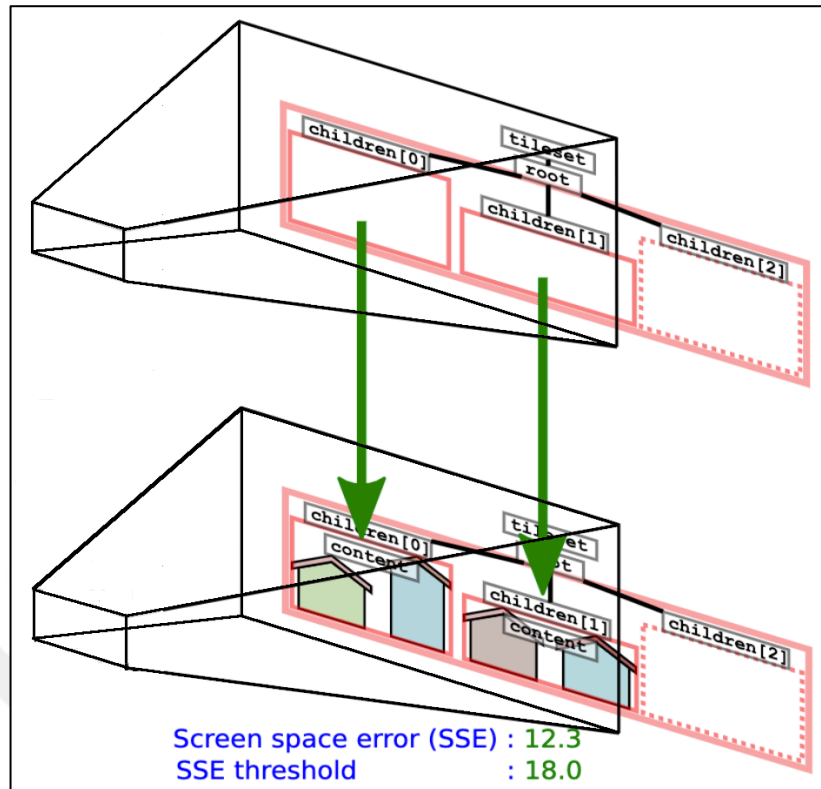


Figure 34. View Frustum and tiles (URL-19)

When the user zooms in or zooms out to the tileset, since the distance between camera and the tile changes, a new SSE value is calculated at runtime using the formula (2.2). If SSE exceeds a pre-defined value, then the next level of tiles is considered for rendering. Then, next level childs are tested against the current view frustum and only intersected ones are loaded and rendered according to the refinement method (Figure 35).

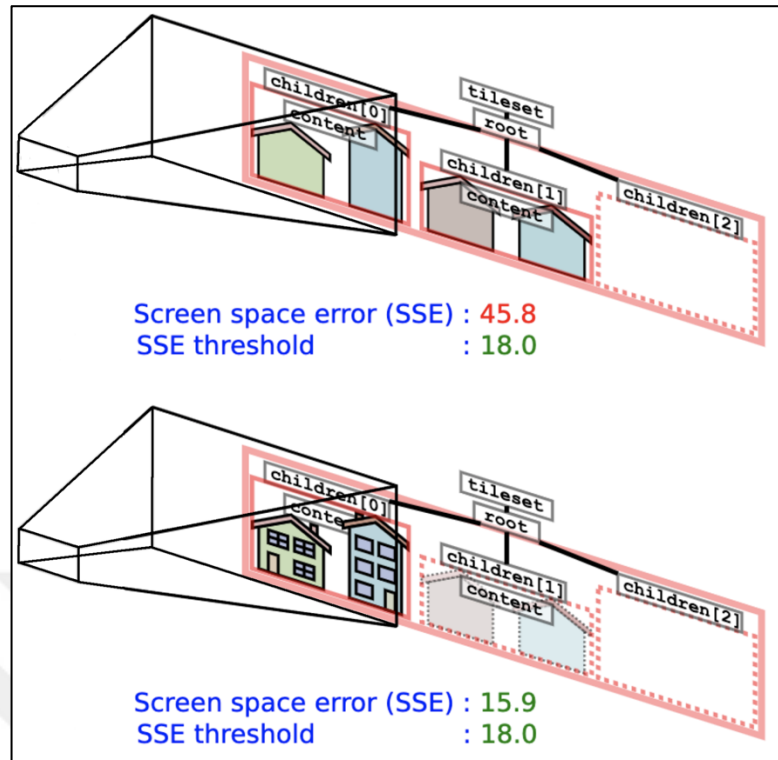


Figure 35. View frustum and tiles (URL-19)

Above mentioned rendering strategy based on scene graph, geometric error and SSE value work well with all hierarchical data structures contrary to rendering strategy of Gaillard et al. (2015) and (Krämer and Gutbell, 2015).

2.3.3.5. Handling of Varying Level of Details (LODs)

For a multi LOD dataset, a multi-representation refinement method has been developed. While interacting with a 3D tileset, if the user zooms in to a tile, while child tile is rendered as in the previous topic, resolution of the parent tile is increased, and a higher level of detail is rendered in place of the current parent. Such an approach cannot be implemented with the current status of the 3D Tiles standard at the time of writing the thesis. Because, in tileset.json file content of a tile is pointed out using a single uri. Multiple LoD of a 3D object cannot be stored in a single b3dm file. Hence, in order to implement the proposed approach, the 3D Tiles standard has been extended. Up to 4 varying numbers of LODs are supported in order to support 4 different LODs of the CityGML. For each LOD of a content in a tile, separate uri's are used to point to separate b3dm files (Figure 36).

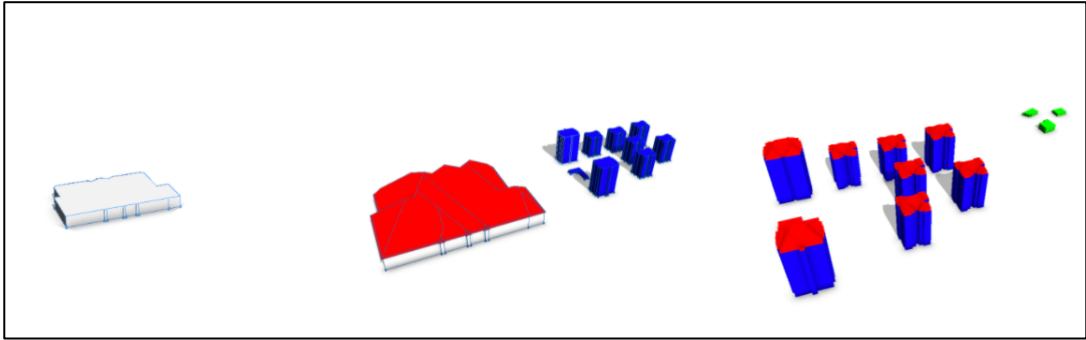


Figure 36. Displaying varying LODs according to the distance of the camera

At first, LOD1 is rendered for a tile (Figure 36, left), if camera zooms in to the tile, child is rendered as LOD1 and also parent is refined as LOD2 (Figure 36, middle), if camera goes on to zoom in to child, new level is rendered as LOD1 and child is refined as LOD2. Extended tileset.json file is shown in Figure 37.

```

"boundingVolume" : {
  "region" : [ 0.6947392307760392, 0.7146410206129109, 0.6954892
},
"geometricError" : 129.18604091253025,
"refine" : "REPLACE",
"content" : {
  "uri_LOD1" : "0LOD1.b3dm",
  "uri_LOD2" : "0LOD2.b3dm",
  "uri_LOD3" : "0LOD3.b3dm",
  "uri_LOD4" : "0LOD4.b3dm",
  "boundingVolume" : {
    "region" : [ 0.6947392307760392, 0.7146410206129109, 0.69548
  }
},
"children" : [ [ {
  "boundingVolume" : {
    "region" : [ 0.6947392307760392, 0.7146410206129109, 0.69519
  },
  "geometricError" : 64.59302045626512,
  "refine" : "REPLACE",
  "content" : {
    "uri_LOD1" : "1LOD1.b3dm",
    "uri_LOD2" : "1LOD2.b3dm",
    "uri_LOD3" : "1LOD3.b3dm",
    "uri_LOD4" : "1LOD4.b3dm",
    "boundingVolume" : {
      "region" : [ 0.6947392307760392, 0.7146410206129109, 0.695

```

Figure 37. Generated tileset.json file for a multi LOD dataset

2.3.4. Development of the RESTful Web Services

Proposed solutions have been implemented through RESTful web services. Client and server communicate with each other through web services developed by using Java Jersey

rest framework. The system overview can be seen in Figure 38. Deployment of the application is done using Amazon Lambda Serverless. Serverless stands for not needing your own servers to execute the application on a server and Amazon Lambda is serverless platform provided by Amazon Web Services (AWS). End users can upload 3D geospatial data in the CityGML format to the server. In the server, the uploaded file is stored in an Amazon S3 bucket. S3 stands for simple storage service and bucket is a data container for uploaded objects in Amazon S3. Amazon lambda functions have been used to process the uploaded file in S3 bucket. Using the citygml4j open-source java library CityGML file is parsed and using the earcut4j open source java library 3D polygons of the city objects are triangulated. Then, using our 3D tilers based on R-Tree and Adaptive QuadTree, the 3D tileset is generated in accordance with the OGC 3D Tiles standard. Then, the end user can download the 3D tileset to its local machine or display the generated tileset using the client of the application. For developing the client, Cesium.js, HTML5, WebGL and Prime Faces technologies have been used. Cesium.js has been used to render 3D tileset and Prime Faces which is an open-source java server pages library has been used to develop the user interface of the client.

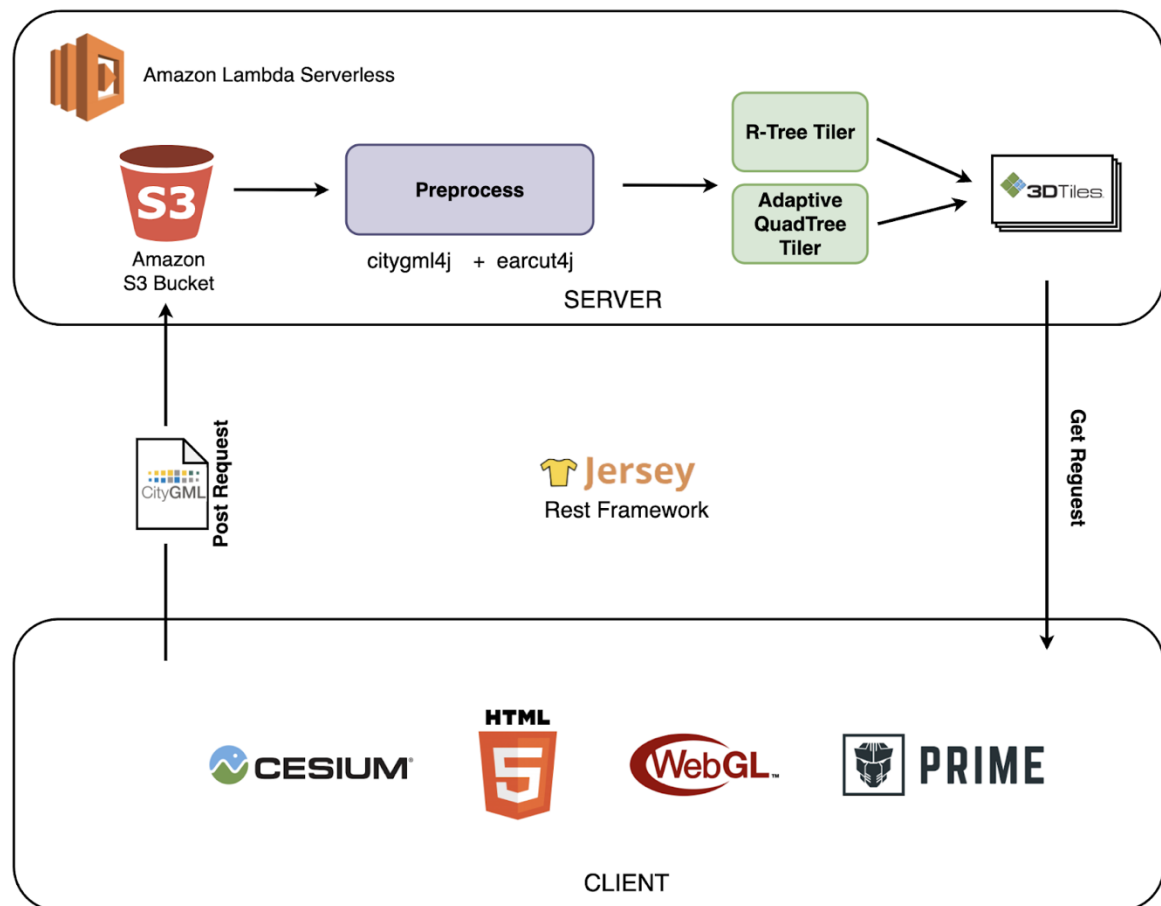


Figure 38. System Overview

2.3.5. Tiling of the Terrain

2.3.5.1. Tile Map Service (TMS) Specification

TMS is a specification for serving maps as tiles on the web that is developed by the Open Source Geospatial Foundation (OSGEO). TMS is also the predecessor of another protocol called Web Map Tiles Service which is developed and published by OGC (TMS, 2010). It aims to provide interoperability between web map applications by standardizing, requesting, and accessing the tiles.

TMS provides a layout based on quadtree for streaming vector data in our case terrain. A quadtree is a hierarchical tree data structure in which each internal node has exactly four children which are exactly equal in size.

TMS uses the $z/x/y$ tile naming scheme for naming quadtree tiles. Z is the zoom level, X is the column number and Y is the row number (Figure 39). It is worth mentioning that

the Y coordinates start from bottom left in TMS unlike many vector tiles such as Google Maps, Bing and OSM which their Y coordinates start from top left.

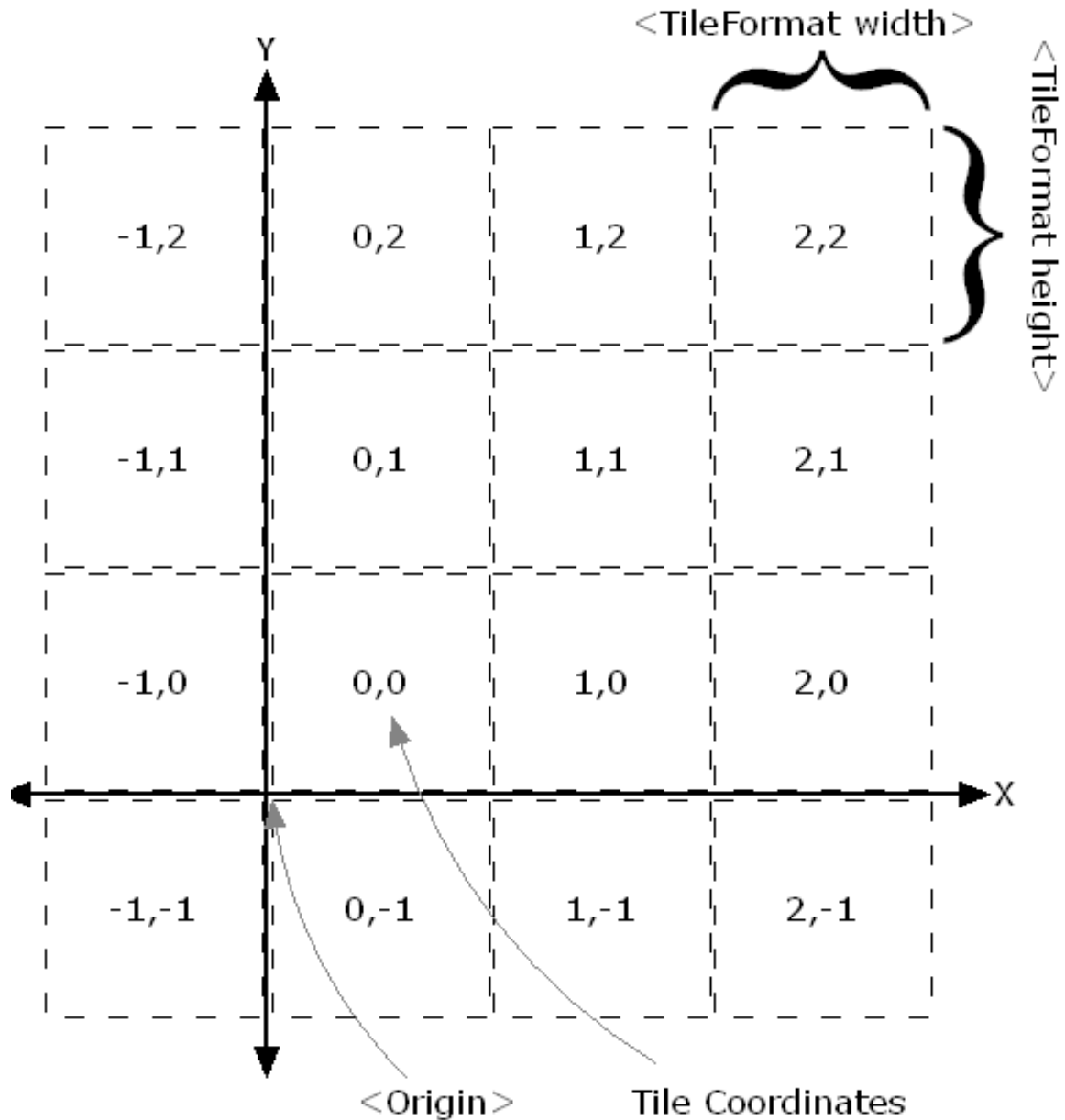


Figure 39. TileMap Diagram according to TMS (TMS, 2010)

2.3.5.2. Quantized Mesh Specification

Quantized-Mesh is a specification and format for streaming massive vector terrain datasets for 3D visualization (URL-20). Quantized-Mesh is based on TMS global geodetic

profile. According to this profile coordinate system must be EPSG 4326. First 3 levels of tiles can be seen in Figure 40. Note that there are 2 root tiles at level 0.

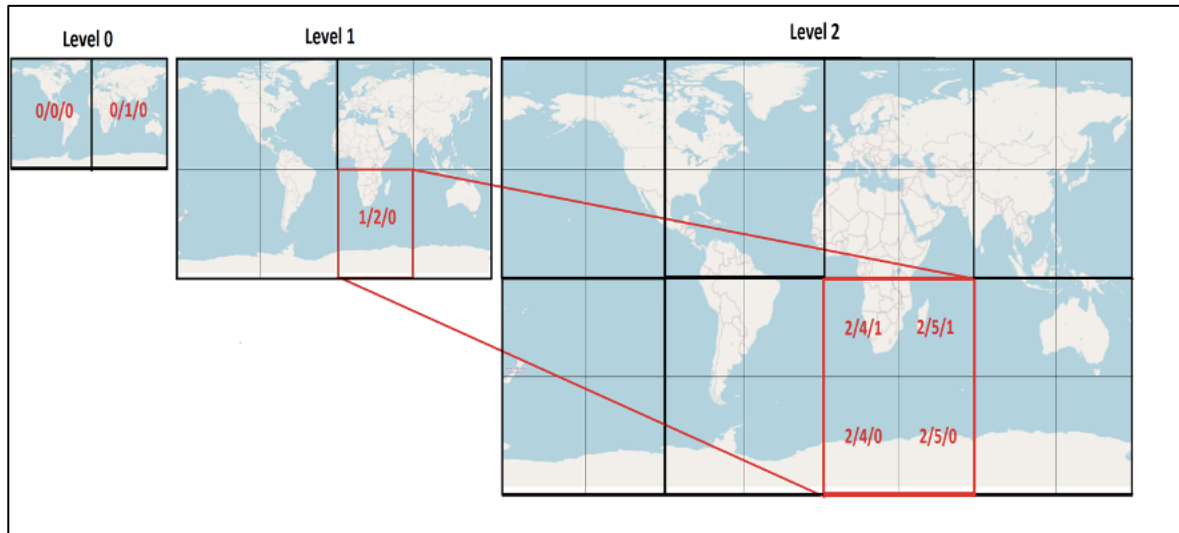


Figure 40. First 3 levels of tiles in the quantized-mesh specification

A terrain tileset in quantized-mesh-1.0 format is a simple multi-resolution quadtree pyramid of heightmaps (URL-20). A JSON file that consists of metadata about this quadtree terrain tileset that is called “layer.json” must be in the root folder of the tileset on the server. Context of the file can be found in Figure 41.

```
{
  "attribution": "",
  "available": [
    [{"endX": 1, "endY": 0, "startX": 0, "startY": 0}],
    [{"endX": 3, "endY": 1, "startX": 0, "startY": 0}]]],
  "bounds": [-180, -90, 180, 90],
  "description": "",
  "extensions": ["watermask", "octvertexnormals"],
  "format": "quantized-mesh-1.0",
  "maxzoom": 18,
  "minzoom": 0,
  "name": "",
  "projection": "EPSG:4326",
  "scheme": "tms",
  "tiles": ["{z}/{x}/{y}.terrain?v={version}"],
  "version": "1.0.0"
}
```

Figure 41. layer.json file for our terrain dataset.

It is worth mentioning the important information in the layer.json file. “available” key contains values of for each tile zoom level, a list of tile ranges is included that defines the tiles that are available in this tileset. Each tile range is defined by x and y TMS coordinates that bound a range of tiles. In Figure 41, only the first two levels are included in order to minimize the figure. “bounds” is the maximum extent of available tiles. “projection” contains coordinate system information. If it is not defined, projection is EPSG:4326 by default. The other option for projection in quantized mesh specification is EPSG:3857 which is used by Google Maps. “scheme” is the tile naming scheme. Available values are “tms” or “slippymap”. The difference between these options Y coordinate numbers which was explained more detailed in the previous section.

2.3.5.3. Generation of TIN Pyramid

The developed tiling algorithm takes a regular raster DTM in geotiff format and creates a TIN pyramid from it. First, contour lines are generated from given raster input and points of the contours are extracted. In order to generate multiple level of details, data must be tiled according to a data structure in this case it is quadtree.

Contours may intersect with boundaries of multiple tiles. In such a case, to ensure data coherence new points are created as a result of intersection of contour and tile border. These points are referred to as “steiner points” in the literature. After the generation of the tiles, for lower levels of tiles, data has been simplified in order to control the data size threshold which is 50kb for terrains.

2.3.5.4. Simplification Process

For simplification of contours Ramer Douglas Peucker (RDP) algorithm has been used. RDP algorithm works recursively. RDP divides a line to sub segments and removes some points until no point is left to remove. A line simplification using RDP is shown step by step from top to bottom in Figure 42. First, start (A) and end points (B) of the lines are marked as kept and a straight line between A and B points is constructed. The point that has the maximum perpendicular distance from this line is selected (Point C in Figure 42). Then two line segments are constructed using this point. One segment is from A to C and the other one is C to B. The point that has the maximum perpendicular distance from these new lines

are found (D and E points respectively in Figure 42). Then again new line segments are constructed using D and E points and points that have the lower distance value than a predefined threshold often called as epsilon in literature are removed from the lines (red points in Figure 42). This process repeats itself until there is no point to remove in the lines.

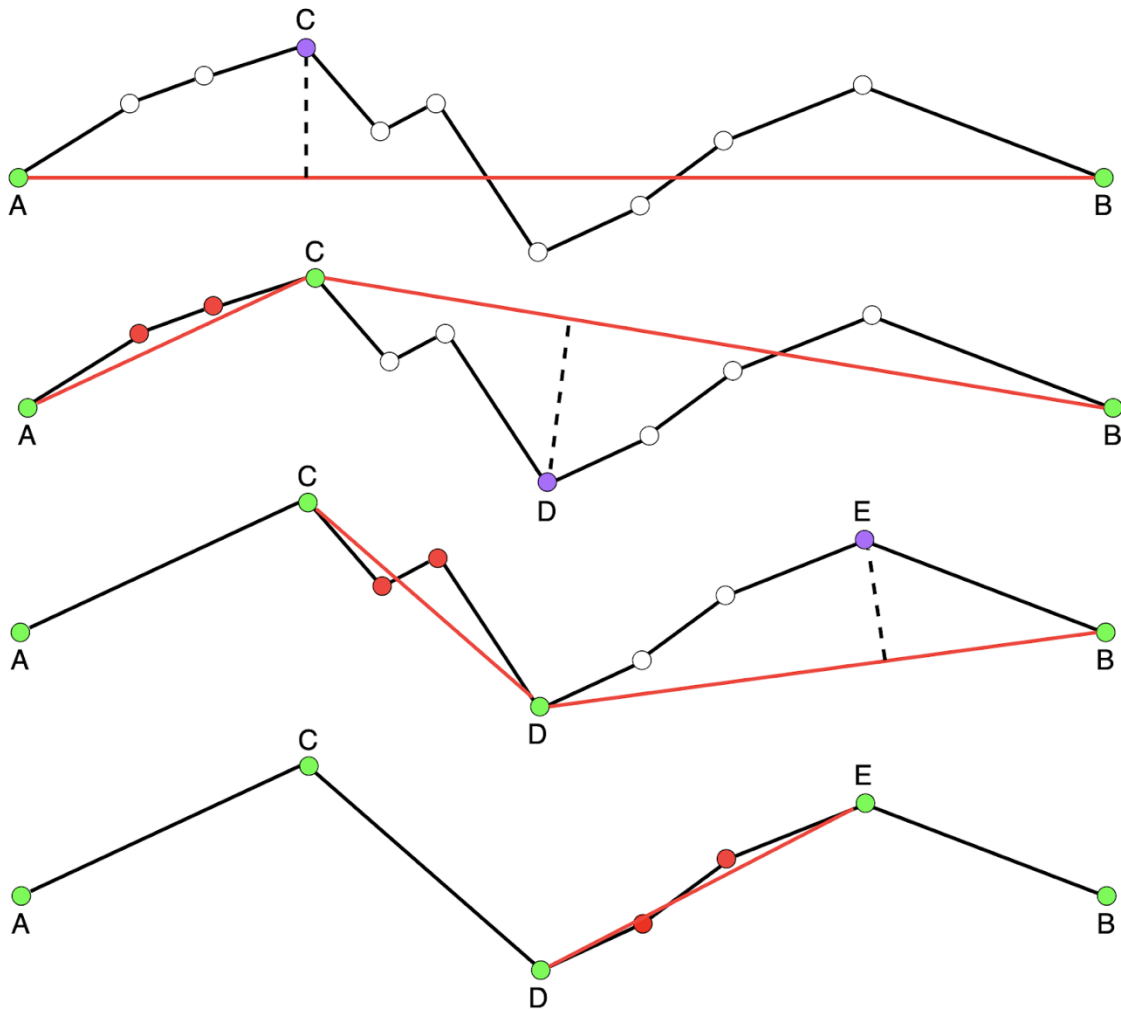


Figure 42. Step by step RDP Line Simplification. Green points are selected as points to keep and red points are removed during the simplification process.

A varying epsilon value has been used for generating varying levels of details. Big vector tile vendors such as Mapbox and Swiss Federal Office of Topography recommend that tiles should not exceed 50KB in size which is approximately equal to 4000 vertices in the terms of data density. To keep data size under 50KB tiles, the starting epsilon value has been used as 8. For a lower level, area multiplied by 4 which means data density approximately multiplied by 4. In order to not exceed 4000 vertices, the epsilon value is

multiplied by 4 for the previous level of tile. In this way, different levels of details have been generated using varying epsilon values which are controlled by the developed tiling algorithm itself. After the generation of multiple levels of details, these lines are triangulated by using a Delaunay triangulation algorithm and TINs are generated for each tile.

2.3.5.5. Implementation of the Quantized Mesh Specification

Data format for terrain tiles is basically a little endian binary blob in quantized-mesh specification and data files have .terrain extension. The header section of the terrain file should contain the following information.

- Center coordinates of the tile in the ECEF coordinate system
- Minimum and maximum heights in the area covered by tile
- Coordinates and radius of tile's bounding sphere in the ECEF coordinate system
- Coordinates of the horizon occlusion point in the ECEF coordinate system.

After the header information, there is the geometric information about vertices of the TINs. vertex count and coordinates of the vertices are stored in this section of the terrain file. In order to implement the quantized mesh, these values must be calculated for each tile.

2.3.5.5.1. Calculation of the Horizon Occlusion Point

In the virtual globe-based engines like Cesium and Google Earth, view frustum culling is not enough for determining invisible objects. In the following image (Figure 43), view frustum is represented as thick white lines. The green points are visible because they are in the viewing frustum, but red points are outside of the frustum hence they need to be culled. But although the blue point is in the viewing frustum, it is invisible to the viewer due to the spherical shape of the earth itself. For culling these "blue points", a horizon occlusion point (HOP) is calculated then if the object is below the HOP it is culled otherwise it is rendered.

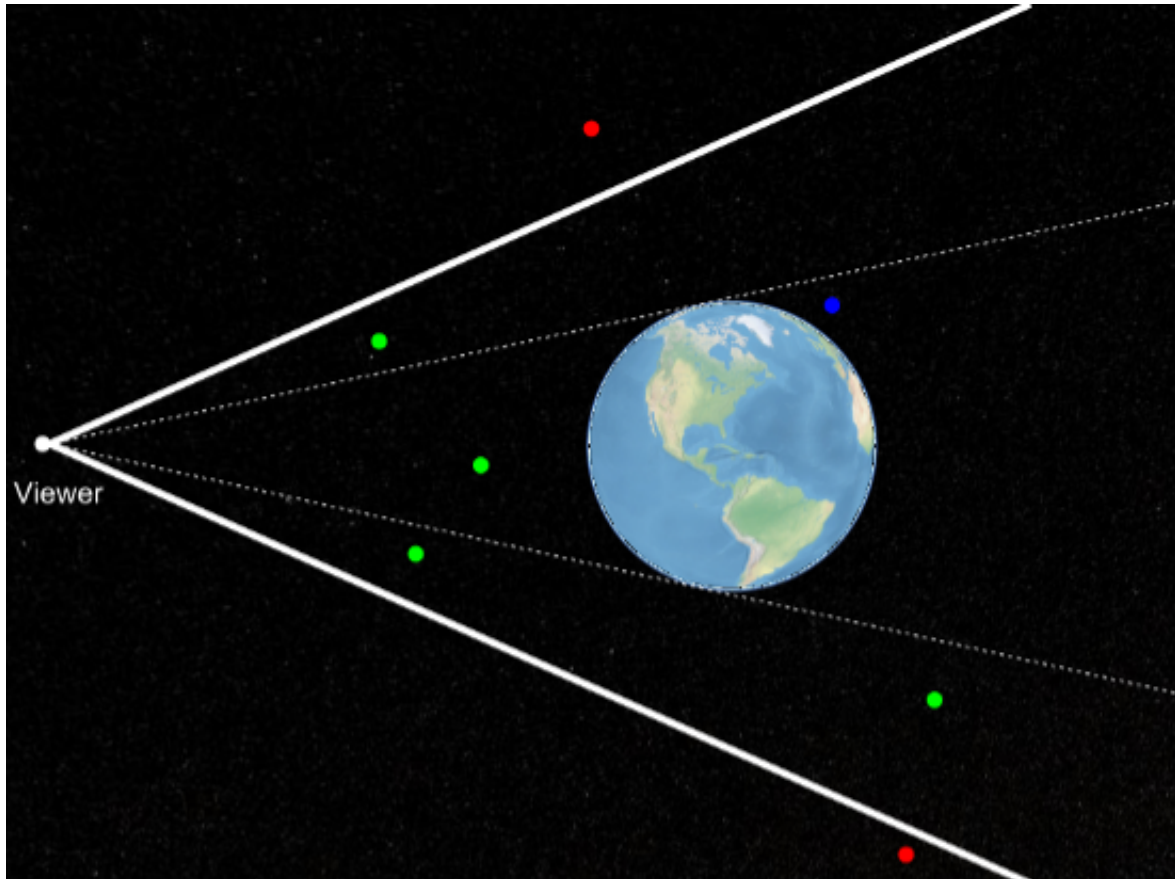


Figure 43. The blue point is invisible to the viewer due to the spherical shape of the earth (URL-21).

In the run time, Cesium implements horizon culling using pre calculated HOP values. The calculation of the HOP in this thesis is based on two great blog posts about horizon culling (URL-21 and URL-22).

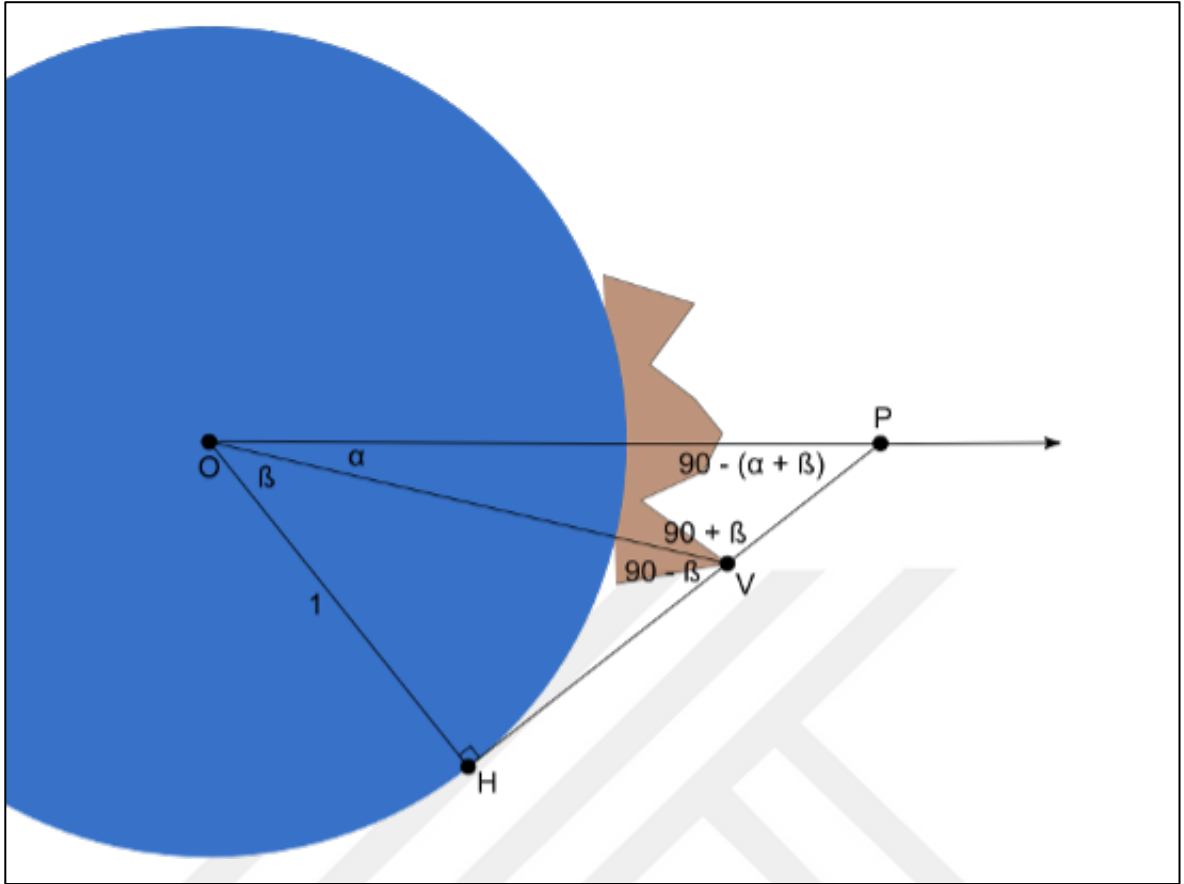


Figure 44. Computation of Point P, the HOP (URL-22)

In Figure 44, the world is represented as a blue unit sphere and terrain tile is brown polygon. “O” is the center of the earth and “P” is the HOP and “V” is a point in the terrain tile. First, all the coordinates are transformed to the ellipsoid-scaled space by multiplying each coordinate with the inverse of the radius of the WGS84 ellipsoid. X, Y, and Z components of the radius are as follows.

$$R_x = 6378137.0$$

$$R_y = 6378137.0$$

$$R_z = 6356752.3142451793.$$

After this transform, for each vertex in the tile a HOP value “P” is calculated using the following formula (2.6)

$$\|\overrightarrow{OP}\| = 1/\cos \alpha \cos \beta - \sin \alpha \sin \beta \quad (2.1)$$

Then the furthest P point from the ellipsoid is selected as the HOP and written to terrain file as double values.

2.3.5.5.2. Calculation of Vertex Coordinates

In order to improve performance of web applications, it is important to reduce data size transmitted between server and the client. Vertex Quantization is a geometry compression technique that compresses data size by normalizing vertex coordinates. In this way, vertices can be represented with less memory footprints.

In WebGL, floats are stored as 32-bit floating-point numbers which are 4KB in size. Using quantization, vertex coordinates can be stored as 16-bit unsigned integers which are 2KB in size. Quantized-Mesh specification encourages this technique and vertex coordinates must be quantized into the 0-32767 range. For quantization, vertex coordinates have been transformed to 0-32767 range using linear interpolation. Since the center of the tile is stored in ECEF coordinates, engines like Cesium can decompress vertex coordinates at run time. Figure 45 shows the Java class that has been used to store terrain data. Observe that since there are no unsigned types in java vertex coordinates are represented using short type instead of unsigned integer.

```

public class QMesh {
    int vertexCount;
    int triangleCount;

    double CenterX;
    double CenterY;
    double CenterZ;

    float MinimumHeight;
    float MaximumHeight;

    double BoundingSphereCenterX;
    double BoundingSphereCenterY;
    double BoundingSphereCenterZ;
    double BoundingSphereRadius;

    double HorizonOcclusionPointX;
    double HorizonOcclusionPointY;
    double HorizonOcclusionPointZ;

    short [] u = new short [vertexCount];
    short [] v = new short [vertexCount];
    short [] h = new short [vertexCount];
    short [] indices = new short [triangleCount*3];
}

```

Figure 45. QMesh class used for storing terrain data in quantized mesh format

2.4. Findings and Discussion

2.4.1. Performance Comparison of R-Tree and Adaptive QuadTree

Proposed tiling methods have been tested on two different datasets:

- Kaşüstü (1282 buildings, 10 km², 20mb)
- Trabzon (56602 buildings, 4685 km², 1.05gb)

Two data structures have been compared based on construction, addition, deletion, and kNN (k nearest neighbour) spatial query performances. The Performances of the operations are measured based on execution time of the algorithms in seconds (Table 5).

Table 5. Performance metrics of the data structures

	Kaşüstü Dataset		Trabzon Dataset	
	R-Tree	Adaptive QuadTree	R-Tree	Adaptive QuadTree
Construction	420.67s	86.51s	2513.80s	483.57s
Addition	0.83s	0.05s	1.12s	0.09s
Deletion	0.67s	0.04s	0.90s	0.09s
kNN Query	0.61s	1.13s	2.14s	4.18s

Since construction of the data structures requires processing each feature in datasets, construction times increase linearly with the data size. However, since addition, deletion and kNN query operations are performed on the only relevant nodes through spatial search, execution times of these operations do not increase linearly with data size. Although the Trabzon dataset is 52 times larger than the Kaşüstü dataset in size, addition, deletion and kNN queries are only approximately 2 times slower than Kaşüstü dataset in Trabzon dataset. These results show the effectiveness of the addition, deleting and kNN query operations in the terms of scalability.

When we compare R-Tree and Adaptive QuadTree, for both dataset there is a huge difference in construction times between R-Tree and Adaptive QuadTree. Generation of R-Tree is approximately 6 times slower than generation of Adaptive QuadTree. Addition and deletion operations are faster in Adaptive QuadTree while kNN queries are faster in R-Tree. Depending on the results in Table 4, if the application uses a static 3D geospatial data and requires kNN queries, R-Tree must be used as the tiling algorithm. If application requires dynamic geospatial data with heavy updates Adaptive QuadTree must be used as the tiling algorithm.

2.4.2. Hierarchical Data Structures or Regular Data Structure

In the related works, 3D geospatial data decomposed to tiles using regular data structures or hierarchical data structures. However, none of the works explain why regular data structure was used or why a hierarchical data structure was used.

In regular data structure, the extent of the data set is subdivided equal sized rectangular tiles. Hence, construction of the data structure is fast and easy to implement when it is compared to hierarchical data structures. The simplicity comes from that there is no check for tile size threshold and there is no check for integrity of the objects. After the generation of the tileset, all the tiles are at the same level without a hierarchy. In such a structure, optimizations based on the hierarchical information can not be implemented. For instance, each tile must be tested against frustum culling for rendering or for a kNN query, range calculation must be done for each tile. Another disadvantage of the regular data structure is heterogeneous tile sizes. Since decomposition is done spatially without considering data density and distribution of the objects, there may be more objects in some tiles than the others. Hence, fetching and loading times of the tiles may vary and can not be controlled. Also, the same is true for rendering. Rendering performance can not be controlled and rendering times and fps may vary.

In contrast to regular data structure, in hierarchical data structures, data set is subdivided into tiles based on a tile size threshold. Thus, a hierarchy and parent-child relationship between tiles is constructed. Hence, construction of the data structure is slower and more complicated than its regular counterpart. However, this disadvantage comes with a lot of benefits. In such a hierarchical data structure, many optimizations can be performed. Using hierarchical information and scene graphs, optimizations can be done for frustum culling on the visualization pipeline. If the bounding box of a tile is outside the view frustum, with that information it is also guaranteed that the childs of the tile are outside frustum as well, hence there is no need to perform visibility checks for the childs. An update operation can be performed efficiently by traversing the hierarchical tree and finding the relevant tiles and performing updates on them. However, for the same update operation, every tile had to be searched in a regular data structure. Another advantage of the hierarchical data structures are homogenous tile sizes. Since decomposition is done using a tile size threshold, tile sizes, loading times of the tiles, rendering times of the tiles and fps can be controlled.

2.4.3. Examining the Differences Between the Proposed Methodology and the Related Works

The related work and their differences from the proposed methodology in this thesis have been given in related work of this chapter. Under this topic, most similar works have been discussed in more detail.

Gaillard et al. (2015) and Krämer and Gutbell (2015) have used a regular data structure for tiling the dataset. In the methodology proposed in this thesis, hierarchical data structures, R-Tree and Adaptive QuadTree have been used. Hence, our work has advantages described in the previous topic. Gaillard et al. (2015) and Krämer and Gutbell (2015) both developed a rendering strategy based on the regular data structure. Tiles that contain the camera and their neighbour tiles are fetched and rendered. The neighbour tiles can be easily found in regular data structure using row and column numbers of a tile. However, this rendering strategy is very hard to implement with hierarchical data structures such as in our methodology. Because neighbour tiles are hard to find in a hierarchical dataset and a tile may have many neighbour tiles that are inefficient to render at once. Hence their rendering strategy does not work with other data structures. In contrast, our rendering strategy is based on hierarchical data structures and works well with all of the other data structures such as kd-Trees, R-Trees, QuadTrees as long as the data structure is a hierarchical tree. Additionally, Gaillard et al. (2015) and Krämer and Gutbell (2015) do not implement a 3D standard such as 3D Tiles and interoperability of the works is limited when it is compared with ours.

The most similar work to this thesis is Gaillard et al. (2020). In their work, a hierarchical data structure has been used and the 3D Tiles standard has been implemented. Instead of using a tile size threshold to create hierarchy between tiles, an interesting approach has been implemented. Hierarchy has been constructed using the road network. The buildings adjacent to main roads have been placed in higher tiles in hierarchy while the buildings not adjacent to main roads have been placed in lower tiles. Such a hierarchy does not represent data size precisely as in this thesis and heterogenous tile sizes may occur. Data size and rendering performance are not controllable as in our methodology.

Cesiumion is the most mature software component to generate 3D tileset according to 3D Tiles standard and it is the most similar product to our developed framework. As in our developed framework, 3D geodata can be uploaded, tiled and visualized using Cesiumion. The main difference from the developed framework in this thesis is that, when 3D geodata is updated, re-tiling must be done for the whole dataset. In our framework, a 3D tileset can be updated without re-generating the whole tileset with update functions such as “insert” and

“delete”. For the Trabzon dataset, generating a tileset is 483 second while updating it is 0.09s (Table 5). There is a huge difference and re-generating the tileset must be avoided as long as possible. None of the components of work support varying LODs as the developed framework.

2.4.4. Layer-Based or Object-Based Tiling Scheme

After the generation of the tileset, when we tested the render performance for displaying tiles, we see that while zooming or rotating render performance drops under 60 fps. The reason was DTM. Rendering a textured DTM is more computationally intensive than rendering untextured 3D buildings. Since tile size threshold is determined using only 3D objects without considering the terrain, rendering performance drops under 60 fps. In order to guarantee the 60fps rendering performance, while determining tile size threshold, terrain also has been considered (Figure 46) and determined as 180KB.



Figure 46. Testing the rendering performance

Above mentioned situation has raised the discussion; tiling should be done as object based or layer based. For a layer-based tiling, each layer is tiled individually. Each layer conforms to the tile size threshold and guarantees 60fps individually. Advantage of the layer-based tiling is that layers can be displayed or hidden individually. The disadvantage is that when users display multiple layers at once, fps may drop below 60 fps.

In object-based tiling, multiple layers are tiled together. Bounding box of a feature is searched in other layers and all the relevant data that correspond to the bounding box in other layers are considered for tiling. The advantage is that multiple layers can be displayed without compromising 60 fps. The disadvantage is that tiling must be done dynamically and adding new layers requires a re-tiling process.

2.4.5. Order of the Operation During Tiling Process

Operations that affect the tile size and render performance must be performed before the tiling. Also, some operations are impossible to perform after tiling.

Clamping to the terrain must be done before the tiling operation. Although a modified model view matrix can be used to adjust heights according to terrain, for a single b3dm file only a single model view matrix is applied hence, buildings of the tile cannot be adjusted individually, the same offset value is applied to all the buildings in the tile. Because of this limitation of 3D Tiles, clamping to the terrain must be performed for each building before the tiling operation.

Calculation of vertex normals and triangulation operations increases the size of the geometric data of the features hence, to not to compromise tile size threshold, these operations must be done before the tiling. Calculation of the vertex normals must be done before the triangulation. Because after the triangulation, although surface normals do not change, since the number of surfaces increases, calculation of the vertex normals will be slower.

2.4.6. High Precision Rendering of the 3D Tileset

In WebGL, coordinates of the vertices are stored as 32-bit single precision floats but in georeferenced 3D geospatial datasets, coordinates are stored as 64-bit double precision values. Georeferenced coordinates are larger than seven decimal digits and exceed the limits of the 32-bit single precision floats. Hence, precision loss occurs in coordinates while mapping them to WebGL. Because of the precision loss, unwanted jittering effect occurs at the rendering stage which affects the quality of the rendering.

To overcome precision loss, the center of each tile is stored in the feature table in earth centred earth fixed (ECEF) coordinate system as RTC values. Coordinates in the tile are

transformed to local coordinates which 32-bit single precision is sufficient by translating them using RTC values. At runtime rendering has been made using translated local coordinates to avoid the jittering effect.

2.4.7. Review of the 3D Tiles Specification

3D Tiles is designed for efficient streaming and rendering massive geospatial datasets and most of the concepts borrowed from Computer Graphics. Understanding and implementing these concepts (Figure 47), requires a steep learning curve for a person that has a GIS oriented background. Moreover, these concepts are not clear and not defined well in the standard.

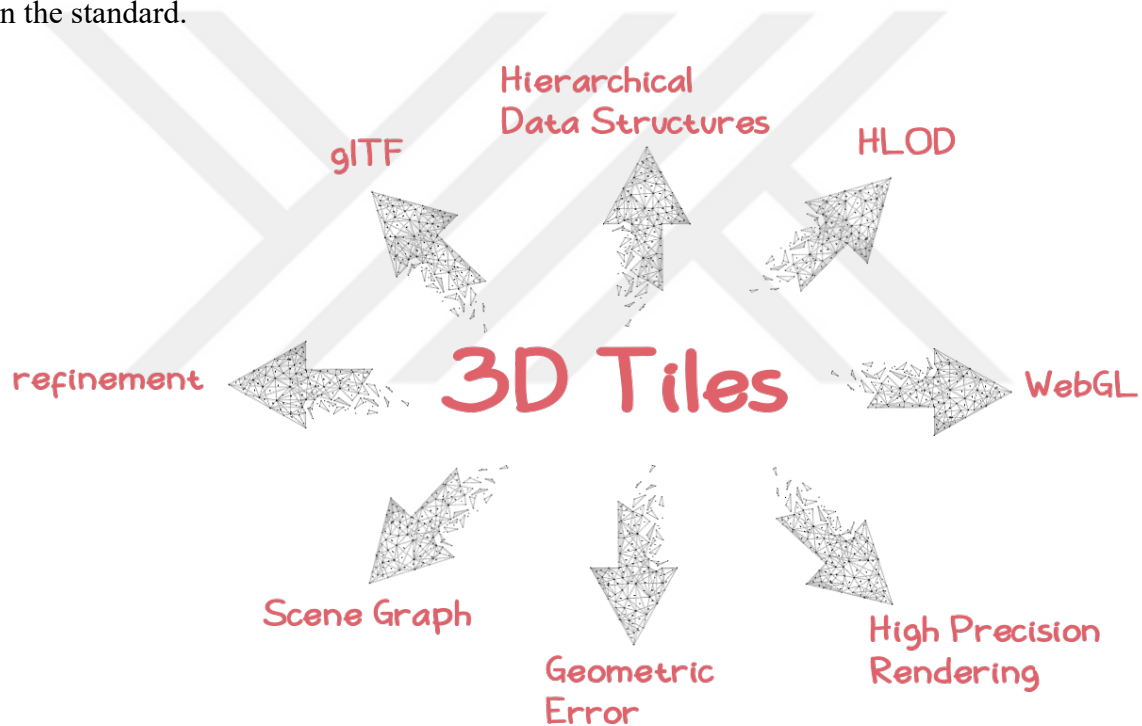


Figure 47. Concepts and Technologies Used in 3D Tiles Standard

There is a “Geometric Error” definition in the standard verbally which is not clear enough to implement. There is no mathematical definition for it hence, calculation of the geometric error is not defined. To overcome this issue a formula has been developed to calculate geometric error which was described in methodology.

Although 3D Tiles specification imposes hierarchical data structures, tile size threshold is not defined, tile size threshold is defined in this thesis based on rendering performance in methodology.

Knowledge about WebGL is required in order to implement the 3D Tiles standard. Error warnings about WebGL must be understood in order to solve problems. Some WebGL errors encountered during the development process were “GL_INVALID_OPERATION: Insufficient buffer size.”, and GL_INVALID_OPERATION : glDrawElements: range out of bounds for buffer. The first error was caused due to writing more data than exists into a buffer for an index array. The second error was caused due to the wrong indexing of the vertices while creating the index buffer. Debugging and solving these problems are not easy due to the verbosity of the WebGL. Error messages are not identical to problems and many other problems would have caused the same error messages.

3D Tiles uses glTF as a 3D model format. glTF is the most efficient format today on the web but debugging and encoding it is not easy. glTF is a binary format hence not human readable which makes it difficult to notice errors. Binary data is represented as Base64 encoded text. Here is a part of our geometric data of a tile as Base64 string, "uri":"data:application/octet-stream;base64,7mQfQqLZI0IAAAA22QfQo7ZI0IAAAAxGQfQpvZI0IAAAAyWQfQqDZI0IAAAAuWQ...". Data is heterogenous, vertices, normals, indices are all in this string. To be able to decode and read data, it should be known which type of data is between which bytes in the buffer. Hence this makes it difficult to debug and decode the data when it is compared to formats such as GeoJSON.

Also, glTF specification dictates 4-byte boundary in order to align data and make data access efficient. Hence, in order to align data in the buffer according to the 4-byte boundary, padding must be done in some parts of the buffer. Sizes of the structures in glTF such as bufferviews and accessors must be a multiple of 4 bytes. While generating the contents, padding is implemented to guarantee that sizes of were a multiple of 4 byte. Although sizes of all glTF structures were a multiple of 4 bytes, we still kept getting warning about the 4-byte boundary on the console of the browser. The reason was that beyond glTF structures, each index must be a multiple of 4 bytes; however, an indice is encoded as an integer array that has tree integer elements. Hence, an integer is 2 bytes, an index is 6 bytes in size. 2-byte padding implemented for each index in the buffer hence, this solved the issue. This situation is not described in glTF specification clearly and wasted more time than it is worth.

2.4.8. Evaluation of the Terrain Tiling and Quantized Mesh Specification

One of the problems encountered during tiling of the terrain was data coherence. Some triangles intersect with the boundary of two or more different tiles (Figure 48).

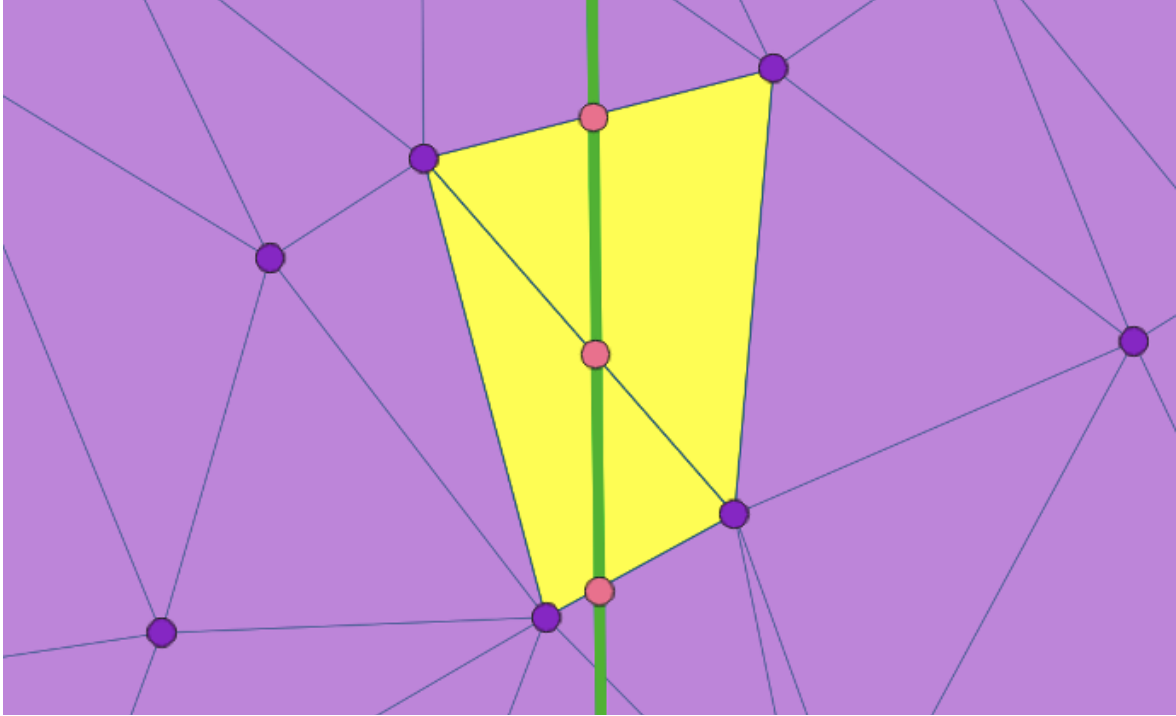


Figure 48. Intersection of triangles (Yellow) with tile borders (Green)

This intersection splits triangles and creates additional points (pink points in the figure 38). The split of triangles changes the topology of the TIN and creates new polygons. These polygons require an additional triangulation process. If these polygons are not handled properly “cracks” that look like “shark teeth” occur on the tile edges (Figure 49).

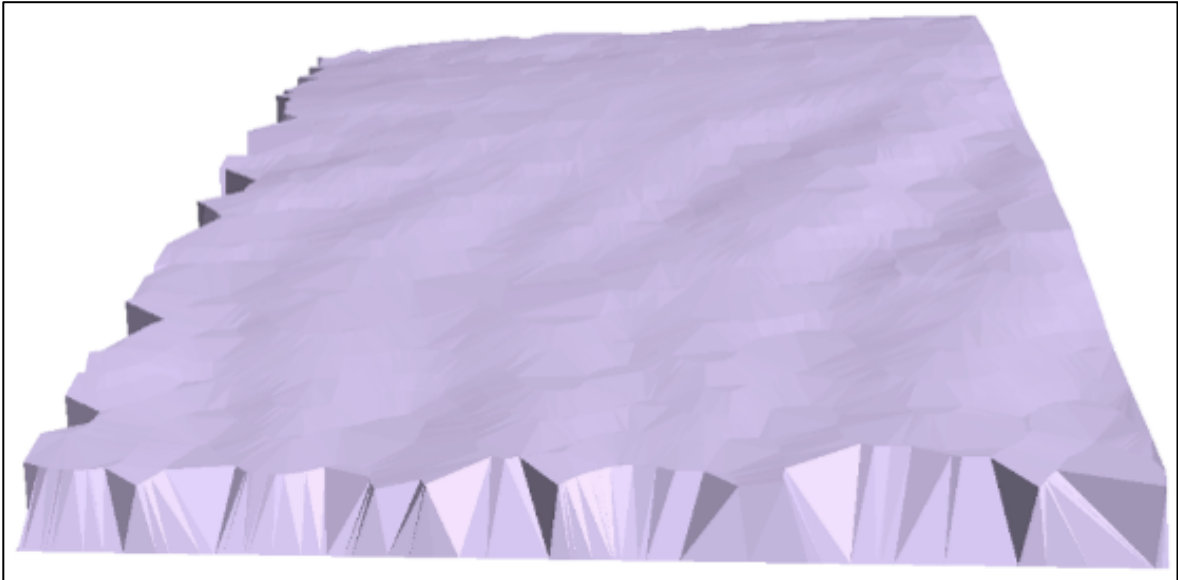


Figure 49. Cracks on the left and bottom edges of the tile.

Additional triangulation of these polygons at the tile borders is a computationally expensive process. In order to find intersecting triangles with the tile borders, each triangle must be tested with tile borders. Hence, finding intersecting triangles is a $O(n)$ process in the term of time complexity.

In order to avoid this computationally expensive additional triangulation process, original data has been tiled before triangulation using contours. Then TINs have been generated by triangulating points of these contours.

While implementing Quantized Mesh specification most of the encountered errors were due to the encoding of the terrain file format not due to the calculations. Because as in the 3D Tiles specification, formats are easy to parse, easy to render but hard to encode and hard to debug.

2.5. Conclusion

In this chapter, the tiling component of the web framework has been developed by implementing the 3D Tiles standard and Quantized Mesh standard. 3D Tiles have been implemented and tested using R-Tree and Adaptive QuadTree data structures for 3D geospatial data. The Quantized Mesh standard has been implemented and tested using QuadTree data structure. End-users can upload, tile, and display their 3D geospatial data and 2.5D terrain using the developed framework through a browser without any software

component installation. Supported formats for 3D geospatial data are OBJ, glTF, CityGML, glb, b3dm and CityJSON and supported format for terrain is Geotiff.

Performance of the hierarchical data structures are compared with each other using two different datasets which vary in size. R-Tree performs better in spatial queries while Adaptive QuadTree performs better in updates and construction.

3D Tiles standard has been extended to support multi-LOD data and multi representation of 3D data has been achieved and one of the limitations of 3D Tiles has been eliminated.

One of the limitations of the tiling component is textures. Textures of 3D models are not supported and not considered in the tiling algorithm. In one of the future works texture support will be developed.

3D Tiles is designed for client-based rendering. In one of the future works, it will be extended to support server-based rendering.

Another important future work would be multithread support. Using multi-threaded programming, tiling processes can parallelize across the multiple cores of the modern computers and performance would be increased.

3. CHAPTER 3 WEB-BASED 3D ANALYSIS AND QUERY OF 3D GEOSPATIAL DATA

3.1. Introduction

With the development and increasing popularity of the concepts and technologies such as smart cities and digital twins, beyond visualization, the need to perform web-based 3D analysis and simulations on 3D geospatial data has arisen. The true power of GIS is to analyze geometric and topological properties of features and extract new information from them.

3D analysis and query capabilities of existing works are quite limited. In most of the works named as “web-based 3D analysis”, analyses are performed on the desktop as offline, and results of the analyses integrated to a web application. In such an application only, the results are visualized and can be queried as online. The major drawback of this type of application is that the user has to work in multiple environments and with multiple software components. For an analysis, when a value of a parameter changes, the user has to re-perform analysis on desktop and pack and send the results to the client for visualization and query. Additionally, two different software cannot interact with each other automatically. Hence, most of the time a user intervention is required to complete the remaining steps for the web-based visualization of the analyzed results. This type of applications cannot be called as true 2D WebGIS applications where all the interactions of the user can be performed on a single web-based environment and software.

In this chapter, 3D analysis and query components of the framework have been developed. Thanks to the developed components, users can perform 3D analysis and queries on 3D geospatial data on a single web-based software and can visualize results. In order to analyze 3D geospatial data efficiently on the web, a new fast 3D intersection algorithm has been developed. To be able to visualize results of the analyses, analysis components have been integrated with the tiling component which was described in the previous chapter.

3.2. Related Work

Many existing works entitled as 3D WebGIS were about the visualization of the 3D analysis on a client application, rather than performing 3D analysis itself online. Common point of aforementioned works in this paragraph is that although they are named as 3D WebGIS they do not perform 3D spatial analysis online. Achere et al. (2016) visualized results of a flood analysis as 3D. In another work entitled as 3D WebGIS Feng et al. (2011) visualized terrains in 3D. Similarly, Chen et al. (2016) developed a framework for 2D and 3D visualization of geospatial data to manage landslide hazards. Xiaoqing et al (2010) visualized results of a shortest path analysis in 3D by integrating ArcGIS and Skyline software. Li et al. (2015) visualized earthquakes on a globe as 3D. Schwerin et al. (2013) developed a VR application in which you can interact with 3D models. With this tool, users can search and query, in real time via a virtual reality (VR) environment, segmented 3D models of multiple resolutions that are linked to attribute data stored in a spatial database but cannot perform 3D spatial analysis. Pispidikis et al. (2016) developed a web-based tool to query and visualize CityGML data. Although they mentioned 3D spatial analysis in their article, which spatial analysis can be performed are not explained and also, results of the work do not show 3D spatial analysis. They mostly focused on querying the CityGML data and visualizing queried results.

The most similar works to our proposed methodology are Chaturvedi (2014) and Auer and Zipf (2018). Chaturvedi developed a web-based tool to perform 3D buffer and 3D intersect analyses. The drawbacks of this work are that analyses are performed on the client using not on the server. Hence, when the analyzed 3D geospatial data exceeds the memory of the browser, the application does not work. Application is highly dependent on the size of the input data. Another limitation is that, in 3D intersection analysis, the application cannot calculate intersection points for partially intersected objects. As can be seen from the images of the results, partially intersected buildings are considered to be completely inside the intersected object, in that case the sphere. In their work Auer and Zipf (2018) developed a 3D WebGIS application that performs 3D line of sight analysis on the browser. The limitation is that analysis performed on the client as in Chaturvedi, hence application is highly dependent on the size of the input data.

3.3. Methodology

3.3.1. Storage of the 3D Geospatial Data

File systems offer rapid access to data which is helpful for fast reading and extracting of the necessary information from data when it is compared to databases. On the other hand, storing data in databases is a more convenient way for analysis and query operations. A hybrid solution which benefits from both approaches is adopted in this thesis and in the developed framework. After the tiling 3D geospatial dataset 3D tileset is stored directly in the file system on the server with a metadata file (tileset.json) referencing file locations. Such an approach which was used and described in the previous chapter is beneficial for visualizing purposes. For analysis and query operations 3D tileset is stored in a database.

In relational databases, data is stored in the tables according to a schema. Mostly to avoid data redundancy, different classes of the data model are stored in different tables in the database. To query a feature, multiple numbers of tables join together. Join operations are the main bottleneck of the relational databases. Update operations require updating many records in many tables.

Unlike relational databases, in NoSQL databases, data is not stored in rigid table structures. Data may be stored in columns, as key-value pairs, in graphs or in documents according to the type of the NoSQL database. The data to be stored in the database may have very different shapes and sizes and designing the schema in advance may be a real pain. In NoSQL databases, it is possible to store polymorphic data in a single document. Hence this schema-less approach gives developers greater flexibility while developing applications.

In this thesis open-source mongoDB which is a document-based NoSQL database is used. A flat schema has been designed to store hierarchical tiles in the database. Our application database consists of two collections. One collection is for the tile contents and scene graph data and the other collection stores non-spatial attributes of the features of the tiles.

To store a tree-like hierarchical data structure in mongoDB in our case 3D tiles, one way is organizing documents by storing references to "child" nodes in "parent" nodes. There are four more different ways to store hierarchical data in mongoDB which can be found in the URL-23. Since each operation in our API starts with traversing the 3D Tileset from top to down and finding relevant nodes, tiles are stored within a single collection as each tile is

a single document in this collection with references to child tiles. 3D tiles are stored in the “tiles” collection (Figure 50) and feature attributes are stored in the “attributes” collection (Figure 51).

```

{
  "tile_id":0,
  "child_id":1,
  "bbox":[0.6947392307760392, 0.7146410206129109, //Xmin, Xmax, Ymin, Ymax, Zmin, Zmax
         0.695489298613394, 0.7150729689986538,
         0.0, 42.0],
  "geometricError": 129.18604091253025,
  "content":{
    "positions":[0.694940567933935, 0.695075572148087, 0.0,
                0.7146470433193278, 0.7147813380204292,18.0], //x1,y1,z1,x2,y2,z2...
    "indices":[[0,1,2],[0,2,3]...],
    "normals":[0.695172701999445, 0.7147489602737959,
               0.6952780587499182, 0.7148553978585277...],
    "batch_id":[0,0,0, 0,0,0, 1,1,1, 1,1,1, ...],
    "bbox":[0.6947392307760392, 0.7146410206129109,
            0.695489298613394, 0.7150729689986538,
            0.0, 42.0]
  },
  "BATCH_LENGTH":8,
  "RTC_CENTER": [3703348.46439685,
                 3089920.9581108703, // X, Y, Z center in ECEF coordinates
                 4159531.4757498046]
}

```

Figure 50. An example of a single document in the “tiles” collection.

“tile_id” is the id of the tile and “child_id” is the “tile_id” of the child tile. “bbox” is the coordinate array of the bounding volume of the tile in the order of Xmin, Xmax, Ymin, Ymax, Zmin, Zmax. “positions” are the array of vertex coordinate triplets in the local coordinate system. “indices” are the triangle surfaces those values refers to vertices in the positions array. “normals” are the vertex normals and “batch_id” values are the ids for each feature in the tile. “BATCH_LENGTH” is the individual feature number in the tile and “RTC_CENTER” values are the coordinates of the center of the tile in the ECEF coordinate system. Note that some values are truncated to adjust the size of the figure.

```
[{
  "tile_id":1,
  "batch_id":0,
  "property":{
    "building_height":30,
    "number_of_storeys":10
  }},
{
  "tile_id":1,
  "batch_id":1,
  "property":{
    "building_height":48,
    "number_of_storeys":16
  }}
...
]
```

Figure 51. An example of two documents in the “attributes” collection

In the “attributes” collection, “tile_id” is the id of the tile which can be used to make queries faster while querying features of a specific tile. “batch_id” is the id of the feature in the tile and “property” is an object that can have any number of non-spatial attributes. In our example features have two attributes: “building_height” and “number_of_storeys”.

3.3.2. The 3D Intersection Algorithm

3.3.2.1. Detection of the Collision

The 3D intersection algorithm consists of two parts. First part is to detect whether there is an intersection between two 3D objects and the second part is to construct the intersected section as a 3D polygonal mesh.

The algorithm starts with figuring out whether two input 3D polygonal mesh models intersect or not and return a boolean value as a result. To determine the collision between objects a plane is swept through objects. Vertices of the objects are stored in an array and as they intersect the surface while the plane is being swept, they are removed from the array. If the surface intersects with the vertices of the second object before vertices of the first object completely removed, there is an intersection, and this part of the algorithm returns “true” Boolean value. Otherwise, objects do not intersect with each other.

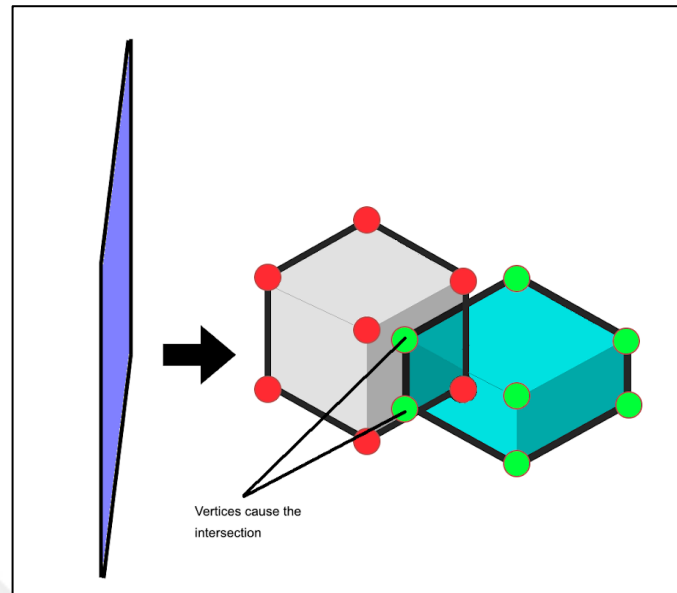


Figure 52. Sweep Plane and collision detection

3.3.2.2. Construction of the Intersected 3D Polygonal Mesh Model

To construct the intersected 3D mesh properly, additional topological information such as adjacency of the vertices and adjacency of the surfaces is needed. For this purpose, a topological polygonal mesh data structure called “Half-Edge” data structure is implemented.

Half-Edge (HE) is a combinatorial data structure, and it suits well for our intersection algorithm as long as the 3D data is manifold which means each edge is incident to at most two faces. In HE, edges vertically divided to two directed half edges (Figure 43). A simpler data structure for storing 3D data which traversing is faster than HE such as Circular Doubly Linked List (CDLL) is not suited well. The problem is that a vertex may have more than two adjacency vertices but CDLL points to only one next and one previous element. A data structure that points to more than one element is needed and this is where HE comes to usage.

In HE;

- Each vertex has a reference to its half edge where the half edge starts at this vertex.
- Each face has a reference to one of the half edges that bounds it.
- Each half edge has a reference to its start vertex, to a face that it belongs to, to its twin half edge and its previous and next half edges.

Using this information, a face can be traversed with its vertices, half edges, and adjacent faces. An example of a HE records given in Table 6.

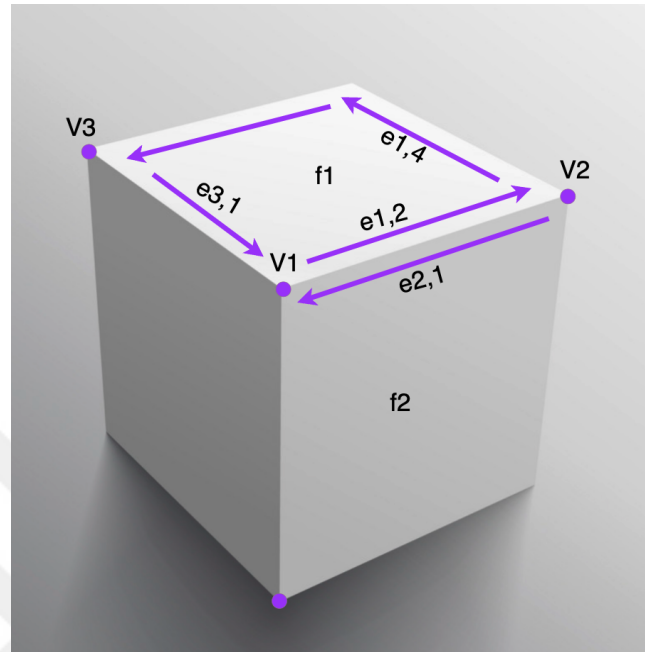


Figure 53. Half-Edge data structure

Table 6. Example of Half-Edge Data Structure

Edge	Start	Opposite	Face	Next Edge	Prev. Edge
e1,2	V1	e2,1	f1	e1,4	e3,1
e3,1	V3	e1,3	f1	e1,2	e4,3

Figure 54 Shows two objects to calculate intersection between them. Please note that objects do not intersect each other yet. To calculate intersection, first, Object B was translated, and intersection occurred in Figure 55.

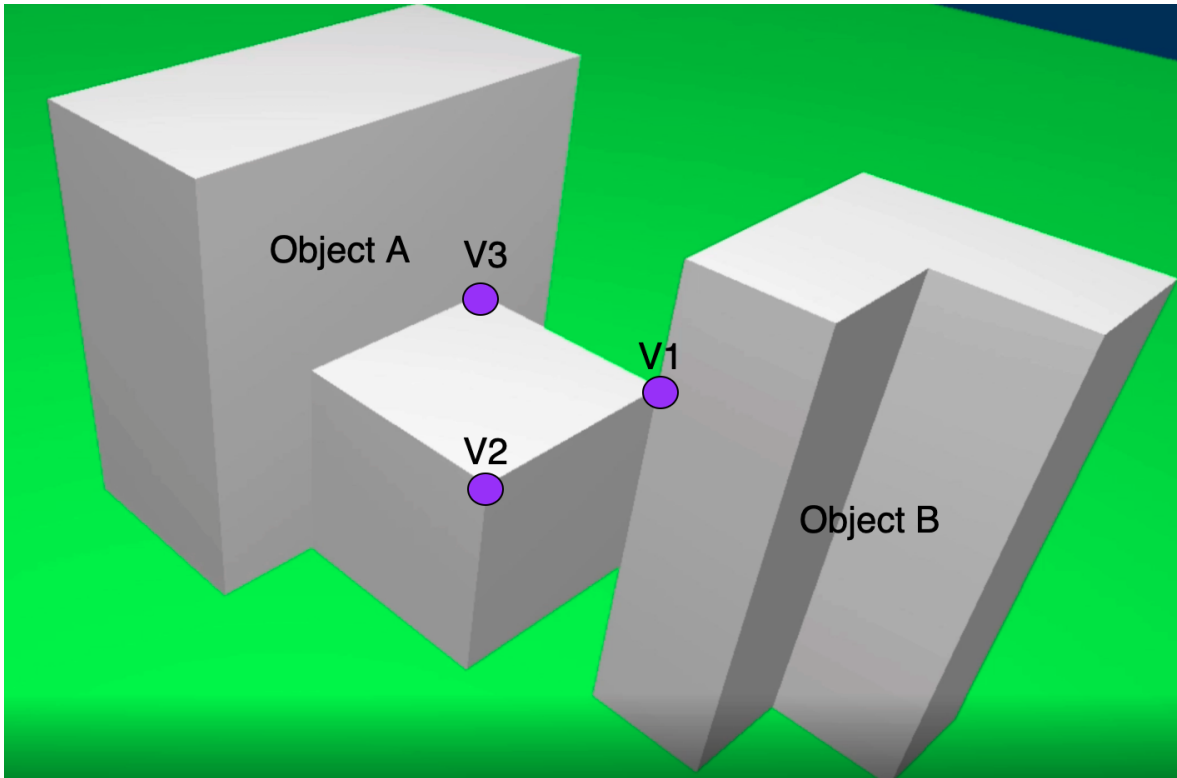


Figure 54. Two objects are about to intersect.

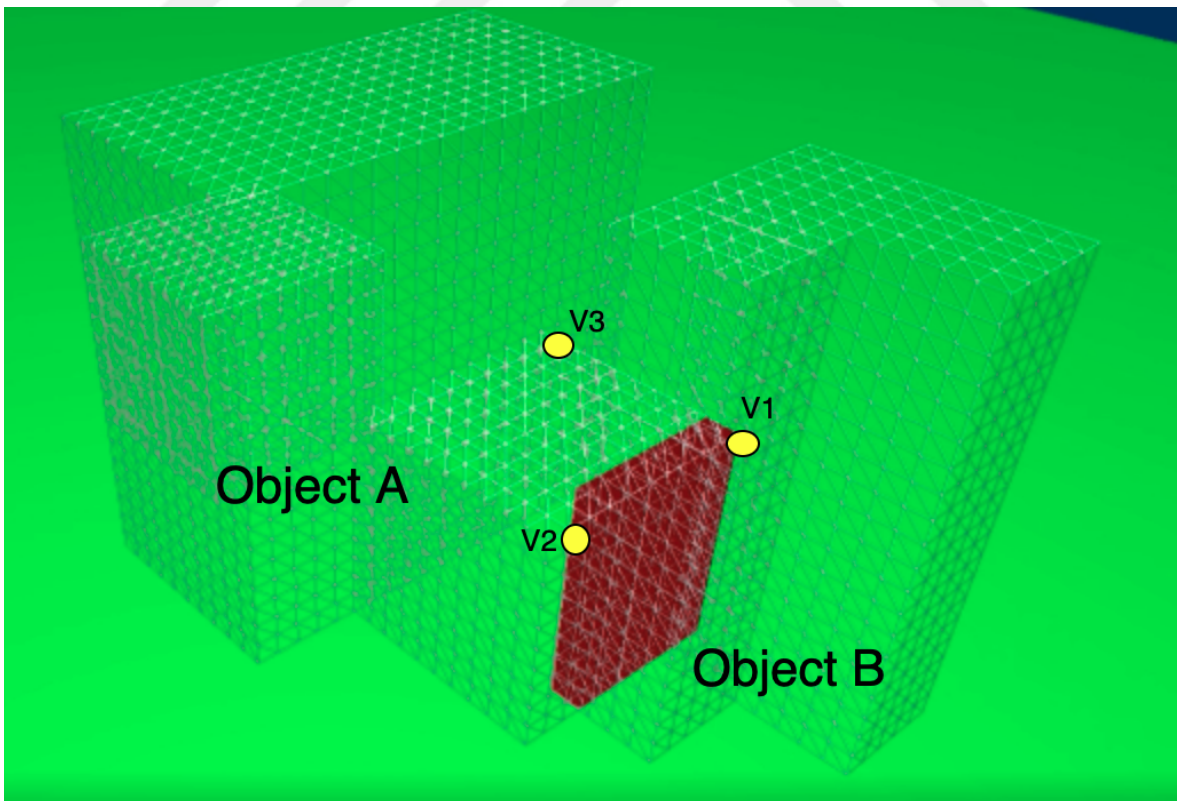


Figure 55. Intersection is calculated and rendered as a red polygonal mesh.

To calculate intersection, first, the one of the original vertices of Object A which is completely inside Object B is found. In the example in Figure 45, it is V1. Then the algorithm finds neighbor vertices from HE data structure. Using V1 and its neighbors V2 and V3, the algorithm sends rays along the directions of V1-V2 and V1-V3. Then using ray-surface intersection tests, the points where the rays intersect with the surfaces of the Object B is calculated. Using HE structure, new points which occur as a result of intersection are indexed in the right order and stored as a new HE structure.

Using the 3D intersection algorithm many types of 3D overlay analyze also referred to as Boolean operations in Computer Graphics and Computational Geometry can be calculated. Implemented types are 3D intersect, 3D difference, and 3D clip. In 3D Clip, outside of the intersected part of the first input with the second input extracted from the first input, only the intersected part remains in the first input. In 3D Difference, the intersected part is extracted from the first input.

To test our 3D intersection algorithm with real world datasets, by utilizing 3D Difference, LOD1 block models of Kaşüstü dataset and Trabzon datasets used in Chapter 2, extracted from LOD2 models of the same datasets and thus, roof geometries are derived for Kaşüstü and for Trabzon.

3.3.3. Integration of the 3D Intersection Algorithm with Tiling and 3D Tiles

In the previous topic, the 3D intersection algorithm is explained using two 3D objects. However, it is designed to calculate intersection between two different 3D tileset which consists of many 3D objects. Note that, in the previous section 3D objects have been tessellated to many small triangles to represent data density of a 3DCM on a single object which is easier to debug degeneres caused by the algorithm.

Having to work via multiple data chunks in web environments is used to our advantage in order to accelerate the 3D intersection algorithm and integrate it with tiling structure. To integrate the 3D intersection algorithm with the tiling structure, first input 3D geospatial data is tiled and the tileset stored in mongoDB. Then, the second 3D geospatial dataset is tiled and stored in mongoDB. Then, tiling structure is used as an index structure for the 3D intersection algorithm. Thus, for each tile in the first 3D tileset, by using bounding boxes of the tiles, only relevant features derived from the second 3D tileset and intersection calculations are made tile by tile. The result is a different 3D tileset as long as the tile size

threshold is exceeded. As the intersections are calculated, a new tileset is constructed by importing intersected objects to a null R-Tree or Adaptive QuadTree structure. If the density of the results do not exceed the tile size threshold, the result is constructed as a single tile not a tileset.

3.3.4. Other Types of 3D Analyses Implemented in the Framework

Additionally, to 3D Clip, 3D Intersection and 3D Difference, 3D Buffer is also implemented. As a result of the 3D Buffer analysis, a point is converted to a 3D sphere and a polyline is converted to a cylindrical polygonal mesh (Figure 56). Buffer distance can be given by the user in meters. Another important parameter is “quad_segs” which is the number of the segments used to approximate a quarter circle. The default quad_segs value is 8. The lower values can be used to create more simplified versions of the spheres and cylinders which makes calculation faster or higher values can be used to construct more precise 3D shapes.

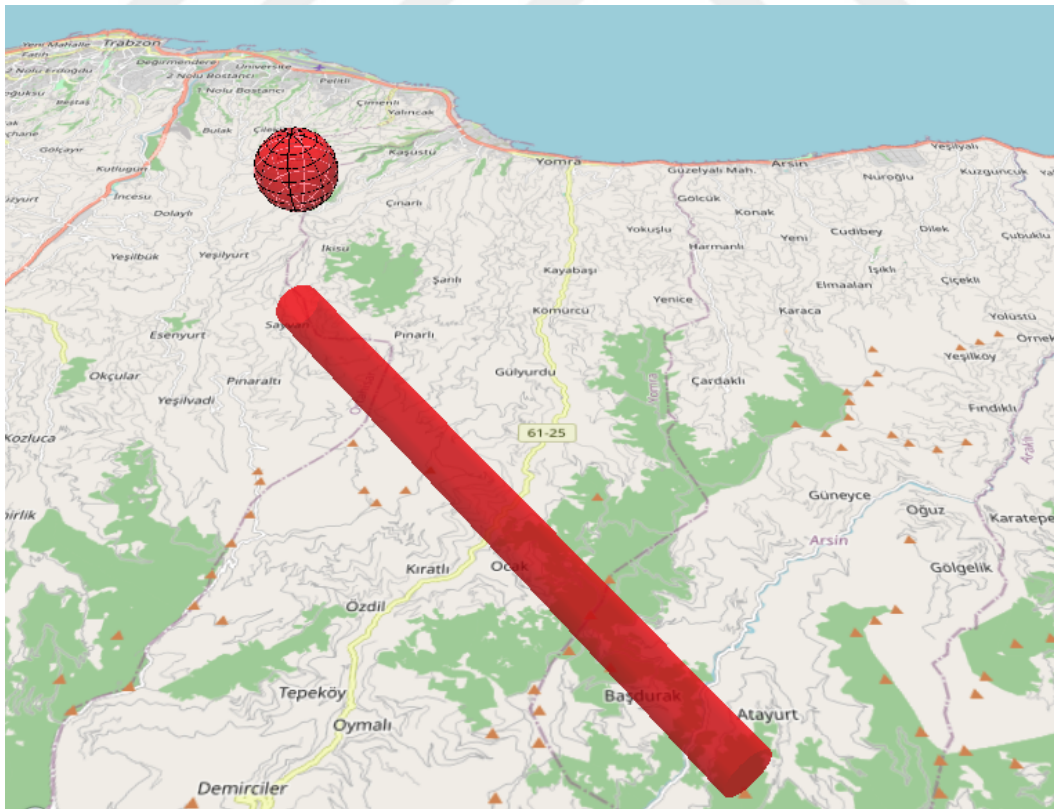


Figure 56. A sphere and a cylinder constructed as a result of 3D Buffer analysis

3.4. Findings and Discussion

3.4.1. Performance of the 3D Intersection Algorithm

3D Difference analysis is used to test the 3D intersection algorithm using both Kaşüstü dataset and Trabzon dataset. In the test for both datasets, LOD1 models are extracted from LOD2 models and thus, 3D models of the roofs are derived. The results of the performance tests can be found in Table 7.

Table 7. Performance of the 3D Difference Based on 3D Intersection Algorithm.

Dataset	Number of the Vertices	Execution Time (ms)
Kaşüstü	303275	16.78
Trabzon	1334020	29.63

Note that execution time for construction of the tilesets for input datasets is excluded. Also, rendering time of the result tileset is excluded. Only calculation of the intersections and construction of the 3D polygonal meshes are considered. Without the use of the hierarchical data structures, calculation of intersection is an $O(n^2)$ process and n is the number of the vertices. But using the hierarchical data structures as an indexing structure, although the number of the vertices increased approximately 5 times in the Trabzon dataset, execution time is only approximately 2 times slower and shows the scalability of the 3D intersection calculation with hierarchical data structures.

3.4.2. Limitations

One of the limitations of the algorithm is that it cannot handle non manifold 3D shapes. Because in the HE data structure only left and right faces of an edge can be represented but in non-manifold shapes an edge is incident to more than two faces (Figure 57). Hence the proposed algorithm cannot calculate 3D touches.

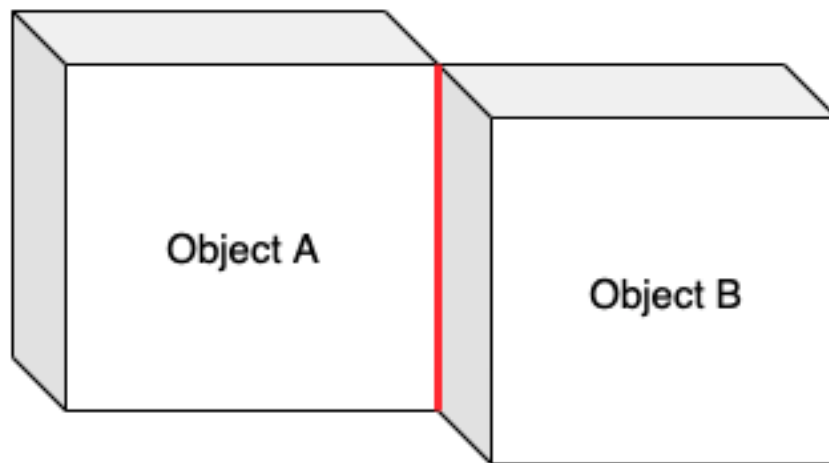


Figure 57. Non manifold shape. Red edge is incident to four faces

Another limitation is a special case that is encountered during testing and debugging the algorithm. If one of the vertices of an object is not completely inside the other object (Figure 58) the proposed algorithm cannot construct the intersection as a 3D polygonal mesh. Because the construction process begins with one of the vertices of an object that is completely inside the other object. Of course, this can be handled by increasing the ray-surface intersections, but this occurs very rarely in real world data and increasing the number of the ray-surface intersection tests decrease the performance of the algorithm hence not implemented.

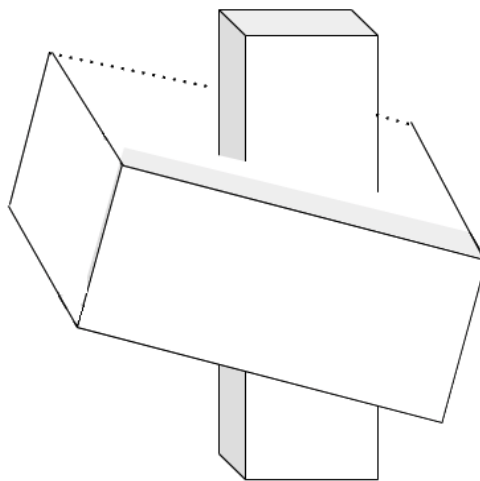


Figure 58. A special case none of the vertices of the objects are inside the other object

Calculation of the points where ray intersects with the surface requires high coordinate precision, hence, in a floating-point environment this situation has caused some degenerations that the 3D intersection algorithm must handle. While calculating intersections between rays and planes and determining new positions of vertices, a tolerance value must be predefined and vertices with small distances than this value to the plane must be accepted as intersected. In our implementation this tolerance value is 0.05 in meters.



3.5. Conclusion

In this chapter, 3D analyses, and query components of the framework have been designed and developed. To query and analyze the 3D geospatial data, data is stored in a NoSQL database called mongoDB. A new 3D intersection algorithm has been developed for 3D polygonal meshes. Using the 3D intersection algorithm under the hood, 3D Clip and 3D Difference analyzes also implemented. In addition to 3D intersection, 3D Buffer analysis has been implemented. The proposed algorithm tested with both synthetic and real-world data. Synthetic data is used for detecting special cases and degeneres, while real world data is used to test real world scenarios. Some limitations have been determined and reasons for them are explained. For the first time with the developed components, many decision making processes can be analyzed in 3D in a web environment using only a browser without any software installation.

One of the future works is to support more 3D analysis capabilities and also support more 3D representations such as Voxels. Also, to overcome limitations of the algorithm, a more complex data structure than HE will be implemented.

Additionally, many real-world scenarios will be analyzed using the developed framework such as detecting buildings outside the zonal plan in 3D.

4. CHAPTER 4 WEB-BASED PROCEDURAL MODELLING OF 3D MODELS

4.1. Introduction

With the rapid development in data acquisition methods and computer graphics, 3D City Models (3DCM) are widely used, mostly in the city planning industry. A literature review about 3DCMs and their applications can be found at (Biljecki et al., 2015). With the spread of open data policies, which is based on institutions to make their data publicly available (Donker and Leonen 2016), many institutions in Europe and the United States share 3DCMs as open data via web using web technologies HTML5 and WebGL. Rotterdam, Amsterdam, Delft, Berlin and New York are notable ones.

Most of the 3DCMs are found as LoD1 block models in practice, since LoD2 models are much more difficult to obtain because of the need for time consuming process and expensive data acquisition techniques (Biljecki and Dehbi, 2019). The minimum required level of detail in 3DCMs varies according to use cases (Biljecki et al, 2018). While LoD1 models are sufficient for some use cases such as shadow analysis, some use cases require general roof geometries such as solar analysis (Biljecki et al 2019 and Weiler et al 2019).

In this work, the possibility of constructing 3D models using 2D data such as building footprints and an aerial image has been investigated. It is very difficult to model building roofs from 2D building footprints without roof type information. To overcome this issue, deep learning techniques (DL) have been used to derive roof types from aerial imagery. There are four contributions in this work. First, a methodology to obtain roof type automatically from aerial images without any user input has been developed. Then roof geometries have been constructed using procedural modelling and only 2D data. The Straight Skeleton algorithm has been extended to be able to model roof surfaces. And finally the proposed methodology has been implemented as a web-based solution using web services.

DL, is a machine learning technique that a computer model learns to perform classification tasks directly from images, text, or sound. DL models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labelled data and neural network architectures that contain many layers (URL-

23). Thus, a DL model called CNN has been used to classify 3 main roof types with an aerial image in this work. After the classification, extracted roof type information has been used to construct roof geometries. The 3 main roof types are Hipped, Gable and Flat.

Procedural modelling (PM) can be used to generate 3DCMs as LoD2 (Tsiliakou et al., 2014; Martinovic, 2015). PM is an umbrella term which includes many modelling techniques such as L-Systems, fractals and shape grammars. All of these techniques aim to create multiple instance models via utilizing a set of predefined rules and algorithms. Batch modelling approach of PM minimizes user interaction and labour. To be able to construct roof geometries, PM technique has been used with roof type information that is derived via DL.

LoD1 block models have been generated via extrusion (Ledoux, H. and Meijers, M., 2011; Ohori, K., A., et al., 2015). Roof geometries have been constructed using Straight Skeleton Algorithm (Aichholzer et al., 1995) and Sweep Line Algorithm (Souvaine., 2005). Then roof geometries added to the top of block models. To be able to implement proposed methodology as a web-based solution, RESTful web services have been developed with web technologies and finally, constructed roof geometries and generated 3DCM have been visualized in browser via HTML5 and WebGL.

4.2. Background and Related Work

One of the widely used methods to generate 3DCM is extrusion from 2D footprints (Ledoux, H. and Meijers, M. 2011), (Arroyo, O., K. et al 2015). But 3DCMs generated by this method lack roof geometries. The lack of roof geometries hinders the widespread use of 3DCMs. While some 3D spatial analysis can be performed using LoD1 data such as shadow analysis, some analysis such as solar potential requires minimum LoD2 data (Biljecki et al 2019 and Weiler et al 2019).

Another popular method to generate 3DCM as LoD2 is using LIDAR point cloud data (Tomljenovic, I. et al 2015), (Bauchert J., P. and Lafarge F., 2019). Roof shape and building shell is the most valuable geometric information that can be extracted from point cloud data. However, this process requires additional data such as 2D building footprints and computationally intensive data process workflows to be able to classify and construct city objects.

Procedural modelling (PM) can be used to generate 3DCMs as LoD2 (Tsiliakou, E. et al 2014), (Martinovic, A., 2015). PM is an umbrella term which includes many modelling techniques such as L-Systems, fractals and shape grammars. All of these techniques aim to create multiple instance models via utilizing a set of predefined rules and algorithms. Batch modelling approach of PM which minimizes user interaction and labour, fits very well into GIS applications which deal with management of big data. To be able to generate 3DCMs as LoD2, PM technique requires roof type information.

To obtain roof type information some respectively new studies have been done based on Deep Learning. In practical terms, deep learning is a subfield of machine learning and functions in the same way but it has different capabilities. Unlike other machine learning (ML) techniques, deep learning models can learn the discrimination between classes in given dataset without feature extraction. But while doing this, deep learning techniques need more amounts of training data than ML techniques such as Support Vector Machine (SVM), Decision Trees, PCA. Deep learning techniques or more specifically CNNs are widely used for visual cognitive tasks as classification to derive discriminant functions between classes from images. Castagno and Atkins (2018) use CNNs to classify roof types for multicopter emergency landing site selection. First, they use polygon of building roof outline for cropping data from a LIDAR based DSM image. Then they use the polygon for cropping from the RGB image in the same way. After the preparation of data, they fuse RGB and LIDAR images as input to CNN. Patrovi et al. (2019) choose a deep learning based approach. They use LIDAR based DSM and pansharpened VHR (Very High Resolution) satellite images but their approach has drawbacks with non-rectilinear roof shapes. Axelson et al. (2018) uses CNNs with photogrammetric point clouds obtained from aerial images for classifying roof types. But they consider the roofs as only two classes: ridge and flat roofs. Biljecki and Dehbi (2019) has obtained roof types from LoD1 models with machine learning without using point clouds but they have not yet constructed roof geometries using this information. Also, they used the LoD1 city model, not 2D building footprints.

4.3. Methodology

4.3.1. Obtaining Roof Type Information

A Python script has been written to generate train images from aerial images. These images are served as open data and can be found on (Bradbury et al 2016). Geopandas library has been used to extract geometry and attributes from 2D footprints. With this script, multiple buildings have been cropped from an aerial image and saved as separate image files using building envelopes that are extracted from 2D building footprints. Using Microsoft Custom Vision Service (URL-24) that eases training process, train images are labeled as “Hipped”, “Gable”, “Flat” and “Negative” to train data. The “Negative” label is used when the image cannot be identified as hipped, gable or flat. A general workflow for the training process is shown in Figure 59.

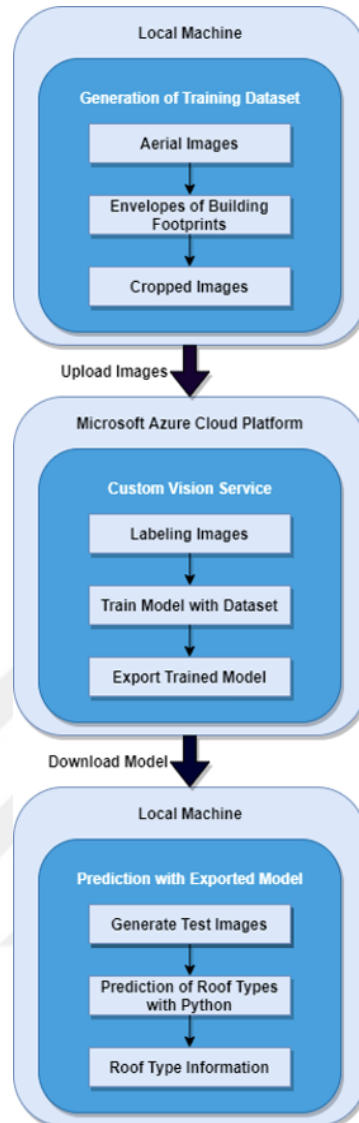


Figure 59. Training and Prediction Processes.

After the training process, the model has reached %89,9 accuracy for predicting roof type. The trained model exported from Custom Vision as a TensorFlow graph file to be used in a prediction process. A trained model can be saved as a model file and can be used in a different environment or application for prediction process. This is often called “transfer of learning” in literature. To be able to predict roof types a Python script has been coded. The model file has been parsed using TensorFlow (URL-25) and roof types have been predicted via this script automatically. A shapefile which consists of 2D building footprints has been enriched with roof types as an attribute. All the workflow for prediction of roof type can be found on a publicly available Github repository (URL-26) that includes train data, test data and scripts.

4.3.2. Extrusion and Block Model Generation

A 3D model of the city can be obtained via extrusion of the building footprints. For this purpose, normal vectors of the building footprints are calculated. Normal vector is a unit vector perpendicular to the surface (Figure 50). Extracting P1 from P2 and P2 from P3 vectors V1 and V2 are obtained. Using the cross product of these vectors, surface normal is calculated.

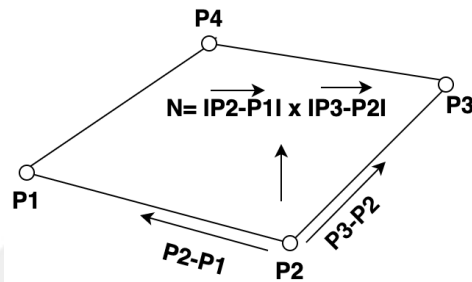


Figure 60. Calculation of the surface normal

After the calculation of surface normals, using the “height” attribute of the building footprints, building footprints are translated along surface normals and top surfaces are constructed. By indexing the top and footprint points, vertical surfaces are constructed and LOD1 block model is derived. (Figure 61).

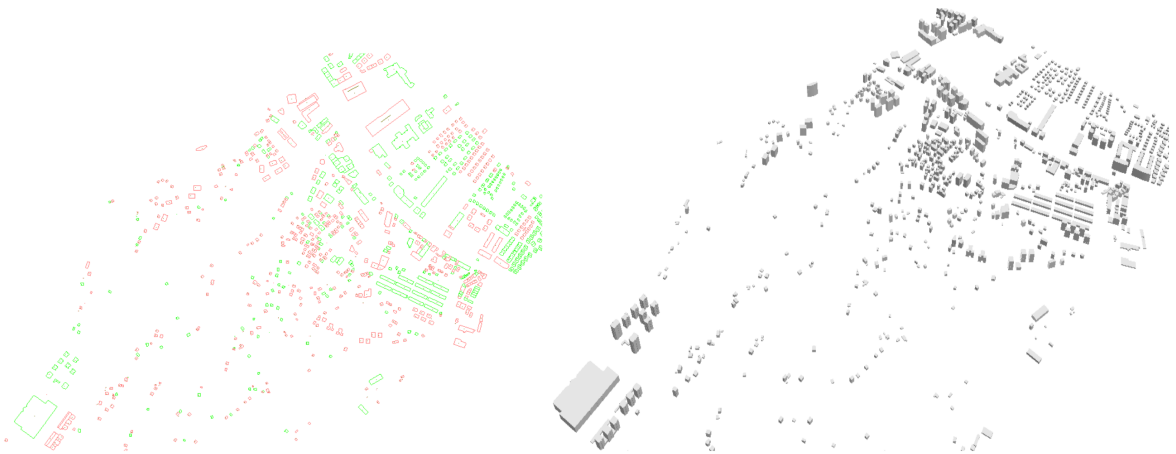


Figure 61. 2D building footprints (left) 3D block model (right).

There are condominium unit plans (CUP) drawn by geomatic engineers that have boundaries of condominiums as 2D (Figure 62).

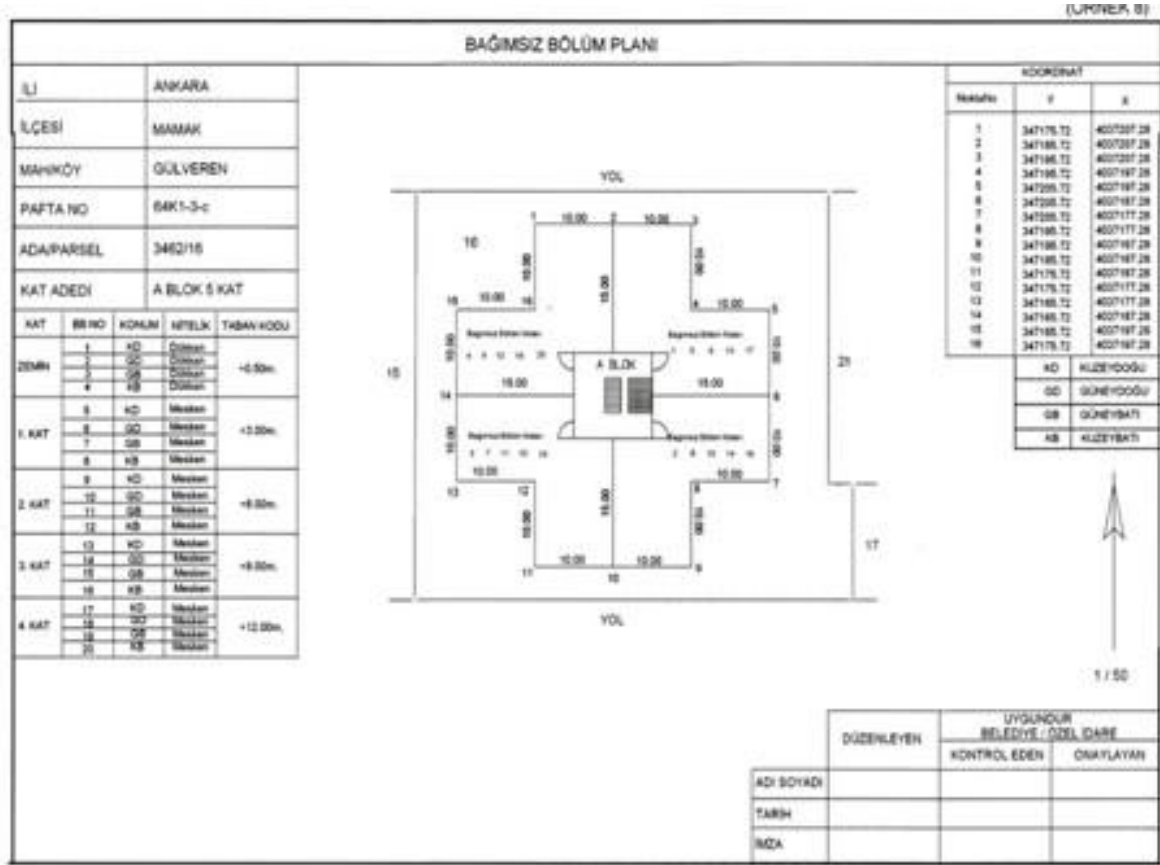


Figure 62. Condominium unit plan (Çağdaş, 2012).

If CUPs are provided, our modeler can model condominium units as well at city scale. For this purpose each condominium is translated along the surface normal according to the height of the condominium in CUP. Then translated polygons are extruded and condominiums are modelled procedurally (Figure 63).



Figure 63. Polygons (left) multiplied and translated (middle) condominium unit in 3D.

Geometries of the condominium units are stored as CityJSON. Since condominium units are not defined in the CityGML data model, an ADE is developed for storing condominium units Figure 64.

```

{
  "type": "CityJSON",
  "version": "0.9",
  "metadata": {
    "geographicalExtent": [
      300060.181,
      5040722.785,
      8.444,
      301073.701,
      5042282.343,
      129.092
    ]
  },
  "CityObjects": {
    "B-206391182646-3B58D3D11C14": {
      "type": "Building",
      "condominiums": [
        {
          "id": "C-206391182646-3B58D3D11C14",
          "type": "MultiSurface",
          "boundaries": [[0,1,2], [0,2,3], [2,3,4], ...]],
          "geometry": [
            {
              "type": "MultiSurface",
              "boundaries": [
                [
                  [
                    30203,
                    30204,
                    30205
                  ],
                  [
                    30205,
                    30206,
                    30207
                  ]
                ]
              ]
            }
          ]
        }
      ]
    }
  }
}

```

Figure 64. 3D condominiums stored in the CityJSON file.

4.3.3. Construction of the Roof Geometries

In order to model roof geometries, a straight skeleton algorithm has been used. The definition of straight skeleton (SS) is also its construction. SS is a geometric construct that consists of only straight edges, as a result of the shrinking process of a polygon (Figure 65).

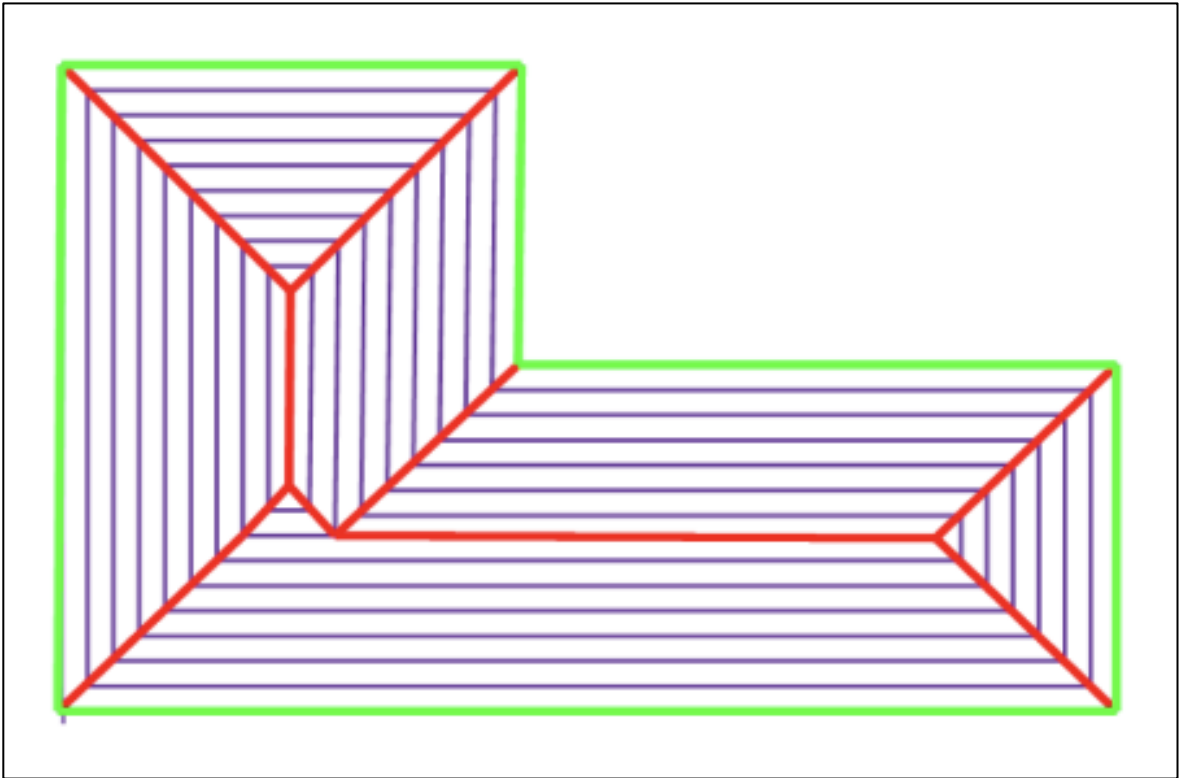


Figure 65. Input polygon (green), wave fronts (purple lines) and SS (red lines).

In this shrinking process, each edge of the polygon moves inwards of the polygon in a self-parallel manner. Two events change the topology of the input polygon. Edge Event occurs when an edge shrinks to zero and creates a node in skeleton Figure 66 and Split Event occurs when a reflex vertex touches to a non-consecutive edge and creates a node in skeleton Figure 67.

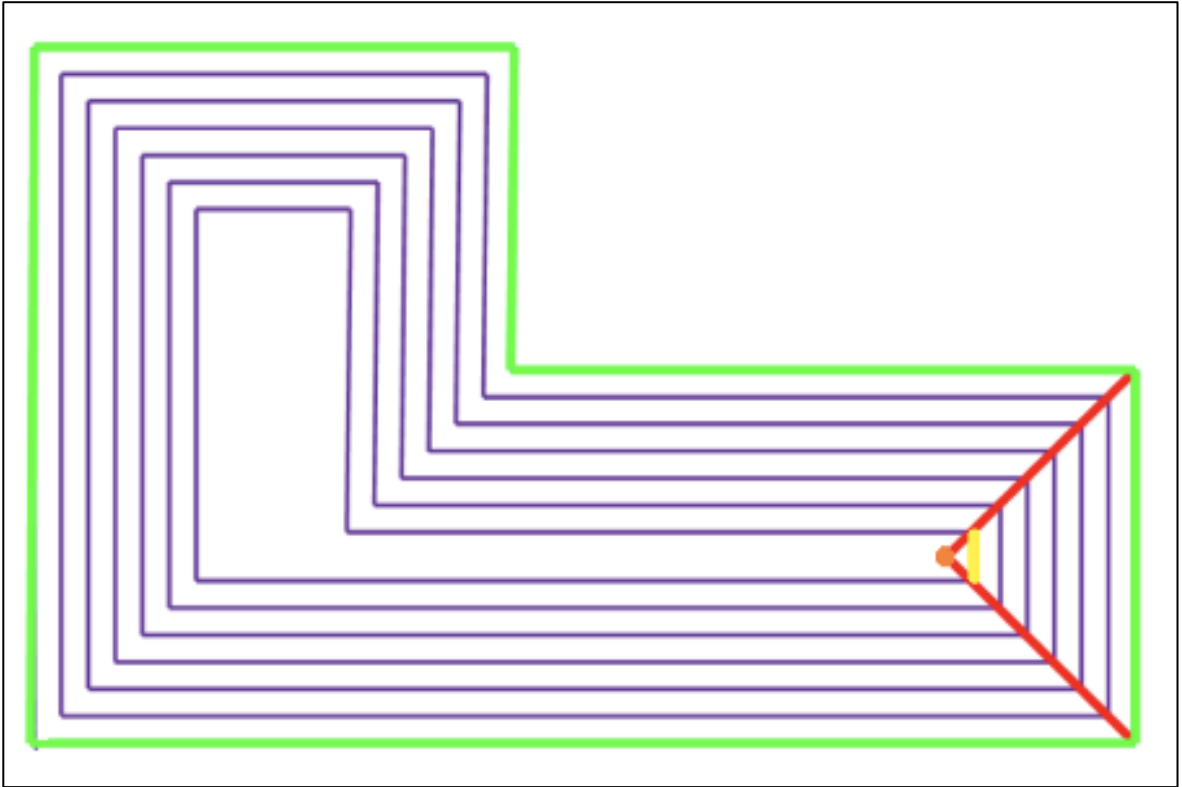


Figure 66. Edge Event (Yellow edge shrinks to zero at the orange point)

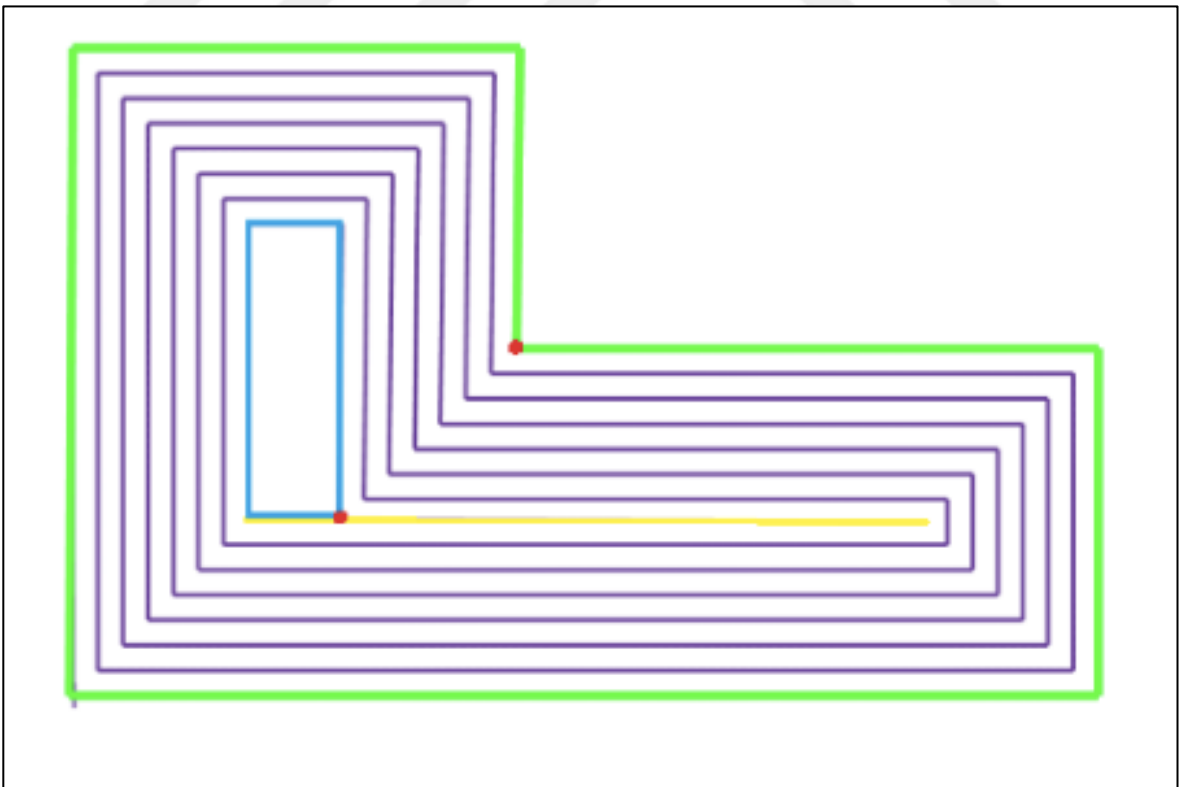


Figure 67. Split Event (yellow edge meets red reflex vertex and this creates blue polygon).

To stimulate this shrinking process, there are two approaches. The 2D construction method is based on angular bisectors (Felkel and Obdrzalek, 1998) and the 3D construction method is based on sweep planes firstly introduced by (Eppstein and Erickson 1999) and used by (Kelly 2015).

In the bisector-based approach, in order to create roof geometries in 3D, the skeleton must be traversed and Z values of nodes that are created by the skeleton must be manipulated according to proper roof height. Since sweep plane approach does not require this additional step, sweep plane approach has been used in this work.

In sweep plane approach, all nodes of the skeleton are detected by intersection of planes. Edge events have been detected by collision of three consecutive direction planes with sweep plane and split events have been detected by collision of two consecutive and one non-consecutive direction planes with sweep plane.

In the proposed algorithm which takes polygons as input, vertices of polygons stored in circular doubly linked list, every vertice therefore, has pointers to next and previous vertices and also next and previous direction planes. Sweep plane, which is a plane parallel to the input polygon, moves in the direction of the Z axis and the polygon shrinks. Edge events and split events are stored in a priority queue according to the height of sweep plane which gives the order of nodes in SS. When every edge of the input polygon shrinks to zero algorithm finishes and SS completed.

The result of SS is a directed graph. In order to generate roofs, roof surfaces must be generated from this graph. To accomplish this, a 3D sweep line algorithm has been implemented. In this algorithm, first, nodes of SS is grouped according to original edges of input polygon (Orange points in Figure 5 for green edge). Observe that, a node is can be related more than one group. For instance in Figure. 5, node 2 in both in the group of edges E1 and E2. Then, for every group, a sweep line that is perpendicular to the related edge, moves from start to the end of the edge. Intersection order of nodes with this sweep line, determines the order of the points for the roof polygon (Figure 68).

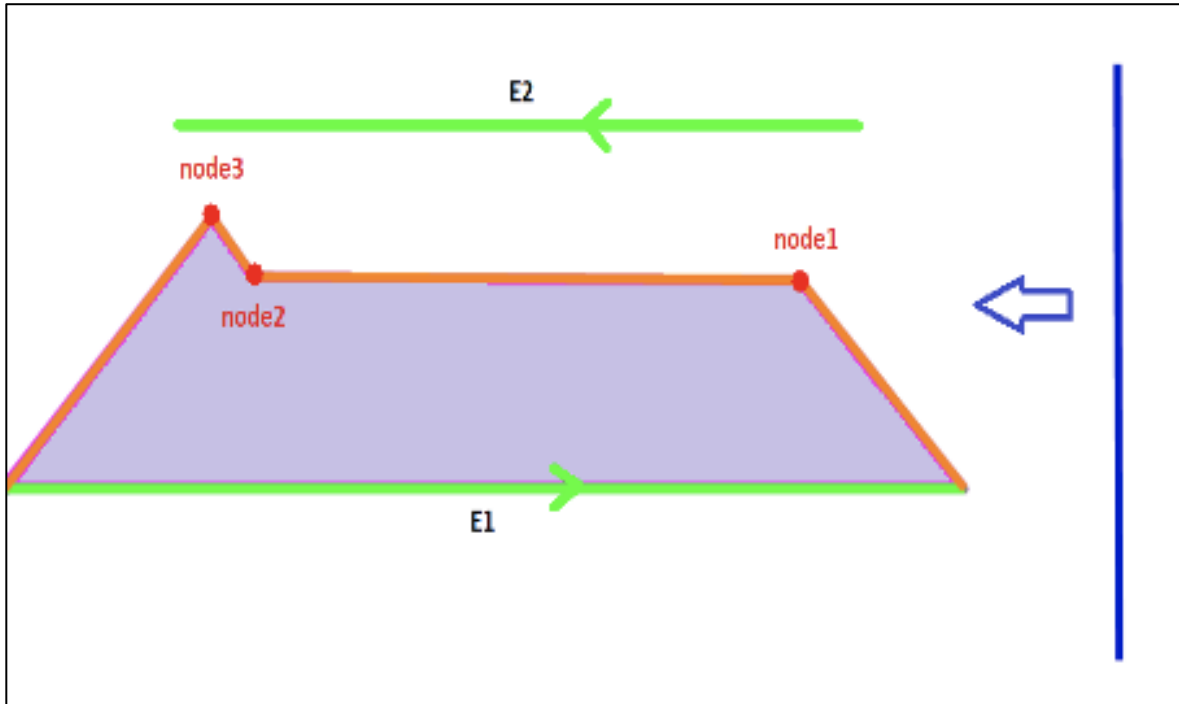


Figure 68. Sweep Line Process.

SS and sweep line have been implemented via web services as a web-based solution using RESTful architecture. For each roof type there are two web services. One service is for convex polygons and the other is for non-convex polygons. Web application consists of 3 sub modules: storage module, process module and web server module. General system architecture has been given below (Figure 69). Application has been deployed on an Amazon S3 Bucket cloud storage environment. 2D building footprints can be upload to a MongoDB instance on the bucket via a rest service that has been developed using Java Jersey web framework.

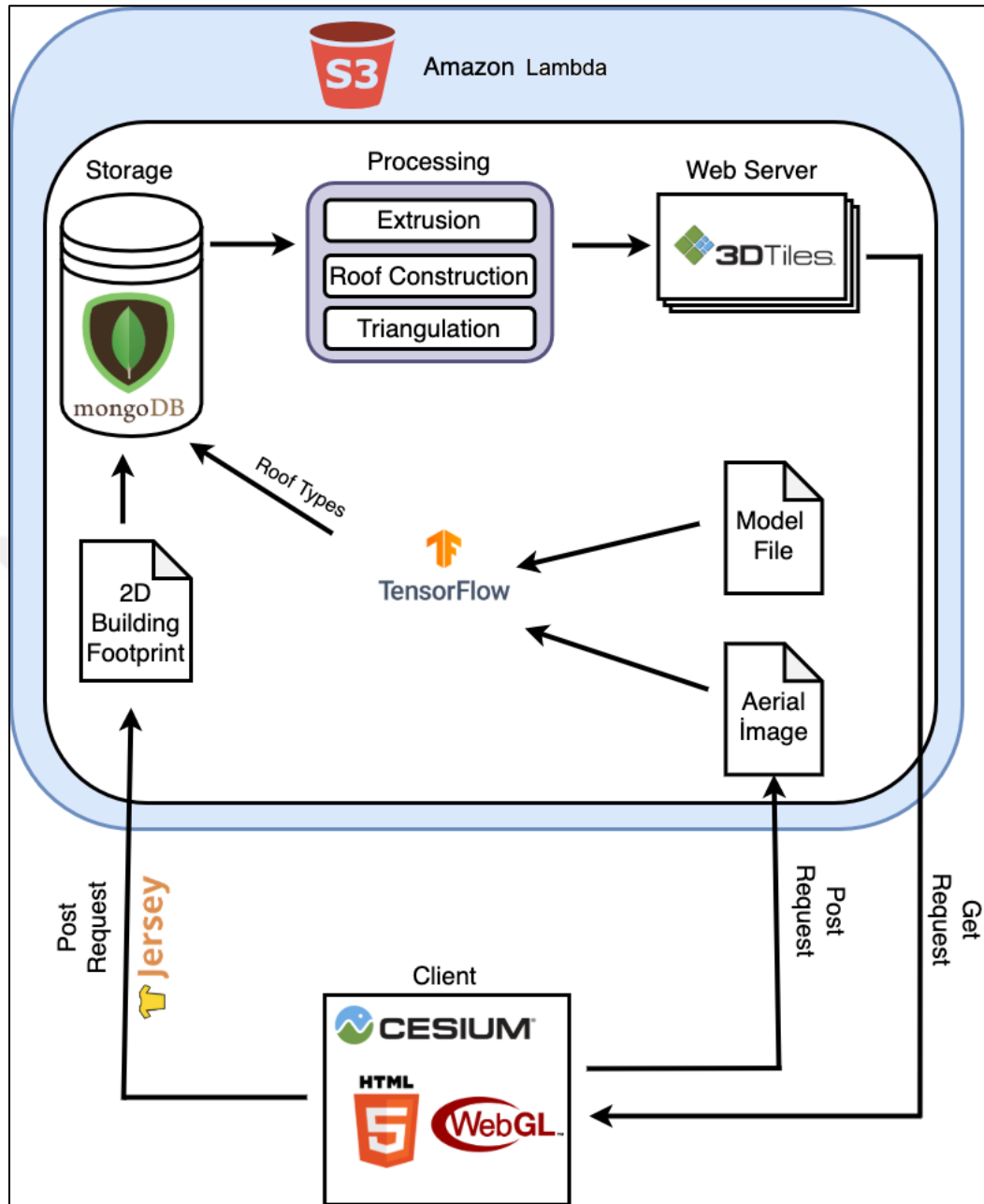


Figure 69. General System Architecture.

4.3.4. Visualization of the 3DCM via Browser

3DCMs are huge in size, to be able to visualize them via browsers, tiling approaches has been used with web technologies such as HTML5 and WebGL. 3D roof geometries are added to 3D models of the tiles thus, generated roofs are integrated to the tiling system that was described in Chapter 2. Thus, only needed tiles based on user's current view in the scene are fetched from server.

Since WebGL supports only triangles as primitives for representation of surfaces, 3D polygons must be triangulated. An “Ear Clipping” algorithm has been used for this purpose and 3D polygon surfaces that belongs to roof geometries have been triangulated.

After generation of tileset, in order to visualize tileset in browser open source javascript library Cesium.js has been used in this study. Since Cesium.js is built on WebGL, 3DCM has been rendered using client’s GPU and without any additional plug-in (Figure 70).

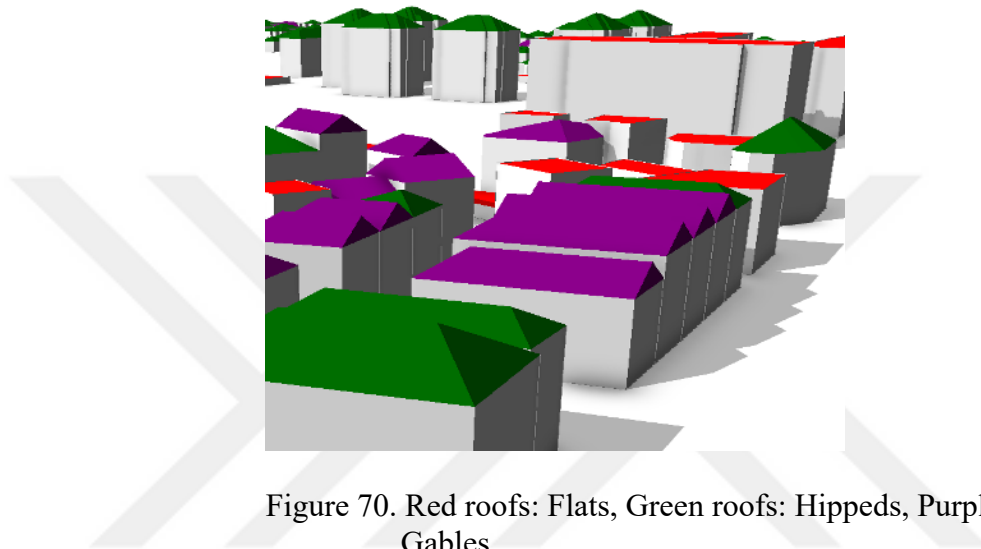


Figure 70. Red roofs: Flats, Green roofs: Hippeds, Purple roofs: Gables.

4.4. Findings and Discussion

4.4.1. Performance Metrics of the Roof Type Prediction

3423 roof type images that belong to hip, gable and flat roofs have been used as training data. Flat 751, gable 1762, hip 835 The accuracy for predicting roof types is 0.89,9. Performance metrics are represented as AP, Recall and Precision (Figure 71). The overall average precision (AP) value as 0.89 is highly acceptable and proves the training is enough for prediction.

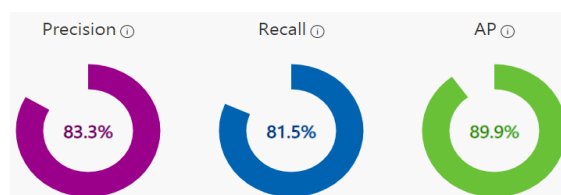


Figure 71. Performance metrics of training process.

The accuracy is higher than (Biljecki and Dehbi 2019). The main reason for this result is that in this work only 3 roof types has been classified. In (Biljecki and Dehbi 2019) 6 different roof types has been classified. Also, to obtain roof types, aerial image has been used in this work but LoD1 3DCM in theirs. The lack of aerial imagery reduces the overall performance especially between similar types such as hipped and gable. The accuracy of predicting for individual roof types has been given as confusion matrix (Table 8).

Table 8. Confusion matrix of prediction process

	Flat	Gable	Hipped	Negative
Flat	53	5	4	0
Gable	8	96	10	0
Hipped	1	14	143	0
Negat.	1	10	2	14
Total	63	125	159	14

And comparison of predicted results with ground truth values (Table 9). Precisions of classes are not directly proportional with counts of images that used to train the model. Because, quality of the images also effects the results, not only count of the images.

Table 9. Performance metrics of prediction process

Class	Ground Truth	Prediction	Accuracy	Precision	Recall
Flat	63	62	94,74%	0,85	0,84
Gable	125	114	86,98%	0,84	0,77
Hipped	159	158	91,41%	0,91	0,9
Negativ.	14	27	96,40%	0,52	1

4.4.2. Special Cases and Floating-Point Issues

Implementing SS which requires high coordinate precision in a floating point environment has been caused some degenerations that construction algorithm must be handle. While calculating intersections between planes and determining new positions of vertices, some vertices that must have exact same coordinates can have different coordinates with minimal difference due to the floating point arithmetic. To be able to handle this situation, a tolerance value must be predefined and vertices with small coordinate differences with this value must be union into one vertex. If these situations are not handled, these small differences can lead to big changes in roof geometries (Figure 72).

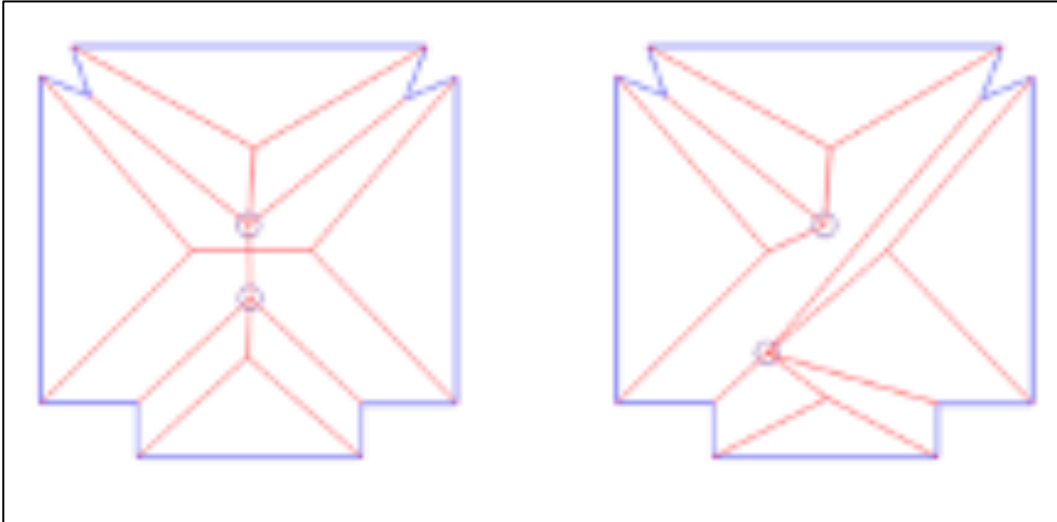


Figure 72. Degeneration. Two Edge Events must meet on the same point (left) but not properly they do not meet (right).

4.5. Conclusion

In this work, roof type information has been obtained from aerial image with deep learning and using this information, roof geometries have been constructed procedurally without any human intervention. A general workflow has been proposed to generate LOD2 3D models from 2D datasets.

This work indicates that web-based generation of useful 3D models as LOD2 from 2D datasets which are already derived by local governments can be a beneficial solution. All pipeline and process can be done using only web-based technologies, open-source software components.

In this proposed pipeline for automatic modelling of roof geometries, in the time of writing paper, limited to 3 class as hip, gable and flat. Hence, there is an on-going work on this topic and this work will be extended to include modeling other roof types as well such as pyramid, shed etc. Also, it is aimed to detect and model chimneys on the roofs as well.

5. REFERENCES

- Achere, S., Glas, H., Beullens, J., Druyter, G., Wulf, A., and Maeyer, P. 2016. Development of A 3D Dynamic Flood Web GIS Visualization Tool. International Journal of Safety and Security Engineering, 6, 3, 560-569.
- Anvandning, O., 2010. On the Use Of OpenGL ES 2.0 Shaders for Mobile Phones Using Cross Platform Middle-Ware., Semantic Scholar, 2010.
- Axelsson, M., Soderman, U., Berg, A., and Lithen, T., 2018. Roof Type Classification Using Deep Convolutional Neural Networks on Low Resolution Photogrammetric Point Clouds From Aerial Imagery. In 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 1293-1297.
- Biljecki, F., Ledoux, H. and Stoter, J., 2017. Generating 3D City Models without Elevation Data, Computers, Environment, and Urban Systems, 64, 1-18.
- Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S., and Çöltekin, A., 2015. Applications of 3D City Models: State of Art Review ISPRS International Journal of Geo-Information, 4, 2842-2889.
- Bishop, C. M., 2006. Pattern recognition and machine learning. springer.
- Bradbury, K., Brigman, B., Collins, L., Johnson, T., Lin, S. and Newell, R. 2016. Atlanta, Georgia - Aerial imagery object identification dataset for building and road detection and building height estimation. Figshare. Collection.
- Castagno, J., and Atkins, E. M. Automatic classification of roof shapes for multicopter emergency landing site selection, Aviation Technology, Integration, and Operations Conference, June 2018, Atlanta, Proceedings, 3977.
- Chaturvedi, K., Willenborg, B., Sindram, M., and Kolbe TH., 2017. Solar Potential analysis and Integration of Time-Dependent Simulation Results for Semantic 3D City Models Using Dynamizers. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, IV-4/W5, 25-32
- Chaturvedi, K., and Kolbe TH., 2017. Future City Pilot 1 Engineering Report. Lehrstuhl für Geoinformatik.
- Chaturvedi, K., Yao, Z., and Kolbe TH., 2015. Web-based Exploration of and Interaction with Large and Deeply Structured Semantic 3D City Models using HTML5 and WebGL. Bridging Scales, March 2015, Köln, Conference Paper in Proceedings 296-306.
- Chen, W., HE, B., Zhang, L., and Nver D. 2016. Developing An Integrated 2D and 3D WebGIS-Based Platform for Effective Landslide Hazard Management. International Journal of Disaster Risk Reduction 20, 26-38.

- Doyle, A. and Cuthbert, A., 1998. OpenGIS Project Document 98-061: Essential Model of Interactive Portrayal, 1998.
- Evans, A., Romeo, M., Bahrehmand, A., Agenjo, J., and Blat, J., 2014. 3D Graphics On the Web: A Survey, Computers and Graphics. 41. 43-61.
- Gaillard, J, Peytavie, A., and Gesquiere G., 2020. Visualization and Personalization of Multi-Representations City Models. International Journal of Digital Earth. Vol 13. No.5. P.627-644. 2020.
- Gaillard, J., Vienne, A., Baume, R., Pedrinis, F., and Peytavie, A. 2015. Urban data visualisation in a web browser. Web3D 2015, Jun 2015, Heraklion, Greece. pp.81-88, Proceedings of the 20th International Conference on 3D Web Technology.
- Gesquiere, G., and Manin, A. 2012. 3D Visualization of Urban Data Based on CityGML with WebGL. International Journal of 3-D Information Modeling (IJ3DIM), 1(3), pp. 1-15.
- Gutbell, R., Pandikow, L., Coors, V., and Kammeyer, Y., 2016. A Framework for Server Side Rendering Using OGC's 3D Portrayal Service. Web3D 2016. July 2016. California.
- Guttman, A., 1984. R-Trees: A Dynamic Index Structure for Spatial Searching. SIGMOD'84. Boston.
- Hagedorn, B., 2010. Web View Service Discussion Paper, Version 0.3.0. 2010.
- Hildebrandt, D., 2014. A Software Reference Architecture for Service-Oriented 3D Geovisualization Systems. ISPRS International Journal of Geo-Information 3 4, pp 1445-1490. 2014.
- Jaillet, V., 2020. 3D, Temporal and Documented Cities: Formalization, Visualization and Navigation. PhD Thesis. University of Lyon. 2020.
- Jankowski, J., Ressler, S., Jung, Y., Behr, J., and Slusallek, P. Declarative Integration of 3D Graphics into the World Wide Web: Principles, Current Approaches, and Research Agenda. Proceedings of the 18th International conference on 3D Web Technology (Web3D'13), 2013. P. 39-45.
- Kilsedar, CE., Fissore, F., Pirotti, F., and Brovelli, MA., 2019. Extraction and Visualization of 3D Building Models in Urban Areas for Flood Simulation. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XLII-2/W11, 2019.
- Klimke, J., 2019. Web-Based Provisioning and Application of Large-Scale Virtual 3D City Models. PhD Thesis. Hasso Plattner institute, Postdam University, 2019.

- Koukofikis, A., Coors, V. and Gutbell, R., 2018. Interoperable Visualization of 3D City Models Using OGC's Standard 3D Portrayal Service. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume IV-4
- Kramer, M, and Gutbell R., 2015. A case study on 3D geospatial applications in the Web using state-of-the-art WebGL frameworks. Web3D 2015, Jun 2015, Heraklion. Greece.
- Kutzner, T., and Kolbe, TH., 2018. CityGML 3.0: Sneak Preview. PFGK18-Photogrammetrie-Fenerkundung-Geoinformatik-Kartographie, 37. Jahrestagung in München 2018, 835-839.
- Kutner, T., Chaturvedi, K., and Kolbe, TH., 2020. CityGML 3.0: New Functions Open Up New Applications. Journal of Photogrammetry, Remote Sensing and Geoinformation Science. 88. P 43-61. 2020.
- Ledoux, H. and Meijers, M., 2011. *Topologically Consistent 3D City Models Obtained By Extrusion*. International Journal of Geographical Information Science. 25, 4, 557-574.
- Ledoux, H., Ohori KA., Kumar, K., Dukai, B., Labetski, A., and Vitalis S., 2019. CityJSON: A Compact and Easy to Use Encoding of the CityGML Data Model. Open Geospatial Data, Software and Standards. 2019.
- Li, B., Wu, J., Pan, M., and Huang, J. 2015 Application of 3D WebGIS and Real-Time Technique in Earthquake Information Publishing and Visualization. Earthquake Science, 28, 223-231.
- Lu, M., Wang, X., Liu, X., Chen, M., Bi, S., Zhang, Y., and Lao, T., 2020. Web-Based Real-Time Visualization of Large-Scale Weather Radar Data Using 3D Tiles. Transactions in GIS.
- Martinovic, A. Inverse Procedural Modeling of Buildings. Ph.D. Thesis, KU Leuven, Leuven, ' Belgium, 2015.
- OGC, CityGML 2.0. OGC CityGML Encoding Standard. 12-019.
- Ohori, K., A., Ledoux, H. and Stoter, J., 2015. A Dimension Independent Extrusion Algorithm Using Generalised Maps. International Journal of Geographic Information Science. 29, 7, 1166-1186.
- Parisi, T., 2014. Programming 3D Applications with HTML5 and WebGL: 3D Animation and Visualization for Web Pages 1st Edition O'Reilly, United States of America, 574 p.
- Partovi, T., Fraundorfer, F., Bahmanyar, R., Huang, H., & Reinartz, P., 2019. Automatic 3-D Building Model Reconstruction from Very High Resolution Stereo Satellite Imagery. Remote Sensing, 11(14), 1660.

- Pispidikis, I and Dimopoulou, E. 2016. Development of a 3D WebGIS System for Retrieving and Visualizing CityGML Data Based on their Geometric and Semantic Characteristics by Using Free and Open Source Technology. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume IV-2/W1, 2016, pp.47-53
- Schilling, A., and Kolbe TH., 2010. Draft for OpenGIS Web 3D Service Implementation Standard Version 0.4.0. 2010.
- Shverin, J, Risetto, H., Remondio, F., Agugario, G., and Girardi, G. 2013. The MayaArch3D Project: A 3D WebGIS for Analyzing Ancient Architecture and Landscapes. Literary and Linguistic Computing. 28, 736-753.
- Taraldsvik, M., 2011. Exploring the Future: is HTML5 the solution for GIS Applications on the World Wide Web?. Technical Report, NTNU 2011.
- TMS, 2010. Tile Map Service Implementation Standard, 2010.
- Tomljenovic, I.; Höfle, B.; Tiede, D.; Blaschke, T. 2015. Building extraction from airborne laser scanning data: An analysis of the state of the art. Remote Sensing., 7, 3826–38620
- Tsiliakou, E.; Labropoulos, T., Dimopoulou, E. Procedural modeling in 3D GIS environment. International. Journal of 3-D Information Modelling. 2014, 3, 17–34.
- Xiaoqing, Z., Jixin L., and Yonghua, X., 2010. Architecture and Application of 3D WebGIS based on Skyline and ArcGIS. 2nd International Conference on Computer Engineering and Technology. 2010
- URL-1, <https://tr.wikipedia.org/wiki/HTML5>. Accessed on 15 May 2021.
- URL-2, Cortana 3D. <http://www.cortona3d.com/de/cortona3dviewer>. Accessed on 19 May 2021.
- URL-3, FreeWRL. <http://freewrl.sourceforge.net/>. Accessed on 19 May 2021.
- URL-4, XNavigator. <http://xnavigator.sourceforge.net/doku.php>. Accessed on 20 May 2021.
- URL-5, Nasa World Wind. <https://worldwind.arc.nasa.gov/>. Accessed on 20 May 2021.
- URL-6, Google Earth. <https://www.google.com/intl/de/earth/desktop/>. Accessed on 20 May 2021.
- URL-7, GeoPortail. <https://www.geoportail.gouv.fr/>. Accessed on 21 May 2021.

URL-8, 3D Macau. <http://www.3dmacau.com/>. Accessed on 21 May 2021.

URL-9, Open3DGIS. www.open3dgis.org. Accessed on 21 May 2021.

URL-10, Vulkan. <https://www.khronos.org/vulkan/>. Accessed on 23 May 2021.

URL-11, DirectX12. <https://tr.wikipedia.org/wiki/DirectX>. Accessed on 23 May 2021.

URL-12, Metal. [https://en.wikipedia.org/wiki/Metal_\(API\)](https://en.wikipedia.org/wiki/Metal_(API)). Accessed on 23 May 2021.

URL-13, W3C GPU for The Web Community Group. Accessed on 23 May 2021.

URL-14, Web 3D Consortium. <https://www.web3d.org/>. Accessed on 24 May 2021.

URL-15, Indexed 3D Scene Specification. <https://github.com/Esri/i3s-spec>. Accessed 04 June 2021.

URL-16, OGC 3D Tiles Standard. <https://github.com/CesiumGS/3d-tiles>. Accessed on 04 June 2021.

URL-17, obj23dtiles. <https://github.com/PrincessGod/objTo3d-tiles>. Accessed on 05 June 2021.

URL-18, citygml23dtiles. <https://github.com/njam/citygml-to-3dtiles>. Accessed on 06 June 2021.

URL-19, 3D Tiles Overview. <https://github.com/CesiumGS/3d-tiles/blob/main/3d-tiles-overview.pdf>. Accessed on 02 March 2020.

URL-20, Quantized Mesh Specification. <https://github.com/CesiumGS/quantized-mesh>. Accessed on 06 July 2020.

URL-21, Horizon Culling. <https://cesium.com/blog/2013/04/25/horizon-culling/>. Accessed on 06 July 2020.

URL-22, Computing the Horizon Occlusion Point. <https://cesium.com/blog/2013/05/09/computing-the-horizon-occlusion-point/>. Accessed on 06 July 2020.

URL-23, Machine Learning vs Deep Learning, <https://medium.com/@mail2princeyadav/machine-learning-vs-deep-learning-b5c5a4fc5c>. Accessed on 07 July 2021.

URL-24, Microsoft Custom Vision, <https://www.customvision.ai>. Accessed on 07 July 2021.

URL-25, Tensor Flow, <https://www.tensorflow.org>. Accessed on 08 July 2021.

URL-26, <https://github.com/alpertungakin/Roof-Type-Classification>. Accessed on 09 July 2021

Weiler, V., Stave, J. and Eicker, U., 2019. Renewable Energy Generation Scenarios Using 3D Urban Modeling Tools—Methodology for Heat Pump and Co-Generation Systems with Case Study Application. *Energies* 12 (3), pp. 403.

Wendel, J., Murshed, S.M., Sriramulu, A., Nichersu, A., 2016. Development of a Web-Browser Based Interface for 3D Data—A Case Study of a Plug-in Free Approach for Visualizing Energy Modelling Results, *Progress in Cartography*. Springer International Publishing, Cham, pp. 185–205.

Willenborg, B., 2015. Simulation of explosions in urban space and result analysis based on CityGML-City Models and a cloud-based 3D-Webclient. Master Thesis. Technical University Munich Chair of Geoinformatics. 2015.

Xu, P., Chang, X., Ren P., Ling, G., Chen, X., and Hauptmann A., 2020. A Survey of Scene Graph: Generation and Application. *IEEE Transactions on Neural Networks and Learning Systems*.

3DPIE, 2012. OGC 3D Portrayal Interoperability Experiment Final Report.

3DPS, 2017. OGC 3D Portrayal Service Document.

CURRICULUM VITAE

He graduated from Akçaabat Anatolian High School in 2005. He received his B.Sc. degree from the Department of Geomatic Engineering, Yıldız Technical University in 2010. He received his M.Sc. degree from the Graduate School of Natural and Applied Sciences in 2014 from Karadeniz Technical University. He has been working as a research assistant at the same Graduate School since 2012. His main research topics include 3D GIS, 3D Computer Graphics, 3D Computational Geometry and 3D WebGIS.

