# IMPLEMENTATION AND WEB-BASED VISUALIZATION OF 3D CITY MODELS

# 3 BOYUTLU ŞEHİR MODELİ OLUŞTURMA VE WEB TABANLI GÖRSELLEŞTİRME

**MEHMET BÜYÜKDEMİRCİOĞLU**

**Assoc.Prof.Dr. SULTAN KOCAMAN**

**Supervisor**

Submitted to Graduate School of Science and
Engineering of Hacettepe University
As a Partial Fulfillment to the Requirements
For the Award of the Degree Master of Science
in Geomatics Engineering

May 2018

This work named "**IMPLEMENTATION AND WEB-BASED VISUALIZATION OF 3D CITY MODELS**" by **MEHMET BÜYÜKDEMİRCİOĞLU** has been approved as a thesis for the Degree of **MASTER OF SCIENCE IN GEOMATICS ENGINEERING** by the below mentioned Examining Committee Members.

Prof.Dr.M. Tekin YÜRÜR
Head ........................................

Assoc.Prof.Dr.Sultan KOCAMAN
Supervisor ........................................

Prof.Dr.Mustafa ŞAHMARAN
Member ........................................

Assoc.Prof.Dr.Cevdet Coşkun AYDIN
Member ........................................

Assoc.Prof.Dr.Ümit IŞIKDAĞ
Member ........................................

This thesis has been approved as a thesis for the Degree of **MASTER OF SCIENCE IN GEOMATICS ENGINEERING** by Board of Directors of the Institute for Graduate Studies in Science and Engineering.

Prof.Dr.Menemşe GÜMÜŞDERELİOĞLU
Director of the Insitute of
Graduate Studies of Science and Engineering

# YAYINLAMA VE FİKRİ MÜLKİYET HAKLARI BEYANI

Enstitü tarafından onaylanan lisansüstü tezimin/raporumun tamamını veya herhangi bir kısmını, basılı (kağıt) ve elektronik formatta arşivleme ve aşağıda verilen koşullarla kullanıma açma iznini Hacettepe üniversitesine verdiğimi bildiririm. Bu izinle Üniversiteye verilen kullanım hakları dışındaki tüm fikri mülkiyet haklarım bende kalacak, tezimin tamamının ya da bir bölümünün gelecekteki çalışmalarda (makale, kitap, lisans ve patent vb.) kullanım hakları bana ait olacaktır.

Tezin kendi orijinal çalışmam olduğunu, başkalarının haklarını ihlal etmediğimi ve tezimin tek yetkili sahibi olduğumu beyan ve taahhüt ederim. Tezimde yer alan telif hakkı bulunan ve sahiplerinden yazılı izin alınarak kullanması zorunlu metinlerin yazılı izin alarak kullandığımı ve istenildiğinde suretlerini Üniversiteye teslim etmeyi taahhüt ederim.

☒ **Tezimin/Raporumun tamamı dünya çapında erişime açılabilir ve bir kısmı veya tamamının fotokopisi alınabilir.**
(Bu seçenekle teziniz arama motorlarında indekslenebilecek, daha sonra tezinizin erişim statüsünün değiştirilmesini talep etseniz ve kütüphane bu talebinizi yerine getirse bile, tezinin arama motorlarının önbelleklerinde kalmaya devam edebilecektir.)

☐ **Tezimin/Raporumun ………….. tarihine kadar erişime açılmasını ve fotokopi alınmasını (İç Kapak, Özet, İçindekiler ve Kaynakça hariç) istemiyorum.**
(Bu sürenin sonunda uzatma için başvuruda bulunmadığım taktirde, tezimin/raporumun tamamı her yerden erişime açılabilir, kaynak gösterilmek şartıyla bir kısmı ve ya tamamının fotokopisi alınabilir)

☐ **Tezimin/Raporumun ………….. tarihine kadar erişime açılmasını istemiyorum, ancak kaynak gösterilmek şartıyla bir kısmı veya tamamının fotokopisinin alınmasını onaylıyorum.**

☐ **Serbest Seçenek/Yazarın Seçimi**

28/05/2018

*Mehmet BÜYÜKDEMİRCİOĞLU*

# ETHICS

In this thesis study, prepared in accordance with the spelling rules of Institute of Graduate Studies in Science of Hacettepe University,

I declare that

- all the information and documents have been obtained in the base of the academic rules
- all audio-visual and written information and results have been presented according to the rules of scientific ethics
- in case of using others Works, related studies have been cited in accordance with the scientific standards
- all cited studies have been fully referenced
- I did not do any distortion in the data set
- and any part of this thesis has not been presented as another thesis study at this or any other university.

28/05/2018

MEHMET BÜYÜKDEMİRCİOĞLU

# ABSTRACT

# IMPLEMENTATION AND WEB-BASED VISUALIZATION OF 3D CITY MODELS

**Mehmet BÜYÜKDEMİRCİOĞLU**

**Master of Science, Department of Geomatics Engineering**

**Supervisor: Assoc.Prof.Dr.Sultan KOCAMAN**

**May 2018 , 79 pages**

3D city models have become crucial for a better city management, and can be used for various purposes such as disaster management, navigation, solar potential computation and planning simulations.3D city models are not only visual models, and thematic queries and analyzes can be performed with the help of semantic data.They can be produced using different data sources and methods.

In this thesis, basemaps and large-format aerial images, which are produced in accordance with the regulations of large-scale map production in Turkey (called BOHYY) have been used to develop a workflow for semi-automatic 3D city model generation. The aim of this study is to generate a 3D city model from existing aerial photogrammetric datasets without any additional data acquisition or costly manual processing. During the prodution, open-source software or other tools with free educational/research licences is preferred.

As a result, a 3D city model has been generated with semi-automatic methods at LoD2 of CityGML using the data of study area near Çesme / Izmir. The generated model is automatically textured and visualized on the web along with the attribute information and digital terrain model. The model is published on the web at www.cesme3d.com. The problems encountered throughout the study and approaches to solve them are presented here.

**KEYWORDS:** 3D City Model, Visualisation, DSM, DTM

# ÖZET

# 3 BOYUTLU ŞEHİR MODELİ OLUŞTURMA VE WEB TABANLI GÖRSELLEŞTİRME

**Mehmet BÜYÜKDEMİRCİOĞLU**

**Yüksek Lisans, Geomatik Mühendisliği Bölümü**

**Tez Danışmanı: Doç.Dr.Sultan KOCAMAN**

**Mayıs 2018 , 79 Sayfa**

3 boyutlu şehir modelleri günümüzde şehirlerin etkili yönetimi için artık bir zorunluluk haline gelmiştir ve afet yönetimi, navigasyon, güneş potansiyeli ve planlama simülasyonları gibi farklı amaçlarla kullanılabilmektedir. Bu modeller sadece görsel bir model olmanın dışında, sahip oldukları semantik bilgiler yardımıyla tematik sorgular ve analizler yapılmasına imkan vermektedir. Şehir modelleri üretimi farklı veri kaynaklarından farklı yöntemler kullanılarak yapılabilmektedir.Bu çalışmada BÖHYY (Büyük Ölçekli Harita Yapım Yönetmeliği) göre Türkiye'de üretilmekte olan fotogrametrik halihazır harita verileri ve stereo hava fotoğrafları kullanılarak bir yarı-otomatik 3 boyutlu şehir modeli üretimi için iş akış şeması geliştirilmiştir. Çalışmanın amacı fotogrametrik yöntemlerle üretilmiş olan verilere ek yapmadan şehir modeli üretmektir. Üretim esnasında olabildiğince açık kaynak kodlu veya eğitim/araştırma lisansı ücretsiz olan yazılımlar tercih edilmiştir.

Yöntemin uygulaması için Çeşme / İzmir bölgesine ait geniş formatlı dijital hava kamerası görüntüleri ve bu projeden üretilmiş halihazır harita verileri yardımıyla 3 boyutlu şehir modeli üretimi LoD2 seviyesinde yarı-otomatik olarak gerçekleştirilmiştir. Üretilen model otomatik olarak fotograflardan giydirilmiş, öznitelik bilgileri ve sayısal arazi modeli ile birlikte web üzerinde görselleştirilmiştir.Web üzerinde görselleştirilen model www.cesme3d.com adresinde Internet üzerinden yayına açılmıştır. Çalışmada boyunca karşılaşılan sorunlar not edilerek bu sorunların çözümüne dair yaklaşımlar belirtilmiştir.

**Anahtar Sözcükler:** 3 boyutlu şehir modeli, görselleştirme, sayısal yüzey/arazi modeli

# ACKNOWLEDGEMENTS

First and foremost, i would like to thank Dr.Sultan Kocaman for supervising my thesis and advising me to work on 3D city models topic.This thesis would not be completed without her valuable guidance and continous support. Besides being expert in Geomatics, she is a great leader and inspiring person.

I also would like to thank Dr.Ümit Işıkdağ for being co-supervisor for this thesis, giving valuable feedbacks about my work and being my co-examiner.He is a true expert in GIS, working with him was both instructive and enjoyable.

Another big thanks goes to Alpaslan Tütüneken from UVM Systems Turkey for sharing his experience and knowledge in 3D city models. His continous contributions and support led thesis work to another stage.

I wish to thank my long-time friend Gökhan for encouraging me whenever i needed, also thanks to my classmates and close friends Murat, Gizem and Ilyas for their friendship and support.

Finally, i would like to thank my family for their endless support and encouragement.

# TABLE OF CONTENTS

**Page**

ABSTRACT ........ i

ÖZET ........ ii

ACKNOWLEDGMENTS ........ iii

TABLE OF CONTENTS ........ iv

LIST OF TABLES ........ vi

LIST OF FIGURES ........ vii

SYMBOLS AND ABBREVIATIONS ........ x

1. INTRODUCTION ........ 1
    - 1.1. Motivation ........ 2
    - 1.2. Objectives and Research Questions ........ 3
    - 1.3. Thesis Structure ........ 4

2. THEORETICAL BACKGROUND ........ 5
    - 2.1. 3D City Models ........ 5
    - 2.2. CityGML ........ 8
    - 2.2.1. Levels of Detail in CityGML ........ 8
    - 2.2.2. Semantics in CityGML ........ 11
    - 2.2.3. Terrain Intersection Curve ........ 12
    - 2.3. 3D City Model Reconstruction ........ 13
    - 2.3.1. Model-driven Methods ........ 15
    - 2.3.2. Data-driven Methods ........ 16
    - 2.4. Digital Surface Model Generation ........ 19
    - 2.5. The Role of Automatic Texturing in 3D City Modeling ........ 21
    - 2.6.Visualization of 3D City Models ........ 23

3. MATERIALS AND METHODS ........ 24
    - 3.1. Study Area and Dataset ........ 24
    - 3.2. Hardware and Software ........ 26
    - 3.3. Model Generation Workflow ........ 27

4. IMPLEMENTATION OF 3D CITY MODEL ........ 28
    - 4.1. Photogrammetric Triangulation ........ 28
    - 4.2. Building Footprint and Attribute Generation from Basemaps ........ 29
    - 4.3. Digital Terrain Model Generation ........ 32

  4.4. Digital Surface Model Generation .................................................... 35
  4.5. Automated Building Reconstruction of the 3D City Model ............................ 38
5. AUTOMATIC TEXTURING BUILDINGS ........................................................ 45
  5.1. CityGRID Texturizer ................................................................ 45
  5.2. Image Acquisition and Radiometric Pre-processing .................................... 46
  5.3. Image Rectification ................................................................ 47
  5.4. Database Creation and Data Import .................................................. 48
  5.5. Texture Mapping Process ............................................................ 49
6. WEB–BASED VISUALIZATION ............................................................. 54
  6.1. CesiumJS Virtual Globe ............................................................. 54
  6.2. KML/COLLADA/glTF ................................................................... 56
  6.3. Cesium 3D Tiles .................................................................... 61
  6.4. Terrain Visualization with Cesium .................................................. 67
7. VISUALIZATION BASED ON UNITY GAME ENGINE ............................................ 74
8. CONCLUSIONS AND FUTURE WORK ......................................................... 77
REFERENCES ............................................................................ 80
APPENDIX .............................................................................. 87
CURRICULUM VITAE ...................................................................... 102

# LIST OF TABLES

**Page**

**Table 2.1** Accuracy requirements of different LODs by OGC ........................................... 11

**Table 3.1** Hardware specifications of the portable personal computer used in the study .. 27

**Table 3.2** List of the software used in the study ................................................................ 27

**Table 4.1** Photogrammetric block properties and triangulation results .............................. 29

**Table 4.2** Automatic reconstruction accuracy of 1000 buildings generated by BuildingReconstruction 2018 software evaluated by visual checks. .................................. 45

# LIST OF FIGURES

**Page**

**Figure 1.1** Konya 3D city model project ................................................................ 1

**Figure 2.1** Different use cases of 3D city models used in various application fields ........ 7

**Figure 2.2** Levels of Detail in CityGML ................................................................ 8

**Figure 2.3** LoD0 building and UML diagram ........................................................ 9

**Figure 2.4** LoD1 to LoD4 .......................................................................................... 9

**Figure 2.5** An improved LoD specification for CityGML. .................................... 10

**Figure 2.6** Examples of representations of ground surfaces.................................... 12

**Figure 2.7** Example of the Terrain Intersection Curve represented as a black line ........... 13

**Figure 2.8** Reconstruction method given by Yi et al. ........................................... 14

**Figure 2.9** BuildingReconstruction 2018 software roof library.............................. 15

**Figure 2.10** Segmented roofs: Aerial photo, point cloud sorted by height and point cloud segmented ............................................................................................................. 17

**Figure 2.11** Different image acquisition methods and angles for texturing city models.... 21

**Figure 2.12** Assignment of linear features between image and 3D model ............ 22

**Figure 2.13** Vienna virtual city model based on Unity game engine ...................... 23

**Figure 3.1** Overall view of the study area from Google Earth ................................ 24

**Figure 3.2** Level of detail in one image and the ground signalization of one GCP ........... 25

**Figure 3.3** Aerial photogrammetric image block and the ground control point distribution over the study area in Çeşme , İzmir .............................................................. 25

**Figure 3.4** The overallworkflow of the methodology............................................. 27

**Figure 4.1** Manual feature extraction from stereo aerial images ............................ 29

**Figure 4.2** FME workbench for generating attributes and converting DGN file into Shapefile ........................................................................................................... 30

**Figure 4.3** A part of basemap with all CAD layers and the building footprints extracted from it .............................................................................................................. 32

**Figure 4.4** Attribute table of the generated building footprints .............................. 32

**Figure 4.5** General view of the manually extracted elevation data and close view............ 33

**Figure 4.6** FME workbench for raster DTM generation.......................................... 34

**Figure 4.7** DTM with null values and null values fixed .......................................... 34

**Figure 4.8** Photogrammetric blocks generated for the area ..................................... 35

**Figure 4.9** Generated Tie points and dense point cloud .......................................... 36

**Figure 4.10** An overview and a close view of the generated DSM ......................... 37

**Figure 4.11** BuildingReconstruction 2018 software workflow ........................................ 38
**Figure 4.12** BuildingReconstruction 2018 project configuration wizard .......................... 39
**Figure 4.13** BuildingReconstruction graphical user interface .......................................... 40
**Figure 4.14** 25cm resolution DSM and 1m resolution DTM input for BuildingReconstruction software ........................................................................ 41
**Figure 4.15** Automatically generated buildings with BuildingReconstruction 2018 software ........................................................................................................ 42
**Figure 4.16** Automatically generated roof geometries overlayed on the orthophoto with 10 cm resolution ................................................................................................ 43
**Figure 5.1** CityGRID Texturizer automatic texturing workflow....................................... 45
**Figure 5.2** Raw aerial image and pre-processed image of Çeşme..................................... 46
**Figure 5.3** Image orientation checking with CityGRID software for automatic texturing. 48
**Figure 5.4** Importing aerial images and building models to database ............................... 49
**Figure 5.5** Automatic texturing properties ........................................................................ 50
**Figure 5.6** Textured CityGML export properties .............................................................. 53
**Figure 5.7** Exported CityGML buildings with textures...................................................... 53
**Figure 6.1** CesiumJS user interface ................................................................................... 54
**Figure 6.2** Workflow for visualizing with 3DCityDB software ......................................... 56
**Figure 6.3** Adding ITRF96 coordinate system to PostGIS................................................. 57
**Figure 6.4** Database connection with 3DCityDB .............................................................. 58
**Figure 6.5** Bounding box of the imported CityGML object............................................... 58
**Figure 6.6** 3DCityDB Spreadsheet generator ..................................................................... 59
**Figure 6.7** Attribute table of the selected buildings in the 3DCityDB ............................... 59
**Figure 6.8** Different visualization forms of models from LoD0 to Textured LoD2 on Google Earth.......................................................................................................... 60
**Figure 6.9** Textured LoD2 buildings and attributes visualizing on 3DCityDB Web map client .................................................................................................................. 61
**Figure 6.10** An overview of the 3D Tiling pipeline by Cesium.com ................................ 62
**Figure 6.11** Textured CityGML converted to 3D Tiles format and visualized on CesiumJS ......................................................................................................................... 63
**Figure 6.12** Showing selected untextured LoD2 buildings to attribute table on the customized web interface ......................................................................................... 64
**Figure 6.13** Çeşme city model colored grouped by their latitude using 3D Tiles styling .. 65
**Figure 6.14.** Textured mesh visualization on Cesium Web Globe .................................... 65

**Figure 6.15** Point cloud visualization on Cesium Web Globe .......................................... 66
**Figure 6.16** The Heightmap 1.0 format represents terrain with regular grid tiles .............. 69
**Figure 6.17** Irregular triangle meshes as Quantized-mesh 1.0 format............................... 69
**Figure 6.18** Cesium Terrain Builder command list .......................................................... 70
**Figure 6.19** layer.json file with all metadata information that describes the tileset ........... 71
**Figure 6.20** A view of generated 1m resolution DTM on Cesium .................................... 72
**Figure 6.21.** Buildings hanging in the air without terrain and buildings clamped to the terrain.......................................................................................................................... 73
**Figure 6.22** Wireframe view of the terrain in Heightmap format and Quantized-mesh format .......................................................................................................................... 73
**Figure 6.23** CityGRID Builder control center .................................................................... 74
**Figure 6.24** A scene from Unity Game Engine visualization generated with CityGRID Builder ......................................................................................................................... 76
**Figure 6.25** Trees over the roofs of the buildings.............................................................. 78
**Figure 6.26** An example of unclosed building footprint .................................................... 78
**Figure A.1** Camera locations and image overlaps of block 1............................................ 87
**Figure A.2** Camera locations and image overlaps of block 2............................................ 87
**Figure A.3** Camera locations and image overlaps of block 3............................................ 88
**Figure A.4** Camera locations and image overlaps of block 4............................................ 88
**Figure A.5** Camera locations and image overlaps of block 5............................................ 89
**Figure A.6** Camera locations and image overlaps of block 6............................................ 89
**Figure A.7** Camera locations and image overlaps of block 7............................................ 90

# SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| CAD | Computer Aided Design |
| DEM | Digital Elevation Model |
| DTM | Digital Terrain Model |
| DSM | Digital Surface Model |
| FME | Feature Manipulation Engine |
| FPS | Frame Per Second |
| GIS | Geographical Information System |
| GNSS | Global Navigation Satellite System |
| GUI | Graphical User Interface |
| GPU | Graphics Processing Unit |
| GSD | Ground Sampling Distance |
| HTML | HyperText Markup Language |
| INS | Inertial Navigation System |
| ITRF | International Terrestrial Reference Frame |
| IMU | Inertial Measurement Unit |
| LiDAR | Light Detection and Ranging |
| LOD | Level of Detail |
| OGC | Open Geospatial Consortium |
| RANSAC | Random Sample Consensus |
| SQL | Structured Query Language |
| TIC | Terrain Intersection Curve |
| TIN | Triangular Irregular Network |
| UAV | Unmanned Air Vehicle |
| UML | Unified Modeling Language |
| XML | Extensible Markup Language |

# 1. INTRODUCTION

According to United Nations World Urbanization Prospects [1], more than half of the of the world's population (54%) are currently living in cities, and this number will be 66% in 2050. For efficient management of large cities and urban population, new technologies must be developed so that higher living standards can be obtained.

In parallel to the fast growth of technology, 3D modeling and visualization have become popular in many engineering fields, especially in Geomatics Engineering. Nowadays, 3D city models are being produced and developed by many countries worldwide and it is a trending topic also in Turkey. With the help of such models, simulation and visualization tasks can be done in many application fields such as navigation, city planning, and disaster management [2]. Berlin [3], Vienna [4] and Konya [5] 3D city models can be shown as examples of pioneer models. A part of Konya 3D City Model is provided in Figure 1.1

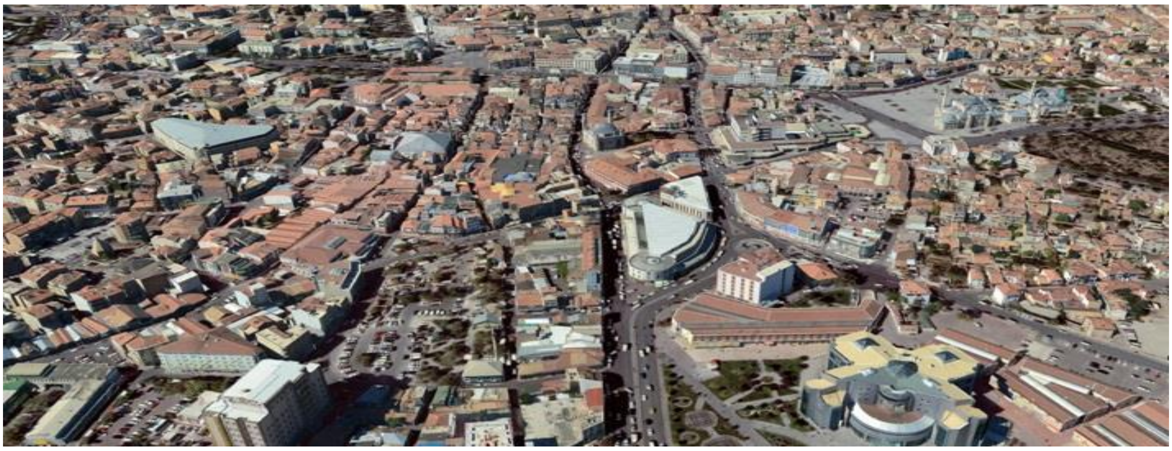


Figure 1.1 Konya 3D city model project

As a base, a city model involves a 3D description of buildings, and manual mensuration those, especially in large cities, requires a great deal of time, money and labor. Photogrammetric approaches are widely used for 3D extraction of surfaces and features, and it is one of the most preferred technologies in data production for 3D city models [6]. With the help of photogrammetric methods, feature geometries and digital elevation models (DEM) can be extracted automatically from stereo images [7]. Semi-automatic reconstruction of buildings can be done by combining DSMs (Digital Surface Models) with the cadastral footprints so that manual production time and labor can be reduced. Such DSMs are mostly obtained by stereo and/or multi-image matching approaches and by using aerial LiDAR sensors. So far it is not possible to reconstruct building geometries fully automatically with high accuracy and level of detail (i.e. resolution). Fully-automatic 3D city model generation is still an active agenda which attracts the attention of many researchers. There are several research studies on automation of the production and reliability improvements [8].

## 1.1 Motivation

In Turkey, use of aerial photogrammetry for map production is a lasting tradition, and aerial LiDAR sensors are not commonly used for point cloud acquisition as it is an expensive method with high initial costs [9]. In addition, although LiDAR sensors can provide highly accurate and dense elevation data, the spectral and radiometric resolutions of the intensity images are quite low in comparison to large format aerial cameras, thus not suitable for feature extraction for basemap production purposes. Stereo data acquisition with large format aerial cameras is still the most suitable approach for 3D city model generation, as point clouds and other types of information can be extracted from those with sufficient density, accuracy, and resolution (level of detail, feature types, etc.). It is possible to produce a point cloud with 100 points per square meter density from stereo aerial images with a resolution of 10 cm (image matching per pixel). Such a point cloud is denser than most LiDAR datasets in practical projects.

The production of different basemaps (small and large scale) and high-resolution orthophotos has been carried out by several Turkish government agencies, e.g. General Command of Mapping, General Directorate of Land Registry and Cadastre, etc., for many years with aerial photogrammetric methods. These maps have been produced on the basis of Turkish Large Scale Map Production Regulations, which defines the standards on the processing methods and accuracy requirements for different measurement and data acquisition techniques especially for large scale (e.g. 1:500, 1:1000) basemap generation. The basemaps include most objects that are visible in aerial photographs, such as buildings, roads, electricity poles etc. In addition, information and features that represent the characteristics of topography, such as the contour lines, break lines, etc. are manually measured by photogrammetry operators in stereo mode. The topographical information is used for computation of grid DEMs with a coarse grid interval, i.e. 5 m, which are mainly needed for orthophoto production.

A true ortophoto production project has been carried out in 2015 for urban areas in Turkey by the Ministry of Environment and Urbanisation. Within this project, vertical aerial images with 8 cm GSD (ground sampling distance) have been acquired and true orthophotos with 10cm spatial resolution have been produced. A major advantage of using true-orthophotos for building reconstruction is fast manual extraction of building footprints without the need for stereo vision.

Considering the amount of collected and produced data for over decades with aerial photogrammetric methods in Turkey, the possibility of using these datasets for automatic or semi-automatic 3D city model generation without additional data acquisition cost is discussed in this study. A new approach is proposed for this purpose and applied using the data of photogrammetric flight mission carried out over Çeşme town near Izmir in 2016. In addition, 3D city model visualization methods, integration with the geographical information systems (GIS) and potential 3D queries have been investigated. The approach has been validation using reference data, and the problems encountered during the processing and the potential improvements to the approach have been investigated. Adequacy of the dataset and outputs have also been analyzed and the results are presented here. The study is intended to be one of the pioneers for 3D city model production in Turkey and show basic guidelines for such projects in the future.

## 1.2 Objectives and Research Questions

The aim of this thesis is to propose a workflow for automatic building reconstruction method for 3D city models and their visualization using existing data obtained from aerial photogrammetric flight missions in Turkey without any additional cost or data acquisition. The following questions have been answered during this study:

- Is it possible to produce a city model automatically or semi-automatically with the use of vertical aerial imagery and basemaps produced from these imagery?
- What would be the best level of detail for such a city model obtained from these datasets and also the most suitable format to store and represent the models?
- What is the accuracy of fully-automatically generated building geometries?
- What would be the quality of automatic texturing of buildings using aerial vertical aerial images?

- Is it possible to generate DTMs with high density and quality using the elevation data (e.g. elevation points, contour lines, etc.) that exist in the basemaps?
- What is the most appropriate interface, method and format for high-performance and high-quality visualization of the 3D models on the web?
- What kind of 3D queries and cartographic styling can be done on the web interface?
- Can 3D city models be explored with Virtual Reality with high fidelity?

In order to answer these questions, existing literature on 3D city modeling has been analyzed and a workflow has been developed considering existing methods and available datasets. The results of this workflow is analyzed and presented here.

## 1.3 Thesis Structure

The structure of this thesis is as follows:

**Chapter 2** – A literature review on the 3D city model production methods and existing applications is provided in this Chapter.

**Chapter 3** – A description of the application dataset and a general overview on the processing workflow is given here.

**Chapter 4** – This Chapter involves data preprocessing steps, automatic building reconstruction methodology and an assessment on the accuracy of the reconstructed buildings.

**Chapter 5** – Automatically texturing potential of produced building models from nadir aerial images is investigated and the results are discussed here.

**Chapter 6** – This Chapter covers the web-based visualization of 3D city models and the DTMs on a globe together with 3D queries.

**Chapter 7** – Investigations on the visualization and exploration of 3D city models in the Unity game engine using Virtual Reality are given in this Chapter.

**Chapter 8** – The last Chapter describes the problems encountered during this study, discusses the findings, proposes future work on potential improvements to the system and gives further recommendations to those who intend to build such models.

# 2. THEORETICAL BACKGROUND

## 2.1 3D City Models

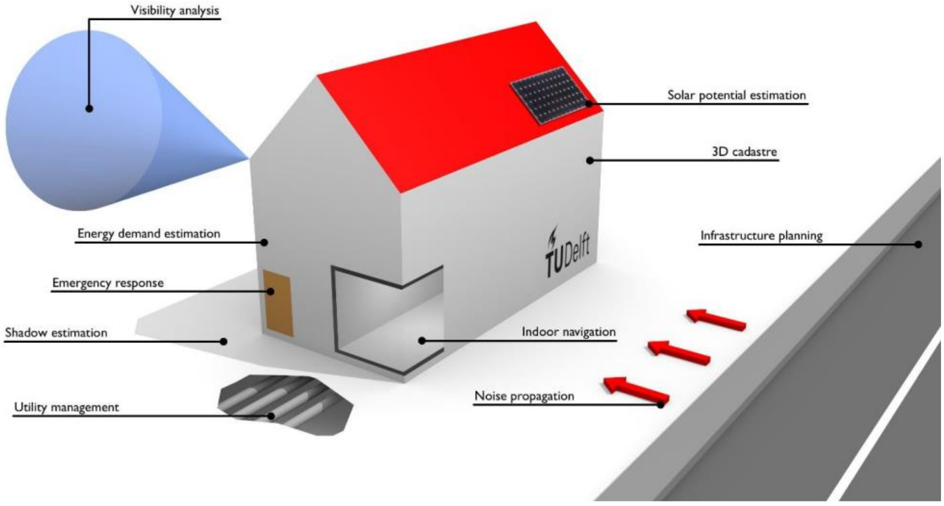
A 3D city model is a digital representation of the building and objects in the urban area, as well as the terrain model and represents urban geographically referenced spatial data [10]. 3D city models usually consist of digital terrain models (DTMs), building models, street-space models and green space models [11]. They can be used to perform simulations in different scenarios in virtual environments [12]. In parallel with the technological developments of the day, the traditional concepts of mapping and geospatial data has changed interactively, and new functions and applications for the field of geovisualization have been developed. Although the visualization of virtual city models is a preliminary plan, it is now used for many different purposes and is becoming more and more valuable every day. New technologies help users to over-come certain limitations of traditional 2D maps and explore 3D world [13]. Because of technological improvements and advances in computer hardware and software, 3D city models also become a visualization tool for communicating 3D knowledge to users. 3D city models equipped with interactivity and visual dynamics are becoming a reliable tool for information research and analysis.

3D city models should reflect the mutual relations between city objects and their complexities due to different purposes. [10]. Geometric, topological and semantic accuracy have an important role in the determination of the 3D city model quality [14]. In addition to building geometry, these 3D city models include different types of information such as plant height, road height, land use, and land use information. It is important to store information about these models together in the database and to make it easier to use and work with these models. Creating building geometry along with the appearance of the other objects with their necessary complexity is required to create more realistic 3D city models. However, it may be desirable to add semantic data to make these models understandable by computers. Semantic information of the city objects can be added semi-automatically [15]. The model will allow new species inquiries to be made with modeled semantic and topological aspects. Some of them are examples of thematic queries, analysis tasks and spatial data mining with the aid of semantics [11].

3D city models with semantic information require extra work compared to conventional city models [2]. They also require much more storage compared to 2D spatial data [16].For this reasons, it should be kept in mind that it is sensible to create these models if, in economic terms, customers can obtain something new from these products if the same data can be used in multiple applications by different customers. Today, three-dimensional representations of geographical information provide users more flexible and precise mathematical models, useful tools and user interfaces for geographical tools [17]. 3D city modeling is a rising research topic in different application areas. 3D city models are used in different areas such as computer graphics and architecture, engineering, construction and facility management [12].

3D city models are becoming important tools for urban decision-making processes and information systems, especially planning, simulation, documentation, heritage planning, mobile networking planning and navigation. The power to support naive geography is used by the broader public and results in easy-to-understand geospatial presentations. Unlike traditional methods with highly abstract content such as 2D maps, 3D map elements follow some natural/naive coding that is readily accessible to human worthy even in map reading. 3D city models also provide necessary information for different types of disaster management. These models primarily keep the general shapes and structure of a city. In case the city infrastructure suffers severe damage, it is impossible to calculate the magnitude of the damage quickly, to guide the assistance, and finally to reconstruct damaged areas. 3D city models are also suitable for visualizing localization in indoor and outdoor areas [12].

The focus in 3D GIS lies in the management of more comprehensive and exhaustive 3D models of the construction, management of domains, architectural, engineering, construction and facility management, wide area and georeferenced 3D models. Computer graphics are focused on the visualization of 3D models. Utilizations of a 3D city model depend on four different representation directions, geometry, topology, semantics and graphical visualization. While topology along with geometry describe the spatial structure of 3D city models, semantic properties include thematic structures, qualities, and reciprocal relationships. For the visualization of the model, information is given about graphical representations such as facade textures, object colors and signatures [18]. Different disciplines (e.g., architecture, engineering, etc.) emphasize different aspects and only those aspects are included in the model; For this reason, a model that is considered good for the identification of certain phenomena may not be suitable for someone else [19]. Biljecki et al. [20] identify and organize 29 different use cases used in various application fields and provides examples for each use case (Figure 2.1).



| § | Use case | Example of an application |
|---|---|---|
| **1.1** | Estimation of the solar irradiation | Determining the suitability of a roof surface for installing photovoltaic panels |
| **1.2** | Energy demand estimation | Assessing the return of a building energy retrofit |
| **1.3** | Aiding positioning | Map matching |
| **1.4** | Determination of the floorspace | Valuation of buildings |
| **1.5** | Classifying building types | Semantic enrichment of data sets |
| **2.1** | Geo-visualization and visualization enhancement | Flight simulation |
| **2.2** | Visibility analysis | Finding the optimal location to place a surveillance camera |
| **2.3** | Estimation of shadows cast by urban features | Determination of solar envelopes |
| **2.4** | Estimation of the propagation of noise in an urban environment | Traffic planning |
| **2.5** | 3D cadastre | Property registration |
| **2.6** | Visualization for navigation | Navigation |
| **2.7** | Urban planning | Designing green areas |
| **2.8** | Visualization for communication of urban information to citizenry | Virtual tours |
| **2.9** | Reconstruction of sunlight direction | Object recognition |
| **2.10** | Understanding SAR images | Interpretation of radar data |
| **2.11** | Facility management | Managing utilities |
| **2.12** | Automatic scaffold assembly | Civil engineering |
| **2.13** | Emergency response | Planning evacuation |
| **2.14** | Lighting simulations | Planning lighting of landmarks |
| **2.15** | Radio-wave propagation | Optimizing radio infrastructure |
| **2.16** | Computational fluid dynamics | Predicting air quality |
| **2.17** | Estimating the population in an area | Crisis management |
| **2.18** | Routing | Understanding Accessibility |
| **2.19** | Forecasting seismic damage | Insurance |
| **2.20** | Flooding | Mitigating damage to utility management |
| **2.21** | Change detection | Urban inventory |
| **2.22** | Volumetric density studies | Urban studies |
| **2.23** | Forest management | Predicting tree growth |
| **2.24** | Archaeology | Visualizing ancient sites |

Figure 2.1 Different use cases of 3D city models used in various application fields (Biljecki et al. [20])

## 2.2 CityGML

CityGML is an XML-based open-source data format for storing and exchanging 3D city models [21]. This format is adopted by Open Geospatial Consortium (OGC) as an international standard format [22]. CityGML has a structure suitable for presenting city models with semantic data. These semantic features allows users to perform functions that are not possible without the metadata provided by CityGML. Modules in CityGML reflect the appearance, spatial and theme characteristics of an object [23]. In addition to the geometric information of the city model, CityGML contains semantic and thematic information. These data enable users to make new simulations and queries and analyzes of 3D city models [2].

In CityGML-based city models, buildings are separated by different surfaces due to logical criteria. For example, a building can be divided into different parts depending on different roof types [2]. CityGML ensures that features have non-spatial properties. It is also possible to connect objects to related objects in external clusters such as spatial databases. Features are arranged in smaller sub-sections called modules such as Building modules or Plant modules. Modules can be combined to create a new feature called profilers [21].

### 2.2.1 Levels of Detail in CityGML

There are five different levels of detail defined in CityGML data schema (Figure 2.2). As the LoD level gets higher, it has a more detailed architecture with the complexity of the structures, so different levels of detail can be used for different purposes. CityGML allows the same objects to be visualized at different levels of detail. Thus, analysis can be done in different LODs [2].

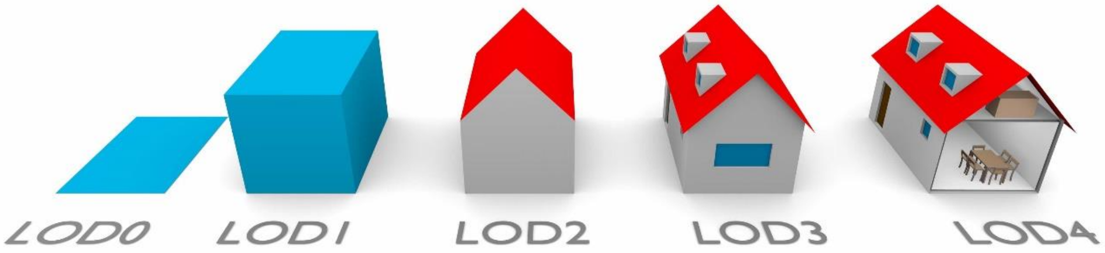


Figure 2.2 Levels of Detail in CityGML by Biljecki et al. [24]

A building in LoD0 is represented by 2.5D polygons either at roof level height or at ground level height [25]. LoD0, as represented in Figure 2.3, can also be called building footprints.

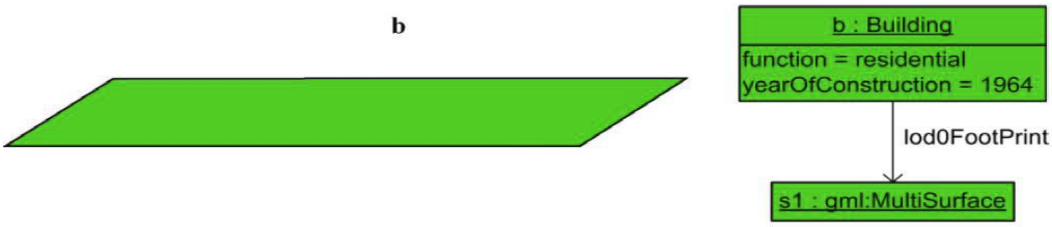


Figure 2.3 LoD0 building (left) and UML diagram (right) by Gröger and Plümer [21].

In LoD1, the building is represented as a solid model or a multi-faced block model without any roof structures (Figure 2.4). The building can be separated into different building surfaces called "BuildingParts". The geometric and thematic properties of these building components may differ from each other. Buildings and building parts potentially have the same spatial and non-spatial characteristics. The attribute values of a segment are valid for that segment only, whereas the attribute values of a building belong to all segments. If multiple parts are available, the building geometry should be represented by parts only.

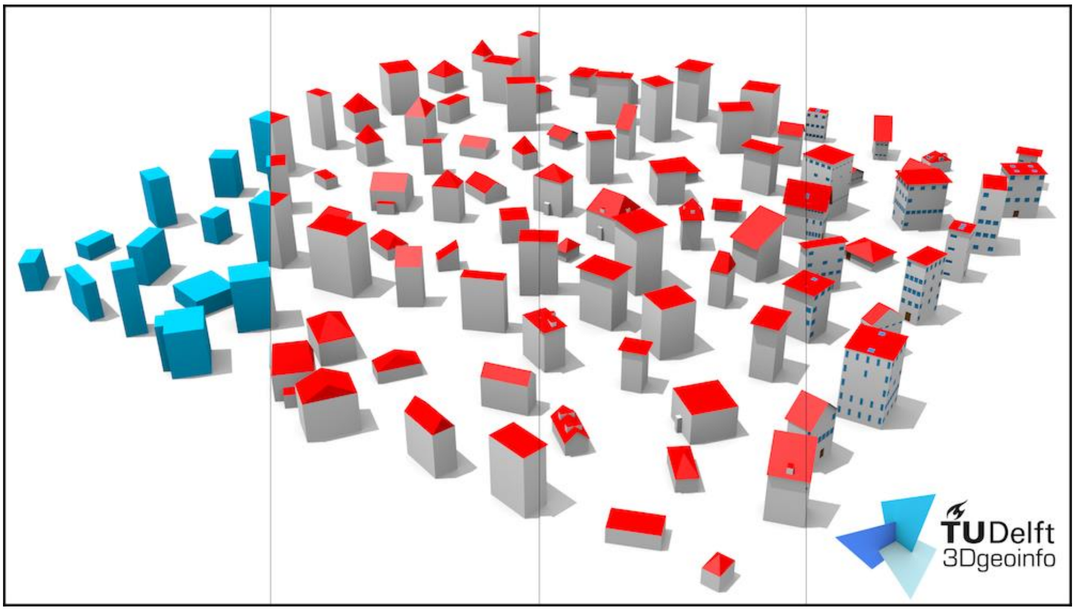


Figure 2.4. LoD1 to LoD4 from left to right by Biljecki et al. [24]

LoD2 adds generalized roof structures to LoD1. In addition, thematic details can be used to represent the boundary surfaces of a building. The main difference between LOD1 and LOD2 is that the outer walls and roof of a building can be represented by more than one face and the curved geometry of the building can be represented in the model structure [21].

Vertical wall surfaces “WallSurface”, the surfaces that cover the building from the top called ”RoofSurface”, and the horizontal surfaces that bound the bottom of the building from the floor are represented by “GroundSurfaces” in UML diagram.

LoD3 is generated by expanding LoD2 with openings (windows, doors), detailed roof structures (roof windows, chimneys, roof tops) and detailed façade constructions. These objects are represented as details with their own attributes and surface geometry. The roofs are more detailed, and details such as windows and doors have been added to the walls. [21].

LoD4 is the highest level of detail and adds interior architecture to the building. At this level of detail, all interior details such as furniture and rooms are represented with textures. For partially closed objects such as tunnels, special surfaces must be used to close the elements. These surfaces do not actually have any contrast and are only considered when they are needed for calculating the volumes and can be neglected if they are not needed for visualization purposes.

Each detail level in the CityGML data structure has features that make them unique. These features may change with the forthcoming CityGML 3.0. At the time of this writing, CityGML does not support relative 3D point accuracy, but this feature will be added in the future. Table 2.1 shows the different features that each level of detail have in CityGML. For example, in LOD2 the positional and height accuracy of points should be 2m or less, and all objects with a footprint of at least 4m by 4m should be considered in the model. Figure 2.5 shows an improved LoD specification for CityGML proposed by Biljecki et al. [24].

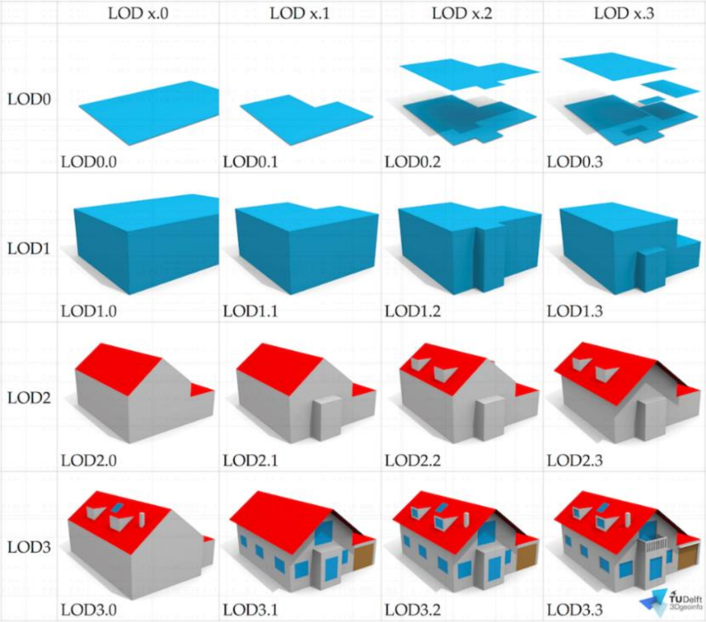


Figure 2.5 An improved LoD specification for CityGML by Biljecki et al. [24]

| | LOD0 | LOD1 | LOD2 | LOD3 | LOD4 |
|---|---|---|---|---|---|
| **Model scale descriptor** | Regional, landscape | City, region | City, city districts, projects | City districts, architectural models (exterior), landmark | architectural models (interior), landmark |
| **Class of accuracy** | Lowest | Low | Middle | High | Very High |
| **Absolute 3D point accuracy for position and height** | Lower than LOD1 | 5m | 2m | 0.5m | 0.2m |
| **Generalization** | Maximal generalization | Object blocks as generalized features; >6*6m/3m | Object as generalized features; >4*4m/2m | Object as real features; >2*2m/1m | Constructive elements and openings are represented |
| **Building installations** | No | No | Yes | Representative exterior features | Real object form |
| **Roof structure** | Yes | Flat | Differentiated roof structures | Real object form | Real object form |
| **Roof overhanging parts** | Yes | No | Yes, if known | Yes | Yes |
| **City Furniture** | No | Important objects | Prototypes, generalized objects | Real object form | Real object form |
| **Vegetation objects** | No | Important objects | Prototypes, higher than 6m | Prototypes, higher than 2m | Prototypes, the real object form |
| **Vegetation areas** | no | >50*50m | >5*5m | < LOD2 | < LOD2 |

Table 2.1 Accuracy requirements of different LODs by OGC [22].

### 2.2.2 Semantics in CityGML

The main feature that distinguishes CityGML from other formats is that it supports the semantic data as well. Semantics can be added to building model in a semi-automatic way. Automatically detecting small independent objects such as building windows and doors is still a challenging issue to work on [15]. World objects at semantics are represented with buildings, walls, windows, and rooms. These definitions also include attributes, associations and association hierarchies between the details. Hence, the part-of relationship between details can be achieved only at the semantical level, regardless of the geometry. However, at the spatial level, geometry objects are assigned to details representing their geographical location and scope. Thus, the model consists of the semantic and geometric hierarchies that the conjugate objects. The advantage of this approach is that it can be arbitrarily navigated between and within both hierarchies to answer thematic and/or geometric queries or to perform analyses. If both types of hierarchies exist for a given object, they must be consistent. In other words, they must match and adapt to each other. For example, if a building's wall at a semantic level has two windows and a door, it is imperative that the geometry showing the wall contains the geometry parts of both windows and the door at the same time [22]. CityGML's semantic model includes class definitions for the most important thematic object types in virtual 3D city models, including buildings, DTM, water masses, transportation, vegetation, and urban facilities [2].

### 2.2.3 Terrain Intersection Curve

Building and terrain integration in city models is one of the important issues [26]. This feature describes one of the usual technical problems that arise when the terrain model is combined with a 3D object with a different height value. In principle, a 3D object must conform to the terrain model. However, if the digital terrain model and the 3D object are created at different levels of detail or by different data providers, a significant problem arises at their intersection [27]. Terrain Intersection Curve is a CityGML feature that identifies the intersection of land and structures. This curve refers to the precise location of the ground where the building touches the building and is represented by a closed ring surrounding the building. If the land and the building are incompatible, it is possible to raise the building up or down to make it fit. In Figure 2.7, the TIC indicating the exact location of the intersection of the building and the land can be seen [21]. Figure 2.6 shows how TIC should be modeled according to the CityGML standard.

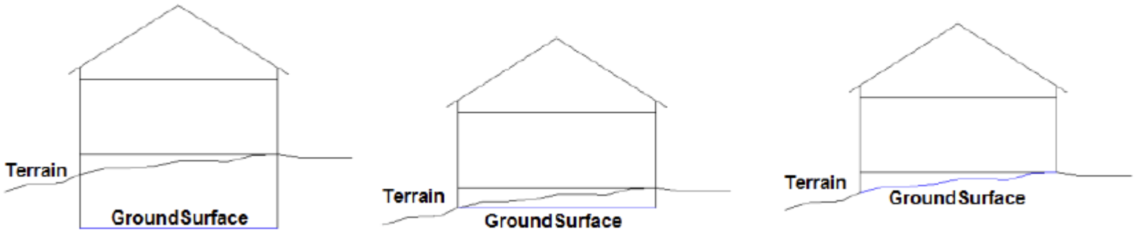


Figure 2.6 Examples of representations of ground surfaces .

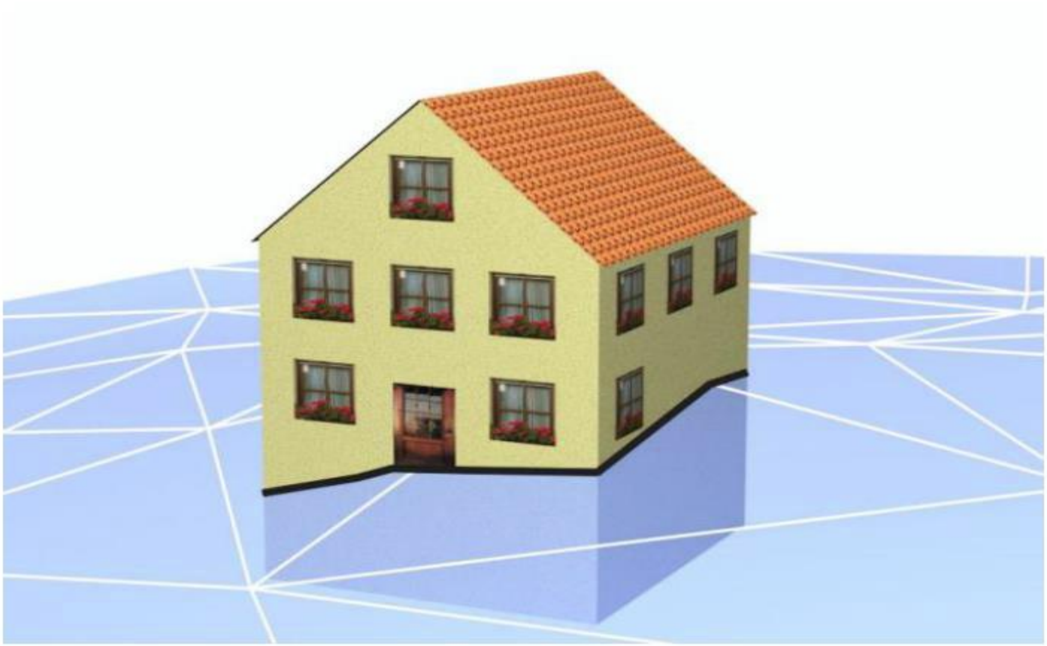


Figure 2.7 Example of the Terrain Intersection Curve represented as a black line [22].

### 2.3 3D City Model Reconstruction

3D city models can be generated manually, semi-automatically or fully-automatically. In manual reconstruction, buildings are usually measured by photogrammetry operators using stereo photography. In semi-automatic process, some parts of the reconstruction are done by the users while the software does the rest. In fully-automatic reconstruction, software does all the job without any manual intervention [28, 29].

3D city model reconstruction can be performed from different data resources such as LiDAR point clouds [30], airborne images [31, 32], satellite images [33, 34], UAV images or combination of DSM data with cadastral maps [35]. The building reconstruction mainly can be done in three process steps: building detection, extraction, and reconstruction [36]. The reconstruction of the buildings is a process for generating a model using features obtained from building detection and extraction. With the use of photogrammetric methods, building rooftops can be measured manually from stereo pairs or extracted by edge detection methods with their radiometric characteristics and or classification methods can be applied. They can be also extracted from point clouds generated from photogrammetric image matching methods [37] or LiDAR sensors.

The accuracy of 3D city models relies on the resolution and the source data density. In the photogrammetric procedure, GSD (ground sampling distance) is the most important factor in reconstructing building geometries and identifying roof structures. Kocaman et al. [33] extracted 3D city models from DSMs generated from IKONOS (1 m GSD) and QuickBird (0.7 m GSD) satellite images. They have investigated the precise orientation and radiometric characteristics of the images. Reconstruction of 3D city model is done by CyberCity Modeler software. They concluded that building models with the main roof structure can be successfully extracted by very high-resolution satellite images.

Yi et al. [38] presented a novel method for automated reconstruction of volumetric design and shapes of the buildings from point clouds acquired with LiDAR. When large-scale LiDAR data from a group of urban units are considered, they took benefit from the "divide and conquer" strategy to break down all point clouds into several sub-clusters, each corresponding to a single building. For each building in the city model, a separate vertical straight line is produced and the corresponding points are separated into blocks by examining the distribution of building properties in this straight line. Next, they proposed a novel algorithm, Spectral Residual Clustering (SRC), to extract the primitive elements within the contours of blocks from the sectional point set, which is formed by registering the series of consecutive slicing points. Subsequently, they detect the geometric constraints among primitive elements through the individual fitting and perform constrained fitting overall primitive elements to obtain the accurate contour. On this basis, they execute 3D modeling operations, like extrusion, lofting or sweeping, to generate the 3D models of blocks. The final accurate 3D models are generated by applying the union Boolean operations over the block models. Figure 2.8 shows a schematic representation of this workflow.

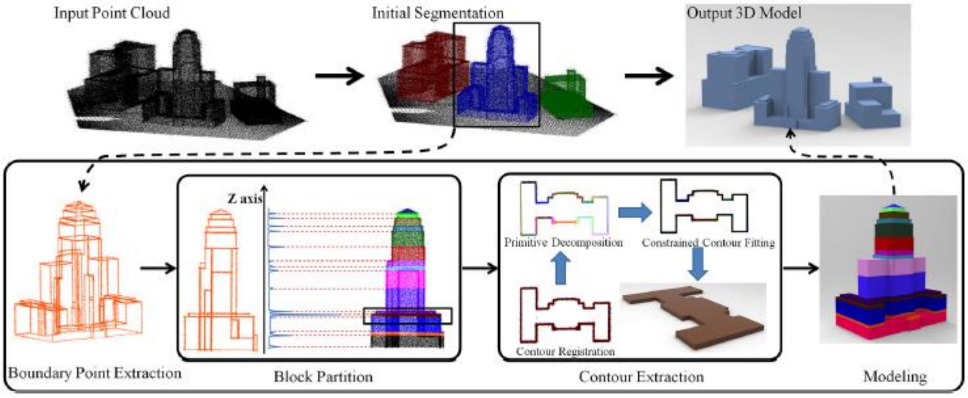


Figure 2.8 Reconstruction method given by Yi et al. [26], which essentially proceeds as block partition, contour extraction, and building modeling.

The most widely used method for reconstructing 3D cities is combining different data resources. Haala et al. [39] combined laser altimetry data with multispectral imagery for building, tree and green area extraction while in obtaining the generation of the building geometry, LiDAR point cloud and cadastral maps are used. Another combination was applied by Suveg et al. [29], where aerial images were analyzed with help of large scale 2D GIS data.

According to Dorninger et al. [40], reconstruction methods of the city models can be divided into two different categories: model-driven methods and data-driven methods. The aim of the model-driven method is to match the DSM generated roof geometry with the roof types that it has in its library [30]. In data-driven methods, the algorithm tries to reconstruct a complex building with the aid of different complex operations. They try to reconstruct buildings by using the DSM as main data source and try to analyze it as a whole without associating it to any set of parameters [41].

### 2.3.1 Model-driven Methods

Model-driven methods try to match the roof types in its library with the DSM and reconstruct the building using the best-matched roof type. This means that some preliminary assumptions made on the roof require that only some predefined parameters be calculated. [41]. This algorithm proposed the given study shows the best result with low resolution DSMs or sparse point clouds. This method also guarantees that the reconstructed roof model is topologically correct, problems may occur if the roof shapes don’t have any match in the roof library [40]. Huang et al. [42] have listed standard roof shapes such as flat, gabled and hipped roofs. A sample library is given in Figure 2.9

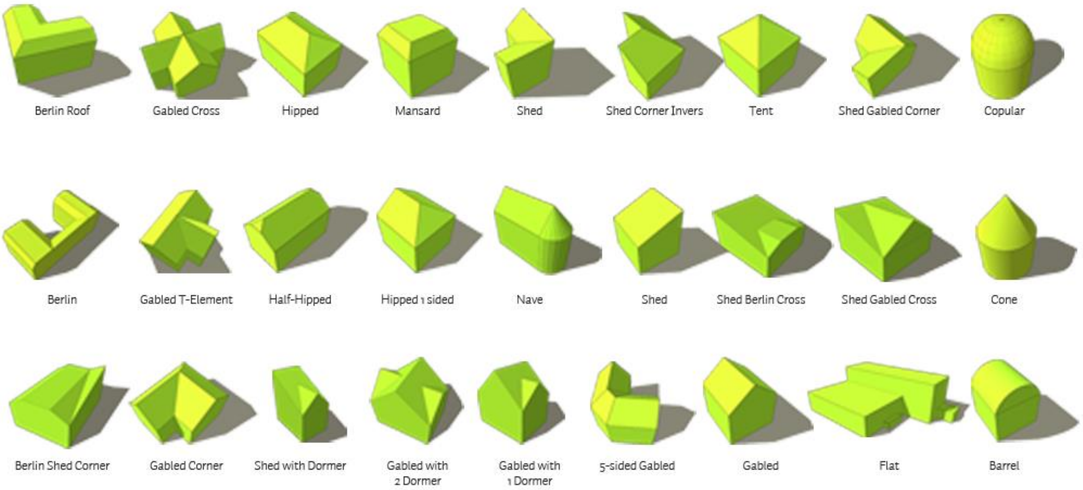


Figure 2.9 BuildingReconstruction 2018 software roof library

Brenner et al. [43] used building footprints as a limitation for the reconstruct area of the buildings. It should be noted that, depending on the building and the object to be modeled, the footprints represent the entire building area, not just the roof area. They assumed that each building is the result of the combination of different surfaces, and that each object can be modeled with the help of these surfaces. Building parts have an important role in modeling buildings with complex geometry. They are disassembled into smaller objects, then these smaller objects are modeled using primitives [41]. Algorithms in model-driven methods try to find the areas to compare with models and decide the model that will be used for each case. The recognition step gives a number of approximate positions for each approach, then adjusts these approaches to the data. The number of parameters to work depends on the complexity of the primitives that are used in the transformations in the matching process [44].

Huang et al. [45] reconstruct buildings without footprint data. Building footprints are generated from LiDAR point cloud data. Primitives that matches with the building are extracted and merged to reconstruct building geometry. Talandier et al. [46] reconstruct buildings without partitioning building footprints. A uniform eaves elevation is generated and an inclined angle is retrieved for each edge. If the sloping ceiling segment does not pass through the footprint edge, a gabled roof is assumed. According to Tarsha-Kurdi et al. [41], parameters used in model-driven methods can be classified into two types: building footprint and building space. Building footprint defines the location, alignment and the properties along with the façade equations. The building space defines the roof plane equations.

The model-driven methods have several advantages over data-driven methods. The reconstructed model will always be geometrically correct without any gaps, overlapping segments or visual deformations. As the parameter calculation for models does not require computational power, the reconstruction speed is faster compared to data-driven methods. But the main disadvantage of the model-driven methods is the shape of the roof is limited to a pre-defined library of roof types to compare DSM with it [41].

### 2.3.2 Data-driven Methods

Data-driven methods do not require any pre-defined roof library. In this approach, the algorithm reconstructs the buildings directly from given DSMs. Buildings and roof parts are reconstructed with different algorithms. The main problem with the data-driven method is that they may cause topological or geometrical errors, such as intersecting different segments with each other without a gap [40]. This method analyzes DSM as a single block without connecting it with any parameter sets and reconstructs buildings DSM as prime data source. The biggest advantage of this method is that there are no limitations on the roof shapes like in the model-driven methods [41]. There are a couple of different approaches for segmenting a DSM into planar regions [47].

Segmentation can be divided into two sub-methods according to the techniques that they describe homogenous areas: edge-based and surface based. Edge-based methods relies on detecting edges and borders, then segment the points inside this area using these detected shapes. Surface-based methods use local geometry as a sign of similarity for points. Building geometry is generated from newly merged region, which is done by merging surface characteristics and points with same spatial characteristics [47].

According to Elberink [48], the workflow for data-driven methods can be expressed in 5 different sections: data acquisition, segmentation, building detection, feature extraction and 3D reconstruction. The most widely used approaches for data-driven methods are segmentation, RANSAC, Hough transform, and clustering. Figure 2.10 shows examples to segmentation results from aerial image datasets.

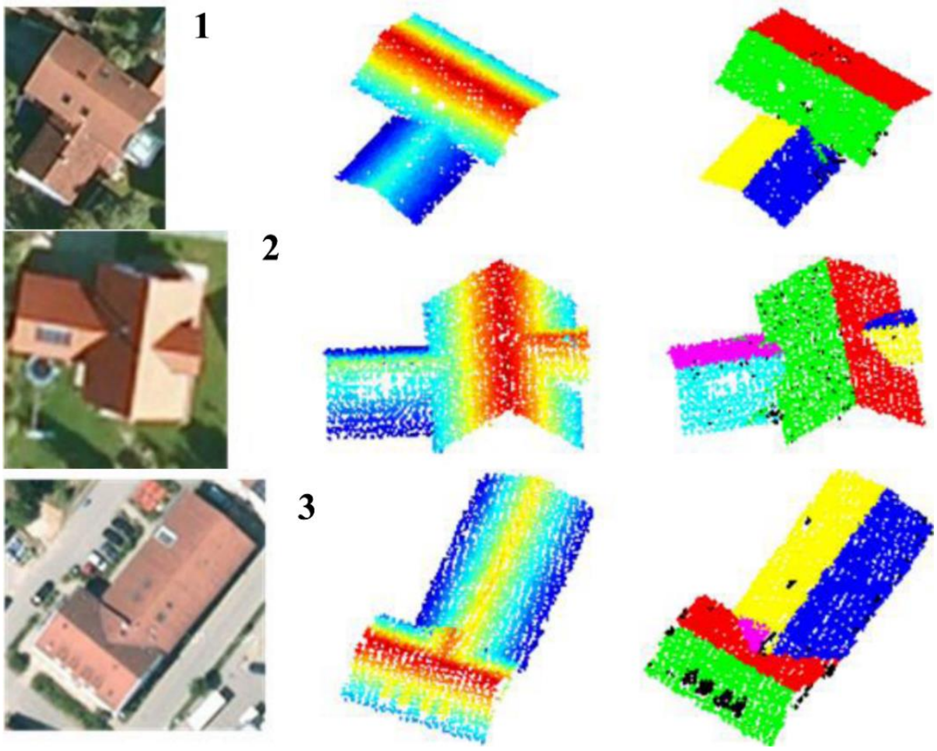


Figure 2.10 Segmented roofs: Aerial photo (left), point cloud sorted by height (middle) and point cloud segmented (right) Fan et al. [49]

The Random Sample Consensus (RANSAC) approach finds the optimal plane for a given point cloud. It uses a point in the dataset randomly and repeatedly to determine the model's parameters. The generated parameters are then tested on the entire dataset. This process proceeds until the number of points that matches with the parameters reaches the intended threshold number. If the criterion cannot be met, a new set of parameters are determined with the new data set. This determination should be done in a recursive fashion, because only one set of parameters is tested in one round [41, 47].

The Hough transformation algorithm is a method that all patterns passing through each property point are voted. In this case, the pattern with the highest vote is regarded as the correct one. This method is sensitive to noise and discontinuities in patterns. These patterns can be for example straight lines, circles or ellipses [50].

Clustering algorithms can be used to group planes according to their normal vectors and to find planar segments such as image segmentation algorithms. Liu and Xiong [51] use the cell average shift by adopting the average shift algorithm. Sampath and Shan [47] perceive roof sections with fuzzy k. Using a potential-based approach, the number of clusters has been determined to be improved. A disadvantage of the orientation-based clusters is that different roof sections with the same orientation cannot be separated. This can be solved in the next splitting step. Another option is to add the points (x, y, z) to the feature vector. The disadvantage in this method is that large roof sections can be segmented due to large differences in the locations of the points. The general advantage of clustering methods is that each data point can be set simultaneously to the cluster to which it belongs. For this reason, clustering algorithms can be made parallel, which makes them very efficient.Due to the increased point density of laser scanners, data-driven methods can be assumed to generate more accurate and robust models. It shows that the smaller details of this roof can be modeled and that geometric parts belonging to the building can be produced more accurately [48]. The most important problems of data-driven method are the possibility of producing deformed models and long reconstruction periods.Tarsha-Kurdi et al. [41] list some causes of possible deformations by using data-driven methods based on the following:

- Generally, modeling is based on the assumption that a structure consists of planes and lines. This is usually not true from a mathematical perspective, and for this reason, the equations for planes are only approximate values.

- Errors in point clouds can cause problems. These errors may be due to errors in position or accuracy. Different artifacts and multi-way reflections are possible sources of error.

- The unevenly distributed point cloud can cause deformations in certain parts of the building during production.

- There is a direct relationship between the density of the point cloud and the possible deformations in the modeling phase. A more dense point cloud causes less deformation, while a less frequent point cloud causes more deformation.

- Interpolation of the point cloud may cause some effects on the model. It may allow the building façade to be lifted from the center to get a regular spot grid and correct some mistakes. If the sampling range for grid cells is very different from the point density, or there is free space in the point cloud, it can cause undesirable effects.

- The noise and small roof details found in the point cloud make the segmentation process difficult.

## 2.4 Digital Surface Model Generation

Generation of dense point clouds to model earth surface has become easier with the development of new sensors and processing methods. There are many methods used in the production of digital surface models (DSMs) from airborne and satellite platforms [52]. Matching of stereo satellite [53] and aerial imagery, DSM and DTM generation from airborne LiDAR data [54] are the most commonly used approaches for this purpose. 3D building reconstruction studies are often performed with LiDAR point clouds due to quick acquisition of the DSM from the pulse of the light and high positional accuracy [53]. In addition, algorithms for aerial and satellite image matching have been improved and can produce better results [29, 33, 34, 54].

According to Şengül [6], there are two basic approaches for obtaining spatial information about the city. The first one is to take a photo with a digital camera. The advantage of this approach is that it is a low-cost method compared to other methods. On the other hand, photographs should be taken from every aspect of the building, but it is sometimes difficult to take photos due to security and privacy issues [7]. The other disadvantage of the method is that it takes more time to find the correct model in photo matching. Users also need to manage a large number of image files as you need to take a few photos with all the heights of each building. Besides, it is difficult to match two images of the same building from different flight columns due to shadows. The texture of buildings extracted from the same image is not necessary to correct the color balance of the image, but it is more costly than taking terrestrial photographs. In addition, it is necessary to adjust the flight plan and internal and external steering parameters in case of the wind.

The second approach is extracting information from LiDAR point clouds. LiDAR point clouds are automatically produced with high quality, accuracy, and density. Photogrammetric point clouds or DSM provide geometric information in contrast to LiDAR data containing multi-echo beats or density information. Nex et al. [8] claim that image matching can generate denser point cloud than LiDAR data. The density of the point cloud produced from optical imagery mainly depends on the ground resolution, the image matching algorithm, image quality, and the surface characteristics (mainly texture). Under optimal conditions, matching at pixel interval, i.e. at Ground Sampling Distance level, is possible and results in 100 points per square meter with a GSD of one centimeter. A typical LiDAR flight for city modeling applications provide data in the order of 15 – 20 points per square meter.” Oude Elberink et al. [54] argues that the extraction of a higher number object points allows the modeling of discontinuities, such as building structures, which allows a better way for the geometric modeling.

Rothermel et al. [55] generated dense and accurate digital surface model from multi-view stereo images using the Semi-Global Matching (SGM) [56] method. Then, a fusion step is performed to merge redundant depth estimations across single stereo models. They presented an adaptive structure for the SGM method in which matching results of low-resolution pyramids are used to limit disparity search ranges for high-resolution pyramids.

DSMs, which are generated from stereo aerial imagery using mage matching methods, can be an alternative for LiDAR point clouds. The main obstacles of DSM generation from stereo aerial imagery are caused by image radiometric differences, congestion, shadows, radiometric problems such as blurriness/noise and some missing parts in point clouds due to acquisition methods [53]. Building geometries can be reconstructed using DSM generated with image matching from stereo aerial imagery [39]. Some 3D model reconstruction algorithms can extract the roof geometries. These rods can be seen in aerial images and satellite imagery.

## 2.5 The Role of Automatic Texturing in 3D City Modeling

One of the main criteria for 3D city model production is to create models that are close to reality both in geometry and also appearance (visual fidelity). Textures obtained from optical images are used for providing visually correct models quickly and economically. There are also cases where untextured building models are in use, however, 3D city models should be textured where the visualization is your front planer and the user needs to have an idea of the area at first sight. Regardless the level of detail in their geometries, untextured 3D city models will always be visually incomplete.

With the development of aerial photogrammetric systems equipped with nadir and oblique camera systems, automatic texturing of 3D city models became quite possible (Figure 2.11). Nadir cameras are the most ideal for texturing roofs of the buildings whereas the oblique cameras are better at texturing the building façades. Similarly, for automatic texturing of 3D city models, the best strategy is to select the rooftop textures from the nadir images and building façades from the oblique images. The view of the façades can be obtained clearly with this approach and the spatial resolution would also be higher. One can get a closer to the street level while looking at the 3D city model. Similar to the nadir image acquisition, the season and the time of the flight would affect the radiometric quality of the oblique images.

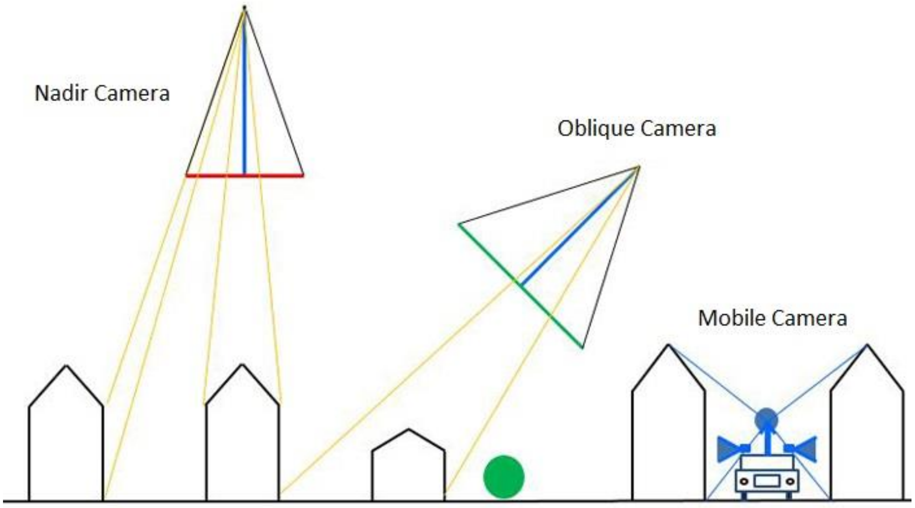


Figure 2.11 Different image acquisition methods and angles for texturing city models

Frueh et al. [57] have used an approach in their related work which combines 3d city model obtained from aerial and ground-based laser scans with oblique aerial imagery for texture mapping. They have proposed methods for texturing an existing 3D city model with multiple airborne images at oblique angles captured both façades and rooftops.

Images have been automatically registered by matching 2D image lines with projections of 3D lines from the city model. Then, for each model triangle that is modeled, the best image has been selected, by taking occlusion, image resolution, surface normal orientation, and consistency with neighboring triangles into account. Finally, the texture patches used in all images are combined into a single texture atlas for compact presentation and efficient image creation.

Früh and Zakhor [58] presented a method that merges ground-based and airborne laser scans and images. Additional terrestrial data used in the method naturally leads to more detailed models but also increases the size of the datasets considerably. A more efficient technique to model the façades is to extract textures from terrestrial images and locate them on the polygonal models. This process is done in a manual fashion, however, texturing of a single building is a tedious task and can easily take up to several hours. For a large number of building façades, such an approach is not efficient and usually not applicable.

Kada et al. [59] presented a new approach that automatically extracts façade textures from terrestrial photographs and maps them to geo-referenced 3D building models (Figure 2.12). They realized texture mapping using a programmable graphics processing unit. In contrast to the application of standard viewers, this technique allows to model even complex geometric effects like self-occlusion or lens distortion. This allows for a very fast and flexible on-the-fly generation of façade texture using real-world imagery.

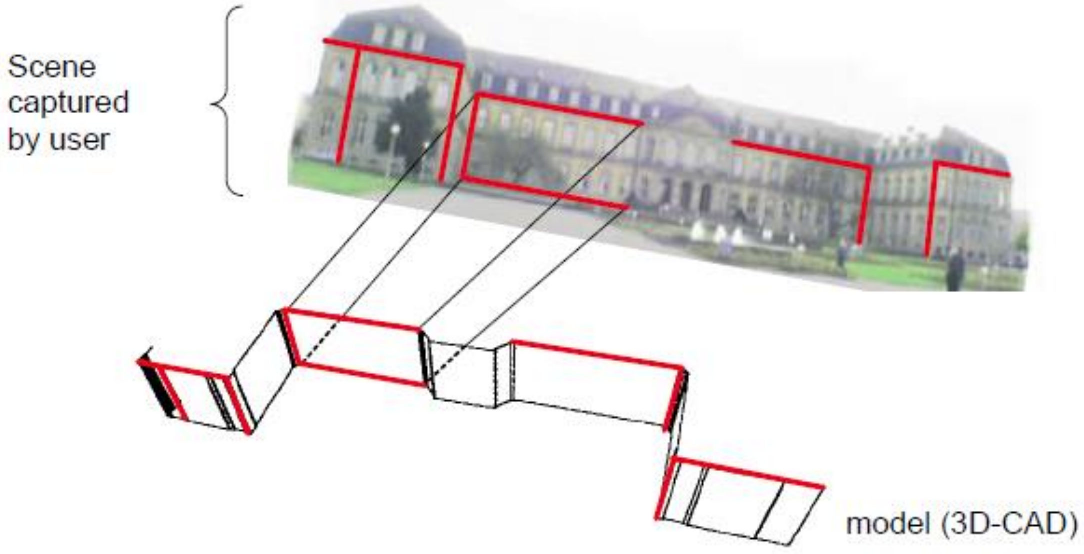


Figure 2.12 Assignment of linear features between image and 3D model Kada et al [59].

## 2.6 Visualization of 3D City Models

Nowadays, 3D city models are mostly used for visualization purposes. However, their applications areas are increasing continuously and they are being employed for various tasks beyond visualization. Capabilities of 3D city models are improving faster when compared to 2D models [60]. An example to a virtual city model based on Unity game engine is given in Figure 2.13. These capabilities can also be improved with different visualization methods. CityGML is an exchange and definition format, rather than a visualization format. They can be viewed on CityGML viewers like FME Inspector, but visualizing CityGML dataset requires some post-processing and conversion to other formats. They can be visualized on virtual web globes, smartphones or tablets [61] or even in game engines like Unity for exploring them with Virtual Reality [62]. With the recent technological developments in computer graphics and web, visualizing geospatial data on web browsers has become more common. The biggest advantage of using web browsers is that they can be accessed everywhere without any additional software requirements. The reason for using a Virtual Globe is to allow the users to visualize and navigate the world in 3D [63].

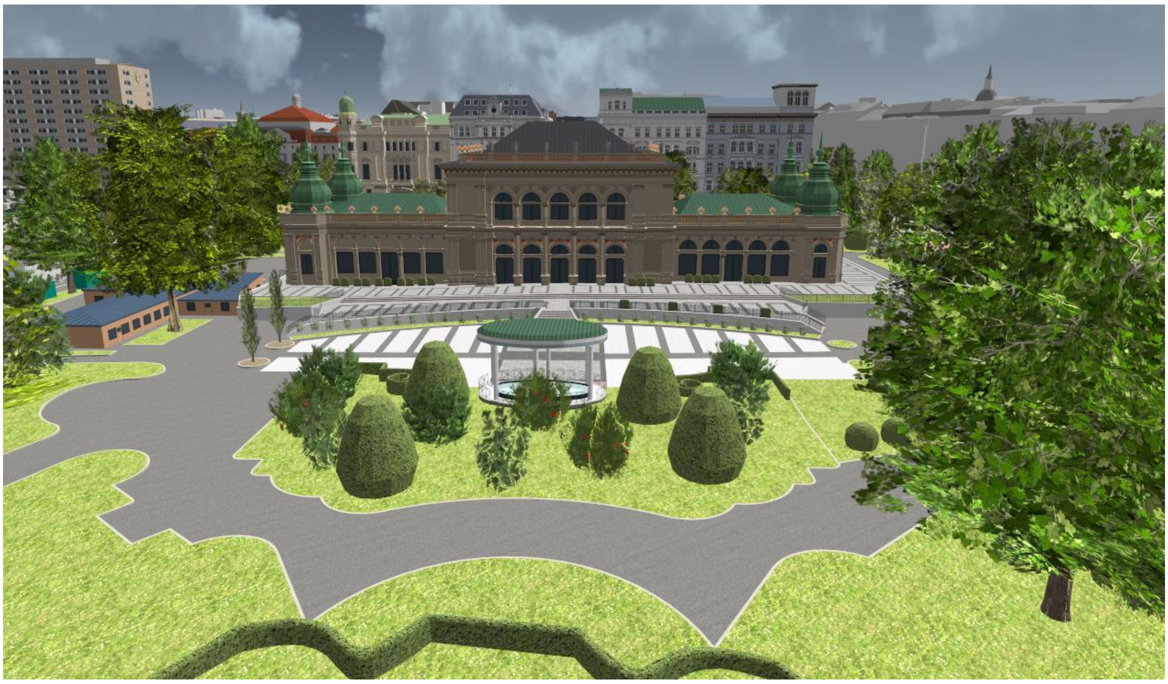


Figure 2.13 Vienna virtual city model based on Unity game engine – UVM Systems [64]

Some of the visualization cases can be listed as following: e.g. web-based visualization [65-68], volunteered geographic information [69-71], energy modeling [72, 73],gaming [74], navigation [75], forest simulation [76], augmented reality [77], real estate [78], crime mapping [79], flight simulators [80], air quality data [81] and tsunami analysis [82].

# 3. MATERIALS AND METHODS

## 3.1 Study Area and Dataset

Izmir is the largest metropolitan city in Turkey’s Aegean Region. Çeşme is a coastal town located 85 km west of Izmir and also very popular holiday resort with several historical sites. It is believed that Çeşme, which is known as Cyssus in antiquity, is now called "fountains" from which marines supply water.

The Çeşme Castle is located in the center of the Çeşme town, which was built in 1508. It was originally a seaside castle, but because of the alluvial deposits, it is currently slightly inland. A large majority of the structures in the Çeşme area are in Aegean architecture style. The Aegean architecture includes wood and tile, the exterior surfaces of the structures have embroidery motifs. Most of the buildings are located on the coastline and are usually at low altitude. The highest building in Çeşme is a hotel which has a height of approximately 45 meters. An overview of the study area is given in Figure 3.1

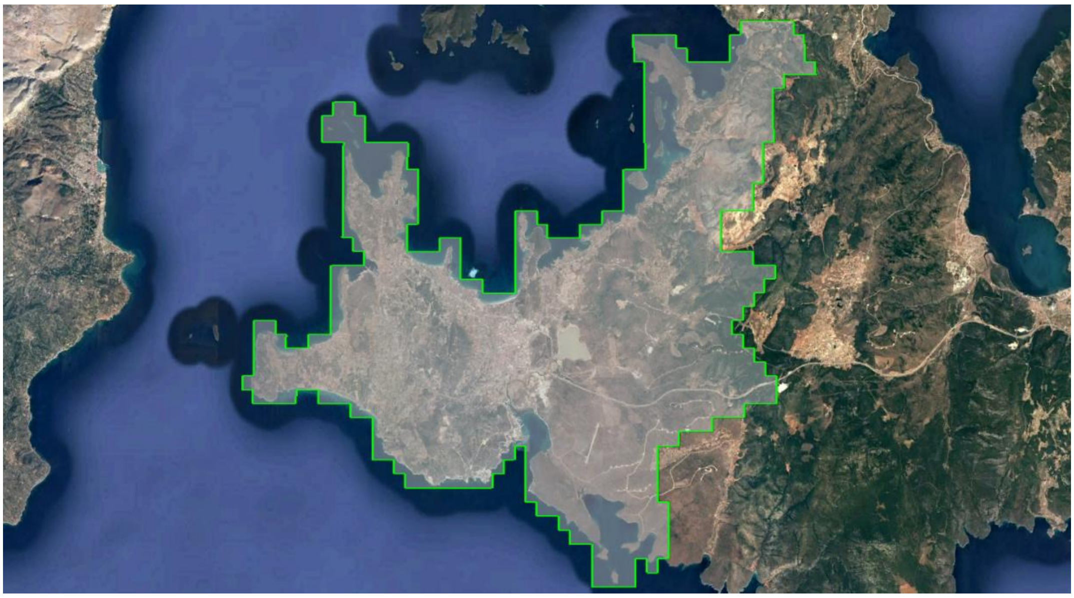


Figure 3.1. Overall view of the study area from Google Earth.

The study area covers the whole Çeşme region with a size of 260 km². Within the project, a total of 4468 aerial photos have been taken in May 2016 with %80 forward overlap and %60 lateral overlap using UltraCam Falcon large-format digital camera from Vexcel Imaging [83]. The photos have been taken from an average altitude of 1500 meters from mean sea level and have an average GSD of 8 cm and a footprint of 140 m x 90 m, forming a total of 7 sub-blocks of the whole area.  Figure 3.2 shows the level of detail in the images and the ground signalization of one control point, which was established prior to the flight.

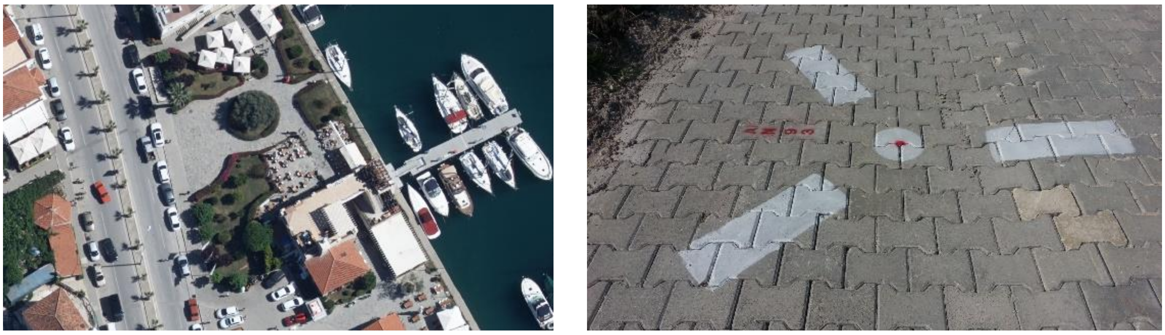


Figure 3.2 Level of detail in one image (left) and the ground signalization of one GCP (right).

306 Ground Control Points (GCPs) has been established prior to the flight with an interval of 1.5 km inside of every block. The block configuration is provided in Figure 3.3 Orthometric heights have been measured by leveling method and planimetric coordinates have been measured using Global Navigation Satellite System (GNSS) devices. The GCPs have been used in the photogrammetric bundle block adjustment.

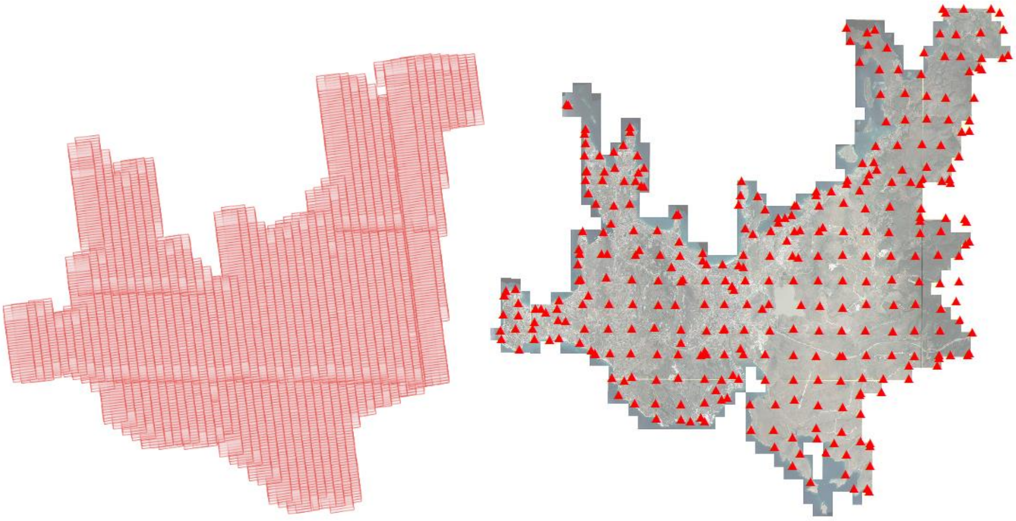


Figure 3.3 Aerial photogrammetric image block (left) and the ground control point distribution (right) over the study area in Çeşme, Izmir.

Manual feature extraction for base map production from stereo aerial images has been done by photogrammetry operators. Building footprints have been drawn from the outline of the roofs. Contour lines (1m, 2m, 5m, 10m) and breaklines have been mesured along with generated photogrammetric height points, which have later been converted to TIN (triangular irregular network) and used for generating the Digital Terrain Model (DTM). Lastly, orthophotos with 10 cm pixel size have been generated for the whole project area.

### 3.2 Hardware and Software

A portable personal computer (Dell 7577) has been used to process the data and implement the model. The hardware specifications is given in Table 3.1 A set of software and tools has been used for the processing, modeling, and visualization on the web. A complete list is provided  in Table 3.2.

| | |
|---|---|
| Operating System | Windows 10 |
| CPU | Intel Core i7-7700HQ @ 2.8GHz |
| Memory | 32 GB DDR4 2400MHz |
| Storage | 256GB SSD + 1 TB HDD |
| Graphics Card | Nvidia GeForce GTX1060 6GB |

Table 3.1 Hardware specifications of the portable personal computer used in the study.

| Software Package | Purpose of Use |
|---|---|
| 3DCityDB Importer / Exporter [84] | Converting CityGML to KML/COLLADA/glTF |
| Adobe Photoshop CC 2017 [85] | Radiometric pre-processing of aerial photos |
| Agisoft Photoscan Pro [86] | Point cloud and raster DSM generation |
| ArcGIS 10.2.2 [87] | Data visualization and editing |
| Bentley Microstation [88] | Editing and transforming basemaps |
| BuildingReconstruction 2018 [89] | Reconstructing buildings in CityGML format |
| CesiumJS 1.45 [90] | Visualization and development for web interface |
| Cesium Terrain Builder [91] | Terrain visualization for use in CesiumJS |
| CityGRID Builder [92] | Visualization on Unity game engine for VR |
| CityGRID Texturizer [93] | Automatic texturing buildings from aerial photos |
| FME 2018.0 [94] | Data conversation and manipulation |
| Google Earth Pro [95] | Visualizing 3D city model with KML format |
| Microsoft Visual Studio 2017 [96] | Compiling Cesium Terrain Builder C++ library |
| PostgreSQL with PostGIS [97] | Importing textured CityGML into the database |
| Sublime Text 3 [98] | JavaScript development for CesiumJS |
| QGIS 3.0.2 [99] | Data visualization and editing |
| Quick Terrain Reader [100] | DTM, DSM, and Point cloud visualization |
| XAMPP [101] | Local Apache HTTP serving for the web interface |

Table 3.2 List of the software used in the study.

### 3.3 Model Generation Workflow

After the photogrammetric processing, (i.e. GCP signalization on the ground, image acquisition and triangulation, manual feature extraction for basemap production), the buildings have been reconstructed automatically using this dataset. All of the features and building footprints extracted manually in Bentley Microstation [88] have been converted into ESRI Shapefile with FME software [94]. The DSM is generated with Agisoft Photoscan Pro [86] with a 30cm resolution for the whole project area. All contour lines, breaklines, and photogrammetric height points have been manually measured in Bentley Microstation [88] are triangulated, then converted into single raster digital terrain model with FME software. The buildings are reconstructed using BuildingReconstruction software from VirtualCity Systems GmbH [89] and the generated models have been converted into CityGML format as output. The texturing has been done automatically using the CityGRID Texturizer software from UVM Systems [64]. At last, the model has been integrated with the DTM and visualized on the web using open source CesiumJS library [90] and Unity game engine. 3D spatial queries have been added to the system by modifying the libraries of Cesium. The overall workflow of the study is provided in Figure 3.4.

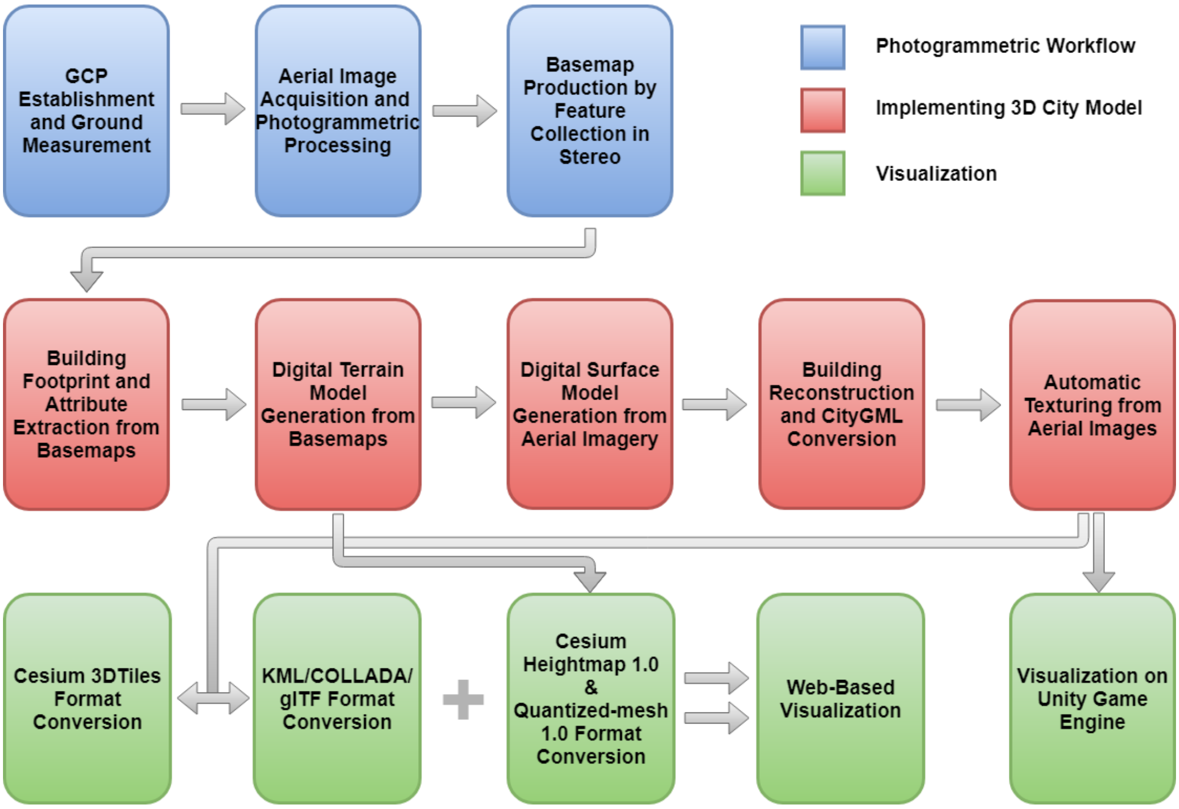

GCP Establishment and Ground Measurement
Aerial Image Acquisition and Photogrammetric Processing
Basemap Production by Feature Collection in Stereo
Photogrammetric Workflow
Implementing 3D City Model
Visualization
Building Footprint and Attribute Extraction from Basemaps
Digital Terrain Model Generation from Basemaps
Digital Surface Model Generation from Aerial Imagery
Building Reconstruction and CityGML Conversion
Automatic Texturing from Aerial Images
Cesium 3DTiles Format Conversion
KML/COLLADA/ glTF Format Conversion
Cesium Heightmap 1.0 & Quantized-mesh 1.0 Format Conversion
Web-Based Visualization
Visualization on Unity Game Engine

Figure 3.4 The overall workflow of the methodology.

# 4. IMPLEMENTATION OF THE 3D CITY MODEL

## 4.1 Photogrammetric Triangulation

Exterior orientation parameters have been measured during the photogrammetric flight mission using GNSS (Global Navigation Satellite System) antenna and INS (Inertial Navigation System) and used in the photogrammetric triangulation process as weighted observations. The selection and measurement of the intial tie points in strip and cross-strip directions have been done manually by operators, and further tie points have been measured automatically with Intergraph Z / I Automatic Triangulation module. Thus, the degree of freedom in the bundle block adjustment has been increased. The measured image coordinates are adjusted by the Intergraph Z / I Automatic Triangulation module with self-calibration. Blunders have automatically been removed by statistical checks during the adjustment. Additional parameters have been included to model systematic errors such as atmospheric refraction, Earth curvature, and and camera distortion parameters during the adjustment.

The project area has been splitted into seven sub-blocks for process optimization. The number of images in each block, the number and properties of the GCPs, size of the block area and the accuracy measures (sigma naught of the adjustment and the standard deviations of GCPs in image space) are shown in Table 4.1.

| **Sub-block no** | **No. Of images** | **Number and features of GCPs** | | | **Sigma Naught (microns)** | **GCP image $\sigma_x$ (microns)** | **GCP image $\sigma_y$ (microns)** |
|---|---|---|---|---|---|---|---|
| | | **Full (XYZ)** | **Planimetric (XY)** | **Height (Z)** | | | |
| 1 | 424 | 34 | - | - | 1.9 | 1.3 | 1.0 |
| 2 | 661 | 51 | - | 1 | 1.7 | 1.1 | 0.8 |
| 3 | 287 | 22 | - | - | 1.5 | 0.9 | 0.7 |
| 4 | 745 | 55 | 3 | - | 1.6 | 1.0 | 0.8 |
| 5 | 751 | 61 | - | - | 1.8 | 1.1 | 0.9 |
| 6 | 807 | 62 | - | 1 | 1.2 | 1.0 | 0.8 |
| 7 | 793 | 61 | - | 1 | 1.4 | 0.8 | 0.7 |

Table 4.1 Photogrammetric block properties and triangulation results

## 4.2 Building Footprint and Attribute Generation from Basemaps

Although aerial photographs contain almost all visible information related to the surface of the project area, they cannot be compared to traditional vector maps before they are interpreted by humans or computers. Any good map user can hardly understand and interpret aerial photographs directy and image interpretation for map production is still an expert task. A vector map is an abstraction of selected details of the scene/project area. Figure 4.1 shows an overlay of an aerial orthoimage and manually extracted features stored as vectors. A vector map involves points, lines and polygons and stored in files or databases. In addition, maps contain symbols and text for as further information. Interpretation, analysis, implementation, and use of vector map information is easier than orthophotos. All vector detail and attribute information in vector maps are gathered by models from the stereo analysis so that they are clearer, understandable and interpretable. Interpretation of orthophoto images, application to the land and identification of the details on it requires special knowledge, experience, and expertise.

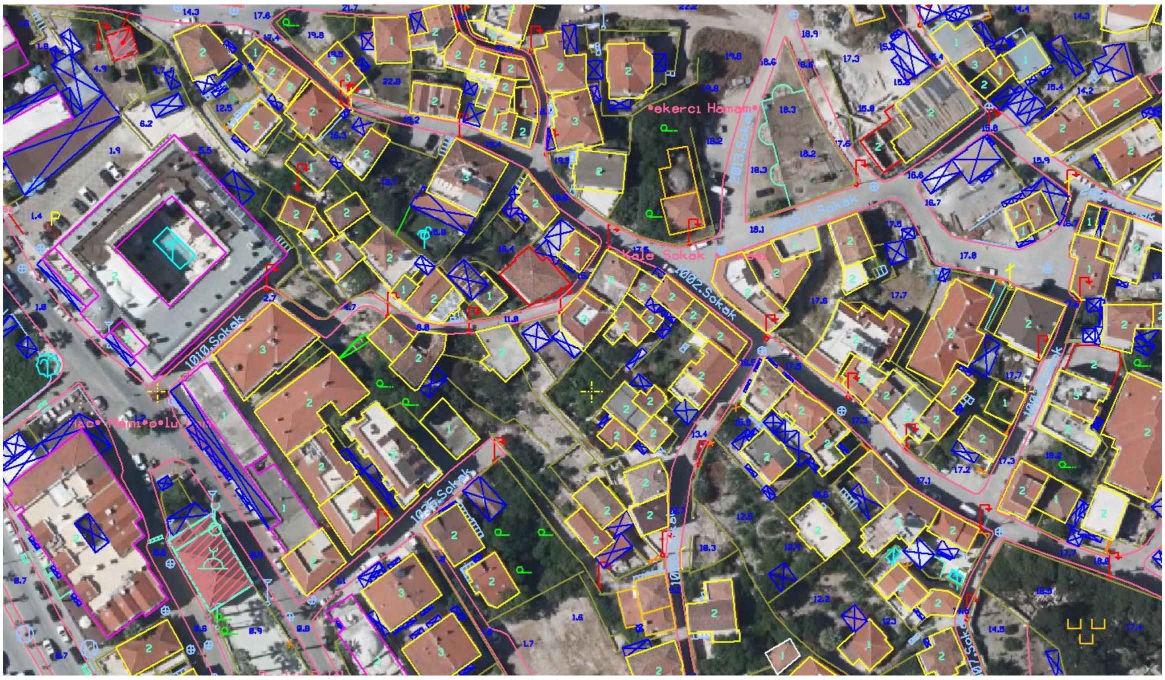


Figure 4.1 Manual feature extraction from stereo aerial images

Terrain elevation information is also produced during the processing of aerial photos, and contour lines, DTM grid points, break-lines and steep slopes are measured manually and semi-automatically. Features showing the terrain characteristics, such as creeks, ridges, etc., are also collected and stored as vector datasets.

As the next step, orthoimage mosaics are produced using the DTM grid points integrated with the contour lines and terrain characteristic features.

BuildingReconstruction software, which is used for 3D city model generation in this study, requires building footprints and attributes as input in ESRI Shapefile (.shp) format. In Turkey, basemaps produced in aerial photogrammetric projects are stored in CAD (Computer Aided Design) formats as standard and every project generally contains more than 200 layers. The data and information stored in these layers are usually adequate for extracting the building footprints and a few types of attributes. The building footprints can be considered as roof boundaries since it is usually not possible to see the building footprints on the ground in aerial images. The buildings are collected and stored in different CAD layers such as residential, commercial, public, school, etc. according to their use types. In addition to the buildings, other structures such as sundries, lighting poles or roads are also measured by operators.

In BuildingReconstruction software, it is required to store the building geometries and attributes in a single ESRI Shapefile. These geometries and attributes will later be converted into CityGML format while exporting the building geometry after the reconstruction is completed. The attributes can be used to implement queries on the reconstructed city model. But Microstation design files are just vector data and don’t have any attribute information. While converting this data into shapefile, also some attributes about the buildings were generated using transformers of the FME software (Figure 4.2).

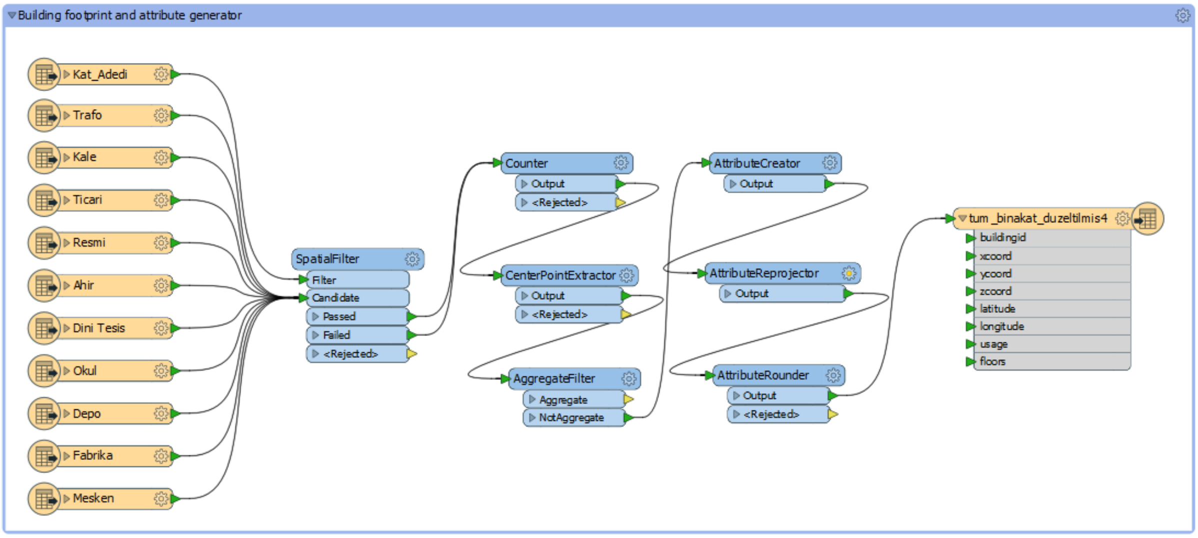


Figure 4.2 FME workbench for generating attributes and converting DGN file into Shapefile.

The number of the building floors is located inside the building area and saved as text in a different layer. The "SpatialFilter" transformer of FME is used to transform these numbers into attribute information of buildings. The “SpatialFilter” transformer compares two sets of features to see if their spatial relationships meet the selected test conditions. Every Candidate feature enters a spatial test that is specified by user. If the feature passes the given test, they output through "Passed" port while other features outputs through "Failed" port. A number of floors” attribute given as “Filter”, building footprints given as “Candidate”, and “Spatial Predicates to Test” value assigned as “Filter is Within Candidate” to the transformer. So if a number (text) is located inside a polygon, it is assigned as attribute to that polygon (i.e. building).

The BuildingReconstruction software expects each building to have a unique ID number in the selected attribute column. While it can be done manually for small areas, it is almost impossible to assign IDs to thousands of buildings manually. For this reason, the unique ID is automatically assigned to each building footprint "Counter" transformer of FME. This transformer generates a unique numeric attribute for the feature. This transformer is useful for assigning unique, numeric ID number to a set of features. It can also be used to count number of features or fast attribute generation for given data. The unique ID for each building is written in the attribute table of the generated shapefile file.

“CenterPointExtractor” transformer calculates the 3D coordinates of the feature's selected point and gives as an output. This point can be either in the center of the feature's bounding box, some random place in bounding box or at the center of mass. Using this transformer, center coordinates of building footprints are extracted and written as three separate attributes to the shapefile.

“AttributeReprojector” reprojects coordinates stored as attributes from one coordinate system to another and the $X$ and $Y$ (Easting and Northing) coordinates of the building footprints have been reprojected to geographical coordinate system on WGS84 datum and saved as two new attributes named “Latitude” and “Longitude”. These attributes have later been used in the web interface for displaying the buildings in Google StreetMap.While producing the basemaps, every type of structure is stored in a different layer with a specific name. Thus, the layer names from which the buildings collected and converted has been added as attribute information to that building.

As a last step, all building footprints have been converted to ESRI Shapefile format with 8 attributes and become ready to be input to the BuildingReconstruction software. A part of the project area with all CAD layers and the building footprints extracted thereof are given in Figure 4.3.

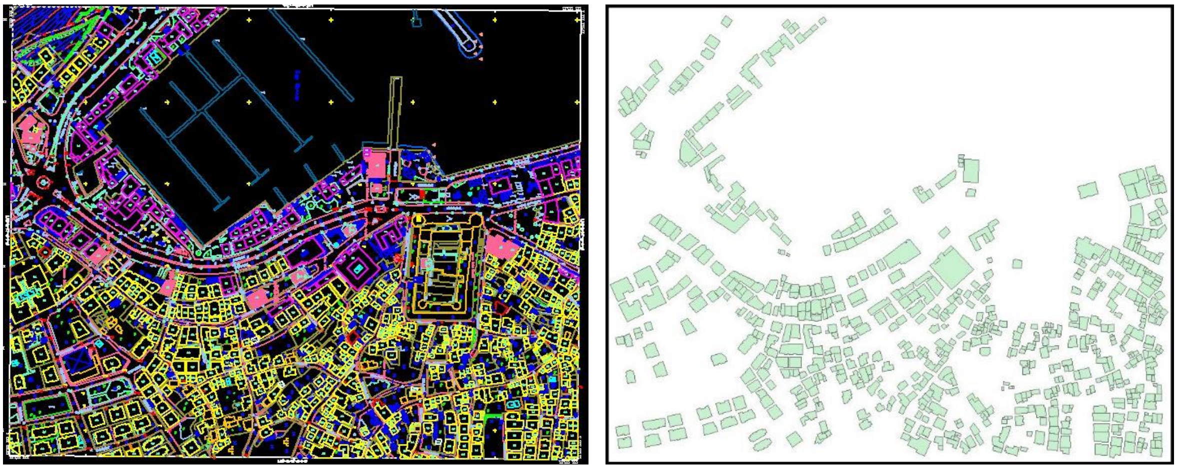
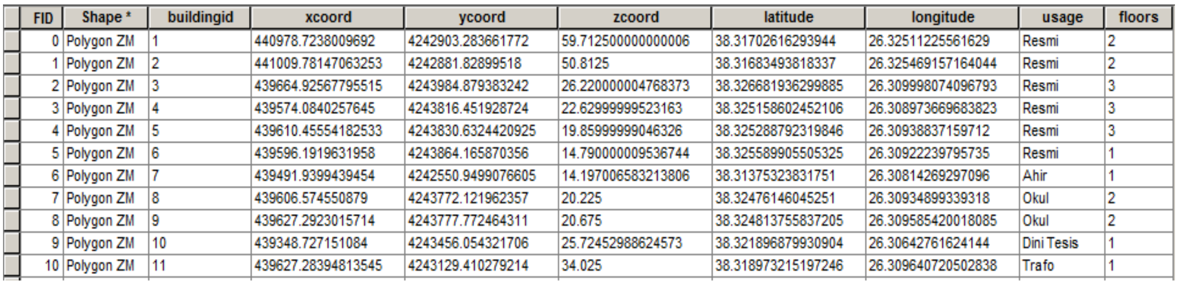


Figure 4.3 A part of the basemap with all CAD layers (left) and the building footprints exracted from it (right).



| | FID | Shape * | buildingid | xcoord | ycoord | zcoord | latitude | longitude | usage | floors |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | Polygon ZM | 1 | 440978.7238009692 | 4242903.283661772 | 59.712500000000006 | 38.31702616293944 | 26.3251122561629 | Resmi | 2 |
| | 1 | Polygon ZM | 2 | 441009.78147063253 | 4242881.82899518 | 50.8125 | 38.31683493818337 | 26.325469157164044 | Resmi | 2 |
| | 2 | Polygon ZM | 3 | 439664.92567795515 | 4243984.879383242 | 26.220000004768373 | 38.326681936299885 | 26.309998074096793 | Resmi | 3 |
| | 3 | Polygon ZM | 4 | 439574.0840257645 | 4243816.451928724 | 22.62999999523163 | 38.325158602452106 | 26.308973669683823 | Resmi | 3 |
| | 4 | Polygon ZM | 5 | 439610.45554182533 | 4243830.6324420925 | 19.85999999046326 | 38.325288792319846 | 26.30938837159712 | Resmi | 3 |
| | 5 | Polygon ZM | 6 | 439596.1919631958 | 4243864.165870356 | 14.790000009536744 | 38.325589905505325 | 26.3092239795735 | Resmi | 1 |
| | 6 | Polygon ZM | 7 | 439491.9399439454 | 4242550.9499076605 | 14.197006583213806 | 38.31375323831751 | 26.30814269297096 | Ahir | 1 |
| | 7 | Polygon ZM | 8 | 439606.574550879 | 4243772.121962357 | 20.225 | 38.32476146045251 | 26.30934899339318 | Okul | 2 |
| | 8 | Polygon ZM | 9 | 439627.2923015714 | 4243777.772464311 | 20.675 | 38.324813755837205 | 26.309585420018085 | Okul | 2 |
| | 9 | Polygon ZM | 10 | 439348.727151084 | 4243456.054321706 | 25.72452988624573 | 38.321896879930904 | 26.30642761624144 | Dini Tesis | 1 |
| | 10 | Polygon ZM | 11 | 439627.28394813545 | 4243129.410279214 | 34.025 | 38.318973215197246 | 26.309640720502838 | Trafo | 1 |

Figure 4.4 Attribute table of the generated building footprints

## 4.3 Digital Terrain Model (DTM) Generation

DTMs are 3D representations of the terrain elevations found on the bare earth’s surface. They are usually generated from data collected by airborne LiDAR, photogrammetry, SAR Interferometry methods or extracted from existing vector data such as contour lines and breaklines. DTMs are becoming important base data for a great number of applications in many engineering fields.

In this work, the DTMs are required to make sure that buildings are located correctly on the terrain, not under or above of it. It is also used to calculate the base height of a building from the terrain. The resolution requirements for the DTMs are substantially lower than DSMs in the methodology proposed here. A grid resolution between 1 m and 5 m is usually sufficient. The general recommendation from the BuildingReconstruction software is to use a grid interval of 1 meters. The BuildingReconstruction software works best with the tile-based raster data since raw LiDAR data can be very massive, and it is recommended by the software vendors to create grid DTMs from LiDAR data and store them in ESRI ASCII Grid Format. Figure 4.5 shows an overview of the manually extracted elevation data and a zoomed view of it.

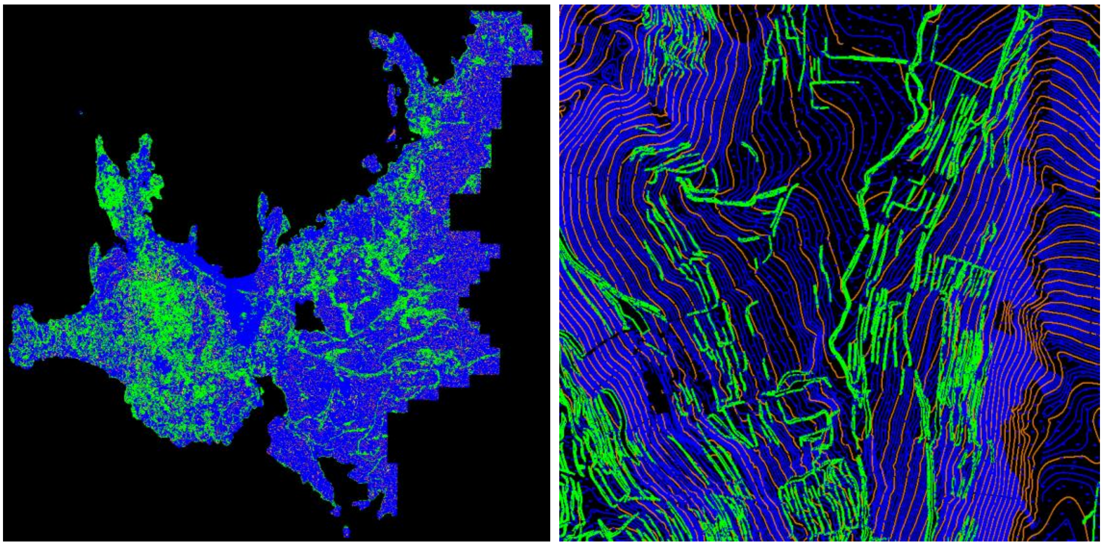


Figure 4.5 General view of the manually extracted elevation data (left) and close view (right).

Within the photogrammetric flight project, contour lines with different intervals (i.e. 1m, 2m, 5m, 10m), breaklines and grid DTM points have been manually measured using Bentley MicroStation [88] software by photogrammetry operators. More than 1.2 million height points, thousands of contour lines and break lines merged into one single file at the data preparation stage for building reconstruction, and converted into a single 1-meter resolution raster DTM using FME 2018 software. The software has a translator called “RasterDEMGenerator” which takes contour lines, break lines and points as input and constructs TIN with Delaunay approach, then converts TIN into intented raster formats.

A grid DTM is then interpolated from the TIN and stored in raster format. The output data resolution is specified by “Output DEM Cell Spacing” parameter, then the software produces a single raster file consist of equidistant, 3D height points arranged in the grid structure. For this project, a single DTM file covering the whole project area (260 km$^2$) with 1-meter resolution is generated using FME 2018. The FME workbench for raster DTM generation is depicted in Figure 4.6.

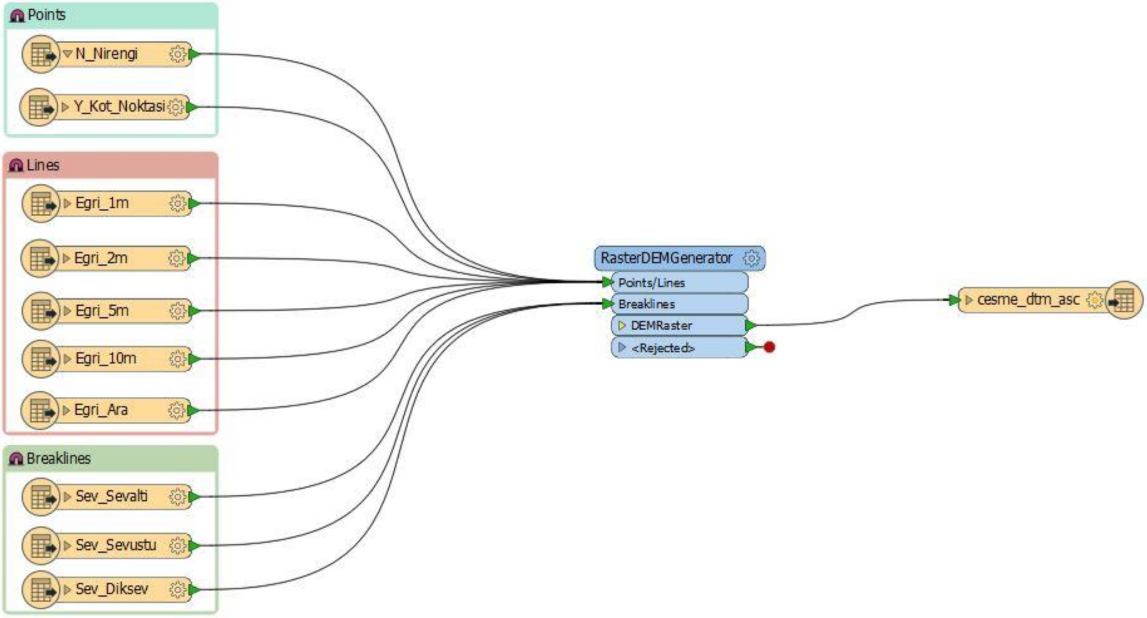


Figure 4.6 FME workbench for raster DTM generation

One of the important point while generating DTM is checking the parameter “No data Values”. All output raster cells that are not calculated while generating terrain model will be assigned the value of this parameter. If this parameter left blank it is interpreted as NaN (Not a Number), otherwise, these values will be assigned to given value. After fixing null values, generated raster DTM has become ready to be input for BuildingReconstruction software and for visualizing on the web. Figure 4.7 shows the DTM with and without null values.

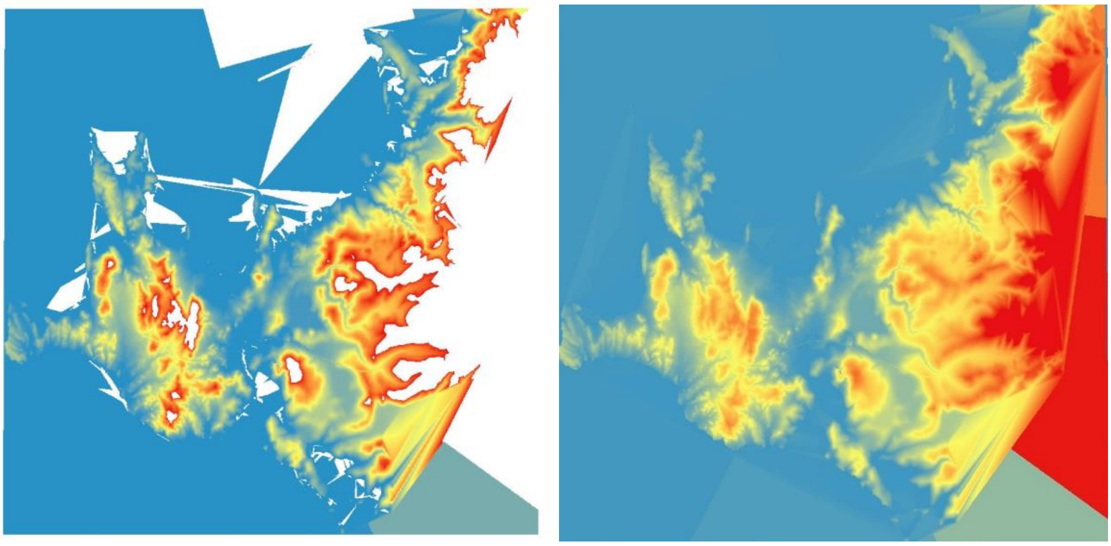


Figure 4.7 DTM with null values (left) and null values fixed (right)

## 4.4 Digital Surface Model (DSM) Generation

For the DSM generation, a number of different software has been investigated to obtain an optimum balance between the DSM quality and production speed. Taking these two parameters into consideration, a high-resolution DSM was generated using Agisoft Photoscan Pro [86]. The most important reason for choosing this software is that it allows parallel processing by allowing GPU usage along with the CPU during the DSM generation. Using the GPU at the process led to a dramatic drop in the processing time and let the total time needed to produce the DSM to become one hour on the personal computer specified in Table 3.1. Another reason for choosing the Agisoft Photoscan Pro software was its ability to export point clouds and textured mesh models in Cesium 3D Tiles format directly, which was later used to visualize the data in 3D on the web without the need for a format conversion.The DSM of the project area has been generated in 7 sub-blocks as these blocks have already been formed for the triangulation. These sub-blocks are depicted in Figure 4.8. Since the BuildingReconstruction can reconstruct a maximum of a couple of thousand buildings in a single process, the larger blocks have again been divided into more sub-blocks to decrease the processing area and time. While generating DSMs for the sub-block parts, stereo aerial photographs along with adjusted exterior orientation parameters and GCP coordinates were used for acquiring best result at reconstruction and georeferencing.

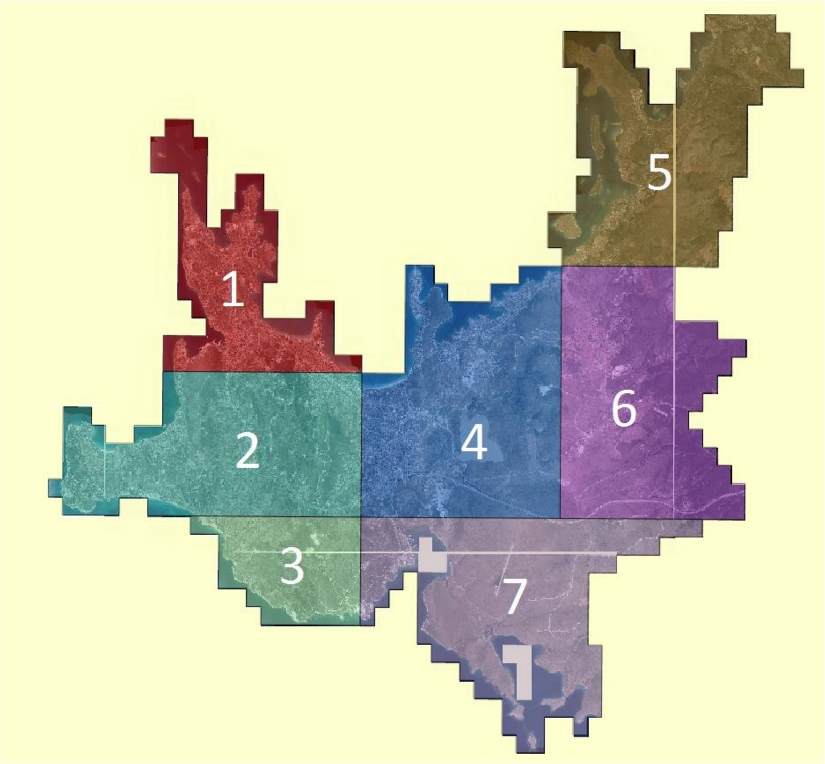


Figure 4.8 Photogrammetric blocks generated for the area

Camera interior orientation parameters have been used as input in the software for camera calibration. Aerial images and the GCPs in the project area have also been added to the project. While selecting the aerial photographs in every sub-block part, it has been taken consideration that buildings near the outer boundary of the project area should be visible at least 6 or more photographs for better reconstruction. To achieve this, image footprints have been used to create a buffer zone at the outer boundary of the production area. Also when choosing GCPs in every sub-block part, it has been taken care that every GCP appears in 6 or more photographs for better georeferencing.

For the DSM production in Agisoft Photoscan, the GPU usage has been activated from the Software Settings. Photos of the project area have later been imported into the software. A text file containing adjusted exterior orientation parameters of all photos must be prepared before starting the process. These parameters have been imported into the software with respective reference coordinate system. While importing the parameters, the type of the units (yaw-pitch-roll or omega-phi-kappa) or units (degree – grad) should be noted. Finally, GCPs adjusted ground coordinates have been imported and the accuracy of these parameters have been specified using the software GUI before starting the DSM generation.

At "Align Photos" stage, the software matches tie points between stereo images and estimates camera location for every photo, then generates sparse point cloud. Within this study, "Highest" accuracy has been selected for aligning the photos (i.e. tie point generation), which takes a longer time to process but gives more accurate camera positions. After this process is completed, ground control points have been measured on photos. Using GCPs together with the exterior orientation parameters leads better reconstruction results and precise georeferencing of images. After measuring all GCPs on the images, we can start "Dense cloud" generation process. Based on the established camera orientations, the software generates a dense point cloud (i.e. DSM). Figure 4.9 depicts the tie points and the dense point cloud generated in one part.

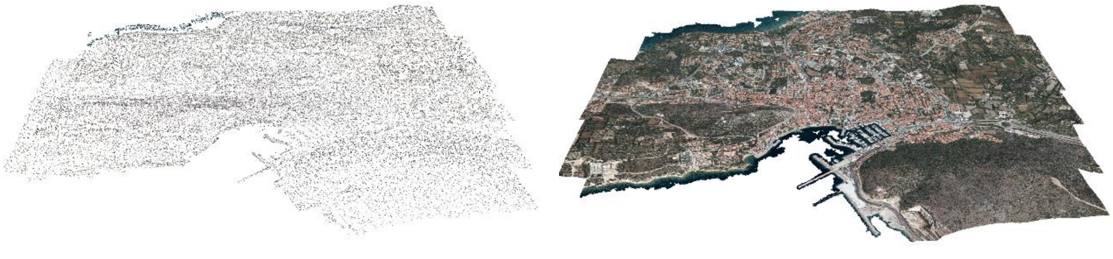


Figure 4.9 Generated Tie points (left) and dense point cloud (right)

Dense point clouds in Agisoft Photoscan can be generated in 5 different resolutions such as 1px, 2px, 4px, 8px and 16 px. Higher quality requires more hardware resources and have longer processing times. Lower quality has lower processing times and requires less computational resources. A DSM with a resolution of 25cm is sufficient for the BuildingReconstruction software to generate building models with optimum accuracy. Therefore, the DSM production has been performed at a resolution of 4 pixels (30cm) considering both production speed and accuracy criteria. After the dense cloud has been reconstructed, it has been converted into TIN and grid DSMs.

The software allows users to generate and visualize both point cloud and raster DSMs. A raster DSM is a grid-based representation of point cloud surface model with height values. They can be generated from different sources such as dense cloud, sparse cloud or mesh model. The accuracy of the DSMs increase with the point cloud density. The DSM density is specified with the pixel size parameter. The grid DSMs can be stored in different formats such as GeoTIFF or ESRI Grid format. The optimal results with the BuildingReconstruction software are achieved with a DSM between 25cm and 50cm resolution for LoD2 buildings. Considering this fact, the DSMs have been generated at a resolution of 30cm which covers the whole project area. These DSMs have afterwards been clipped according to the bounding box of the production area and converted into ESRI ASCII Grid format for being input to the BuildingReconstruction software. An overview and a zoomed view of the generated DSMs from one part is shown in Figure 4.10.

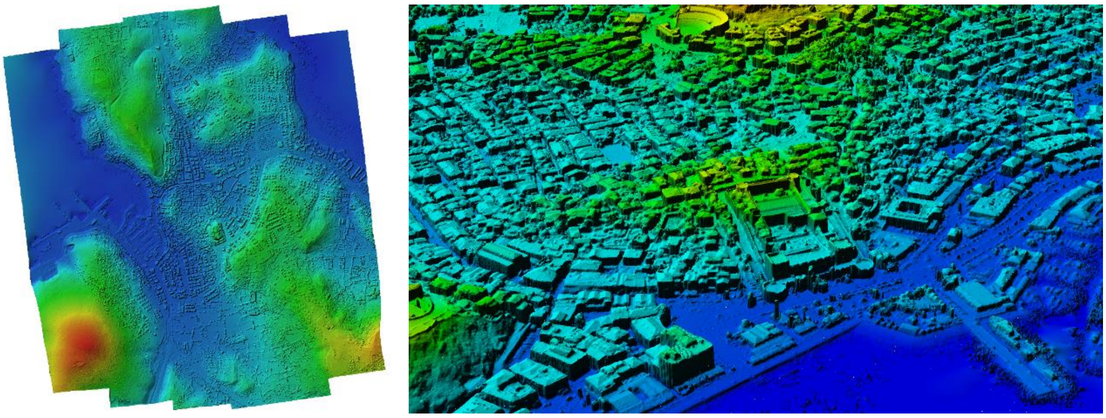


Figure 4.10 An overview (left) and a close view (right) of the generated DSM.

## 4.5 Automated Building Reconstruction

In this study, BuildingReconstruction 2018 software has been used for automatic reconstruction of buildings for generation of 3D city model in CityGML format. The BuildingReconstruction 2018 software can reconstruct CityGML-based 3D city models from raw GIS data. It enables automatic generation and manual editing of building models in LoD1 and LoD2. BuildingReconstruction reconstruct 3D building models in city scale efficiently based on DSMs generated from different sources such as LiDAR or photogrammetry, existing DTMs, and building footprints. The main advantage of this method is automatic reconstruction of the 3D city models along with the rich semantic information and thematic attributes of the buildings. BuildingReconstruction 2018 software works best with tile-based input files. If an entire region is required to be reconstructed, users should generate tiles with 1 km² or smaller sizes with the footprint data, laser data and orthophotos. The general workflow of the BuildingReconstruction software is given in Figure 4.11.

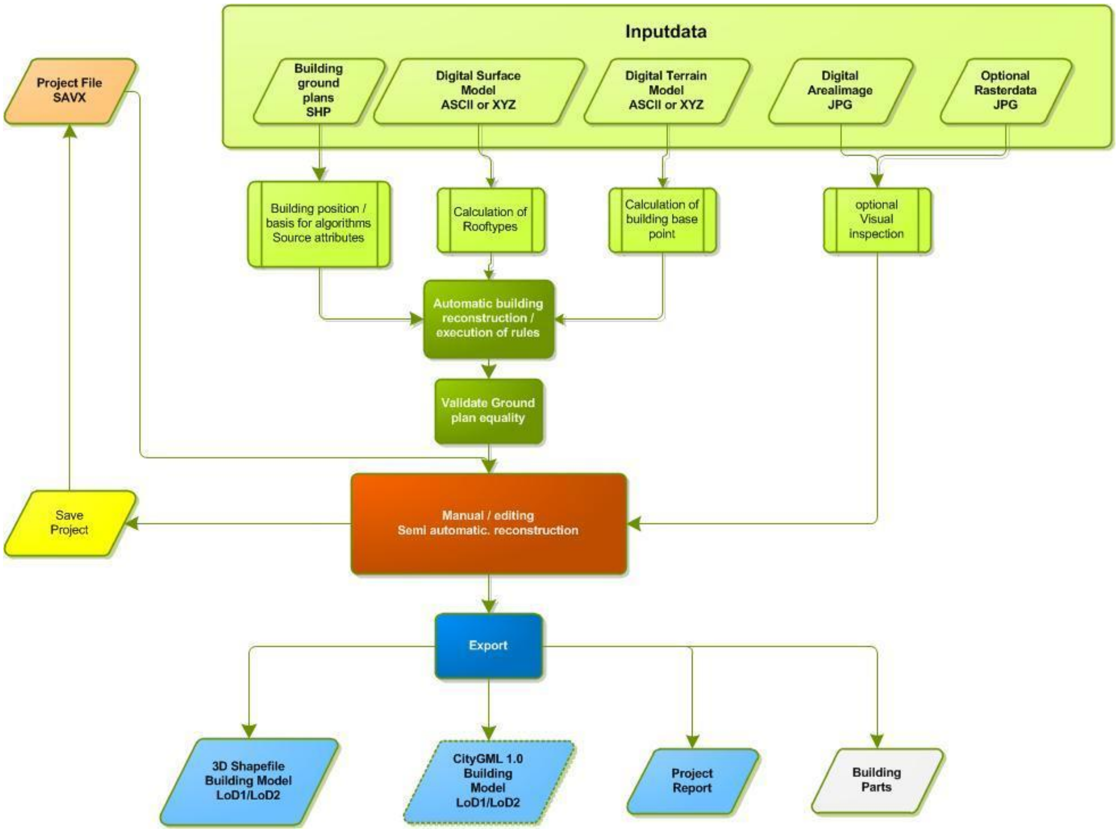
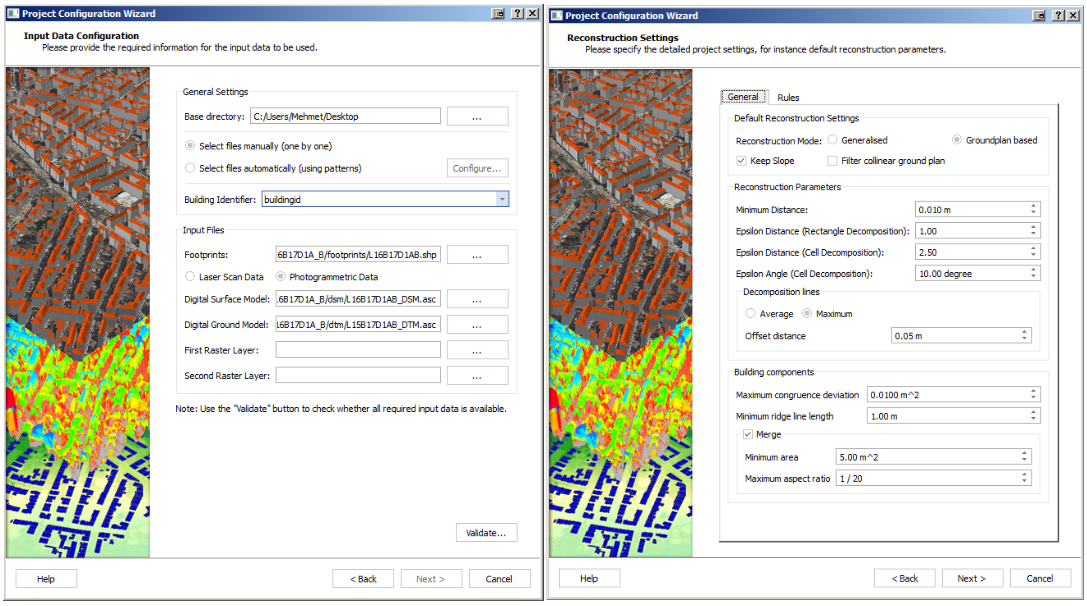

Inputdata
Project File SAVX
Building ground plans SHP
Digital Surface Model ASCII or XYZ
Digital Terrain Model ASCII or XYZ
Digital Arealimage JPG
Optional Rasterdata JPG
Building position / basis for algorithms Source attributes
Calculation of Rooftypes
Calculation of building base point
optional Visual inspection
Automatic building reconstruction / execution of rules
Validate Ground plan equality
Save Project
Manual / editing Semi automatic. reconstruction
Export
3D Shapefile Building Model LoD1/LoD2
CityGML 1.0 Building Model LoD1/LoD2
Project Report
Building Parts

Figure 4.11 BuildingReconstruction 2018 software workflow [89].

The 3D building geometries are reconstructed automatically in CityGML LoD1 or LoD2. A model-driven approach was used in BuildingReconstruction software to detect and reconstruct the general roof geometry and roof type using a roof library containing 32 types of roof geometry popularly used in urban areas around the world. For every given polygon in input footprint shapefile, a discrete 3D building geometry is reconstructed. This reconstructed building's geometry is ensured to be fully closed without any holes or gaps between surfaces, and also be fully compatible with given building footprint. Along with the automated reconstruction, the software offers an interactive editor. Manual editing on the building geometries can be done using this tool. The city model can be exported in two different formats: CityGML and ESRI Shapefile. Both of these formats can store semantic information about buildings. In addition to the building models, BuildingReconstruction allows adding attributes by enriching the semantic data and by calculating different building properties such as building height, roof type, roof slope which can be used in analysis processes. In Figure 4.12, two screenshots from the GUI of BuildingReconstruction 2018 software for project configuration are given. Figure 4.13 shows further views from the software GUI.


Project Configuration Wizard

**Input Data Configuration**
Please provide the required information for the input data to be used.

General Settings
Base directory: C:/Users/Mehmet/Desktop
● Select files manually (one by one)
○ Select files automatically (using patterns)
Building Identifier: buildingid

Input Files
Footprints: 6B17D1A_B/footprints/L16B17D1AB.shp
○ Laser Scan Data ● Photogrammetric Data
Digital Surface Model: .6B17D1A_B/dsm/L16B17D1AB_DSM.asc
Digital Ground Model: l6B17D1A_B/dtm/L15B17D1AB_DTM.asc
First Raster Layer:
Second Raster Layer:

Note: Use the "Validate" button to check whether all required input data is available.

Project Configuration Wizard

**Reconstruction Settings**
Please specify the detailed project settings, for instance default reconstruction parameters.

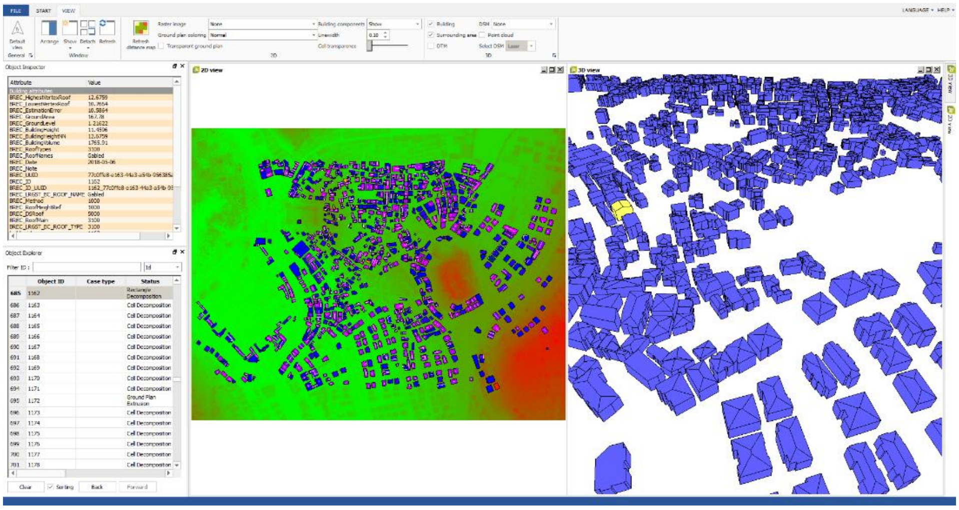
Default Reconstruction Settings
Reconstruction Mode: ○ Generalised ● Groundplan based
☑ Keep Slope ☐ Filter collinear ground plan

Reconstruction Parameters
Minimum Distance: 0.010 m
Epsilon Distance (Rectangle Decomposition): 1.00
Epsilon Distance (Cell Decomposition): 2.50
Epsilon Angle (Cell Decomposition): 10.00 degree
Decomposition lines
○ Average ● Maximum
Offset distance 0.05 m

Building components
Maximum congruence deviation 0.0100 m^2
Minimum ridge line length 1.00 m
☑ Merge
Minimum area 5.00 m^2
Maximum aspect ratio 1 / 20

Figure 4.12 BuildingReconstruction 2018 project configuration wizard

BuildingReconstruction 2018 achieves optimal results from DSMs with grid intervals between 25 cm and 50 cm (16 ppm and 4 ppm). 4ppm/ 50cm data is generally sufficient to fulfill all requirements for LoD2. With a data density of 16ppm/ 25cm, the optimal result can be achieved with the reconstruction algorithms (with a minimally slower automatic reconstruction process). Working with a higher data density in BuildingReconstruction 2018 does not yield any further improvements in the automatic result, but it does adversely affect the overall performance of the software.



Raster image: None
Ground plan coloring: Normal
☐ Transparent ground plan
Building components: Show
Linewidth: 0.10
Cell transparency
☑ Building
☑ Surrounding area
☐ DTM
DSM: None
☐ Point cloud
Select DSM: Laser

| Attribute | Value |
|---|---|
| Building attributes | |
| BREC_HighestVertexRoof | 12.6759 |
| BREC_LowestVertexRoof | 10.7654 |
| BREC_EstimationError | 10.5864 |
| BREC_GroundArea | 167.78 |
| BREC_GroundLevel | 1.21622 |
| BREC_BuildingHeight | 11.4506 |
| BREC_BuildingHeightNN | 12.6759 |
| BREC_BuildingVolume | 1765.91 |
| BREC_RoofTypes | 3100 |
| BREC_RoofNames | Gabled |
| BREC_Date | 2018-05-06 |
| BREC_Note | |
| BREC_UUID | 77c0ffe8-e163-44a3-a54b-096385 |
| BREC_ID | 1162 |
| BREC_ID_UUID | 1162_77c0ffe8-e163-44a3-a54b-9 |
| BREC_LRGST_BC_ROOF_NAME | Gabled |
| BREC_Method | 1000 |
| BREC_RoofHeightRef | 1000 |
| BREC_DGRoof | 9000 |
| BREC_RoofMain | 3100 |
| BREC_LRGST_BC_ROOF_TYPE | 3100 |

Filter ID : Id

| | Object ID | Case type | Status |
|---|---|---|---|
| 685 | 1162 | | Rectangle Decomposition |
| 686 | 1163 | | Cell Decomposition |
| 687 | 1164 | | Cell Decomposition |
| 688 | 1165 | | Cell Decomposition |
| 689 | 1166 | | Cell Decomposition |
| 690 | 1167 | | Cell Decomposition |
| 691 | 1168 | | Cell Decomposition |
| 692 | 1169 | | Cell Decomposition |
| 693 | 1170 | | Cell Decomposition |
| 694 | 1171 | | Cell Decomposition |
| 695 | 1172 | | Ground Plan Extrusion |
| 696 | 1173 | | Cell Decomposition |
| 697 | 1174 | | Cell Decomposition |
| 698 | 1175 | | Cell Decomposition |
| 699 | 1176 | | Cell Decomposition |
| 700 | 1177 | | Cell Decomposition |
| 701 | 1178 | | Cell Decomposition |

☑ Sorting

Figure 4.13 BuildingReconstruction graphical user interface

The required input data for the BuildingReconstruction software are as follows:

- Digital Surface Model (DSM) with between 25-50 cm resolution as ASCII Grid or XYZ Point Cloud
- Digital Terrain Model with 1-meter resolution as ASCII Grid or XYZ Point Cloud
- 2D building footprints with intended attributes (polygon with a unique ID)
- Digital Orthophoto for visual control with minimum 40cm resolution

Figure 4.14 shows screenshots of one DSM and one DTM from the project area. The DSMs are used to determine roof shape and the properties of the roof, whereas the DTMs are used to calculate the height and the ground surface of the building. Footprints are used to define the areas in which the reconstruction is performed, and they are used to determine the geometric structures of the walls in the building model. The software also supports processing without a DTM and if no terrain model available, the base height of a building is determined from a buffer in which the lowest point around a building is determined. But this only works in areas where building density is low. For areas which have high building density, the terrain points are often insufficient to calculate the DTM.

Since the BuildingReconstruction software uses model-driven approach to reconstruct building and its parts from input data, it is suitable for working with DSMs having low point density as well. Currently, BuildingReconstruction supports 32 different roof types for reconstructing 3D building geometries. The buildings with the simple roof types, roof heights, and roof types can be in general reconstructed more accurately than complex ones. This problem arises because the software cannot match the complex roof types with the roof models in its library. Also, small parts on the roofs are mostly ignored by the software. These kinds of roofs don’t exist in its library because there can be unlimited variations of them. The main advantage of the model-driven method is reconstructing models without any gaps or holes between surfaces with correct geometry. This method's production speed is slightly better than other methods. The main handicap of this method is the requirement of a very comprehensive roof library for comparing roof geometries generated from given DSM.

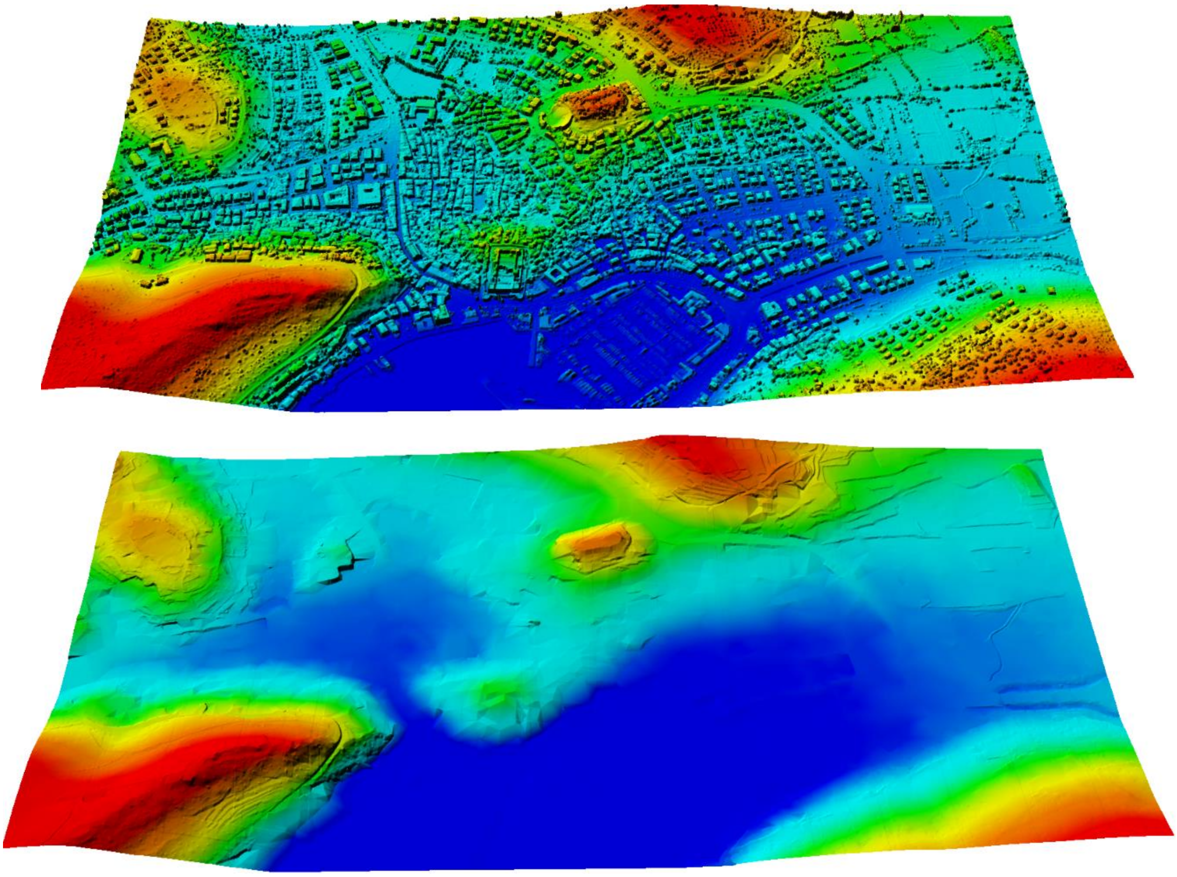


Figure 4.14 25cm resolution DSM and 1m resolution DTM input for BuildingReconstruction software

The project area is split into several tiles based on the overall extent of the available input data. For each tile, the input data is loaded into the BuildingReconstruction software. Building outlines/footprints help the software to identify the location of single buildings. All building outlines/footprints should be delivered as closed polygons. Every footprint will be mapped onto a separate 3D building model. The input DSM is analyzed for each building footprint to detect the roof type using the following three algorithms: Rectangle intersection, Cell decomposition, and Footprint Extrusion. Simple geometries such as rectangular buildings with gabled or hipped roofs are processed with rectangle intersection. The software creates several intersecting rectangles and squares from the basis of the footprint to construct buildings in 3D. An analysis of point clusters on the surfaces of the roof is taken from the digital height model to create a solid geometry.

The cell decomposition algorithm is used for buildings with multiple roof forms and/or with different heights. The building is divided into several cells lying within the building footprint. A built-in library of more than 32 main and connecting roof types is used to detect the best fitting roof type for each generated cell. Neighboring cells are analyzed to connect the roof geometries in order to reconstruct even complex roof shapes. Cells can be merged or even deleted to create a more accurate 3D building model. The footprint extrusion algorithm is used to create city-wide LoD1 models. In addition, this reconstruction method can be used for flat LoD2 buildings. To accomplish this, the building height is set to the height where the largest number of points is found; in this way, a false building height, for instance, due to chimneys, is avoided. This process ensures that roofs do not overshoot the building footprint and thus guarantees compliance with the official 2D basemap. The DTM used to calculate the height of the building's ground surface. This guarantees that the building perfectly grounded on the digital terrain model rather than floating over or sinking into it. Figure 4.15 shows an example from the automatically generated buildings with BuildingReconstruction 2018 software from the project area.

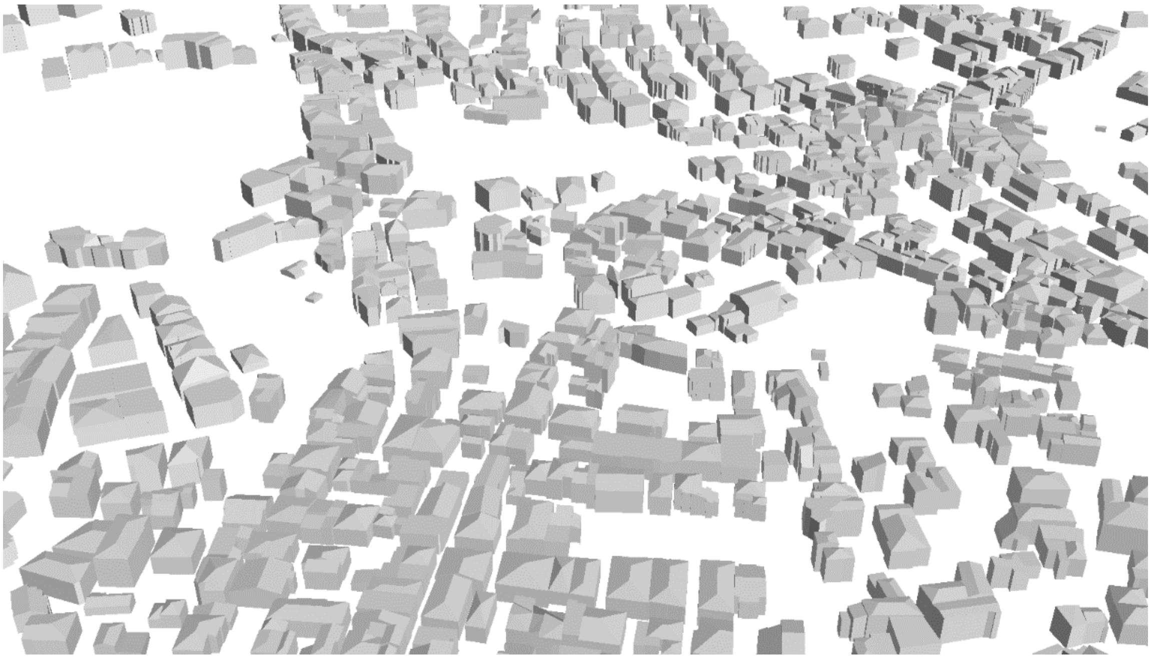


Figure 4.15 Automatically generated buildings with BuildingReconstruction 2018 software

After generating city model, manual editing of the buildings can be done from "Advanced Cell Editor" tab. It includes extensive editing functions for more accurate LoD2 buildings. If necessary, a particular roof shape and its orientation can be forced. In cell decomposition mode, cells can be reconstituted, deleted or merged. With complex roof shapes such as domed roofs, various parameters can be changed. As the last step before exporting city model, the semantic data need to be added correctly to the exported CityGML and Shapefile to make queries on GIS software or web interface.

Figure 4.16 shows a part of the automatically generated roof geometries overlayed on the orthophoto with 10 cm GSD. The reconstruction accuracy has been analysed by visual checks on 1000 buildings and the results are provided in Table 4.2. During the visual assessments, even if roof structures smaller than the grid interval have not been detected, they are considered to be true. In addition, if the main roof structure is modeled correctly, it is also considered as true.

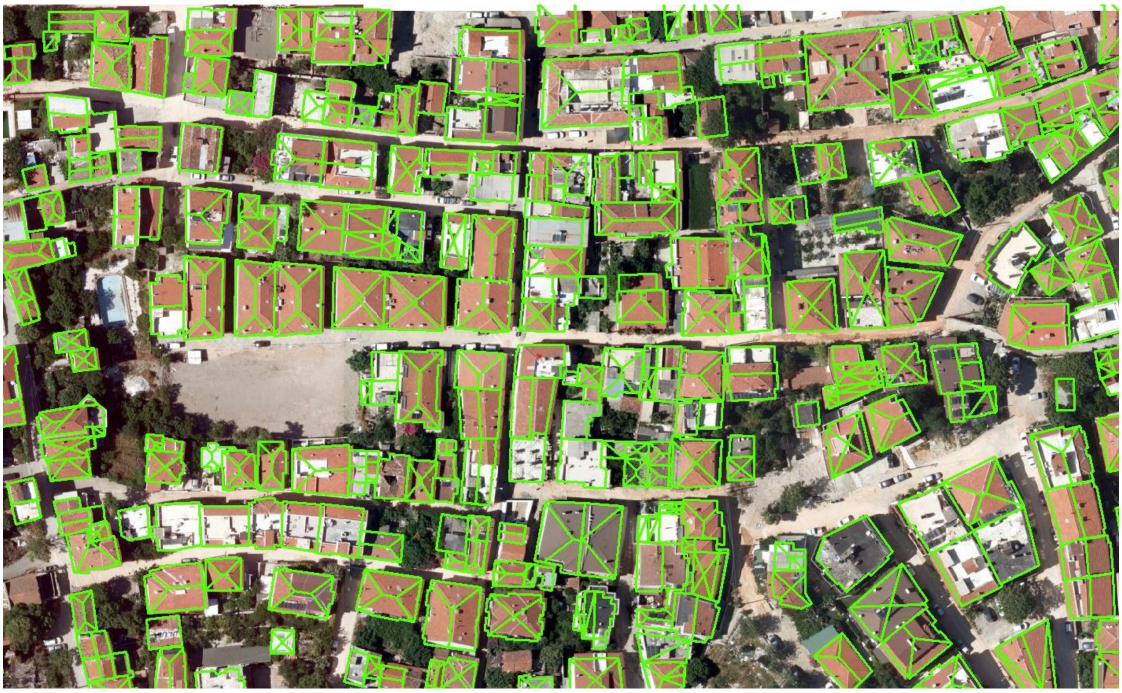


Figure 4.16 Automatically generated roof geometries overlayed on the orthophoto with 10 cm resolution.

| Reconstruction Accuracy of 1000 buildings generated by BuildingReconstruction 2018 software | | | |
|---|---|---|---|
| **DSM Resolution** | **Process Time** | **LoD1 Successfulness** | **LoD2 Successfulness** |
| **10 cm** | 6 min 50 sec | 100% | 73.8% |
| **25 cm** | 1 min 54 sec | 100% | 72.7% |
| **50 cm** | 1 min 2 sec | 100% | 65.4% |
| **100 cm** | 39 sec | 100% | 55.1% |

Table 4.2 Automatic reconstruction accuracy of 1000 buildings generated by BuildingReconstruction 2018 software evaluated by visual checks.

Advantages of the BuildingReconstruction software can be listed as follows:

- Large-area reconstruction of LoD1 and LoD2 building models
- Building models are suitable for texturing with nadir/oblique aerial images
- Direct export to CityGML and Shapefile with flexible attribute mapping
- Fast processing of 3D Buildings with semantic information
- Automatic calculation of attributes to enrich the building model
- High geometric accuracy with a ground plan based reconstruction

Disadvantages of the BuildingReconstruction software can be listed as:

- Reconstruction accuracy is limited to roof library due to a model-driven method
- Small roof parts are ignored while reconstruction
- Cannot reconstruct complex roof types
- The manual editing interface is not user-friendly

# 5. AUTOMATIC TEXTURING BUILDINGS

Automatic-texturing of buildings was carried out with CityGRID Texturizer software by UVM Systems GmbH [64]. The following sub-sections will elaborate on the details of model texturing.

## 5.1 CityGRID Texturizer

Existing 3D city models can be automatically textured from oriented images using CityGRID Texturizer. As a result, the attractiveness of the digital city model can be significantly improved with little effort. Oriented aerial photographs can be used for the texturing of roofs and facades, mobile mapping for the high-resolution texturing of street-side facades.Roof and terrain are textured automatically if a texture image together with its correct orientation parameters was imported into the data source and is assigned to the model. The facade texture is assigned interactively or fully automatic from oriented images. Terrain texture is always calculated online, whereas roof texture and facade texture is calculated for each Unit and saved with the Unit. Roof texture is derived automatically and can be calculated for each Unit separately, where the texture coordinates can be revised and corrected if the orientation parameters and/or the building model are not accurate enough) or for all Units of a Model in one step. Orthophotos of the facades are generated by simple affine transformation in the texture tool. The photos are loaded, assigned to the respective facade of the Unit and saved with the Unit in the data source. Automatic texturing workflow of CityGRID Texturizer is given in Figure 5.1.

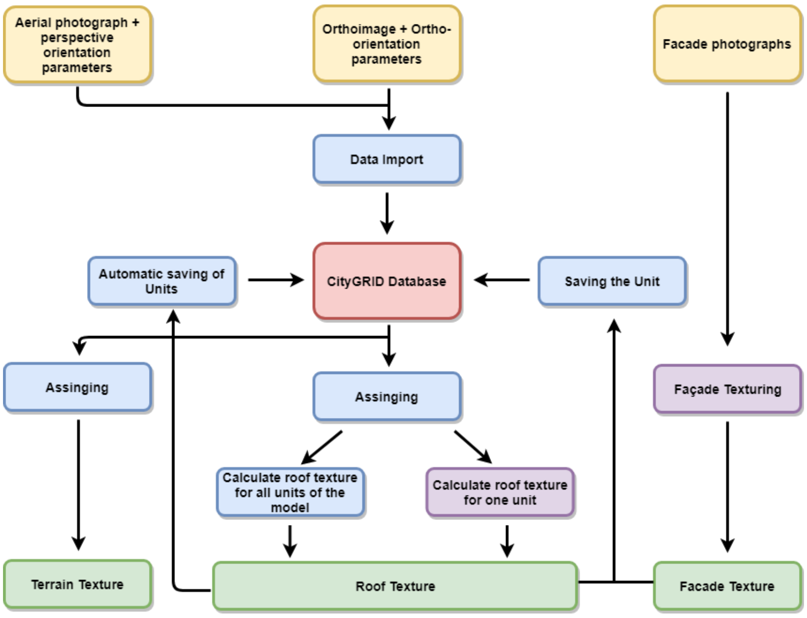

Aerial photograph + perspective orientation parameters
Orthoimage + Ortho-orientation parameters
Facade photographs
Data Import
Automatic saving of Units
CityGRID Database
Saving the Unit
Assinging
Assinging
Façade Texturing
Calculate roof texture for all units of the model
Calculate roof texture for one unit
Terrain Texture
Roof Texture
Facade Texture

Figure 5.1 CityGRID Texturizer automatic texturing workflow [64].

## 5.2 Image Acquisition and Radiometric Pre-Processing

There are several criteria for the acquisition of airborne images for automatic texturing. The accepted optimal resolution for automatic texturing is usually 10 cm or better. The flight altitude to obtain this resolution is between 800 and 2500 meters from terrain, but these values can range depending on different factors such as airport restrictions in the area, building heights, terrain elevation, camera type (e.g. camera format), camera parameters (in particular focal length), image acquisition method (vertical or oblique) and required image GSD. In this project, the images have been acquired from an altitude of 1500 meters with 8 cm resolution with the aim of production of basemaps for Çeşme City. In this type of acquisition, the vividness of the colors is not considered as an important aspect in the "Base Map" generation process, however in order to generate a fine texture the sharpness of images together with the vividness of colors is of a great importance. In order to obtain a more vivid and impressive texture, the images need to go through a color enhancement process. This process involves the utilization of scripts (in a photo editing software) for improving the realistic look of color tones in the texture. As a result of this process, the images (and textures/scenes composed of them) become more vivid, sharp and realistic. Figure 5.2 shows an example from the original images and the color-enhancement result.

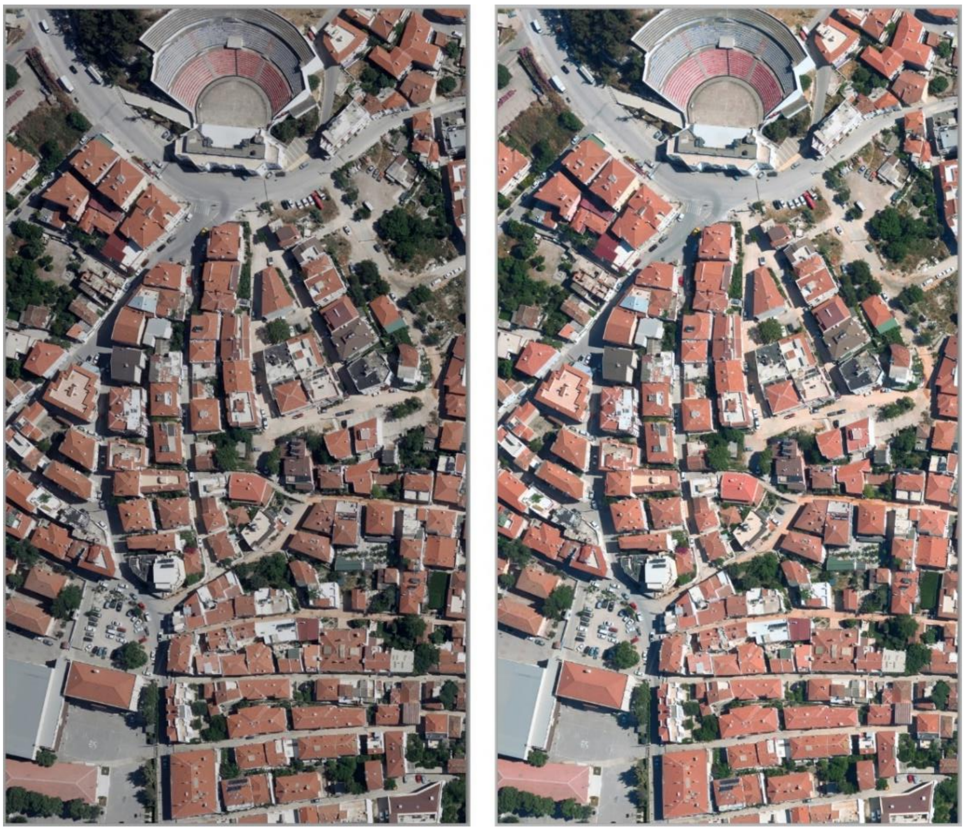
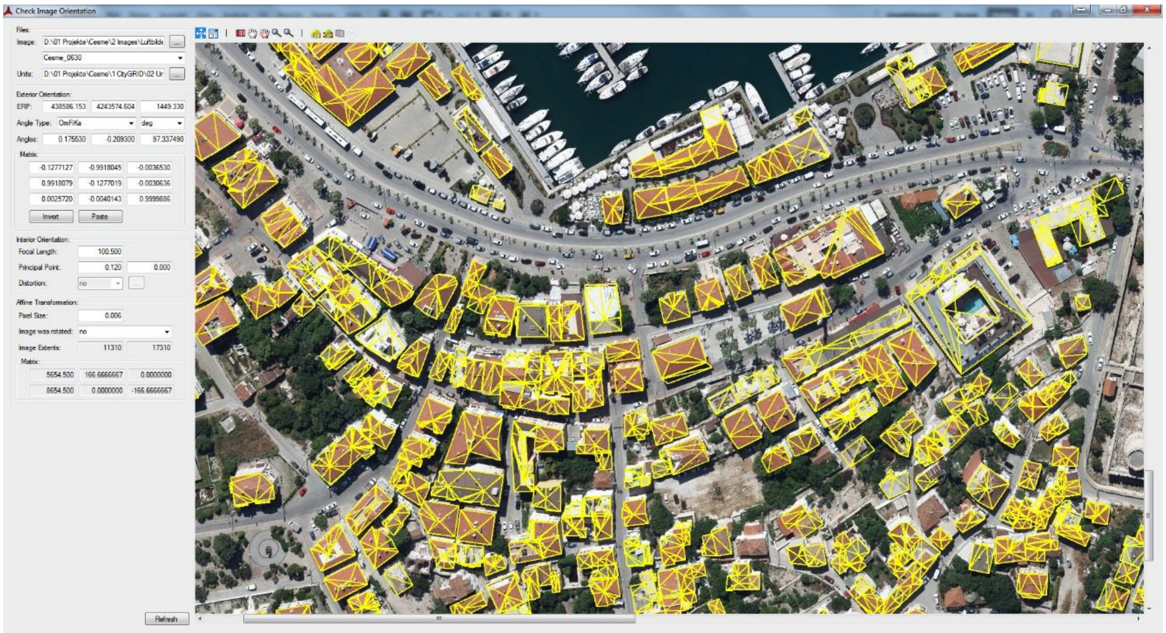


Figure 5.2 Raw aerial image (left) and pre-processed (color-enhanced) image (right) of Çeşme

## 5.3 Image Rectification

In order to accomplish the automatic texturing of vector models such as building representations in 3D, the models and the images should be in the same reference coordinate system. This can be done by transforming the coordinates of either the images or the building representations to the specified reference coordinate system. Usually, airborne imagery is acquired and processed before the generation of building models. Thus, the common practice is using the coordinate system of the acquired images as the base reference system and generation of building representations using the coordinate system of the images. This practice stems from the idea that the acquired images would be rectified right after the image acquisition process and final images would have a coordinate system where the building geometries can comply. In some exceptional situations, the building models are generated prior to the aerial image acquisition and this might result in two different representations of the building (such as vector or photogrammetric) in 2 different reference coordinate systems. In such cases, a coordinate transformation of building representations is required.

Following this stage, the characteristics of the camera system need to be input into the texturizer software i.e. CityGRID. For instance, if the images are captured by using (a large format) an UltraCam camera (such as the implementation in this project) the exterior orientation parameters would be acquired directly from the parameter files generated by the IMU (Inertial Measurement Unit) device. In some other medium-format cameras more manual effort for parameter input can be necessary to input the parameters in correct order. The acquired airborne images can appear as rotated as a result of an intermediary process and this also should be taken into account when inputting the parameters to the texturizer software. The texturizer software also checks the direction and path of the flight, along with if there exists any unplanned change of direction or path. Following this, checks are made to prevent matching errors between the building representations and the rectified images. If no errors are found the process carries on with database setup and data import to the database. An example to the image orientation checking with CityGRID software for automatic texturing is given in Figure 5.3.



Files:
Image: D:\01 Projekte\Cesme\2 Images\Luftbilde
Cesme_0630
Units: D:\01 Projekte\Cesme\1 CityGRID\02 Ur

Exterior Orientation:
ERP: 438586.153 4243574.604 1449.330
Angle Type: OmFiKa deg
Angles: 0.175530 -0.209300 97.337490
Matrix:
-0.1277127 -0.9918045 -0.0036530
0.9918079 -0.1277019 -0.0030636
0.0025720 -0.0040143 0.9999886

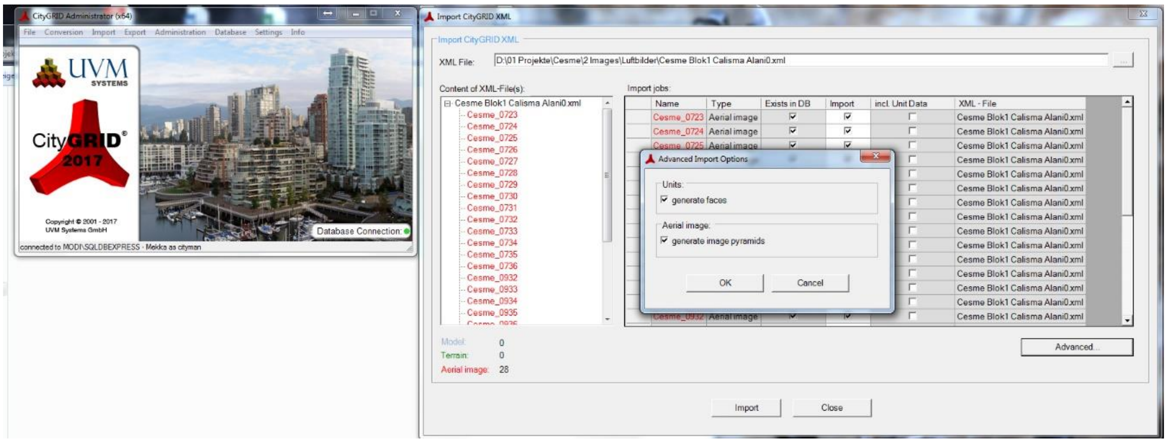
Interior Orientation:
Focal Length: 100.500
Principal Point: 0.120 0.000
Distortion: no

Affine Transformation:
Pixel Size: 0.006
Image was rotated: no
Image Extents: 11310 17310
Matrix:
5654.500 166.6666667 0.0000000
8654.500 0.0000000 -166.6666667

Figure 5.3 Image orientation checking with CityGRID software for automatic texturing

## 5.4 Database Creation and Data Import

The automatic-texturing software used in this study works on a database system. To run the automatic-texturing process, first aerial photographs, and their parameters together with buildings should be imported into a database such as Oracle or SQL. Generally, city models are generated based on maps, but the aerial image acquisition route is different from the maps. The flight is in the form of a column, so the project area needs to be transformed into a model space. A model space is an area that is covered by imported aerial photographs and this model has four main elements: orthophoto, DTM, building models and stereo photographs that correspond each model. The model space area is expanded with a buffer zone so the buildings near the outer border are included completely in the project area. A structure between software and the model space are generated as with this approach. Figure 5.4 shows the import GUI of CityGRID software.



connected to MODI\SQLDBEXPRESS - Mekka as cityman

Import CityGRID XML

XML File: D:\01 Projekte\Cesme\2 Images\Luftbilder\Cesme Blok1 Calisma Alani0.xml

Content of XML-File(s):

- Cesme Blok1 Calisma Alani0.xml
  - Cesme_0723
  - Cesme_0724
  - Cesme_0725
  - Cesme_0726
  - Cesme_0727
  - Cesme_0728
  - Cesme_0729
  - Cesme_0730
  - Cesme_0731
  - Cesme_0732
  - Cesme_0733
  - Cesme_0734
  - Cesme_0735
  - Cesme_0736
  - Cesme_0932
  - Cesme_0933
  - Cesme_0934
  - Cesme_0935

Import jobs:

| Name | Type | Exists in DB | Import | incl. Unit Data | XML - File |
|---|---|---|---|---|---|
| Cesme_0723 | Aerial image | ☑ | ☑ | ☐ | Cesme Blok1 Calisma Alani0.xml |
| Cesme_0724 | Aerial image | ☑ | ☑ | ☐ | Cesme Blok1 Calisma Alani0.xml |
| Cesme_0725 | Aerial image | ☑ | ☑ | ☐ | Cesme Blok1 Calisma Alani0.xml |
|  |  |  |  | ☐ | Cesme Blok1 Calisma Alani0.xml |
|  |  |  |  | ☐ | Cesme Blok1 Calisma Alani0.xml |
|  |  |  |  | ☐ | Cesme Blok1 Calisma Alani0.xml |
|  |  |  |  | ☐ | Cesme Blok1 Calisma Alani0.xml |
|  |  |  |  | ☐ | Cesme Blok1 Calisma Alani0.xml |
|  |  |  |  | ☐ | Cesme Blok1 Calisma Alani0.xml |
|  |  |  |  | ☐ | Cesme Blok1 Calisma Alani0.xml |
|  |  |  |  | ☐ | Cesme Blok1 Calisma Alani0.xml |
|  |  |  |  | ☐ | Cesme Blok1 Calisma Alani0.xml |
|  |  |  |  | ☐ | Cesme Blok1 Calisma Alani0.xml |
|  |  |  |  | ☐ | Cesme Blok1 Calisma Alani0.xml |
| Cesme_0932 | Aerial image | ☑ | ☑ | ☐ | Cesme Blok1 Calisma Alani0.xml |

Model: 0
Terrain: 0
Aerial image: 28

Advanced Import Options

Units:
☑ generate faces

Aerial image:
☑ generate image pyramids

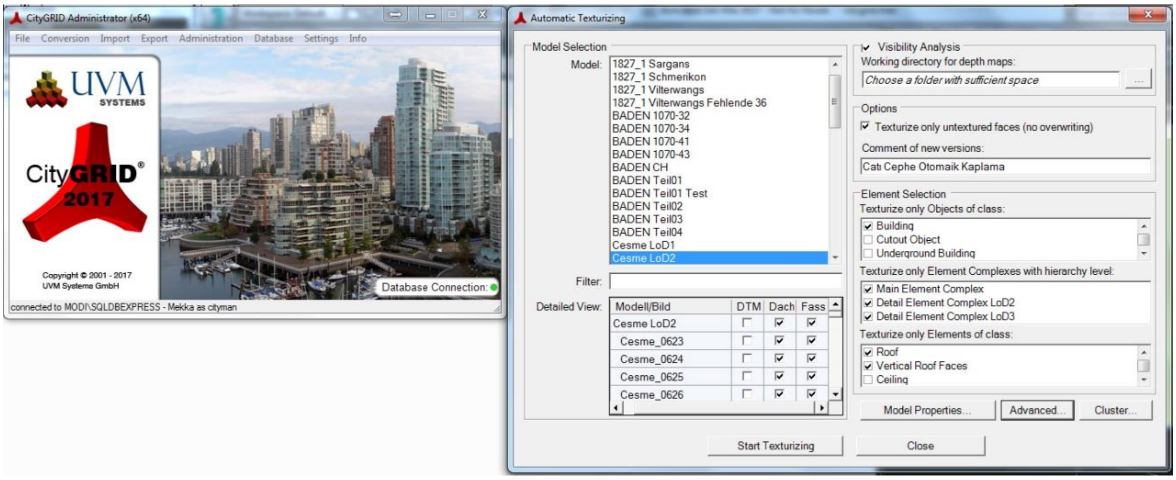
Figure 5.4 Importing aerial images and building models to database

This model space usually generated in rectangular-shape based or neighborhood-based in the layout. Neighborhood-based model space is rarely used as it can be very large. So, we put the city model on a diagram in the database and organize it on a rectangular-shape basis. After importing all data into a database, aerial photographs and adjusted exterior orientation parameters are matched with buildings, DTMs and orthophotos in the database. The relationship between the database and CityGRID Texturizer software is done with FME workbench. This relationship promotes the software that which aerial photographs, buildings, and DTMs are what the model space is made of. To summarize the model area, compatibility between aerial photographs, interior and exterior orientation parameters and buildings are checked and the process is started if there is no warning or errors.

## 5.5 Texture Mapping Process

In the CityGRID software, it is possible to choose the photo acquisition type such as aerial oblique imagery, aerial nadir imagery or mobile vehicle. All these parameters are calculated by the software and the algorithm determines the texturing style for the buildings. Once the automatic-texturing starts, hundreds of photos can be defined for a model area. But for a very large area like Çeşme, just using photos corresponding to the model area will always make software work faster. For example, if there are 40 photos corresponding to the model area and 100 photos are given as input, the software would try to find the right photos corresponding to that area and eliminate redundant photos. Therefore, using correct photos for the model area would avoid redundant work and help to reduce the processing time. To find correct photos corresponding to a building object, the algorithm splits the a building into objects (sub-parts) and assigns each of them to different classes. The buildings are usually split into 4 different objects such as roof, façade, base, and fringe.

This object structure is similar to the object structure in CityGML specification. The software can directly use CityGML object structure in the automatic-texturing process. The software GUI for this process is provided in Figure 5.5.



connected to MODI\SQLDBEXPRESS - Mekka as cityman

Model Selection

Model:
1827_1 Sargans
1827_1 Schmerikon
1827_1 Vilterwangs
1827_1 Vilterwangs Fehlende 36
BADEN 1070-32
BADEN 1070-34
BADEN 1070-41
BADEN 1070-43
BADEN CH
BADEN Teil01
BADEN Teil01 Test
BADEN Teil02
BADEN Teil03
BADEN Teil04
Cesme LoD1
Cesme LoD2

Filter:

Detailed View:

| Modell/Bild | DTM | Dach | Fass |
|---|---|---|---|
| Cesme LoD2 | ☐ | ☑ | ☑ |
| Cesme_0623 | ☐ | ☑ | ☑ |
| Cesme_0624 | ☐ | ☑ | ☑ |
| Cesme_0625 | ☐ | ☑ | ☑ |
| Cesme_0626 | ☐ | ☑ | ☑ |

☑ Visibility Analysis
Working directory for depth maps:
*Choose a folder with sufficient space*

Options
☑ Texturize only untextured faces (no overwriting)
Comment of new versions:
Catı Cephe Otomaik Kaplama

Element Selection
Texturize only Objects of class:
☑ Building
☐ Cutout Object
☐ Underground Building
Texturize only Element Complexes with hierarchy level:
☑ Main Element Complex
☑ Detail Element Complex LoD2
☑ Detail Element Complex LoD3
Texturize only Elements of class:
☑ Roof
☑ Vertical Roof Faces
☐ Ceiling

Figure 5.5 Automatic texturing properties

The building objects entering automatic-texturing process are the roof, the vertical surfaces of the roof and the façade. The fringes and the base part of the building are not used in the process. In the software, these objects are decomposed and can be textured according to the data type used in the project such as stereo photos, oblique photos or true-orthophotos. The parts of the buildings that will be used in the texturing process can be selected by the user. For example, only roofs can be textured in a project and other parts of the building can be ignored. It is also possible to texture building façades with oblique images, and texture roofs from orthophotos with the software. In oblique camera systems, both oblique and nadir photos are being acquired. The software can handle both datasets at the same time to texture different parts of the buildings. The nadir (vertical) stereo images in this study have been taken with 80% forward and 60% lateral overlap, which is sufficient for texturing both the roofs and the façades.

The most important factor in texturing roofs is the center point of the aerial photographs. The best textures for the roofs are found in the photo where the building is located close to the image center. In other words, the principal axis of the image should intersect the roof top perpendicularly if possible. For example in the application dataset, the same roof can be found in 9 or 10 different photos due to high overlap. The software finds the most suitable photograph for the texturing roof by comparing the center point coordinates of the roof and the aerial photographs. After the comparison, the photo with the closest center point to the roof center is used for texturing. For this reason, the software doesn’t require “True Orthophoto” for texturing roof objects.

Façade texturing is done with a different approach from the roof texturing. This approach is flexible and affected mainly by the camera system and city structure. Other parameters are whether the camera is large- or medium-format, how high the buildings in the city are, how the vegetation is covered, whether there is fringe in the buildings, and topographical characteristics. Image orientation angles along with center coordinates are of great importance for the façade texturing. The façades appear to be better for texturing in the regions closer to the outer border of the photograph. In terms of viewing angle, the building has clearer outward appearance, but the camera distortions should also be taken into account in this case.

Even though some companies produce quasi distortion-free cameras, the photos are partially visually distorted.  The photos to be used for the façade texturing are usually analyzed visually to find out the best image zone between the photo center and the outer border, although this is also subject to change due to parameters mentioned previously. After the decision of the best zone, existing photographs of the building are detected automatically by the software based on a coordinate comparison. Finally the software determine the ideal pictures for texturing the façades calculate the distance between the photo center and the façade. The photos, in which the building is located close to the outer border, are eliminated and the remaining photos are analyzed to find the most appropriate photograph with a suitable angle. The angle between the surface normal of the façade and the principal axis of the photograph is calculated and the image that has the smallest angle with the façade normal is used for texturing.

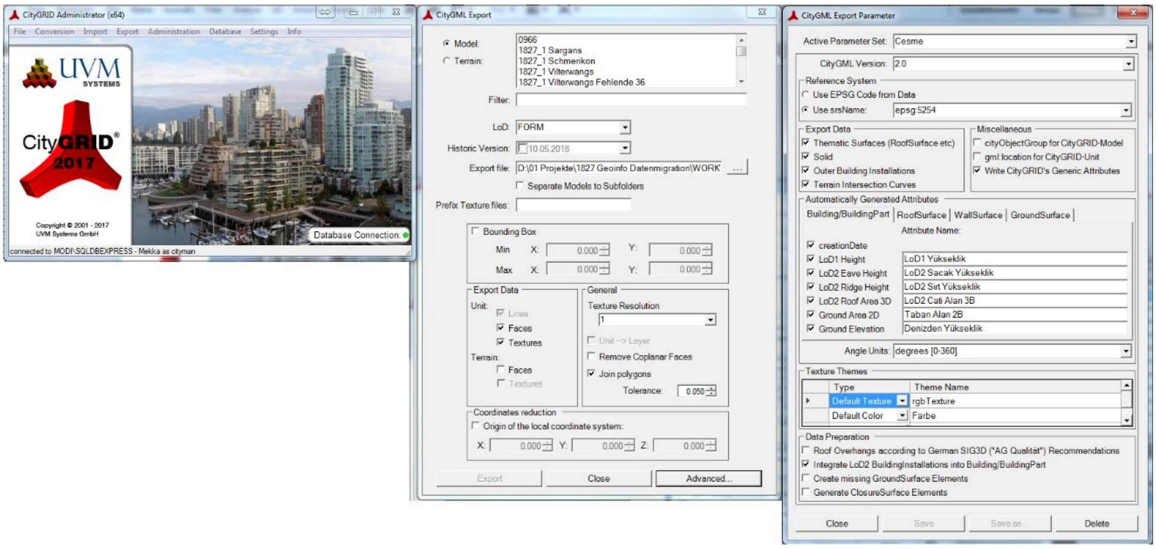
Another criterion for selection of the best photograph for automatic façade texturing is the number of pixels that the building has in an image. On the other hand, if “Visibility Analysis” feature is activated, a visibility check is performed to find out obstacles in the scene, e.g. other buildings in that angle of vision. In other words, the façade texture may contain pixels from other buildings. Therefore, even though an image has the most appropriate geometry for a particular building model, it may fail in the visibility analysis. Although the image geometry is worse, the photo that has the best view of the façade is selected to protect the realism. A façade can be seen in photographs taken from different flight strips and different directions. Due to the temporal difference in image acquisition, there might be variations in the direction of the sun or the shadow of the clouds between the images. In such cases, two different façades facing the same direction are textured from the same photo to preserve homogeneity. Thus, the appearance of absurd textures on two different façades facing the same direction is prohibited.

The software cuts out the chosen photos for all façades and saves them in separate image files. While cutting out the aerial photos, a predefined buffer zone is kept at the peripherals. Thus, if any editing is needed to be done in the building geometry, gaps in the building envelope can be prevented. Each texture is then used in that part of the building according to the CityGML data structure. The extracted image is not directly textured to the façade, but it is related one-to-one based on file names. In other words, the texture parts cut out from each aerial photograph is unique and can only be used for one façade belonging to one building. Every texture image is used only for one building or façade.

In this way, a building is textured with dozens of photographs taken from different aerial photographs. Thus, the number of textures in city models can reach hundreds of thousands depending on the size of the city model. Each new element created in the database has a unique ID number. Working in database prevents users to manually assign a unique ID to all the texture files each time, which is done automatically by the database management system.

Automatic texturing supports parallel processing, multiple threads can be started at the same time. Depending on the size of the buildings and the area, texturing of an average building takes between 5 to 10 seconds. If “Visibility Analysis” feature is activated, this process takes up to 3 times longer than the standard texturing. The software import photos into RAM (random access memory), texture the buildings, and then delete the photo from RAM so that the memory is released. The larger the photo size, the longer it takes to import and export the photo to the RAM. This process is faster in photos with smaller size, but this time the number of photos will increase, so the time usually remains the same. Multi-object texturing (i.e. “Texture Atlassing” feature of the software) is not used for the automatic texturing, since each object has a different structure and must be handled separately. Editing a single photo to fit the structure of that object will be more efficient than editing all the different structures of the entire building together. “Texture Atlassing” is done as a last step for improving the presentation. This operation can be done on building or area basis.

Figure 5.6 shows the CityGRID GUI for exporting the textured models into CityGML format. A part of the textured city model area is presented in Figure 5.7.



connected to MODI\SQLDBEXPRESS - Mekka as cityman

CityGML Export:

● Model:
○ Terrain:

0966
1827_1 Sargans
1827_1 Schmerikon
1827_1 Vilterwangs
1827_1 Vilterwangs Fehlende 36

Filter:
LoD: FORM
Historic Version: ☐ 10.05.2018
Export file: D:\01 Projekte\1827 Geoinfo Datenmigration\WORK
☐ Separate Models to Subfolders
Prefix Texture files:

☐ Bounding Box
Min X: 0.000 Y: 0.000
Max X: 0.000 Y: 0.000

Export Data
Unit:
☑ Lines
☑ Faces
☑ Textures
Terrain:
☐ Faces
☐ Textures

General
Texture Resolution: 1
☐ Unit -> Layer
☐ Remove Coplanar Faces
☑ Join polygons
Tolerance: 0.050

Coordinates reduction
☐ Origin of the local coordinate system:
X: 0.000 Y: 0.000 Z: 0.000

CityGML Export Parameter:

Active Parameter Set: Cesme
CityGML Version: 2.0

Reference System
○ Use EPSG Code from Data
● Use srsName: epsg:5254

Export Data
☑ Thematic Surfaces (RoofSurface etc)
☑ Solid
☑ Outer Building Installations
☑ Terrain Intersection Curves

Miscellaneous
☐ cityObjectGroup for CityGRID-Model
☐ gml:location for CityGRID-Unit
☑ Write CityGRID's Generic Attributes

Automatically Generated Attributes
Building/BuildingPart | RoofSurface | WallSurface | GroundSurface

| | Attribute Name: |
|---|---|
| ☑ creationDate | |
| ☑ LoD1 Height | LoD1 Yükseklik |
| ☑ LoD2 Eave Height | LoD2 Sacak Yükseklik |
| ☑ LoD2 Ridge Height | LoD2 Sirt Yükseklik |
| ☑ LoD2 Roof Area 3D | LoD2 Cati Alan 3B |
| ☑ Ground Area 2D | Taban Alan 2B |
| ☑ Ground Elevation | Denizden Yükseklik |

Angle Units: degrees [0-360]

Texture Themes

| Type | Theme Name |
|---|---|
| Default Texture | rgbTexture |
| Default Color | Farbe |

Data Preparation
☐ Roof Overhangs according to German SIG3D ("AG Qualität") Recommendations
☑ Integrate LoD2 BuildingInstallations into Building/BuildingPart
☐ Create missing GroundSurface Elements
☐ Generate ClosureSurface Elements

Figure 5.6 Textured CityGML export properties

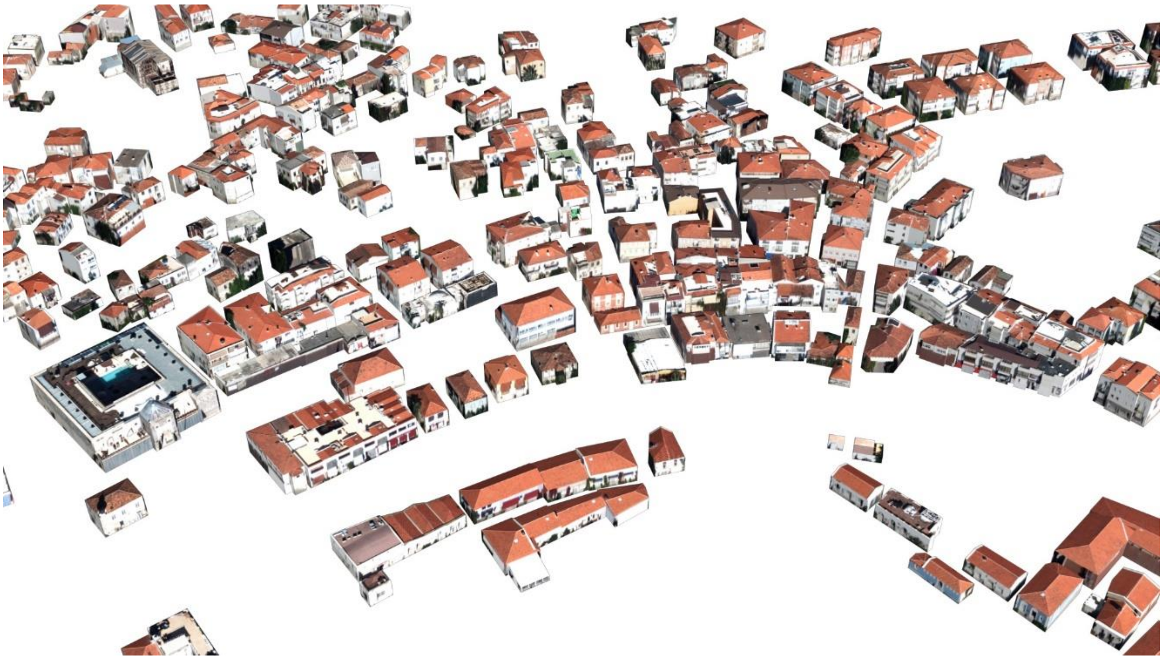
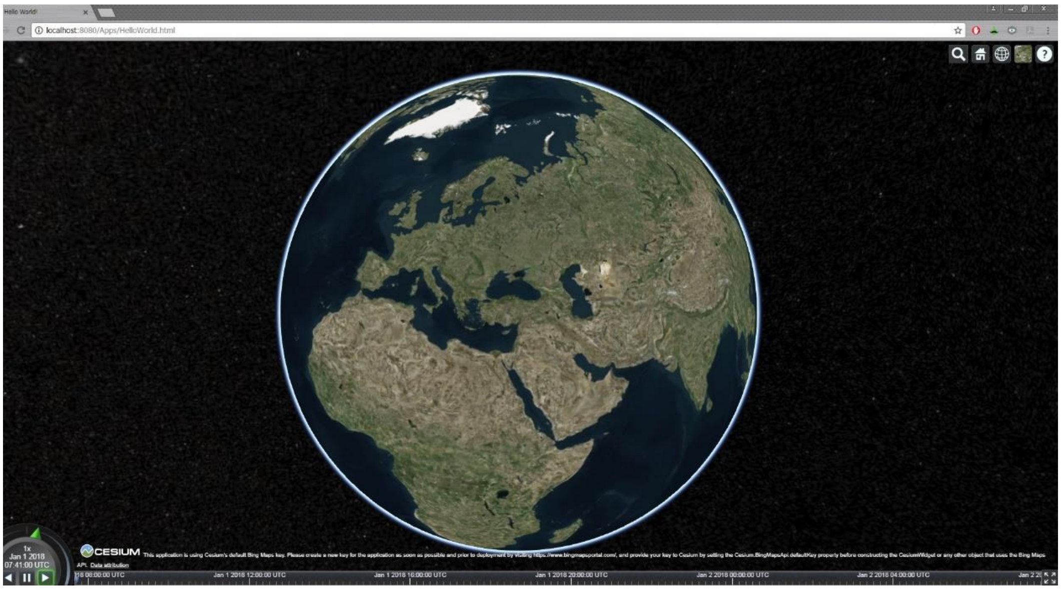


Figure 5.7 Exported CityGML buildings with textures

# 6. WEB-BASED VISUALIZATION

## 6.1 CesiumJS Virtual Globe

CesiumJS [90] is an open-source JavaScript library for visualizing terrain and geospatial 2D and 3D data on a web interface (Figure 6.1). It works on WebGL using computer's GPU that provides the high FPS (Frame per Second) and performance on large datasets. WebGL also provides optimum usage of computer hardware without installing any plugin on any operating system or web browser. The library can be modified and new features can be added depending on user's requirements. Cesium only supports visualizing static data, loading portions of large 3D city models dynamically from a database is not supported yet.



This application is using Cesium's default Bing Maps key. Please create a new key for the application as soon as possible and prior to deployment by visiting https://www.bingmapsportal.com/, and provide your key to Cesium by setting the Cesium.BingMapsApi.defaultKey property before constructing the CesiumWidget or any other object that uses the Bing Maps

Figure 6.1 CesiumJS user interface

WebGL (Web Graphics Library) is an API for visualizing data on web interface without installing any plug-ins. A WebGL context is created by requesting a WebGL context from an HTML canvas element. WebGL supports visualizing 3D objects on directly in all major web browsers running on all major operating systems without needing additional plug-ins or extensions. WebGL changed web-based map approach from 2D maps to impressive 3D maps [102]. It is designed as a rendering context for the HTML Canvas element. WebGL brings OpenGL to JavaScript, and since JavaScript is mainly a web-based programming language, it can be said that WebGL brings OpenGL to the web. JavaScript is a high-level, dynamic and untyped interpreted programming language. So the only requirement to run JavaScript code on a web interface is to add the tag of JavaScript file into HTML code for each file [103].

Chaturvedi et al. [104] indicates the main features of Cesium as:

- It supports visualizing dynamic data on web-interface. Cesium Language (CZML) is a JSON based schema for visualizing geospatial data and their attributes that continuously changing or moving over time.
- It supports different Web Map Services (WMS) such as Open Street Map, Bingmaps, Stamen, Mapbox or any other custom WMS. Every service is displayed as background map with different image radiometric values.
- It allows users to visualize 2D objects along with the 3D objects. The users can show vector data as points, polyline, etc.
- Different data formats such as KML or JSON can be visualized directly on Cesium.
- Exploring a virtual globe with different camera angles and zoom levels with using keyboard or mouse is possible.
- Cesium allows users to change object façade appearances along with the custom material support.
- It supports popular coordinate systems such as WGS84 and ICRF (International Celestial Referance Frame), and allows users to reproject coordinate system with the Cesium math libraries.

In addition to the features mentioned above, some other features can be listed as follows:

- User interface and features can be customized for the purpose of the project.
- It uses Cesium 3D Tiles format for streaming, styling and interacting 3D datasets. Cesium 3D tiles supports building models, other 3D objects and dense point clouds.
- It supports visualization of high-resolution global terrain datasets.
- Selection and infobox widgets for highlighting objects and displaying information.
- It supports creating visual effects such as atmospheric changes, shadows, fog, sun light or water.
- Hardware optimization for reducing GPU, CPU and power usage while visualizing the data.

## 6.2 KML/COLLADA/glTF visualization using 3DCityDB

KML (Keyhole Markup Language) is an XML based file format that allows users to visualize geographic data on CesiumJS or other software. COLLADA (Collaborative Design Activity) is also an XML based file format that represents 3D geometries. After integration of COLLADA, KML can be used to visualize 3D models on the virtual globes. Although conversion the CityGML data to COLLADA format makes it thematically designable in an advanced 3D software environment, the main disadvantage of conversion into this format is that all semantic data that are stored in CityGML data will no longer available after the conversion. In other words, all possible database queries and aggregations regarding CityGML semantics and also graphical data structure, will no longer available in COLLADA format environment. 3DCityDB software allows users to visualize CityGML data with the help of KML/COLLADA integration.

3DCityDB software [84] is a software package for visualizing and managing CityGML based 3D city models. The software allows users to export CityGML data in KML/COLLADA/glTF formats for visualization. These data can be visualized in various applications such as CesiumJS and Google Earth. Setting up a 3DCityDB environment requires a running installation of several software tools which are freely available. The workflow for visualizing city models with 3DcityDB software is provided in Figure 6.2.

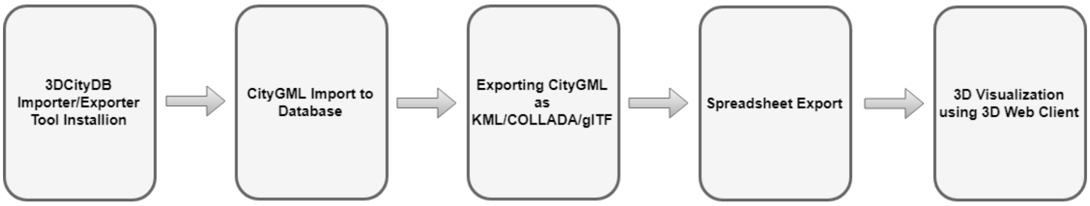
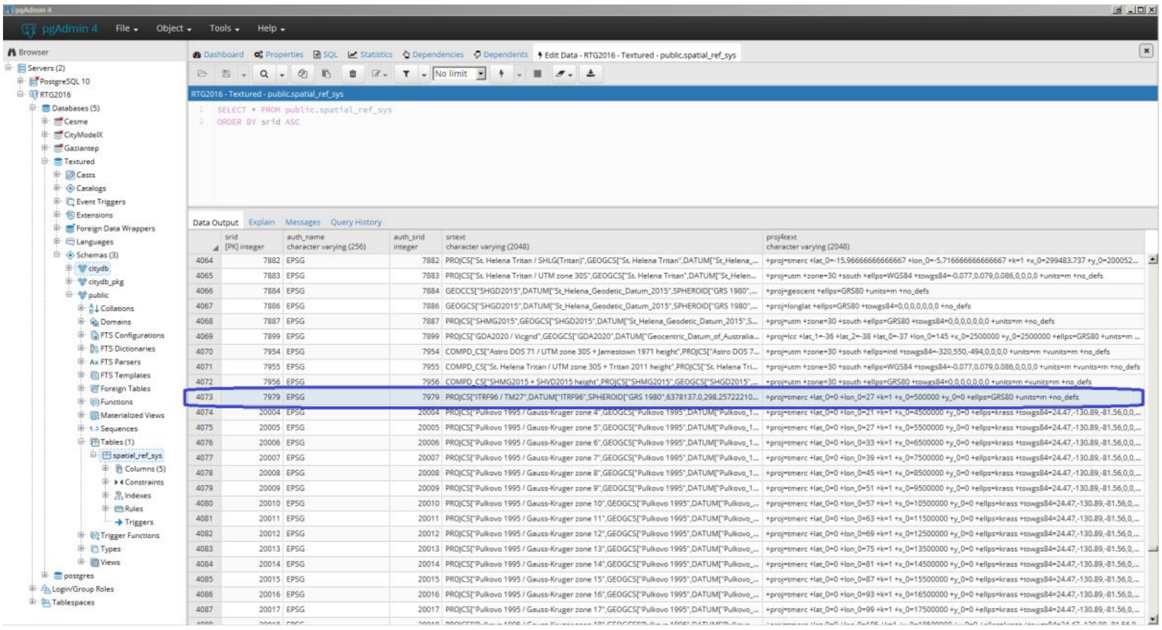

3DCityDB Importer/Exporter Tool Installion
CityGML Import to Database
Exporting CityGML as KML/COLLADA/glTF
Spreadsheet Export
3D Visualization using 3D Web Client

Figure 6.2 Workflow for visualizing with 3DCityDB software

Additional software which is specifically required in the environment are listed as below:

- Java 8 Runtime Environment
- PostgreSQL and PostGIS
- 3DCityDB Importer/Exporter Tool, Scripts, and Web map Client
- 3DCityDB Spreadsheet Generator Plugin
- Google Earth Pro
- Node.js

After installing all the necessary software mentioned above, the settings for PostgreSQL and PostGIS have to be made. As the first step, we create a new database as a superuser in PostgreSQL. ITRF96 reference coordinate system that is being used in Turkey doesn't exist in the coordinate system table of the PostGIS software. This coordinate system is required to georeference the data correctly and to generate tables in the database. The ITRF96 coordinate system is added to the "spatial_ref_sys" table as shown in Figure 6.3.



```sql
SELECT * FROM public.spatial_ref_sys
ORDER BY srid ASC
```

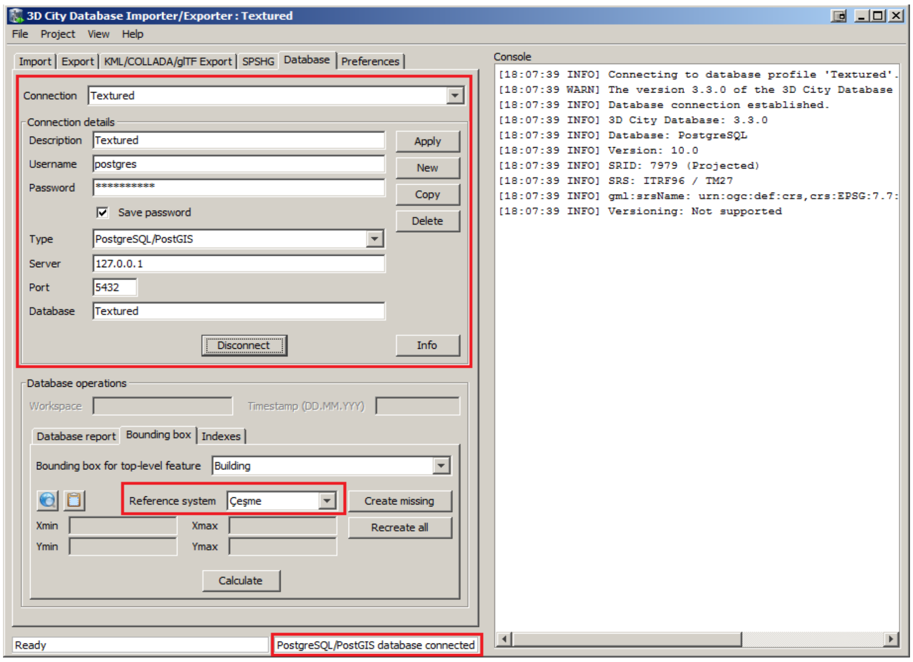
| | srid [PK] integer | auth_name character varying (256) | auth_srid integer | srtext character varying (2048) | proj4text character varying (2048) |
|---|---|---|---|---|---|
| 4064 | 7882 | EPSG | 7882 | PROJCS["St. Helena Tritan / SHLG(Tritan)",GEOGCS["St. Helena Tritan",DATUM["St_Helena_... | +proj=tmerc +lat_0=-15.96666666666667 +lon_0=-5.716666666666667 +k=1 +x_0=299483.737 +y_0=200052... |
| 4065 | 7883 | EPSG | 7883 | PROJCS["St. Helena Tritan / UTM zone 30S",GEOGCS["St. Helena Tritan",DATUM["St_Helen... | +proj=utm +zone=30 +south +ellps=WGS84 +towgs84=-0.077,0.079,0.086,0,0,0,0 +units=m +no_defs |
| 4066 | 7884 | EPSG | 7884 | GEOCCS["SHGD2015",DATUM["St_Helena_Geodetic_Datum_2015",SPHEROID["GRS 1980",... | +proj=geocent +ellps=GRS80 +units=m +no_defs |
| 4067 | 7886 | EPSG | 7886 | GEOGCS["SHGD2015",DATUM["St_Helena_Geodetic_Datum_2015",SPHEROID["GRS 1980",... | +proj=longlat +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +no_defs |
| 4068 | 7887 | EPSG | 7887 | PROJCS["SHMG2015",GEOGCS["SHGD2015",DATUM["St_Helena_Geodetic_Datum_2015",S... | +proj=utm +zone=30 +south +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs |
| 4069 | 7899 | EPSG | 7899 | PROJCS["GDA2020 / Vicgrid",GEOGCS["GDA2020",DATUM["Geocentric_Datum_of_Australia... | +proj=lcc +lat_1=-36 +lat_2=-38 +lat_0=-37 +lon_0=145 +x_0=2500000 +y_0=2500000 +ellps=GRS80 +units=m ... |
| 4070 | 7954 | EPSG | 7954 | COMPD_CS["Astro DOS 71 / UTM zone 30S + Jamestown 1971 height",PROJCS["Astro DOS 7... | +proj=utm +zone=30 +south +ellps=intl +towgs84=-320,550,-494,0,0,0,0 +units=m +vunits=m +no_defs |
| 4071 | 7955 | EPSG | 7955 | COMPD_CS["St. Helena Tritan / UTM zone 30S + Tritan 2011 height",PROJCS["St. Helena Tri... | +proj=utm +zone=30 +south +ellps=WGS84 +towgs84=-0.077,0.079,0.086,0,0,0,0 +units=m +vunits=m +no_defs |
| 4072 | 7956 | EPSG | 7956 | COMPD_CS["SHMG2015 + SHVD2015 height",PROJCS["SHMG2015",GEOGCS["SHGD2015",... | +proj=utm +zone=30 +south +ellps=GRS80 +towgs84=0,0,0,0,0,0,0 +units=m +vunits=m +no_defs |
| 4073 | 7979 | EPSG | 7979 | PROJCS["ITRF96 / TM27",DATUM["ITRF96",SPHEROID["GRS 1980",6378137.0,298.25722210... | +proj=tmerc +lat_0=0 +lon_0=27 +k=1 +x_0=500000 +y_0=0 +ellps=GRS80 +units=m +no_defs |
| 4074 | 20004 | EPSG | 20004 | PROJCS["Pulkovo 1995 / Gauss-Kruger zone 4",GEOGCS["Pulkovo 1995",DATUM["Pulkovo_1... | +proj=tmerc +lat_0=0 +lon_0=21 +k=1 +x_0=4500000 +y_0=0 +ellps=krass +towgs84=24.47,-130.89,-81.56,0,0,... |
| 4075 | 20005 | EPSG | 20005 | PROJCS["Pulkovo 1995 / Gauss-Kruger zone 5",GEOGCS["Pulkovo 1995",DATUM["Pulkovo_1... | +proj=tmerc +lat_0=0 +lon_0=27 +k=1 +x_0=5500000 +y_0=0 +ellps=krass +towgs84=24.47,-130.89,-81.56,0,0,... |
| 4076 | 20006 | EPSG | 20006 | PROJCS["Pulkovo 1995 / Gauss-Kruger zone 6",GEOGCS["Pulkovo 1995",DATUM["Pulkovo_1... | +proj=tmerc +lat_0=0 +lon_0=33 +k=1 +x_0=6500000 +y_0=0 +ellps=krass +towgs84=24.47,-130.89,-81.56,0,0,... |
| 4077 | 20007 | EPSG | 20007 | PROJCS["Pulkovo 1995 / Gauss-Kruger zone 7",GEOGCS["Pulkovo 1995",DATUM["Pulkovo_1... | +proj=tmerc +lat_0=0 +lon_0=39 +k=1 +x_0=7500000 +y_0=0 +ellps=krass +towgs84=24.47,-130.89,-81.56,0,0,... |
| 4078 | 20008 | EPSG | 20008 | PROJCS["Pulkovo 1995 / Gauss-Kruger zone 8",GEOGCS["Pulkovo 1995",DATUM["Pulkovo_1... | +proj=tmerc +lat_0=0 +lon_0=45 +k=1 +x_0=8500000 +y_0=0 +ellps=krass +towgs84=24.47,-130.89,-81.56,0,0,... |
| 4079 | 20009 | EPSG | 20009 | PROJCS["Pulkovo 1995 / Gauss-Kruger zone 9",GEOGCS["Pulkovo 1995",DATUM["Pulkovo_1... | +proj=tmerc +lat_0=0 +lon_0=51 +k=1 +x_0=9500000 +y_0=0 +ellps=krass +towgs84=24.47,-130.89,-81.56,0,0,... |
| 4080 | 20010 | EPSG | 20010 | PROJCS["Pulkovo 1995 / Gauss-Kruger zone 10",GEOGCS["Pulkovo 1995",DATUM["Pulkovo_... | +proj=tmerc +lat_0=0 +lon_0=57 +k=1 +x_0=10500000 +y_0=0 +ellps=krass +towgs84=24.47,-130.89,-81.56,0,... |
| 4081 | 20011 | EPSG | 20011 | PROJCS["Pulkovo 1995 / Gauss-Kruger zone 11",GEOGCS["Pulkovo 1995",DATUM["Pulkovo_... | +proj=tmerc +lat_0=0 +lon_0=63 +k=1 +x_0=11500000 +y_0=0 +ellps=krass +towgs84=24.47,-130.89,-81.56,0,... |
| 4082 | 20012 | EPSG | 20012 | PROJCS["Pulkovo 1995 / Gauss-Kruger zone 12",GEOGCS["Pulkovo 1995",DATUM["Pulkovo_... | +proj=tmerc +lat_0=0 +lon_0=69 +k=1 +x_0=12500000 +y_0=0 +ellps=krass +towgs84=24.47,-130.89,-81.56,0,... |
| 4083 | 20013 | EPSG | 20013 | PROJCS["Pulkovo 1995 / Gauss-Kruger zone 13",GEOGCS["Pulkovo 1995",DATUM["Pulkovo_... | +proj=tmerc +lat_0=0 +lon_0=75 +k=1 +x_0=13500000 +y_0=0 +ellps=krass +towgs84=24.47,-130.89,-81.56,0,... |
| 4084 | 20014 | EPSG | 20014 | PROJCS["Pulkovo 1995 / Gauss-Kruger zone 14",GEOGCS["Pulkovo 1995",DATUM["Pulkovo_... | +proj=tmerc +lat_0=0 +lon_0=81 +k=1 +x_0=14500000 +y_0=0 +ellps=krass +towgs84=24.47,-130.89,-81.56,0,... |
| 4085 | 20015 | EPSG | 20015 | PROJCS["Pulkovo 1995 / Gauss-Kruger zone 15",GEOGCS["Pulkovo 1995",DATUM["Pulkovo_... | +proj=tmerc +lat_0=0 +lon_0=87 +k=1 +x_0=15500000 +y_0=0 +ellps=krass +towgs84=24.47,-130.89,-81.56,0,... |
| 4086 | 20016 | EPSG | 20016 | PROJCS["Pulkovo 1995 / Gauss-Kruger zone 16",GEOGCS["Pulkovo 1995",DATUM["Pulkovo_... | +proj=tmerc +lat_0=0 +lon_0=93 +k=1 +x_0=16500000 +y_0=0 +ellps=krass +towgs84=24.47,-130.89,-81.56,0,... |
| 4087 | 20017 | EPSG | 20017 | PROJCS["Pulkovo 1995 / Gauss-Kruger zone 17",GEOGCS["Pulkovo 1995",DATUM["Pulkovo_... | +proj=tmerc +lat_0=0 +lon_0=99 +k=1 +x_0=17500000 +y_0=0 +ellps=krass +towgs84=24.47,-130.89,-81.56,0,... |

Figure 6.3 Adding ITRF96 coordinate system to PostGIS

In order to enable all spatial functions and data types, the PostGIS extension needs to be added to the new database and this can only be done by the superuser. The extension is added with the following command "CREATE EXTENSION postgis;". A new database is created with all required extensions to store and manage spatial features at this step. In the next steps, CityGML data schema is loaded to the created database instance.

The 3DCityDB software package comes with a package of SQL (Structured Query Language) scripts to create the required schemas on the spatial database system (PostgreSQL/PostGIS) and with a group of scripts to manage the 3D city model stored in the database. While creating schemas, the chosen reference system is applied to the spatial columns which is ITRF96 for this project. The 3DCityDB schema will be created and the database is ready for importing/exporting CityGML data.

3DCityDB Importer/Exporter offers both a graphical user interface (GUI) and a command line interface (CLI). The graphical user interface is organized into four main components as a menu bar, operations window, console window and a status bar. The database tab of the operations windows allows the user to manage and establish database connections and execute database operations. In order to connect to an instance of the 3D City Database, valid connection parameters must be provided on the database tab. If the required parameters have been correctly entered, the information of the connected 3DCityDB instance will be printed in the console window on the right side of the Importer/Exporter user interface, as can be seen in Figure 6.4.



Connection: Textured

Connection details

Description: Textured
Username: postgres
Password: **********
☑ Save password
Type: PostgreSQL/PostGIS
Server: 127.0.0.1
Port: 5432
Database: Textured

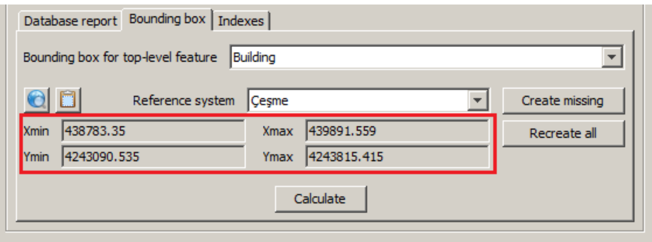
Bounding box for top-level feature: Building
Reference system: Çeşme

Ready
PostgreSQL/PostGIS database connected

```
[18:07:39 INFO] Connecting to database profile 'Textured'.
[18:07:39 WARN] The version 3.3.0 of the 3D City Database
[18:07:39 INFO] Database connection established.
[18:07:39 INFO] 3D City Database: 3.3.0
[18:07:39 INFO] Database: PostgreSQL
[18:07:39 INFO] Version: 10.0
[18:07:39 INFO] SRID: 7979 (Projected)
[18:07:39 INFO] SRS: ITRF96 / TM27
[18:07:39 INFO] gml:srsName: urn:ogc:def:crs,crs:EPSG:7.7:
[18:07:39 INFO] Versioning: Not supported
```

Figure 6.4. Database connection with 3DCityDB

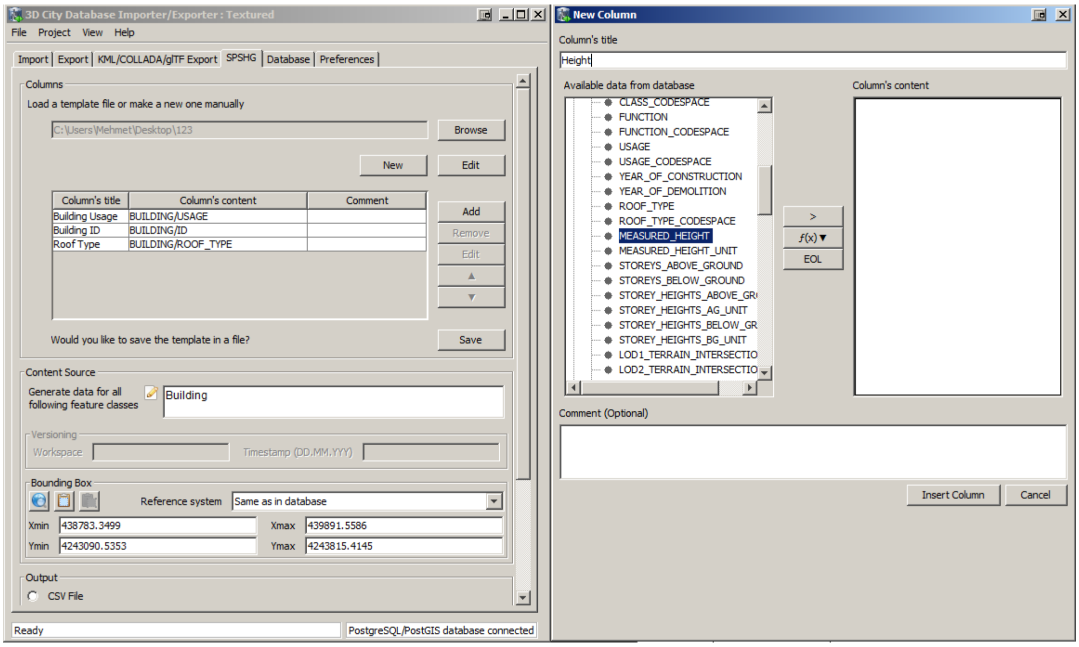
Prior to the data import, the CityGML files should be validated against the CityGML XML schema. It is strongly recommended that only CityGML files which have successfully passed the validation process are imported into the database. Otherwise, errors in the data may lead to unexpected behavior. After the validation process is successfully completed, the CityGML file can be imported into the database. The spatial extent of the imported CityGML objects can also be easily determined by using the Importer/Exporter Tool. The coordinate values of the lower left (Xmin, Ymin) and upper right (Xmax, Ymax) corner of the calculated bounding box are printed in the coordinate value fields of the dialog given in Figure 6.5.



Bounding box for top-level feature: Building
Reference system: Çeşme
Xmin: 438783.35
Xmax: 439891.559
Ymin: 4243090.535
Ymax: 4243815.415

Figure 6.5 Bounding box of the imported CityGML object

Spreadsheet Generator tool (Figure 6.6) lets users to export data from a 3DCityDB instance into a Comma-separated values (CSV) or Microsoft Excel (XLSX) file. This spreadsheet file may be imported to either a spreadsheet application like Microsoft Excel or to a web-based online spreadsheet service like Google Fusion Table. In addition, online spreadsheets offer features like sharing and collaborating in real-time with other users and can be easily accessed via Internet.



Load a template file or make a new one manually

C:\Users\Mehmet\Desktop\123

| Column's title | Column's content | Comment |
|---|---|---|
| Building Usage | BUILDING/USAGE | |
| Building ID | BUILDING/ID | |
| Roof Type | BUILDING/ROOF_TYPE | |

Would you like to save the template in a file?

Generate data for all following feature classes: Building

Reference system: Same as in database

Xmin 438783.3499
Xmax 439891.5586
Ymin 4243090.5353
Ymax 4243815.4145

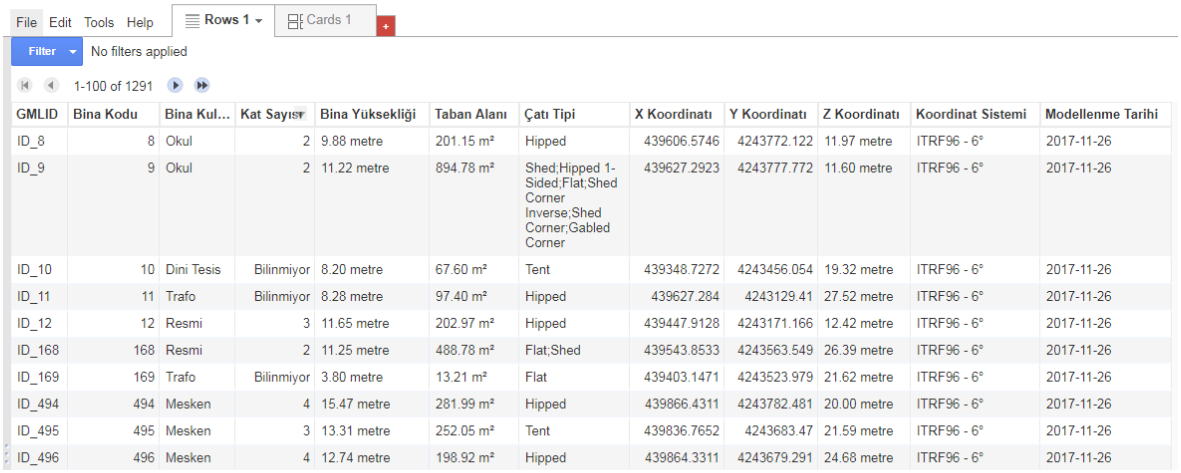
○ CSV File

Ready
PostgreSQL/PostGIS database connected

Column's title: Height

Figure 6.6. 3DCityDB Spreadsheet Generator

A bounding box needs to be specified to make sure that the concerned CityGML features lie within a given geographic area. Then the selected content information of all CityGML features stored in the database are exported as Microsoft Excel file and uploaded to Google Cloud Service (Figure 6.7).



| GMLID | Bina Kodu | Bina Kul... | Kat Sayısı | Bina Yüksekliği | Taban Alanı | Çatı Tipi | X Koordinatı | Y Koordinatı | Z Koordinatı | Koordinat Sistemi | Modellenme Tarihi |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ID_8 | 8 | Okul | 2 | 9.88 metre | 201.15 m² | Hipped | 439606.5746 | 4243772.122 | 11.97 metre | ITRF96 - 6° | 2017-11-26 |
| ID_9 | 9 | Okul | 2 | 11.22 metre | 894.78 m² | Shed;Hipped 1-Sided;Flat;Shed Corner Inverse;Shed Corner;Gabled Corner | 439627.2923 | 4243777.772 | 11.60 metre | ITRF96 - 6° | 2017-11-26 |
| ID_10 | 10 | Dini Tesis | Bilinmiyor | 8.20 metre | 67.60 m² | Tent | 439348.7272 | 4243456.054 | 19.32 metre | ITRF96 - 6° | 2017-11-26 |
| ID_11 | 11 | Trafo | Bilinmiyor | 8.28 metre | 97.40 m² | Hipped | 439627.284 | 4243129.41 | 27.52 metre | ITRF96 - 6° | 2017-11-26 |
| ID_12 | 12 | Resmi | 3 | 11.65 metre | 202.97 m² | Hipped | 439447.9128 | 4243171.166 | 12.42 metre | ITRF96 - 6° | 2017-11-26 |
| ID_168 | 168 | Resmi | 2 | 11.25 metre | 488.78 m² | Flat;Shed | 439543.8533 | 4243563.549 | 26.39 metre | ITRF96 - 6° | 2017-11-26 |
| ID_169 | 169 | Trafo | Bilinmiyor | 3.80 metre | 13.21 m² | Flat | 439403.1471 | 4243523.979 | 21.62 metre | ITRF96 - 6° | 2017-11-26 |
| ID_494 | 494 | Mesken | 4 | 15.47 metre | 281.99 m² | Hipped | 439866.4311 | 4243782.481 | 20.00 metre | ITRF96 - 6° | 2017-11-26 |
| ID_495 | 495 | Mesken | 3 | 13.31 metre | 252.05 m² | Tent | 439836.7652 | 4243683.47 | 21.59 metre | ITRF96 - 6° | 2017-11-26 |
| ID_496 | 496 | Mesken | 4 | 12.74 metre | 198.92 m² | Hipped | 439864.3311 | 4243679.291 | 24.68 metre | ITRF96 - 6° | 2017-11-26 |

Figure 6.7. Attribute table of the selected buildings in the 3DCityDB

The spatial data stored in the 3DCityDB can be exported in KML, COLLADA, and glTF formats for presentation, viewing, and visual inspection in a different application areas such as CesiumJS Virtual Globe. After configuring some settings for export from "Preferences" tab, 4 different types of buildings are exported as shown in Figure 6.8.

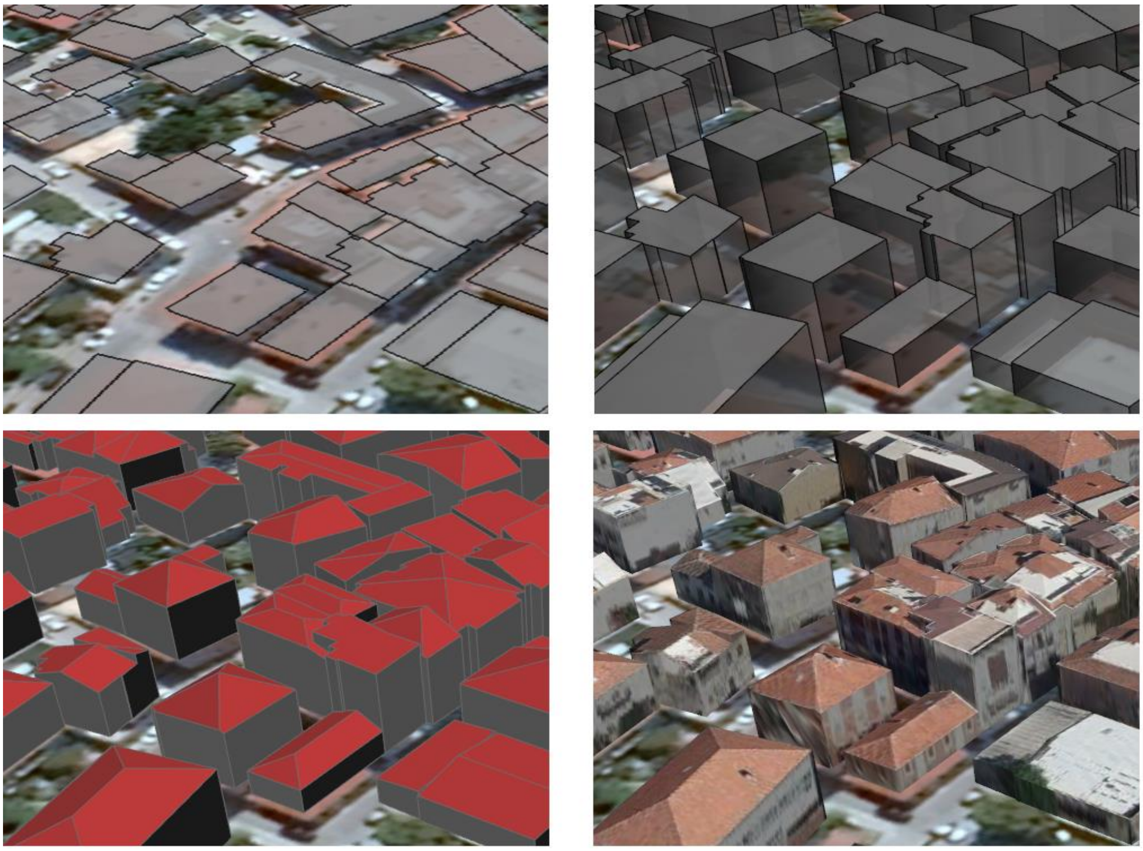
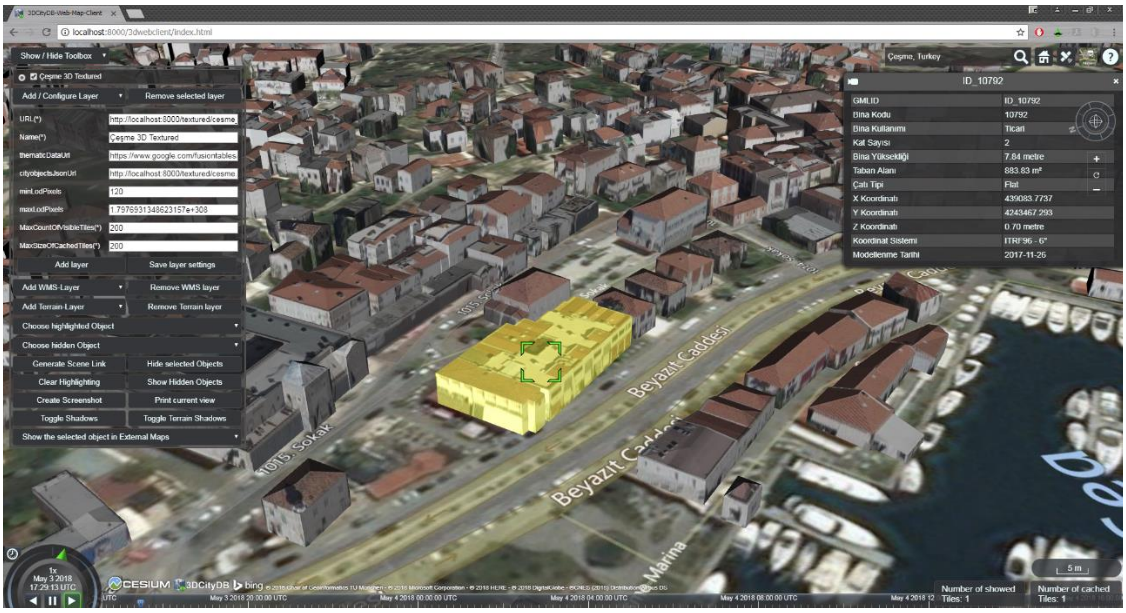


Figure 6.8. Different visualization forms of models from LoD0 to Textured LoD2 on Google Earth

3DCityDB software package includes a Cesium-based web interface tool named 3DCityDB Web-map-client for data visualization. This tool helps users to explore and visualize 3D City models with semantic data. Various changes and modifications have been made to the Cesium interface and source code to make it easier for users to visualize and explore 3D city models.

The 3DCityDB web client (Figure 6.9) is a static web application written in HTML and JavaScript and can, therefore, be easily deployed by uploading its files to a simple web server. The client comes with a lightweight JavaScript-based HTTP server that is mainly meant to test the functionality of the 3D web client on local machines. For running this web server, Node.js is required to be installed on the computer. After that, simply running command "node server.js" in the client folder will make the client available via the selected localhost port. After that last step, the client is ready for visualizing exported data on the web interface.



URL(*) http://localhost:8000/textured/cesme
Name(*) Çeşme 3D Textured
thematicDataUrl https://www.google.com/fusiontables
cityobjectsJsonUrl http://localhost:8000/textured/cesme
minLodPixels 120
maxLodPixels 1.7976931348623157e+308
MaxCountOfVisibleTiles(*) 200
MaxSizeOfCachedTiles(*) 200

ID_10792

| | |
|---|---|
| GMLID | ID_10792 |
| Bina Kodu | 10792 |
| Bina Kullanımı | Ticari |
| Kat Sayısı | 2 |
| Bina Yüksekliği | 7.84 metre |
| Taban Alanı | 883.83 m² |
| Çatı Tipi | Flat |
| X Koordinatı | 439083.7737 |
| Y Koordinatı | 4243467.293 |
| Z Koordinatı | 0.70 metre |
| Koordinat Sistemi | ITRF96 - 6° |
| Modellenme Tarihi | 2017-11-26 |

Figure 6.9. Textured LoD2 buildings and attributes visualizing on 3DCityDB Web Map Client.

The advantages of the storing the data in KML/COLLADA/glTF format are:

- City model can be visualized both on Google Earth or CesiumJS Web Globe
- Attribute information can be changed directly from the spreadsheet
- User-friendly interface and easy use of 3DCityDB Web map Client

The disadvantages of the storing the data in KML/COLLADA/glTF format are:

- No query or styling capabilities
- Building geometry, attribute spreadsheet and textures are generated separately
- Database requirement
- Complex process steps and too many software needs to be used
- No Gzip support for reducing file sizes for better performance

### 6.3 Cesium 3D Tiles

Cesium 3D Tiles [105] is an open source initiative for streaming massive heterogeneous 3D geospatial datasets. It is capable of streaming 3D content like buildings, trees, point clouds, textured meshes and vector data on the CesiumJS virtual globe. It is optimized for streaming and rendering datasets interactively for users. It enables Hierarchical Level of

Detail (HLOD) for performance so only visible tiles on the web are streamed and those tiles are the ones that are the most important for a given 3D view. Instead of relying on zoom levels, 3D Tiles are based on geometric error for Level-of-Detail selection and a tunable pixel error so that the performance is built for multiple zoom levels in the same view.

While CityGML is a versatile and exhaustive data storage and exchange format for city infrastructure and landscapes, storing data in large XML files does not lend itself very well to high performance streaming and rendering. This requires converting CityGML to a runtime format such as 3D Tiles (Figure 6.10). It is also crucial not to lose any semantic information during this conversion, as that is a key feature of CityGML. Currently, there is no open-source tool for converting CityGML to Cesium 3D Tiles. FME 2018.0 software, which provide free educational and research licenses, was used to convert untextured CityGML to Cesium 3D Tiles. A dataset with textures requires a very different approach than one without textures. Another example would be an urban area with dense data versus a dataset that consists mostly of sparse suburbs. Taking performance issues with the textures into the consideration, textured CityGML conversion has been done by Cesium Team.

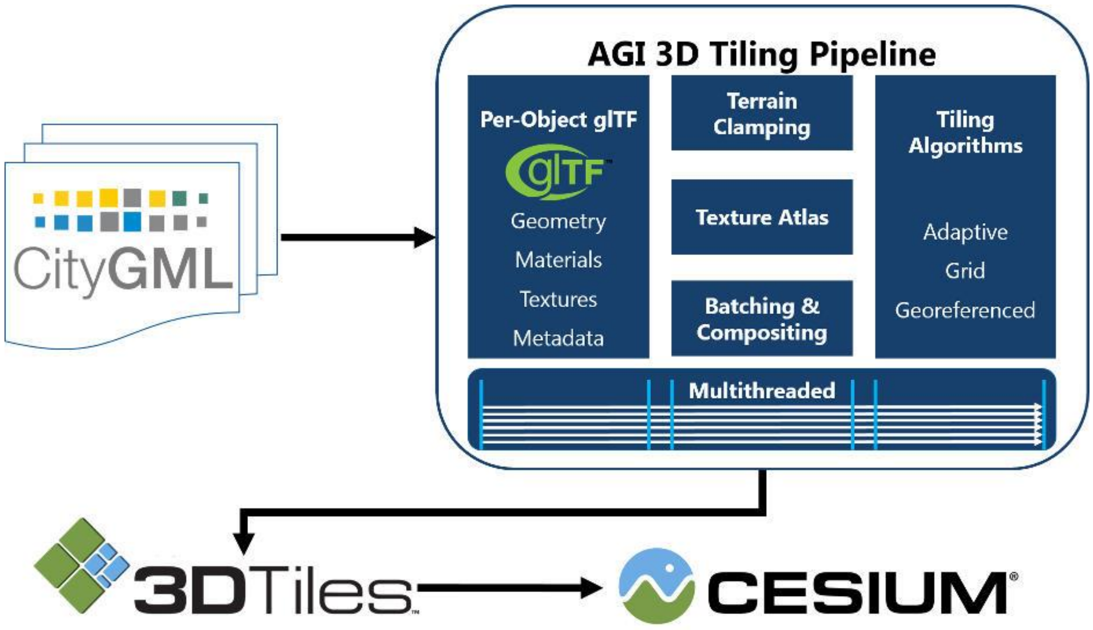


Figure 6.10 An overview of the 3D Tiling pipeline by Cesium.com [90].

The 3D Tiling pipeline supports regular CityGML models, such as buildings, as well as implicit models such as trees, lamps, city furniture, etc. For buildings and other regular models, batched tiles are used, while implicit models are converted to instanced tiles. This allows to batch and combines the data into optimal tiles from both a streaming and a rendering perspective. 3D Tiles Batched 3D Model Tiles [106] are used for CityGML buildings and other models in each tile. Each batch processed tile can contain numerous objects and their metadata. If each building's glTF could be to be stored independently per tile, this would significantly affect runtime performance and make it considerably slower. Combining buildings into a single model drastically improves runtime performance by reducing the rendering overhead. An example of the textured CityGML data converted to 3D Tiles format is given in Figure 6.11

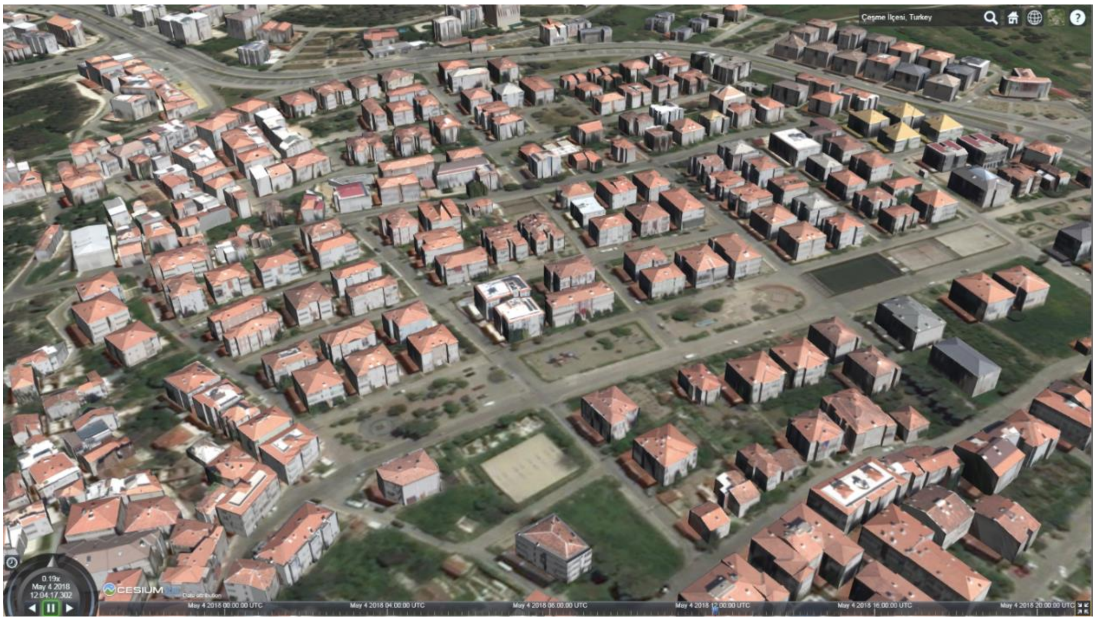
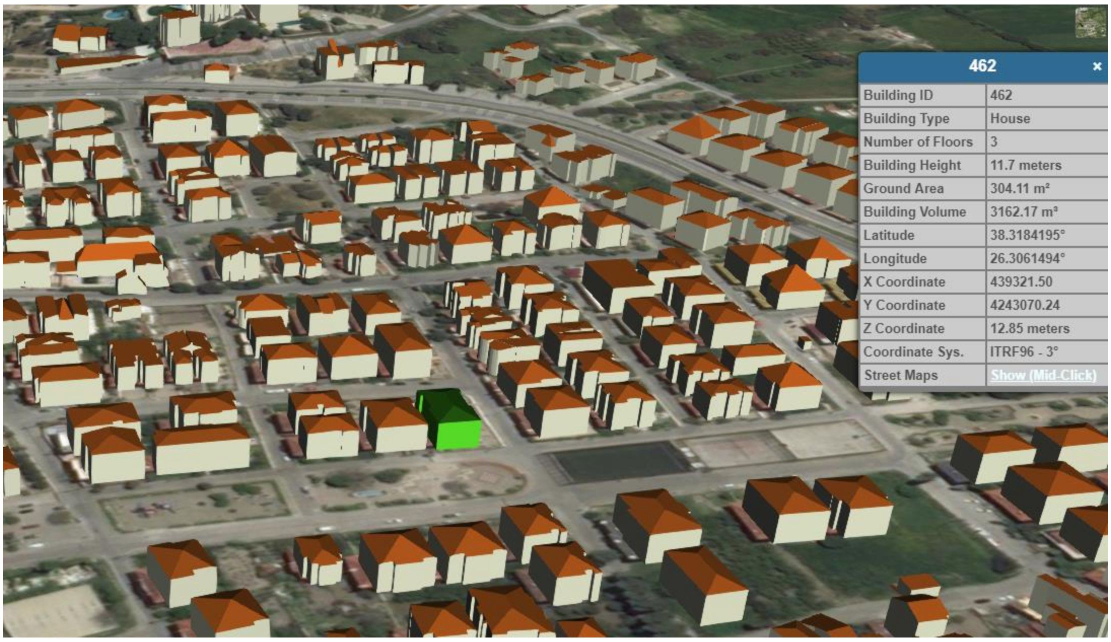


Figure 6.11 Textured CityGML converted to 3D Tiles format and visualized on CesiumJS

Once the models have been combined into batched and instanced tiles, the attribute information of each model is preserved using the 3D Tiles Batch and Feature tables. This allows picking, styling, and querying as if each building were its own model. Another key point while data conversion was the selection of the right attribute types. All attributes with numeric values must be written as "float", and other attributes must be written as "string" type. Otherwise, 3D Tiles styling on web interface will not work. Another important issue was correct georeferencing of the dataset. The study dataset was georeferenced in Turkey's coordinate system ITRF96 – which wasn't included in the coordinate systems list in FME software. This coordinate system has been added manually to coordinate system list. After this step, the untextured CityGML data could be converted into Cesium 3D Tiles format with predefined attributes. Figure 6.12 shows a number of selected untextured LoD2 buildings and their attribute table on the customized web interface.



462

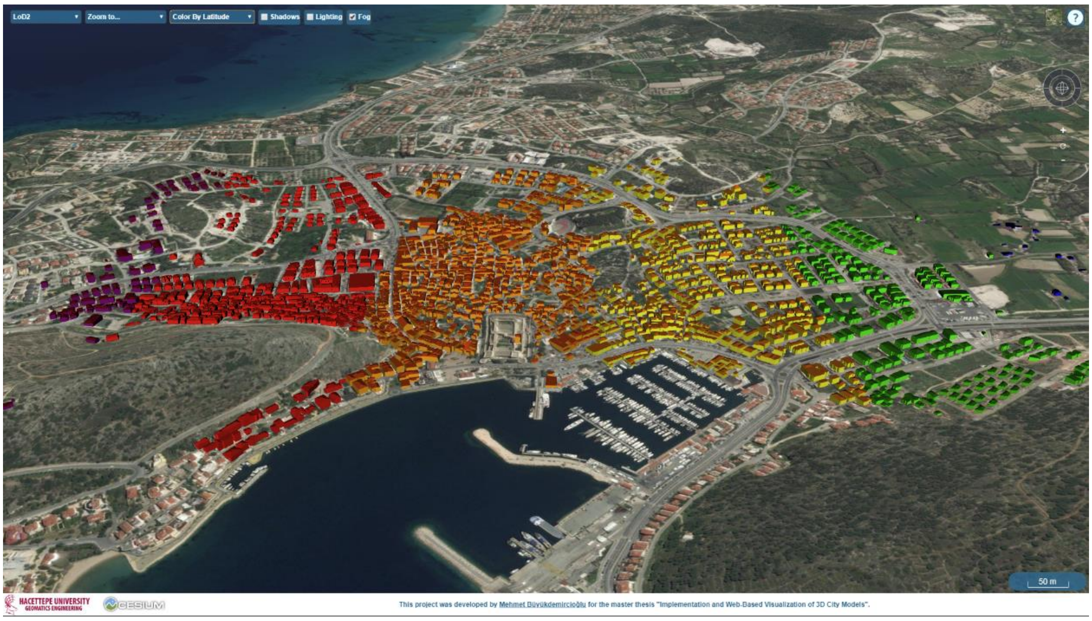
| | |
|---|---|
| Building ID | 462 |
| Building Type | House |
| Number of Floors | 3 |
| Building Height | 11.7 meters |
| Ground Area | 304.11 $m^2$ |
| Building Volume | 3162.17 $m^3$ |
| Latitude | 38.3184195° |
| Longitude | 26.3061494° |
| X Coordinate | 439321.50 |
| Y Coordinate | 4243070.24 |
| Z Coordinate | 12.85 meters |
| Coordinate Sys. | ITRF96 - 3° |
| Street Maps | Show (Mid-Click) |

Figure 6.12 Showing selected untextured LoD2 buildings to attribute table on the customized web interface.

After the generation of the buildings with different levels of detail and conversion into 3D Tiles format, a new web interface has been developed using CesiumJS library and HTML5. In the new interface, as shown in Figure 6.12, it has become possible to query the attribute information of the buildings. The following queries are added to the interface:

- grouping the buildings with respect to their heights and displaying each group with a different building color (Figure 6.13)
- displaying other types of objects according to their elevation
- measuring building heights directly on the displayed model
- displaying selected buildings on Google Street Maps
- providing visual effects such as building shadows and fog
- the ability to select between 2 different 1-meter resolution DTM from the drop-down menu, and
- many other features

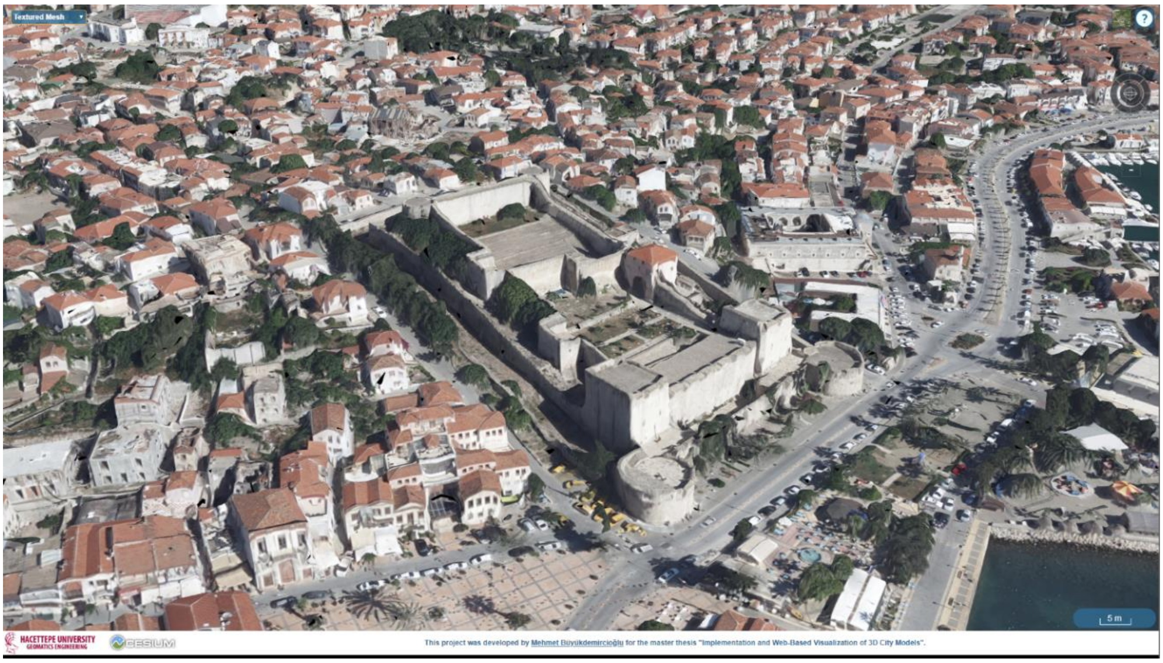
The model is published online at www.cesme3d.com. Figures 6.13, 6.14 and 6.15 show different visualization aspects of the implemented model and the interface.



This project was developed by Mehmet Büyükdemircioğlu for the master thesis "Implementation and Web-Based Visualization of 3D City Models".

Figure 6.13 Çeşme city model colored grouped by their latitude using 3D Tiles styling.

Cesium 3D Tiles format also supports point clouds and textured mesh models. The tiled model can be generated based on the point cloud or mesh data. This hierarchical tile format offers a high-performance visualization for large-scale city modeling. It allows high-performance and high-resolution visualization of city-level 3D models. Tiled model is build based on dense point cloud, or mesh, or depth maps data. Tiled model is textured from source data. Agisoft Photoscan Pro software can directly export point clouds and textured mesh models in Cesium 3D Tiles format, ready for publishing at the web.



This project was developed by Mehmet Büyükdemircioğlu for the master thesis "Implementation and Web-Based Visualization of 3D City Models".

Figure 6.14 Textured mesh visualization on Cesium Web Globe

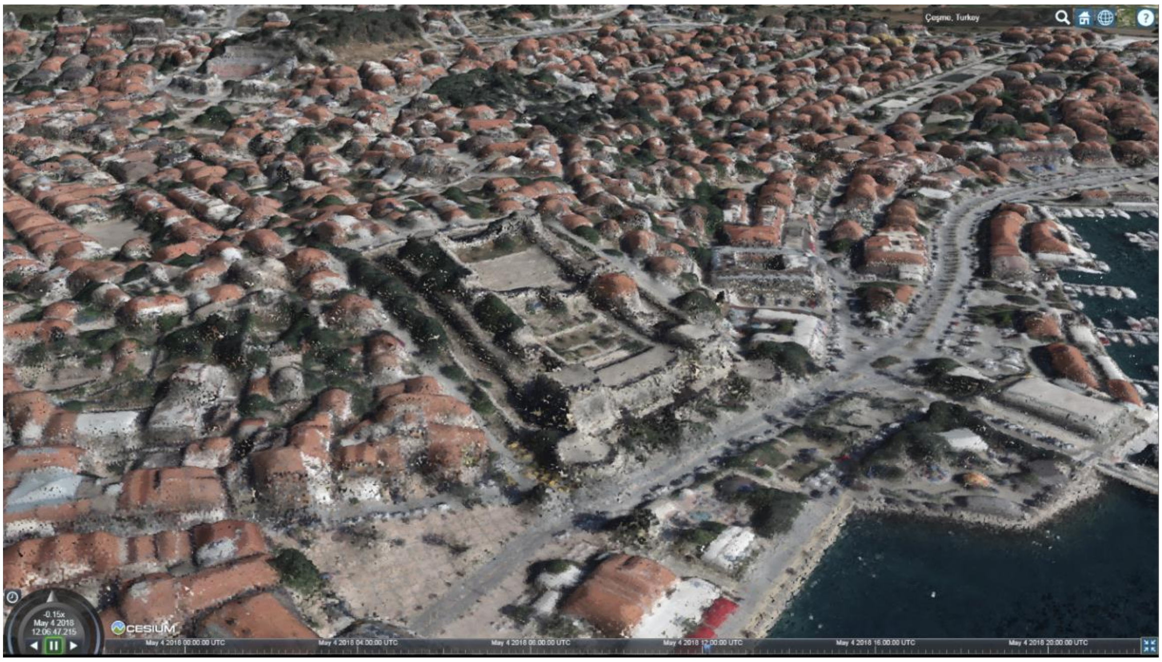


Figure 6.15 Point cloud visualization on Cesium Web Globe

The advantages of storing data in Cesium 3D Tiles format are:

- High performance while visualizing large and complex datasets
- Point cloud and Textured mesh support
- No database or many software requirements for implementation
- Building geometry attributes and textured stored in the same file
- Gzip support for reducing file size
- Supports styling such as color or show
- Direct conversion from CityGML to 3D Tiles with FME software

The disadvantages of storing data in Cesium 3D Tiles format are:

- Only can be visualized on Cesium Web Globe
- Attribute information of the buildings cannot be changed
- No open-source conversion tool from CityGML to 3D Tiles

## 6.4 Terrain Visualization with Cesium

Terrain rendering is important for a large range of game and simulation applications. Computer hardware and algorithms improve rapidly, which enhances our ability to render more realistic terrains. The shift in recent years in consumer hardware from CPU bound graphics pipelines to high-speed special purpose graphics processing units (GPUs) has drastically changed the tradeoffs in the geometric level of detail (LOD) terrain rendering algorithms. These new tradeoffs have sparked the development of GPU-oriented aggregated LOD algorithms. Furthermore, the increased quantity and detail of the data sets to be rendered has raised the demand for good renderers and highlighted the importance of handling data sets which do not fit in RAM. Meanwhile, the advances in shading and image quality, in general, have raised the end-users expectations for image quality and lowered their tolerance for visual artifacts.

The ideal terrain renderer would let the viewer stand at the summit of a tall mountain and see the surrounding countryside in perfect detail. The viewer would freely move around the scene, walking through dense forests or flying rapidly through the air directly to some distant spot, where both the local surroundings and distant vistas would always be perfectly detailed to within the limits of the display. The place represented by the ideal terrain renderer could be a faithful model of part of the real world or some imaginary locale, but in either case should be visually rich and interesting. At no point should the viewer be aware of any distracting artifacts arising from the shuffling of data or reorganizing of geometry within the renderer itself.

Using triangulated terrain with different zoom levels has several important advantages:

- Efficient use of TIN structure
- Textures integrating accordingly to the LOD of the terrain
- Better terrain representation at rough terrain
- High performance and smooth vertex morphing
- Low CPU and hardware usage

The negative characteristics include:

- Terrain data must be pre-processed
- Only static terrain data can be visualized
- Generates more TINs than a basic algorithm for the same screen-space error.

- Sometimes data sizes are unnecessary big, depending on lossyness of data.

Terrain rendering is a smaller, but important sub-problem of constructing the ideal terrain renderer described above. The terrain is not strictly necessary, and in fact are not sufficient in themselves for ideal rendering, especially when the viewer gets closer to the ground, but the terrain format is a very useful approximation. It embodies a basic method of compression and a framework for organizing data and textures. There are a number of good solutions for terrain level of detail rendering which effectively decreases the count of triangles which need to be rendered, at the expense of doing some run-time meshing calculations.

Cesium is capable of visualizing and streaming DTMs. Visualizing the world with the terrain model uses the advantage of showing terrain features and objects with high slope such as mountains compared to 2D maps. Cesium only supports displaying one terrain model at a time, so a single raster terrain file should be converted into supported formats for visualizing big areas. Cesium currently supports two different formats for visualizing terrain: heightmap 1.0 and quantized-mesh 1.0.

The heightmap 1.0 format (Figure 6.16) is simple multi-resolution quadtree pyramid of heightmaps according to the Tile Map Service (TMS) layout and global-geodetic profile. The tiles are 65x65 vertices and overlaps with neighboring tiles at border points where they intersect to form a continuous terrain model. In other words, at the root, the eastern-most column of heights in the western tile is identical to the western-most column of heights in the eastern tile. Cesium converts height tiles into TIN mesh structure. The main disadvantage of this terrain format is its ability to represent the terrain with different slopes, both rugged and rough terrain with regular tiles and same amount of data.

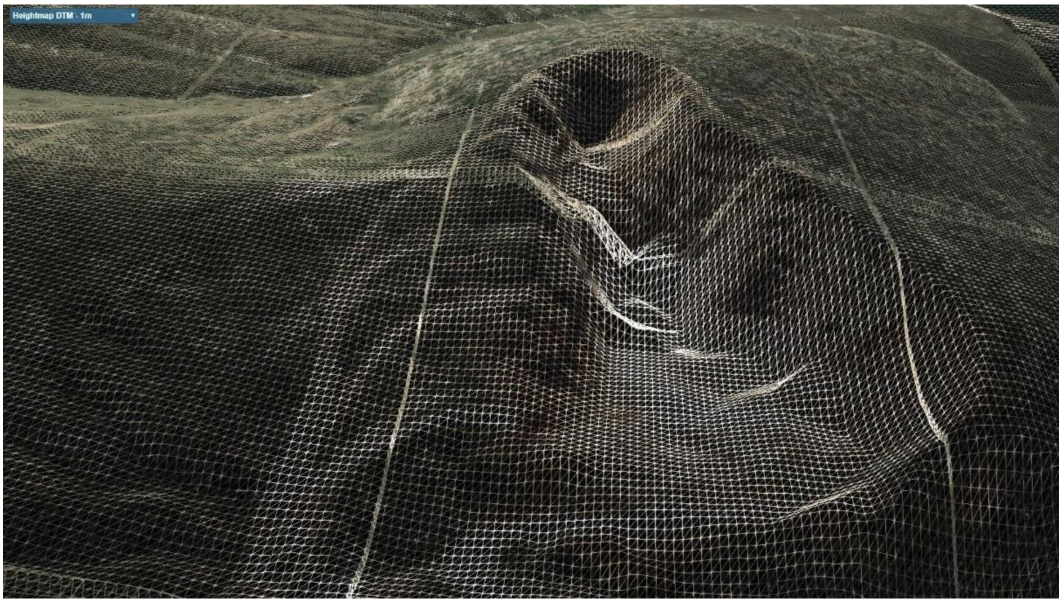


Figure 6.16 The Heightmap 1.0 format represents terrain with regular grid tiles.

Quantized-mesh 1.0 format (Figure 6.17) represents a terrain structure consisting of irregular triangles. It has the same tile structure as the format's heightmap format, but each tile is better optimized to represent larger-scale areas more accurately. Instead of generating regular dense tiles for representing the terrain, TIN mesh for each tile is generated and pre-rendered for each tile. This structure represents the terrain better compared to the heightmap format with denser triangles in more rugged terrain, while forming larger and less detailed tiles in rough terrain. The quantized-mesh terrain uses less memory and better at rendering terrain.

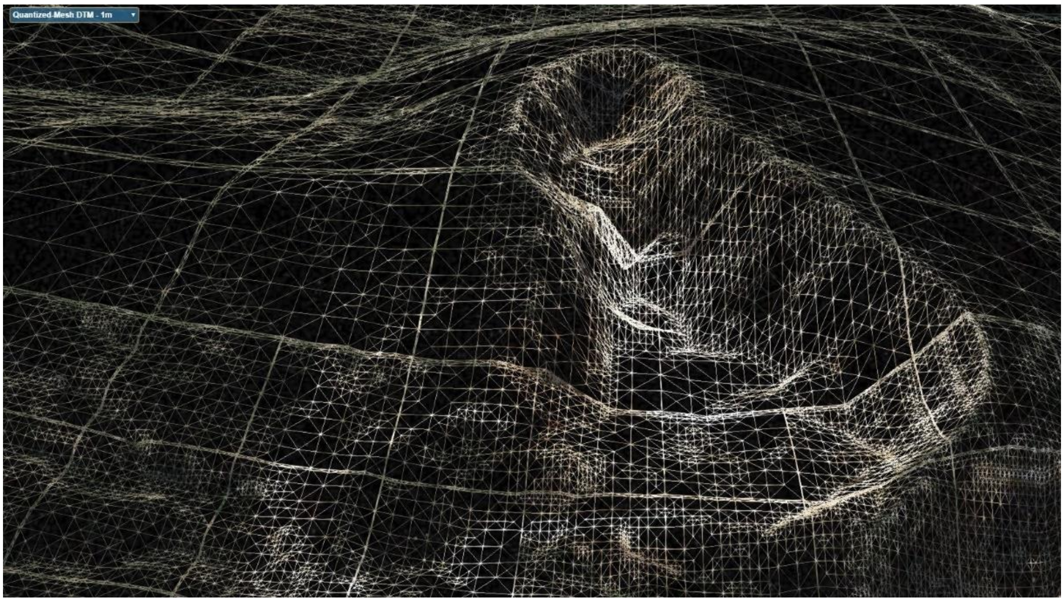
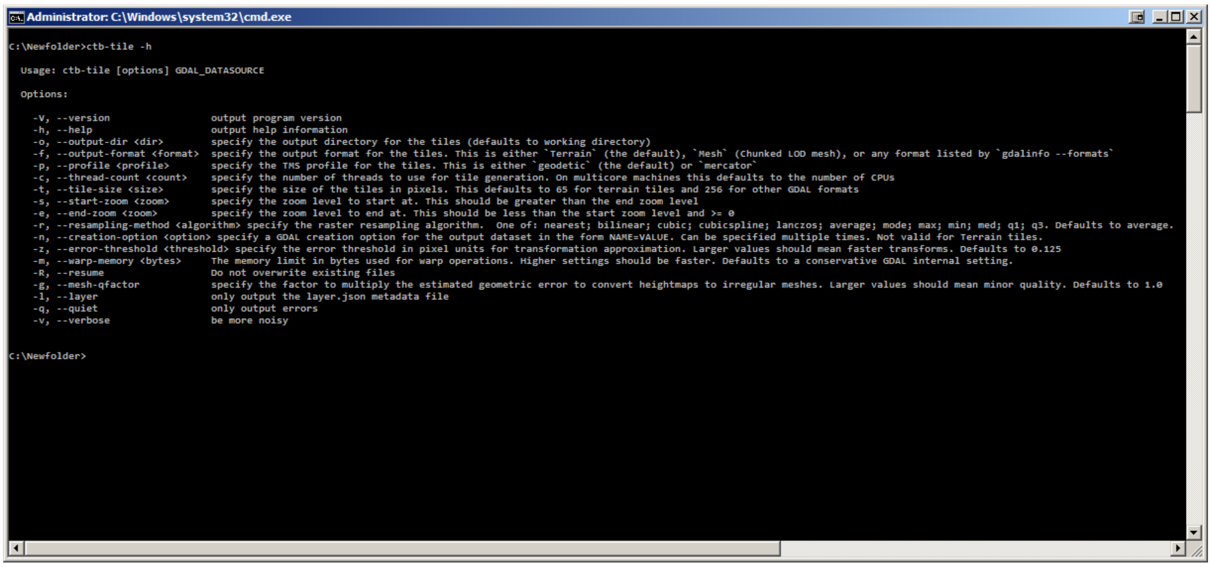


Figure 6.17 Irregular triangle meshes as Quantized-mesh 1.0 format.

Currently, Cesium Terrain Builder [91] is the only open-source tool for generating heightmap and quantized-mesh tiles (Figure 6.18). It is a C++ library for generating terrain tiles for use with the CesiumJS. This library creates gzipped terrain tiles from a GDAL raster representing a terrain, saving the resulting tiles to a directory. The code compiled under Windows using Visual Studio 2017 and GDAL. After compiling, the software would be ready to use.

The input raster should contain data representing elevations relative to sea level. NODATA (null) values are not currently dealt with: these should be filled using interpolation in a data preprocessing step. In the case of multiband rasters, only the first band is used as the input DTM. Before generating tiles from input raster, it is recommended that the input raster is in the same spatial reference system as the output tile grid in order to bypass the need to reproject the data. For terrain data, this is World Geodetic System (WGS 84). If the source data is in another spatial reference system, however, the tool will attempt to reproject the data but with an associated performance penalty. For this recommendation, the input raster dataset reprojected to WGS84 coordinate system from ITRF96 coordinate system with FME.



```
C:\Newfolder>ctb-tile -h

  Usage: ctb-tile [options] GDAL_DATASOURCE

  Options:

    -V, --version                output program version
    -h, --help                   output help information
    -o, --output-dir <dir>       specify the output directory for the tiles (defaults to working directory)
    -f, --output-format <format> specify the output format for the tiles. This is either `Terrain` (the default), `Mesh` (Chunked LOD mesh), or any format listed by `gdalinfo --formats`
    -p, --profile <profile>      specify the TMS profile for the tiles. This is either `geodetic` (the default) or `mercator`
    -c, --thread-count <count>   specify the number of threads to use for tile generation. On multicore machines this defaults to the number of CPUs
    -t, --tile-size <size>       specify the size of the tiles in pixels. This defaults to 65 for terrain tiles and 256 for other GDAL formats
    -s, --start-zoom <zoom>      specify the zoom level to start at. This should be greater than the end zoom level
    -e, --end-zoom <zoom>        specify the zoom level to end at. This should be less than the start zoom level and >= 0
    -r, --resampling-method <algorithm> specify the raster resampling algorithm.  One of: nearest; bilinear; cubic; cubicspline; lanczos; average; mode; max; min; med; q1; q3. Defaults to average.
    -n, --creation-option <option> specify a GDAL creation option for the output dataset in the form NAME=VALUE. Can be specified multiple times. Not valid for Terrain tiles.
    -z, --error-threshold <threshold> specify the error threshold in pixel units for transformation approximation. Larger values should mean faster transforms. Defaults to 0.125
    -m, --warp-memory <bytes>    The memory limit in bytes used for warp operations. Higher settings should be faster. Defaults to a conservative GDAL internal setting.
    -R, --resume                 Do not overwrite existing files
    -g, --mesh-qfactor           specify the factor to multiply the estimated geometric error to convert heightmaps to irregular meshes. Larger values should mean minor quality. Defaults to 1.0
    -l, --layer                  only output the layer.json metadata file
    -q, --quiet                  only output errors
    -v, --verbose                be more noisy


C:\Newfolder>
```

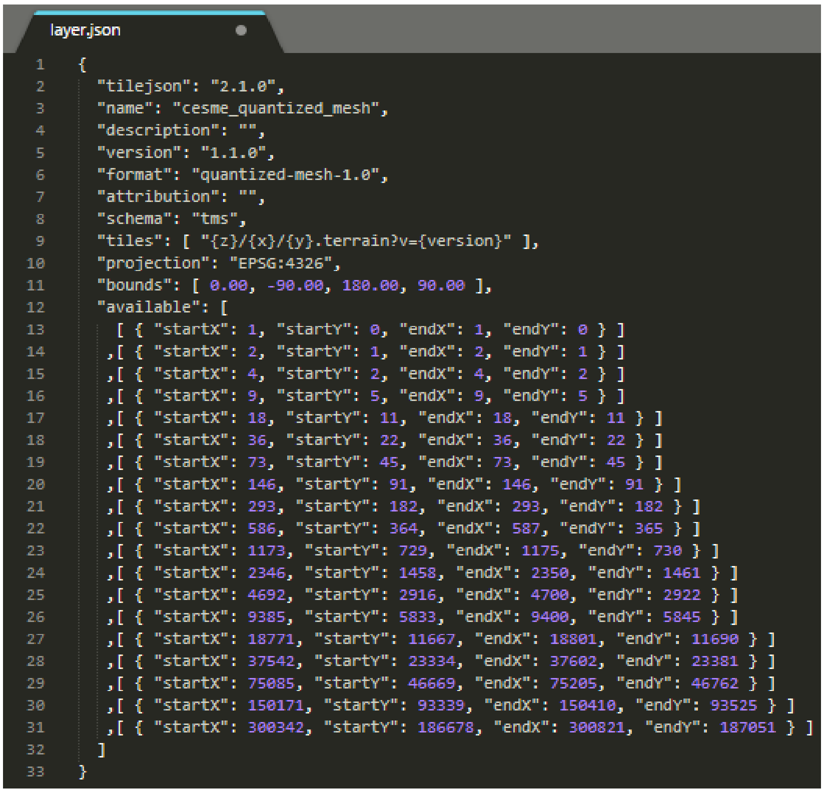
Figure 6.18 Cesium Terrain Builder command list

“ctb-tile” command will resample data from the source dataset when generating tilesets for the various zoom levels. This can lead to performance issues and datatype overflows at lower zoom levels (e.g. level 0) when the source dataset is very large. To overcome this the tool can be used on the original dataset to only create the tileset at the highest zoom level (e.g. level 18) using the --start-zoom and --end-zoom options. Once this tileset is generated it can be turned into a GDAL Virtual Raster dataset for creating the next zoom level down (e.g. level 17). Repeating this process until the lowest zoom level is created means that the resampling is much more efficient (e.g. level 0 would be created from a VRT representation of level 1). Because terrain tiles are not a format supported by VRT datasets you will need to perform this process in order to create tiles in a GDAL DEM format as an intermediate step. VRT representations of these intermediate tilesets can then be used to create the final terrain tile output.

Quantized-mesh 1.0 : ctb-tile --output-dir ./qmesh --output-format Mesh --start-zoom 19 --end-zoom 0 cesme_wgs84.img

Heightmap 1.0 : ctb-tile --output-dir ./heightmap --output-format Terrain --start-zoom 19 --end-zoom 0 cesme_wgs84.img

Using previous commands, 1-meter resolution digital terrain model in 2 different formats and 20 zoom levels was created. But it doesn’t satisfy the requirements for visualizing terrain. And to finish, the "-l" (or "--layer") option will allow you to create the "layer.json" file with all metadata information that describes the tileset (Figure 6.19). It fills the "available" json-attribute too, it seems mandatory to successfully load the terrain in Cesium.



```
{
  "tilejson": "2.1.0",
  "name": "cesme_quantized_mesh",
  "description": "",
  "version": "1.1.0",
  "format": "quantized-mesh-1.0",
  "attribution": "",
  "schema": "tms",
  "tiles": [ "{z}/{x}/{y}.terrain?v={version}" ],
  "projection": "EPSG:4326",
  "bounds": [ 0.00, -90.00, 180.00, 90.00 ],
  "available": [
      [ { "startX": 1, "startY": 0, "endX": 1, "endY": 0 } ]
     ,[ { "startX": 2, "startY": 1, "endX": 2, "endY": 1 } ]
     ,[ { "startX": 4, "startY": 2, "endX": 4, "endY": 2 } ]
     ,[ { "startX": 9, "startY": 5, "endX": 9, "endY": 5 } ]
     ,[ { "startX": 18, "startY": 11, "endX": 18, "endY": 11 } ]
     ,[ { "startX": 36, "startY": 22, "endX": 36, "endY": 22 } ]
     ,[ { "startX": 73, "startY": 45, "endX": 73, "endY": 45 } ]
     ,[ { "startX": 146, "startY": 91, "endX": 146, "endY": 91 } ]
     ,[ { "startX": 293, "startY": 182, "endX": 293, "endY": 182 } ]
     ,[ { "startX": 586, "startY": 364, "endX": 587, "endY": 365 } ]
     ,[ { "startX": 1173, "startY": 729, "endX": 1175, "endY": 730 } ]
     ,[ { "startX": 2346, "startY": 1458, "endX": 2350, "endY": 1461 } ]
     ,[ { "startX": 4692, "startY": 2916, "endX": 4700, "endY": 2922 } ]
     ,[ { "startX": 9385, "startY": 5833, "endX": 9400, "endY": 5845 } ]
     ,[ { "startX": 18771, "startY": 11667, "endX": 18801, "endY": 11690 } ]
     ,[ { "startX": 37542, "startY": 23334, "endX": 37602, "endY": 23381 } ]
     ,[ { "startX": 75085, "startY": 46669, "endX": 75205, "endY": 46762 } ]
     ,[ { "startX": 150171, "startY": 93339, "endX": 150410, "endY": 93525 } ]
     ,[ { "startX": 300342, "startY": 186678, "endX": 300821, "endY": 187051 } ]
  ]
}
```

Figure 6.19 layer.json file with all metadata information that describes the tileset.

To visualize terrain data in CesiumJS, generated terrain data must be served on a Cross-Origin Resource Sharing (CORS) enabled server. For this purpose, terrain data served on a CORS-enabled local Apache server. Once the server settings have been set and the CORS activated, we can visualize our terrain on Cesium. A view generated 1m resolution DTM on Cesium is given in Figure 6.20.

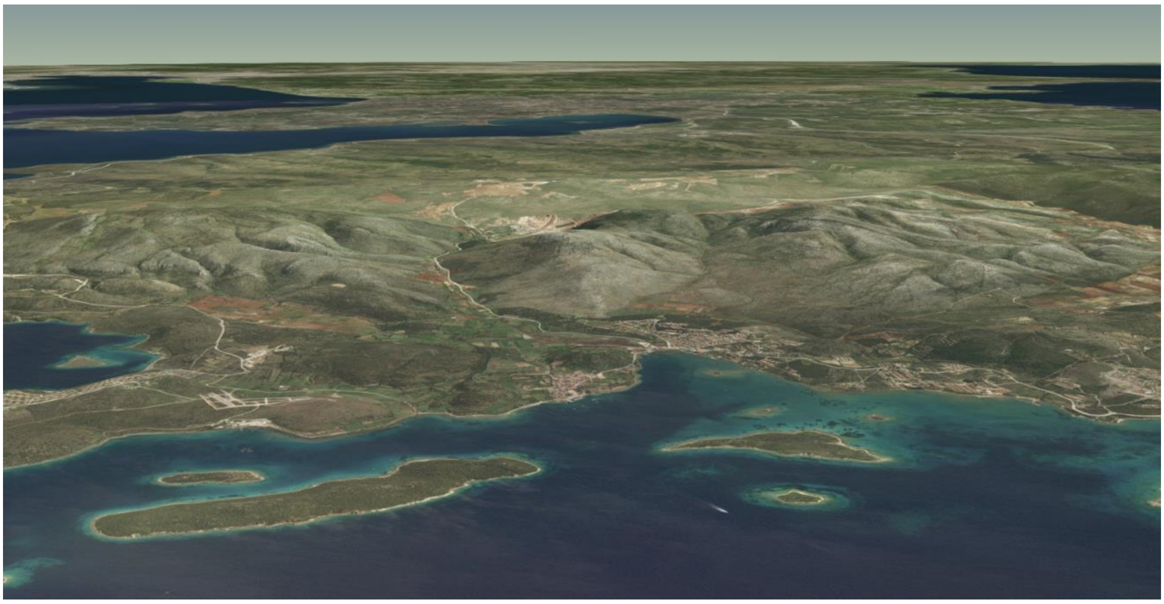


Figure 6.20 A view of generated 1m resolution DTM on Cesium

After generating DTM for the whole project area, 6.4 GB raster terrain size reduced to 940 MB for Heightmap 1.0 format and 938 MB for Quantized-mesh 1.0 format. A DTM with a resolution of 1 meter or better is usually required to clamp building to the terrain. If a lower resolution DTM is used, some parts of the buildings or whole building may be buried in the ground or building could hang in the air. If a 3D city model will be visualized without a terrain model, reducing the altitudes of the buildings to zero or WGS84 ellipsoid will prevent the buildings hanging on the air. As a result, a high-resolution digital terrain model will always be required to precisely and effectively visualization. A 3D city model without an underlying Digital Terrain Model will always be visually incomplete. Examples to the terrain visualization from the project are given in Figures 6.21 and 6.22.

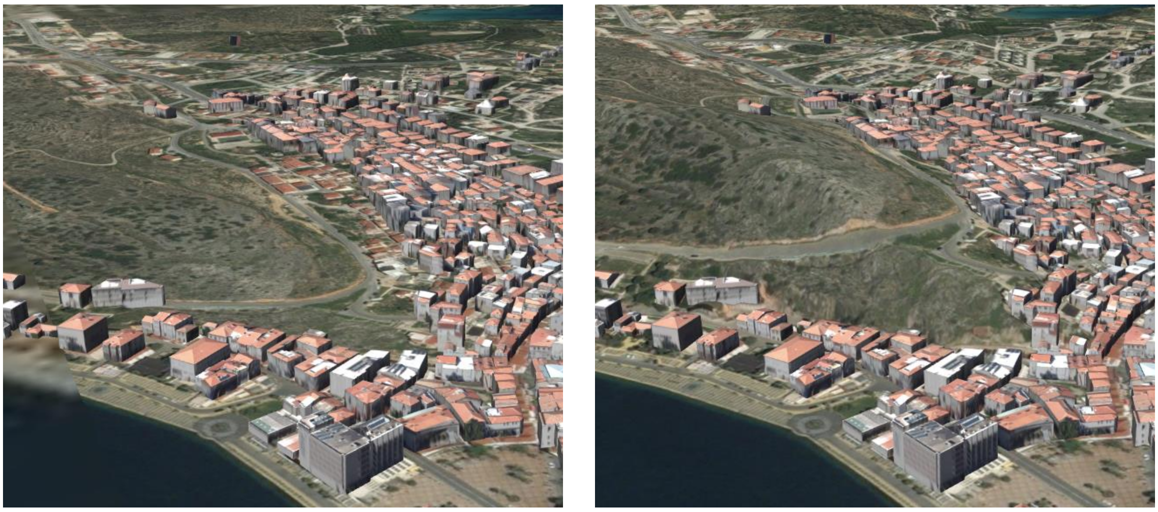


Figure 6.21 Buildings hanging in the air without terrain (left) and buildings clamped to terrain (right)

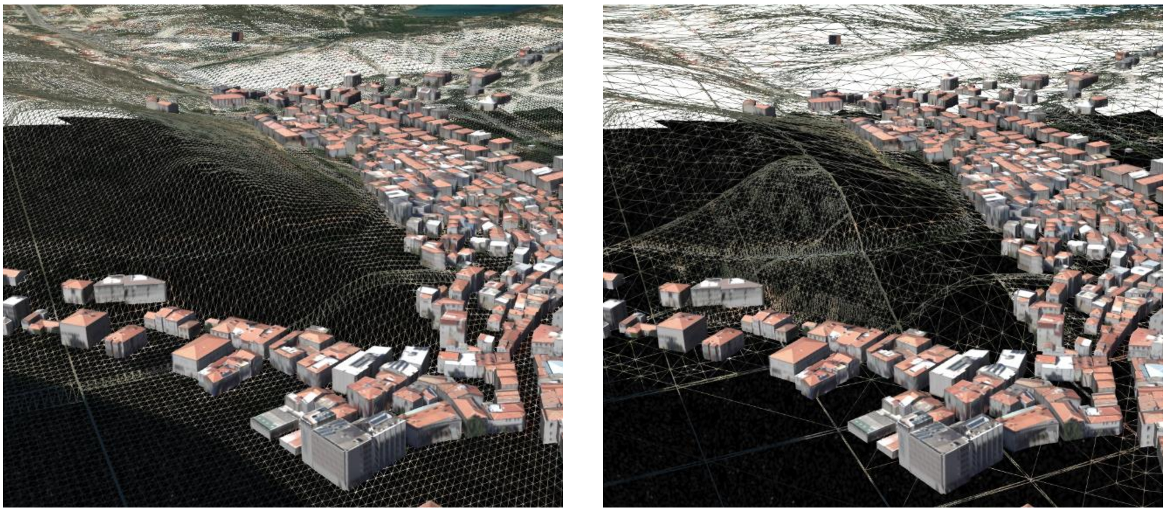
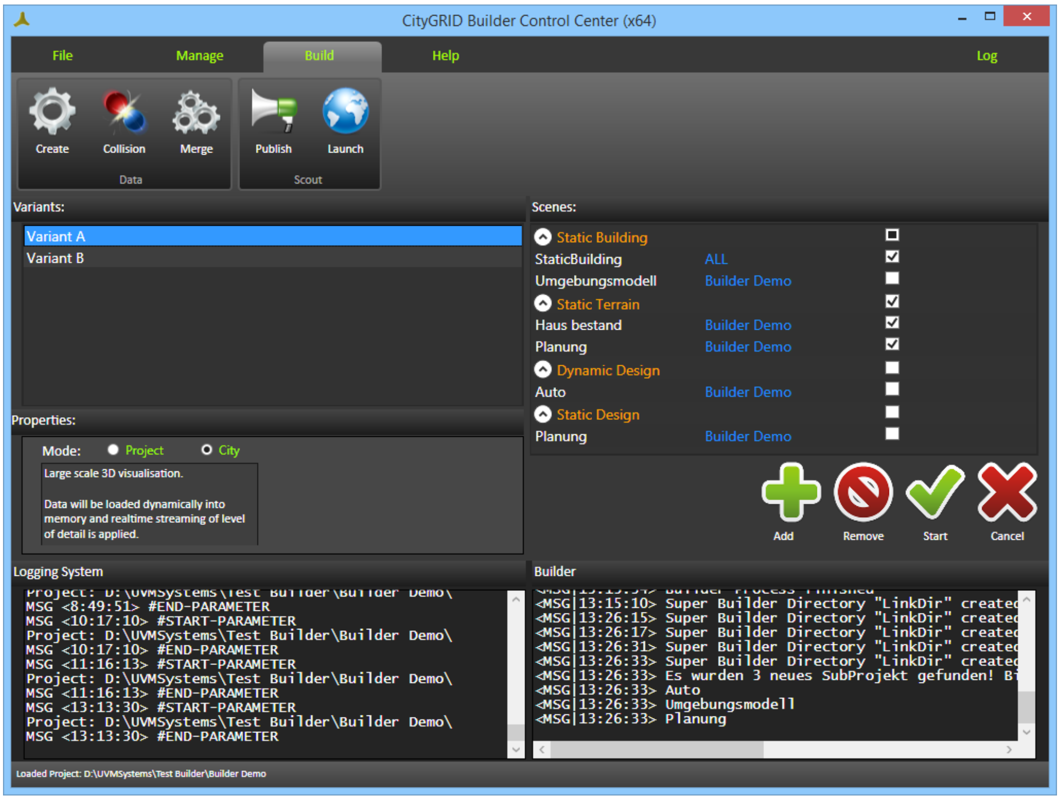


Figure 6.22 Wireframe view of terrain in Heightmap format (left) and Quantized-mesh format (right)

# 7. VISUALIZATION BASED ON UNITY GAME ENGINE

Unity is a cross-platform game engine designed to support and develops 2D and 3D video games, simulations for computers, virtual reality, consoles and mobile devices platform [107]. It is commercial software developed by Unity Technologies. Unity 3D provides three dimensional manipulation and simulation through functions defined using programming languages. The main advantage of visualizing city models with Unity game is it can run in different operating systems such as Windows, Mac or Linux.

CityGRID Builder (Figure 6.23) software was used to visualize 3D city models in the Unity game engine. The main function of this software is to create scenes of 3D city model in Unity game engine for exploring cities with Virtual Reality googles. The imported textured city model and terrain model can be exported as scene after optimization.



Variants:

Variant A
Variant B

Scenes:

| | | |
|---|---|---|
| Static Building | | ☐ |
| StaticBuilding | ALL | ☑ |
| Umgebungsmodell | Builder Demo | ☐ |
| Static Terrain | | ☑ |
| Haus bestand | Builder Demo | ☑ |
| Planung | Builder Demo | ☑ |
| Dynamic Design | | ☐ |
| Auto | Builder Demo | ☐ |
| Static Design | | ☐ |
| Planung | Builder Demo | ☐ |

Properties:

Mode: ○ Project ● City

Large scale 3D visualisation.

Data will be loaded dynamically into memory and realtime streaming of level of detail is applied.

Logging System

```
Project: D:\UVMSystems\Test Builder\Builder Demo\
MSG <8:49:51> #END-PARAMETER
MSG <10:17:10> #START-PARAMETER
Project: D:\UVMSystems\Test Builder\Builder Demo\
MSG <10:17:10> #END-PARAMETER
MSG <11:16:13> #START-PARAMETER
Project: D:\UVMSystems\Test Builder\Builder Demo\
MSG <11:16:13> #END-PARAMETER
MSG <13:13:30> #START-PARAMETER
Project: D:\UVMSystems\Test Builder\Builder Demo\
MSG <13:13:30> #END-PARAMETER
```

Builder

```
<MSG|13:15:10> Super Builder Directory "LinkDir" create
<MSG|13:26:15> Super Builder Directory "LinkDir" create
<MSG|13:26:17> Super Builder Directory "LinkDir" create
<MSG|13:26:31> Super Builder Directory "LinkDir" create
<MSG|13:26:33> Super Builder Directory "LinkDir" create
<MSG|13:26:33> Es wurden 3 neues SubProjekt gefunden! Bi
<MSG|13:26:33> Auto
<MSG|13:26:33> Umgebungsmodell
<MSG|13:26:33> Planung
```

Loaded Project: D:\UVMSystems\Test Builder\Builder Demo

Figure 6.23 CityGRID Builder control center

All scene data from FME and 3D Studio Builder have been optimized and restructured prior to loading in CityGRID Scout. In this process, the previous files have been resolved and combined to variants. In addition LoD levels of geometry and images are created, data trees (voxel octree) are formed and new files which are optimized for performance will be generated. "Scenario" function is used to present in real-time in CityGRID Scout with scene difference combinations and switch between scenarios.

A scene from the Builder project can be used in multiple scenarios. Builder Projects can contain multiple scenarios. Scenarios determine when to show or turn off which data is running Scout. The data are calculated once and sent to the graphics card when needed (when the scenario is changed).

Depending on the size of the Scout project, there are two alternative optimization methods: Project and City. These two methods are offered in the Properties Window. The two methods process the Scout data in different ways and have different demands on used software modules and computing times. Also, the data handling in the Scout is fundamentally different. The processing mode Project is intended for local, small-scale projects. All data are processed so that they can be completely held in the main memory or graphics memory. Applying processing mode Project, there will be no reloading of data. The textures are processed in different resolutions (LOD) and the geometry files generated are optimized for the Scout. At the start of the project Scout determines the amount of available storage and automatically selects the possible resolution of the textures. The geometry is always displayed with the full amount of data.

The process mode City is available for the visualization of large-scale projects (entire cities or regions) with high data density. In contrast to Project mode, Scout now applies a distance-dependent visualization of the geometry and texture data. Scouts load the data dynamically at runtime, so at close range Scout always visualizes the best possible resolution of geometry and textures. The further away data are from the current focus of Scout project, the more generalized they are displayed. As a prerequisite, CityGRID Builder organizes the geometry data in the form of data trees, which store the data in its original resolution and additionally in generalized Levels of Detail (LoDs). 3D Studio Max functions are used for generating the LoDs. For that reason, a license of 3D Studio Max has to be available at the computer. If no 3D Studio Max is available, just a simplified tree structure can be calculated without the LoD stages. Due to this, the range of visibility will be limited when Scout visualizes this simplified structure. A scene from Unity Game Engine visualization generated with CityGRID Builder is shown in Figure 6.24.

.

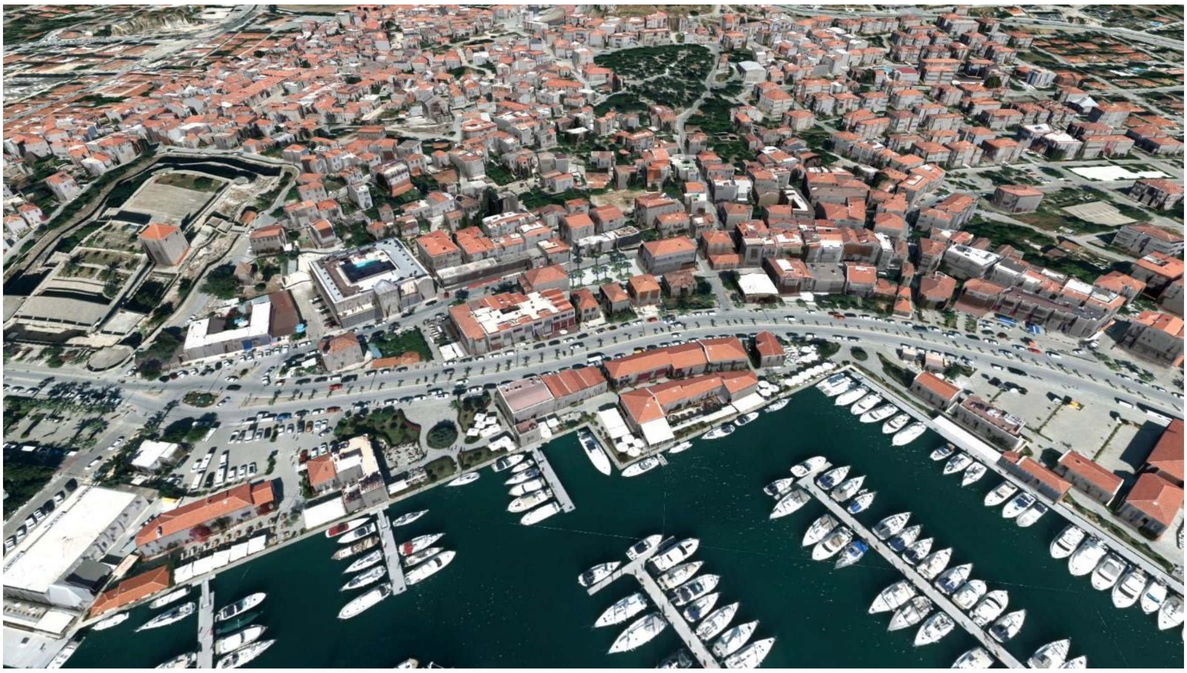


Figure 6.24 A scene from Unity Game Engine visualization generated with CityGRID Builder

Scout projects are interactive and freely navigable 3D scenes and allow viewing the project from every angle. Users can explore the scene with Mouse / Keyboard, Gamecontroller, Body Motion or Virtual Reality Glasses. Produced scenes can be copied and distributed in an unlimited number of times. Some of the features of are listed as below:

- High-performance visualization of 3D city models with terrain
- Shadow effect by daytime
- Coordinate and distance measurement
- Moving object support such as cars, ships or train
- Building queries for attributes
- WMTS map support for navigation
- Support for video sequence render
- Offline visualization

# 8. CONCLUSIONS AND RECOMMENDATIONS

In this thesis, a workflow has been developed for the production and visualization of 3D city models, which have now become a necessity for better management of cities. Large-format aerial imagery and vector maps manually produced from this data have been used for this purpose. During this study, open-source software and other software with free academic licenses have been preferred whenever possible. Potential fixes and improvements have been noted by reviewing the problems encountered. In Turkey, 3D city models in LOD2 and DSMs can be produced using existing aerial photogrammetric datasets and their project outputs without additional cost and effort as shown in this thesis. The created 3D model is automatically textured from aerial images and published online on the web at www.cesme3d.com . Some of the problems encountered during the thesis work and suggestions for their solutions are as follows:

- **Excessive amount of data preprocessing and file format conversion in the project.**

Many of the data used in this project was not directly in the format to be input to the software and needs to be pre-processed. For example, building footprints needs to be converted to ESRI Shapefile from Microstation DGN file and assign unique ID to each building, DSM and DTM file in GeoTiff format to needs to be converted ESRI ASCII Grid (.asc) file, and orthophotos needs to be converted to .jpg format from .tif file. Texturing and other similar processes require continuous file format conversion and modification. This is an inevitable situation, but all file conversions and modifications can be performed with FME software. Although many conversions can be done without FME software, it is easy to do all pre-processing in one software.

- **The building roof models, which are under single or blocks trees or other objects, are reconstructed incorrectly from DSMs.**

In order to fix this problem, which is also depicted in Figure 6.25, the generated DSM must be visually checked quickly and manual editing on the point cloud should be done. After the objects are cleaned, it should be converted to raster DSM and become an input for the BuildingReconstruction software.

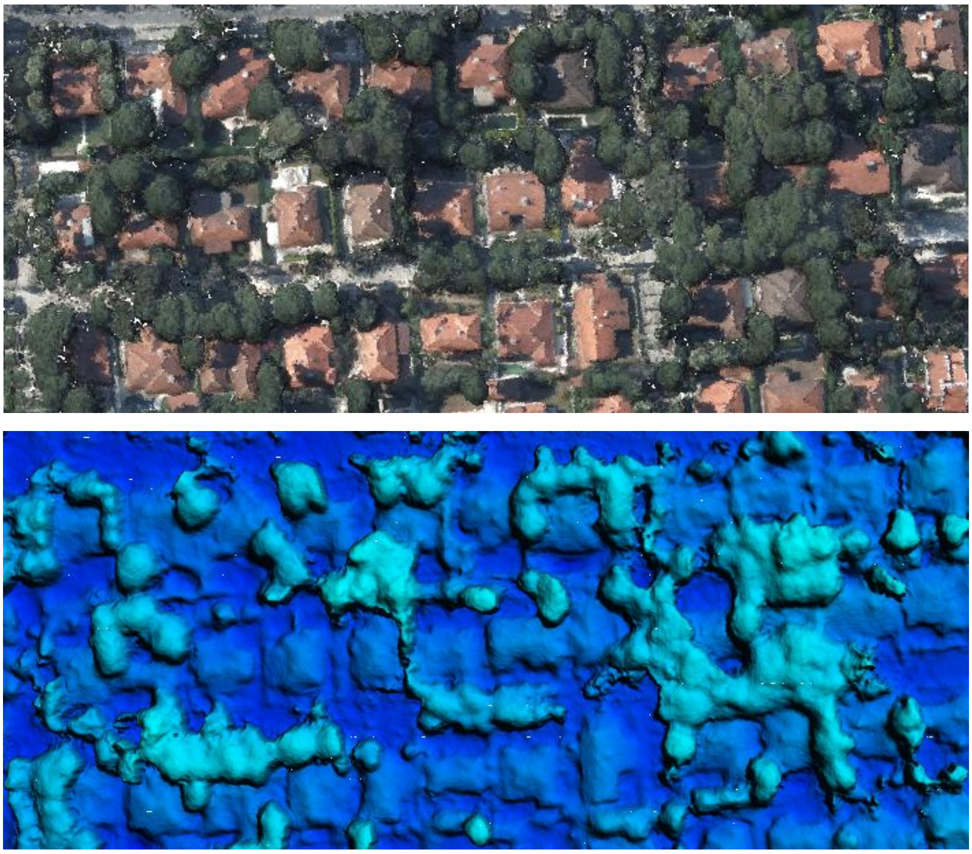


Figure 6.25 Trees over the roofs of the buildings

**• BuildingReconstruction software problems due to topological errors**

If the building footprints do not close properly (as seen in Figure 6.26), BuildingReconstruction software will stop the production due to error. In order to prevent this, all building polygons have to be closed properly. Checking and correcting the closure can be done manually in small areas, but impossible for a large number of buildings. Therefore, it must be checked and corrected automatically with FME software.

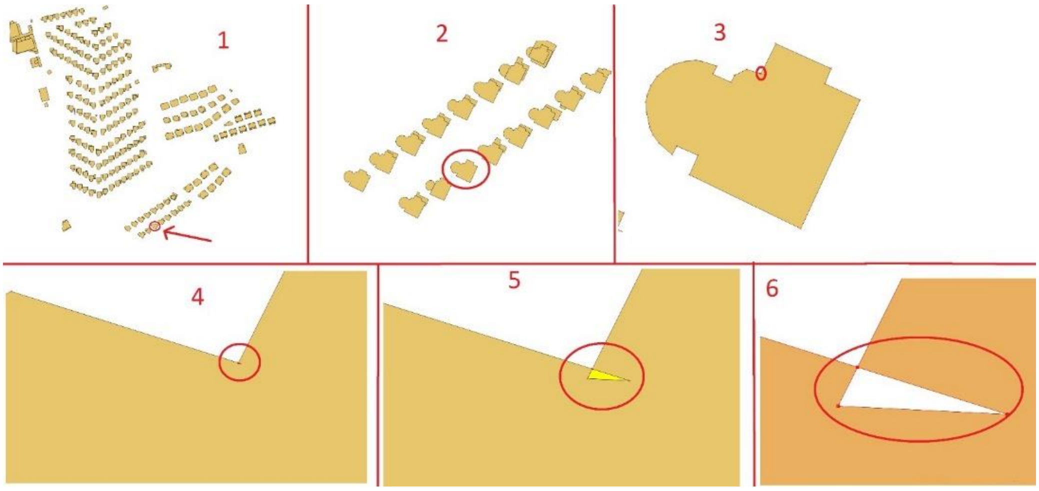


Figure 6.26 An example of unclosed building footprint

**• BuildingReconstruction software limitations in large areas (> 1 km$^2$)**

In this study 43,158 buildings were reconstructed automatically in an area of 270 km2. In order to be able to perform an automatic reconstruction on such a large area, the working area is divided into small pieces and then divided into folders by producing DSM, DTM and building footprint for each area. The segments were separately produced and exported as CityGML, and these separate files were later converted into a single CityGML file covering the entire city with the aid of FME software.

**• The country coordinate system (ITRF96) is not available by default in some software, which obstruct correct georeferencing of the data.**

FME and PostGIS software doesn’t have Turkish Reference Coordinate System (ITRF96) in their coordinate list. The coordinate system of the input data muse be specified for georeference our input data. To prevent this problem, the parameters of the ITRF96 coordinate system are manually entered into the file where the two software programs contain the coordinate systems.

As a final recommendation, an automatic quality control and quality assessment should be implemented for the 3D city models. Such methods are especially crucial for the QC & QC of large datasets.

# REFERENCES

[1] Nations, U., World Urbanization Prospects Highlights. **2014**

[2] Kolbe, T.H., Representing and Exchanging 3D City Models with CityGML. *In 3D geo-information sciences*: p. 15-31, **2009**

[3] Kada, M., The 3D Berlin Project. *Photogrammetric Week '09*, **2009**

[4] Agugiaro, G., First Steps Towards an Integrated Citygml-Based 3d Model of Vienna. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*. III-4: p. 139-146, **2016**

[5] Özerbil, T., et al., Konya Büyükşehir Belediyesi Eğik (Oblique) Görüntü Alımı, 3 Boyutlu Kent Modeli ve 3 Boyutlu Kent Rehberi Projesi. *V. Uzaktan Algılama ve Coğrafi Bilgi Sistemleri Sempozyumu (UZAL-CBS 2014), 14-17 Ekim 2014, İstanbul*, **2014**

[6] Şengül, A., Extracting Semantic Building Models From Aerial Stereo Images and Conversion to CityGML. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. XXXIX-B3, **2012**

[7] Kobayashi, Y., Photogrammetry and 3D City Modeling. *Proceedings of Digital Architecture*, **2006**

[8] Nex, F. and F. Remondino, Automatic Roof Outlines Reconstruction From Photogrammetric DSM. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. I-3(XXII ISPRS Congress, 25 August – 01 September 2012, Melbourne, Australia): p. 257 - 262, **2012**

[9] Novel, C., et al., Comparing aerial photogrammetry and 3d laser scanning methods for creating 3d models of complex objects. *Capturing Reality Forum. Bentley Systems, Salzburg*: p. 15, **2015**

[10] Stadler, A. and T.H. Kolbe, Spatio-semantic coherence in the integration of 3D city models. *In Proceedings of the 5th International Symposium on Spatial Data Quality*, **2007**

[11] Döllner, J., et al., The Virtual 3D City Model of Berlin - Managing, Integrating, and Communicating Complex Urban Information. In Proceedings of the 25th Urban Data Management Symposium UDMS. **2006**

[12] Kolbe, T., G. Gröger, and L. Plümer, CityGML: Interoperable access to 3D city models Geo-information for disaster management. *Springer*: p. 883-899, **2005**

[13] Schilling, A., V. Coors, and K. Laakso, Dynamic 3D maps for mobile tourism applications Map-based Mobile Services. *Springer*: p. 227-239, **2005**

[14] Wagner, D., et al., Geometric-semantical consistency validation of CityGML models. *In Progress and new trends in 3D geoinformation sciences*: p. 171-192, **2013**

[15] Prieto, I., J.L. Izkara, and F.J. Delgado, From point cloud to web 3D through CityGML. *18th international IEEE conference on virtual systems and multimedia (VSMM)*: p. 405-412, **2012**

[16] Ujang, U., et al., 3D Hilbert Space Filling Curves in 3D City Modeling for Faster Spatial Queries. *International Journal of 3-D Information Modeling, 3(2), 1-18, April-June 2014*: p. 1-17, **2014**

[17] Zhu, Q., et al., Research and practice in three-dimensional city modeling. *Geo-spatial Information Science*. 12(1): p. 18-24, **2009**

[18] Kolbe, T., G. Gröger, and L. Plümer, CityGML – 3D city models and their potential for emergency response. *Geospatial information technology for emergency response*. 257, **2008**

[19] Zlatanova, S., 3D GIS for urban development. *International Inst. for Aerospace Survey and Earth Sciences (ITC)*, **2000**

[20] Biljecki, F., et al., Applications of 3D City Models: State of the Art Review. *ISPRS International Journal of Geo-Information*. 4(4): p. 2842-2889, **2015**

[21] Gröger, G. and L. Plümer, CityGML – Interoperable semantic 3D city models. *ISPRS Journal of Photogrammetry and Remote Sensing*. 71: p. 12-33, **2012**

[22] City Geography Markup Language (CityGML) Encoding Standard Version 2.0.0, http://www.opengis.net/spec/citygml/2.0,

[23] Buyukaslih, I., U. Isikdag, and S. Zlatanova, Exploring the processes of generating LOD (0-2) CityGML models in greater municipality of Istanbul. *In 8th 3DGeoInfo Conference & WG II/2 Workshop, Istanbul, Turkey. ISPRS Archives Vol. II-2/W1.*, **2013**

[24] Biljecki, F., H. Ledoux, and J. Stoter, An improved LOD specification for 3D building models. *Computers, Environment and Urban Systems*. 59: p. 25-37, **2016**

[25] Arefi, H., et al., Level of Detail in 3D Building Reconstruction from LIDAR Data. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 37(B3b): p. 485-490, **2008**

[26] Kolbe, T.H., G. Grögerr, and L. Plümer, CityGML - Interoperable Access to 3D City Models. *Proceedings of the International Symposium on Geo-Information for Disaster Management on 21-23 March 2005 in Delft, Springer Vertag*, **2005**

[27] Fan, H. and L. Meng, A Three-Step Approach of Simplifying 3D Buildings Modeled by CityGML. *International Journal of Geographical Information Science*. 26(6): p. 1-18, **2012**

[28] Hu, J., S. You, and N. Ulrich, Approaches to Large-Scale Urban Modeling. *IEEE Computer Society*, **2003**

[29] Suveg, I. and G. Vosselman, Reconstruction of 3D building models from aerial images and maps. *ISPRS Journal of Photogrammetry and Remote Sensing*. 58, **2004**

[30] Kada, M. and L. McKinley, 3D Building Reconstruction From LiDAR Based On a Cell Decomposition Approach. *IAPRS*. 38(3/W4): p. 47 - 52, **2009**

[31] Haala, N. and M. Kada, An update on automatic 3D building reconstruction. *ISPRS Journal of Photogrammetry and Remote Sensing*. 65(6): p. 570-580, **2010**

[32] Haala, N., M. Rothermel, and S. Cavegn, *Extracting 3D urban models from oblique aerial images*, in *2015 Joint Urban Remote Sensing Event (JURSE)*. 2015. p. 1-4.

[33] Kocaman, S., et al., 3D City Modeling From High-Resolution Satellite Images.

[34] Kraus, T., M. Lehner, and P. Reinartz, Modeling of urban areas form high resolution stereo satellite images. *Proceedings of the ISPRS Hannover Workshop 2007 High-Resolution Earth Imaging for Geospatial Information*, **2007**

[35] Flamanc, D., G. Maillet, and H. Jibrini, 3D City Models: An Operational Approach Using Aerial Images And Cadastral Maps. *ISPRS Archives*. XXXIV, **2003**

[36] Kabolizade, M., H. Ebadi, and A. Mohammadzadeh, Design and implementation of an algorithm for automatic 3D reconstruction of building models using genetic algorithm. *International Journal of Applied Earth Observation and Geoinformation*. 19: p. 104-114, **2012**

[37] El Garouani, A., A. Alobeid, and S. El Garouani, Digital surface model based on aerial image stereo pairs for 3D building. *International Journal of Sustainable Built Environment*. 3(1): p. 119-126, **2014**

[38] Yi, C., et al., Urban building reconstruction from raw LiDAR point data. *Computer-Aided Design*. 93: p. 1 - 14, **2017**

[39] Haala, N. and C. Brenner, Extraction of buildings and trees in urban environments. *ISPRS Journal of Photogrammetry & Remote Sensing*. 54: p. 130-137, **1999**

[40] Dorninger, P. and N. Pfeifer, A Comprehensive Automated 3D Approach for Building Extraction, Reconstruction, and Regularization from Airborne Laser Scanning Point Clouds. *Sensors*. 8:11: p. 7323 - 7343, **2008**

[41] Tarsha-Kurdi, F., et al., Model-driven and data-driven approaches using lidar data: analysis and comparison. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*. 36: p. 87 - 92, **2007a**

[42] Huang, H., C. Brenner, and M. Sester, 3d building roof reconstruction from point clouds via generative models. *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*: p. 16 – 24, **2011**

[43] Brenner, C. and N. Haala, Fast Production of Virtual Reality City Models. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*. 32:4, **1998**

[44] Schindler, K. and J. Bauer, A Model-based Method for Building Reconstruction. *Higher-Level Knowledge in 3D Modeling and Motion Analysis*: p. 74 - 82, **2003**

[45] Huang, H., C. Brenner, and M. Sester, A generative statistical approach to automatic 3d building roof reconstruction from laser scanning data. *ISPRS Journal of Photogrammetry and Remote Sensing*. 79: p. 29-43, **2013**

[46] Taillandier, F., Automatic building reconstruction from cadastral maps and aerial images. *International Archives of Photogrammetry and Remote Sensing*. 36(3): p. 105-110, **2005**

[47] Sampath, A. and J. Shan, Segmentation and reconstruction of polyhedral building roofs from aerial lidar point clouds. *IEEE Transactions on geoscience and remote sensing*. 48(3): p. 1554-1567, **2010**

[48] Elberink, S.O., Problems in automated building reconstruction based on dense airborne laser scanning data. *Proceedings of the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 37(B3a): p. 93-98, **2008**

[49] Fan, H., W. Yao, and Q. Fu, Segmentation of sloped roofs from airborne lidar point clouds using ridge-based hierarchical decomposition. *Remote Sensing*. 6(4): p. 3284-3301, **2014**

[50] Karsli, F. and O. Kahya, Building extraction from laser scanning data. *Proceedings of the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 37(B3b): p. 289-294, **2008**

[51] Liu, Y. and Y. Xiong, Automatic segmentation of unorganized noisy point clouds based on the gaussian map. *Computer-Aided Design*. 40(5): p. 576-594, **2008**

[52] Büyüksalih, G. and K. Jacobsen, Comparison of DEM Generation by Very High Resolution Optical Satellites: EARSeL. Band "New Developments and Challenges in Remote Sensing". Rotterdam: Millpress,Warschau. p. 627-637, **2007**

[53] Tack, F., G. Buyuksalih, and R. Goossens, 3D building reconstruction based on given ground plan information and surface models extracted from spaceborne imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*. 67(52-64), **2012**

[54] Oude Elberink, S. and G. Vosselman, Quality analysis on 3D building models reconstructed from airbone laser scanning data. *ISPRS Journal of Photogrammetry and Remote Sensing*. 66: p. 157 - 165, **2011**

[55] Rothermel, M., et al., SURE: Photogrammetric Surface Reconstruction from Imagery.

[56] Hirschmüller, H., Stereo processing by semiglobal matching and mutual information. *EEE Transactions on Pattern Analysis and Machine Intelligence*. 30: p. 328-341, **2008**

[57] Frueh, C., R. Sammon, and A. Zakhor, Automated Texture Mapping of 3D City Models With Oblique Aerial Imagery.

[58] Früh, C. and A. Zakhor, Constructing 3D City Models by Merging Aerial and Ground Views. *IEEE Computer Graphics and Applications*. 23(6): p. 52-61, **2003**

[59] Kada, M., D. Klinec, and N. Haala, Façade Texturing for Rendering 3D City Models. *ASPRS 2005 Annual Conference March 7 - 11 2005 Baltimore, Maryland*, **2005**

[60] Ellul, C. and J. Altenbuchner, Investigating approaches to improving rendering performance of 3D city models on mobile devices. *Geo-spatial Information Science*. 17(2): p. 73-84, **2014**

[61] Klimke, J., B. Hagedorn, and J. Döllner, Scalable Multi-Platform Distribution of Spatial 3D Contents. *International Journal of 3-D Information Modeling, 3(3), 35-49, July-September 2014*: p. 35-49, **2014**

[62] Buyuksalih, I., et al., 3d Modelling and Visualization Based on the Unity Game Engine – Advantages and Challenges. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*. IV-4/W4: p. 161-166, **2017**

[63] Schultz, R.B., J.J. Kerski, and T.C. Patterson, The use of virtual globes as a spatial teaching tool with suggestions for metadata standards. *Journal of Geography*. 107(1): p. 27 - 34, **2008**

[64] UVM Systems GmbH, http://www.uvmsystems.com/index.php/en/software/soft-city, Last Access: May 2018

[65] Prandi, F., et al., 3D web visualization of huge CityGML models. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. XL-3/W3: p. 601-605, **2015**

[66] Christen, M., S. Nebiker, and B. Loesch, Web-Based Large-Scale 3D-Geovisualisation Using WebGL: The OpenWebGlobe Project. *International Journal of 3-D Information Modeling, 1(3), 16-25, July-September 2012*: p. 16-25, **2012**

[67] Jochem, R. and M. Goetz, Towards Interactive 3D City Models on the Web. *International Journal of 3-D Information Modeling, 1(3), 26-36, July-September 2012*: p. 26-36, **2012**

[68] Gesquière, G. and A. Manin, 3D Visualization of Urban Data Based on CityGML with WebGL. *International Journal of 3-D Information Modeling*. 1(3): p. 1-15, **2012**

[69] Goetz, M. and A. Zipf, Towards Defining a Framework for the Automatic Derivation of 3D CityGML Models from Volunteered Geographic Information. *International Journal of 3-D Information Modeling, 1(2), 1-16, April-June 2012*: p. 1-16, **2012**

[70] Brovelli, M.A., M. Minghini, and G. Zamboni, Three Dimensional Volunteered Geographic Information: A Prototype of a Social Virtual Globe. *International Journal of 3-D Information Modeling, 3(2), 19-34, April-June 2014*: p. 19-34, **2014**

[71] Zardiny, A.Z. and F. Hakimpour, 3D Web Services for Visualization and Data Sharing in 3D Cadastre. *International Journal of 3-D Information Modeling, 4(4), 1-15, October-December 2015*: p. 1-15, **2015**

[72] Bahu, J., et al., Towards a 3D Spatial Urban Energy Modelling Approach. *International Journal of 3-D Information Modeling, 3(3), 1-16, July-September 2014*: p. 1-16, **2014**

[73] Kaden, R. and T. Kolbe, Simulation-Based Total Energy Demand Estimation of Buildings using Semantic 3D City Models. *International Journal of 3-D Information Modeling, 3(2), 35-53, April-June 2014*: p. 35-52, **2014**

[74] Rüppel, U. and K. Schatz, Designing a BIM-based serious game for fire safety evacuation simulations. *Advanced Engineering Informatics*. 25(4): p. 600-611, **2011**

[75] Musliman, I., A. Abdul-Rahman, and V. Coors, 3D Navigation for 3D-GIS — Initial Requirements. *Innovations in 3D Geo Information Systems,Springer, Berlin, Heidelberg,*: p. 259–268, **2006**

[76] Rossmann, J., M. Hoppen, and A. Bücken, GML-Based Data Management and Semantic World Modelling for a 4D Forest Simulation and Information System. *International Journal of 3-D Information Modeling, 3(3), 50-67, July-September 2014*: p. 50-67, **2014**

[77] Portalés, C., J. Lerma, and S. Navarro, Augmented reality and photogrammetry: A synergy to visualize physical and virtual city environments. *ISPRS Journal of Photogrammetry and Remote Sensing*. 65(1): p. 134-142, **2010**

[78] Rau, J. and C. Cheng, A cost-effective strategy for multi-scale photo-realistic building modeling and web-based 3-D GIS applications in real estate. *Computers, Environment and Urban Systems*. 38: p. 35-44, **2013**

[79] Wolff, M. and H. Asche, Towards Geovisual Analysis of Crime Scenes – A 3D Crime Mapping Approach. *Advances in GIScience, Springer, Berlin, Heidelberg*: p. 429-448, **2009**

[80] Yan, J., Advances in Computer-Generated Imagery for Flight Simulation. *IEEE Computer Graphics and Applications*. 5(8): p. 37-51, **1985**

[81] San José, R., J. Pérez, and R. González-Barras, 3D Visualization of Air Quality Data. *Proceedings of the 11th International Conference "Reliability and Statistics in Transportation and Communication" (RelStat'11). Riga, Latvia*: p. 1-9, **2011**

[82] Kemec, S., et al., Selecting 3D urban visualisation models for disaster management: Fethiye tsunami inundation case. *Proceedings of the 3rd International Conference on Cartography and GIS. Nessebar, Bulgaria.*, **2010**

[83] Vexcel Imaging Ultracam Falcon, http://www.vexcel-imaging.com/ultracam-falcon/, Last Access: May 2018

[84] 3D City Database, https://www.3dcitydb.org/3dcitydb/3dcitydbhomepage/, Last Access: May 2018

[85] Adobe Photoshop CC, https://www.adobe.com/products/photoshop.html, Last Access: May 2018

[86] Agisoft Photoscan Pro, http://www.agisoft.com/features/professional-edition/, Last Access: May 2018

[87] ESRI ArcGIS, https://www.arcgis.com/, Last Access: May 2018

[88] Bentley Microstation, https://www.bentley.com/en/products/product-line/modeling-and-visualization-software/microstation, Last Access: May 2018

[89] VirtualCity Systems BuildingReconstruction, http://www.virtualcitysystems.de/en/products/buildingreconstruction, Last Access: May 2018

[90] CesiumJS, https://cesiumjs.org/, Last Access: May 2018

[91] Cesium Terrain Builder, https://github.com/geo-data/cesium-terrain-builder, Last Access: May 2018

[92] CityGRID Builder, http://www.uvmsystems.com/index.php/en/software/soft-vis, Last Access: May 2018

[93] CityGRID Texturizer, http://www.uvmsystems.com/index.php/en/software/soft-city, Last Access: May 2018

[94] Safe Software FME, https://www.safe.com/, Last Access: May 2018

[95] Google Earth Pro, https://earth.google.com, Last Access: May 2018

[96] Microsoft Visual Studio, https://www.visualstudio.com/, Last Access: May 2018

[97] PostGIS, https://postgis.net/, Last Access: May 2018

[98] Sublime Text 3, https://www.sublimetext.com/3, Last Access: May 2018

[99] QGIS, https://qgis.org/, Last Access: May 2018

[100] Quick Terrain Reader, http://appliedimagery.com/, Last Access: May 2018

[101] XAMPP, https://www.apachefriends.org/index.html, Last Access: May 2018

[102] Cozzi, P. and D. Bagnell, A WebGL globe rendering pipeline. *GPU Pro 4: Advanced Rendering Techniques AK Peters/CRC Press*: p. 39-48, **2013**

[103] Cozzi, P. and C. Riccio, OpenGL Insights. *AK Peters/CRC Press*, **2012**

[104] Chaturvedi, K., Z. Yao, and T.H. Kolbe, Web-based Exploration of and Interaction with Large and Deeply Structured Semantic 3D City Models using HTML5 and WebGL. **2015**

[105] Cesium 3D Tiles, https://github.com/AnalyticalGraphicsInc/3d-tiles, Last Access: May 2018

[106] Cesium Batched 3D Model, https://github.com/AnalyticalGraphicsInc/3d-tiles/tree/master/TileFormats/Batched3DModel, Last Access: May 2018

[107] Unity Documentation, https://docs.unity3d.com/, Last Access: May 2018

# APPENDIX

## A. Camera locations and image overlap of photogrammetric blocks.

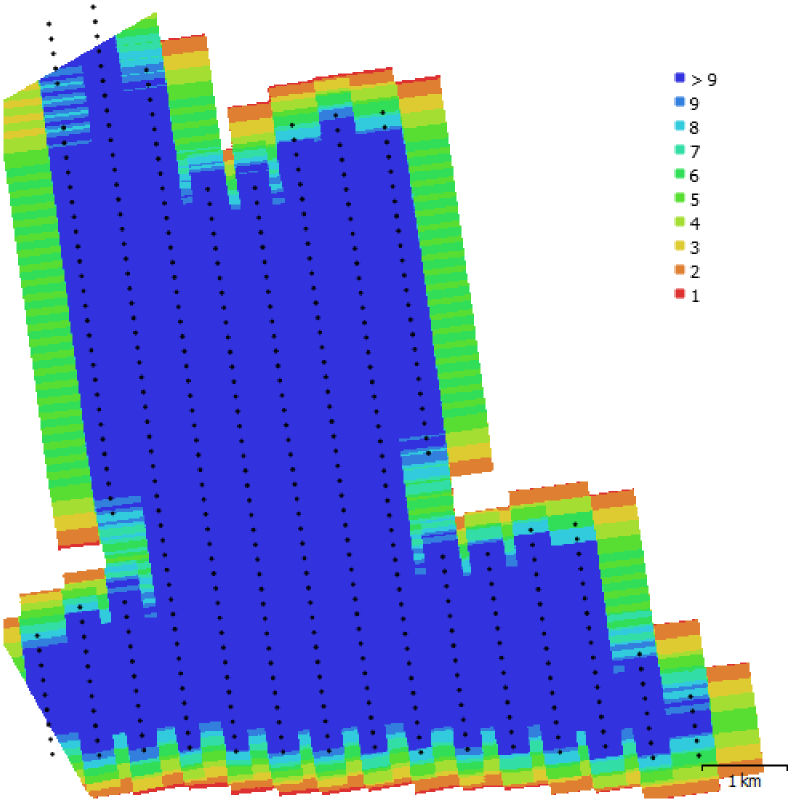


Figure A.1 Camera locations and image overlaps of block 1

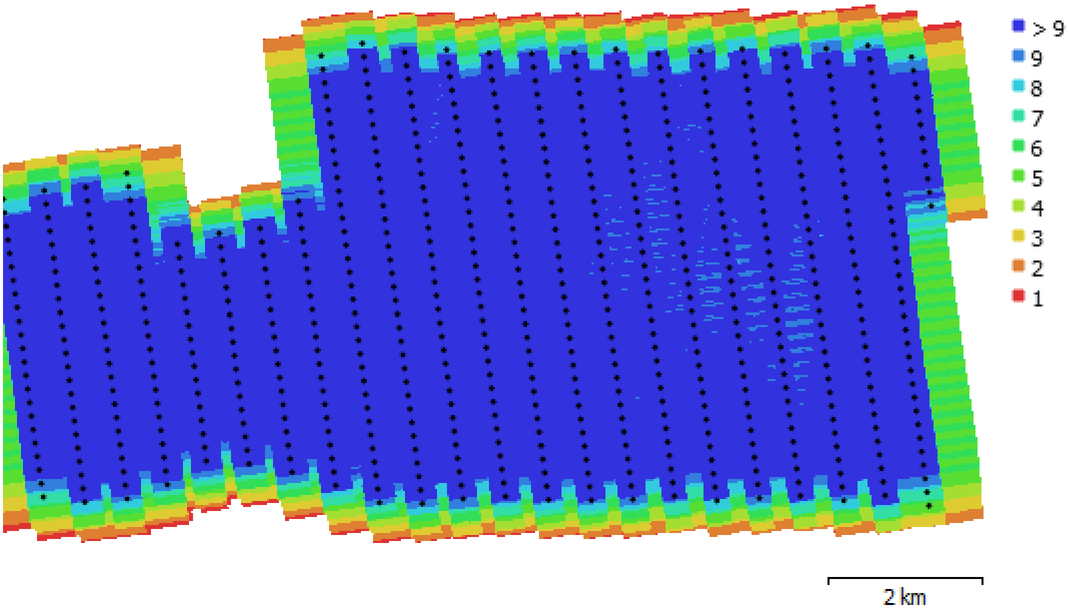


Figure A.2 Camera locations and image overlaps of block 2

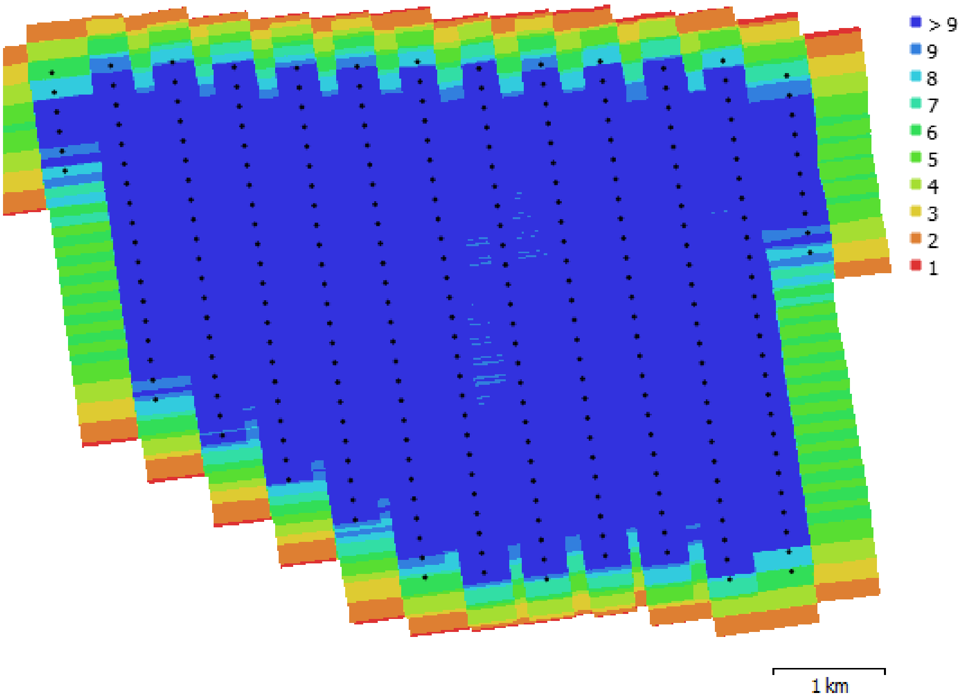


Figure A.3 Camera locations and image overlaps of block 3

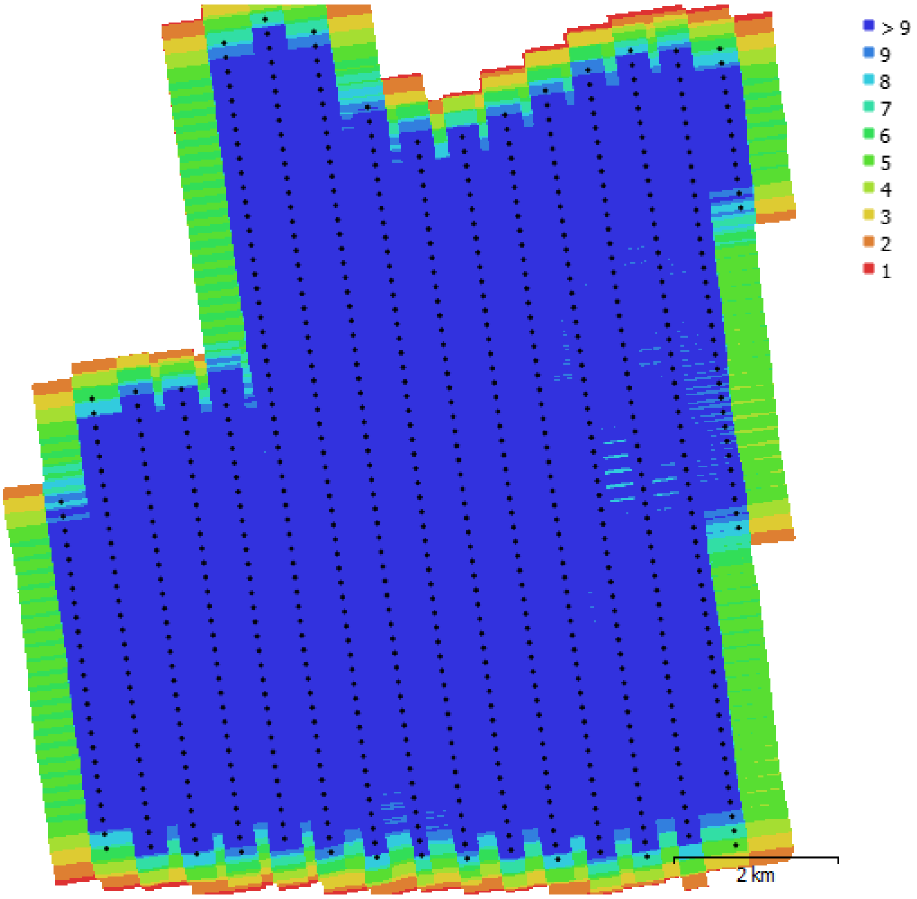


Figure A.4 Camera locations and image overlaps of block 4

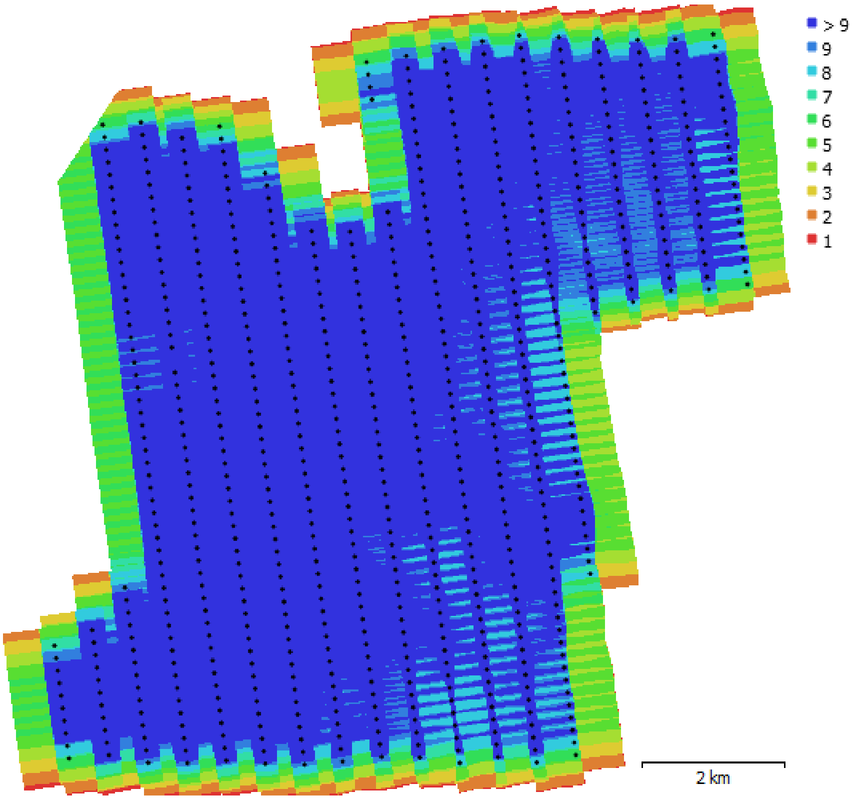


Figure A.5 Camera locations and image overlaps of block 5

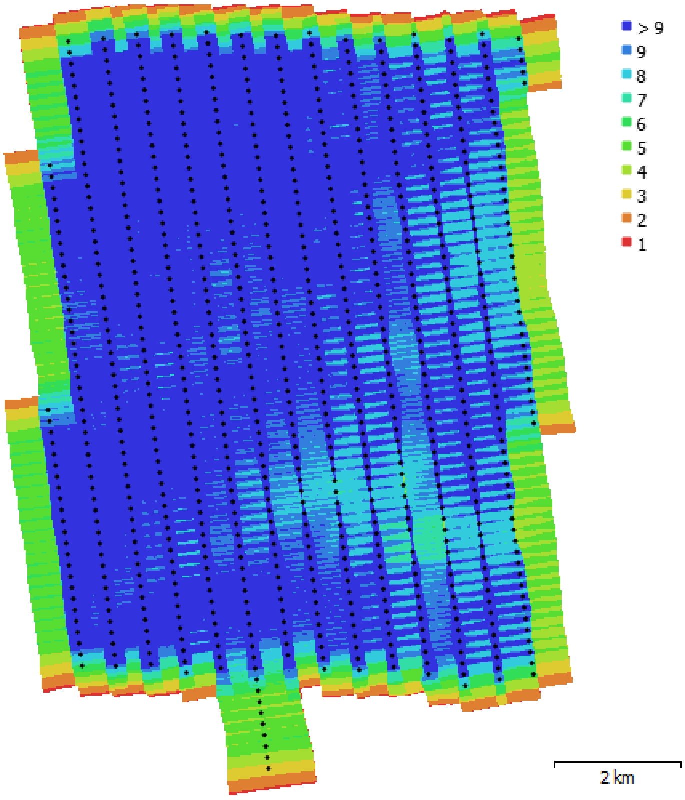


Figure A.6 Camera locations and image overlaps of block 6

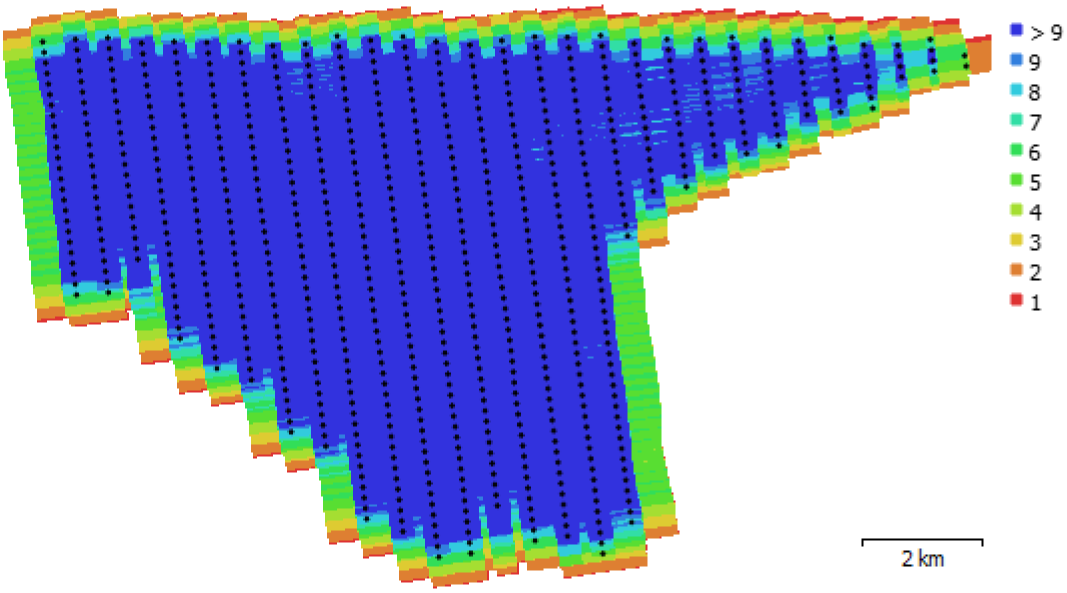


Figure A.7 Camera locations and image overlaps of block 7

## B.Developed JavaScript code for CesiumJS Virtual Globe Web Interface

```
// A demo of 3D City model of Çeşme
var viewer = new Cesium.Viewer('cesiumContainer', {
    animation : false,
    geocoder : false,
    timeline : false,
    fullscreenButton : false,
    homeButton : false,
    scene3DOnly : true,
    navigationHelpButton : true
});

// Set the initial camera view to look at Çeşme
var initialPosition = Cesium.Cartesian3.fromDegrees(26.285, 38.3217, 1000);
var initialOrientation = new Cesium.HeadingPitchRoll.fromDegrees(90.0, -30.0, 0.0);
viewer.scene.camera.setView({
    destination: initialPosition,
    orientation: initialOrientation,
    endTransform: Cesium.Matrix4.IDENTITY
});

// Cesium Navigation Widget
viewer.extend(Cesium.viewerCesiumNavigationMixin, {});

//Hacettepe Geomatics Credit
var credit1 = new Cesium.Credit({
    text : 'Hacettepe University Geomatics Engineering',
    imageUrl : 'Build/hacettepe.png',
    link : 'http://geomatik.hacettepe.edu.tr',
    showOnScreen: true,
});
viewer.scene.frameState.creditDisplay.addDefaultCredit(credit1);

//CesiumJS Credit
var credit2 = new Cesium.Credit({
    text : 'CesiumJS',
    imageUrl : 'Build/cesium.png',
    link : 'http://www.cesiumjs.org',
    showOnScreen: true,
});
viewer.scene.frameState.creditDisplay.addDefaultCredit(credit2);

var scene = viewer.scene;
var viewModelTileset;

function reset() {
    viewer.entities.removeAll();
    viewer.scene.primitives.removeAll();
    viewModelTileset = undefined;
}

// The viewModel tracks the state of our mini application.
var viewModel = {
    maximumScreenSpaceError : 8.0,
    geometricErrorScale : 1.0,
    maximumAttenuation : 0, // Equivalent to undefined
    baseResolution : 0, // Equivalent to undefined
    eyeDomeLightingStrength : 1.0,
    eyeDomeLightingRadius : 1.0,
    rightClickAction: 'annotate',
    middleClickAction: 'hide'
};

function tilesetToViewModel(tileset) {
    viewModelTileset = tileset;

    var pointCloudShading = tileset.pointCloudShading;
    viewModel.maximumScreenSpaceError = tileset.maximumScreenSpaceError;
    viewModel.geometricErrorScale = pointCloudShading.geometricErrorScale;
    viewModel.maximumAttenuation = pointCloudShading.maximumAttenuation ?
pointCloudShading.maximumAttenuation : 0;
    viewModel.baseResolution = pointCloudShading.baseResolution ? pointCloudShading.baseResolution : 0;
    viewModel.eyeDomeLightingStrength = pointCloudShading.eyeDomeLightingStrength;
    viewModel.eyeDomeLightingRadius = pointCloudShading.eyeDomeLightingRadius;
}

function loadtexturedlod2() {
    var tileset = new Cesium.Cesium3DTileset({ url: 'lod2tx' });
    viewer.scene.primitives.add(tileset);
    tileset.maximumScreenSpaceError = 8.0;
    tilesetToViewModel(tileset);
}

function loadlod2() {
    var tileset = new Cesium.Cesium3DTileset({ url: 'lod2' });
    viewer.scene.primitives.add(tileset);
    tileset.maximumScreenSpaceError = 8.0;
    tilesetToViewModel(tileset);
}

function loadlod1() {
    var tileset = new Cesium.Cesium3DTileset({ url: 'lod1' });
    viewer.scene.primitives.add(tileset);
    tileset.maximumScreenSpaceError = 8.0;
    tilesetToViewModel(tileset);
}

function loadlod0() {
    var tileset = new Cesium.Cesium3DTileset({ url: 'lod0' });
    viewer.scene.primitives.add(tileset);
    tileset.maximumScreenSpaceError = 8.0;
    tilesetToViewModel(tileset);
}

function loadpcloud() {
    var tileset = new Cesium.Cesium3DTileset({ url: 'pointcloud' });
    viewer.scene.primitives.add(tileset);
    tileset.maximumScreenSpaceError = 4.0; // For better performance, due to how this tileset treats
geometric error.
    tileset.pointCloudShading.maximumAttenuation = 8.0; // Don't allow points larger than 8 pixels.
    tileset.pointCloudShading.baseResolution = 0.05; // Assume an original capture resolution of 5
centimeters between neighboring points.
    tileset.pointCloudShading.geometricErrorScale = 1.0; // Applies to both geometric error and the base
resolution.
    tileset.pointCloudShading.attenuation = true;
    tileset.pointCloudShading.eyeDomeLighting = false;
    tilesetToViewModel(tileset);
}

function loadmesh() {
    var tileset = new Cesium.Cesium3DTileset({ url: 'mesh' });
    viewer.scene.primitives.add(tileset);
    tileset.maximumScreenSpaceError = 0.0;
    tilesetToViewModel(tileset);
}

loadtexturedlod2();

// Convert the viewModel members into knockout observables.
Cesium.knockout.track(viewModelTileset);

// Load Çeşme Digital Terrain Model - 1m Resolution - Quantized Mesh Tile
var cesiumTerrainProviderMeshes = new Cesium.CesiumTerrainProvider({
    url : 'qmesh',
    requestWaterMask : false,
    requestVertexNormals : false
});
viewer.terrainProvider = cesiumTerrainProviderMeshes;

//  Quantized Mesh Tile DTM Baselayer Menu
var quantizedmeshterrain = viewer.baseLayerPicker.viewModel;
var quantizedmeshterrainProvider = new Cesium.ProviderViewModel({
    name: 'Çeşme Quantized-Mesh 1M',
    iconUrl: 'Build/terrainicon.jpg',
    tooltip: 'Çeşme Quantized-Mesh 1M DTM',
    creationFunction: function () {
        return new Cesium.CesiumTerrainProvider({
            url: 'qmesh'
        });
    }
});
quantizedmeshterrain.terrainProviderViewModels.push(quantizedmeshterrainProvider);
quantizedmeshterrain.selectedTerrain = quantizedmeshterrainProvider;

//  Heightmap DTM Baselayer Menu
var heightmapterrain = viewer.baseLayerPicker.viewModel;
var heightmapterrainProvider = new Cesium.ProviderViewModel({
    name: 'Çeşme Heightmap 1M',
    iconUrl: 'Build/terrainicon.jpg',
    tooltip: 'Çeşme Heightmap 1M',
    creationFunction: function () {
        return new Cesium.CesiumTerrainProvider({
            url: 'heightmap'
        });
    }
});
heightmapterrain.terrainProviderViewModels.push(heightmapterrainProvider);

// HTML overlay for showing feature name on mouseover
var nameOverlay = document.createElement('div');
viewer.container.appendChild(nameOverlay);

// Information about the currently selected feature
var selected = {
    feature: undefined,
    originalColor: new Cesium.Color()
};

// Information about the currently highlighted feature
var highlighted = {
    feature: undefined,
    originalColor: new Cesium.Color()
};

// An entity object which will hold info about the currently selected feature for infobox display
var selectedEntity = new Cesium.Entity();

// Color a feature yellow on hover.
viewer.screenSpaceEventHandler.setInputAction(function onMouseMove(movement) {
    // If a feature was previously highlighted, undo the highlight
    if (Cesium.defined(highlighted.feature)) {
        highlighted.feature.color = highlighted.originalColor;
        highlighted.feature = undefined;
    }

    // Pick a new feature
    var pickedFeature = viewer.scene.pick(movement.endPosition);
    if (!Cesium.defined(pickedFeature)) {
        nameOverlay.style.display = 'none';
        return;
    }

 // Highlight the feature if it's not already selected.
    if (pickedFeature !== selected.feature) {
        highlighted.feature = pickedFeature;
        Cesium.Color.clone(pickedFeature.color, highlighted.originalColor);
        pickedFeature.color = Cesium.Color.LIME;
    }
}, Cesium.ScreenSpaceEventType.MOUSE_MOVE);

// Color a feature on selection and show metadata in the InfoBox.
var clickHandler =
viewer.screenSpaceEventHandler.getInputAction(Cesium.ScreenSpaceEventType.LEFT_CLICK);
viewer.screenSpaceEventHandler.setInputAction(function onLeftClick(movement) {
    // If a feature was previously selected, undo the highlight
    if (Cesium.defined(selected.feature)) {
        selected.feature.color = selected.originalColor;
        selected.feature = undefined;
    }

    // Pick a new feature
    var pickedFeature = viewer.scene.pick(movement.position);
    if (!Cesium.defined(pickedFeature)) {
        clickHandler(movement);
        return;
    }

    // Select the feature if it's not already selected
    if (selected.feature === pickedFeature) {
        return;
    }
    selected.feature = pickedFeature;

    // Save the selected feature's original color
    if (pickedFeature === highlighted.feature) {
        Cesium.Color.clone(highlighted.originalColor, selected.originalColor);
        highlighted.feature = undefined;
    } else {
        Cesium.Color.clone(pickedFeature.color, selected.originalColor);
    }
    // Highlight newly selected feature
    pickedFeature.color = Cesium.Color.LIME;

    // Set feature infobox description
    var featureName = pickedFeature.getProperty('buildingid');
    selectedEntity.name = featureName;
    selectedEntity.description = 'Loading <div class="cesium-infoBox-loading"></div>';
    viewer.selectedEntity = selectedEntity;
    selectedEntity.description = '<table class="cesium-infoBox-defaultTable"><tbody>' +
        '<tr><td>Building ID</th><td>' + pickedFeature.getProperty('buildingid') + '</td></tr>' +
        '<tr><td>Building Type</th><td>' + pickedFeature.getProperty('usage') + '</td></tr>' +
        '<tr><td>Number of Floors</th><td>' + pickedFeature.getProperty('floors') + '</td></tr>' +
        '<tr><td>Building Height</th><td>' + pickedFeature.getProperty('height') + '</th> meters</tr>' +
        '<tr><td>Ground Area</th><td>' + pickedFeature.getProperty('groundarea') + '</th> m²</tr>' +
        '<tr><td>Building Volume</th><td>' + pickedFeature.getProperty('volume') + '</th> m³</tr>' +
        '<tr><td>Latitude</th><td>' + pickedFeature.getProperty('latitude').toFixed(7) + '</th>°</tr>' +
        '<tr><td>Longitude</th><td>' + pickedFeature.getProperty('longitude').toFixed(7) + '</th>°</tr>' +
        '<tr><td>X Coordinate</th><td>' + pickedFeature.getProperty('xcoord').toFixed(2) + '</td></tr>' +
        '<tr><td>Y Coordinate</th><td>' + pickedFeature.getProperty('ycoord').toFixed(2) + '</td></tr>' +
        '<tr><td>Z Coordinate</th><td>' + pickedFeature.getProperty('zcoord').toFixed(2) + '</th> meters</tr>'
+
        '<tr><td>Coordinate Sys. </th><td>' + '</th>ITRF96 - 3°</th>' +
        '<tr><td>Street Maps</th><td>' + '<a href="http://data.mapchannels.com/dualmaps5/map.htm?lat=' +
pickedFeature.getProperty('latitude') + '&lng=' + pickedFeature.getProperty('longitude') + '&z=18&slat=' +
pickedFeature.getProperty('latitude') + '&slng=' + pickedFeature.getProperty('longitude') + '&sh=-
150.75&sp=-0.897&sz=1&gm=0&bm=2&panel=sb&mi=1&md=0" target="_blank">Show (Mid-Click)</a>'
+
        '</tbody></table>';
}, Cesium.ScreenSpaceEventType.LEFT_CLICK);

// Color buildings based on their height.
function colorByHeight() {
    viewModelTileset.style = new Cesium.Cesium3DTileStyle({
        color: {
            conditions: [
                ["${height} >= 30", "rgba(45, 0, 75, 1)"],
                ["${height} >= 20", "rgb(102, 71, 151)"],
                ["${height} >= 18", "rgb(170, 162, 204)"],
                ["${height} >= 16", "rgb(224, 226, 238)"],
                ["${height} >= 13", "rgb(252, 230, 200)"],
                ["${height} >= 10", "rgb(248, 176, 87)"],
                ["${height} >= 5", "rgb(198, 106, 11)"],
                ["true", "rgb(127, 59, 8)"]
            ]
        }
    });
}

// Color buildings by their total area.
function colorByArea() {
    viewModelTileset.style = new Cesium.Cesium3DTileStyle({
        color: "mix(color('red'), color('cyan'), min(${groundarea} / 150.0, 1.0))"
    });
}

// Transparent buildings
function transparent() {
    viewModelTileset.style = new Cesium.Cesium3DTileStyle({
        color: "rgba(255, 255, 255, 0.5)"
    });
}

// Color buildings by their latitude
function colorByLatitude() {
    viewModelTileset.style = new Cesium.Cesium3DTileStyle({
        color: {
            conditions: [
                ["${latitude} >= 38.330", "color('purple')"],
                ["${latitude} >= 38.326", "color('red')"],
                ["${latitude} >= 38.322", "color('orange')"],
                ["${latitude} >= 38.318", "color('yellow')"],
                ["${latitude} >= 38.314", "color('lime')"],
                ["${latitude} >= 38.310", "color('blue')"],
                ["true", "color('cyan')"]
            ]
        }
    });
}

// Color buildings by their longitude
function colorByLongitude() {
    viewModelTileset.style = new Cesium.Cesium3DTileStyle({
        color: {
            conditions: [
                ["${longitude} >= 26.314", "color('purple')"],
                ["${longitude} >= 26.311", "color('red')"],
                ["${longitude} >= 26.308", "color('orange')"],
                ["${longitude} >= 26.305", "color('yellow')"],
                ["${longitude} >= 26.302", "color('lime')"],
                ["${longitude} >= 26.299", "color('blue')"],
                ["true", "color('cyan')"]
            ]
        }
    });
}

// Color buildings by distance from a landmark.
function colorByDistance() {
    viewModelTileset.style = new Cesium.Cesium3DTileStyle({
        defines : {
            distance : "distance(vec2(${longitude}, ${latitude}), vec2(26.30353346148452,
38.32355096813952))"
        },
        color : {
            conditions : [
                ["${distance} < 0.0003", "color('white')"],
                ["true", "mix(color('red'), color('yellow'), ${distance} / 0.01)"]
                /*["${distance} >= 0.0085", "color('purple')"],
                ["${distance} >= 0.007", "color('red')"],
                ["${distance} >= 0.0055", "color('orange')"],
                ["${distance} >= 0.004", "color('yellow')"],
                ["${distance} >= 0.0025", "color('lime')"],
                ["${distance} >= 0.001", "color('blue')"],*/
                /*["true", "color('white')"]*/

            ]
        }
    });
}
```

```
// Color buildings with a '79' in their name.
function colorByNameRegex() {
    viewModelTileset.style = new Cesium.Cesium3DTileStyle({
      color : "(regExp('79').test(${buildingid})) ? color('red', 1) : color('white', 1)"
    });
}

// Show only buildings greater than 13 meters in height.
function hideByHeight() {
    viewModelTileset.style = new Cesium.Cesium3DTileStyle({
        show : "${height} > 13"
    });
}

// Shows only school buildings.
function showOnlySchools() {
    viewModelTileset.style = new Cesium.Cesium3DTileStyle({
        show : "regExp('School').test(${usage})"
    });
}

function createScreenshot() {
    viewer.render();
    var imageUri = viewer.canvas.toDataURL();
    var imageWin = window.open("");
    imageWin.document.write("<html><head>" +
            "<title>" + imageUri + "</title></head><body>" +
            '<img src="' + imageUri + '" width= "100%">' +
            "</body></html>");
    return imageWin;
}

Sandcastle.addToolbarMenu([{
    text : 'Choose a tileset...',
}, {
    text : 'Textured LoD2',
    onselect : function() {
     reset();
     loadtexturedlod2();
    }
}, {
    text : 'LoD2',
    onselect : function() {
     reset();
     loadlod2();
    }
}, {
    text : 'LoD1',
    onselect : function() {
     reset();
     loadlod1();
    }
}, {
    text : 'LoD0',
    onselect : function() {
     reset();
     loadlod0();
    }
}, {
    text : 'Point Cloud',
    onselect : function() {
     reset();
     loadpcloud();
    }
}, {
    text : 'Textured Mesh',
    onselect : function() {
     reset();
     loadmesh();
    }

}]);

// Çeşme Locations Toolbar
Sandcastle.addDefaultToolbarMenu([{
    text : 'Zoom to...',
}, {
    text : 'Çeşme Marina',
    onselect : function() {
        viewer.scene.camera.flyTo({
            destination : Cesium.Cartesian3.fromDegrees(26.285, 38.3217, 1000),
            orientation : Cesium.HeadingPitchRoll.fromDegrees(90.0, -30.0, 0.0),

        });
    }
}, {
    text : 'Çeşme Castle',
    onselect : function() {
        viewer.scene.camera.flyTo({
            destination : Cesium.Cartesian3.fromDegrees(26.2995, 38.3235, 200),
            orientation : Cesium.HeadingPitchRoll.fromDegrees(90.0, -30.0, 0.0),

        });
    }
}, {
    text : 'Çeşme Amphitheatre',
    onselect : function() {
        viewer.scene.camera.flyTo({
            destination : Cesium.Cartesian3.fromDegrees(26.308, 38.325, 200),
            orientation : Cesium.HeadingPitchRoll.fromDegrees(170.0, -35.0, 0.0),

        });
    }
}, {
    text : 'Ildırı',
    onselect : function() {
        viewer.scene.camera.flyTo({
            destination : Cesium.Cartesian3.fromDegrees(26.472, 38.377, 400),
            orientation : Cesium.HeadingPitchRoll.fromDegrees(90.0, -30.0, 0.0),

        });
    }
}, {
    text : 'Alaçatı',
    onselect : function() {
        viewer.scene.camera.flyTo({
            destination : Cesium.Cartesian3.fromDegrees(26.376, 38.285, 500),
            orientation : Cesium.HeadingPitchRoll.fromDegrees(0.0, -90.0, 0.0),
        });
    }
}, {
    text : 'Windsurf School',
    onselect : function() {
        viewer.scene.camera.flyTo({
            destination : Cesium.Cartesian3.fromDegrees(26.400, 38.255, 1300),
            orientation : Cesium.HeadingPitchRoll.fromDegrees(260.0, -50.0, 0.0),

        });
    }
}, {
    text : 'Ilıca Beach',
    onselect : function() {
        viewer.scene.camera.flyTo({
            destination : Cesium.Cartesian3.fromDegrees(26.370, 38.317, 600),
            orientation : Cesium.HeadingPitchRoll.fromDegrees(170.0, -30.0, 0.0),

        });
    }
}, {
    text : 'Goldensand Beach',
    onselect : function() {
        viewer.scene.camera.flyTo({
            destination : Cesium.Cartesian3.fromDegrees(26.258, 38.261, 550),
            orientation : Cesium.HeadingPitchRoll.fromDegrees(-5.0, -35.0, 0.0),

        });
    }
}], 'toolbar');

Sandcastle.addToolbarMenu([{
     text : '3D Queries',
}, {
    text : 'Color By Height',
    onselect : function() {
        colorByHeight();
    }
}, {
    text : 'Color By Area',
    onselect : function() {
        colorByArea();
    }
}, {
    text : 'Color By Distance',
    onselect : function() {
        colorByDistance();
    }
}, {
    text : 'Color By Latitude',
    onselect : function() {
        colorByLatitude();
    }
}, {
    text : 'Color By Longitude',
    onselect : function() {
        colorByLongitude();
    }
}, {
    text : 'Color By Name Regex',
    onselect : function() {
        colorByNameRegex();
    }
}, {
    text : 'Hide By Height',
    onselect : function() {
        hideByHeight();
    }
}, {
    text : 'Hide By Type',
    onselect : function() {
        showOnlySchools();
    }
}, {
    text : 'Transparent Buildings',
    onselect : function() {
        transparent();
    }
}, {
    text : 'Take Screenshot',
    onselect : function() {
        createScreenshot();
    }

}]);

Sandcastle.addToggleButton('Shadows', viewer.shadows, function(checked) {
    viewer.shadows = checked;
});

Sandcastle.addToggleButton('Lighting', viewer.scene.globe.enableLighting, function(checked) {
    viewer.scene.globe.enableLighting = checked;
});

Sandcastle.addToggleButton('Fog', viewer.scene.fog.enabled, function(checked) {
    viewer.scene.fog.enabled = checked;
});

Cesium.knockout.track(viewModel);

var toolbar = document.getElementById('toolbar');
Cesium.knockout.applyBindings(viewModel, toolbar);

var annotations = scene.primitives.add(new Cesium.LabelCollection());

var handler = new Cesium.ScreenSpaceEventHandler(viewer.canvas);

handler.setInputAction(function(movement) {
    var feature = viewer.scene.pick(movement.position);
    if (!Cesium.defined(feature)) {
        return;
    }

    var action = viewModel.rightClickAction;
    if (action === 'annotate') {
        annotate(movement, feature);
    }
}, Cesium.ScreenSpaceEventType.RIGHT_CLICK);

handler.setInputAction(function(movement) {
    var feature = viewer.scene.pick(movement.position);
    if (!Cesium.defined(feature)) {
        return;
    }

    var action = viewModel.middleClickAction;
    if (action === 'hide') {
        feature.show = false;
    }
}, Cesium.ScreenSpaceEventType.MIDDLE_CLICK);

function annotate(movement, feature) {
    if (scene.pickPositionSupported) {
        var cartesian = scene.pickPosition(movement.position);
        if(Cesium.defined(cartesian)){
            var cartographic = Cesium.Cartographic.fromCartesian(cartesian);
            var height = cartographic.height.toFixed(2) + ' m';

            annotations.add({
                position : cartesian,
                text : height,
                showBackground : true,
                font : '14px monospace',
                horizontalOrigin : Cesium.HorizontalOrigin.LEFT,
                verticalOrigin : Cesium.VerticalOrigin.BOTTOM,
                disableDepthTestDistance : Number.POSITIVE_INFINITY
            });
        }
    }
}
```

# CURRICULUM VITAE

## Credentials

Name, Surname : Mehmet BÜYÜKDEMİRCİOĞLU

Place of Birth : Istanbul, Turkey

Marital Status : Single

E-mail : mbuyukdemircioglu@hacettepe.edu.tr

Adress : Ankara, Turkey

## Education

High School : Ar-El College

Bsc. : Selçuk University, Dept. of Geomatics Engineering

Msc. : -

PhD. : -

## Foreign Languages

English (Good)

## Work Experience

2012 - ... : Uncuoğlu Engineering and Consultancy

## Areas of Experiences

Photogrammery, Geographical Information Systems, Web Programming

## Projects and Budgets

Gaziantep Smart City 2023 Project

## Publications

-

## Oral and Poster Presentations

-

## HACETTEPE UNIVERSITY
## GRADUATE SCHOOL OF SCIENCE AND ENGINEERING
## THESIS/DISSERTATION ORIGINALITY REPORT

**HACETTEPE UNIVERSITY**
**GRADUATE SCHOOL OF SCIENCE AND ENGINEERING**
**TO THE DEPARTMENT OF GEOMATICS ENGINEERING**

Date: 07/06/2018

Thesis Title : IMPLEMENTATION AND WEB-BASED VISUALIZATION OF 3D CITY MODELS

According to the originality report obtained by my thesis advisor by using the *Turnitin* plagiarism detection software and by applying the filtering options stated below on 07/06/2018 for the total of 80 pages including the a) Title Page, b) Introduction, c) Main Chapters, d) Conclusion sections of my thesis entitled as above, the similarity index of my thesis is 8%.

Filtering options applied:

1. Bibliography/Works Cited excluded
2. Quotes excluded / ~~included~~
3. Match size up to 5 words excluded

I declare that I have carefully read Hacettepe University Graduate School of Sciene and Engineering Guidelines for Obtaining and Using Thesis Originality Reports; that according to the maximum similarity index values specified in the Guidelines, my thesis does not include any form of plagiarism; that in any future detection of possible infringement of the regulations I accept all legal responsibility; and that all the information I have provided is correct to the best of my knowledge.

I respectfully submit this for approval.

07/06/2018

**Name Surname:** Mehmet BÜYÜKDEMİRCİOĞLU
**Student No:** N15223142
**Department:** Geomatics Engineering
**Program:** Geomatics Engineering
**Status:** ☒ Masters ☐ Ph.D. ☐ Integrated Ph.D.

**ADVISOR APPROVAL**

APPROVED.

Assoc.Prof.Dr. Sultan KOCAMAN