

POLITECNICO DI MILANO
Scuola di Ingegneria dell'Informazione



POLO TERRITORIALE DI COMO
Master of Science in Computer Engineering

DESIGN AND IMPLEMENTATION OF A FRAMEWORK TO GENERATE SYNTHETIC 3D URBAN RECONSTRUCTION DATASETS

Advisor: Prof. Sara COMAI

Co-Advisor: Prof. Matteo MATTEUCCI

Master Thesis by:

Berke Cagkan TOPTAS, ID 814483

Academic Year 2014/2015

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Prof. Sara Comai and Prof. Matteo Matteucci for introducing me to the topic as well for the support on the way. I am really grateful for their patience and tolerance throughout the year. I have to admit that, they provided almost everything in theory and all I had to do was connecting the pieces.

My second thanks goes to OpenStreetMap community for making this thesis real. I cannot even imagine how I would finish the work without these valuable contributors. Besides, I thank to community of Stack Exchange, especially Stack Overflow group for helping me to solve numerous coding issues during my sleepless nights.

Last but not least, I would like to thank my family for their endless support from first moment to last during my master education in Italy.

SOMMARIO

Questa tesi propone un framework per generare dataset sintetici per algoritmi di ricostruzione di ambienti urbani 3D attraverso la raccolta di dati generati virtualmente in ambienti urbani 3D.

La ricostruzione 3D è un argomento ampiamente studiato nell'ambito della computer vision e i metodi impiegati stanno aumentando rapidamente. Nonostante la sua popolarità mancano dataset di riferimento nel campo e questo rende difficile confrontare le prestazioni di metodi diversi.

Attualmente, le prestazioni dei metodi di ricostruzione vengono confrontati utilizzando dataset piuttosto piccoli. Questi però non sono adeguati per l'analisi delle prestazioni di metodi di ricostruzione di aree più grandi, come gli ambienti urbani. Esiste quindi la necessità di avere dataset per questi tipi di metodi.

La raccolta dei dati di test dal mondo reale è costoso e difficile. Per questo motivo, viene proposto un framework per la generazione di ambienti urbani che consente agli utenti di effettuare varie modifiche del modello e per la raccolta dei dataset attraverso una telecamera virtuale e un laser scanner.

In questa tesi, inizialmente si discute la necessità di un nuovo generatore di ambienti urbani al posto dei generatori esistenti. Successivamente si spiega come tale generatore è stato implementato e come i dati di test vengono raccolti dalla scena. Infine, si discutono la qualità dei risultati ed i possibili lavori futuri.

ABSTRACT

This thesis proposes a framework to generate synthetic datasets for 3D urban reconstruction algorithms by harvesting data from virtually generated 3D urban environments.

3D reconstruction is a popular topic in computer vision and state-of-the-art methods are growing rapidly. Despite its popularity, there is a lack of benchmark dataset in the field and this makes it difficult to compare the performance of different methods. Currently, performance of the reconstruction methods are compared by using small object datasets. However, there exist methods concerning the reconstruction of larger areas, such as urban environments. Therefore, small datasets are unable to provide a sufficient analysis of performance and there is a need for convenient datasets. Collecting test data from the real-world has been expensive and cumbersome. For this reason, a framework is implemented generating synthetic urban environments allowing users to make various edits on the model and collecting data by virtual camera set and a laser scanner.

In this thesis, first we discuss the necessity of a new urban generator instead of using existing generators. Then, we explain how our urban generator is implemented and how the test data is collected from the scene. Finally, we discuss the quality of the results and future work for improvement.

Table of Contents

Chapter 1 : INTRODUCTION	1
1.1. MOTIVATION.....	1
1.2. GOALS.....	1
1.3. OVERVIEW.....	2
1.4. OUTLINE OF THE THESIS.....	3
Chapter 2 : STATE OF THE ART	5
2.1. OpenStreetMap	5
2.1.1. Introduction	5
2.1.2. OSM Xml Data	5
2.2. EXISTING 3D URBAN GENERATORS.....	7
2.2.1. OSM 3D	7
2.2.2. OSM 2 World.....	8
2.2.3. Esri City Engine.....	11
2.3. RENDERING ENVIRONMENT	12
Chapter 3 : URBAN GENERATOR	15
3.1. TERRAIN	16
3.1.1. Geographical Overview.....	16
3.1.2. Mesh Generation	18
3.1.3. Texture Generation.....	22
3.1.4. Capabilities.....	27
3.2. BUILDING.....	28
3.2.1. OSM Representation.....	29
3.2.2. Preprocessing.....	29
3.2.3. Rendering.....	32
3.3. BARRIER.....	34
3.3.1. OSM Representation.....	35
3.3.2. Preprocessing.....	35
3.4. HIGHWAY	38
3.4.1. OSM Representation.....	38
3.4.2. Preprocessing.....	39
3.4.3. Rendering.....	51

3.5. 3D OBJECTS	52
3.5.1. OSM Representation	52
3.5.2. Preprocessing	53
Chapter 4 : DATA COLLECTOR	55
4.1. CONTROLLER	56
4.1.1. Camera Van	56
4.1.2. Trekker	57
4.1.3. LOG File Generation	58
4.2. VIRTUAL CAMERA	59
4.3. VIRTUAL LASER SCANNER (LiDAR)	59
4.3.1. Point Cloud Data (PCD)	60
4.3.2. PCD Generation	61
Chapter 5 : USER INTERFACE	63
5.1. LOAD/SAVE MENU	64
5.1.1. Create New Project	64
5.1.2. Load/Save Project	65
5.2. DEFAULT SETTINGS MENU	66
5.2.1. Default Building Settings	66
5.2.2. Default Barrier Settings	68
5.2.3. Default Highway Settings	69
5.2.4. Skybox Settings	70
5.3. EDIT MENU	71
5.3.1. Building Edit	71
5.3.2. Barrier Edit	73
5.3.3. Highway Edit	74
5.3.4. 3D Object Edit	75
5.4. ADD OBJECT MENU	76
5.5. DATA COLLECTOR MENU	78
5.5.1. Camera Settings	78
5.5.2. Laser Settings	80
Chapter 6 : RESULTS AND EVALUATION	83
6.1. SAMPLE OUTPUT GENERATION	83
6.2. QUALITY EVALUATION	88

6.3 PERFORMANCE EVALUATION	90
6.3.1. Urban Generator Performance	90
6.3.2. Data Collector Performance.....	93
Chapter 7 : CONCLUSIONS AND FUTURE WORK	95
7.1. CONCLUSIONS.....	95
7.2. FUTURE WORK	95
BIBLIOGRAPHY.....	97
APPENDIX A: OSM XML FILE	99
APPENDIX B: CLASS DIAGRAMS.....	111

List of Figures

Figure 1.1 - Framework Component Diagram	2
Figure 2.1 - OSM tagging illustration	6
Figure 2.2- Sample OpenStreetMap Xml File.....	6
Figure 2.3- Map View of Heidelberg (DE) and Munich (DE) in OSM-3D	7
Figure 2.4- OSM-3D Processing Stages [12].....	8
Figure 2.5 - Osm2world different output formats (a) 3D .obj file (b) rendered with POVRay (c) rendered and converted to .png file.....	9
Figure 2.6 - OSM2World Processing Stages [14].....	10
Figure 2.7- CityEngine Processing Stages [Source: Esri, 2015]	11
Figure 2.8- Esri City Engine Modelling Sample.....	11
Figure 2.9- Tools Market Distribution [18]	13
Figure 2.10 - Registered Unity Developers [17].....	13
Figure 3.1 - Urban Generator Component Diagram	15
Figure 3.2- Terrain Object	16
Figure 3.3- (a) WGS 84 Datum (b) Web Mercator projection.....	17
Figure 3.4- Bounds Tag OSM xml	19
Figure 3.5- Terrain Bounds.....	20
Figure 3.6 – Terrain Mesh (Brunate, Como)	21
Figure 3.7 – Tile at Zoom Level 0	22
Figure 3.8 - Tiles at Zoom Level 1.....	23
Figure 3.9 - Sample 256x256 Tiles Zoom: 18, x: 137683, y: 93456 (a) Bing Aerial (b) OpenStreetMap (c) Bing Street (d) MapQuest	25
Figure 3.10 - Uncropped Tiles and Terrain Mesh Pack	26
Figure 3.11 - Textured Terrain Surface (Brunate, Como).....	27
Figure 3.12 - Building objects.....	28
Figure 3.13- Building represented with <way> tag.....	29
Figure 3.14- Building represented with <relation> tag.....	29
Figure 3.15- Building with (a) handled and (b) unhandled Building Height.....	30
Figure 3.16- OSM roof types [Source: OSM wiki]	31
Figure.3.17 - Triangulated Concave Roof Polygon.....	32
Figure 3.18 - Building Facade Mesh	32
Figure 3.19- Building way orientations. (a) Counter-clockwise (b) clockwise	33
Figure.3.20- Sample Building Material Textures (a) Color Texture (b) Normal map (c) Specular Map.....	34
Figure 3.21 - Barrier Object.....	34
Figure 3.22 - Barrier OSM Xml sample	35
Figure 3.23 - Fence type barrier, Chain-link.....	36
Figure 3.24 - Wall type Barriers: (a) retaining wall (b) wall (c) city wall	36
Figure 3.25 - Wall type barrier mesh generation.....	37
Figure 3.26 - Highway object	38
Figure 3.27 - Highway OSM xml sample	38
Figure 3.28 – (a) Initial Points (b) Way with a width (c) Problem when angle changes (d) Angle changing Solution	40

Figure 3.29 - Highway drawn with handled and unhandled slope changes	41
Figure 3.30 - Intersection Processing Stages	42
Figure 3.31 - Road Half Vectors and Angles.....	43
Figure 3.32 - Sample steps of intersection processing	44
Figure 3.33 - Sample steps of truth table processing	44
Figure 3.34- (a) After intersection with holes (b) Intersection holes filled.....	45
Figure 3.35 - Side walk representation	45
Figure 3.36- Road with Sidewalk on both sides	46
Figure 3.37 - (a) & (c): Unhandled sidewalk intersections, (b) & (d): corrected versions	46
Figure 3.38 – (a) Road with Height Map visibility Problem (b) Draped Highway with pin Points	47
Figure 3.39 - Road Draping	48
Figure 3.40 - Adding vertices for 2 different cases	49
Figure 3.41 - (a) Dissymmetry Problem while draping, (b) Solution.....	50
Figure 3.42- Highways that remains under terrain.....	50
Figure 3.43- Highway Mesh	51
Figure 3.44 - Road Texture Samples	51
Figure 3.45 - 3D Objects.....	52
Figure 3.46 - Samples of 3D object tags.....	52
Figure 3.47 - Highway Mesh. Original Vertices (red) and generated vertices (green)	53
Figure 4.1 - (a) Google Trekker (b) Google Camera Car [28].....	55
Figure 4.2 - Data Collector Component Diagram.....	55
Figure 4.3 - Camera Van (a) mesh and colliders (b) texturized mesh.....	56
Figure 4.4 - Trekker (From Unity Standard Package).....	57
Figure 4.5 - Sample Controller Log File	58
Figure 4.6 - LiDAR Ray casting.....	61
Figure 5.1- User Interface Component Diagram.....	63
Figure 5.2- Load/Save Menu	64
Figure 5.3 - File Browser Dialog (a) Select File Mode (b) Save File Mode.....	65
Figure 5.4 - Default Settings Menu	66
Figure 5.5 - Default Building Settings.....	67
Figure 5.6 - Default Barrier Settings.....	68
Figure 5.7 - Default Highway Settings.....	69
Figure 5.8 - Default Skybox Settings	70
Figure 5.9 - Different day times in the scene.....	70
Figure 5.10 - Edit Building Menu.....	71
Figure.5.11 - Edit Facade Texture Menu.....	72
Figure 5.12 - Barrier Edit Menu.....	73
Figure 5.13 - Highway Edit Menu.....	74
Figure 5.14 - 3D Object Edit Menu.....	75
Figure 5.15 - Transform Gizmos.....	75
Figure 5.16 - Add Object Menu.....	76
Figure 5.17 - Environment Object List Menu.....	77
Figure.5.18 - Data Collector Menu.....	78
Figure 5.19 - Camera Settings Menu.....	79

Figure 5.20 - Laser Scanner Settings Menu.....	80
Figure 6.1 – Sample Recording Path	83
Figure 6.2 - Sample Log File	84
Figure 6.3 - Sample Output Frames, Front Camera (Cam 1).....	85
Figure 6.4 - Sample Output Frames, Left Camera (Cam 2)	86
Figure 6.5 - Sample Output Frames, Right Camera (Cam 3)	87
Figure 6.6 - Sample PCD File (Frame 1).....	88
Figure 6.7- “Via Provinciale per Lecco”, (a) Generated Scene (b) Original, Google Street View	89
Figure 6.8 - Test Scenes (a) Pannilani (b) Brunate (c) Como City Center	90

List of Tables

- Table 6.1 – Test Scene Parameters 90
- Table 6.2 - Terrain Rendering Times 91
- Table 6.3- OSM File Parsing Times 91
- Table 6.4 - Building Rendering Times..... 92
- Table 6.5 - Highway Rendering Times..... 92
- Table 6.6 - Barrier + 3D Object Rendering Times..... 92
- Table 6.7 - Total Render Time and FPS values 93
- Table 6.8 – Test Cases for Data Controller..... 93
- Table 6.9 - Video Frame Generation Times 94
- Table 6.10 – PCD File Generation Times 94

Chapter 1 : INTRODUCTION

1.1. MOTIVATION

In computer vision, 3D reconstruction is the process of capturing the shape and appearance of real objects using a set of 2D images or point cloud data. Over the past years, several 3D reconstruction algorithms have been published and the popularity in the field is increasing rapidly. Unfortunately, the lack of benchmark datasets makes it difficult to compare the performance of these algorithms and to therefore focus research on the most needed areas of development.

Currently, state of the art systems compare their performances by using a small object dataset [1]. However, some of these systems concern the reconstruction of a large scene, such as an urban environment [2] [3]. In those cases, the previous dataset cannot give a significant analysis of the performances. Instead, a new dataset of urban scenes taken from different perspectives would be very useful.

One way to create an urban test dataset is equipping a car with a camera set, laser scanner and a GPS localization system, then capturing the data by driving around the city [4]. However, collecting the test data using this method is expensive and cumbersome. For this reason, we propose a framework which creates synthetic test datasets from custom 3D city models by virtually navigating the environment.

1.2. GOALS

This thesis aims at developing a framework to generate flexible synthetic datasets for urban reconstruction algorithms. In order to achieve that, the framework should be able to perform the following operations:

- Create automatically a 3D urban model
- Texturize the urban model
- Support various material types with proper lightings
- Generate video stream data from virtual camera set
- Generate point cloud data from virtual laser scanner set

With the framework, we aim to generate urban datasets easily with no cost. Automated urban generation saves users from equipment cost like cameras and laser scanners. Besides the cost, we aim to reduce the amount of time spent during data generation. It can take hours or

1.3. OVERVIEW

even days to prepare equipment and collect data from the real world while it takes several minutes with our framework.

Our second aim is to give users full control over the test scenes. Real-world test data comes as it is and it is not possible to make any edit on the scene. However with a synthetic scene, any desired object can be added/removed or lighting conditions can be changed. With this property, we aim to allow users to generate their unique scenes to test specific cases for their reconstruction methods.

1.3. OVERVIEW

Our framework consists of three main components, which are “Urban Generator”, “Data Collector” and “User Interface” (Figure 1.1). The Urban Generator creates the 3D model using Urban Data and Render Settings coming from user interface. The Urban generator parses and processes the urban data and outputs the 3D scene, which consists of buildings, highways, trees etc.

Once the scene is generated, users can customize the generated world with the user interface. Users have full control over the scene. From changing building heights and textures to selecting day time and adding any desired objects (e.g., cars, sculptures, trees), any edits can be made with the user interface.

When the scene is ready, users can activate the data collector component and produce video streams and point cloud data (PCD). Our framework shows 2 options for users as a data collector, which are a Camera Van or a Trekker. By controlling them over the city, scene data can be collected with little effort.

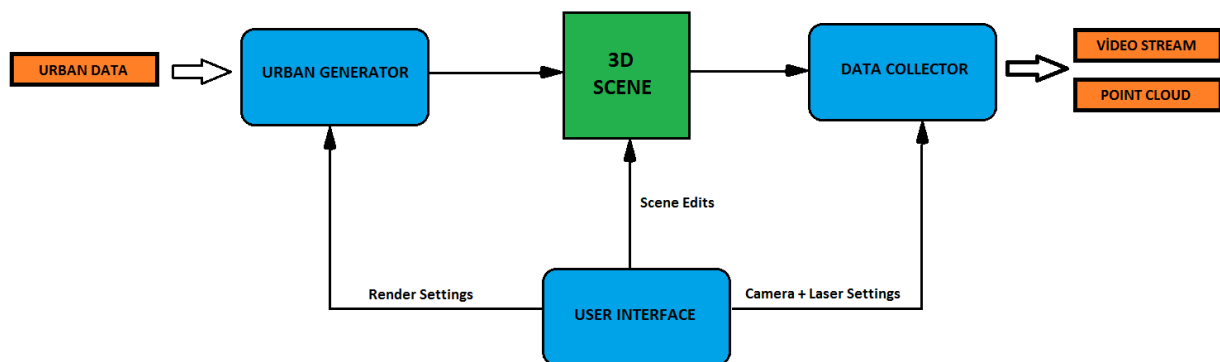


Figure 1.1 - Framework Component Diagram

1.4. OUTLINE OF THE THESIS

The thesis is organized as follows:

- In Chapter 2, we first describe the urban data set (OpenStreetMap) that is used both by our framework and the state of the art 3D urban generators. Then we present each of the existing 3D urban generators and briefly discuss why they are inadequate for our framework. Finally, we present the rendering tool (Unity) we use in our framework.
- In Chapter 3, we present how the “Urban Generator” component of the framework works. We explain how each component of the urban model is generated including pre-processing and rendering steps.
- In Chapter 4, we present how the “Data Collector” component of the framework works. We explain how camera images and the laser data are generated.
- In Chapter 5, we present our “User Interface” component of the framework. We explain each menu available in the framework and discuss the capabilities of the users.
- In Chapter 6, we present the results of the framework by illustrating a sample output generation.
- In Chapter 7, we summarize the whole work, discussing the obtained results. Then, we indicate the possible improvements for each module of the framework.
- In Appendix A, we explain the process of generating an input file to our framework from the OpenStreetMap webpage and we provide a sample input file.
- In Appendix B, we provide class diagrams of the framework for each of our three component.

1.4. OUTLINE OF THE THESIS

Chapter 2 : STATE OF THE ART

2.1. OpenStreetMap

2.1.1. Introduction

OpenStreetMap is a free, editable map of the whole world that is being built by volunteers largely from scratch and released with an open-content license. With currently more than 2,000,000 members [5], the community bears an enormous potential of "*humans acting as remote sensors*" [6].

By the very nature of the wiki-style process, there is no guarantee of accuracy of any kind of data. The essence of a wiki-style process is that all users have a stake in having accurate data. If one person puts in inaccurate data, maliciously or accidentally, the other 99.9% of people can check it, fix it, or get rid of it. The vast majority of good-intentioned participants can automatically correct for the few bad apples [7].

The OpenStreetMap License allows free access to map images and all of underlying map data. The project aims to promote new and interesting uses of this data.

2.1.2. OSM Xml Data

Contribution to OpenStreetMap is kept simple: Users provide **nodes** that are geo-referenced points with longitude and latitude information in UTM projection. For defining line geometries, several nodes can be combined to **ways**. A *closed way* represents a polygon (e.g., a building or an area), while a *non-closed way* represents a line array (e.g., a street or a wall). For defining complex relationships or complex polygons with holes (e.g., building with holes), users can create **relations**.

Beside geometry indicators, it is possible to **tag** OSM features, using key-value pair structures. Thereby, the key describes some kind of information or information domain and the value refines this information. What information is added, is up to the contributors. They can add any kind of information as well as an arbitrary amount of information. However, there are standardized tags which should be used for common map features such as streets or buildings. A list of all tags that are standardized are listed in [8]. Figure 2.1 depicts a map with common OSM features, as well as the corresponding OSM key-value pairs [9].

2.1. OpenStreetMap



Figure 2.1 - OSM tagging illustration

Collected information from users can be accessed from a single xml file named “Planet.osm” that contains all the nodes, ways and relation data. A new version is released every week. It's a big file (XML variant over 576.6GB uncompressed, 42GB bz2 compressed and 28.8GB PBF at 2015/05/19) [10].

Users can also request smaller data by querying the bounding box of the desired area. As a result, OpenStreetMap server generates an xml file similar to Figure 2.2. This file is one of the inputs for our Framework.

```

<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.0.2">
  <bounds minlat="54.0889580" minlon="12.2487570" maxlat="54.0913900" maxlon="12.2524800"/>
  <node id="298884269" lat="54.0901746" lon="12.2482632" user="SvenHRO" uid="46882" visible="true" version="1" changeset="676636" timestamp="2008-09-21T21:37:45Z"/>
  <node id="261728686" lat="54.0906309" lon="12.2441924" user="PikoWinter" uid="36744" visible="true" version="1" changeset="323878" timestamp="2008-05-03T13:39:23Z"/>
  <node id="1831881213" version="1" changeset="12370172" lat="54.0900666" lon="12.2539381" user="lafkor" uid="75625" visible="true" timestamp="2012-07-20T09:43:19Z">
    <tag k="name" v="Neu Broderstorf"/>
    <tag k="traffic_sign" v="city_limit"/>
  </node>
  ...
  <node id="298884272" lat="54.0901447" lon="12.2516513" user="SvenHRO" uid="46882" visible="true" version="1" changeset="676636" timestamp="2008-09-21T21:37:45Z"/>
  <way id="26659127" user="Masch" uid="55988" visible="true" version="5" changeset="4142606" timestamp="2010-03-16T11:47:08Z">
    <nd ref="292403538"/>
    <nd ref="298884289"/>
    ...
    <nd ref="261728686"/>
    <tag k="highway" v="unclassified"/>
    <tag k="name" v="Pastower Straße"/>
  </way>
  <relation id="56688" user="kmvar" uid="56190" visible="true" version="28" changeset="6947637" timestamp="2011-01-12T14:23:49Z">
    <member type="node" ref="294942404" role=""/>
    ...
    <member type="node" ref="364933006" role=""/>
    <member type="way" ref="4579143" role=""/>
    ...
    <member type="node" ref="249673494" role=""/>
    <tag k="name" v="Küstenbus Linie 123"/>
    <tag k="network" v="VWV"/>
    <tag k="operator" v="Regionalverkehr Küste"/>
    <tag k="ref" v="123"/>
    <tag k="route" v="bus"/>
    <tag k="type" v="route"/>
  </relation>
  ...
</osm>

```

Figure 2.2- Sample OpenStreetMap Xml File

2.2. EXISTING 3D URBAN GENERATORS

Before a decision was made to implement a new urban generator, existing projects were examined. Although there are several projects, evaluation was made in 3 main projects:

- OSM 3D
- OSM2World
- Esri City Engine

2.2.1. OSM 3D

OSM-3D is a research project carried out at *the GIScience Research Group, University of Heidelberg* [11]. It investigates how Volunteered Geographic Information and freely available data sets can be incorporated in a 3D Spatial Data Infrastructure on a global scale.

OSM-3D use OpenStreetMap xml file for generating 3D urban model. OpenStreetMap has a very active and dynamic community which can adapt to new circumstances and requirements very quickly. As more details such as absolute height and color of building and facade materials are provided by users, they are aiming to model complete 3D city models from OSM. Figure 2.3 shows some examples of screenshots from OSM-3D.

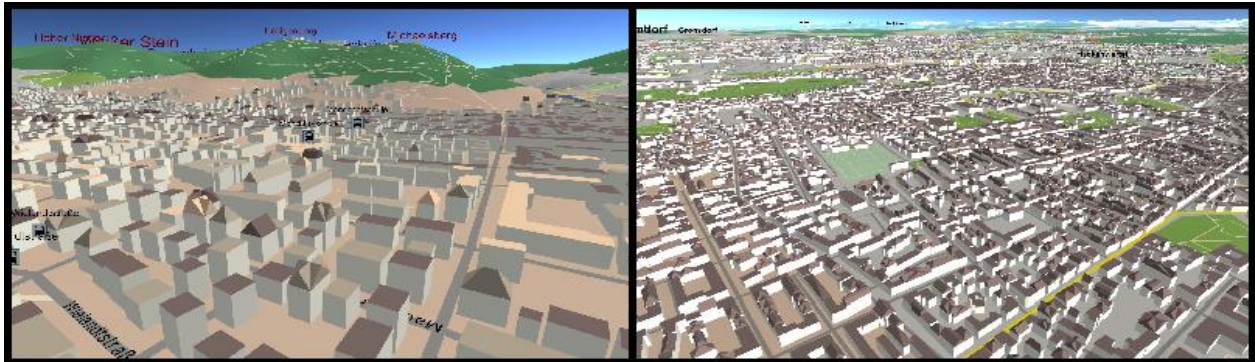


Figure 2.3- Map View of Heidelberg (DE) and Munich (DE) in OSM-3D

Figure 2.4 shows the general structure of the OSM-3D working principle. OSM data is retrieved as change sets via the API using Osmosis. It is then passed through a chain of update processes. **DEMtile Generator** combines SRTM and OSM data and creates triangulated terrain tiles in various sizes. **Building Generator** render buildings as polyhedron, extruding from footprints with flat roofs. After processing Labels, building and Terrain Tiles are gathered in **3D database** and items are put in a container called W3DS. **W3DS** is a portrayal service for three-dimensional geodata such as landscape models, city models, textured building models, vegetation objects, and street furniture. **XNavigator** client is used as 3D viewer which uses Java3D scenegraph technology and JOGL [12].

2.2. EXISTING 3D URBAN GENERATORS

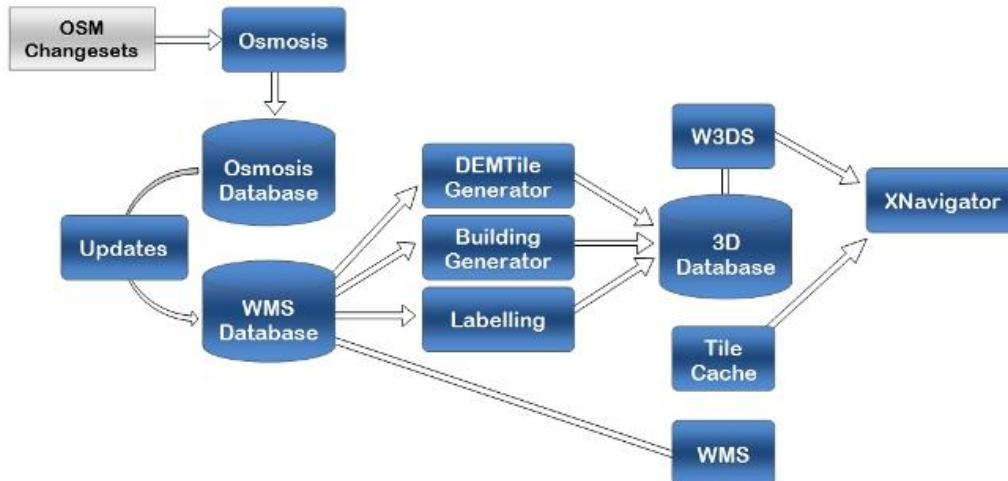


Figure 2.4- OSM-3D Processing Stages [12]

OSM-3D was one of the best 3D urban generators using OpenStreetMap data and Our Framework’s working principle has similarities with OSM-3D. However, OSM-3D had missing properties when it comes to rendering a “photo-realistic” model. As seen from Figure 2.3, OSM-3D renders building without textures and the project has limited material support.

In addition to material properties, OSM-3D does not contain a physics engine. A physics engine is necessary to add external 3D objects into scene and to transform them. Besides this, vehicle physics is required to implement Camera Van component.

In order to deal with advanced material and physics properties easily, a game engine had become a must for our framework. Therefore, OSM-3D source code, which is implemented with JOGL (Java OpenGL API), was not preferred.

2.2.2. OSM 2 World

OSM2World is a converter that creates three-dimensional models of the world from OpenStreetMap data. It can be used as a stand-alone tool, on a server or as a library in Java programs [13]. OSM2World supports 3 output types:

1. OBJ files that contain a full urban model (Figure 5 –a)
2. POV files for the POV-Ray¹ ray tracer (Figure 5- b)
3. PNG images generated with JOGL (Figure 5 – c)

¹ POV-Ray : Persistence of Vision Ray tracer, A free software tool that create 3D images using ray tracing



Figure 2.5 - Osm2world different output formats (a) 3D .obj file (b) rendered with POVray (c) rendered and converted to .png file

OSM2World library generates good results when it outputs a POV file for ray tracing. However, since we are required to record video stream with multiple camera, it is not possible to render all camera frames using ray tracing which could take hours to complete depending on record time.

An OBJ file would be a better output for our case but because OSM2World is an incomplete project, the “Terrain Elevation” component has not been implemented yet (Figure 2.6). Since OSM2World library does not accept height map it damages the “photo realistic” property.

In addition to this, although OSM2World has slightly better material support than OSM3D, it is still not enough since it does not accept advanced shader properties (e.g., Normal map, specular map). Besides, the physics properties are again missing in this library.

2.2. EXISTING 3D URBAN GENERATORS

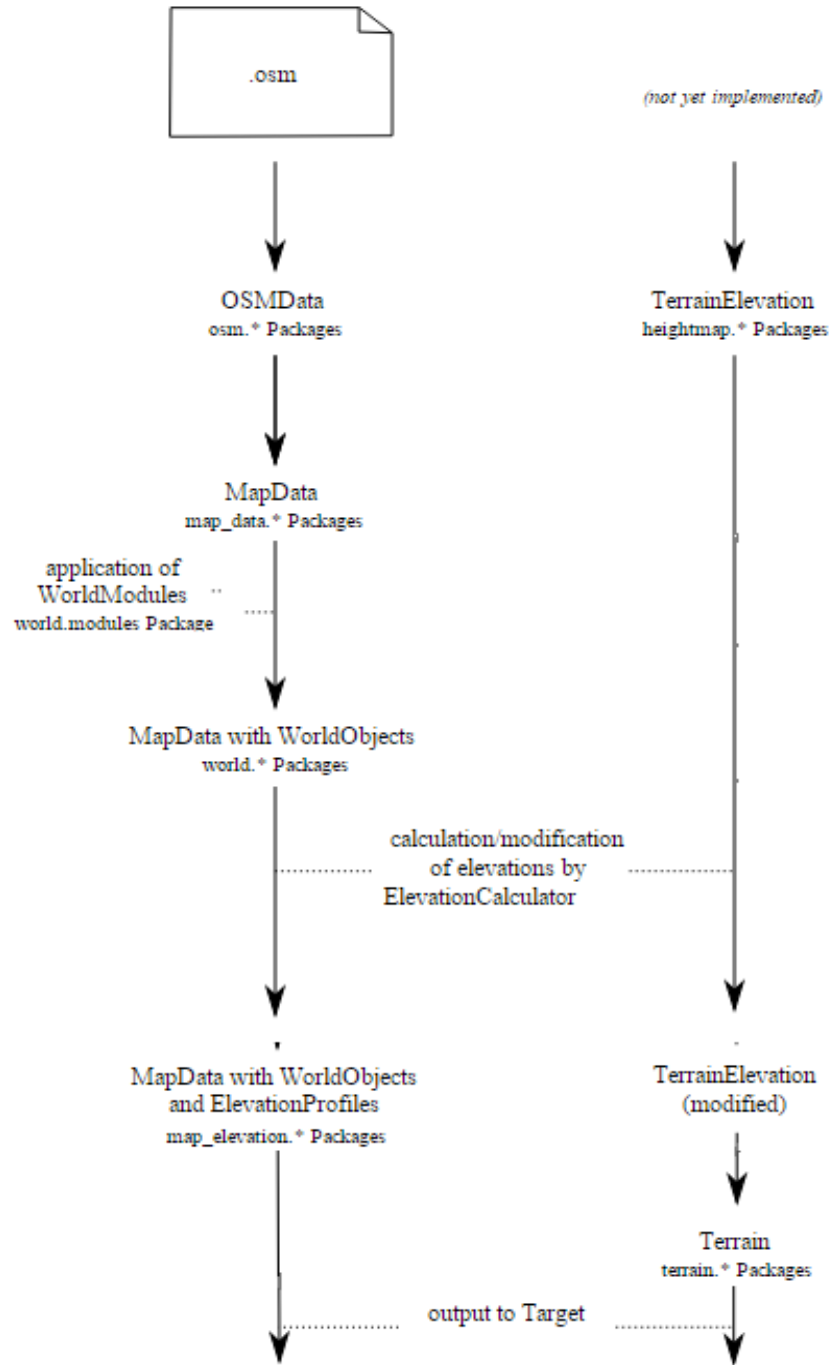


Figure 2.6 - OSM2World Processing Stages [14]

2.2.3. Esri City Engine

Esri CityEngine is a stand-alone software product that provides professional users in architecture, urban planning, entertainment, simulation, GIS, and general 3D content production with a unique conceptual design and modeling solution for the efficient creation of 3D cities and buildings [15]. CityEngine accepts OpenStreetMap data as input and generates 3D models in 5 stages process (Figure 2.7).



Figure 2.7- CityEngine Processing Stages [Source: Esri, 2015]

Since CityEngine is a professional application, it is possible to generate advanced and photo-realistic scenes (Figure 2.8). Even though the results are impressive, it was not used for our framework since it is a commercial product and closed source. Instead, CityEngine gave us ideas about the feature set for our framework's user interface (e.g., Facade Texturing Feature) and became a useful source during the development phase.



Figure 2.8- Esri City Engine Modelling Sample

2.3. RENDERING ENVIRONMENT

Unity 5

Unity is a cross-platform game engine developed by Unity Technologies and used to develop video games for PC, consoles, mobile devices and websites. On March 3, 2015 with the release of Unity 5.0, Unity Technologies made the complete engine available for free including all features, less source code and support [16].

Here are some of the features that are supported by Unity 5:

- Multi-Platform Build
- Object Transform Hierarchy
- Advance Material Properties (Various Shaders, Bump Mapping, Reflection Mapping, Occlusion Mapping, ...)
- Real Time Dynamic Shadows
- Physic engine (NVidia PhysX)
- UI and Event system
- C# .Net scripting

Considering the features listed above, a game engine like Unity is a good choice for implementing our framework rather than using low level graphic libraries (Opengl, DirectX). Among other game engines Unity is preferred for the following reasons:

- Multi-Platform Build
- Community Support

Our framework initially was planned as a desktop application for current needs. However, with the later versions, the framework can be served from a web browser. At this point, the multi-platform support of Unity creates a big difference. Besides support for stand-alone build for Windows, OS X and Linux, Unity supports WebGL and a web player called “Unity Web player”. The current framework can be built in one of the supported web platforms with slight updates in the user interface, if there is a need.

Another reason why Unity is preferred is their great Community Support. Unity game engine is far more popular among developers (Figure 2.9) than any other game development software. The proportion of developers relying on Unity as their primary development tool and using Unity is growing all the time [17].

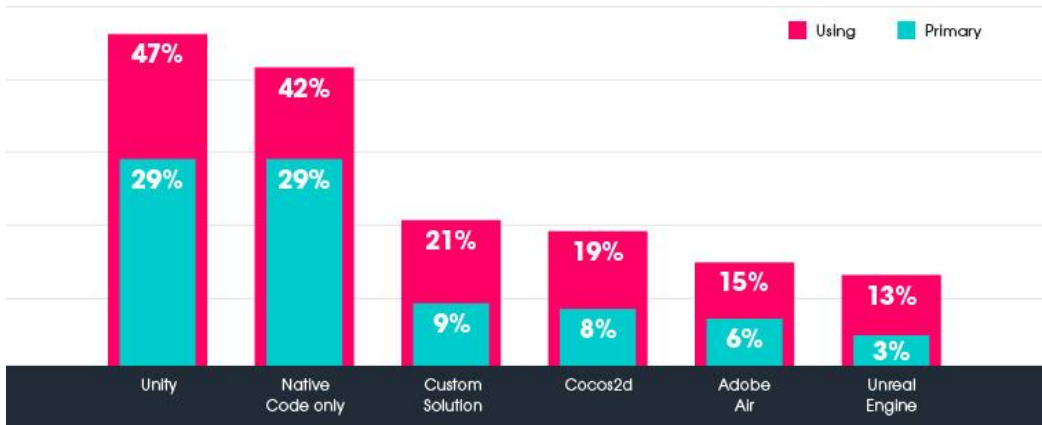


Figure 2.9- Tools Market Distribution [18]

With over 4 million developers (Figure 2.10) and several community platforms (answers.unity3d.com, forum.unity3d.com, stackoverflow.com/tags/unity3d) any type of problem encountered can be easily solved. The second appropriate platform “Unreal Engine”, has three times less users. Any problems arising with Unreal Engine are harder to solve since there is limited community support.

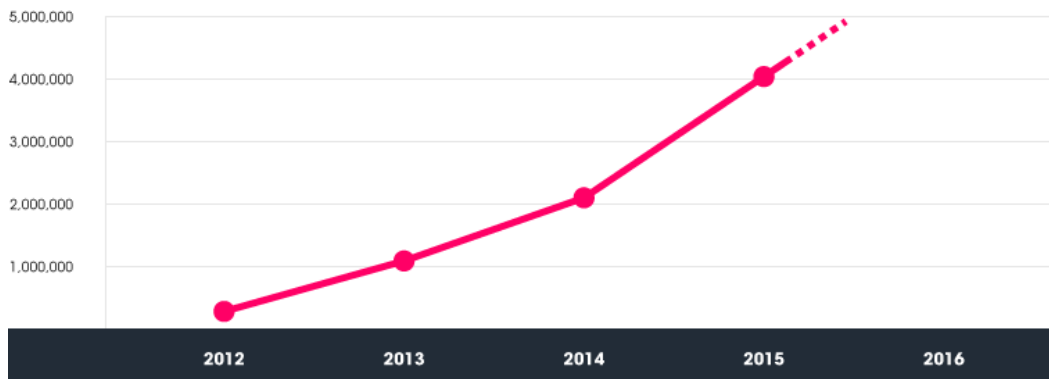


Figure 2.10 - Registered Unity Developers [17]

2.3. RENDERING ENVIRONMENT

Chapter 3 : URBAN GENERATOR

The Urban generator is the biggest component in the framework and responsible for generating 3D urban model procedurally without user interaction. It takes an OpenStreetMap (OSM) xml file as input and produces a 3D scene. Inner components and processing orders are shown in Figure 3.1.

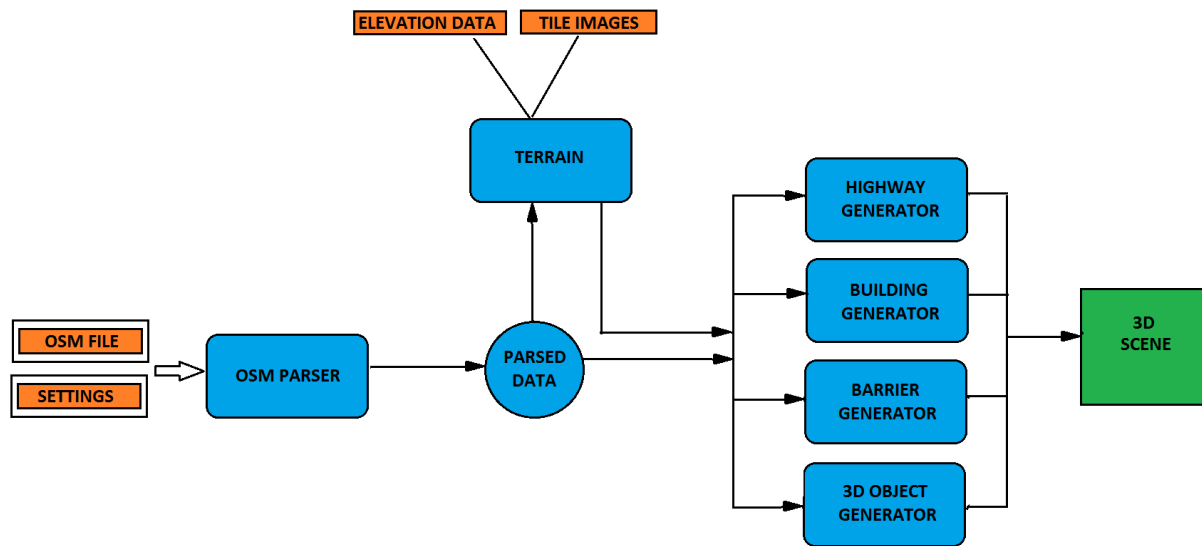


Figure 3.1 - Urban Generator Component Diagram

OSM parser first reads the scene boundaries and passes it to the terrain component. Then the terrain component downloads the necessary elevation data and textures from their sources and generates the 3D terrain.

OSM parser then parses the remaining raw data and stores all elements inside structure lists. It reads the tags assigned to each element and divides them into 4 main categories:

- Building
- Barrier
- Highway
- 3D Object

Once the elements are categorized, 2D data is sent to related generators. At each generator, further categorizations are made and a 3D scene is generated.

3.1. TERRAIN

3.1. TERRAIN

Terrain is the fundamental component of the city generator and it is generated primarily in the scene. Height maps from NASA are used to create 3D mesh and tile images from different map servers are used as texture in order to generate realistic terrains (Figure 3.2).

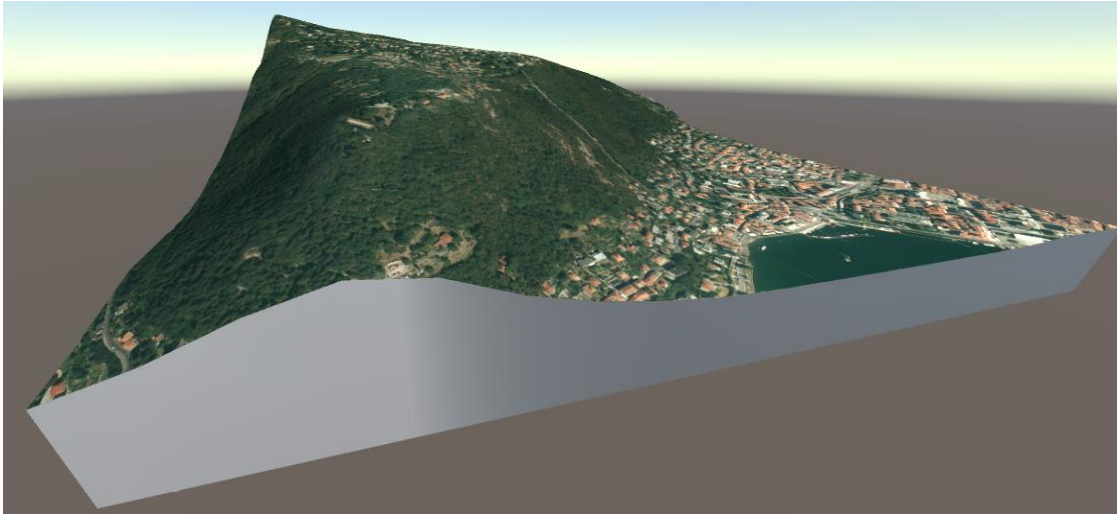


Figure 3.2- Terrain Object

At the beginning, height map data and tile images are incompatible with each other therefore, they require a set of geographical calculations in order to use them together. After necessary geographical conversions are made, 3D terrain mesh is generated using the bounding box from OSM xml file and the NASA height map. Finally, a set of map image is downloaded and applied as texture to the generated surface after few cropping operations.

3.1.1. Geographical Overview

3.1.1.1. WGS 84 Datum

The Earth is shaped like a flattened sphere. This shape is called an ellipsoid. A **Datum** is a model of the earth that is used in mapping. The datum consists of a series of numbers that define the shape and size of the ellipsoid and its orientation in space. A datum is chosen to give the best possible fit to the true shape of the Earth [19].

There are plenty of different datum in use. Many of them are optimized for use in one particular part of the world. WGS-84 (EPSG:4326) is a datum that is used globally (Figure 3.3-A). It comprises a standard coordinate system for use in cartography, geodesy, and navigation including by GPS.

3.1.1.2. Web Mercator Projection

The Earth is curved and maps are flat. The process of flattening out the Earth onto a flat piece of paper or computer screen is a mathematical process called a **Projection**. No matter how you try, the resulting maps always have distortions [19].

Web Mercator (EPSG:3857) is a variation of the Mercator projection and is the de facto standard for Web mapping applications. It rose to prominence when used in the first Google Maps in 2005. It is used by virtually all major online map providers, including Google Maps, Bing Maps, OpenStreetMap and many others [20]. In Web Mercator: north is up everywhere, meridians are equally spaced vertical lines, but areas near the poles are greatly exaggerated (Figure 3.3-B).

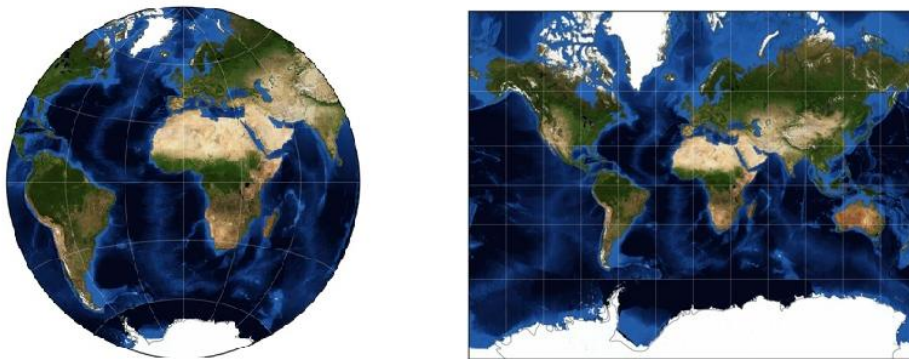


Figure 3.3- (a) WGS 84 Datum (b) Web Mercator projection

3.1.1.3. WGS 84 – Web Mercator Conversions

Not all the sources are using the same coordinate system. That's why it is necessary to make conversions for the compatibility of different sources. WGS² 84 lat/lon coordinate and Web Mercator meter coordinate conversions are done with the equations below [21]:

WGS 84 to Web Mercator Conversion:

$$originShift = 2 * \pi * \frac{6378137.0}{2.0}$$

$$meter_x = longitude * \frac{originShift}{180.0}$$

$$meter_y = \frac{\log\left(\tan\left((90 + latitude) * \frac{\pi}{360}\right)\right)}{\frac{\pi}{180.0}} * \frac{originShift}{180.0}$$

² WGS: World Geodetic System

3.1. TERRAIN

Web Mercator to WGS 84 Conversion:

$$originShift = 2 * \pi * \frac{6378137.0}{2.0}$$

$$longitude = \frac{meter_x}{originShift} * 180.0$$

$$latitude = \frac{180.0}{\pi} * (2 * \operatorname{atan}\left(e^{\frac{meter_y}{originShift} * \pi}\right) - \frac{\pi}{2.0})$$

3.1.2. Mesh Generation

3.1.2.1. NASA SRTM Data

For the elevation data of terrain in the framework, NASA SRTM³ data was used. SRTM data sets result from a collaborative effort by the National Aeronautics and Space Administration (NASA) and the National Geospatial-Intelligence Agency (NGA - previously known as the National Imagery and Mapping Agency, or NIMA), as well as the participation of the German and Italian space agencies, to generate a near-global digital elevation model (DEM) of the Earth using radar interferometry [22].

SRTM data is organized into individual rasterized cells, or tiles, each covering one degree by one degree in latitude and longitude. Sample spacing for individual data points is either 1 arc-second, 3 arc-seconds, or 30 arc-seconds, referred to as SRTM1, SRTM3 and SRTM30, respectively. SRTM3 and SRTM30 dataset is globally available while SRTM1 dataset is available for the US only.

SRTM data is processed and delivered continent-by-continent and data for each continent is located in a separate directory on the server. The names of individual data tiles in the directory refer to the WGS 84 longitude and latitude of the lower-left (southwest) corner of the tile. For example, the tile N30W105.hgt is at 30 degrees north latitude and 105 degrees west longitude.

SRTM3 data is sampled at three arc-seconds and contain 1201 lines and 1201 samples with similar overlapping rows and columns. Each sample is a 16-bit signed integer where the byte order is Big Endian⁴ and ranges from -32767 to 32767 meters [22].

³ SRTM: Shuttle Radar Topography Mission

⁴ Big Endian: A standard for ordering bytes of a word where the most significant byte is first.

3.1.2.2. Getting SRTM Data

In the OSM xml file, bounds of the area to be rendered is given under the <bounds> tag (Figure 3.4).

```
<bounds minlat="45.8007000" minlon="9.0806000" maxlat="45.8130000" maxlon="9.0960000"/>
```

Figure 3.4- Bounds Tag OSM xml

Knowing the bounds of the area, appropriate DEM data can be downloaded automatically without asking the user. After reading the bounds, URL request is generated with the following equations:

Filename = (minlat > 0 ? "N" : "S") + String(floor(minlat)) + (minlon > 0 ? "E" : "W")
 + String(floor(minlon)) + ".hgt.zip"

URL = "http://dds.cr.usgs.gov/srtm/version2_1/SRTM3/" + @Continent + "/" + @Filename

Generated URL is requested via a Web Client and a zip file is extracted with UnityZip Library [23]. Then the raw file containing 1201x1201 16-bit signed integers are read into a 2D Array, considering endianness.

3.1.2.3. Mesh Generation Using SRTM Data

Downloaded SRTM data covers a huge area (~100km x ~100km) and it is necessary to crop the elevation data for the rendered area. The elevation data is a 1201x1201 array forming a 1200x1200 grid which has height values at the intersections. The size of each grid rectangle is 1/1200 degree latitude and 1/1200 degree longitude. Knowing the array indexes of Terrain, they are calculated using following equations:

$$Index_{LEFT} = floor(left - floor(left)) * 1200$$

$$Index_{RIGHT} = ceil(right - floor(right)) * 1200$$

$$Index_{TOP} = floor(ceil(top) - top) * 1200$$

$$Index_{BOTTOM} = ceil(ceil(bottom) - bottom) * 1200$$

Indexes calculated above represent the smallest area that covers OSM Bounds (Figure 3.5).

3.1. TERRAIN

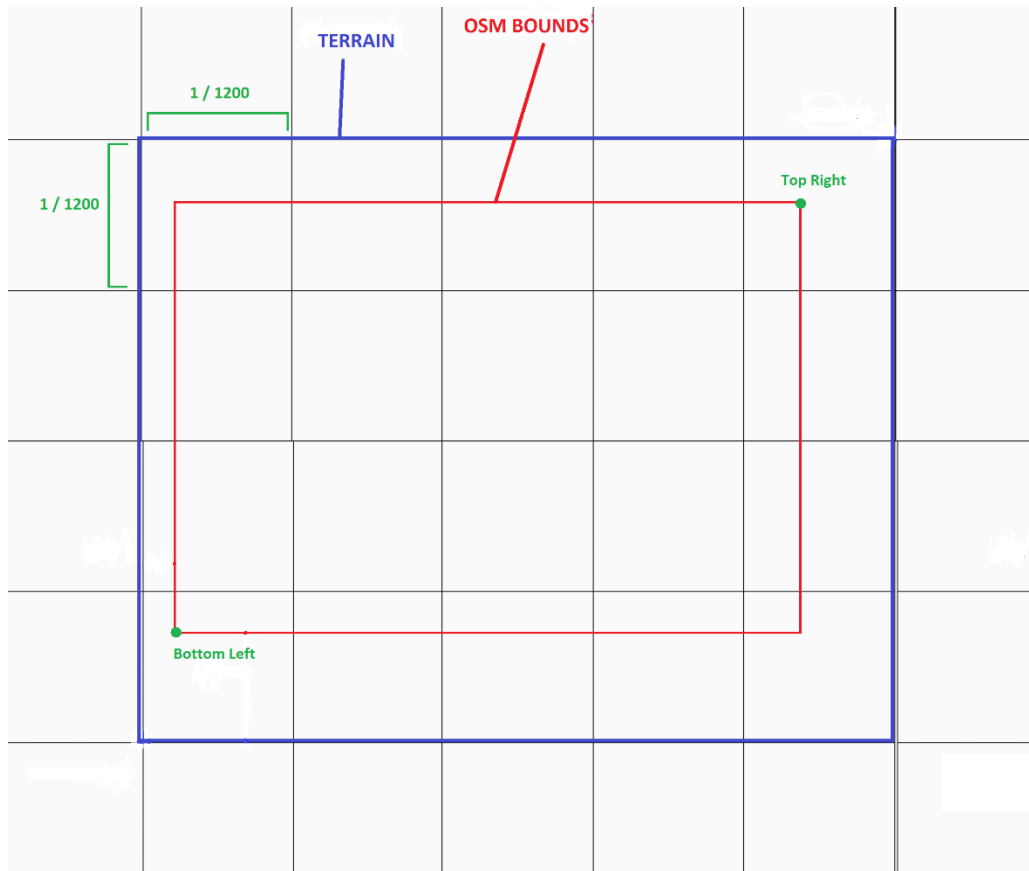


Figure 3.5- Terrain Bounds

As mentioned in Section 3.1.1.1 and 3.1.1.2, WGS 84 datum is designed for a spherical surface. In order to generate Terrain Mesh, flattening of coordinates is necessary by using a proper projection. Web Mercator projection was chosen for the following reasons:

- Popularity among map providers (Bing, Google, OpenSreetMap)
- Conversion from WGS 84 datum is simple
- Uses meter as unit (Meter coordinates are used as world positions inside Unity).

Terrain mesh consists of 2 parts which are terrain surface and the side walls. Surface Mesh is a set of triangle shown in Figure 3.6. XZ-coordinates are Web Mercator coordinates converted from lat/lon and Y-coordinates are the height values taken from SRTM data. Similarly, side walls of terrain are a set of triangles where the upper vertices are taken from outside the boundary of terrain and the lower vertices are the same vertices with height = 0.

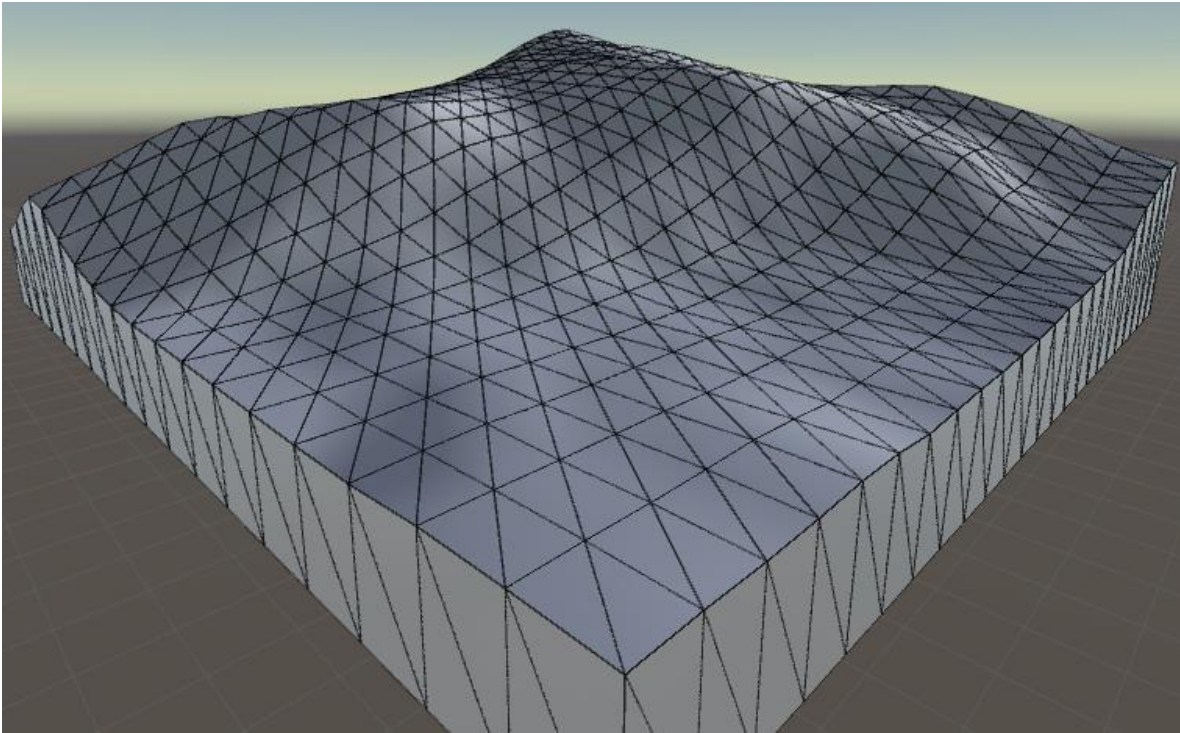


Figure 3.6 – Terrain Mesh (Brunate, Como)

In order to calculate the Normal of the vertices, *Finite Difference Method* is used. For each vertex, samples from 4 sides are taken and the normal vector is calculated with the following formula:

```
# P.xy store the position for which we want to calculate the normals
# height() is a function that return the height at a point in the terrain
# delta is a vector for unit of difference (delta = (1,1,0))
```

```
float hL = height(P.xy - delta.xz);
float hR = height(P.xy + delta.xz);
float hD = height(P.xy - delta.zy);
float hU = height(P.xy + delta.zy);
```

```
Normal.x = hL - hR;
Normal.y = 2.0;
Normal.z = hD - hU;
Normal = normalize(Normal);
```

3.1. TERRAIN

3.1.3. Texture Generation

3.1.3.1. Map Tile Servers

Web maps are now so ubiquitous that it can be easy to forget the qualities that distinguish them from a typical paper city map or world atlas. When you browse a web map, the experience is like panning across a very large, continuous image. By zooming in and out, the amount of detail increases from continents and oceans to streets and buildings. [24].

A continuous image of the world at street level detail would be millions of pixels wide – much too large to download or hold in memory at once. In reality, web maps are made up of many small, square images called tiles. These tiles are typically 256×256 pixels and are placed side-by-side in order to create the illusion of a very large seamless image.

The way that we can see more detail in maps, the difference between country-level maps and street maps, is zoom levels. Higher zoom levels increase the physical size of the displayed map but also increase the amount of detail shown.

To organize these millions of images, web maps use a simple coordinate system. Each tile has a z coordinate describing its zoom level and x and y coordinates describing its position within a square grid for that zoom level: z/x/y.

The very first tile in the web map system is at 0/0/0. Zoom level 0 covers the entire globe (Figure 3.7).

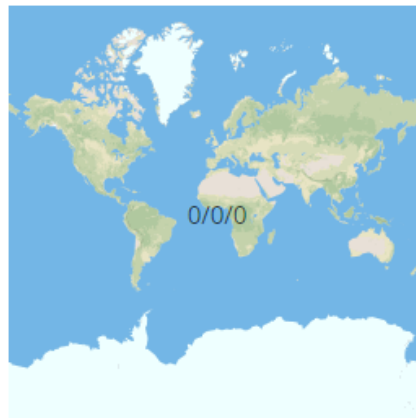


Figure 3.7 – Tile at Zoom Level 0

The next zoom level divides z_0 into four equal squares such that 1/0/0 and 1/1/0 cover the northern hemisphere while 1/0/1 and 1/1/1 cover the southern hemisphere (Figure 3.8). Continuously each zoom level divides the previous zoom level by four. At the zoom level N number of tile is $2^N * 2^N = 4^N$.

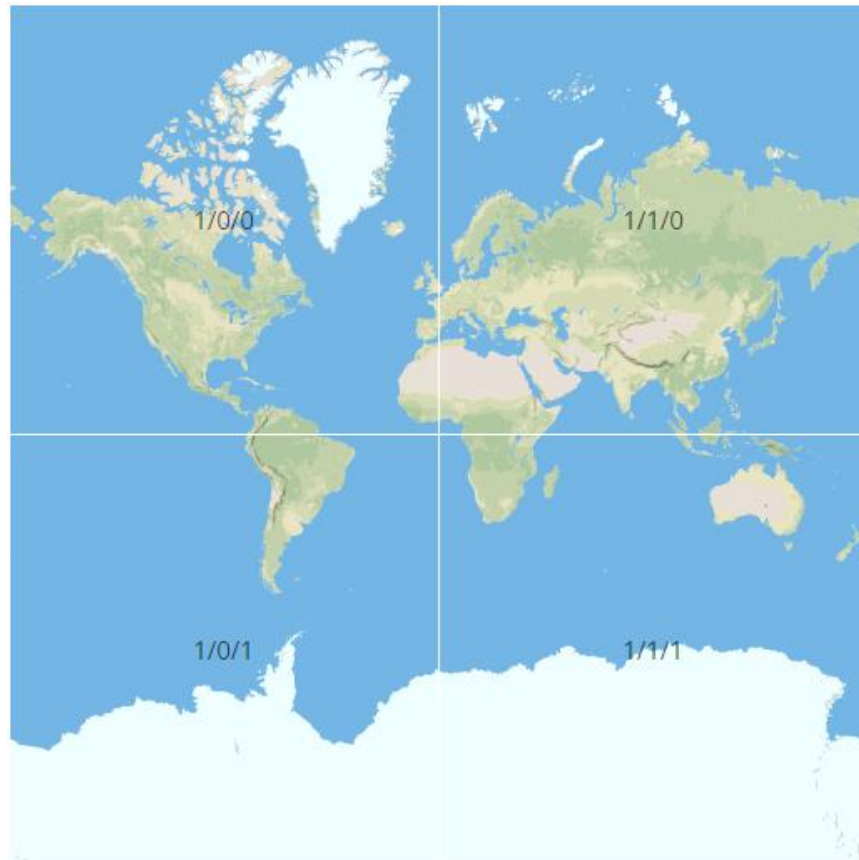


Figure 3.8 - Tiles at Zoom Level 1

There are several map servers that provide tile images between Zoom Level 0 and 20 including Google, Bing and OpenStreetMap. While Google does not allow users to directly access their tiles, it is possible to get tile images from Bing and OpenStreetMap with their XYZ [25] tile servers.

3.1.3.2. Getting Tiles and Caching

In order to download a tile image, 3 parameters are needed which are the x and y index of the tile and the zoom level. The Zoom level is set at 18 as a default. The X and y index of the tile is unknown and should be calculated using meter coordinates.

The first step in finding the tile index is to transform meter coordinates to pixel coordinates. Pixel coordinates can be calculated with the following equations:

3.1. TERRAIN

$$InitialResolution = 2.0 * \pi * \frac{6378137.0}{256}$$

$$Resolution_{ZOOM18} = \frac{InitialResolution}{2^{18}}$$

$$Pixel_x = \frac{(meter_x + \pi * 6378137.0)}{Resolution_{ZOOM18}}$$

$$Pixel_y = \frac{(meter_y + \pi * 6378137.0)}{Resolution_{ZOOM18}}$$

The second step is to find at which tile the point resides. Knowing that a tile size is 256 pixel, the tile index is calculated as follows:

$$Tile_x = Ceiling(Pixel_x/256) - 1$$

$$Tile_y = Ceiling(Pixel_y/256) - 1$$

In our framework, 4 different tile servers are supported:

- Bing Aerial Images (Figure 3.9-a)
- Bing Street Map (Figure 3.9-c)
- OpenStreetMap (Figure 3.9-b)
- MapQuest (Figure 3.9-d)

With the calculated parameters (zoom, tileX, tileY), tile URL's are generated and tiles are downloaded using web requests. Example URL format for MapQuest and OpenStreetMap are:

MapQuest: `http://otile[1234].mqcdn.com/tiles/1.0.0/osm/zoom/x/y.jpg`

OpenStreetMap: `http://[abc].tile.openstreetmap.org/zoom/x/y.png`

Downloading tiles is a time consuming process and depends on the area rendered. Therefore downloaded tiles need to be cached locally. A cache directory is chosen as the Application's persistence data path. The following naming strategy is used to store tile images:

Path: `Application.PersistenceDataPath + "/Textures/Tiles/" + MapType + x_y_zoom.png`



Figure 3.9 - Sample 256x256 Tiles Zoom: 18, x: 137683, y: 93456 (a) Bing Aerial (b) OpenStreetMap (c) Bing Street (d) MapQuest

3.1.3.3. Texture Generation Using Map Tiles

The terrain surface mesh consists of triangles as stated in Section 3.1.2.3. For each triangle, a corresponding tile should be downloaded from the server. Since it would be very slow to search for a tile for each triangle, 8 triangles (4 rectangles) are packed together each time and tile images are downloaded for that pack (Figure 3.10).

Tile indexes are created using the corner vertexes of each mesh pack. After detecting the tile indexes and downloading from the server, the following operations should be done:

- Merging tiles and generating a single image
- Cropping image to fit in mesh pack (Figure 3.10)

3.1. TERRAIN

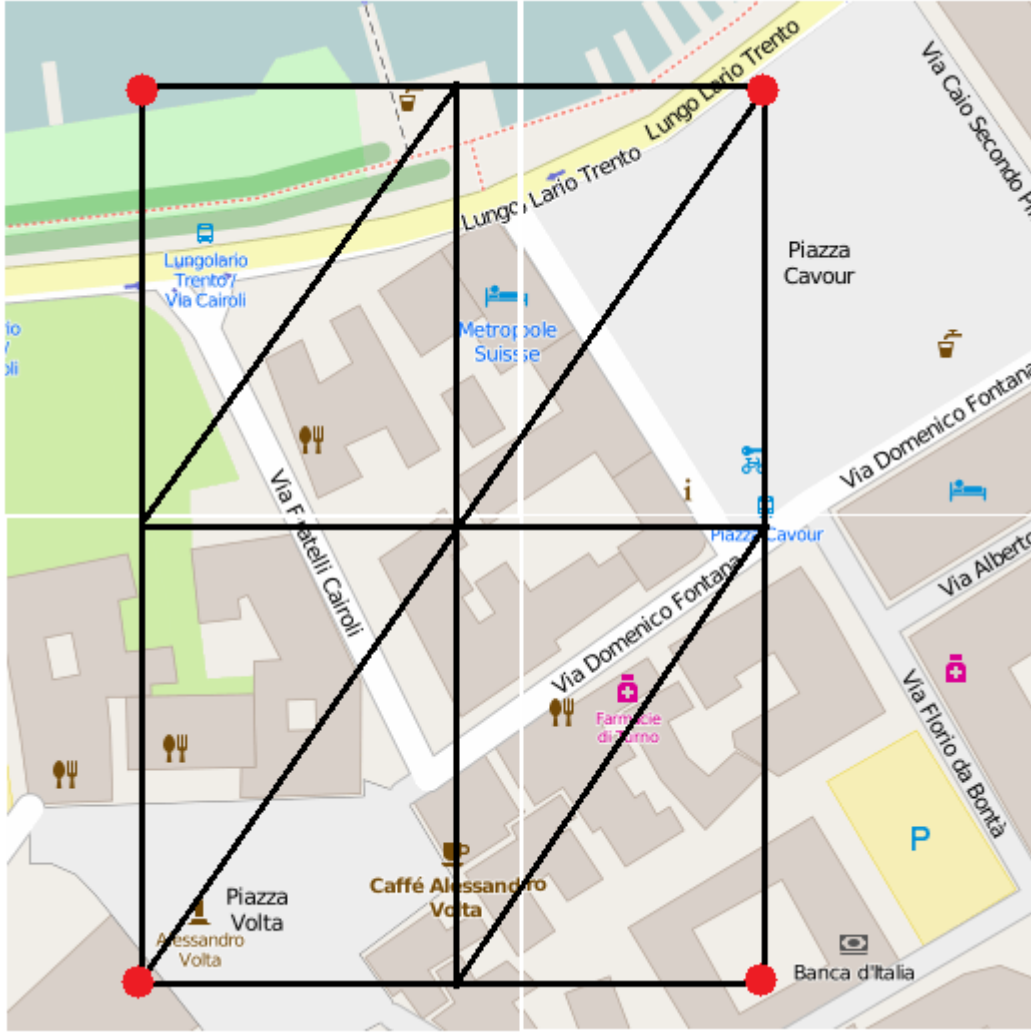


Figure 3.10 - Uncropped Tiles and Terrain Mesh Pack

In order to crop the merged tile image, local pixel coordinates of red points in Figure 3.10 should be calculated. Global pixel coordinates for red points were already calculated while getting tile indexes. To get local pixel coordinate, global pixel coordinate of the top left is calculated for the merged tile which has the (0,0) local pixel coordinate. The difference between global pixels is then accepted as local pixel coordinates.

$$InitialResolution = 2.0 * \pi * \frac{6378137.0}{256}$$

$$Resolution_{ZOOM18} = \frac{InitialResolution}{2^{18}}$$

$$Pixel_{LOCALLEFT} = \frac{Round(meter_{Left} + \pi * 6378137.0)}{Resolution_{ZOOM18}} - (Pixel_{GLOBALLEFT} * 256)$$

$$Pixel_{LOCALRIGHT} = \frac{Round(meter_{Right} + \pi * 6378137.0)}{Resolution_{ZOOM18}} - (Pixel_{GLOBALLEFT} * 256)$$

$$Pixel_{LOCALBOTTOM} = \frac{Round(-meter_{Bottom} + \pi * 6378137.0)}{Resolution_{ZOOM18}} - (Pixel_{GLOBALBOTTOM} * 256)$$

$$Pixel_{LOCALTOP} = \frac{Round(-meter_{Top} + \pi * 6378137.0)}{Resolution_{ZOOM18}} - (Pixel_{GLOBALBOTTOM} * 256)$$

After getting texture coordinates, terrain mesh is updated to its textured version (Figure 3.11).

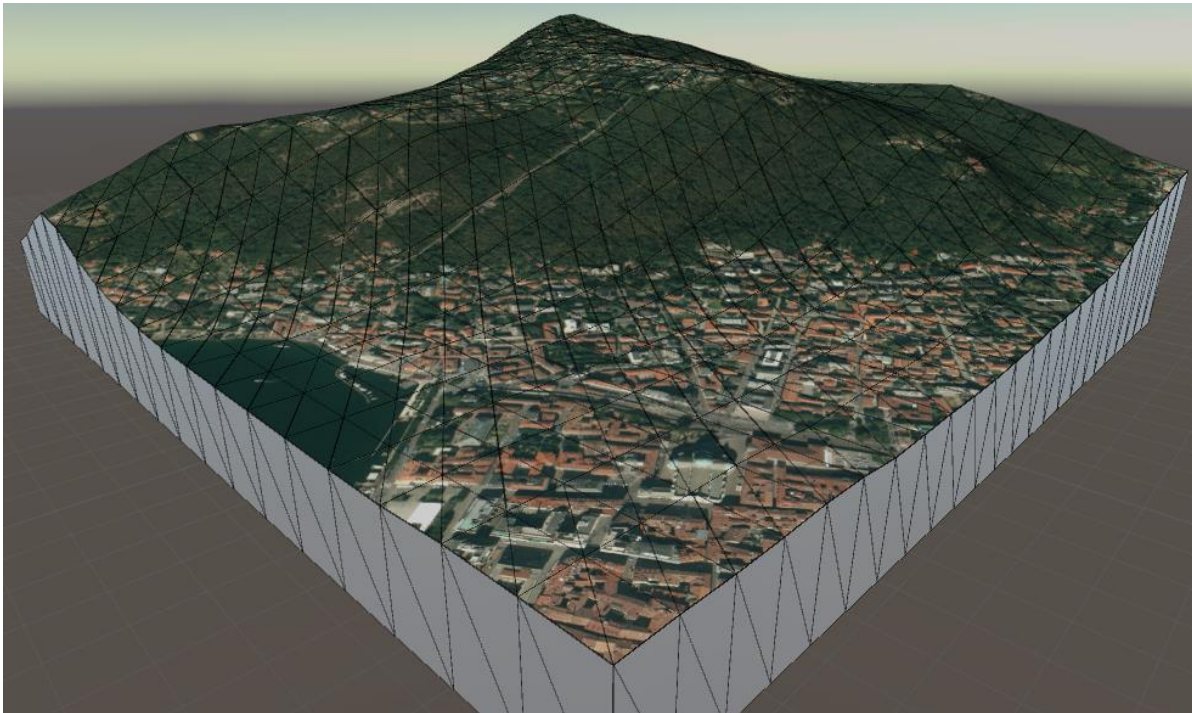


Figure 3.11 - Textured Terrain Surface (Brunate, Como)

3.1.4. Capabilities

Coordinate system is the responsibility of the terrain in the scene. It provides two methods for all other elements in the scene, which are “getHeight()” and “getMeter()”. These methods are used to render elements at the correct world position.

3.1.4.1. Get Height

In OSM xml data, all elements have coordinate values in the XZ-plane and height values are not declared. Y-coordinate of the nodes should be calculated using the elevation data that for terrain object. Terrain object provides a function “getHeight()” to provide a height value for the given coordinates:

3.2. BUILDING

```
float GetHeight(float latitude, float longitude);  
float GetHeight2(float meterX, float meterY);
```

Resolution of elevation data is not high (~90m), therefore heights of objects cannot be assigned to the nearest height value and there is a need for interpolation. Terrain mesh is rendered using triangles and using triangles results in the interpolation of elevation data linearly on the Terrain. For that reason, linear interpolation technique was used to calculate height values using the triangle vertices.

3.1.4.2. Get Meter Coordinates

Unlike the tile server of OSM, xml raw data uses WGS 84 datum for vertex coordinates. As mentioned in Section 3.1.2.3, instead of WGS 84, Web Mercator projection has chosen. Elements getting coordinate values from OSM xml need a coordinate conversion. Terrain component in the framework is responsible for coordinate conversions and it provides 2 functions, using formulas explained in Section 3.1.1.3:

```
Vector2 LatLontoMeters(double latitude, double longitude);  
Vector2 MeterstoLatLon(double meterX, double meterZ);
```

3.2. BUILDING

OSM xml file contains small amount of information for the building objects, therefore some assumptions are made while converting 2D OSM building into 3D. Although generated buildings have lower mesh quality compared to the real world, realistic buildings can be created with textures applied on the each building facade (Figure 3.12).



Figure 3.12 - Building objects

3.2.1. OSM Representation

In osm.xml file, buildings are represented either under <way> or <relation> tag given “key = building”. <way> tag is used when a building can be represented as a single polygon (Figure 3.13). Starting vertex and ending vertex are the same in the data to represent a closed shape which is the footprint of a building. Each way has an id, a set of Nodes that are WGS 84 point coordinates and tags for additional data (e.g. Name, Height).

```
<way id="80505483" visible="true" version="4" changeset="25004933" timestamp="2014-08-25T14:25:03Z" user="Taurus77" uid="2044340">
  <nd ref="939154805"/>
  <nd ref="939154840"/>
  <nd ref="939154842"/>
  <nd ref="939154826"/>
  <nd ref="939154833"/>
  <nd ref="939154792"/>
  <nd ref="939154805"/>
  <tag k="building" v="school"/>
  <tag k="name" v="Scuola media"/>
</way>
```

Figure 3.13- Building represented with <way> tag

Another representation for a building can be done with <relation> tag. If there is a building that needs to be represented as a multi polygon (e.g. buildings with holes), <relation> tag is useful since it consists of a set of different ways. Each relation has one “outer” way which is the outer wall of building and a set of “inner” ways which represent inner holes in the building. (Figure 3.14)

```
<relation id="1120265" visible="true" version="2" changeset="10085597" timestamp="2011-12-10T22:43:12Z" user="mdk" uid="178186">
  <member type="way" ref="71837991" role="outer"/>
  <member type="way" ref="71837994" role="inner"/>
  <member type="way" ref="71837993" role="inner"/>
  <member type="way" ref="71837997" role="inner"/>
  <member type="way" ref="71837989" role="inner"/>
  <member type="way" ref="71837990" role="inner"/>
  <tag k="building" v="yes"/>
  <tag k="type" v="multipolygon"/>
</relation>
```

Figure 3.14- Building represented with <relation> tag

3.2.2. Preprocessing

3.2.2.1. Building Facade

The OSM file only consists of footprints of buildings which are defined by a set of vertices in 2D. In order to construct a building in 3D, building facades must be generated using 2D vertex

3.2. BUILDING

data. A building facade is simply a rectangle for which the lower two vertices are the corners of the footprint and the upper two vertices are the same points shifted by the height of the building on the y -axis.

For building height, OSM has defined 2 tags which are “key= building:height” and “key=building:levels”. The key building:height accepts meters or inches as parameters and the key building:levels accepts the number of a floors in the building. Despite the existence of these tags, they are not commonly used by contributors and are left empty. Therefore, most of the time, the height of building in OSM xml file is undefined. To solve this problem, a default height interval is defined and buildings in the scene are generated randomly with a height in the interval.

While generating vertices for a building facade, the shape of the terrain should be considered. Even if a building is placed on uneven terrain, building’s up vector should be $V_{up}(0,1,0)$. That means the roof of the building should be parallel to the xz -plane (Figure 3.15).



Figure 3.15- Building with (a) handled and (b) unhandled Building Height

In order to make buildings parallel to the xz -plane, an additional variable “equalized building height” value is defined. This value is the y -component of the roof plane. In order to calculate equalized building height, the lowest point of the building is detected. After that, building height is added to the lowest y -value.

$$\text{Equalized Building Height} = \text{Plowest.y} + \text{Building Height}$$

3.2.2.2. Roof

For roof data, OSM xml has “key=roof:shape” tag to define the roof type of the building (Figure 3.16). In addition to roof shape, “roof:height”, “roof:orientation”, “roof:color” and several others tags are available depending on the roof type.









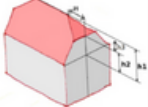
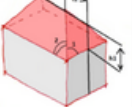
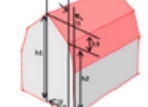
Image								
roof:shape	flat	gabled	half-hipped	hipped	pyramidal	gambrel	dome	round
Necessary parameters	-	roof:height and roof:orientation	 roof:height (h1)	 roof:height (h1)	roof:height	 roof:height (h1)	roof:height	roof:height
unnecessary parameters	-	gabled	one of parameters: highest_pitch:min_height (h2) or highest_pitch:height (h3)	pitch:length (l1)	-	one of parameters: lowest_pitch:height (h2), highest_pitch:height (h3), lowest_pitch:length (l1), highest_pitch:length (l2)	-	-
default values	roof:height=0	roof:orientation=along [long side of bbox]	roof:orientation=along, h2=0.5*h1	roof:orientation=along, angle 1 = angle 2 (redefines by pitch:length)	-	roof:orientation=along, lowest_pitch:height=roof:height*0.5 (redefines by one of parameters: lowest_pitch:height (h2), highest_pitch:height (h3), lowest_pitch:length (l1), highest_pitch:length (l2))	-	roof:orientation=along

Figure 3.16- OSM roof types [Source: OSM wiki]

Although these tags exist in the OSM xml scheme, generally most roof data is empty for buildings (e.g., no roof data for the whole Como city). Therefore “roof:shape” tag has not been considered in our framework. Another reason for ignoring roof shapes is the aim of the framework. Main goal for our framework is to collect data from streets and most of the time roofs are invisible in the image records. That’s why custom roof features have low priority in our framework feature list and may be implemented for the next versions.

For the roof of buildings in the framework, a flat roof shape is selected as default. There are 3 cases for drawing a flat roof plane:

- Building has convex shape
- Building has concave shape
- Building have holes inside

3.2. BUILDING

Buildings which have convex shapes can be simply defined as a triangle fan. However for concave-shape buildings and buildings with holes, triangulation operation is necessary. *Triangulation is the decomposition of a polygonal area P into a set of triangles* [26]. For the triangulation of roof polygon, source code at [27] was used (Figure 3.17).

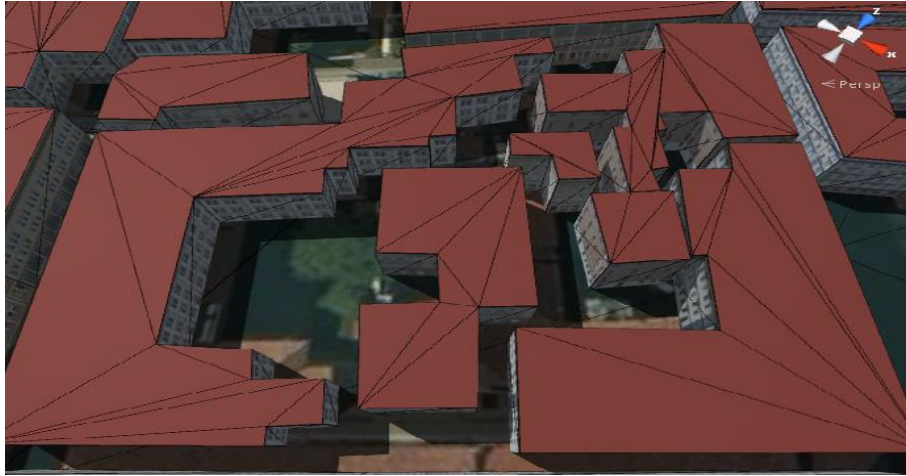


Figure.3.17 - Triangulated Concave Roof Polygon

3.2.3. Rendering

Buildings consist of a list of building facades and a default flat roofs. Building facades are rectangles and can be represented with two triangles (Figure 3.18).

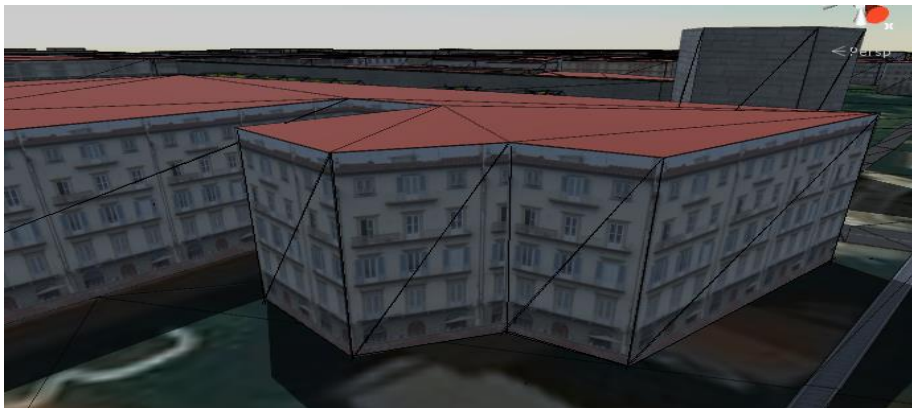


Figure 3.18 - Building Facade Mesh

Unity game engine supports back-face culling method which improves performance of the program. Back-face culling determines whether a polygon of a graphical object is visible. It is a step in the graphical pipeline that tests whether the points in the polygon appear in clockwise or

counter-clockwise order when projected onto the screen. If the polygon is clockwise, it is rendered in the scene. Conversely, if the polygon is counter-clockwise, it is marked as invisible and ignored in the rendering process.

In OSM data, there is no standard for orientation of the building vertices. It can be either clockwise or counter-clockwise (Figure 3.19). In order to benefit from back-face culling, building way orientation should be determined. *Shoelace algorithm* was used to determine way orientation.

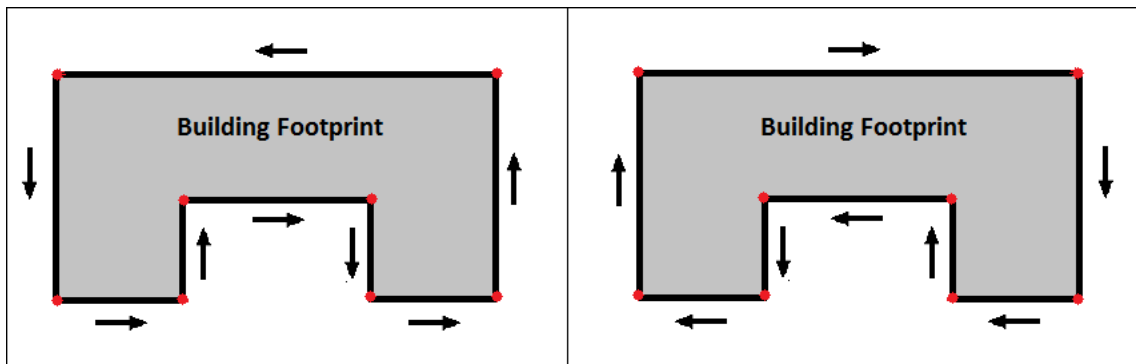


Figure 3.19- Building way orientations. (a) Counter-clockwise (b) clockwise

Shoelace algorithm is normally used to calculate the area of 2D convex & concave polygons where the x and y coordinates of the vertices are known. Another important property of the algorithm is ability to detect the vertex order. Instead of taking the absolute value of the final sum, sign of the result is checked and if the value is positive that means the vertices are ordered clockwise otherwise the vertices are ordered counter-clockwise.

$$\begin{aligned}
 A &= \frac{1}{2} \left| \sum_{i=1}^{n-1} x_i y_{i+1} + x_n y_1 - \sum_{i=1}^{n-1} x_{i+1} y_i - x_1 y_n \right| \\
 &= \frac{1}{2} |x_1 y_2 + x_2 y_3 + \dots + x_{n-1} y_n + x_n y_1 - x_2 y_1 - x_3 y_2 - \dots - x_n y_{n-1} - x_1 y_n|
 \end{aligned}$$

For building facades, 3 different texture types were used to create the material in order to get “photo-realistic” results (Figure 3.20):

- Color Texture
- Normal Map
- Specular Map (Shininess Map)

3.3. BARRIER

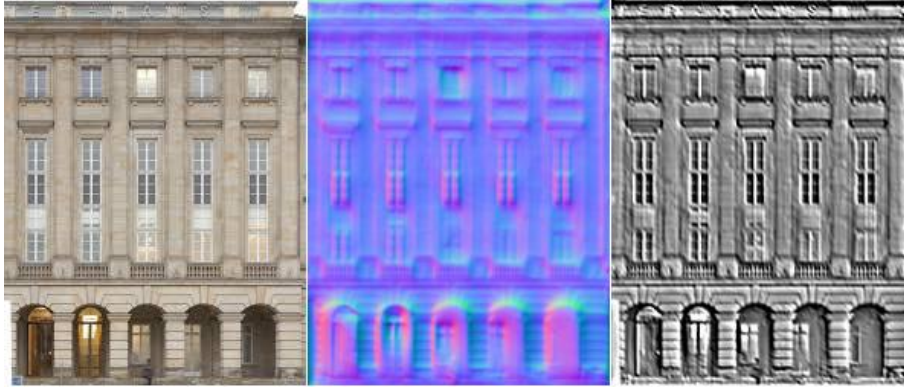


Figure.3.20- Sample Building Material Textures (a) Color Texture (b) Normal map (c) Specular Map

For default, 25 texture sets were added to the framework which are distributed randomly among buildings in the scene. Each texture set has different defined texture sizes and depending on the wall size to which it is applied, the textures are repeated on the wall.

3.3. BARRIER

A barrier is a physical structure which blocks or impedes movement. Walls (Figure 3.21), hedges and fences are examples of barriers. Similar to the buildings, OSM xml file contains less amount of information and some assumptions has to be made such as height and thickness in order to construct the 3D mesh.

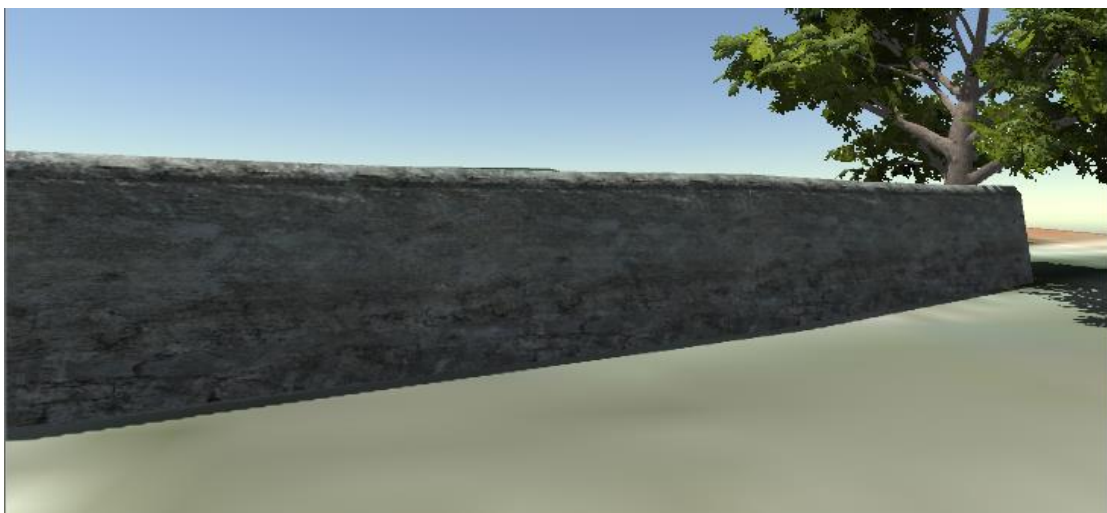


Figure 3.21 - Barrier Object

3.3.1. OSM Representation

In the OSM xml file, barriers are represented under the <way> tag given “key=barrier”. Each way has an id, a set of Nodes that are WGS 84 point coordinates and tags for declaring the barrier type and the area inside (Figure 3.22).

```
<way id="105585981" visible="true" version="2" changeset="8796655" timestamp="2011-07-22T11:24:52Z" user="DarkFlash" uid="131835">
  <nd ref="1215865673"/>
  <nd ref="1215865702"/>
  <nd ref="1215865681"/>
  <nd ref="1369875326"/>
  <nd ref="1215865675"/>
  <nd ref="1215865690"/>
  <nd ref="1215865671"/>
  <nd ref="1369875316"/>
  <nd ref="1215865685"/>
  <nd ref="1215865673"/>
  <tag k="barrier" v="fence"/>
  <tag k="leisure" v="playground"/>
</way>
```

Figure 3.22 - Barrier OSM Xml sample

Barrier types that are supported in the framework are listed below:

- Fence
- Wall
- Retaining Wall
- City Wall (Historic)

3.3.2. Preprocessing

The OSM file only provides a 2D vertex coordinate array of barriers similar to highways and this vertex array needs to be transformed into 3D. There are four different barrier types supported in our framework. However, when they are categorized by their features, there are two different types, fences and walls.

3.3.2.1. Fence

Fences in the scene have two components which are poles and a 2D mesh which supports transparent textures (Figure 3.23).

3.3. BARRIER

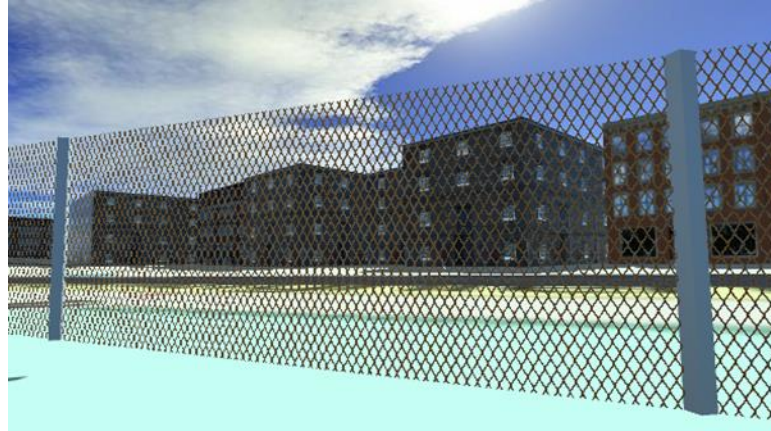


Figure 3.23 - Fence type barrier, Chain-link

Poles are rectangular prisms, with default size 15x15x200 cm. Poles are placed every 4 meters along the fence and are guaranteed to be placed at the corners. 2D mesh generation of the fence is identical to the building facades. 2D mesh of the fence is a rectangle which the lower 2 vertices are taken from the barrier segment and the upper 2 vertices are the same points shifted of the height of barrier. As a default, a chain-link type fence is selected as texture which can be modified via settings.

3.3.2.2. Wall

Wall, retaining wall and historic city wall are categorized as wall types since they have the same mesh types with changing size and texture. Unlike fence, wall mesh is designed as a 3D rectangular prism (Figure 3.24).



Figure 3.24 - Wall type Barriers: (a) retaining wall (b) wall (c) city wall

In order to convert 2D vertex arrays (Figure 4 red dots) to 3D rectangle prisms (Figure 3.25 green dots) the following calculations are made for each Osm Barrier Way segment:

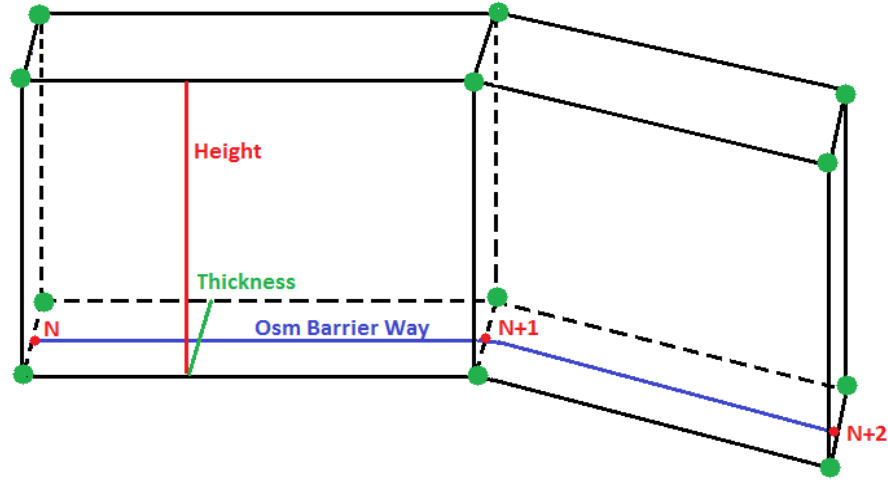


Figure 3.25 - Wall type barrier mesh generation

$$\vec{V}_{Forward} = P_{N+1} - P_N$$

$$\vec{V}_{Up} = (0,1,0)$$

$$\vec{V}_{Right} = \text{Normalise} (\vec{V}_{Forward} \times \vec{V}_{Up})$$

$$\vec{V}_{Left} = -\vec{V}_{Right}$$

$$P_{LowerLeft1} = P_N + \left(\frac{Thickness}{2}\right) * \vec{V}_{Left}$$

$$P_{LowerLeft2} = P_{N+1} + \left(\frac{Thickness}{2}\right) * \vec{V}_{Left}$$

$$P_{LowerRight1} = P_N + \left(\frac{Thickness}{2}\right) * \vec{V}_{Right}$$

$$P_{LowerRight2} = P_{N+1} + \left(\frac{Thickness}{2}\right) * \vec{V}_{Right}$$

$$P_{UpperLeft1} = P_N + \left(\frac{Thickness}{2}\right) * \vec{V}_{Left} + (0, Height, 0)$$

$$P_{UpperLeft2} = P_{N+1} + \left(\frac{Thickness}{2}\right) * \vec{V}_{Left} + (0, Height, 0)$$

$$P_{UpperRight1} = P_N + \left(\frac{Thickness}{2}\right) * \vec{V}_{Right} + (0, Height, 0)$$

$$P_{UpperRight2} = P_{N+1} + \left(\frac{Thickness}{2}\right) * \vec{V}_{Right} + (0, Height, 0)$$

3.4. HIGHWAY

3.4. HIGHWAY

Highways (Figure 3.26) are the most complex components in the urban scene since several steps are required to draw them on the terrain surface.

In order to generate a highway, 3D highway mesh should be generated at first using 2D OSM xml data. After each highway is generated in the scene, intersections (junctions) between highways should be corrected. Then, if they exist, sidewalks should be added to the sides.



Figure 3.26 - Highway object

3.4.1. OSM Representation

In OSM xml file, highways are represented under <way> tag given “key = highway”. Each way has an id, a set of Nodes that are WGS 84 point coordinates and tags for declaring the highway type (Figure 3.27).

```
<way id="112003993" visible="true" version="3" changeset="16769908" timestamp="2013-06-30T21:01:07Z" user="Geofreund1" uid="179581">
  <nd ref="2367810444"/>
  <nd ref="2367810445"/>
  <nd ref="2367810442"/>
  <nd ref="2367810443"/>
  <nd ref="1274363826"/>
  <nd ref="1274364737"/>
  <nd ref="1274362844"/>
  <nd ref="1274364038"/>
  <nd ref="1274362650"/>
  <nd ref="1274363155"/>
  <nd ref="1274362738"/>
  <tag k="highway" v="residential"/>
</way>
```

Figure 3.27 - Highway OSM xml sample

Highway types that are supported in the framework are listed below:

- Residential
- Primary
- Secondary
- Tertiary
- Unclassified
- Tertiary-Link
- Service
- Path
- Railway

For each highway type, a different texture and way width is defined as default. All other features of these types are identical. Railways and rivers in the scene are defined as a highway too because of mesh similarity. However, sidewalk feature and intersection handling features are disabled.

3.4.2. Preprocessing

3.4.2.1. From 2D to 3D processing

The OSM file only provides a 2D vertex coordinate array for highway (Red dots in Figure 3.28-A) and this vertex array is needed to be transformed into 3D.

To generate a 3D Highway from a line segment, Left and Right vectors are calculated and new vertex points are generated by moving size/2 times the left and right vector (Blue dots in Figure 3.28-B). Knowing the up vector and forward vector, calculating left and right vectors is easy. After calculating, the initial point is summed with these vectors and multiplied by the size/2 to obtain left/right side vertex point.

$$\vec{V}_{Up} = (0,1,0)$$

$$\vec{V}_{Forward} = Node_{N+1} - Node_N$$

$$\vec{V}_{Right} = \vec{V}_{Forward} \times \vec{V}_{Up}$$

$$\vec{V}_{Left} = -\vec{V}_{Right}$$

$$P_{Left1} = Node_N + \vec{V}_{Left} * \frac{Size}{2}$$

$$P_{Left2} = Node_{N+1} + \vec{V}_{Left} * \frac{Size}{2}$$

$$P_{Right1} = Node_N + \vec{V}_{Right} * \frac{Size}{2}$$

$$P_{Right2} = Node_{N+1} + \vec{V}_{Right} * \frac{Size}{2}$$

3.4. HIGHWAY

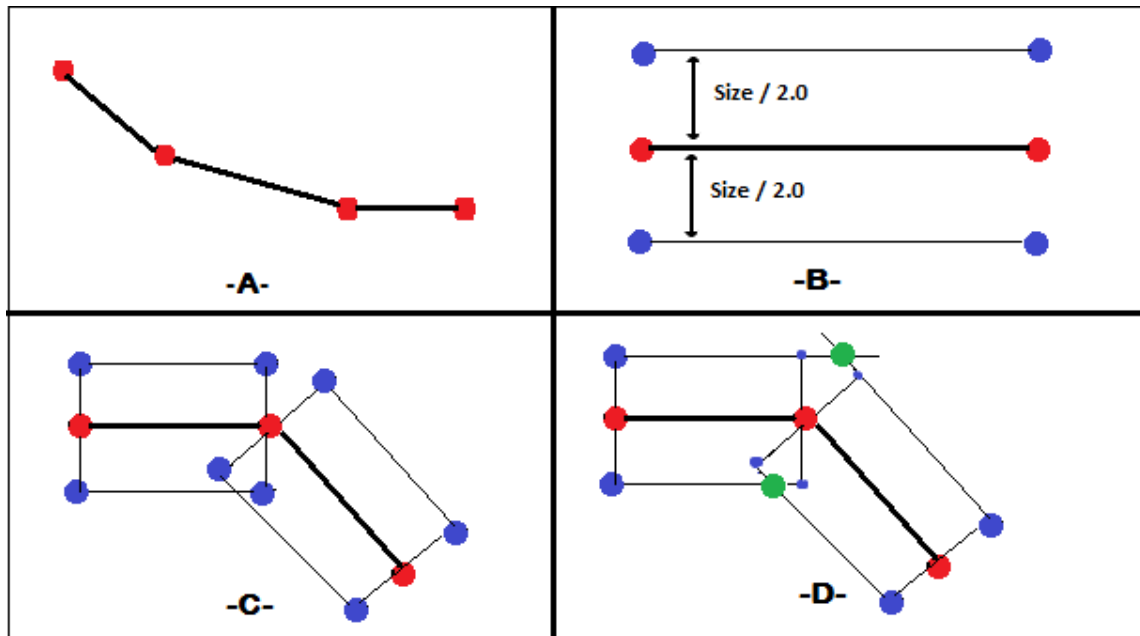


Figure 3.28 – (a) Initial Points (b) Way with a width (c) Problem when angle changes (d) Angle changing Solution

The process introduced so far works only if all the line segments have the same slope. However what happens when a highway consists of line segments with different slopes? (Figure 3.29)

As seen from Figure 3.28 – C for vertices where the slope changes, 2 different left and right vertices are generated while there is only one needed. To prevent this, the algorithm below is followed:

1. For starting node, Left/Right side vertex is generated classically
2. For mid nodes, 2 Left/Right Line Segments Generated (Figure 3.28 –C). Generated Line Segments are intersected with each other and Intersection point is calculated. Calculation point is accepted as Left/Right side vertex (Figure 3.28-D green vertices)
3. For ending node, Left/Right side vertex is generated classically

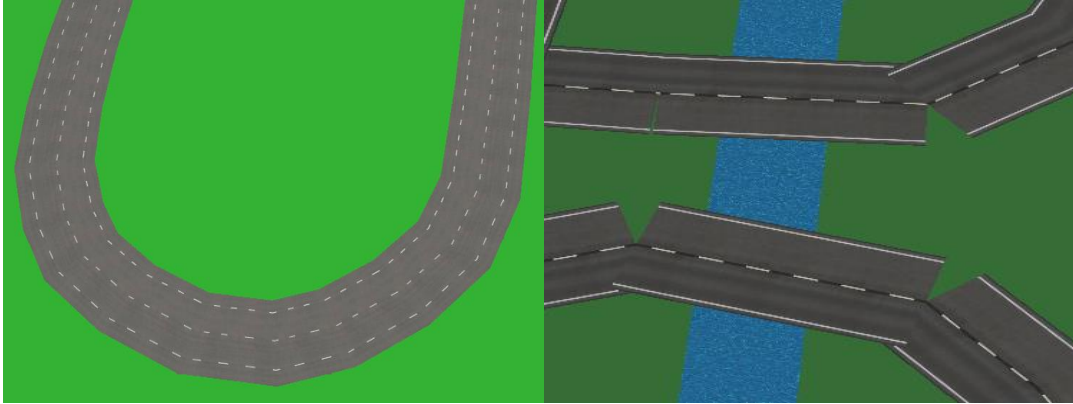


Figure 3.29 - Highway drawn with handled and unhandled slope changes

3.4.2.2. Highway Intersections

If no process is applied for the start and end points of highways, they overlap with each other at the intersection points and a flickering problem of two highway arises. Since the two roads will have the same height value at the intersection point, z-buffer cannot determine which polygon is at the front and that cause inaccuracies. In addition to the flickering problem, for intersection detection it is important not to block highway with a sidewalk at crossroads in case there is a sidewalk defined.

Using OpenStreetMap XML data, intersection detection is easy. 2 highways are intersected if they have a common node. Intersection can be at the beginning node, in the mid nodes or in the end node. Intersection in the mid points requires different techniques compared to the begin-end node intersection. Instead of handling mid-point intersections differently, roads having mid-point intersections are divided from the intersection point. As a result, a defined road only has an intersection at the beginning or at the end. That unification allows one performing single types of intersect correction operation.

For intersection handling, the following process is applied:

1. Identify all intersection points in the scene and make a list that holds intersection data (Figure 3.30-A)
2. Perform intersection algorithm for each intersection (Figure 3.30-B)
3. Fill the empty spaces in the middle (Figure 3.30-C)

3.4. HIGHWAY

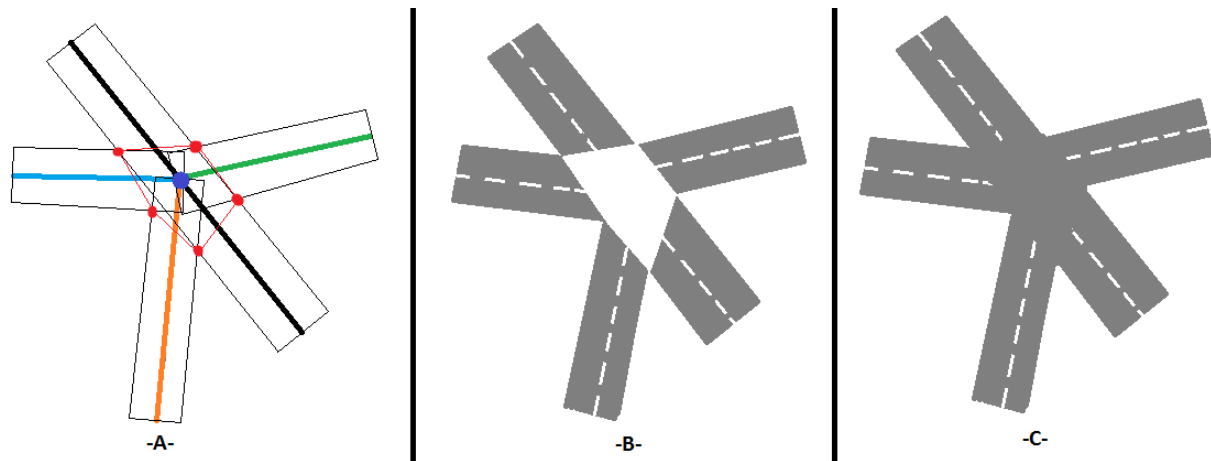


Figure 3.30 - Intersection Processing Stages

To detect an intersection, all of the nodes are held in a buffer with a counter. If a highway contains that node, the counter is increased and the nodes having more than one point are accepted as an intersection.

For each intersection, the following information is stored:

- Way ID for each highway
- Intersection Type for each highway (Front or End)
- Adjacent node coordinate for each highway (To calculate road direction vector)

After all intersections are identified and intersection objects are generated, the following algorithm is applied for each intersection object:

1. Forward vector and Left/Right line segments are generated for each highway
2. Angles between vectors are calculated
3. Intersection truth table is generated
4. Angles are sorted into a buffer in ascending order
5. Until Intersection truth table is completely filled:
 - 5.1. Pop an intersection angle from the top of buffer
 - 5.2. Check truth table if highways belonging to that angle have already been corrected
 - 5.3. If the truth table allows an intersection, intersect the 2 highways Left/Right line segment by checking start or end point intersection and update the highway vertex list. Update the truth table.
 - 5.4. If the truth table does not allow the intersection, change angle to $360 - \alpha$ and push the angle to the end of intersection angles buffer.

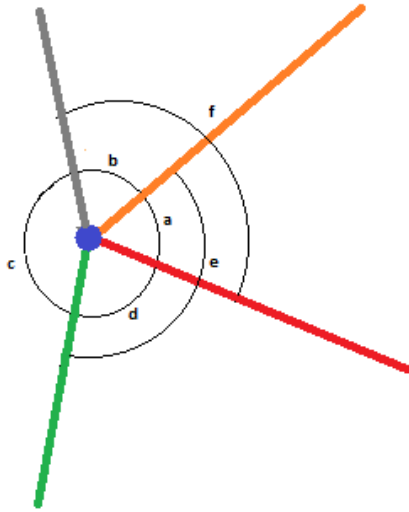


Figure 3.31 - Road Half Vectors and Angles

Angles for highways are calculated using *Cosine theorem*. For N highway in the intersection, $C(N, 2)$ angles are generated (e.g., for 4 highways, 6 different angles are generated in Figure 3.31)

Angles are sorted in ascending order to process the smallest angle each time. The reason for that is to try to handle angles like f and e in Figure-5 as late as possible, since these roads are not directly intersected and those angles should be handled correctly.

After sorting, the first angle is guaranteed to be an acute angle and always has an intersection. For angles >180 , instead of line

segment intersection vector intersection should be used to catch the intersection.

Sample Highway Intersection Process:

Let's take the Figure 3.31 as an example process. The figure represents an intersection of 4 different roads. The first step the algorithm takes is to generate forward vectors and calculate the angle between each vector.

Assuming that $a < b < d < f < c < e$, steps of intersection handling are given in Figure 3.32 and the corresponding steps of the truth table are given in Figure 3.33. As seen from the truth table for the first three steps, intersections are allowed and corresponding highway sides are updated. For step 4 (angle f), the two vectors already have intersections at given sides. So, f angle is reversed and is pushed back to the buffer.

After step 5 is completed, the truth table is completely filled up which is the termination condition for the algorithm.

3.4. HIGHWAY

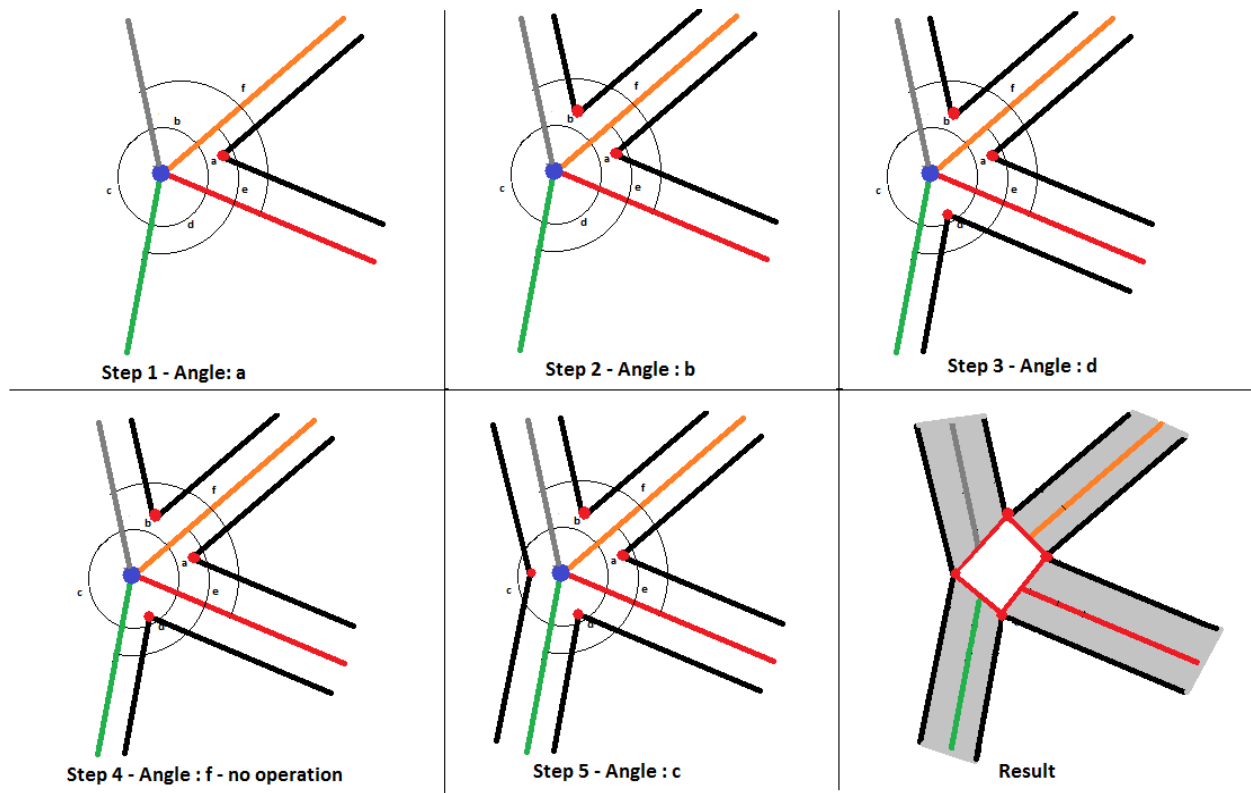


Figure 3.32 - Sample steps of intersection processing

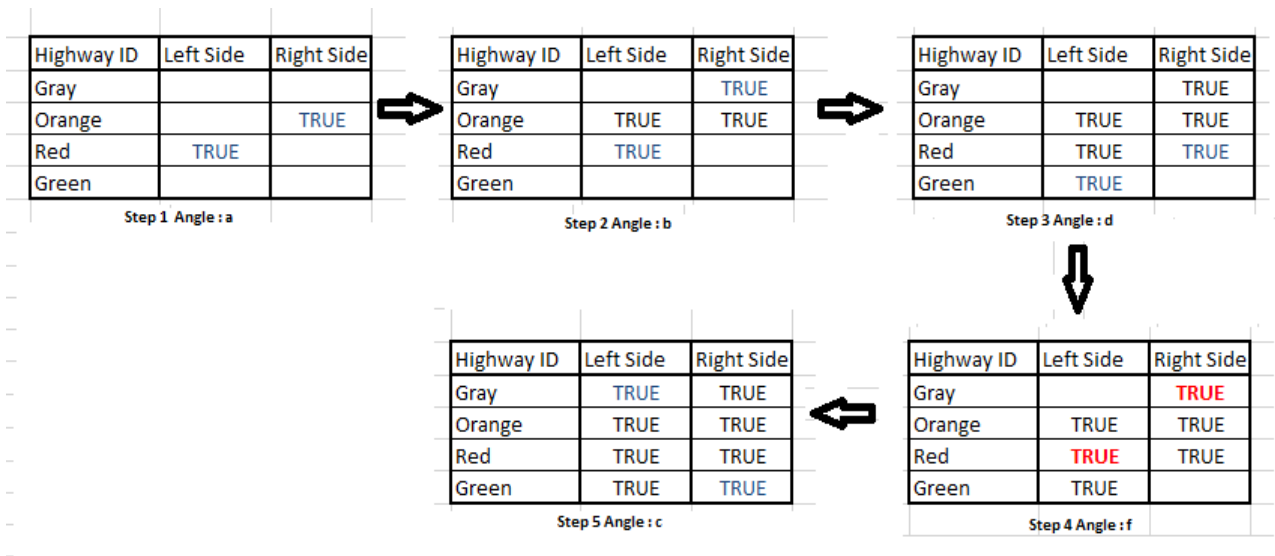


Figure 3.33 - Sample steps of truth table processing

Depending on the road count, the intersection hole can be a triangle, a rectangle or a polygon. For filling up the holes remaining at the intersection, a polygon is rendered accepting intersection points as vertices (Figure 3.34).



Figure 3.34- (a) After intersection with holes (b) Intersection holes filled

3.4.2.3. Sidewalks

In OpenStreetMap XML data, sidewalks are defined with the “sidewalk” tag for highways:

```
<way id="174802721" visible="true" version="2" changeset="16973015" timestamp="2013-07-16T10:50:13Z" user="albertux" uid="118185" >
  <nd ref="1854607351" />
  <nd ref="1854607353" />
  <nd ref="1854607355" />
  <tag k="highway" v="pedestrian" />
  <tag k="sidewalk" v="left" />
  <tag k="sidewalk:size" v="1" />
  <tag k="name" v="Via Albertolli" />
</way>
```

Figure 3.35 - Side walk representation

Highways and sidewalks have very similar structures. Both highways and sidewalks have left and right side vertex array, a width, and a texture. Sidewalks have additional meshes for left and right side stones and stone height data.

To generate the sidewalk mesh, the highway vertex list is used. For example, the right side of the left sidewalk is identical to left side vertices of highway. To generate the left side, left vector is calculated the same as in Section 3.4.2.1 and vertices are shifted to the left side. With a defined stone height, sidewalk surface mesh is escalated by that height and side stone mesh is rendered again using highway vertices (Figure 3.36).

3.4. HIGHWAY



Figure 3.36- Road with Sidewalk on both sides

Sidewalks have similar problems as with highways at intersection points. If intersection is not handled, overlapping and flickering problems appear (Figure 3.37-a) or, depending on the angle, sidewalks may not connect with each other (Figure 3.37 –c).

Detecting intersection for two sidewalks is easy since the necessary values were already calculated and stored while processing highway intersections. The following algorithm is used for processing intersections:

For each sidewalk:

1. Intersected sidewalk id is detected from highway intersection object
2. Line segments are calculated similar to highway intersections
3. Intersection point is found and vertex list is updated

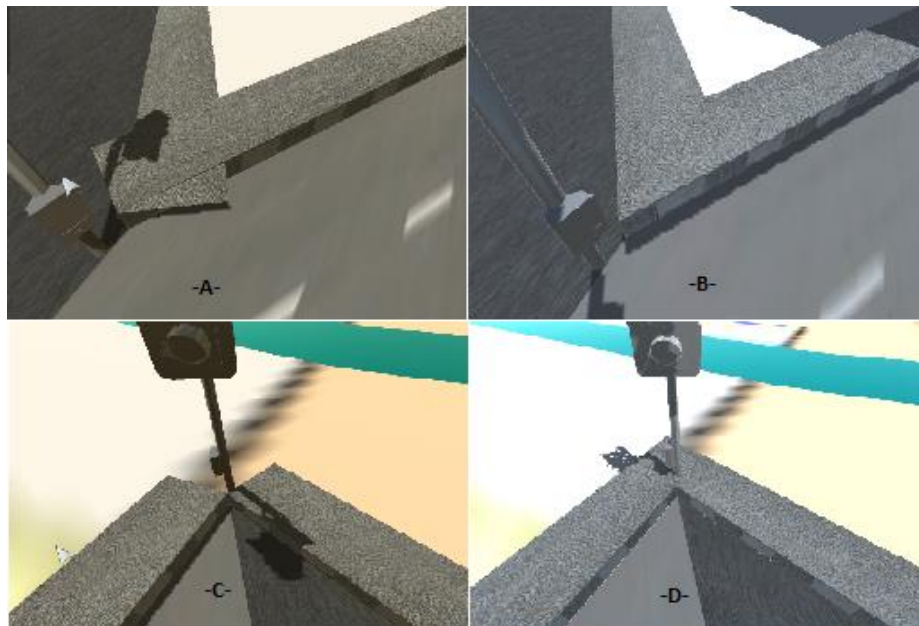


Figure 3.37 - (a) & (c): Unhandled sidewalk intersections, (b) & (d): corrected versions

3.4.2.4. Draping

After roads are converted to 3D and drawn into the scene above the terrain, it can be easily noticed that some parts of the road will remain in the air or some parts will remain invisible under the terrain (Figure 3.38).

One of the main causes of this is that in OpenStreetMap data highways are represented with the minimum number of points. That means, unless there is an angle change in the highway, it is represented as a single long line segment. So far, height samples were only taken for each line segment vertex. Therefore, highways which have long line segments have visibility problems since the height of inter-values were not considered. Terrain's slope change between the two points of the long way segment causes misinterpretation of the height of intermediate points.



Figure 3.38 – (a) Road with Height Map visibility Problem (b) Draped Highway with pin Points

The solution for the problem is “Draping” the highway to the terrain, which is the term for the process of projecting a 2D feature on a 3D surface. For draping roads, left and right side vertices are draped independently to the terrain instead of full polygon draping.

Considering the terrain consists of triangles, roads have to be draped for each edge of the triangle (Figure 3.39).

- Horizontally (Blue dots)
- Vertically (Yellow dots)
- Diagonally (Green dots)

3.4. HIGHWAY

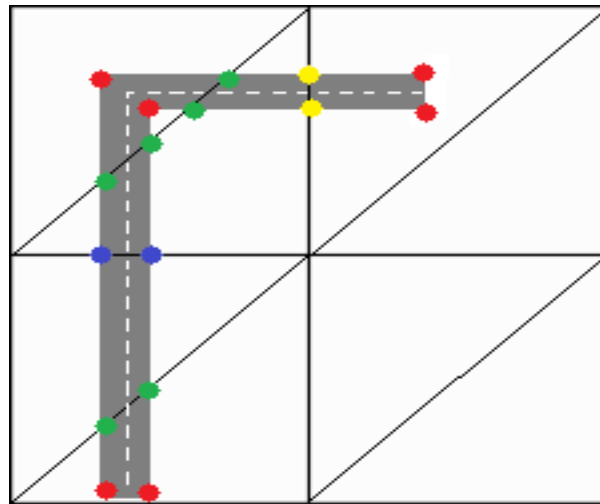


Figure 3.39 - Road Draping

To find intersections, the algorithm below is followed:

For each line segment in highway:

1. From left to right, intersect longitudes. If there is an intersection, add it to point List
2. From top to bottom, intersect latitudes. If there is an intersection, add it to point List
3. From top-left, calculate the diagonal line segment then intersect with it. If there is an intersection, add it to point List

There are three main problems while draping the road to terrain:

- Adding new vertices to the correct place
- Dissymmetrical points for left and right side of the highway
- Line Draping causes some peak points of terrain to remain above the highway

According to the draping algorithm, Intersection control is made for each line segment, starting from left to right. In Figure 3.40 case 1, everything is normal since road direction is left to right and intersection points are added to the point array in order, too.

However in case 2, since the order of the way is right to left, adding point 1 first will cause an error on the highway geometry. To prevent this problem, instead of adding intersections one by one, points are saved in a buffer until all checks for line segment are done. After there are no more intersection points for that line segment, points are added in the detected order.

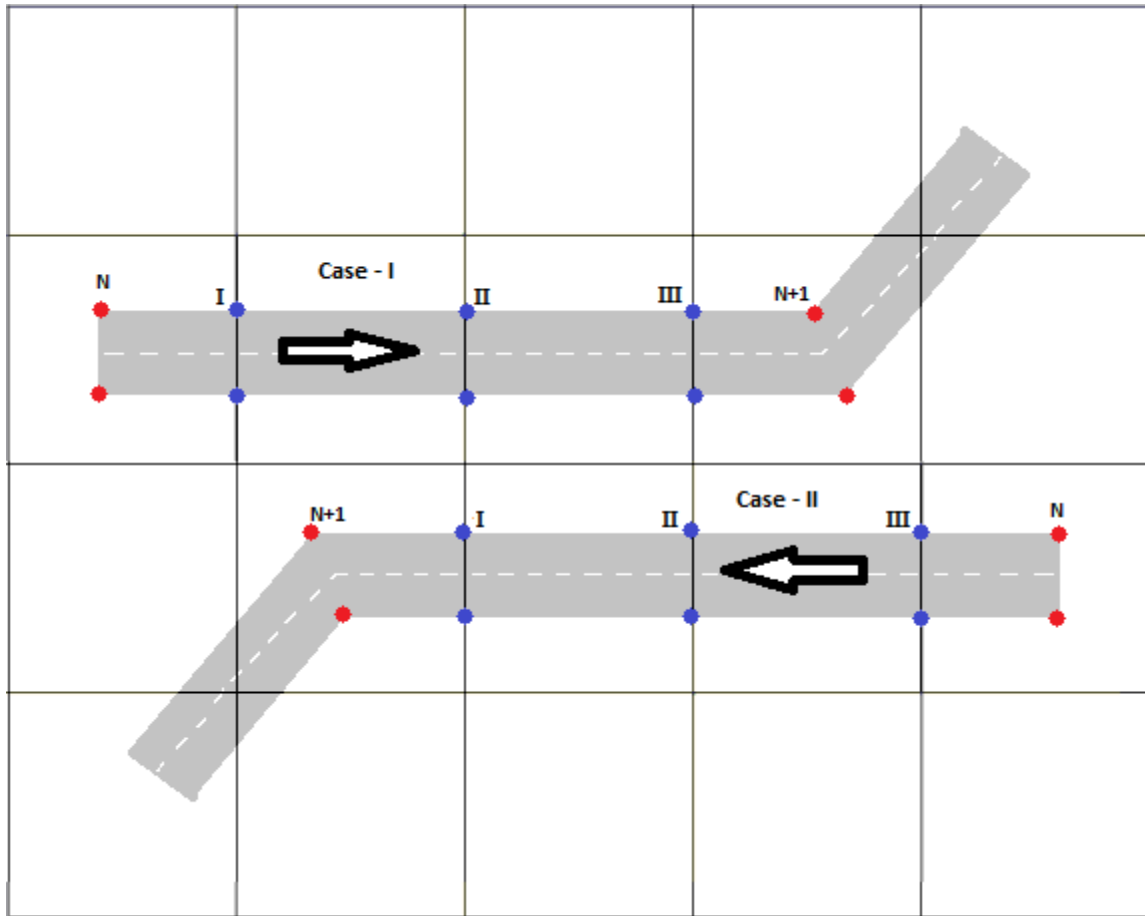


Figure 3.40 - Adding vertices for 2 different cases

Another problem is Dissymmetry in the vertex count. Since left and right side of vertices are considered as a 2D line and draping is done independently, different number of points for each side can occur for the highway which breaks the symmetry. That means, for some intersection points in the left side, there is no corresponding point on the right side (or vice versa). In Figure 3.41a, left side of the road has 5 vertices while right side of the road has 3 vertices.

Dissymmetry in the vertices count makes the highway triangulation and texture coordinate calculation difficult. To solve this problem, where there is no intersection for the other side, a new vertex is created and placed proportionally by looking at intersection placement (Yellow points, Figure-3.41b).

3.4. HIGHWAY

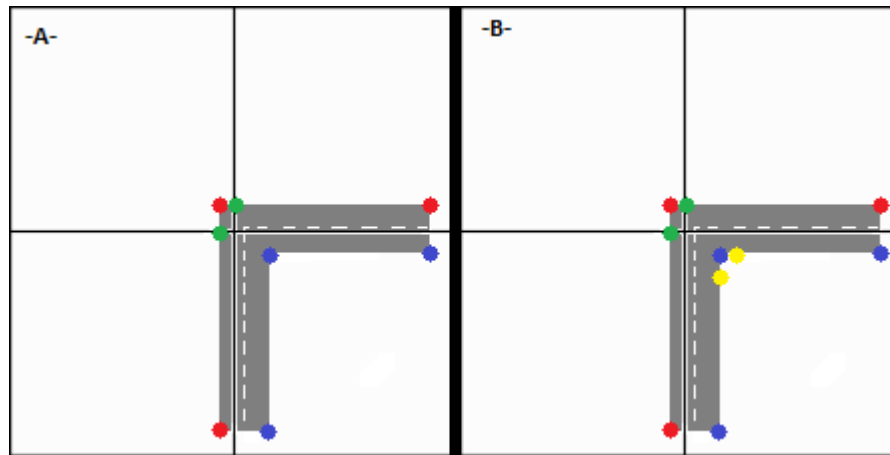


Figure 3.41 - (a) Dissymmetry Problem while draping, (b) Solution

The final problem when draping for some part of the road, if a local peak point remains inside the highway polygon, that part could not be corrected with 2D-line draping and remains above the highway (Figure-3.42). A possible solution for this is generating a third line in the middle. However, doing this will double the triangle count for the highway in the scene and make the highway intersection calculations much more complex. Because of that reason, instead of draping a third line, highways are drawn 20 cm above the terrain. That threshold eliminates most of the peak points, although there still exist some cases where this threshold is insufficient.



Figure 3.42- Highways that remains under terrain

3.4.3. Rendering

Highway mesh is generated by triangulating left and right side vertices. Depending on the OSM xml data and the terrain draping, triangle sizes vary (Figure 3.43).

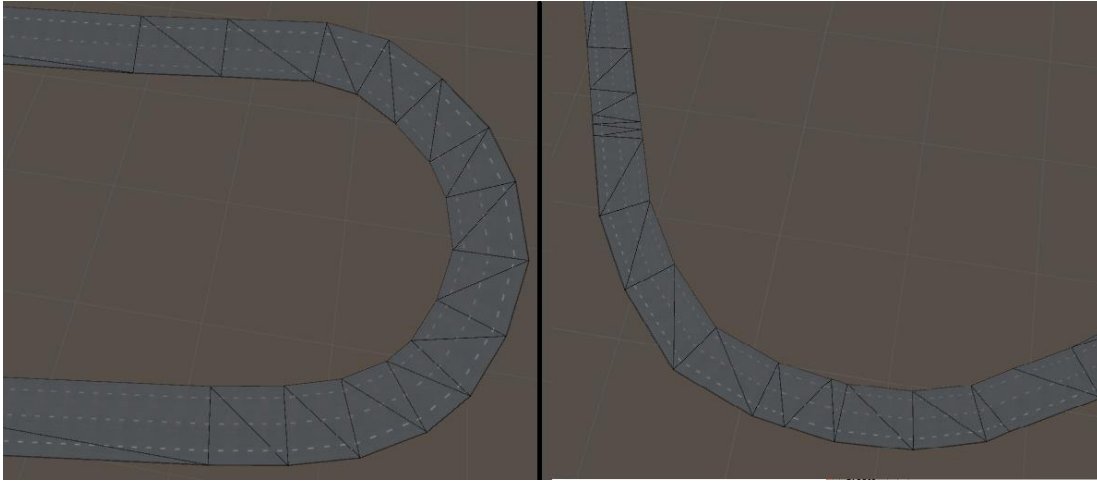


Figure 3.43- Highway Mesh

Texture coordinates are dynamically calculated to make sure that road texture (Figure 3.44) is continuous. Left side vertexes always have 0 value and right side vertexes always have 1 value for their U component of texture coordinates. For V coordinates, length of way segment is calculated for each segment starting from 0. Choosing texture option as “repeat” allow us to assign a coordinate greater than 1. Therefore adding segment length each time to the V coordinate smoothly generates the texture coordinates.



Figure 3.44 - Road Texture Samples

3.5. 3D OBJECTS

3.5. 3D OBJECTS

In addition to highways, buildings and barriers, OSM xml contains dozens of different object data. These objects are gathered in the 3D Object component (Figure 3.45).



Figure 3.45 - 3D Objects

3.5.1. OSM Representation

In OSM xml file, each object is defined under <node> tag which provides WGS 84 coordinates that the object stands and set of key-value pairs for further information (Figure 3.46).

```
<node id="269366847" visible="true" version="12" timestamp="2012-12-01T19:19:52Z" user="ilGianlu" uid="668137" lat="45.8056395" lon="9.0980883">
  <tag k="highway" v="traffic_signals"/>
</node>
<node id="1306629974" visible="true" version="1" timestamp="2011-05-31T11:16:47Z" user="DarkFlash" uid="131835" lat="45.8022805" lon="9.0845693">
  <tag k="natural" v="tree"/>
</node>
<node id="1327205826" visible="true" version="1" timestamp="2011-06-16T08:20:59Z" user="valhalla" uid="18818" lat="45.8115915" lon="9.0842430">
  <tag k="amenity" v="drinking_water"/>
</node>
<node id="1281833929" visible="true" version="1" timestamp="2011-05-12T16:48:40Z" user="DarkFlash" uid="131835" lat="45.8029493" lon="9.0864589">
  <tag k="amenity" v="telephone"/>
</node>
<node id="1281920710" visible="true" version="1" timestamp="2011-05-12T18:53:23Z" user="DarkFlash" uid="131835" lat="45.8013215" lon="9.0863112">
  <tag k="amenity" v="post_box"/>
</node>
```

Figure 3.46 - Samples of 3D object tags

Supported objects in the framework are listed below:

- Drinking Fountain
- Tree
- Traffic Signal
- Telephone Box
- Post Box
- Street Lamp

3.5.2. Preprocessing

Objects such as Trees, Drinking Fountains, Telephone Boxes and Post Boxes do not need preprocessing. Loaded Objects are rendered at the same coordinate given in the xml file. On the other hand, Traffic Signals and Street Lamps coordinates require a correction. These two objects share nodes with highways in the scene. Since highway nodes have been processed and 3D road mesh generated, if the original coordinate is used the objects will appear in the middle of the road (Red dots, Figure 3.47). Instead, they should be translated to the left/right side of the highway (Green dots, Figure 3.47).

Traffic Signals can be placed only at the road intersections. Therefore traffic signal objects are attached to the intersection, specifically the “Intersection Hole” objects. In OSM xml file only one traffic signal is assigned to the intersection point. Instead of drawing a single traffic light, they are given to each highway at the intersection placing highways’ right sides.

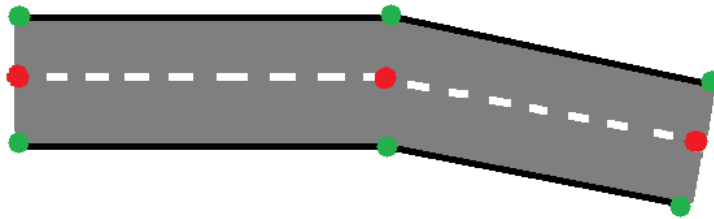


Figure 3.47 - Highway Mesh. Original Vertices (red) and generated vertices (green)

Street Lamps can be any vertex of highway except the begin/end points. OSM xml does not support combining tags with street lamps. Since at the intersections traffic signals will exist, street lamps cannot be at the begin/end Node (The case where there is no Traffic Sign and a Street Lamp exists was omitted). Street Lamps are attached directly to the highway on which they reside. Similar to Traffic Light, Street Lamps are represented by only 1 for each node. There is no tag available for which side of the highway they are, therefore 2 street lamps are generated for each side.

3.5. 3D OBJECTS

Chapter 4 : DATA COLLECTOR

Our main goal with this thesis is generating 3D urban reconstruction datasets. For this purpose, Data Collector component was implemented by allowing users to collect data from the generated 3D urban environment. Similar to the real-world data collectors (Figure 4.1), we simulated two type of collector which are **Camera Van** and **Trekker** which supports both camera and laser scanner (LiDAR⁵). By navigating around the city using them, users are able to generate a collection of video frames and point cloud data (PCD).



Figure 4.1 - (a) Google Trekker (b) Google Camera Car [28]

Before recording starts, users are able to add a set of camera and laser scanner which they can edit their parameters. After recording is finished, the component first generates a log file which contains the positional data during navigation. Then, using the log file and the specified cameras/LiDAR settings, data collector generates the video streams and PCD (Figure 4.2.).

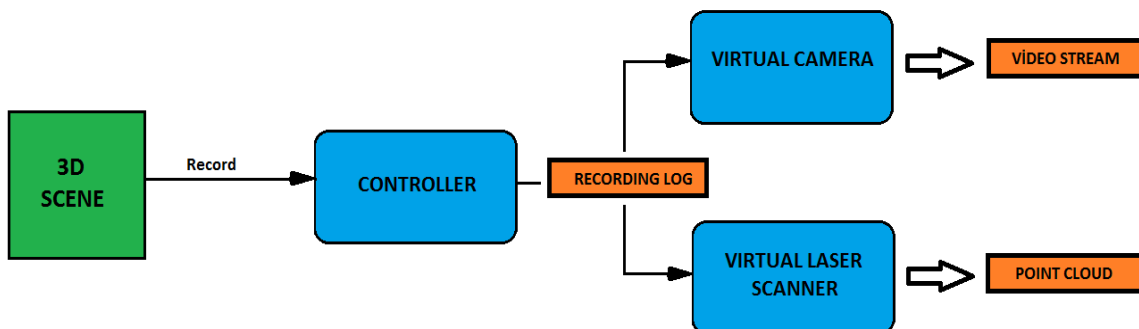


Figure 4.2 - Data Collector Component Diagram

⁵ LiDAR : Laser Imaging Detection and Ranging

4.1. CONTROLLER

4.1. CONTROLLER

4.1.1. Camera Van

Camera van is a 3D van model which consists of 4 wheels and a van body. Using the vehicle physics features of Unity Engine, the van model has been designed similar to its real-world examples. Thus, any distortions in the data caused by the camera van are imitated in the framework.

The Camera Van has a mass of 1000kg and it is affected by the gravity. Center of mass is translated to the bottom of the van (Figure 4.3-A, yellow mark). Because of the angular force if the van makes a sharp turn, it can tilt over. By translating the center of mass, we reduced the effect of angular force and stabilize the van.

Beside the mass, “drag” and “angular drag” parameters were defined for the body of van. Dragging is used to slow down objects. With these parameters, air friction effect was given to body of the van so when there is no applied force the van slows down automatically.

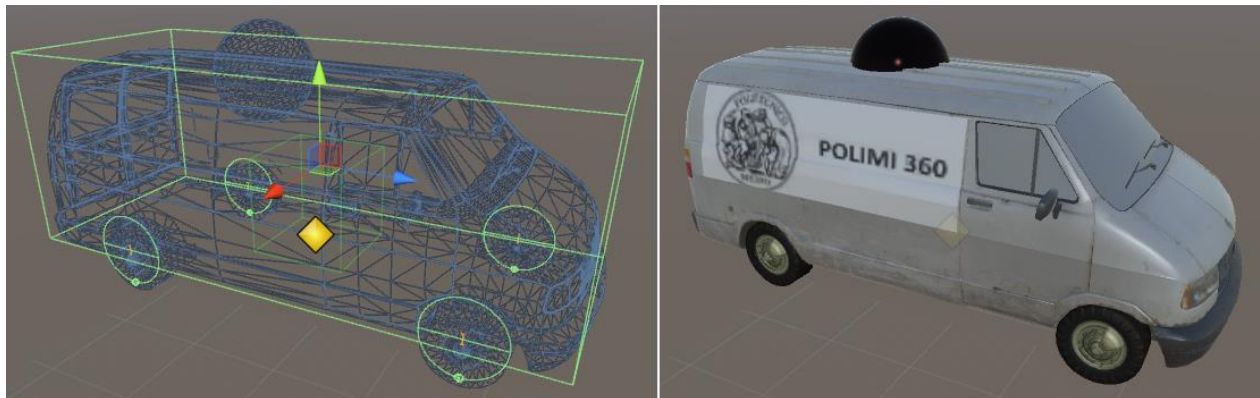


Figure 4.3 - Camera Van (a) mesh and colliders (b) texturized mesh

For each wheel of the van, a “Wheel Collider” is defined (Figure 4.3A, green circles). Wheel colliders allows defining several parameters which are:

- Forward and sideways frictions
- Suspension Distance and Damping Rate
- Motor Torque
- Steer

Friction is necessary to make wheels rotate when a force is applied otherwise they would slip instead of rotating on the highway. For the framework, a default friction value was defined to make the wheel rotate when the van is moving. Different highway surface types (e.g., slippery surfaces where the wheels do not rotate) were ignored.

Suspension is a system that allows relative motion between vehicle and the wheels. It is used to absorb road noises and bumps. For the tires of the camera van, suspension effect was defined and the tires can move up to 10cm on the vertical axis. In addition, a damping value was defined to stop continuous motion of the suspension springs.

In order to move the camera van, a torque power was defined to rear wheels. When the user presses the up arrow key, torque of the wheels are increased up to 100 torque. To turn left/right, again arrow keys are used. A steer angle is defined to front tires which changes from -45 to +45 degree.

4.1.2. Trekker

Trekker enables to feature more places around the generated model – places no car can access. This wearable backpack is outfitted with a camera system on top, and its portability enables us to gather images while maneuvering through tight, narrow spaces or locations only accessible by foot. To simulate the Trekker in our framework, a collection of non-textured 3D human model is used which is taken from Unity's standard assets package (Figure 4.4).

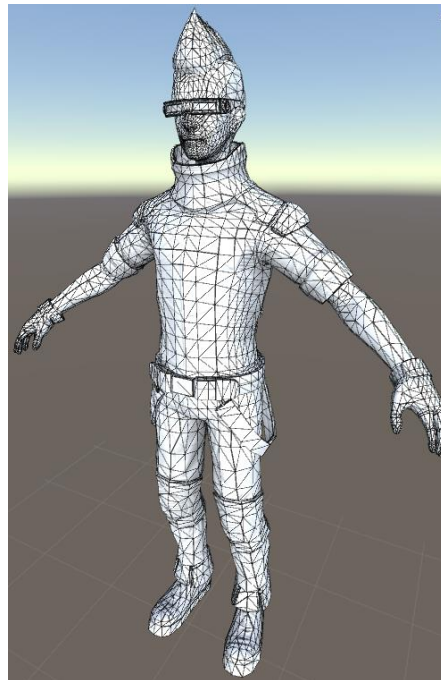


Figure 4.4 - Trekker (From Unity Standard Package)

4.1. CONTROLLER

The Unity package supports different states of the character which are walking, running and jumping. The Trekker can be controlled using arrow keys and space key can be used to jump.

4.1.3. LOG File Generation

Taking screenshots from multiple camera sets and at the same time getting laser data are slow processes and they cannot be done between frames. Therefore, a log file is necessary to hold the positional data of the controller for later processing after the recording is finished. During record, the following parameters are kept for each frame:

- GPS position vector
- Controller rotation vector
- Steer angle
- Velocity

Cameras and lasers scanners may have different frame rates. Therefore, they need different logging entries. For this reason, in addition to parameters above, the type of the log entry is added as parameter. Note that cameras are assumed to have the same frame rates so, only one sample is taken for the camera set. Similarly, only one sample is taken for the laser set assuming they have the same frame rate (Figure 4.5).

Log entries does not contain timestamps which can be calculated with existing values:

$$Timestamp_{EntryX} = \frac{ID_{EntryX}}{Frame Rate}$$

```
<?xml version="1.0" encoding="utf-8"?>
<RecordLog xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <CameraSettings>
    <Cam CameraID="1" Pitch="0" Yaw="90" Roll="0" Position="(0.0, 2.0, 0.0)" FOV="90" />
    <Cam CameraID="2" Pitch="0" Yaw="-90" Roll="0" Position="(0.0, 2.0, 0.0)" FOV="90" />
    <Cam CameraID="3" Pitch="45" Yaw="0" Roll="0" Position="(0.0, 2.0, 0.0)" FOV="90" />
    <Cam CameraID="4" Pitch="0" Yaw="0" Roll="45" Position="(0.0, 2.0, 0.0)" FOV="90" />
  </CameraSettings>
  <LaserSettings>
    <Laser MinDist="1" MaxDist="120" VerticalFOV="40" HorizontalFOV="360" VerticalRes="0.4" HorizontalRes="0.1"
      Position="(0.0, 2.0, 0.0)" Pitch="0" Yaw="0" Roll="0"/>
  </LaserSettings>
  <CameraLog FrameRate="1">
    <CameraEntries>
      <Entry id="0" Rotation="(4.1, 331.5, 0.0)" Position="(-13.6, 233.3, 133.3)" Steer="0" Velocity="18.3686638" />
      <Entry id="1" Rotation="(4.1, 332.0, 359.9)" Position="(-17.0, 232.8, 139.5)" Steer="0" Velocity="31.0804863" />
      <Entry id="2" Rotation="(3.2, 332.8, 359.8)" Position="(-21.7, 232.1, 148.4)" Steer="22.8486652" Velocity="40.50392" />
      <Entry id="3" Rotation="(3.0, 10.1, 1.7)" Position="(-24.0, 231.7, 155.8)" Steer="45" Velocity="6.50709963" />
    </CameraEntries>
  </CameraLog>
  <LaserLog FrameRate="0">
    <LaserEntries />
  </LaserLog>
</RecordLog>
```

Figure 4.5 - Sample Controller Log File

4.2. VIRTUAL CAMERA

Virtual cameras are used to capture video frames from the generated 3D scene. Cameras are implemented using the Unity's camera class and have the following parameters:

- Camera ID
- Pitch, Yaw and Roll
- Position
- Field of View

There can be multi cameras defined for the controller, therefore a unique ID is defined for each camera. Cameras are attached to the controller and orientation of cameras are defined by pitch, yaw and roll which are the rotation values for x, y, and z axis. In addition to orientation, position with respect to controller is defined for each camera by taking center of the controller as the reference point. The field of view (FOV) describes the angular extent of scene by the camera.

When the record was finished and the log file was generated, cameras are activated. Each camera parses the log file and detects the log entries that are belong to their ID's. Then, for each log entry belonging to the camera, position and rotation values are updated using the following equations:

$$ScreenShotPosition_{CAMERA} = Position_{CONTROLLER} + Position_{CAMERA}$$

$$ScreenShotRotation_{CAMERA} = Vector3(Roll, Pitch, Yaw) + Rotation_{CONTROLLER}$$

After the camera is positioned correctly, the screen is captured with the JPG image format. Saved file is named using the following style:

Name: CameraID + “//” + #Frame Number + “.jpg”

4.3. VIRTUAL LASER SCANNER (LiDAR)

Laser scanners (LiDAR) are used to capture 3D point cloud from the environment. By combining them with the video frames, detailed 3D map of the urban environment can be generated. In our framework, a virtual laser scanner was implemented to generate such data.

4.3. VIRTUAL LASER SCANNER (LiDAR)

4.3.1. Point Cloud Data (PCD)

A point cloud is a set of data points in some coordinate system. In a three-dimensional coordinate system, these points are usually defined by X, Y, and Z coordinates, and often are intended to represent the external surface of an object. 3D scanners measure a large number of points on an object's surface, and output a point cloud as a data file. The point cloud represents the set of points that the device has measured.

There are several file formats to store point cloud data and among them, PCD format was chosen for its flexibility and speed. Each PCD file contains a header that identifies and declares certain properties of the point cloud data stored in the file. PCD header contains the following entries [29]:

- **VERSION** – specifies the PCD version
- **FIELDS** – specifies the name of each dimension/field that a point can have. For example:

FIELDS x, y, z	# XYZ coordinates
FIELDS x, y, z, rgb	# XYZ coordinates + Colors
FIELDS x, y, z, normal1, normal2, normal3	# XYZ coordinates + Surface normal

- **SIZE** – specifies the size of each dimension in bytes
- **TYPE** – specifies the type of each dimension as a char. The current accepted types are:
 - **I** - represents signed types int8, int16, and int32
 - **U** - represents unsigned types uint8, uint16 and uint32
 - **F** - represents float types
- **COUNT** – specifies how many elements each dimension has. For example, XYZ data usually has 1 element, but a feature descriptor like the VFH has 308. By default, if COUNT is not present, all dimensions' count is set to 1.
- **WIDTH** – specifies the width of the point cloud dataset in the number of points.
- **HEIGHT** – specifies the height of the point cloud dataset in the number of points.
- **VIEWPOINT** – specifies an acquisition viewpoint for the points in the dataset. The viewpoint information is specified as a translation (tx, ty, tz) + quaternion (qw, qx, qy, qz).
- **POINTS** – specifies the total number of points in the cloud.
- **DATA** – specifies the data type that the point cloud data is stored in. Latest version supports *ascii* and *binary* types.

After the Header is finished in the given order, rest of the data points are listed one entry per line in the PCD file.

4.3.2. PCD Generation

LiDAR is a remote sensing technology that measures distance by illuminating a target with a laser and analyzing the reflected light. A LiDAR is represented with the following parameters:

- Range
- Horizontal FOV
- Vertical FOV
- Horizontal Resolution
- Vertical Resolution
- Frame Rate

Range determines the minimum and the maximum distances that the laser can process. Horizontal and vertical FOV angles defines the viewing plane of the laser and resolutions determines how dense the point cloud is. Finally, frame rate states the capture count per second.

In our Framework, we simulate the lasers with Ray Casting method. Ray Casting is the process of shooting an invisible ray from a point in a specified direction, to detect whether any objects lay in the path of the ray.

To generate a PCD file, a set of Rays are sent from the center of laser scanner to the directions using the FOV and Resolutions parameters of the LiDAR (Figure 4.6). Rays are generated using Unity's Physics engine:

```
bool Raycast(Vector3 origin, Vector3 direction, out RaycastHit hitInfo, float maxDistance);
```

If the ray hits with an object, Raycast function returns true and it fills the *hitInfo* object with the intersection data. If there is no intersection, the function returns false and the *hitInfo* object remains empty. HitInfo object stores the world position and color of the intersection.

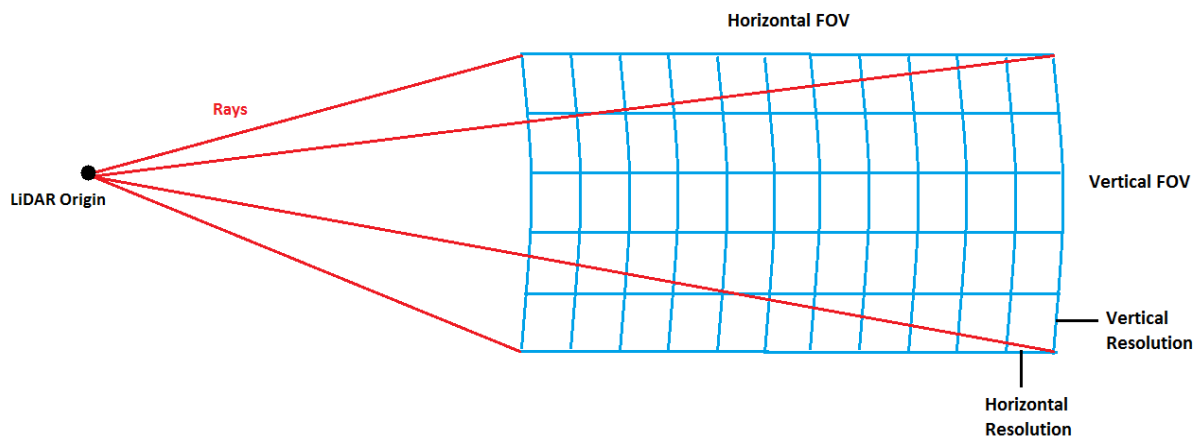


Figure 4.6 - LiDAR Ray casting

4.3. VIRTUAL LASER SCANNER (LiDAR)

In order to get the LiDAR origin, the log file is parsed similar to the camera component and the virtual LiDAR is translated to the corresponding coordinate at each frame. After it is positioned correctly, rays are created using FOV and resolution values which is used to represent the direction vector in spherical coordinate system. Spherical coordinates are converted to the Cartesian coordinates then using the “Raycast” function the distance values are generated. For each laser log entry, different PCD files are created using the following naming style:

Name: “LiDAR” + “/” + #FrameNumber + “.pcd”

Our PCD file is arranged such that, it shows the world position of the intersection points and the distance from the laser. We preferred float as the data type and used ascii format for representing point cloud. A sample PCD header is given below with 360 degree Horizontal FOV and 30 degree Vertical FOV and 1 degree resolutions:

```
#Polimi OSM City Engine PCD
VERSION .7
FIELDS x y z distance
SIZE 4 4 4 4
TYPE F F F F
COUNT 1 1 1 1
WIDTH 360
HEIGHT 30
VIEWPOINT 0 0 0 1 0 0 0
POINTS 10800
DATA ascii
```


Chapter 5 : USER INTERFACE

OSM xml data itself is inadequate to build an urban environment since it is an incomplete dataset. In order to resolve missing information issues, users are required to enter various parameters. As one of our main goals, the generated urban model should be close to its original and users should be able to make edits on the scene to achieve better quality. For these reasons, a user interface is implemented using Unity’s UI system. Figure 5.1 shows components of the user interface which are:

- Load/Save Menu
- Default Settings Menu
- Edit Menu
- Add Object Menu
- Data Collector Menu

Before creating the project, users can edit various parameters on the Urban Model such as textures and sizes of the objects by using the “Default Settings Menu”. Then with the “Load/Save Menu”, users can provide the desired area to be rendered or provide a save file to load an existing project. After the urban model is generated, users can make several edits on specific objects using “Edit Menu” such as editing sizes of highways or editing size of buildings. They can also add new 3D objects (e.g., trees, cars, buildings) using “Add Object Menu”. Finally, users can collect urban data with the “Data Collector Menu” by specifying camera and LiDAR parameters.

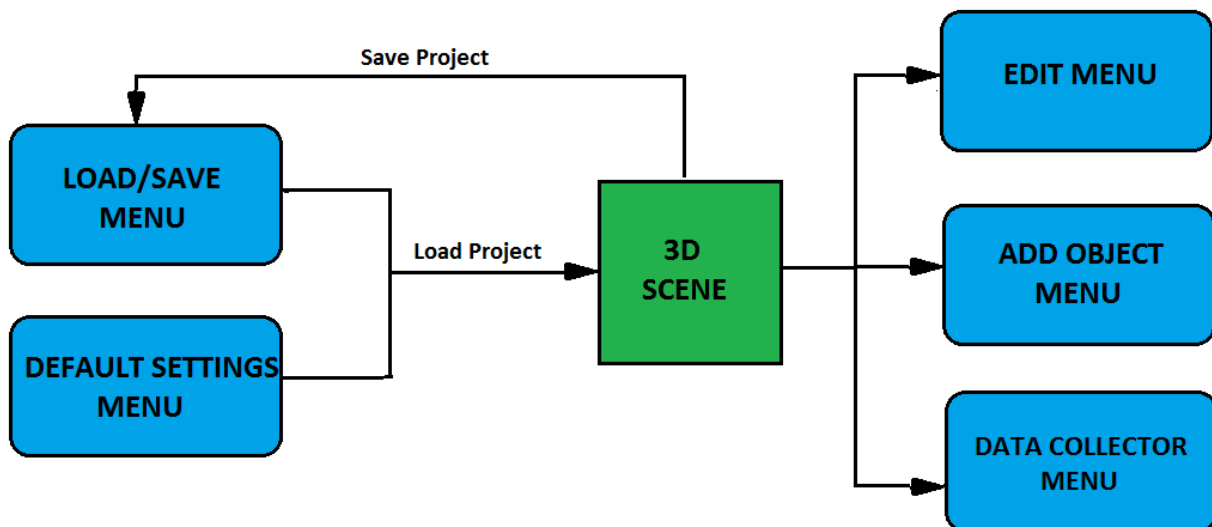


Figure 5.1- User Interface Component Diagram

5.1. LOAD/SAVE MENU

Load/Save Menu is responsible for loading the Urban Model into the scene. The menu button is in the upper left corner of the screen. When clicked, the menu panel is opened as shown in figure 5.2. Using this menu, users can open a new project, load existing projects or save current projects.

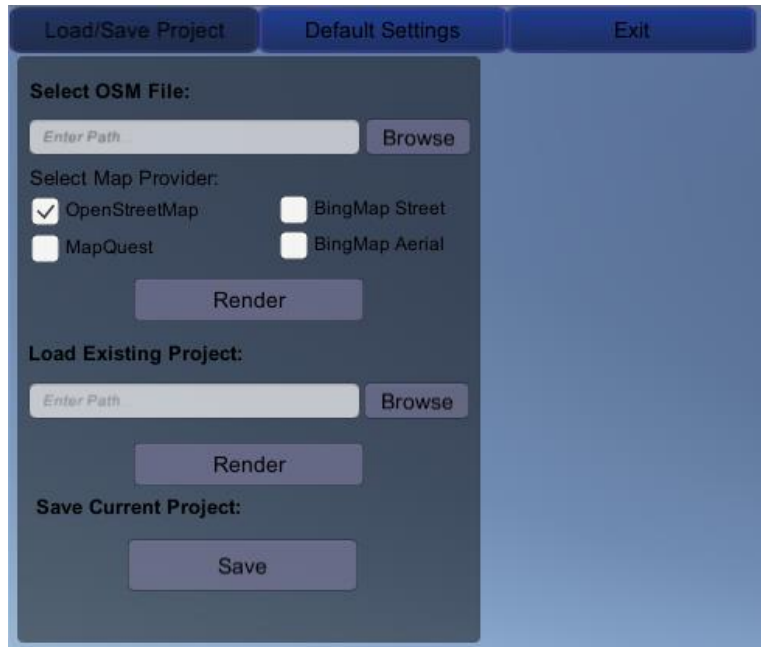


Figure 5.2- Load/Save Menu

5.1.1. Create New Project

In order to create a new project, the user should provide an OSM file. There is a “Browse” button on the menu panel and when clicked a File Browser Dialog appears (Figure 5.3a). Users are allowed to select .osm extension files from their local drives.

After selecting an .osm file path, the user should choose one of the listed Map Providers (Figure 5.2). As explained in Section 3.2, map providers are used to download tile images for texturing the Terrain surface.

When the “Render” button is clicked, the OSM file path and the Map Provider selections are sent to the Urban Generator component and the scene is created.

5.1.2. Load/Save Project

5.1.2.1. Saving Current Project

Users are able to do various edits via the user interface therefore a save point is a must to keep changes and continue working on the same project. The save point for our framework is simply designed as an xml file and contains the following data:

- Original OSM file
- Edits made for Buildings
- Edits made for Highways & Sidewalks
- Edits made for Barriers
- Added/Deleted 3D models and positions

Load/Save menu panel contains a “Save” button in order to save the current project. When it is clicked, a File Browser Dialog appears. Users can select the saving path and type a name for their save file (Figure 5.3b).

5.1.2.2. Loading Existing Project

From the Load/Save menu, users can load existing projects. When the “Browse” button is clicked under the load project section, a File Browser Dialog appears similar to the create project operation and when the “Render” button is clicked, the saved project is created by the Urban Generator component.

Note that, files are saved locally, which means that saved file can only be loaded if it is on the same machine. The reason for this is that, added external texture images, and 3D models are not placed in the save file. Instead, the save file only contains the path of external files.

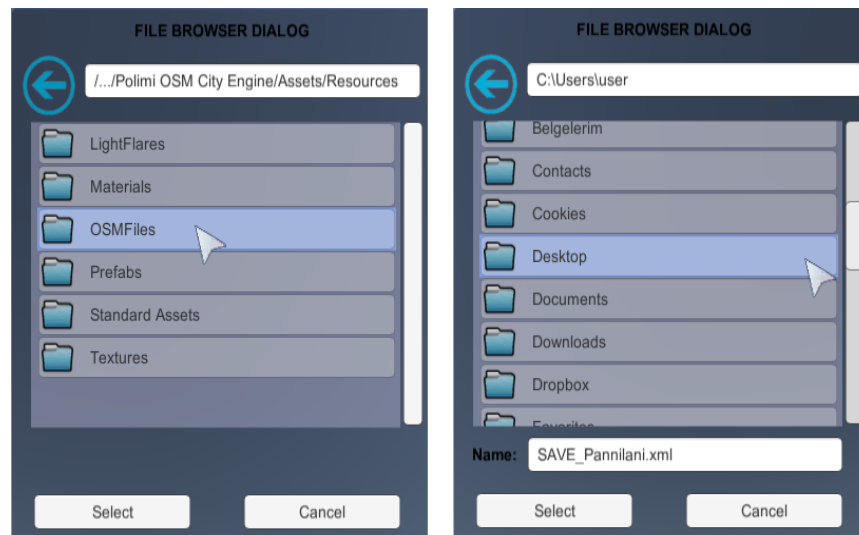


Figure 5.3 - File Browser Dialog (a) Select File Mode (b) Save File Mode

5.2. DEFAULT SETTINGS MENU

The OSM xml file contains plenty of specifications inside and normally, does not need additional parameters. However, OpenStreetMap contributors do not usually completely fill all the specification fields of the elements. For example, there are existing tags for highways defining way width, sidewalk information, and even tags for texture types exist but in our test scene (Como City) there is no data available for sidewalk among +1000 defined streets and only a few of them have a texture defined.

It is necessary to make assumptions for the missing values of items. For this reason, a set of parameters are accepted as default settings and stored in an xml file. Users can access the settings and change them via the settings menu. The “Default Settings” menu button is in the upper left corner of the screen. When clicked, a drop down list appears containing building, highway, barrier and skybox settings (Figure 5.4).



Figure 5.4 - Default Settings Menu

5.2.1. Default Building Settings

Building settings are available from “Building Settings” inside Default settings and are responsible for 2 features:

- Assigning a building height
- Assigning a facade skin

At the top of the menu panel, users can enter an interval of height values (Figure 5.5). When a new scene is created, building heights are randomly assigned between these intervals. Buildings which have a defined height value are not affected by this setting.

At the middle of menu panel, default building skins are shown in a scrollrect (Figure 5.5). Each building skin contains:

- Skin Name
- Color Texture
- Normal Texture
- Specular Texture (Shininess)

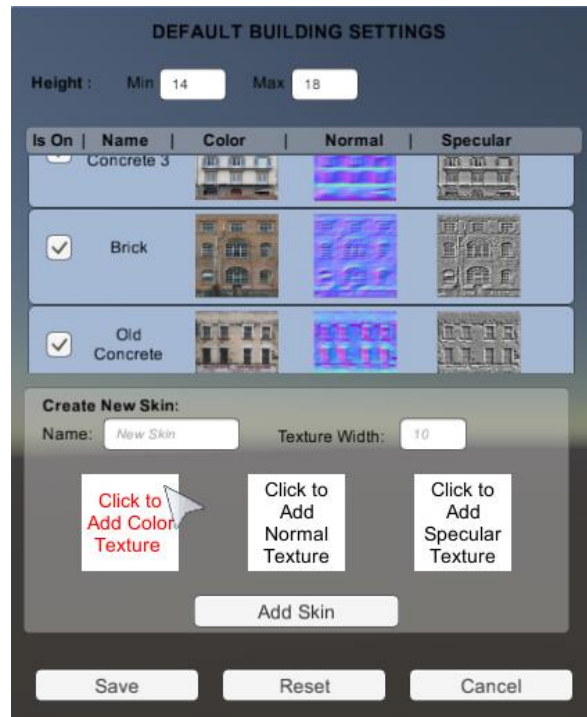


Figure 5.5 - Default Building Settings

As a default, 25 skins are loaded into the building settings. Users can activate or deactivate the skins by clicking the “Is On” box to the left of each item. When a new scene is created, the framework assigns those active skins randomly to buildings.

Users can also create new building skins from the bottom of menu panel. When the boxes are clicked, a file browser dialog appears to ask users for an image file allowing “.png”, “.jpg”, “.bmp” and “.tif” extensions. Texture width determines the size in meters. If the wall is wide, textures are repeated over the wall instead of stretching. When the “Add Skin” button is clicked, a new skin is added at the end of the skin list.

Changes will not be applied until the user clicks on the “Save” button at the bottom of the menu panel. If the “Reset” button is clicked, all changes made are reverted and the framework’s initial settings will be loaded again. The “Cancel” button can be used to exit the menu without saving changes.

5.2. DEFAULT SETTINGS MENU

5.2.2. Default Barrier Settings

Barrier settings are available from “Barrier Settings” inside the Default settings and they are responsible for 2 features:

- Assigning Barrier Size
- Assigning Barrier Skin

Supported barrier types are listed on the menu panel inside a scrollrect (Figure 5.6). Each barrier item has:

- Texture
- Name
- Height
- Thickness

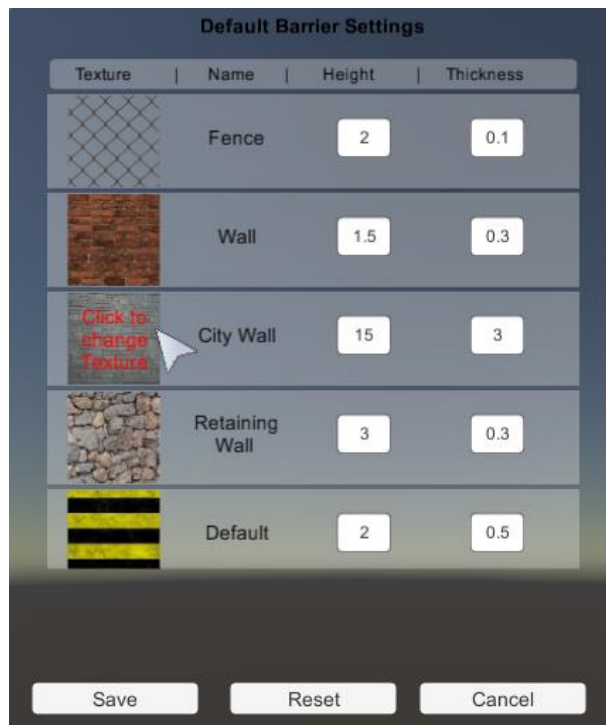


Figure 5.6 - Default Barrier Settings

Users can change the height and thickness of barriers using input fields for each element. In addition, barrier textures can be changed by clicking the texture image. When users hover the mouse into texture image, a text appears saying “Click to Change Texture” on the image. After clicking, a file browser dialog opens and asks for a path of image file accepting “.png”, “.jpg”, “.bmp” and “.tif” extensions.

5.2.3. Default Highway Settings

Highway settings are available from “Highway Settings” inside the Default settings. Supported highway types are listed in the menu panel inside a scrollrect (Figure 5.7). Each highway item has:

- Texture
- Way size
- Left/Right sidewalk activations
- Left/Right sidewalk sizes

Users can change the default texture of highways by clicking the texture image. After clicking, a file browser dialog opens and asks for a path of image file accepting “.png”, “.jpg”, “.bmp” and “.tif” extensions.

The way size can be changed from the input field under size and uses meters as unit. In addition, users can add sidewalks to the sides of highways by default. Left and Right sides are determined by the order of vertices in the OSM xml data. With the last column, sidewalk widths can be determined.

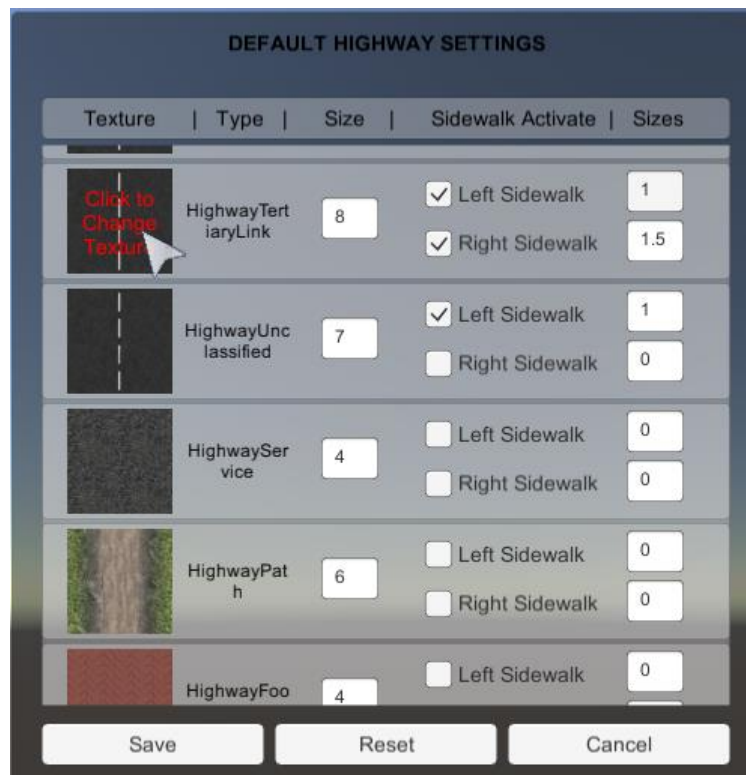


Figure 5.7 - Default Highway Settings

5.2. DEFAULT SETTINGS MENU

5.2.4. Skybox Settings

Skybox settings are available from “Skybox Settings” inside the Default settings. As part of the specifications, users can decide the time of the day by using the scrollbar on the menu panel (Figure 5.8).



Figure 5.8 - Default Skybox Settings

Scrollbar intervals are from 8:00 am to 8:00 pm. Skybox has an active sun and in accordance with the original, the sun rises in the east and sets in the west. Different times of day are shown in Figure 5.9.

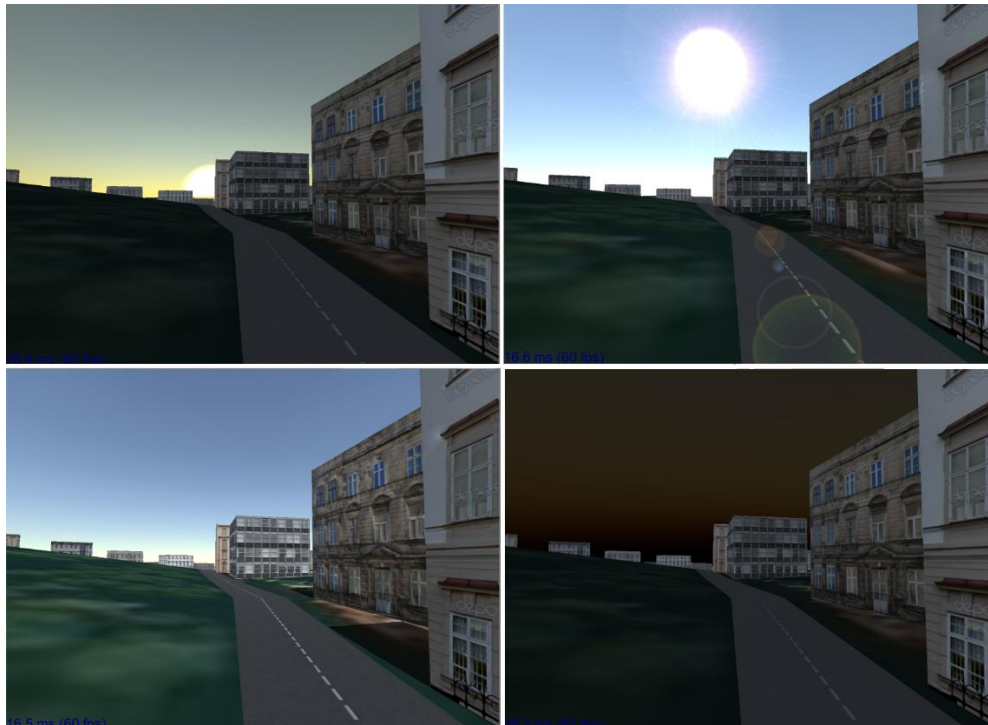


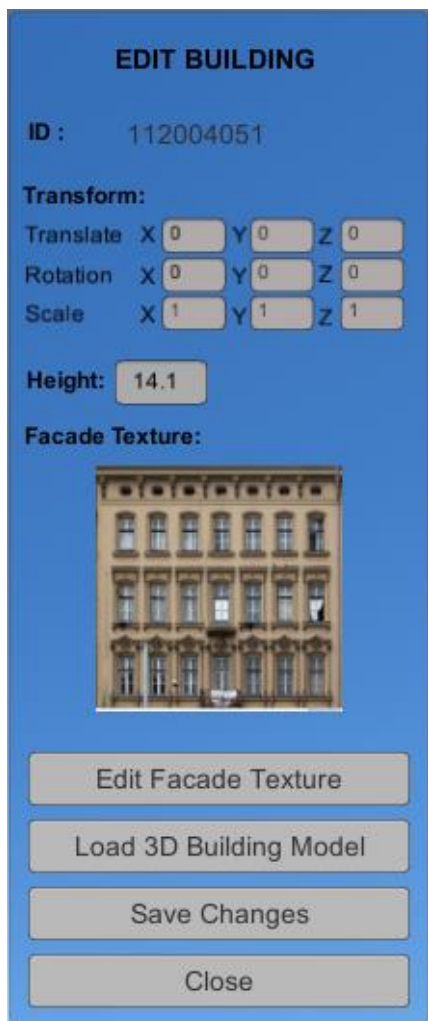
Figure 5.9 - Different day times in the scene

5.3. EDIT MENU

The default settings menu allows users to change many item features in the scene. Changes made with default settings are global and affect all items in that category. Where a specific change is needed, edit menus can be used. Edit menus have similar capabilities to default settings, except the changes can be made after creating a scene and they are item specific.

Edit menus are accessible when an item in the scene is selected using the left mouse button. When selected, the item's color changes into blue and the menu appears at the right-hand side of the screen. Items can be deselected by an empty click, relicking or clicking another item.

5.3.1. Building Edit



When a building facade in the scene is selected, the building edit menu appears containing the selected building's information (Figure 5.10). From the edit menu following operations can be done:

- Edit height of building
- Edit texture of selected facade
- Replace generated building with a 3D model

When the building height is changed, the mesh of each building facade is updated with the new height and a new roof is rendered with the new height value.

Auto-generated building meshes are sometimes inadequate compared to the original building (e.g., churches, modern designed buildings). In this case users can replace the generated building with its 3D model. When "Load 3D Building Model" is clicked, a file browser dialog will appear and ask the user to select an object file. If a building is replaced with its 3D model, it gets 3D object category features and the existing building edit features are disabled.

Figure 5.10 - Edit Building Menu

5.3. EDIT MENU

There are plenty of initial default building skins and users are able to add more via default settings. However the aim of our framework is to generate models closest to the real world, therefore editing facade textures and loading the original texture is necessary.

The edit menu contains a button named “Edit Facade Texture” and when clicked, a facade texture edit menu appears mid screen (Figure 5.11).

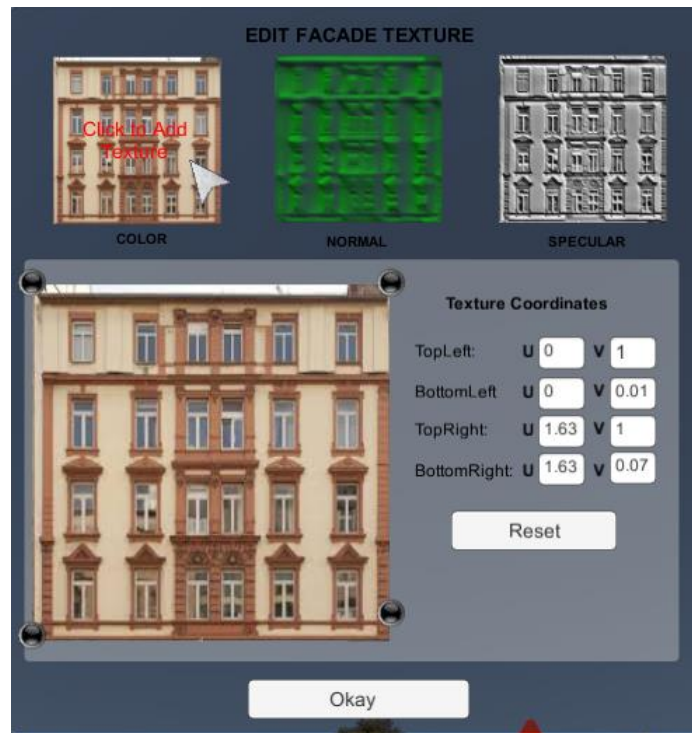


Figure.5.11 - Edit Facade Texture Menu

From the edit facade menu panel, users can upload color, normal and specular textures similar to default settings. In addition, users can crop the image using 4 black pins on the larger texture image. There are texture coordinates on all 4 corners on the right side of large image. When updating the black pins, numbers are changing automatically between 0 and 1. In case repeating texture is needed, users can update coordinate values manually by entering values >1.

5.3.2. Barrier Edit



Figure 5.12 - Barrier Edit Menu

When a barrier in the scene is selected, the barrier edit menu appears containing the selected barrier’s information (Figure 5.12). From the edit menu, the following operations can be done:

- Edit Barrier Size
- Edit Barrier Texture

Users can edit the thickness of the barrier by editing the “Thickness” input field. If the barrier type is fence, only the fence poles are affected by the edit.

Similarly, users can edit the height of the barrier by editing the “Height” input field. If the barrier type is fence, both the fence plane and the poles are affected from the edit.

By clicking the image in the middle of the menu panel, users can edit the barrier texture. When users hover the mouse into the image, a text appears saying “Click to Change Texture”. After clicking, a file browser dialog opens and asks for a path of image file accepting “.png”, “.jpg”, “.bmp” and “.tif” extensions.

5.3. EDIT MENU

5.3.3. Highway Edit



The image shows a blue-themed 'EDIT HIGHWAY' menu. At the top, it says 'EDIT HIGHWAY'. Below that, there are several fields: 'ID: 193968923i1', 'Type: HighwaySecondary', 'Street Name: Via Statale per Lecco (detta Cappelletta)', and 'Way Size: 10'. Under the 'Pavament:' section, there are two options: 'Left Sidewalk' which is checked and has a width of 1.5, and 'Right Sidewalk' which is unchecked and has a width of 0. At the bottom, there are two buttons: 'Save Changes' and 'Close'.

Figure 5.13 - Highway Edit Menu

When a highway in the scene is selected, the highway edit menu appears containing the selected highway's information (Figure 5.13). From the edit menu, the following operations can be done:

- Editing highway width
- Enable/Disable sidewalk

Highways are not independent items and when there is a change in one highway, multiple highways and 3D objects are affected.

In order to edit highway width, the following changes are made in the scene:

- Way intersections are regenerated
- Left/Right sidewalks are regenerated
- Sidewalk intersections are updated
- Street Lamps and Traffic Light positions are updated

Under the pavement section in the highway edit menu, sidewalks can be enabled/disabled and the sidewalk width can be updated. In order to edit sidewalks, the following changes are made:

- Related sidewalk is regenerated
- Related sidewalk intersections are recalculated

5.3.4. 3D Object Edit



When an object in the scene is selected, the object edit menu appears containing the selected object's information (Figure 5.14).

From the object edit menu, users are able to change the position, rotation and scale of objects. Besides this, the same operations can be done via mouse using Gizmos (Figure 5.15). Gizmo is a tool for transforming an object. It consists of arrows or circles representing the x, y and z axis. By dragging arrows or circles objects can be translated, rotated and scaled.

When an object is selected, gizmo is activated and appears at the center of the selected object. By default, Translate gizmo is activated and using the number keys gizmo mode can be changed:

- 1: Translate Gizmo
- 2: Rotate Gizmo
- 3: Scale Gizmo

Figure 5.14 - 3D Object Edit Menu

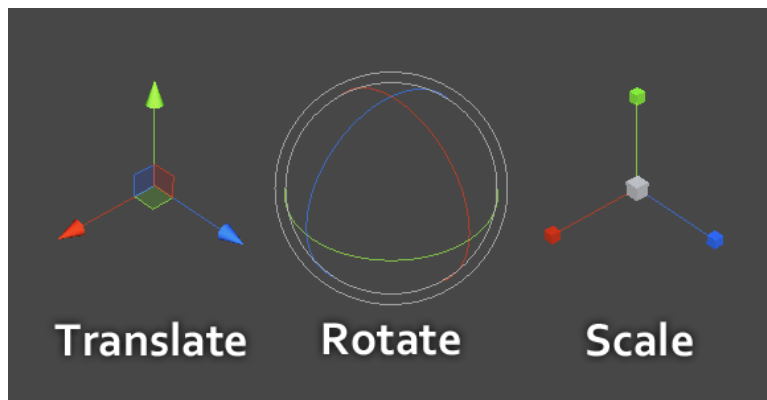


Figure 5.15 - Transform Gizmos

5.4. ADD OBJECT MENU

5.4. ADD OBJECT MENU

OSM xml defines dozens of different object types and several of them are supported in our framework. Object variety in the scene increase the complexity and therefore improves quality. For this reason, “Add Object Menu” is implemented to support objects not defined in the xml file.

Add Object Menu was placed at the bottom center of the screen and consist of 7 sections (Figure 5.16):

- Add Trekker
- Add Camera Van
- Add Vehicle
- Add Environment Object
- Add City Related Object
- Add Wall
- Load External Object



Figure 5.16 - Add Object Menu

Add Trekker button puts a human character into the scene and Camera Van puts a van into the scene which is controllable by arrow keys. They are both used to collect data from the scene as covered in Chapter 4.

In the scene, Trekker and Camera Van cannot be found at the same time. Similarly, the Add Object Menu does not allow users to add more than one Trekker or Camera Van. These two data collectors can be removed and added again using the camera menu.

Supported objects in the framework are categorized into 4 sections which are Vehicle, Environment, City Related and Wall. Each category has a button in the Add Object Menu and when clicked, they activate a menu that contains a list of objects (Figure 5.17).

In the object list, each object is represented by its icon and name. By selecting an item from the list, the corresponding object appears in front of the camera and it can be transformed to the desired place using the object edit gizmos covered in Section 5.3.4.



Figure 5.17 - Environment Object List Menu

Vehicle section contains:

- Bus
- Station Wagon Car
- Police Car
- Taxi
- Van

Environment section contains:

- Broad Leaf Tree 1
- Broad Leaf Tree 2
- Conifer Tree
- Palm Tree
- Broad Leaf Tree 3
- Fountain 1
- Fountain 2
- Sculpture 1
- Sculpture 2
- Sculpture 3

City Related section contains:

- Traffic Light
- Double Sided Traffic Light
- Traffic Light with Longer Pole
- Street Lamp
- Phone Box
- Garden Chair
- Hydrant

Wall Section contains:

- Metal Fence
- Concrete Wall Block

5.5. DATA COLLECTOR MENU

In addition to predefined objects, users can add their own desired objects using the “New” button on the right of Add Object Menu bar. For this component, an external object loader library [30] was used to load .OBJ⁶ file at runtime. When the “New” button is clicked, a file browser dialog appears requesting the object path. After the path is selected, the framework search for material file (.MTL⁷) in the specified path and generates the 3D object.

5.5. DATA COLLECTOR MENU

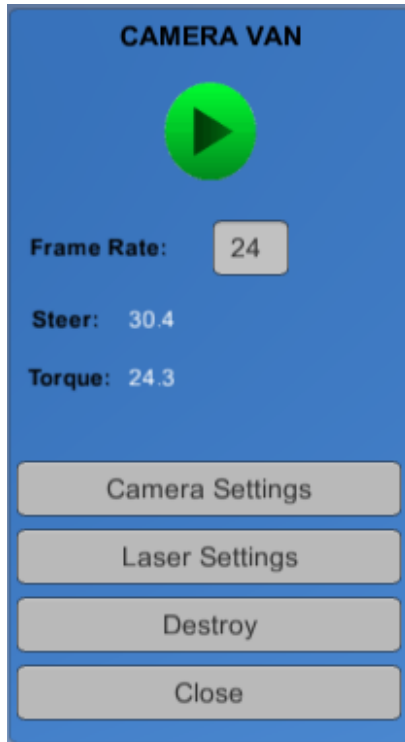


Figure.5.18 - Data Collector Menu

Data collector menu is responsible for the data-collecting component of the framework. This menu appears on the right-hand side of the screen when there is a data collector (camera van or Trekker) in the scene. With the menu, users can:

- Start/Stop data recording
- Access Camera Settings
- Access Laser Settings

Using the play button at the top of menu panel (Figure 5.18), recording process can be initiated and pressing the same button starts generating output data as covered in Chapter 4.

Frame rate of the recording cameras can be set directly from the data collector menu. By default, the camera frame rate is 24 and all cameras defined in the camera settings should have the same value.

Steering of the front wheels and the velocity of the camera van are shown in the menu during recording if camera van is selected as a data collector, otherwise these fields remain empty.

5.5.1. Camera Settings

The Camera Settings Menu is available under the “Camera Settings” button in the “Data Collector Menu” (Figure 5.18). Camera settings contain information of all camera instances belongs to data collector.

⁶ Wavefront OBJ File, is a geometry definition file format developed by Wavefront Technologies

⁷ MTL : Material File (Wavefront), Describes the material properties (e.g., Textures, Colors) of the objects

From camera settings, users can create multiple camera instances using the “Add Camera” button on the bottom of the menu panel (Figure 5.19). When new camera instances are created, they are added to the scrollrect in the middle of the menu.

The screenshot shows a 'CAMERA SETTINGS' menu with three camera instances. Each instance has a 'Camera ID' (1, 2, and 3), a status checkbox (all checked), and input fields for Pitch, Yaw, Roll, FOV, and Position (wrt Center). The Position fields are arranged in a 2x2 grid with values 0, 2, and 0.

Camera ID	Status	Pitch	Yaw	Roll	FOV	Position (wrt Center)
1	<input checked="" type="checkbox"/>	0	0	0	75	0, 2, 0
2	<input checked="" type="checkbox"/>	0	90	0	75	0, 2, 0
3	<input checked="" type="checkbox"/>	0	-90	0	75	0, 2, 0

Buttons at the bottom: Add Camera, Apply Changes, Close.

Figure 5.19 - Camera Settings Menu

Each camera instance has the following parameters:

- Camera ID
- Status (Active or Inactive)
- Field of View
- Pitch, Yaw and Roll
- Position with respect to Data Collector’s center

Each camera has an ID to distinguish their outputs. ID’s are given in increasing order starting from 1. Camera instances can be enabled or disabled using the tick box near the camera id in the menu. Pitch, Yaw and Roll values specifies the rotation of the camera and their range from -180 to 180 degree. Field of View (FOV) value is the view angle of the camera and range from 0 to 180 degree.

5.5. DATA COLLECTOR MENU

5.5.2. Laser Settings

The Laser Settings Menu is available under the “Laser Settings” button in the “Data Collector Menu” (Figure 5.18). Laser settings contain information of the laser scanner which belongs to the data collector.

From this setting, users can edit the following parameters of the Laser Scanner (Figure 5.20):

- Position with respect to the center of the controller
- Rotation (pitch, yaw and roll)
- Min Distance, Max Distance
- Field of View (Vertical)
- Field of View (Horizontal)
- Vertical Resolution
- Horizontal Resolution
- Frame Rate

Real-world laser scanners have effective distance ranges from ~0-5 meter to ~100-150 meters. Although our virtual laser can work for infinitive distances, it was restricted to 200 meters.

LASER SCANNER SETTINGS

Is Active:

Position:

Rotation:

Minimum Distance:

Maximum Distance:

Field of View (Vertical):

Field of View (Horizontal):

Angular Resolution (Azimuth):

Vertical Resolution:

Frame Rate:

Okay

Figure 5.20 - Laser Scanner Settings Menu

Field of View (FOV) is defined both vertically and horizontally. These are angle values and horizontal FOV ranges from 0 to 360 degrees while vertical FOV ranges from 0 to 180 degrees. In order to determine the density of the laser rays, vertical and horizontal resolutions are defined. These two resolution takes angle as parameter and they can be very small variables to increase point count.

By default, laser scanner is assumed to be placed in parallel with the roof of the camera van and front vector of the scanner is same as the van. The position of the laser scanner with respect to center of camera van can be edited from the menu.

Users can enable or disable the laser scanner component by ticking the “Is Active” toggle. If the scanner is not active, all of the input fields are disabled and the recording process does not generate point cloud data.

5.5. DATA COLLECTOR MENU

Chapter 6 : RESULTS AND EVALUATION

In this chapter, we present the results of our framework by generating a tiny output sample using our three component, which are presented in the previous chapters. Later, we compared the generated 3D scene with the real world then discussed about the overall quality. Finally, we evaluate the performance of the framework.

6.1. SAMPLE OUTPUT GENERATION

To generate the sample scene, junction of “via Pannilani” and “via Provinciale per Lecco” in Como city was selected. After downloading the OSM file described in Appendix A, a new project was created.

Initially, the scene was not containing any 3D objects except the traffic signals. After a couple minute of editing process, we added 4 vehicles, 10 trees and 9 metal fence barriers then applied necessary transformations to each of them.

When editing process was finished, a camera van was placed into the scene with 3 cameras and a LiDAR on top of it. To generate a small dataset, a record was taken around 10 seconds using the path shown in Figure 6.1.

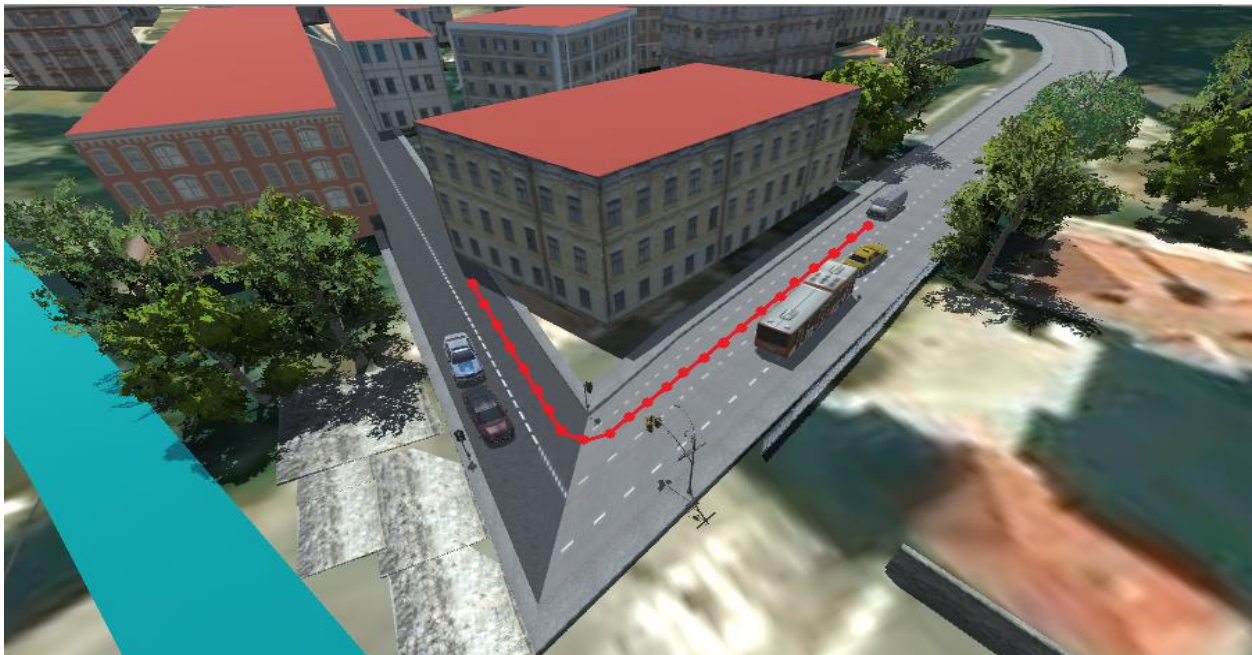


Figure 6.1 – Sample Recording Path

6.1. SAMPLE OUTPUT GENERATION

We oriented the 3 cameras and LiDAR as follows:

- **Camera 1:** FOV = 90, Pitch = 0, Yaw = 0, Roll = 0, PosX = 0, PosY = 2, PosZ = 0
- **Camera 2:** FOV = 90, Pitch = 0, Yaw = 90, Roll = 0, PosX = 0, PosY = 2, PosZ = 0
- **Camera 3:** FOV = 90, Pitch = 0, Yaw = -90, Roll = 0, PosX = 0, PosY = 2, PosZ = 0
- **LiDAR:** Range: 1-120, FOV_{VERTICAL} = 30, FOV_{HORIZONTAL} = 360, Resolution_{VERTICAL} = 7.5, Resolution_{HORIZONTAL} = 45, Pitch = 0, Yaw = 0, Roll = 0, PosX = 0, PosY = 2, PosZ = 0

After the framework finished processing, it generated the following files:

- Log File (Figure 6.2)
- Set of Images for each camera (Figure 6.3, 6.4 and 6.5)
- Set of PCD files (Figure 6.6)

```
<?xml version="1.0" encoding="utf-8"?>
<RecordLog xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <CameraSettings>
    <Cam CameraID="1" Pitch="0" Yaw="0" Roll="0" Position="(0.0, 2.0, 0.0)" FOV="90" />
    <Cam CameraID="2" Pitch="0" Yaw="90" Roll="0" Position="(0.0, 2.0, 0.0)" FOV="90" />
    <Cam CameraID="3" Pitch="0" Yaw="-90" Roll="0" Position="(0.0, 2.0, 0.0)" FOV="90" />
  </CameraSettings>
  <LaserSettings>
    <Laser MinDist="1" MaxDist="120" VerticalFOV="30" HorizontalFOV="360" VerticalRes="7.5" HorizontalRes="45"
      Position="(0.0, 2.0, 0.0)" Pitch="0" Yaw="0" Roll="0" />
  </LaserSettings>
  <CameraLog FrameRate="2">
    <CameraEntries>
      <Entry id="0" Rotation="(4.2, 335.4, 0.2)" Position="(-5.7, 234.6, 117.8)" Steer="0" Velocity="13.6000414" />
      <Entry id="1" Rotation="(4.1, 335.1, 0.2)" Position="(-6.7, 234.4, 119.9)" Steer="0" Velocity="19.608614" />
      <Entry id="2" Rotation="(4.1, 334.7, 0.1)" Position="(-8.0, 234.2, 122.7)" Steer="0" Velocity="24.9719429" />
      <Entry id="3" Rotation="(4.1, 334.2, 0.1)" Position="(-9.6, 233.9, 126.1)" Steer="0" Velocity="29.2054634" />
      <Entry id="4" Rotation="(4.3, 333.9, 0.0)" Position="(-11.5, 233.6, 129.9)" Steer="0" Velocity="30.94313" />
      <Entry id="5" Rotation="(4.3, 333.9, 0.0)" Position="(-13.3, 233.3, 133.6)" Steer="0" Velocity="29.0376511" />
      <Entry id="6" Rotation="(4.3, 333.9, 0.0)" Position="(-15.0, 233.0, 137.2)" Steer="0" Velocity="28.2180767" />
      <Entry id="7" Rotation="(4.3, 333.7, 0.0)" Position="(-16.8, 232.7, 140.8)" Steer="0" Velocity="28.8433342" />
      <Entry id="8" Rotation="(4.2, 333.8, 0.0)" Position="(-18.5, 232.4, 144.3)" Steer="0" Velocity="27.4771671" />
      <Entry id="9" Rotation="(3.7, 333.8, 359.6)" Position="(-20.2, 232.1, 147.6)" Steer="0" Velocity="25.9326077" />
      <Entry id="10" Rotation="(2.8, 333.9, 359.3)" Position="(-21.7, 231.9, 150.7)" Steer="0" Velocity="24.5053883" />
      <Entry id="11" Rotation="(2.9, 333.9, 359.3)" Position="(-23.1, 231.8, 153.6)" Steer="0" Velocity="21.7635269" />
      <Entry id="12" Rotation="(3.1, 342.6, 0.5)" Position="(-24.0, 231.6, 156.2)" Steer="45" Velocity="18.35962" />
      <Entry id="13" Rotation="(2.9, 8.5, 1.9)" Position="(-23.7, 231.5, 158.4)" Steer="45" Velocity="15.567194" />
      <Entry id="14" Rotation="(2.1, 37.7, 3.2)" Position="(-22.2, 231.4, 160.1)" Steer="45" Velocity="16.6218414" />
      <Entry id="15" Rotation="(0.4, 69.6, 4.0)" Position="(-19.8, 231.4, 161.0)" Steer="21.8038788" Velocity="20.8669052" />
      <Entry id="16" Rotation="(359.4, 76.5, 3.6)" Position="(-16.6, 231.4, 161.8)" Steer="16.8737259" Velocity="26.5610638" />
      <Entry id="17" Rotation="(359.4, 87.5, 3.5)" Position="(-12.7, 231.4, 162.2)" Steer="0" Velocity="29.0094185" />
      <Entry id="18" Rotation="(358.8, 81.6, 3.7)" Position="(-8.5, 231.5, 162.6)" Steer="0" Velocity="31.3261242" />
      <Entry id="19" Rotation="(358.9, 80.8, 4.3)" Position="(-4.5, 231.6, 163.3)" Steer="0" Velocity="27.8984833" />
      <Entry id="20" Rotation="(358.9, 80.7, 4.3)" Position="(-0.8, 231.6, 163.9)" Steer="0" Velocity="25.5085144" />
    </CameraEntries>
  </CameraLog>
  <LaserLog FrameRate="1">
    <LaserEntries>
      <Entry id="0" Rotation="(4.2, 335.4, 0.2)" Position="(-5.7, 234.6, 117.8)" Steer="0" Velocity="13.6000414" />
      <Entry id="1" Rotation="(4.1, 334.7, 0.1)" Position="(-8.0, 234.2, 122.7)" Steer="0" Velocity="24.9719429" />
      <Entry id="2" Rotation="(4.3, 333.9, 0.0)" Position="(-11.5, 233.6, 129.9)" Steer="0" Velocity="30.94313" />
      <Entry id="3" Rotation="(4.3, 333.9, 0.0)" Position="(-15.0, 233.0, 137.2)" Steer="0" Velocity="28.2180767" />
      <Entry id="4" Rotation="(4.2, 333.8, 0.0)" Position="(-18.5, 232.4, 144.3)" Steer="0" Velocity="27.4771671" />
      <Entry id="5" Rotation="(2.8, 333.9, 359.3)" Position="(-21.7, 231.9, 150.7)" Steer="0" Velocity="24.5053883" />
      <Entry id="6" Rotation="(3.1, 342.6, 0.5)" Position="(-24.0, 231.6, 156.2)" Steer="45" Velocity="18.35962" />
      <Entry id="7" Rotation="(2.1, 37.7, 3.2)" Position="(-22.2, 231.4, 160.1)" Steer="45" Velocity="16.6218414" />
      <Entry id="8" Rotation="(359.4, 76.5, 3.6)" Position="(-16.6, 231.4, 161.8)" Steer="16.8737259" Velocity="26.5610638" />
      <Entry id="9" Rotation="(358.8, 81.6, 3.7)" Position="(-8.5, 231.5, 162.6)" Steer="0" Velocity="31.3261242" />
      <Entry id="10" Rotation="(358.9, 80.7, 4.3)" Position="(-0.8, 231.6, 163.9)" Steer="0" Velocity="25.5085144" />
    </LaserEntries>
  </LaserLog>
</RecordLog>
```

Figure 6.2 - Sample Log File

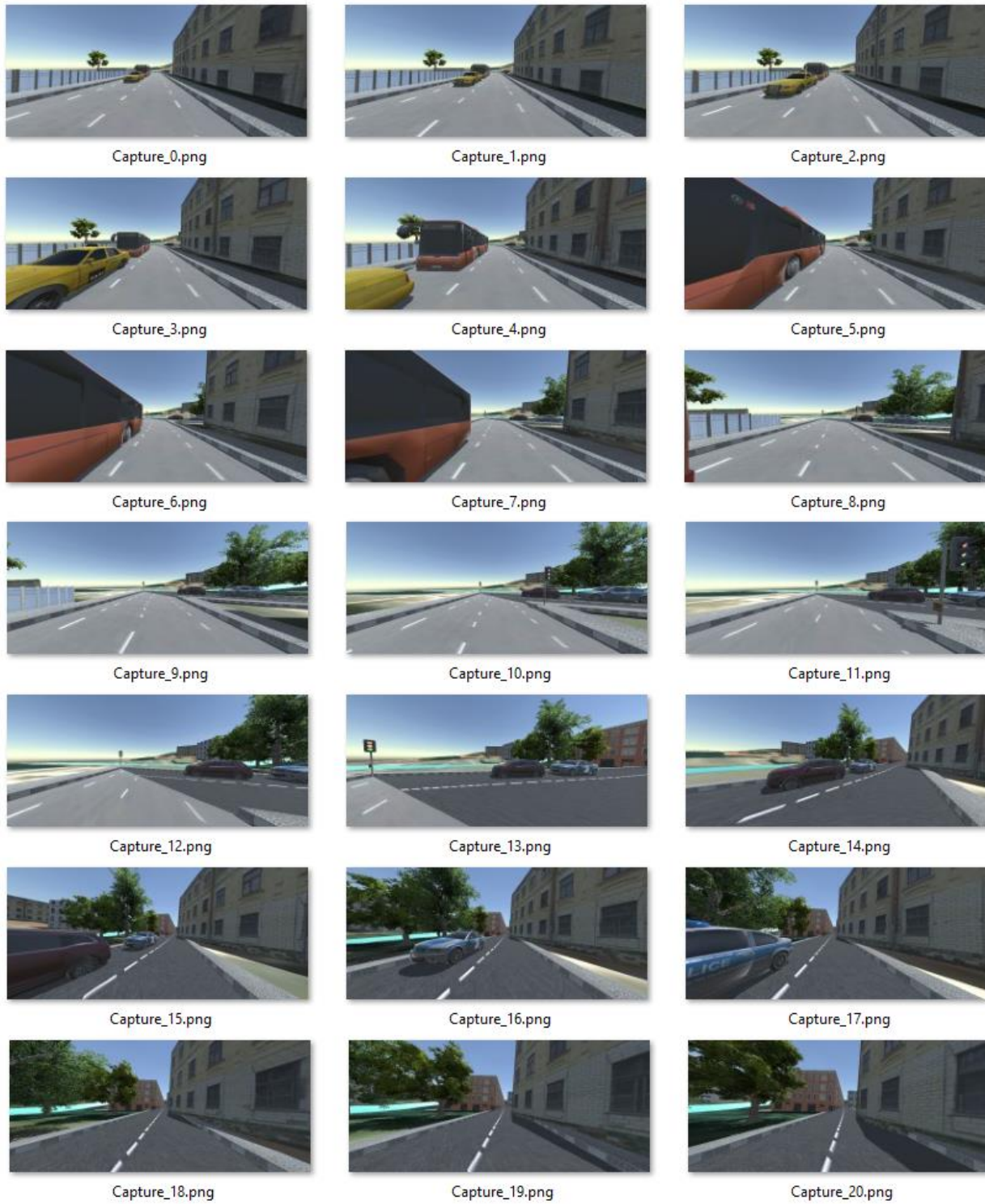


Figure 6.3 - Sample Output Frames, Front Camera (Cam 1)

6.1. SAMPLE OUTPUT GENERATION

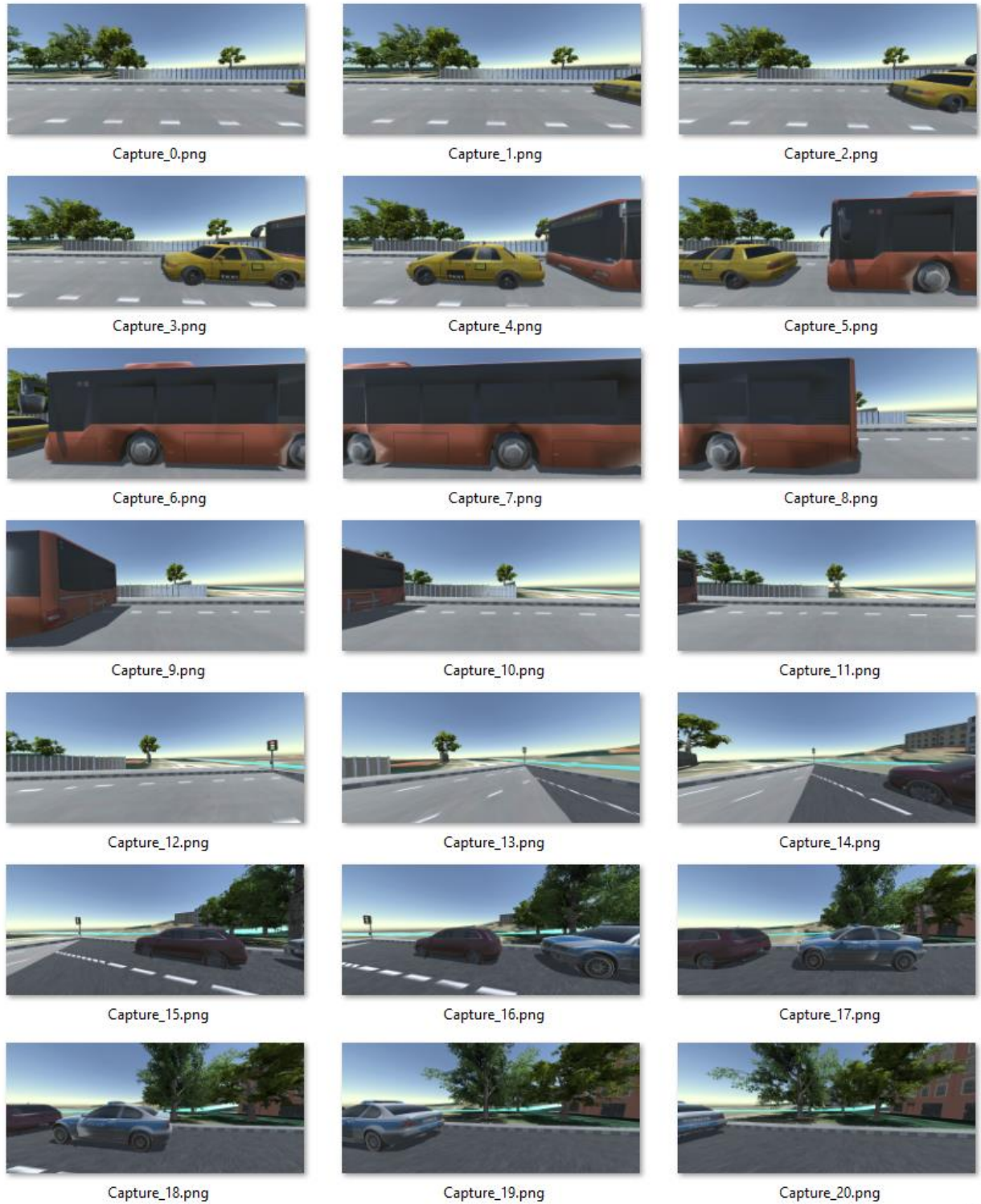


Figure 6.4 - Sample Output Frames, Left Camera (Cam 2)



Figure 6.5 - Sample Output Frames, Right Camera (Cam 3)

6.2. QUALITY EVALUATION

```
#Polimi OSM City Engine PCD
VERSION .7
FIELDS x y z distance
SIZE 4 4 4 4
TYPE F F F F
COUNT 1 1 1 1
WIDTH 8
HEIGHT 4
VIEWPOINT 0 0 0 1 0 0 0
POINTS 32
DATA ascii
0 0 0 0
0 0 0 0
0 0 0 0
-36.93655 251.26143 23.542 52.39
-34.91631 250.27442 12.041 54.48
-40.98111 250.24247 24.126 47.85
-36.93655 251.26143 23.542 60.32
-34.91631 250.27442 12.041 46.34
0 0 0 0
0 0 0 0
-34.11044 246.32222 18.841 38.12
0 0 0 0
-39.81921 245.29315 25.432 23.61
-24.90701 246.24109 23.934 26.47
-39.85082 245.71212 20.297 30.42
-35.11044 246.32222 18.841 24.84
0 0 0 0
0 0 0 0
...
```

Figure 6.6 - Sample PCD File (Frame 1)

6.2. QUALITY EVALUATION

The final quality of the output data set is strictly dependent to the quality of the urban environment. Auto-generated 3D scene has lack of detail for the generated scene so, it drops the overall quality. However, edit options are used for enhancement so that, if a significant time is spent on editing the urban environment, a good result can be obtained. Therefore, it is important for users to have external object support for their need.

In Figure 6.7, two images are shown which are taken from our test location in Section 6.1 where the first one is our generated scene and the second one is the real-world sample. At first glance, a couple of difference can be noticed.

The main difference of the two scene is the lack of background in our generated scene. While there is a hill on the left side of the original image, our scene does not have any. The reason for that is, using a small-size scene which causes framework to crop the terrain with the provided bounding box. That's why, the hill which is far from the scene is not rendered.

Second difference is the location and shape of the buildings. OSM xml data is created by thousands of contributors mostly using the aerial images by Google or Bing. Users define a building by marking surrounds of building roofs in the aerial image. That leads buildings to have lower quality (e.g. lack of balconies) and wrong positioning. Although the difference is tolerable with texture edits on the facades, mispositioning can be significant especially on tall buildings and 3D model replacement supported by our framework is suggested for better result.

Third difference is the terrain elevation. As stated in Section 3.1, NASA SRTM data is used as height map which has low resolution. Therefore, small height changes can not be represented on our terrain.

The remaining differences are related to objects rather than buildings and highways in the scene and they can be altered with importing high-quality external object models using our Object menu.



Figure 6.7- “Via Provinciale per Lecco”, (a) Generated Scene (b) Original, Google Street View

6.3 PERFORMANCE EVALUATION

6.3 PERFORMANCE EVALUATION

In this section, we tested the performance of our two components using the following computing system:

- Processor: Intel i5, 2.27 GHz
- Graphic Card: ATI Radeon HD 5000 Series, 2GB
- Memory: 4GB
- Storage: HDD 5400rpm
- Internet Connection : Polimi Wi-Fi (1 Mbps)

6.3.1. Urban Generator Performance

3D urban generation is a time-taking process and there are many factors that affects the generation time. For evaluation, we generated three project with different sizes around Como City, Italy which are Pannilani Street, Brunate and Como City Center (Figure 6.8).

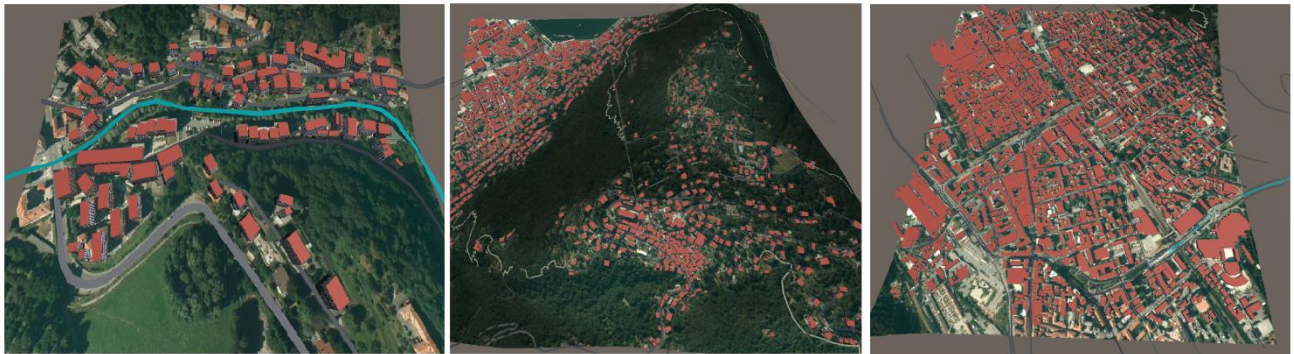


Figure 6.8 - Test Scenes (a) Pannilani (b) Brunate (c) Como City Center

For performance comparison, we defined a set of parameters that has the most effect on rendering:

- Terrain Size
- Building Count
- Highway Count
- Barrier + 3D Object Count

According to the parameters, the values of the test scenes are given in Table 6.1:

Scene Name	Terrain Size	Building Count	Highway Count	Barrier + 3D OBJ Count
Pannilani	0.32 km ²	96	10	2
Brunate	4.75 km ²	878	294	20
Como Center	3.57 km ²	1136	430	75

Table 6.1 – Test Scene Parameters

TERRAIN GENERATION

Rendering Terrain is taking significant amount of time and without the cache file, it is the bottleneck of the generation process. In order to render the terrain, framework needs to download a height map file from the NASA servers and the tile textures from the chosen map provider.

Height map file is a ZIP file which the extracted version has 2,818 KB constant size and the compressed size ranges from 1 to 1.4 MB. Downloading and unzipping the file is taking 5 seconds on average with 1Mbps internet connection. Once the height map is downloaded, it is cached and framework don't download the same file again.

Depends on the terrain size, a set of map tiles should be downloaded from the provider. The tile images are 256x256 jpeg or png images and their sizes range from 1 to 30 KB. Downloading speed of these tiles varies depends on the provider and their current request load. On Average, framework downloads 3 tiles per second (ignoring request time-outs) with 1Mbps internet connection. Once the tiles are downloaded, they are cached and framework don't download the same tile again.

After the height map and tiles are obtained, cropping operations made on the tile images and terrain is generated. For our test scenes with cache, rendering times are as follows:

Scene Name	Terrain Size	Render Time
Pannilani	4x8 Grid	1.1 Second
Brunate	25x19 Grid	7.2 Second
Como Center	21x17 Grid	6.3 Second

Table 6.2 - Terrain Rendering Times

OSM XML PARSE

OSM XML parsing comes after the terrain component. When the OSM file is provided, the framework requires to parse the xml data, enumerate the items and fill the structure lists. For our test scenes, OSM file sizes and their parsing time are as follows:

Scene Name	OSM File Size	Parsing Time
Pannilani	312 KB	0.45 Second
Brunate	2123 KB	10.9 Second
Como Center	2865 KB	26.1 Second

Table 6.3- OSM File Parsing Times

As seen, Parsing time and OSM file size is not linear because of the structure of the xml file. Node ID's are not sorted in the xml file and for each item, necessary nodes should be searched in the Node List starting from the beginning. For this reason, parsing time increase with the node count.

6.3 PERFORMANCE EVALUATION

BUILDING GENERATION

Building generation is a 2-step process which are the initializing building settings and rendering the buildings. In initialization step, building skins are generated by using the textures provided in the Default Building Settings. To generate a building skin, framework needs to read 3 image file which are color, normal and specular textures. By default we have 20 defined skins and generating those takes around 5 seconds.

Rendering building step is a fast process since the buildings are low-polygon structures. For our test scenes, rendering times are as follows:

Scene Name	Building Count	Render Time
Pannilani	96	5.2 Second
Brunate	878	5.9 Second
Como Center	1136	8.0 Second

Table 6.4 - Building Rendering Times

HIGHWAY GENERATION

Highway generation is a multi-step process which includes initialization, intersection processing, terrain draping, sidewalk generation etc. Although highways require the most amount of process in the framework, they are rendered fastest since they do not need any IO operation or Web Request. For our test scenes, rendering times are as follows:

Scene Name	Highway Count	Render Time
Pannilani	10	0.1 Second
Brunate	294	1.6 Second
Como Center	430	1.9 Second

Table 6.5 - Highway Rendering Times

BARRIER + 3D OBJECT GENERATION

Barriers and 3D Objects are combined since an average scene contains less amount of those. Similar to the highways, barriers and 3D objects do not need any IO operation or Web Request. 3D objects are loaded to memory at the beginning of the program by Unity Engine that's why, instantiating them takes no time. For our test scenes, rendering times are as follows:

Scene Name	Barrier + Object Count	Render Time
Pannilani	2	0.01 Second
Brunate	20	0.1 Second
Como Center	75	0.3 Second

Table 6.6 - Barrier + 3D Object Rendering Times

TOTAL RENDERING TIMES & FRAME RATES

After sub-components finish their process, the scene appears on the screen. Depends on the scene size and the number of vertices in the view plane, Frame Rate varies. If the camera look towards an object set, FPS value decreases dramatically depends on the object complexity. For this reason, we chose our default objects which have less than 2K vertices and for the trees, we applied LOD⁸ property.

We measured the FPS values of our test scenes by navigating around the scene for 1 minute each and then, we took the lowest, highest and the average FPS values shown in Table 6.7:

Scene Name	Total Render Time	Worst-Case FPS	Best-Case FPS	Average FPS
Pannilani	7.2 Second	49	62	58.7
Brunate	26.3 Second	23	48	39.4
Como Center	43.6 Second	19	43	33.1

Table 6.7 - Total Render Time and FPS values

6.3.2. Data Collector Performance

Generating outputs require IO operations on the storage therefore, it is a slow process and cannot be done during scene recording. For this reason, a log file is generated during record which is later used to generate the outputs.

To test the performance, we used the edited Pannilani Street as a test scene from section 6.1 with different controller configurations:

Scene Name	Camera Count	Camera FPS	LiDAR Beam Count	LiDAR FPS	Record Time
Pannilani – 1	1	2	-	-	10 Second
Pannilani – 2	1	5	-	-	10 Second
Pannilani – 3	3	5	-	-	10 Second
Pannilani – 4	3	5	-	-	30 Second
Pannilani – 5	-	-	160 (20x8)	2	10 Second
Pannilani – 6	-	-	160 (20x8)	5	10 Second
Pannilani – 7	-	-	14400 (360x40)	2	10 Second
Pannilani – 8	-	-	14400 (360x40)	2	30 Second

Table 6.8 – Test Cases for Data Controller

⁸ LOD: Level of Detail, Process of decreasing complexity of 3D object as camera moves away from the object.

6.3 PERFORMANCE EVALUATION

VIDEO FRAME GENERATION

Video Frames are generated by capturing screen at that frame therefore size of the image is the same as size of framework window. We tested our scene with 1366x597 window size so that our generated images are 1366x597 JPEG images.

In Table 6.8, upper 4 test scenes are chosen to test performance of the Camera Module. We changed one parameter at each time to check whether Camera Component works linear with the increasing data size. For the given parameters, video generation times are as follows:

Scene Name	Capture Count	Generation Time
Pannilani – 1	18	2.95 Second
Pannilani – 2	51	7.92 Second
Pannilani – 3	147 (49 x 3)	23.24 Second
Pannilani – 4	450 (150 x 3)	100.79 Second

Table 6.9 - Video Frame Generation Times

POINT CLOUD GENERATION

PCD Files are generated by sending laser beams from the LiDAR position which we imitated by Ray Casting. LiDAR position and rotation are not important to measure performance. That's why, we used the default positions and changed the resolutions to obtain different number of laser beams.

In Table 6.8, lower 4 test scenes are chosen to test performance of the LiDAR Module. Similar to Camera Module, we changed one parameter at each time to check whether LiDAR Component works linear with the increasing data size. For the given parameters, PCD generation times are as follows:

Scene Name	PCD Count	Generation Time
Pannilani – 5	19	0.46 Second
Pannilani – 6	51	0.75 Second
Pannilani – 7	19	231.45 Second
Pannilani – 8	50	478.37 Second

Table 6.10 – PCD File Generation Times

Chapter 7 : CONCLUSIONS AND FUTURE WORK

7.1. CONCLUSIONS

In this thesis, we have introduced a new framework to generate flexible synthetic datasets for urban reconstruction algorithms.

Considering our needs, we implemented a new 3D urban generator using 2D OpenStreetMap data. Although map data contains plenty of items with their specifications, we took only the important features which are adequate for urban models. Moreover, lack of 3D data led us making assumptions on parameters and it affected the final quality of city model.

Our framework was concerned with generating urban models all around the world. Therefore NASA SRTM data was chosen as the elevation data for our terrain, which is a global dataset. However, resolution of this data is not high enough (~90m) to reflect details, therefore structures like bridges or rivers in the scene could not be represented properly.

In order to harvest data from the generated model, virtual camera and laser scanner (LiDAR) is implemented. Generated data is flawless and does not contain any noise while in real-world examples different type of noises (e.g., motion blur, lens flare) are introduced. Urban reconstruction algorithms are designed to deal with the noise, however the current version of our data collector module does not allow this test.

Combining automated 3D urban generation and users edits, we tested that acceptable scenes can be created. We obtained promising datasets by using our data collector module and saw that with some additional features, our framework can be a popular test bench in the field.

7.2. FUTURE WORK

The work done so far introduces the concept of synthetic urban reconstruction dataset generation. Although current results are promising, there is still a long way to reach “photo realistic” datasets. In order to achieve our goal and take our place in the computer-vision field, all three modules in the framework need some improvements:

Urban Generator

The current urban generator concerns the creation of acceptable 3D models with a fundamental feature set. For the near future, unhandled specifications from OpenStreetMap data can be implemented. Even though current item specifications are mostly left empty by OSM

7.2. FUTURE WORK

contributors, there will be a complete set in the future and a full implementation will lead urban generators to create identical urban environments.

Our urban generator is currently using NASA SRTM dataset as elevation data where the resolution is not high. To improve quality, high resolution local height maps can be accepted as input which may exist in raw data format (the same as SRTM data) or in WFS⁹ format.

Data Collector

Output dataset from data collector component is flawless and noise free which contradicts real-world examples. In the future, options can be created to introduce motion blur and lens flare distortions to create more realistic test data.

User Interface

The current user interface meets all the requirements, although with the improvements which will be made in the “urban generator” and “data collector” in the near future, updates will be necessary in the menus. Our future goal for the framework is to create 3D models with minimal settings or edits made by the user.

⁹ WFS : Web Feature Service, provides an interface allowing requests for geographical features across the web

BIBLIOGRAPHY

- [1] S. M. Seitz and e. al., "A Comparison and Evaluation of Multi-view Stereo Reconstruction Algorithms," in *2006 IEEE Computer Society Conference (CVPR'06)*, 2006.
- [2] M. Pollefeys and e. al., "Detailed Real-time Urban 3d Reconstruction from Video," *International Journal of Computer Vision*, vol. 78, no. 2-3, pp. 143-167, 2008.
- [3] A. Akbarzadeh and e. al., "Towards urban 3D reconstruction from video," in *3D Data Processing, Visualization, and Transmission, Third International Symposium on IEEE*, 2006.
- [4] A. Geiger, L. Philip and U. Raquel, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference*, 2012.
- [5] wiki, "OSM Stats," OpenStreetMap, [Online]. Available: <http://wiki.openstreetmap.org/wiki/Statistics>. [Accessed 27 July 2015].
- [6] M. F. Goodchild, "Citizens as voluntary sensors: spatial data infrastructure in the world of Web 2.0," *International Journal of Spatial Data Infrastructures Research*, pp. 24-32, 2007.
- [7] wiki, "OSM About," OpenStreetMap, [Online]. Available: <http://wiki.openstreetmap.org/wiki/About>. [Accessed 27 July 2015].
- [8] [Online]. Available: <http://taginfo.openstreetmap.org/tags>.
- [9] M. Goetz and A. Zipf, "OpenStreetMap in 3D – Detailed Insights on the Current Situation in Germany," in *AGILE 2012, Avignon*, 2012.
- [10] Wiki, "Planet OSM," OpenStreetMap, [Online]. Available: <http://wiki.openstreetmap.org/wiki/Planet.osm>. [Accessed July 2015].
- [11] A. Zipf, "OpenStreetMap-3D," University of Heidelberg, Department of GIScience, 2012. [Online]. Available: <http://osm-3d.org/home.en.htm>. [Accessed July 2015].
- [12] "Osm-3D Information," University of Heidelberg, Department of GIScience, 2012. [Online]. Available: <http://osm-3d.org/informationen.en.htm>. [Accessed July 2015].
- [13] T. Knerr, "OSM2World," [Online]. Available: <http://osm2world.org/>. [Accessed July 2015].
- [14] "OSM2World," [Online]. Available: <http://osm2world.org/docs/OSM2World%20Overview.pdf>. [Accessed July 2015].
- [15] "What is CityEngine?," ArcGIS, [Online]. Available: <http://resources.arcgis.com/en/communities/city-engine/01w90000000m000000.htm>. [Accessed July 2015].
- [16] "Unity 5," Unity Technologies, [Online]. Available: <https://unity3d.com/5>. [Accessed July 2015].

- [17] "The Leading Global Game Industry Software," Unity Technologies, [Online]. Available: <https://unity3d.com/public-relations>. [Accessed July 2015].
- [18] "Developer Economics Q3 2014: State of the Developer Nation," Vision Mobile, July 2014. [Online]. Available: <http://www.visionmobile.com/product/developer-economics-q3-2014/>. [Accessed July 2015].
- [19] "mapToasterTopo," Integrated Mapping Ltd., 2009. [Online]. Available: <https://www.maptoaster.com/maptoaster-topo-nz/articles/projection/datum-projection.html>. [Accessed July 2015].
- [20] E. S. Battersby, P. M. Finn, E. L. Usery and K. H. Yamamoto, "Implications of Web Mercator and Its Use in Online Mapping," *Cartographica*, vol. 49, no. 2, pp. 85-101, 2014.
- [21] K. Pridal, "Tiles à la Google Maps: Coordinates, Tile Bounds and Projection," MapTiler, 2008. [Online]. Available: <http://www.maptiler.org/google-maps-coordinates-tile-bounds-projection/>. [Accessed July 2015].
- [22] N. I. a. M. Agency, "SRTM Topography," [Online]. Available: http://dds.cr.usgs.gov/srtm/version2_1/Documentation/SRTM_Topo.pdf. [Accessed July 2015].
- [23] Tsubaki, *UnityZip*, GitHub, 2013.
- [24] "How do web maps work?," MapBox, [Online]. Available: <https://www.mapbox.com/guides/how-web-maps-work/>. [Accessed July 2015].
- [25] Wiki, "Slippy map tilenames," [Online]. Available: http://wiki.openstreetmap.org/wiki/Slippy_map_tilenames. [Accessed July 2015].
- [26] M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf, "Polygon Triangulation," in *Computational Geometry (2nd revised ed.)*, Springer, 2000, pp. 45-61.
- [27] wiki, "Triangulator," Unity, [Online]. Available: <http://wiki.unity3d.com/index.php?title=Triangulator>. [Accessed July 2015].
- [28] "Behind the Scene, Street View," Google, 2007. [Online]. Available: <http://www.google.com/maps/about/behind-the-scenes/streetview/>. [Accessed August 2015].
- [29] "The PCD (Point Cloud Data) file format," [Online]. Available: http://pointclouds.org/documentation/tutorials/pcd_file_format.php. [Accessed August 2015].
- [30] Eric, "ObjReader," Star Scene Software, [Online]. Available: <http://www.starscenesoftware.com/objreader.html>. [Accessed September 2015].

APPENDIX A: OSM XML FILE

It is possible to download map data from the OpenStreetMap dataset in a number of ways. The full dataset is available from the OpenStreetMap website download area. When the “Export” button on the top is clicked, a side menu appears at the left-hand side containing exporting parameters. From the side menu, it is possible to select an area to download (Figure 1). After the “Export” button is clicked on the side menu, .osm file is downloaded which comes in the form of XML format.

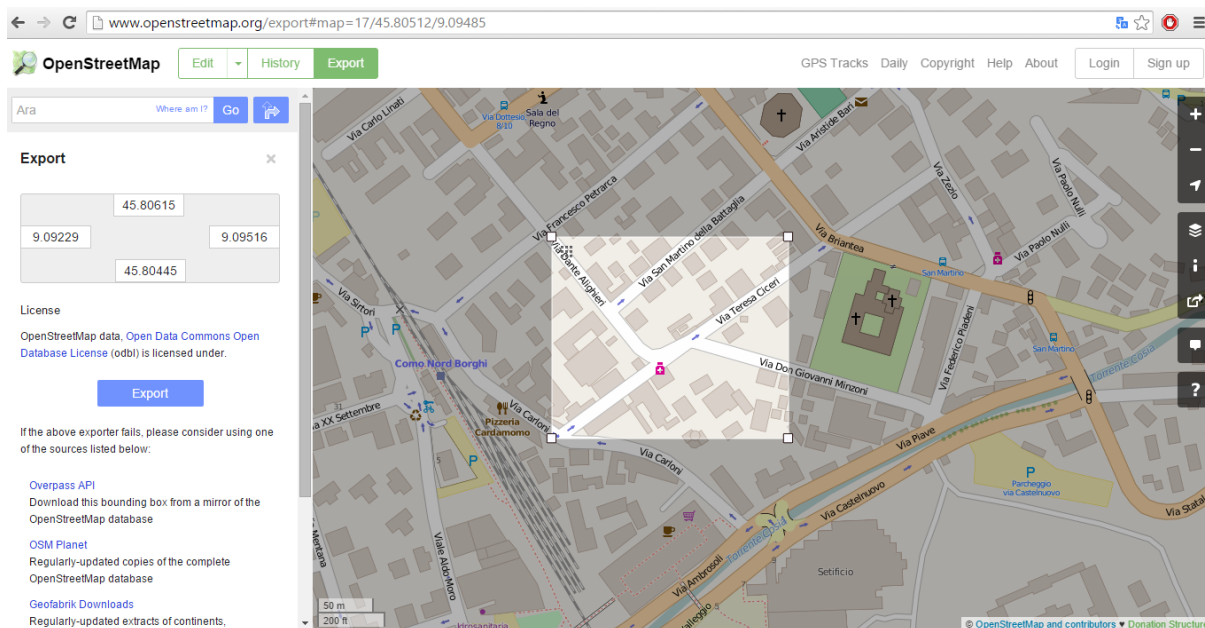


Figure 1: OpenStreetMap exporting xml file

A Complete OSM File Sample:

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.4.0 (23324 thorn-02.openstreetmap.org)"
copyright="OpenStreetMap and contributors"
attribution="http://www.openstreetmap.org/copyright"
license="http://opendatacommons.org/licenses/odbl/1-0/">
  <bounds minlat="45.8046100" minlon="9.0929100" maxlat="45.8056300" maxlon="9.0948700"/>
  <node id="273463279" visible="true" version="4" changeset="4960923" timestamp="2010-06-11T09:19:51Z" user="albertux" uid="118185" lat="45.8046780" lon="9.0969174"/>
  <node id="273858042" visible="true" version="5" changeset="1073713" timestamp="2009-05-04T12:45:36Z" user="Bengatzer" uid="89943" lat="45.8061261" lon="9.0957124"/>
  <node id="273466950" visible="true" version="4" changeset="4969225" timestamp="2010-06-12T11:14:33Z" user="jacopogg83" uid="54561" lat="45.8044650" lon="9.0923772"/>
  <node id="762136715" visible="true" version="1" changeset="4883483" timestamp="2010-06-02T12:28:17Z" user="DarkFlash" uid="131835" lat="45.8057056" lon="9.0936698"/>
  <node id="762136723" visible="true" version="1" changeset="4883483" timestamp="2010-06-02T12:28:17Z" user="DarkFlash" uid="131835" lat="45.8056963" lon="9.0933534"/>
  <node id="762136768" visible="true" version="1" changeset="4883483" timestamp="2010-06-02T12:28:18Z" user="DarkFlash" uid="131835" lat="45.8055944" lon="9.0934704"/>
```

```
<node id="762136792" visible="true" version="1" changeset="4883483" timestamp="2010-06-02T12:28:18Z" user="DarkFlash" uid="131835" lat="45.8058076" lon="9.0935528"/>
<node id="762167812" visible="true" version="1" changeset="4883649" timestamp="2010-06-02T12:48:26Z" user="DarkFlash" uid="131835" lat="45.8048843" lon="9.0949805"/>
<node id="762167840" visible="true" version="1" changeset="4883649" timestamp="2010-06-02T12:48:27Z" user="DarkFlash" uid="131835" lat="45.8049936" lon="9.0941720"/>
<node id="762167895" visible="true" version="1" changeset="4883649" timestamp="2010-06-02T12:48:27Z" user="DarkFlash" uid="131835" lat="45.8050021" lon="9.0950261"/>
<node id="762167902" visible="true" version="1" changeset="4883649" timestamp="2010-06-02T12:48:29Z" user="DarkFlash" uid="131835" lat="45.8050432" lon="9.0948076"/>
<node id="762167908" visible="true" version="1" changeset="4883649" timestamp="2010-06-02T12:48:29Z" user="DarkFlash" uid="131835" lat="45.8048192" lon="9.0940317"/>
<node id="762167932" visible="true" version="1" changeset="4883649" timestamp="2010-06-02T12:48:29Z" user="DarkFlash" uid="131835" lat="45.8049254" lon="9.0947621"/>
<node id="762167942" visible="true" version="1" changeset="4883649" timestamp="2010-06-02T12:48:29Z" user="DarkFlash" uid="131835" lat="45.8049063" lon="9.0939585"/>
<node id="762167960" visible="true" version="1" changeset="4883649" timestamp="2010-06-02T12:48:30Z" user="DarkFlash" uid="131835" lat="45.8049065" lon="9.0942452"/>
<node id="1046730882" visible="true" version="1" changeset="6701922" timestamp="2010-12-19T00:05:05Z" user="DarkFlash" uid="131835" lat="45.8050567" lon="9.0934299"/>
<node id="1046730907" visible="true" version="1" changeset="6701922" timestamp="2010-12-19T00:05:05Z" user="DarkFlash" uid="131835" lat="45.8050097" lon="9.0933441"/>
<node id="1046730950" visible="true" version="1" changeset="6701922" timestamp="2010-12-19T00:05:06Z" user="DarkFlash" uid="131835" lat="45.8051360" lon="9.0933404"/>
<node id="1046730991" visible="true" version="1" changeset="6701922" timestamp="2010-12-19T00:05:07Z" user="DarkFlash" uid="131835" lat="45.8050890" lon="9.0932546"/>
<node id="1148908649" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:15Z" user="albertux" uid="118185" lat="45.8057343" lon="9.0943515"/>
<node id="1148908707" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:20Z" user="albertux" uid="118185" lat="45.8044647" lon="9.0927489"/>
<node id="1148908731" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:22Z" user="albertux" uid="118185" lat="45.8054657" lon="9.0931773"/>
<node id="1148908772" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:26Z" user="albertux" uid="118185" lat="45.8053744" lon="9.0938223"/>
<node id="1148908824" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:31Z" user="albertux" uid="118185" lat="45.8058828" lon="9.0945869"/>
<node id="1148908646" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:13Z" user="DarkFlash" uid="131835" lat="45.8048432" lon="9.0945056"/>
<node id="1148910128" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:22Z" user="albertux" uid="118185" lat="45.8055550" lon="9.0948406"/>
<node id="1148908915" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:38Z" user="albertux" uid="118185" lat="45.8054720" lon="9.0942731"/>
<node id="1148908976" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:43Z" user="albertux" uid="118185" lat="45.8053821" lon="9.0936879"/>
<node id="1148909107" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:48Z" user="albertux" uid="118185" lat="45.8054820" lon="9.0946632"/>
<node id="1148909326" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:55Z" user="albertux" uid="118185" lat="45.8056539" lon="9.0941950"/>
<node id="1148909378" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:57Z" user="albertux" uid="118185" lat="45.8054315" lon="9.0946605"/>
<node id="1148909392" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:58Z" user="albertux" uid="118185" lat="45.8053267" lon="9.0938635"/>
<node id="1148909399" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:58Z" user="albertux" uid="118185" lat="45.8052860" lon="9.0930823"/>
<node id="1148909481" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:00Z" user="albertux" uid="118185" lat="45.8048134" lon="9.0938980"/>
<node id="1148910288" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:29Z" user="albertux" uid="118185" lat="45.8054337" lon="9.0945718"/>
```

```

<node id="1148910298" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:29Z" user="albertux" uid="118185" lat="45.8053887" lon="9.0939419"/>
<node id="1148910469" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:36Z" user="albertux" uid="118185" lat="45.8055764" lon="9.0942753"/>
<node id="1148908930" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:39Z" user="albertux" uid="118185" lat="45.8057159" lon="9.0947671"/>
<node id="1148908940" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:40Z" user="albertux" uid="118185" lat="45.8054874" lon="9.0952077"/>
<node id="1148908982" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:43Z" user="albertux" uid="118185" lat="45.8053662" lon="9.0937958"/>
<node id="1148909158" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:50Z" user="albertux" uid="118185" lat="45.8055043" lon="9.0939554"/>
<node id="1148909175" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:50Z" user="albertux" uid="118185" lat="45.8053574" lon="9.0940907"/>
<node id="1148909253" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:53Z" user="albertux" uid="118185" lat="45.8045720" lon="9.0926425"/>
<node id="1148909438" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:59Z" user="albertux" uid="118185" lat="45.8054167" lon="9.0939158"/>
<node id="1148909443" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:59Z" user="albertux" uid="118185" lat="45.8054539" lon="9.0942910"/>
<node id="1148909788" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:12Z" user="albertux" uid="118185" lat="45.8056132" lon="9.0945713"/>
<node id="1148909833" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:13Z" user="albertux" uid="118185" lat="45.8053891" lon="9.0950127"/>
<node id="1148909876" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:14Z" user="albertux" uid="118185" lat="45.8055504" lon="9.0933255"/>
<node id="1148909895" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:15Z" user="albertux" uid="118185" lat="45.8052536" lon="9.0944413"/>
<node id="1148908992" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:44Z" user="albertux" uid="118185" lat="45.8053170" lon="9.0930444"/>
<node id="1148909563" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:04Z" user="albertux" uid="118185" lat="45.8054409" lon="9.0934542"/>
<node id="1148909590" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:05Z" user="albertux" uid="118185" lat="45.8053384" lon="9.0946140"/>
<node id="1148910177" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:23Z" user="albertux" uid="118185" lat="45.8054668" lon="9.0939906"/>
<node id="1148910181" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:24Z" user="albertux" uid="118185" lat="45.8052564" lon="9.0938219"/>
<node id="1148910219" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:26Z" user="albertux" uid="118185" lat="45.8048048" lon="9.0931259"/>
<node id="1148910358" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:32Z" user="albertux" uid="118185" lat="45.8046752" lon="9.0940394"/>
<node id="1148910521" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:38Z" user="albertux" uid="118185" lat="45.8052286" lon="9.0929835"/>
<node id="1148909598" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:05Z" user="albertux" uid="118185" lat="45.8054048" lon="9.0940282"/>
<node id="1148909906" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:15Z" user="albertux" uid="118185" lat="45.8052881" lon="9.0938815"/>
<node id="1148910330" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:31Z" user="albertux" uid="118185" lat="45.8053446" lon="9.0937231"/>
<node id="1148910342" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:31Z" user="albertux" uid="118185" lat="45.8055017" lon="9.0942436"/>
<node id="1148910555" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:40Z" user="albertux" uid="118185" lat="45.8054052" lon="9.0940433"/>
<node id="1148910607" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:44Z" user="albertux" uid="118185" lat="45.8055419" lon="9.0942074"/>
<node id="1148908576" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:08Z" user="albertux" uid="118185" lat="45.8053345" lon="9.0937363"/>

```



```
<node id="1148908847" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:33Z" user="albertux" uid="118185" lat="45.8053562" lon="9.0933060"/>
<node id="1148908892" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:15Z" user="DarkFlash" uid="131835" lat="45.8046427" lon="9.0934727"/>
<node id="1148908670" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:17Z" user="albertux" uid="118185" lat="45.8053464" lon="9.0938483"/>
<node id="1148908786" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:27Z" user="albertux" uid="118185" lat="45.8047374" lon="9.0937450"/>
<node id="1148910623" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:46Z" user="albertux" uid="118185" lat="45.8053437" lon="9.0943504"/>
<node id="1148910625" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:46Z" user="albertux" uid="118185" lat="45.8052950" lon="9.0938944"/>
<node id="1148910579" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:41Z" user="albertux" uid="118185" lat="45.8045992" lon="9.0938864"/>
<node id="1148910636" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:13Z" user="DarkFlash" uid="131835" lat="45.8045095" lon="9.0948776"/>
<node id="1148910616" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:14Z" user="DarkFlash" uid="131835" lat="45.8047841" lon="9.0928830"/>
<node id="1148910672" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:14Z" user="DarkFlash" uid="131835" lat="45.8050750" lon="9.0924371"/>
<node id="1148910630" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:16Z" user="DarkFlash" uid="131835" lat="45.8047326" lon="9.0932157"/>
<node id="1148908864" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:34Z" user="albertux" uid="118185" lat="45.8054284" lon="9.0945230"/>
<node id="1148908873" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:25:35Z" user="albertux" uid="118185" lat="45.8052596" lon="9.0929406"/>
<node id="1148910706" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:54Z" user="albertux" uid="118185" lat="45.8056533" lon="9.0950357"/>
<node id="1148910713" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:13Z" user="DarkFlash" uid="131835" lat="45.8053091" lon="9.0928448"/>
<node id="1148910709" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:17Z" user="DarkFlash" uid="131835" lat="45.8046219" lon="9.0936509"/>
<node id="1148910572" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:18Z" user="DarkFlash" uid="131835" lat="45.8046535" lon="9.0934954"/>
<node id="1148910068" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:20Z" user="albertux" uid="118185" lat="45.8055793" lon="9.0945101"/>
<node id="1148910109" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:21Z" user="albertux" uid="118185" lat="45.8057801" lon="9.0943911"/>
<node id="1148910396" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:33Z" user="albertux" uid="118185" lat="45.8046974" lon="9.0932323"/>
<node id="1148910499" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:37Z" user="albertux" uid="118185" lat="45.8054842" lon="9.0945744"/>
<node id="1148910512" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:38Z" user="albertux" uid="118185" lat="45.8053731" lon="9.0940591"/>
<node id="1271392804" visible="true" version="1" changeset="8042616" timestamp="2011-05-03T19:12:05Z" user="albertux" uid="118185" lat="45.8058452" lon="9.0931712"/>
<node id="1271393000" visible="true" version="1" changeset="8042616" timestamp="2011-05-03T19:12:34Z" user="albertux" uid="118185" lat="45.8056102" lon="9.0930339"/>
<node id="1271388022" visible="true" version="1" changeset="8042616" timestamp="2011-05-03T19:07:17Z" user="albertux" uid="118185" lat="45.8057495" lon="9.0932871"/>
<node id="1271387605" visible="true" version="1" changeset="8042616" timestamp="2011-05-03T19:06:42Z" user="albertux" uid="118185" lat="45.8057144" lon="9.0929577"/>
<node id="1271388451" visible="true" version="1" changeset="8042616" timestamp="2011-05-03T19:07:42Z" user="albertux" uid="118185" lat="45.8056070" lon="9.0929851"/>
<node id="1271391089" visible="true" version="1" changeset="8042616" timestamp="2011-05-03T19:09:47Z" user="albertux" uid="118185" lat="45.8058494" lon="9.0926908"/>
<node id="1271391221" visible="true" version="1" changeset="8042616" timestamp="2011-05-03T19:09:54Z" user="albertux" uid="118185" lat="45.8058154" lon="9.0931132"/>
```



```

<node id="1271392651" visible="true" version="1" changeset="8042616" timestamp="2011-05-03T19:11:55Z" user="albertux" uid="118185" lat="45.8059376" lon="9.0928281"/>
<node id="1271388625" visible="true" version="1" changeset="8042616" timestamp="2011-05-03T19:07:48Z" user="albertux" uid="118185" lat="45.8057665" lon="9.0931727"/>
<node id="1271391177" visible="true" version="1" changeset="8042616" timestamp="2011-05-03T19:09:51Z" user="albertux" uid="118185" lat="45.8057527" lon="9.0930309"/>
<node id="1271390089" visible="true" version="1" changeset="8042616" timestamp="2011-05-03T19:08:46Z" user="albertux" uid="118185" lat="45.8056772" lon="9.0930080"/>
<node id="1148909551" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:12Z" user="DarkFlash" uid="131835" lat="45.8049351" lon="9.0931794"/>
<node id="1148909922" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:13Z" user="DarkFlash" uid="131835" lat="45.8048696" lon="9.0945568"/>
<node id="1148910002" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:13Z" user="DarkFlash" uid="131835" lat="45.8050545" lon="9.0936794"/>
<node id="1148908747" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:14Z" user="DarkFlash" uid="131835" lat="45.8048309" lon="9.0932085"/>
<node id="1148910702" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:14Z" user="DarkFlash" uid="131835" lat="45.8046235" lon="9.0942801"/>
<node id="1148909614" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:14Z" user="DarkFlash" uid="131835" lat="45.8051107" lon="9.0924853"/>
<node id="1148908958" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:14Z" user="DarkFlash" uid="131835" lat="45.8048933" lon="9.0927685"/>
<node id="1148910301" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:14Z" user="DarkFlash" uid="131835" lat="45.8049430" lon="9.0934444"/>
<node id="1148908835" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:14Z" user="DarkFlash" uid="131835" lat="45.8046836" lon="9.0944471"/>
<node id="1148910189" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:14Z" user="DarkFlash" uid="131835" lat="45.8045172" lon="9.0935077"/>
<node id="1148909512" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:14Z" user="DarkFlash" uid="131835" lat="45.8050916" lon="9.0930929"/>
<node id="1148909682" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:14Z" user="DarkFlash" uid="131835" lat="45.8049640" lon="9.0934794"/>
<node id="1148908683" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:14Z" user="DarkFlash" uid="131835" lat="45.8047210" lon="9.0933084"/>
<node id="1148909003" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:15Z" user="DarkFlash" uid="131835" lat="45.8044908" lon="9.0934521"/>
<node id="1148908615" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:15Z" user="DarkFlash" uid="131835" lat="45.8048612" lon="9.0935751"/>
<node id="1148908750" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:15Z" user="DarkFlash" uid="131835" lat="45.8051862" lon="9.0926163"/>
<node id="1148909982" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:15Z" user="DarkFlash" uid="131835" lat="45.8045374" lon="9.0935757"/>
<node id="1148909872" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:15Z" user="DarkFlash" uid="131835" lat="45.8053773" lon="9.0929632"/>
<node id="1148908583" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:15Z" user="DarkFlash" uid="131835" lat="45.8055264" lon="9.0927863"/>
<node id="1148910535" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:15Z" user="DarkFlash" uid="131835" lat="45.8045379" lon="9.0946930"/>
<node id="1148909504" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:15Z" user="DarkFlash" uid="131835" lat="45.8045962" lon="9.0938398"/>
<node id="1148908939" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:15Z" user="DarkFlash" uid="131835" lat="45.8047809" lon="9.0941085"/>
<node id="1148910191" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:16Z" user="DarkFlash" uid="131835" lat="45.8047033" lon="9.0947348"/>
<node id="1148909052" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:16Z" user="DarkFlash" uid="131835" lat="45.8048888" lon="9.0926569"/>
<node id="1148909775" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:17Z" user="DarkFlash" uid="131835" lat="45.8052897" lon="9.0928591"/>

```

```
<node id="1148908607" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:17Z" user="DarkFlash" uid="131835" lat="45.8045681" lon="9.0936402"/>
<node id="1148908586" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:17Z" user="DarkFlash" uid="131835" lat="45.8046530" lon="9.0946381"/>
<node id="1148909821" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:18Z" user="DarkFlash" uid="131835" lat="45.8047015" lon="9.0944276"/>
<node id="1148910308" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:18Z" user="DarkFlash" uid="131835" lat="45.8052463" lon="9.0925451"/>
<node id="1148909117" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:18Z" user="DarkFlash" uid="131835" lat="45.8050443" lon="9.0930649"/>
<node id="1148910656" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:18Z" user="DarkFlash" uid="131835" lat="45.8049518" lon="9.0937751"/>
<node id="1148910633" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:18Z" user="DarkFlash" uid="131835" lat="45.8052546" lon="9.0923146"/>
<node id="1148909853" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:18Z" user="DarkFlash" uid="131835" lat="45.8045843" lon="9.0936877"/>
<node id="1148908690" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:19Z" user="DarkFlash" uid="131835" lat="45.8048189" lon="9.0944598"/>
<node id="1148909724" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:19Z" user="DarkFlash" uid="131835" lat="45.8049371" lon="9.0944033"/>
<node id="1148909129" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:19Z" user="DarkFlash" uid="131835" lat="45.8053743" lon="9.0927674"/>
<node id="1148909232" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:19Z" user="DarkFlash" uid="131835" lat="45.8045225" lon="9.0936848"/>
<node id="1148909462" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:19Z" user="DarkFlash" uid="131835" lat="45.8047603" lon="9.0933910"/>
<node id="1148910459" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:19Z" user="DarkFlash" uid="131835" lat="45.8047373" lon="9.0945486"/>
<node id="1148909188" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:19Z" user="DarkFlash" uid="131835" lat="45.8053060" lon="9.0928398"/>
<node id="1148908713" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:19Z" user="DarkFlash" uid="131835" lat="45.8048894" lon="9.0927403"/>
<node id="1148909120" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:19Z" user="DarkFlash" uid="131835" lat="45.8046567" lon="9.0937243"/>
<node id="1148908875" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:19Z" user="DarkFlash" uid="131835" lat="45.8047535" lon="9.0949548"/>
<node id="1148910350" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:20Z" user="DarkFlash" uid="131835" lat="45.8050970" lon="9.0924755"/>
<node id="1148908844" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:20Z" user="DarkFlash" uid="131835" lat="45.8046374" lon="9.0937995"/>
<node id="1148909611" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:20Z" user="DarkFlash" uid="131835" lat="45.8047615" lon="9.0932764"/>
<node id="1148910243" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:20Z" user="DarkFlash" uid="131835" lat="45.8049175" lon="9.0927070"/>
<node id="1148909623" visible="true" version="2" changeset="8167387" timestamp="2011-05-16T22:50:20Z" user="DarkFlash" uid="131835" lat="45.8047819" lon="9.0947702"/>
<node id="1827726842" visible="true" version="1" changeset="12248225" timestamp="2012-07-16T17:03:29Z" user="DarkFlash" uid="131835" lat="45.8046897" lon="9.0968225"/>
<node id="1827726844" visible="true" version="1" changeset="12248225" timestamp="2012-07-16T17:03:29Z" user="DarkFlash" uid="131835" lat="45.8047382" lon="9.0966696"/>
<node id="1827726850" visible="true" version="1" changeset="12248225" timestamp="2012-07-16T17:03:29Z" user="DarkFlash" uid="131835" lat="45.8051454" lon="9.0934880"/>
<node id="1827726853" visible="true" version="1" changeset="12248225" timestamp="2012-07-16T17:03:29Z" user="DarkFlash" uid="131835" lat="45.8052133" lon="9.0933454"/>
<node id="1827726857" visible="true" version="1" changeset="12248225" timestamp="2012-07-16T17:03:29Z" user="DarkFlash" uid="131835" lat="45.8055203" lon="9.0929908"/>
<node id="1827726859" visible="true" version="1" changeset="12248225" timestamp="2012-07-16T17:03:29Z" user="DarkFlash" uid="131835" lat="45.8062016" lon="9.0922038"/>
```

```

<node id="1827773691" visible="true" version="3" changeset="23041234" timestamp="2014-06-20T10:32:25Z" user="Taurus77" uid="2044340" lat="45.8067270" lon="9.0969748"/>
<node id="2386516448" visible="true" version="1" changeset="16975875" timestamp="2013-07-16T14:45:47Z" user="albertux" uid="118185" lat="45.8051596" lon="9.0936307"/>
<node id="2386516449" visible="true" version="1" changeset="16975875" timestamp="2013-07-16T14:45:47Z" user="albertux" uid="118185" lat="45.8052698" lon="9.0939954"/>
<node id="1827726854" visible="true" version="2" changeset="16975875" timestamp="2013-07-16T14:46:04Z" user="albertux" uid="118185" lat="45.8052769" lon="9.0939717"/>
<node id="1827726892" visible="true" version="2" changeset="16975875" timestamp="2013-07-16T14:46:04Z" user="albertux" uid="118185" lat="45.8069588" lon="9.0913023"/>
<node id="1827726878" visible="true" version="2" changeset="24353872" timestamp="2014-07-25T16:26:05Z" user="Taurus77" uid="2044340" lat="45.8067124" lon="9.0949638"/>
<node id="3025191133" visible="true" version="1" changeset="24840002" timestamp="2014-08-18T18:13:12Z" user="Taurus77" uid="2044340" lat="45.8050457" lon="9.0935981">
  <tag k="amenity" v="pharmacy"/>
  <tag k="dispensing" v="no"/>
</node>
<way id="60892806" visible="true" version="1" changeset="4883483" timestamp="2010-06-02T12:28:19Z" user="DarkFlash" uid="131835">
  <nd ref="762136768"/>
  <nd ref="762136723"/>
  <nd ref="762136792"/>
  <nd ref="762136715"/>
  <nd ref="762136768"/>
  <tag k="building" v="yes"/>
</way>
<way id="60893507" visible="true" version="1" changeset="4883649" timestamp="2010-06-02T12:48:47Z" user="DarkFlash" uid="131835">
  <nd ref="762167902"/>
  <nd ref="762167932"/>
  <nd ref="762167812"/>
  <nd ref="762167895"/>
  <nd ref="762167902"/>
  <tag k="building" v="yes"/>
</way>
<way id="60893513" visible="true" version="1" changeset="4883649" timestamp="2010-06-02T12:48:48Z" user="DarkFlash" uid="131835">
  <nd ref="762167942"/>
  <nd ref="762167908"/>
  <nd ref="762167960"/>
  <nd ref="762167840"/>
  <nd ref="762167942"/>
  <tag k="building" v="yes"/>
</way>
<way id="90274026" visible="true" version="1" changeset="6701922" timestamp="2010-12-19T00:05:07Z" user="DarkFlash" uid="131835">
  <nd ref="1046730991"/>
  <nd ref="1046730950"/>
  <nd ref="1046730882"/>
  <nd ref="1046730907"/>
  <nd ref="1046730991"/>
  <tag k="building" v="yes"/>
</way>
<way id="99314877" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:56Z" user="albertux" uid="118185">
  <nd ref="1148908690"/>
  <nd ref="1148908646"/>
  <nd ref="1148909922"/>

```

```

<nd ref="1148910191"/>
<nd ref="1148908586"/>
<nd ref="1148910459"/>
<nd ref="1148908690"/>
<tag k="building" v="yes"/>
</way>
<way id="99314879" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:57Z" user="albertux" uid="118185">
  <nd ref="1148908992"/>
  <nd ref="1148908873"/>
  <nd ref="1148910521"/>
  <nd ref="1148909399"/>
  <nd ref="1148908992"/>
  <tag k="building" v="yes"/>
</way>
<way id="99314880" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:57Z" user="albertux" uid="118185">
  <nd ref="1148908576"/>
  <nd ref="1148908982"/>
  <nd ref="1148909906"/>
  <nd ref="1148910181"/>
  <nd ref="1148908576"/>
  <tag k="building" v="yes"/>
</way>
<way id="99314881" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:58Z" user="albertux" uid="118185">
  <nd ref="1148910706"/>
  <nd ref="1148908940"/>
  <nd ref="1148909833"/>
  <nd ref="1148910128"/>
  <nd ref="1148910706"/>
  <tag k="building" v="yes"/>
</way>
<way id="99314885" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:26:59Z" user="albertux" uid="118185">
  <nd ref="1148910625"/>
  <nd ref="1148909392"/>
  <nd ref="1148909598"/>
  <nd ref="1148910512"/>
  <nd ref="1148910625"/>
  <tag k="building" v="yes"/>
</way>
<way id="99314892" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:27:02Z" user="albertux" uid="118185">
  <nd ref="1148910499"/>
  <nd ref="1148909107"/>
  <nd ref="1148909378"/>
  <nd ref="1148910288"/>
  <nd ref="1148910499"/>
  <tag k="building" v="yes"/>
</way>
<way id="99314903" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:27:07Z" user="albertux" uid="118185">
  <nd ref="1148910623"/>
  <nd ref="1148908864"/>
  <nd ref="1148909590"/>
  <nd ref="1148909895"/>
  <nd ref="1148910623"/>

```

```

    <tag k="building" v="yes"/>
  </way>
  <way id="99314886" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:27:00Z" user="albertux" uid="118185">
    <nd ref="1148908583"/>
    <nd ref="1148910633"/>
    <nd ref="1148909614"/>
    <nd ref="1148908750"/>
    <nd ref="1148910308"/>
    <nd ref="1148909129"/>
    <nd ref="1148910713"/>
    <nd ref="1148909872"/>
    <nd ref="1148908583"/>
    <tag k="building" v="yes"/>
  </way>
  <way id="99314887" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:27:00Z" user="albertux" uid="118185">
    <nd ref="1148908707"/>
    <nd ref="1148909253"/>
    <nd ref="1148910219"/>
    <nd ref="1148910396"/>
    <nd ref="1148908707"/>
    <tag k="building" v="yes"/>
  </way>
  <way id="99314888" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:27:01Z" user="albertux" uid="118185">
    <nd ref="1148909504"/>
    <nd ref="1148909232"/>
    <nd ref="1148908607"/>
    <nd ref="1148909982"/>
    <nd ref="1148908892"/>
    <nd ref="1148910572"/>
    <nd ref="1148909462"/>
    <nd ref="1148908683"/>
    <nd ref="1148910189"/>
    <nd ref="1148909003"/>
    <nd ref="1148910630"/>
    <nd ref="1148909611"/>
    <nd ref="1148908747"/>
    <nd ref="1148910301"/>
    <nd ref="1148909120"/>
    <nd ref="1148910709"/>
    <nd ref="1148909853"/>
    <nd ref="1148908844"/>
    <nd ref="1148909504"/>
    <tag k="building" v="yes"/>
  </way>
  <way id="99314890" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:27:01Z" user="albertux" uid="118185">
    <nd ref="1148910555"/>
    <nd ref="1148910342"/>
    <nd ref="1148908915"/>
    <nd ref="1148909443"/>
    <nd ref="1148909175"/>
    <nd ref="1148910555"/>
    <tag k="building" v="yes"/>
  </way>

```

```
<way id="99314894" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:27:03Z" user="albertux" uid="118185">
  <nd ref="1148908615"/>
  <nd ref="1148909682"/>
  <nd ref="1148910002"/>
  <nd ref="1148910656"/>
  <nd ref="1148908615"/>
  <tag k="building" v="yes"/>
</way>
<way id="99314909" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:27:10Z" user="albertux" uid="118185">
  <nd ref="1148910109"/>
  <nd ref="1148908824"/>
  <nd ref="1148908930"/>
  <nd ref="1148909788"/>
  <nd ref="1148910109"/>
  <tag k="building" v="yes"/>
</way>
<way id="99314910" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:27:10Z" user="albertux" uid="118185">
  <nd ref="1148908670"/>
  <nd ref="1148908772"/>
  <nd ref="1148909438"/>
  <nd ref="1148910298"/>
  <nd ref="1148908670"/>
  <tag k="building" v="yes"/>
</way>
<way id="99314912" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:27:11Z" user="albertux" uid="118185">
  <nd ref="1148910330"/>
  <nd ref="1148908976"/>
  <nd ref="1148909158"/>
  <nd ref="1148910177"/>
  <nd ref="1148910330"/>
  <tag k="building" v="yes"/>
</way>
<way id="99314915" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:27:13Z" user="albertux" uid="118185">
  <nd ref="1148909775"/>
  <nd ref="1148909512"/>
  <nd ref="1148908713"/>
  <nd ref="1148910243"/>
  <nd ref="1148909052"/>
  <nd ref="1148910672"/>
  <nd ref="1148910350"/>
  <nd ref="1148909188"/>
  <nd ref="1148909775"/>
  <tag k="building" v="yes"/>
</way>
<way id="99314918" visible="true" version="1" changeset="7263630" timestamp="2011-02-12T11:27:14Z" user="albertux" uid="118185">
  <nd ref="1148908939"/>
  <nd ref="1148910702"/>
  <nd ref="1148909821"/>
  <nd ref="1148908835"/>
  <nd ref="1148910459"/>
  <nd ref="1148908690"/>
  <nd ref="1148908646"/>
</way>
```

```

<nd ref="1148909724"/>
<nd ref="1148908939"/>
<tag k="building" v="yes"/>
</way>
<way id="99314919" visible="true" version="1" changeset="7263630" timestamp="2011-02-
12T11:27:14Z" user="albertux" uid="118185">
  <nd ref="1148909876"/>
  <nd ref="1148909563"/>
  <nd ref="1148908847"/>
  <nd ref="1148908731"/>
  <nd ref="1148909876"/>
  <tag k="building" v="yes"/>
</way>
<way id="99314924" visible="true" version="1" changeset="7263630" timestamp="2011-02-
12T11:27:16Z" user="albertux" uid="118185">
  <nd ref="1148909117"/>
  <nd ref="1148908958"/>
  <nd ref="1148910616"/>
  <nd ref="1148909551"/>
  <nd ref="1148909117"/>
  <tag k="building" v="yes"/>
</way>
<way id="99314925" visible="true" version="1" changeset="7263630" timestamp="2011-02-
12T11:27:17Z" user="albertux" uid="118185">
  <nd ref="1148910607"/>
  <nd ref="1148910469"/>
  <nd ref="1148909326"/>
  <nd ref="1148908649"/>
  <nd ref="1148910068"/>
  <nd ref="1148908915"/>
  <nd ref="1148910342"/>
  <nd ref="1148910607"/>
  <tag k="building" v="yes"/>
</way>
<way id="99314928" visible="true" version="1" changeset="7263630" timestamp="2011-02-
12T11:27:18Z" user="albertux" uid="118185">
  <nd ref="1148908875"/>
  <nd ref="1148910636"/>
  <nd ref="1148910535"/>
  <nd ref="1148909623"/>
  <nd ref="1148908875"/>
  <tag k="building" v="yes"/>
</way>
<way id="99314930" visible="true" version="1" changeset="7263630" timestamp="2011-02-
12T11:27:19Z" user="albertux" uid="118185">
  <nd ref="1148908786"/>
  <nd ref="1148909481"/>
  <nd ref="1148910358"/>
  <nd ref="1148910579"/>
  <nd ref="1148908786"/>
  <tag k="building" v="yes"/>
</way>
<way id="111641019" visible="true" version="1" changeset="8042616" timestamp="2011-05-
03T19:14:47Z" user="albertux" uid="118185">
  <nd ref="1271391177"/>
  <nd ref="1271392651"/>
  <nd ref="1271391089"/>
  <nd ref="1271388451"/>

```



```

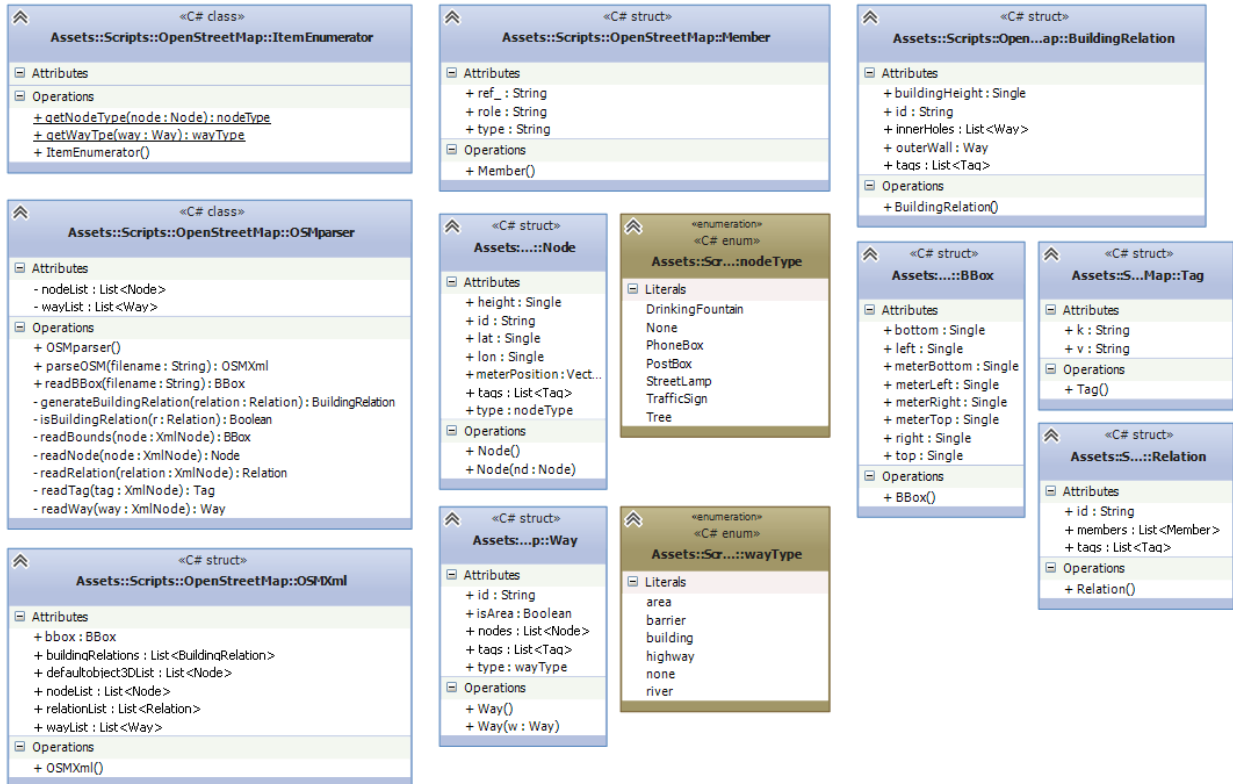
<nd ref="1271393000"/>
<nd ref="1271388022"/>
<nd ref="1271392804"/>
<nd ref="1271391221"/>
<nd ref="1271388625"/>
<nd ref="1271390089"/>
<nd ref="1271387605"/>
<nd ref="1271391177"/>
<tag k="building" v="yes"/>
</way>
<way id="171794553" visible="true" version="6" changeset="13899950" timestamp="2012-11-16T22:14:53Z" user="Gattomagico" uid="1058946">
  <nd ref="1827726857"/>
  <nd ref="1827726878"/>
  <tag k="highway" v="unclassified"/>
  <tag k="name" v="Via San Martino della Battaglia"/>
  <tag k="oneway" v="yes"/>
</way>
<way id="230123130" visible="true" version="1" changeset="16975875" timestamp="2013-07-16T14:45:49Z" user="albertux" uid="118185">
  <nd ref="2386516449"/>
  <nd ref="1827726844"/>
  <nd ref="1827726842"/>
  <nd ref="273463279"/>
  <tag k="highway" v="unclassified"/>
  <tag k="name" v="Via Don Giovanni Minzoni"/>
  <tag k="oneway" v="no"/>
</way>
<way id="171794542" visible="true" version="3" changeset="16975875" timestamp="2013-07-16T14:45:55Z" user="albertux" uid="118185">
  <nd ref="1827726892"/>
  <nd ref="1827726859"/>
  <nd ref="1827726857"/>
  <nd ref="1827726853"/>
  <nd ref="1827726850"/>
  <nd ref="2386516448"/>
  <nd ref="1827726854"/>
  <nd ref="2386516449"/>
  <tag k="highway" v="unclassified"/>
  <tag k="name" v="Via Dante Alighieri"/>
  <tag k="oneway" v="no"/>
</way>
<way id="171794543" visible="true" version="6" changeset="16975875" timestamp="2013-07-16T14:45:59Z" user="albertux" uid="118185">
  <nd ref="273466950"/>
  <nd ref="1827726854"/>
  <nd ref="273858042"/>
  <nd ref="1827773691"/>
  <tag k="highway" v="unclassified"/>
  <tag k="name" v="Via Teresa Cicceri"/>
  <tag k="oneway" v="yes"/>
</way>
</osm>

```


APPENDIX B: CLASS DIAGRAMS

URBAN GENERATOR

1. OSM Parser

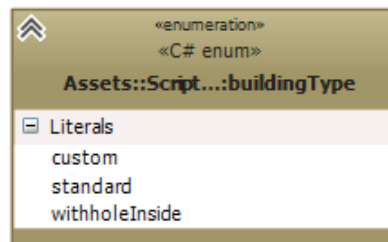
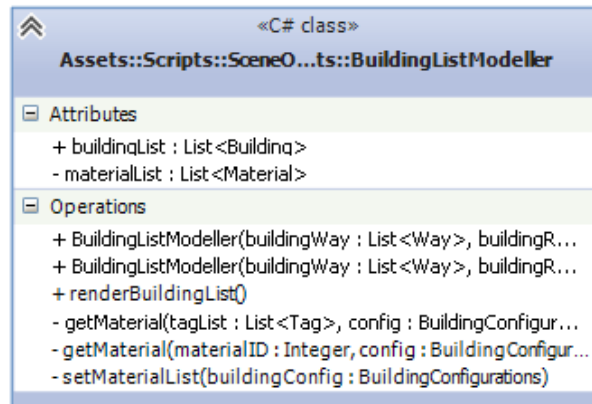
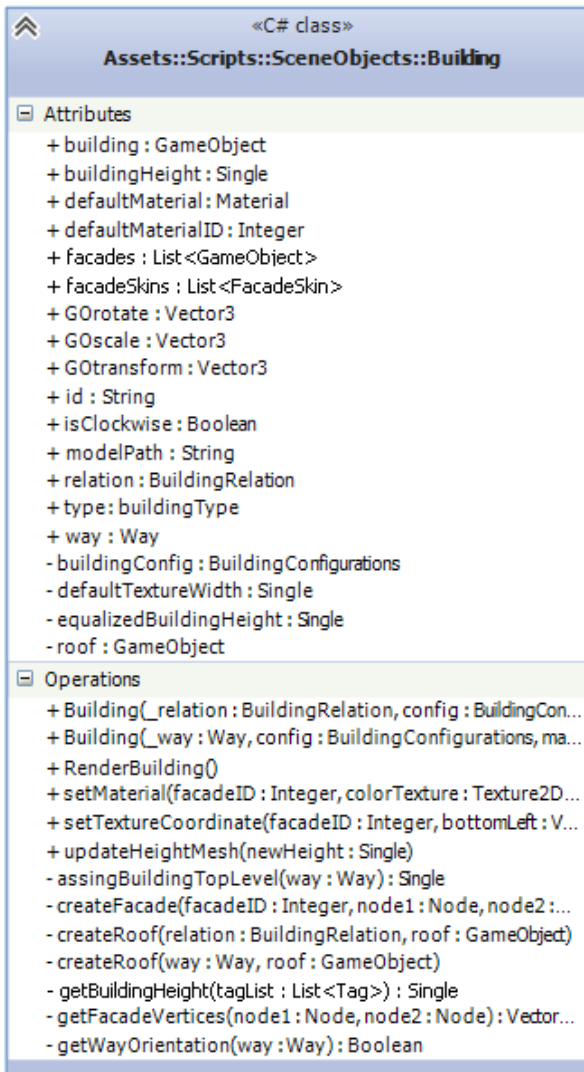


2. Terrain

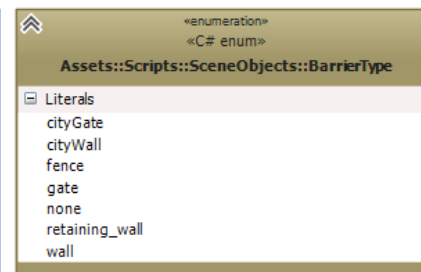
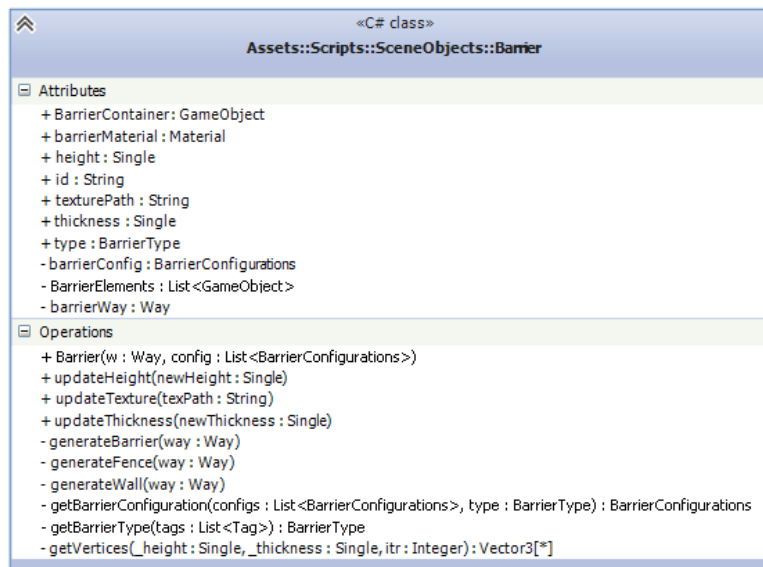
Assets::Scripts::HeightMap

- «C# class»
HeightmapLoader
 - Attributes
 - + baseURL
 - + continent : HeightmapContinent
 - + heightmap : Int16[*]
 - Operations
 - + HeightmapLoader(bbox : BBox, _contin...
 - fillHeightmap(savePath : String)
 - getContinent(bbox : BBox)
 - ReadFully(input : FileStream) : Byte[*]
- «C# class»
TerrainTextureHandler
 - Attributes
 - + tileFolder : String
 - + zoomLevel : Integer
 - tileSize : Integer
 - Operations
 - + generateTexture(provider : MapProvide...
 - + TerrainTextureHandler()
 - ConcatTexture(textures : Texture2D[*],...
 - CropTexture(originalTexture : Texture2D...
 - DownloadTile(tileX : Integer, tileY : Inte...
 - FindTiles(bbox : BBox, mintileCoord : Ve...
 - generateCroppingRect(bbox : BBox) : Rect
 - generateUncroppedTexture(bbox : BBox...
- «enumeration»
«C# enum»
HeightmapCon...
 - Literals
 - Africa
 - Australia
 - Eurasia
 - North_America
 - South_America
- «C# struct»
HeightmapInfo
 - Attributes
 - + coordmax : Vector2
 - + coordmin : Vector2
 - + height : Single
 - + Lat : Integer
 - + Lon : Integer
 - + sizeX : Single
 - + sizeZ : Single
 - + width : Single
 - Operations
 - + HeightmapInfo()
- «C# class»
Geography
 - Attributes
 - + Geography()
 - + LatLontoMeters(lat : ...)
 - + meterstoLatLon(me...
 - + meterstoLatLonDou...
 - + MetersToTile(meter...
 - + metertoPixel(meter...
 - + pixeltoMeter(pixelX...
 - + pixeltoTile(pixelX : ...)
 - + Resolution(zoom : I...
 - Operations
 - + Geography()
 - + LatLontoMeters(lat : ...)
 - + meterstoLatLon(me...
 - + meterstoLatLonDou...
 - + MetersToTile(meter...
 - + metertoPixel(meter...
 - + pixeltoMeter(pixelX...
 - + pixeltoTile(pixelX : ...)
 - + Resolution(zoom : I...
- «C# class»
myTerrain
 - Attributes
 - + gridList : List<GameObject>
 - + scenebbox : BBox
 - + terrainInfo : TerrainInfo
 - + terrainObject : GameObject
 - + textureType : MapProvider
 - heightmap : HeightmapLoader
 - OSMfileName : String
 - Operations
 - + getTerrainHeight(lat : Single, lon : Single)...
 - + getTerrainHeight2(meterx : Single, meterz...
 - + myTerrain(_heightmap : HeightmapLoade...
 - calculateNormals(i : Integer, j : Integer) : ...
 - createGrid(i : Integer, j : Integer)
 - drawBoundsforDebug()
 - drawUnderPlates()
 - getTerrainTriangleHeight(meterx : Single, ...
 - HighPrecisionTerrainHeight(meterx : Single...
- «enumeration»
«C# enum»
MapProvider
 - Literals
 - BingMapAerial
 - BingMapStreet
 - MapQuest
 - OpenStreetMap
 - OpenStreetMapNoLabel

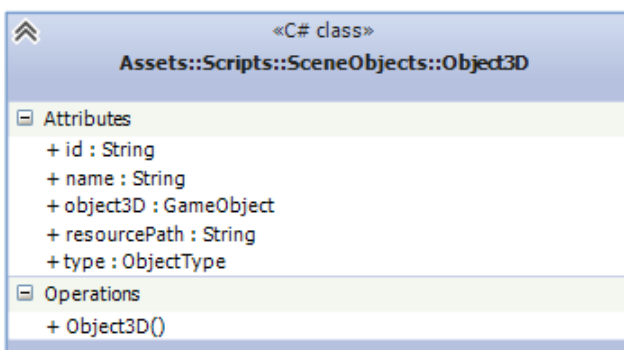
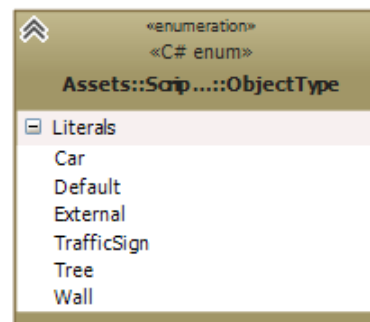
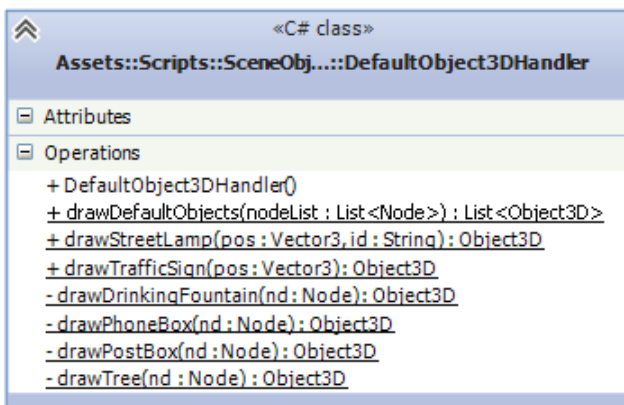
3. Building



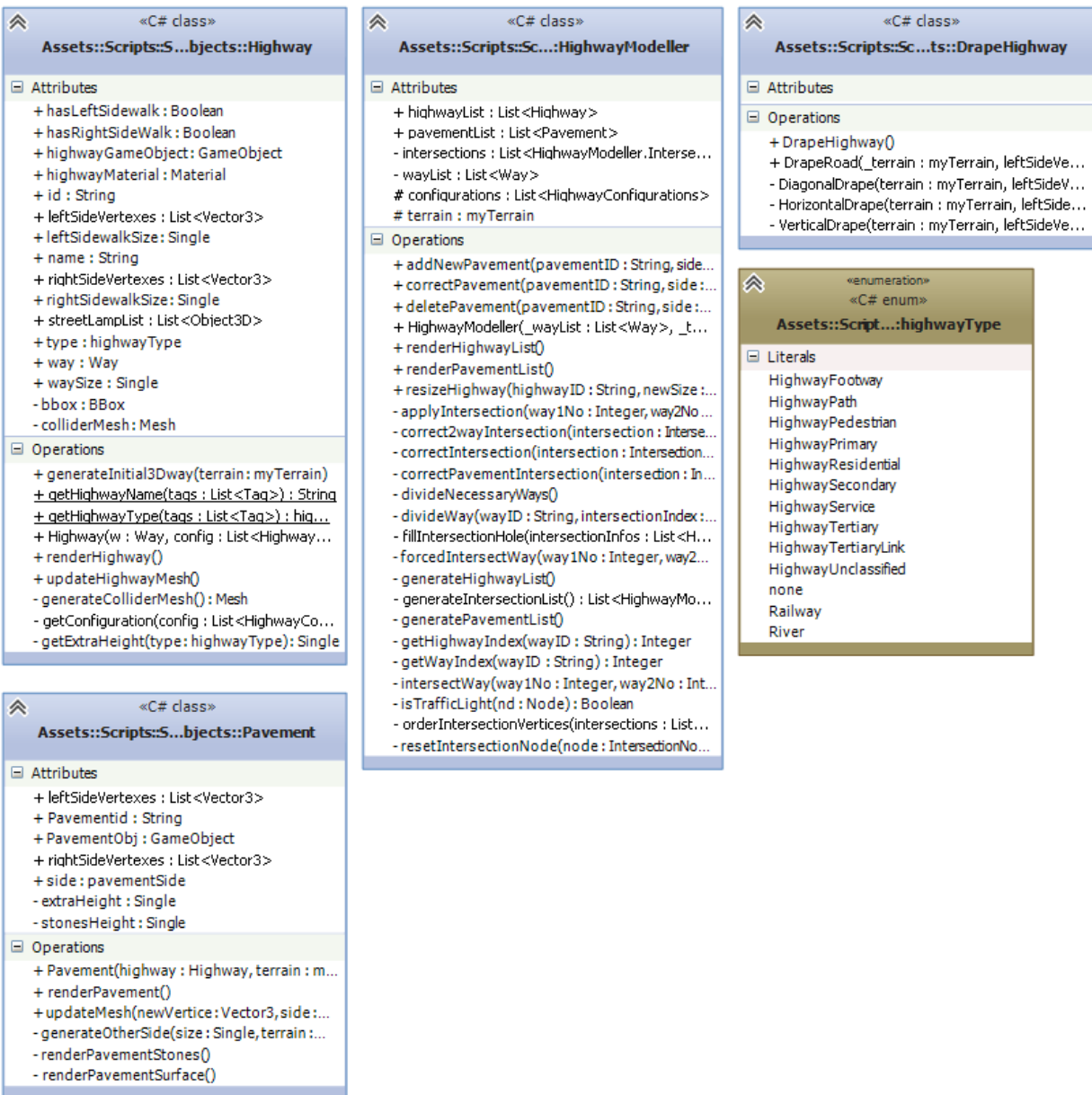
4. Barrier



5. Object 3D



6. Highway



DATA COLLECTOR

```

«C# class»
Assets::Scripts::UnityS...Scripts::ControllerCam
Attributes
+ saveFolder : String
- camera : Camera
- camSetting : CameraSettingItem
- currentLogEntry : LogEntry
- resolutionHeight : Integer
- resolutionWidth : Integer
- rt : RenderTexture
Operations
+ ControllerCam()
+ loadLogEntry(entry : LogEntry)
+ loadSetting(_camSetting : CameraSettingItem)
- generateScreenShotName(saveFolder : String, cameraL...
- OnPostRender()
    
```

```

«C# class»
UnityStandardAssets::...ThirdPersonCharacter
Attributes
- k_Half
- m_Animator : Animator
- m_AnimSpeedMultiplier : Single
- m_Capsule : CapsuleCollider
- m_CapsuleCenter : Vector3
- m_CapsuleHeight : Single
- m_Crouching : Boolean
- m_ForwardAmount : Single
- m_GravityMultiplier : Single
- m_GroundCheckDistance : Single
- m_GroundNormal : Vector3
- m_IsGrounded : Boolean
- m_JumpPower : Single
- m_MoveSpeedMultiplier : Single
- m_MovingTurnSpeed : Single
- m_OrigGroundCheckDistance : Single
- m_Rigidbody : Rigidbody
- m_RunCycleLegOffset : Single
- m_StationaryTurnSpeed : Single
- m_TurnAmount : Single
Operations
+ Move(move : Vector3, crouch : Boolean, jump : Boole...
+ OnAnimatorMove()
+ ThirdPersonCharacter()
- ApplyExtraTurnRotation()
- CheckGroundStatus()
- HandleAirborneMovement()
- HandleGroundedMovement(crouch : Boolean, jump : B...
- PreventStandingInLowHeadroom()
- ScaleCapsuleForCrouching(crouch : Boolean)
- Start()
- UpdateAnimator(move : Vector3)
    
```

```

«C# class»
Assets::Scripts::UnitySid...::ControllerLaserScanner
Attributes
- currentLogEntry : LogEntry
- laser : LaserSetting
Operations
+ ControllerLaserScanner()
+ loadLaserSetting(setting : LaserSetting)
+ loadLogEntry(entry : LogEntry)
- generatePCDname(saveFolder : String, id : Integer) : String
- loadMeshColliders()
    
```

```

«C# class»
UnityStandardAssets::Ch...::ThirdPersonUserControl
Attributes
- m_Cam : Transform
- m_CamForward : Vector3
- m_Character : ThirdPersonCharacter
- m_Jump : Boolean
- m_Move : Vector3
Operations
+ ThirdPersonUserControl()
- FixedUpdate()
- Start()
- Update()
    
```

```

«C# class»
Assets::Scripts::UnitySideScripts::CameraController
Attributes
+ CameraSpeed : Single
+ mode : CameraMode
+ target : Transform
- clampInDegrees : Vector2
- clickedCursorTexture : Texture2D
- isDragging : Boolean
- normalCursorTexture : Texture2D
- sensitivity : Vector2
- smoothing : Vector2
- targetDirection : Vector2
- _mouseAbsolute : Vector2
- _smoothMouse : Vector2
Operations
+ CameraController()
+ cameraUpdate()
- dragScreen()
- freeFlyUpdate()
- Start()
- Update()
    
```

```

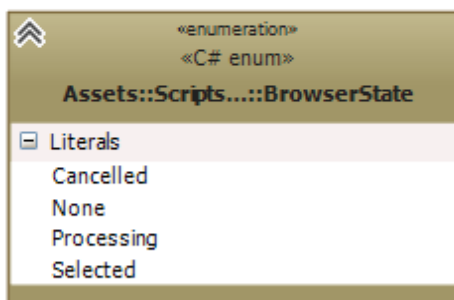
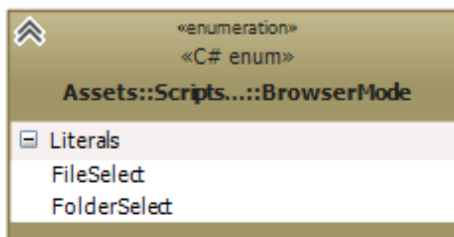
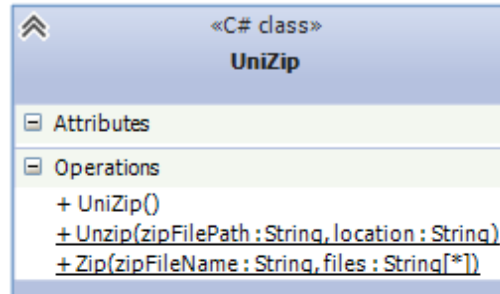
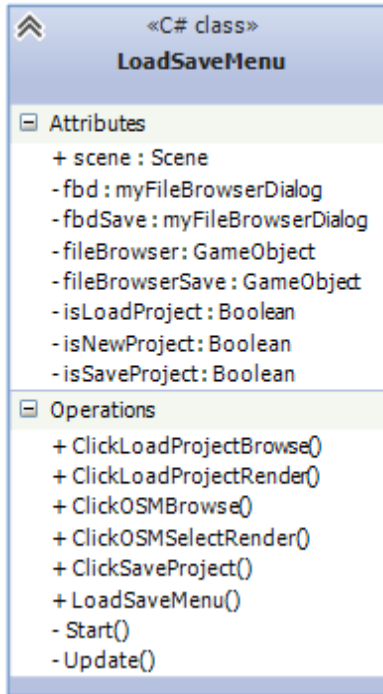
«C# class»
Assets::Scripts...::CarController
Attributes
+ centerOfMass : Transform
+ enginePower : Single
+ wheelColliders : WheelCollider[*]
+ wheelTransforms : Transform[*]
- rigidBody : Rigidbody
Operations
+ CarController()
- FixedUpdate()
- Start()
- Update()
- updateMeshPositions()
    
```

```

«enumeration»
«C# enum»
Assets::Scripts...::CameraMode
Literals
freeFlyMode
recordMode
    
```

USER INTERFACE

1. Load/Save Menu



2. Default Settings Menus

```

«C# class»
Assets::Scripts::UnitySide...s::DefaultSkyboxSettings

Attributes
Operations
+ changeDayTime(_scrollBarObject : GameObject)
+ clickOkay(skyboxmenu : GameObject)
+ DefaultSkyboxSettings()
    
```

```

«C# class»
Assets::Scripts::UnitySide...s::DefaultBuildingSettings

Attributes
- buildingMenu : GameObject
- colorTexture : Texture2D
- colorTexturePath : String
- colorTextureSelect : Boolean
- fbd : myFileBrowserDialog
- fileBrowser : GameObject
- materialList : List<BuildingMaterial>
- normalTexture : Texture2D
- normalTexturePath : String
- normalTextureSelect : Boolean
- specularTexture : Texture2D
- specularTexturePath : String
- specularTextureSelect : Boolean

Operations
+ clickAddSkin()
+ clickCancel()
+ clickColorTexture()
+ clickNormalTexture()
+ clickReset()
+ clickSave()
+ clickSpecularTexture()
+ DefaultBuildingSettings()
- configLoadHelper(config : InitialConfigurations)
- loadConfig()
- Start()
- Update()
    
```

```

«C# class»
Assets::Scripts::UnitySide...us::DefaultBarrierSettings

Attributes
- barrierMenu : GameObject
- fbd : myFileBrowserDialog
- fileBrowser : GameObject
- isTextureChanged : Boolean[*]
- materialPaths : String[*]
- texturePaths : String[*]
- TextureSelection : Integer

Operations
+ ClickCancel()
+ ClickReset()
+ ClickSave()
+ DefaultBarrierSettings()
+ EditTextureClick(skinItem : GameObject)
- configLoadHelper(config : InitialConfigurations)
- loadConfig()
- Start()
- Update()
    
```

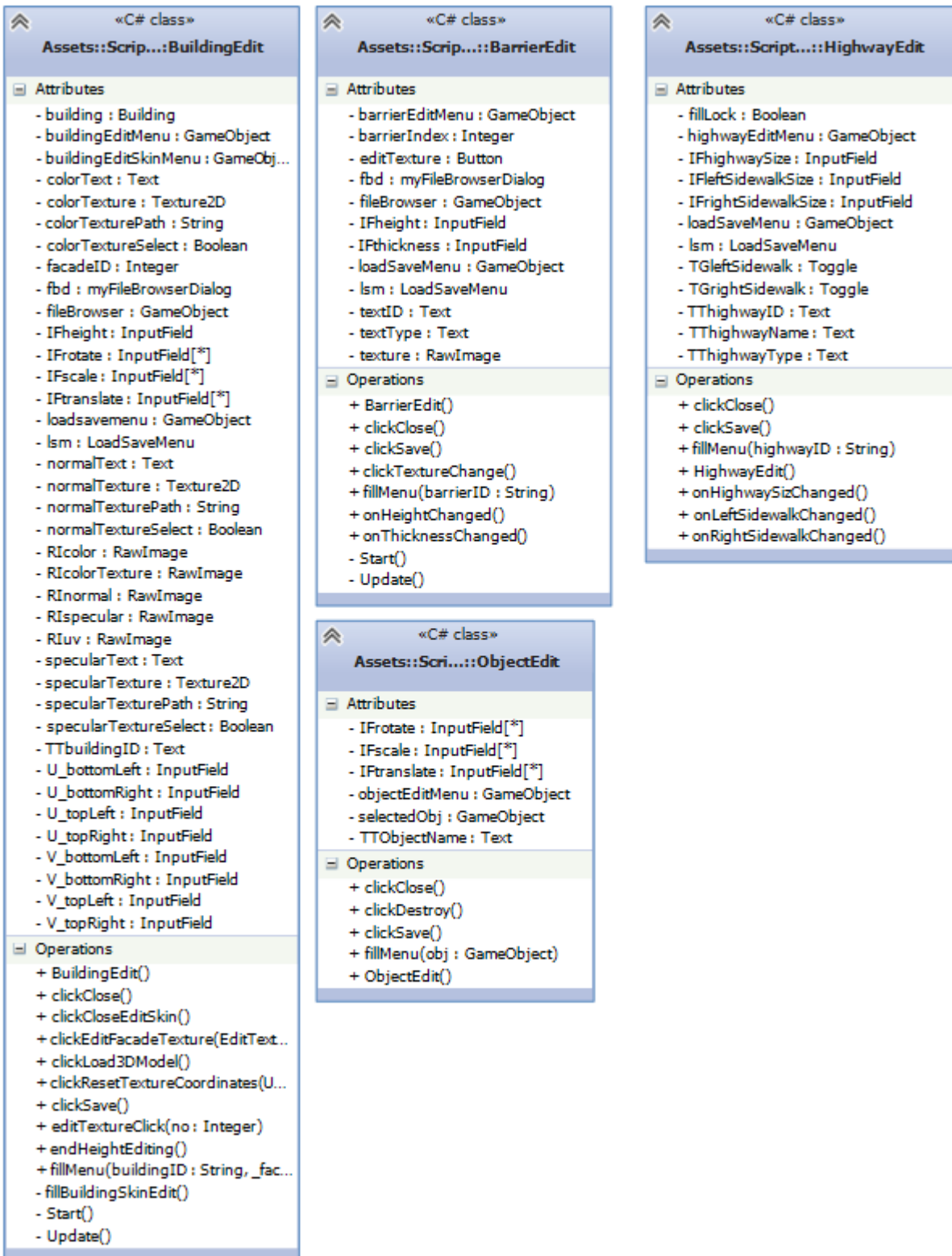
```

«C# class»
Assets::Scripts::UnitySide...::DefaultHighwaySettings

Attributes
- fbd : myFileBrowserDialog
- fileBrowser : GameObject
- highwayMenu : GameObject
- isTextureChanged : Boolean[*]
- materialPaths : String[*]
- texturePaths : String[*]
- TextureSelection : Integer

Operations
+ ClickCancel()
+ ClickReset()
+ ClickSave()
+ DefaultHighwaySettings()
+ EditTextureClick(skinItem : GameObject)
- configLoadHelper(config : InitialConfigurations)
- loadConfig()
- Start()
- Update()
    
```


3. Edit Menus



4. Add Object Menu

«C# class»
Assets::Scripts::Unity...::ObjectTypeSelector

- Attributes
 - + listObject : GameObject
 - objectList : ObjectList
- Operations
 - + barrierClick()
 - + cameraVanClick()
 - + carClick()
 - + newObjectClick()
 - + ObjectTypeSelector()
 - + thirdPersonClick()
 - + trafficSignClick()
 - + treeClick()
 - Start()

«C# class»
Assets::Scripts::UnitySi...ObjectMenu::ObjectList

- Attributes
 - barrierObjectList : List<ObjectList.objectListItem>
 - carObjectList : List<ObjectList.objectListItem>
 - cityRelatedObjectList : List<ObjectList.objectListItem>
 - contentPanel : Transform
 - environmentObjectList : List<ObjectList.objectListItem>
- Operations
 - + clickCancel()
 - + fillBarrierList()
 - + fillCarList()
 - + fillTrafficSignList()
 - + fillTreeList()
 - + ObjectList()
 - fillScrollRect(type : ObjectType)
 - selectItem(prefabPath : Text, name : Text, type : ObjectT...

«C# struct»
Assets::Scripts::...::objectListItem

- Attributes
 - + iconPath : String
 - + name : String
 - + prefabPath : String
- Operations
 - + objectListItem(_name : String, _iconPat...
 - + objectListItem()

«C# class»
Assets::Scripts::Utils::LoadExternalOBJ

- Attributes
 - + materialDefault : Material
 - + MTLwithDminthan1 : Material
 - + objFileName : String
 - MTLon : Boolean
- Operations
 - + LoadExternalOBJ()
 - Start()

5. Data Collector Menu

