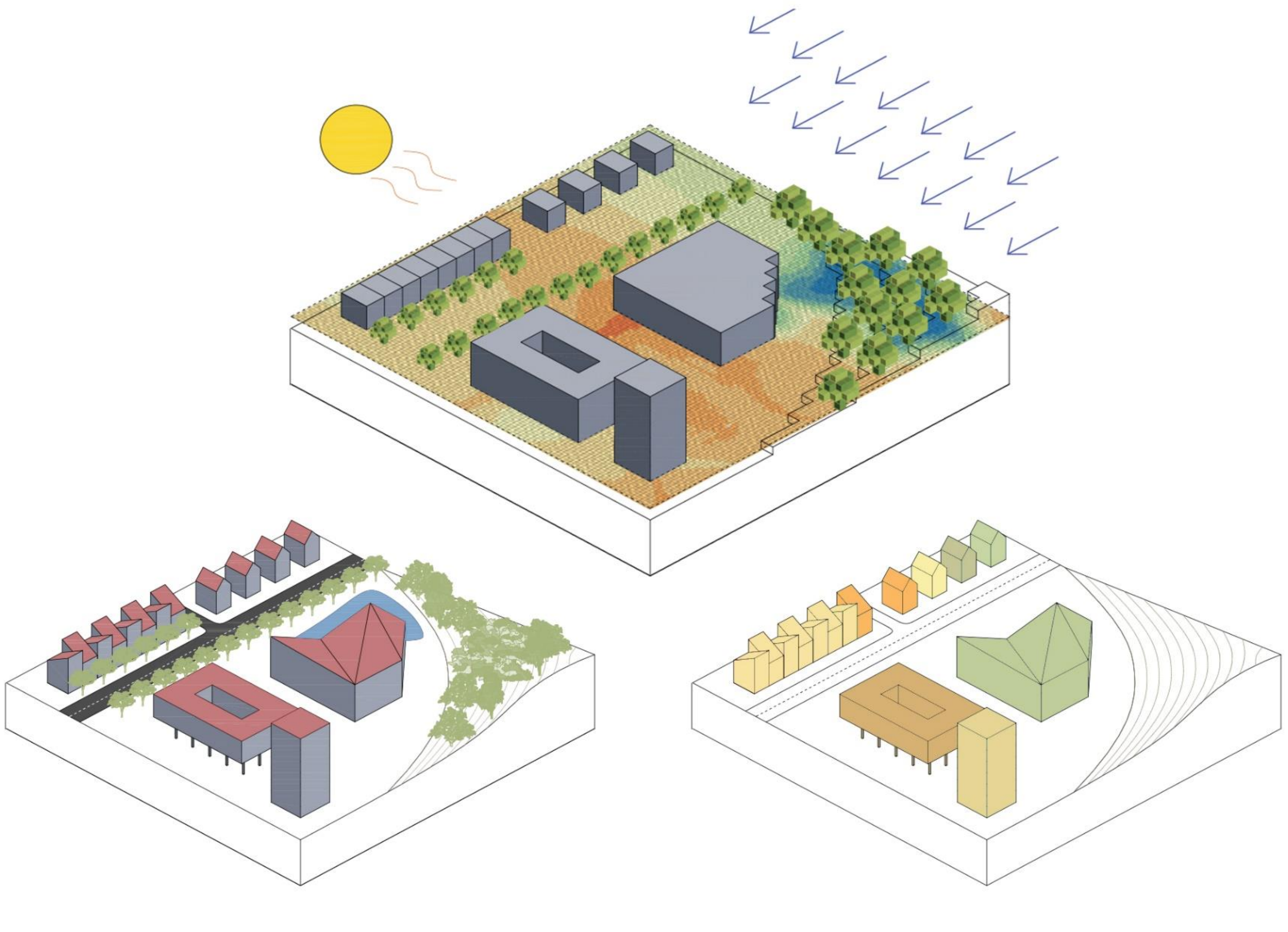
MSc thesis in Geomatics for the Built Environment

The use of 3D digital models in microclimatic studies: First steps in coupling CityGML with ENVI-met

Panagiotis Arapakis
2019



The use of 3D digital models in microclimatic studies: First steps in coupling CityGML with ENVI-met

A THESIS SUBMITTED TO THE DELFT UNIVERSITY OF TECHNOLOGY IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science in Geomatics for the Built Environment

by

Panagiotis Arapakis

October 2019

The work in this thesis was made in the:

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

Supervisors: Dr. Giorgio Agugiaro
Daniela Maiullari
Co-reader: Drs. Wilko Quak

Abstract

In recent years, the need to reduce the global warming of the planet has become more imperative than ever. Global warming and, at local scale, Urban Heat Island phenomena are among the primary effects of the increased building carbon emissions. Nevertheless, understanding and controlling the parameters which intensify, or mitigate, the increasing in temperatures in the surroundings of a building are pivotal, as sustainable design can significantly reduce the buildings' energy demand. Micro-climate simulations can provide more accurate input for building energy simulations since they can accurately simulate the interactions between those parameters to calculate detailed weather data.

Despite the increasing knowledge about the significance of the microclimate, energy simulation users still rely on derived, or interpolated weather data from sparsely located weather stations, located generally outside the urban environment. The reason behind this commonly adopted approach is that the generation of microclimate data is costly in terms of time, and currently standards for storing this generated data have not been developed.

ENVI-met is a microclimatic simulation software that requires a model of an urban area and weather parameters on its boundaries to generate a large extent of data like air, temperature, relative humidity, wind speed etc. Constructing this model manually contains a number of significant limitations, such as high design cost in time and need for data collection from different sources – thus the chance of design errors is high.

In this thesis a novel approach is introduced where the ENVI-met software is used for microclimatic simulations at district scale. However, the input model in this case is created by data extracted from a CityGML-based 3D city model. In addition, the generated microclimatic data is stored back to CityGML, where it can be re-used. The proposed methodology is implemented via a Graphics User Interface, divided in two main phases, serving the required bi-directional data flow. It was designed and implemented based on the following specifications: i) the user involvement in the whole process needs to be minimum, ii) the interface should create simulation-ready input models of various resolutions and iii) it must work with different CityGML datasets.

A data requirement analysis indicated that a CityGML-based city model can feed its data to ENVI-met by the interface, so that the input model required by ENVI-met can be constructed fully automatically. In return, the storage of the generated results data is also possible. Therefore, an automated data flow between a CityGML-based city model and ENVI-met can be achieved, offering the following advantages: i) the ENVI-met input model can be constructed fast and automatically and ii) ENVI-met outputs can be translated to real world coordinates – thus can be visualized and processed in GIS software and ultimately stored back into the CityGML-based 3d city model.

Acknowledgements

The writing of a master thesis is a satisfying, but also a long and lonely affair. Certainly, it would not have been feasible without the aid and support of certain people. I would like to thank everybody here.

First of all, I would like to express my gratitude to my supervisors Giorgio Agugiaro and Daniela Maiullari. Giorgio I would like to thank you for your guidance and your constructive suggestions throughout the planning and development of this project. You helped me digest CityGML and introduced me to the 3DCityDB; the scope of the project might had been wide, but that also helped me developing really useful skills. Daniela, thank you for helping me with ENVI-met and keeping the project on the right track, from the beginning until the end. Also, for your enthusiastic encouragement and support, which was really valuable at moments when I felt lost. I would like also to thank you both for bearing with my abstract ideas and personal-style writing. Understanding me, especially during the first months, must have been a nightmare. I would also like to thank my co-reader, Wilko Quak for his valuable comments and for the information provided in his respective field, during the first year. Writing the SQL queries were one of the few moments I didn’t have to rely on Google.

Furthermore, I would like to mention a number of people that either helped me when I faced a dead-end or inspired me by their work/ way of thinking. First of all, I don’t have words to thank my friend Michalis (Vraxos) for his guidance in programming, through these past two years. Discussing a problem with a skilled programmer and watching the way he approaches it can be more educational than any assignments. The same goes to my friend Ioannis (Touloumpas) for sacrificing his free time whenever I ran for help on writing parts and formatting. Special thanks are of course due to all my teachers and more experienced fellow Geomatics students at TU Delft for the two fascinating years of study.

I would also like to take this opportunity to thank Hans Hoogenboom and Berk Ekici for helping me with the computers in the lab. Also, both Hans and Arno Freeke and all the stuff from the VR lab for their amazing course Beyond 3D Visualization. They allowed me to make some childhood dreams come true, and that was also a wonderful distraction from the second years’ responsibilities.

As always, a big thanks to my family, they were always besides me, believing in me and supporting me in my decisions. Without them, I couldn’t have the opportunity to continue my studies in a field that I was mature enough to select myself. Also, a big hug to all my friends, old and new ones, and fellow classmates; I hope that from now on, I’ll have more time to spend with everybody.

Finally, Lela, you helped me the most; you were the first defense during my anger and uncertainty breakdowns. I don’t have to thank you for the top-level sketches (since you always enjoyed it) but I owe a big thanks for making home (together with M) the place I want to be during my free time. I will return that by animating your amazing landscapes when my respective skills bloom to their full potential.

Panagiotis Arapakis, October 2019

List of Tables & Figures

Tables

Figures

Abbreviations

UHI	Urban Heat Island
OGC	Open Geospatial Consortium
CityGML	City Geography Markup Language
XML	Extensible Markup Language
LOD	Level of Detail
ADE	Application Domain Extension
CFD	Computation Fluid Dynamics
BES	Building Energy Simulation
KML	Keyhole Markup Language
COLLADA	Collaborative Design Activity
glTF	GL Transmission Format
GUI	Graphical User Interface
OSM	Open Street Maps
ID	Identification
Xsd	XML Schema Definition
SRDBMS	Spatial Relational Database Management System
CSG	Constructive Solid Geometry
b-rep	Boundary Representation
CAD	Computer Aided Design
IFC	Industry Foundation Classes
gbXML	Green Building XML
UML	Unified Modelling Language
DEM	Digital Elevation Model
TIN	Triangulated Irregular Network

Table of Contents

1. Introduction

Global warming is one of the most relevant topics of the XXI century. Massive releasing of carbon emissions and vast urbanization that Earth has been experiencing, has already ringed the alarm for humanity. Despite drastic measures and agreements like the Kyoto Protocol (1997) and the more recent Paris Agreement (2015), which set a universal coordinated effort towards the temperatures control, there is still a lot of work to be done (www.afd.fr, 2018). The growth rate of cities in developing countries and the prediction of skyrocketing urban population growth (www.ft.com, 2018) indicate that the battle for sustainable development will be won or lost by the city level of sustainability. At city scale, there is a great potential to achieve significant reduction of emissions, mainly in the building sector (Allegrini et al., 2015), at relative modest cost (Reddy, 2016). This is feasible through a process of energy transition that includes strategies to move from non-renewable energy sources to alternative forms, and to reduce resource consumption per capita. It has been estimated that a potential growth of population will lead to a manifold increase of the urban energy consumption (Gaitani et al., 2011). Furthermore the urban development in China indicates high differences of energy performance among different cities (China Sustainable Cities, 2015), highlighting the importance of sustainable urban planning.

As urban planning is a domain that involves several stakeholders, relevant data need to be stored and disseminated in standard formats in order to make them compatible to widely used tools for urban environmental assessment. However, the derivation of energy related data is a complex and cumbersome process, thereby the integrity and spatial extent of data produced by the existing methods is oftentimes inadequate (Benner, Geiger, & Häfele, 2016). City-wide energy-planning requires an in-depth understanding of the complex system of relations between spatial characteristics, climate conditions and energy flows (Agugiaro et al. , 2015). Both the impact of the Urban Heat Island (UHI), i.e. local increased temperatures in cities, and of the micro-scale processes (interaction of urban fabric, vegetation, water and soil) should be incorporated in buildings energy simulation. It has been estimated that the effect of these factors can lead to deviations of energy consumption results up to ±40% (Gobakis & Kolokotsa, 2017) between expected and real values.

Semantic 3D city models, acting as “digital twins” of cities, can provide integrated and harmonised data for microclimate modelling and thus facilitate the use of more accurate climate boundary conditions for energy modelling. For example, water bodies, which play a big role in UHI simulations, have already been modelled in 3D city models, as well as large amounts of other data describing the building stock, such as building orientation, geometry, materials etc. (Benner et al., 2016). Semantic 3D city models are increasingly used in different cities and countries for a wide range of applications beyond simple visualization. Moreover, city models with the required semantic enrichment serve as integration platforms of knowledge, related to urban and environmental issues, allowing a significant improvement of sustainable management and development (Billen et al., 2014). Semantically enriched spatial data (3D and 2D) are stored and made accessible through common formats that facilitate data exchange and offer interoperability. One of the most widespread open formats is CityGML (City Geography Markup Language), defined by the Open Geospatial Consortium (OGC) for explicitly modelling 3D city models. CityGML is a data model, based on the XML encoding, that defines classes like Buildings, Waterbodies, Vegetation, Transportation, attributes and relations for topographic features, and it comprises properties regarding geometrical, topological, semantic and appearance (Gröger et al.,2012). From a geometrical point of view, it stores the urban data in predefined level of details (LoDs). LoDs are classified in 5 different categories starting from LoD0, which – in the case of the buildings - contains minimal amount of detail (i.e. building footprint) up to LoD4, which contains detailed information (i.e. building interior) (Figure 1).

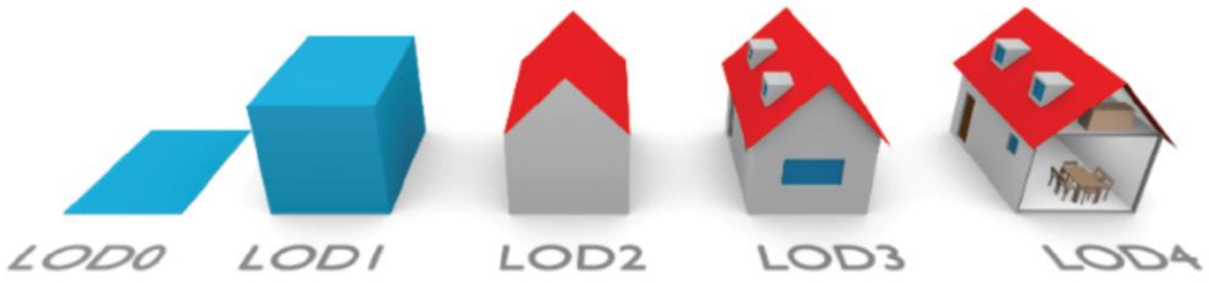


Figure 1: The 5 different CityGML level of details (LoDs) depicted on the Building module, Source: (Biljecki, 2016).

CityGML is an application-independent information model and therefore does not include application specific elements, like for example energy-related ones. However, this is feasible through the Application Domain Extensions (ADE) mechanism, specifically conceived to extend CityGML for additional needs. This can be achieved mainly in two ways: either adding attributes to already existing classes or, alternatively creating new classes with their respective attributes.

Urban energy modelling has seen a progressive development in the last decades, boosted by the shift of the energy transition at city scale level and the evolution of computer hardware (Nouvel et al., 2015). In 2013, the University of Applied Sciences Stuttgart and the Technical University of Munich started working on the harmonization of their data models into a unique extension of CityGML information model, called Energy ADE. The goal of Energy ADE is to support a standard semantic enrichment with environmental related elements and attributes and to provide a city-wide bottom-up energy assessment, mainly focusing on the buildings, their physical properties and systems installed (Giorgio Agugiaro, Benner, Cipriano, & Nouvel, 2018). However, the Energy ADE is currently not supported by commercial software and simulation tools, so an exact determination of which parameters to use is difficult and sometimes possible on a limited way, e.g. heuristically by the user (Benner et al., 2016). In addition, when it comes to larger-scale energy simulations, such as an urban block or a district, the integration of the required detailed local-climate data to Energy ADE comprises a rather challenging task.

Energy simulation and microclimatic simulation tools are becoming more and more popular, driven by change in meso- and local climate. The 3D microclimate model ENVI-met (www.envi-met.com) is a microscale model developed to simulate the interactions between atmosphere and urban surfaces. It does not only focus on a specific parameter, for instance wind speed or radiation flux, but consider the numerous interactions and non-linear correlations among the different aspects of microclimate with a holistic approach. ENVI-met is a software that has been extensively employed in research conducted by architects and urban planners since it can estimate, with high accuracy, parameters like surface temperatures, direct and diffuse solar radiation, mean-radiant-temperature, that are all relevant key parameters for a sustainable city design (Toparlar et al., 2017).

The goal of this thesis is to define and implement a bidirectional data interface to link a semantic 3D city model based on CityGML (and the Energy ADE) and the ENVI-met software. The idea was to exploit the added value of an integrated and harmonised representation of a city thanks to CityGML, and to feed all necessary input data and parameters to ENVI-met in order to perform a micro-climate simulation. Additionally, a selection of meaningful simulation output results was stored back into the 3D city model, in order to enrich it further and to be potentially used by other decision support tools.

1.1 Problem statement

The use of microclimatic simulation software to calculate accurate environmental parameters like air temperature or wind speed is steadily increasing. Toparlar et al. (2017) give a review of the studies where different design scenario analysis included Computation Fluid Dynamic (CFD) simulations and displayed that in the last 3 years, the number of studies equalled the complete number of papers written before that time. In addition, a large part of building energy simulation (BES) software has been developed to perform energy analyses on distinct buildings. They are listed by Allegrini et al. (2015). However, the lack of existing microclimatic data, stored in a standard format do not allow accurate input for simulations, or at least, requires a prior microclimatic simulation on the building’s surroundings. Most studies that employ energy simulation software still use statistically computed weather data, derived, or interpolated from weather stations outside the urban environment. The motivation behind this thesis was that microclimate simulation software should generate reusable data, to reduce the need for further costly iterations and also extend microclimate simulations beyond individual research level. The main obstacle is that, usually, microclimate software supports specific data formats, for input and output, that are not compatible to any standards.

A CityGML-based semantic 3D city model (possibly additionally extended with Energy ADE data) represents a useful hub of integrated energy-related information. Additionally, the energy-specific data can be already stored in an open-source database called 3DCityDB which is implemented as a relational database schema supported by Oracle Spatial and PostgreSQL/PostGIS (3D Geodatabase for CityGML, 2018). Data stored in a relational database such as 3DCityDB is easily accessible and facilitates development of new application programs. Since version 3.3.2, the 3DCityDB was extended to include some ADE data model (i.e. from Energy ADE, Utility Network ADE and Scenario ADE). The 3DCityDB software package includes an importer that can automatically store a valid CityGML file into the pre-defined database schema, query tools, and an exporter that can also extract the spatial information as 3D visualization models in KML, COLLADA, and glTF formats. Potential fill/enrichment of 3DCityDB with Energy ADE data, resulting from microclimatic simulations would be a relevant step towards the further adoption of CityGML for urban energy simulations.

As far as ENVI-met is concerned, as a CFD tool it requires a 3D grid input model for calculations. Normally, such model has to be constructed *inside* ENVI-met by means of a specific 3D designer interface GUI called SPACES. This generally requires data collection from different sources, and in case of large, high resolution models, high design cost in time. Recently, the software has been updated to support vector data as input. The latest ENVI-met version 4.4, includes a module called ENVI-met MONDE, that provides an interface for selecting OSM data or shapefiles as input. Additionally, the software developer Di Nunzio has recently released a set of components for Ladybug in order to couple Rhino geometry with ENVI-met (www.grasshopper3d.com, 28/12/2018). However, both approaches have not reached the point of eliminating human intervention, as they don't provide the required semantic interoperability. Therefore, users still need to familiarize with additional software interfaces, and prepare or collect input data.

CityGML and ENVI-met information model specifications are unrelated. The required coupling was first of all a mapping of the data models between them. In order to transfer data, two-way correspondences had to be established. This procedure requires precise knowledge of both data models, in order to define matching and processing techniques of the data to be transferred. With particular regard to the geospatial domain, conversion among different spatial data models plays a major role. Examples are the conversion between 2D/3D vector data (typical of CityGML) and 2D raster or 3D grid data (typically used in ENVI-met). Other useful spatial operations to perform data transformation such as aggregations included the extraction of voxel centroids to create a point cloud, the extrusion of building facades and point aggregations on the extruded volumes, which are indeed typical of the geomatics domain.

This master thesis tried to bridge ENVI-met and CityGML by designing a bidirectional interface. It aims to facilitate ENVI-met use by skipping the manual model design *inside* ENVI-met and generating it instead automatically with data from the semantic 3D city model. In addition, by using the real-world coordinates of CityGML as an input, the interface is able to utilize the same model for storing useful generated microclimatic data.

1.2 Objectives and research questions

The goal of this thesis was to couple the CityGML data model with the ENVI-met simulation software, in order to exploit the geometrical and semantical CityGML data and feed it to ENVI-met and perform microclimatic simulations. In this way, microclimatic data can be generated and stored automatically. Such data can be for instance, spatially and temporally high-resolution air temperature, relative humidity, wind speed.

A second goal of this thesis was to analyse ENVI-met output data and to store it back to the initial 3D city model, in order to enrich it. This requires that ENVI-met results be transformed in order to be compliant with CityGML and the Energy ADE.

Based on the introduced sections, the main question of this master thesis is:

<u>How and to which extent is it possible to establish a bidirectional information flow between CityGML + Energy ADE and ENVI-met?</u>

To achieve this, the following (sub)questions are relevant:

a) Regarding the coupling between CityGML + Energy ADE and ENVI-met:
    - Which are the data requirements of ENVI-met?
    - Can they be met by CityGML (+ Energy ADE)?
    - How to perform the mapping between them in terms of data model?
        - How to implement the interface that reads/writes data?
    - What is needed in order to enrich the CityGML model with simulation outputs?
        - Should another ADE be needed?

For each of the previous (sub)questions

- Where do the main difficulties lie?
- Could they be overcome by modifying and extending the current Energy ADE data model?

b) Regarding the testing of the methodology within the case study in Almere:
    - How to gather, integrate and harmonise the data to prepare the 3D city model of the test area for the status quo situation?

- How to define and characterise the other simulation scenarios?
- How to generate the models of the different scenarios previously defined?

To successfully complete the mapping, two main obstacles have to be overcome. The first one is the difference between ENVI-met and CityGML data models and file formats. From a geometrical point of view, in order to generate ENVI-met input data, a prior voxelization of the vector- and boundary-representation-based) CityGML data is required. The second obstacle is to keep the geometrical operations and apply a reverse transformation to return ENVI-met results back to the real coordinates. An alternative is to preserve semantics and IDs during the whole process, from data preparation and conversion, through simulation in ENVI-met, and till the post-processing of the results. In CityGML, every city object is identified by a unique ID. IDs are the obvious way to find the correspondences between ENVI-met output files and the CityGML model. However, the preservation of the IDs associated to each object required by ENVI-met had to be examined.

1.3 Case study

The case study used in this research is a new Floriade district in the city of Almere, Netherlands. The project, developed by the municipality, is designed around the concept of green village and it will be implemented in two phases. In the first phase, till 2022, the area will host an international agriculture exposition, and after the event it will become a residential district. The boundary dimensions of the under-development district are approximately 300x450 m and the boundary dimensions of the surrounded influenced area are around 1000x1000 m. A number of green infrastructures will be constructed during the next years, and the road network system will include arboretums with grass and planted trees. At this moment, the municipality (www.2022almere.nl, 01/01/2018) has a preliminary masterplan of the area which has provided spatial data for this thesis. However, except for the existing and recently planted trees, the majority of infrastructure is not implemented (Figure 2).

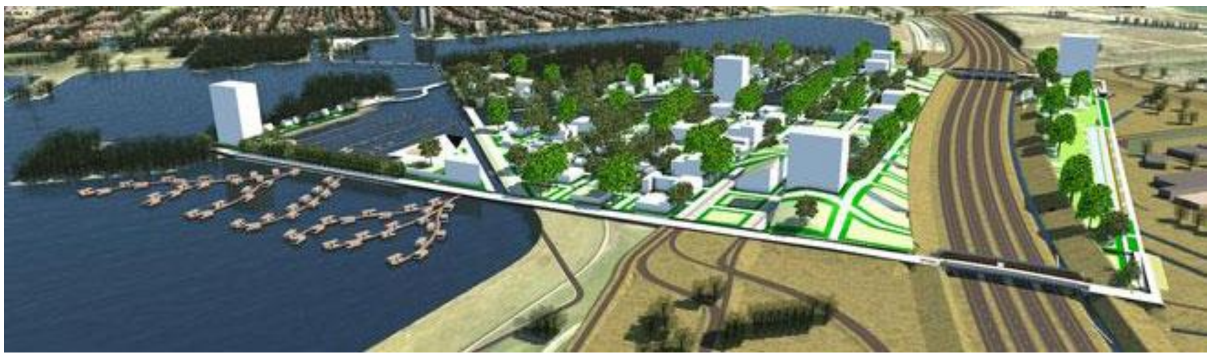


Figure 2: A digital 3D model of the Green Village in Almere, Source: (www.2022almere.nl, 2019)

The results of this thesis are expected to provide an insight on the microclimate of the Floriade area, which can further support a more accurate estimation of the energy performance of the new urban development.

1.4 Methodology and significant findings

This master thesis focuses on i) The formal definition of a methodology to bi-directionally link CityGML with ENVI-met by means of a mapping between the two respective data models and ii) To implement this link in form of a bidirectional software interface allowing for transfer of data. This section outlines the methodology for the execution of this project, which is presented in the following graph (figure 3)

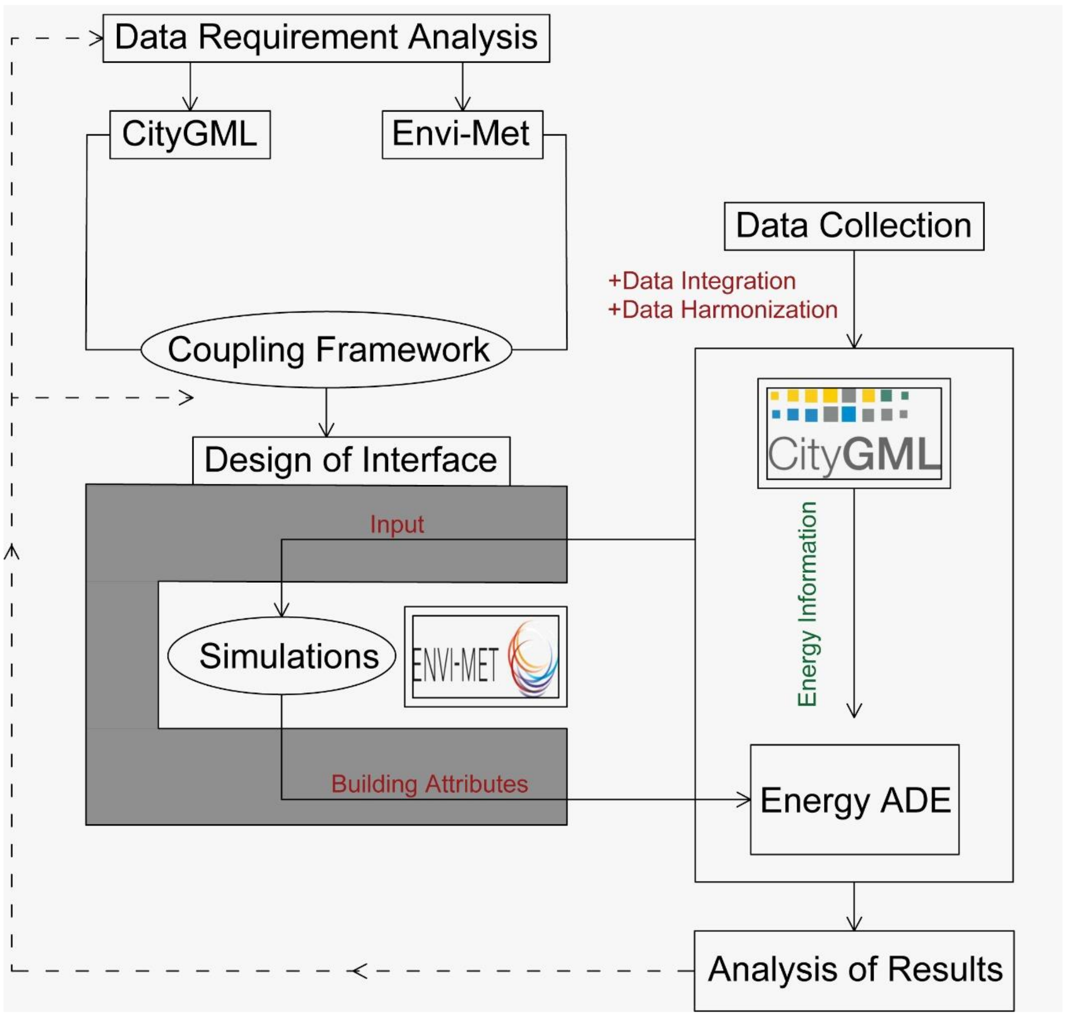


Figure 3: Methodology graph

1.4.1 ENVI-met

ENVI-met software is publicly available on the web page (www.envi-met.comThe "Science version" was used in this thesis. The web page includes a forum, further documentation and an online repository which aid gaining a general understanding of its functionalities. Additionally, some already generated input files as well as the outputs of their simulation are provided as example data. ENVI-met depicts urban structure in layers of dem, surface materials, buildings and vegetation, to form its 3D model area input file. These elements represent the same features of important significance in semantic 3D City Models.

1.4.2 CityGML + Energy ADE data model

CityGML is a rich standard, both on the thematic and geometric-topological level of its data model. On its thematic level, CityGML defines classes and relations for the most relevant objects in cities (Gröger et al., 2012). Objects are classified in 9 main classes (thematic modules) which includes, Buildings, Vegetation, Transportation, Water etc, for which a direct correspondence to the previously referred ENVI-met layers is expected. CityGML is structured with a strict hierarchy which is formalised in its XML schema definition (XSD) file. An XSD file can be used as a reference, both for providing aid in analysing the data structure, for the data mapping and for validating the generated models (Stadler & Kolbe, 2007).

1.4.3 Scope

This thesis examines the extent of the automatic flow of data between the CityGML data model and ENVI-met inputs and outputs, by creating a mapping framework. At the starting moment of this research, there were no reports regarding a previous coupling attempt between them. The general aim is to enable a data flow, by creating an applicable methodology that will provide also a basis for future extensions. This project by any means doesn't try to cover all the optional requirements of ENVI-met, neither include an implementation that maps *all* data from CityGML that can be theoretically transferred, to create a more detailed input for ENVI-met. Also, in the time frame of a master thesis, it is out of scope to examine the whole extent of data that could be transferred back to CityGML, since data structure in ENVI-met output differs between groups of generated parameters. Last, this study doesn't propose an algorithm for modelling buildings according to the full 3D area input file of ENVI-met (it will be discussed on section 2.3 and 3.3).

Furthermore, this thesis will not focus on the results of the microclimatic simulations conducted on the Floriade district; it will only provide numeric and visualized results over 2 different scenarios and 3 test simulations that were selected after a mutual agreement with the people from the municipality of Almere. Last, this project will not further concern with the validation of ENVI-met outputs, however it will include a filtering process in the second phase's methodology.

1.4.4 Methodology

The coupling is established by means of a bidirectional interface that is designed to feed the necessary data from a 3D model based on CityGML to ENVI-met. In the first place, a research on the data requirement analysis of ENVI-met inputs took place. Afterwards raw data was provided by the municipality of Almere and harmonized to CityGML in Safe Software's FME. The required meteorological data for the simulations was derived by the closest weather station, Lelystand. Sequentially, the interface that reads and writes data was created in Python programming language. It first takes a CityGML dataset, stored in a 3DCityDB instance, parses it and feeds the necessary data to ENVI-met, in order to perform microclimatic simulations. This process will be referred as the first phase. When the simulation is completed, the interface reads ENVI-met results, picks useful data (as defined by the user) and writes it back to the 3DCityDB. This process will be referred as the second phase. The second phase requires a geometrical translation from local cartesian coordinates of ENVI-met to real world coordinates (metric reference system of CityGML) and research over compatible classes and attributes that can be used as a hub for the generated data. The results of the second phase will be validated by a number of visualizations. Finally, the analysis of results will indicate ENVI-met limitations and potential changes in the CityGML/ Energy ADE data model in order to satisfy microclimatic applications.

1.4.5 Significant Findings

The result of this thesis is a designed interface called SclGModeLer that can map CityGML datasets to ENVI-met area input files and return atmospheric parameters back to the input dataset, as *Time Series*-based *Weather Data* of the Energy ADE. The implementation of the proposed research questions has been carried out successfully, and district scale, automatic enrichment of CityGML is now possible, with SclGModeLer. Furthermore, ENVI-met results were converted to real world vector geometries and can also be visualized and processed in GIS software. For the scope of this study, a CityGML Floriade model was also created by raw data, which after the implementation test is now enriched with Energy ADE data, and is expected to support decisions for future development. Different scenarios were also simulated in order to acquire an image of the cooling performance of trees that were planted in the district.

1.5 Thesis Outline

The rest of this master thesis report is organized as the statements indicate.

- Chapter 2 provides an introduction on the basic theoretical background behind CityGML, ENVI-met and 3DCityDB. It is absolutely required, before trying, to locate the correspondences between the two data models that are required to perform the mapping. Furthermore, the evolution of ENVI-met through its major versions will be presented to acquire an overview of the increasing demands in resolution that microclimate simulation requires. In addition, it discusses the coupling techniques and software tools used, to automatically generate and store weather data, by using 3D models, alongside their limitations. It will further explain the motivation behind the conduction of this dissertation, and which gaps it aims to bridge.

- Chapter 3 presents the structure of ENVI-met inputs and output formats. Furthermore, it presents the CityGML standard, in terms of geometry-topology, and semantics and goes through each thematic module of CityGML and Energy ADE, and searches for data that ENVI-met requires..
- Chapter 4 analyses the coupling framework, both in conceptual and technically implemented way. It will also introduce SclGModeLer, the Graphics User Interface (GUI) that was created to embrace the methodology and make it really flexible and easy to use.
- Chapter 5 includes the steps followed to harmonize raw data received from the municipality of Almere and integrate it into a Floriade CityGML dataset. Then screenshots and impressions from the usage of the methodology over this dataset are included in this chapter, as a visualization of the results as well as for proof of concept.
- Chapter 6 presents the conclusions derived by this study and potential future work.

2. Theoretical Background

This chapter provides the theoretical background of the topics, related to the thesis. The first part will present 3D modelling and the needs for semantic 3D models in various applications. It will focus on CityGML and its extension methods. Furthermore, it will explain the need of a Relational Database Management System (RDBMS) for the storage and management of spatial and non-spatial data. The second part will explain the role of energy and microclimatic simulation software and outline their application. An abstract of the ENVI-met software will be provided, presenting its functionality, limitations and gradient evolution through the main versions. The third and last part will discuss previous coupling attempts and their results. It will provide an insight on the current state of the art, and more specifically on software that have been used in various other studies to perform simulations and generate energy-related results. The different methodologies will be presented, along with tools that have already been involved in the process of translating information stored on the respective data models.

2.1 Modelling the urban world

Cities are growing with a fast rate and the demand for utilization of 3D city models is increasing, as the geospatial data becomes more and more available (Baig and Rahman, 2012;Gartner et al., 2009). City models can be mainly divided between two different formats. The first is the continuous vector representation, that used points, lines and polygons to form more geometric complexes to model entities, and the second is the contiguous cell representation, known as raster or voxel. Raster based models are easier to construct, since they require less amount of data, however this type of space representation do not allow to reach the same level of accuracy when it comes to application (Leao et al., 2017).

In the early days of their utilization, 3D models were meant to be used for mere visualization purpose. The probably most important representation of vector-based 3D models is the constructive solid geometry (CSG) which is supported by most 3D CAD software and mainly used in architecture, civil engineering and also gaming industry. Geometries of this type are stored in a binary tree (Figure 4), where leaves hold the geometry primitives and the internal nodes contain the main transformations (Coros, n.d.) . In 3D CAD software, geometries can be grouped in layers so they can be somehow classified.

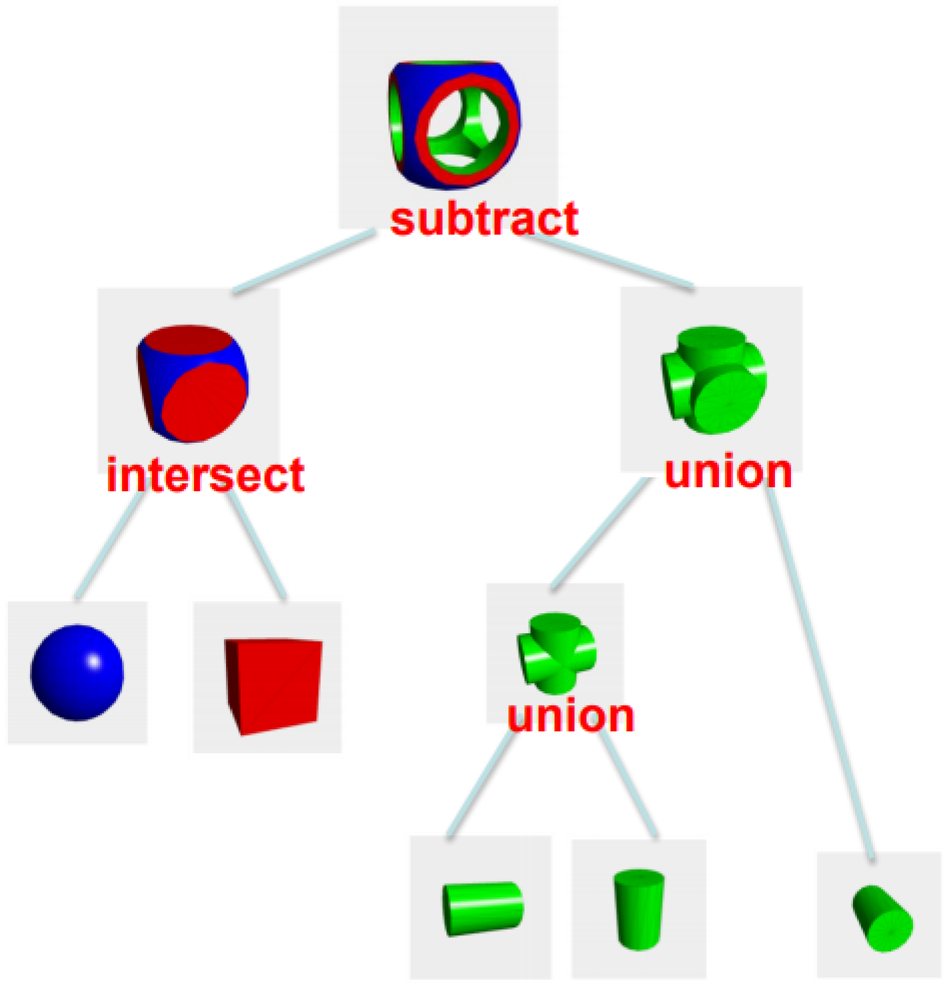


Figure 4: CSG modelling of objects, Source: (Coros, n.d)

However, the need for utilising 3D models for an increasing number of tasks, like noise mapping, disaster management or microclimatic applications, makes them more evident but also demanded modifications on their data model approach (Eriksson, Harrie, & Paasch, 2018). For example, in addition to 3D geometry and appearance, these application requires a clear semantic distinction between geometries, representing different feature and objects (Stadler & Kolbe, 2007).

The semantic modelling of the built environment today requires mainly two different approaches, regarding its potential applications. It can be divided in building-scale level and city-district level. The typical application for the wider city-district scale is the 3D Geographic Information Systems (3D GIS) and for the lower scale the Building Information Models (BIM). The dominant standards are CityGML for 3D GIS and Industry Foundation Classes (IFC) and green building XML (gbXML) for BIM (Agugiaro et al., 2018; Kardinal Jusuf et al., 2017). These two different modelling approaches have some overlap (for the building domain that is the room level) and can therefore be integrated, giving the opportunity to users to combine their value in diverse datasets to enable various applications (https://www.geospatialworld.net, 2019).

Central to City Models is the concept of Level of Detail (LOD). The concept of LOD was initially introduced in the computer graphics domain, where the complexity of objects decreases as they move away from the viewer to facilitate faster rendering of scenes. The LOD in 3D city models, despite defining the geometric complexity, defines also the level of appropriation of the model for particular applications (Biljecki et al., 2016). In semantic 3D city models with an increasing LOD objects become both geometrically and semantically richer. However higher complexity of city models usually results to more effort and higher cost to generate them, and they are prone to errors (F. Biljecki et al., 2016). In general terms, the fully (or semi-)automatic creation of a semantic virtual city model requires the heterogeneous input datasets to be sufficiently "clean" and properly structured, before moving towards the actual data integration process (G. Agugiaro, 2016).

2.1.1 The CityGML standard

CityGML is a data model and storage format maintained by the OGC for storing and representing 3D city models [Open Geospatial Consortium, 2012b]. It was developed to have a standardised way of defining the entities, attributes and relations of 3D city models [Open Geospatial Consortium, 2012b]. Spatial properties of the CityGML features are represented by objects of GML3's geometry model. This model is based on the Standard ISO 19107 spatial schema, representing geometry as Boundary representation (Gröger et al., 2012) in a metric 3-dimensional coordinate system. For each dimension an abstract geometry primitive type is used which is _Point for 0-dimension, _Curve for 1-dimension, _Surface for 2-dimension and _Solid for 3-dimension. Despite the fact that curves are allowed in GML3 geometry model, in CityGML, 2 curves are restricted to be made piecewise of straight segments, hence the GML3 _Linestring class is used. Furthermore, surfaces in CityGML are restricted to be planar. Curves and surfaces have a default orientation in GML which results from the order of the defining points. The semantic model of CityGML employs the ISO TC211 family standards for the modelling of geographic features. According to it, geographic features are abstractions of real-world objects and they are modelled by classes. CityGML divides the most important urban features in 9 main classes/modules, plus city object groups (Figure 5). One of the most important design principles for CityGML is the coherent modelling of semantics and geometrical/topological features. This whole concept makes CityGML very suitable for storing 3D models of various LODs in one single dataset. The data models of every class are conceptually defined by series of Unified Modelling Language (UML) class diagrams.

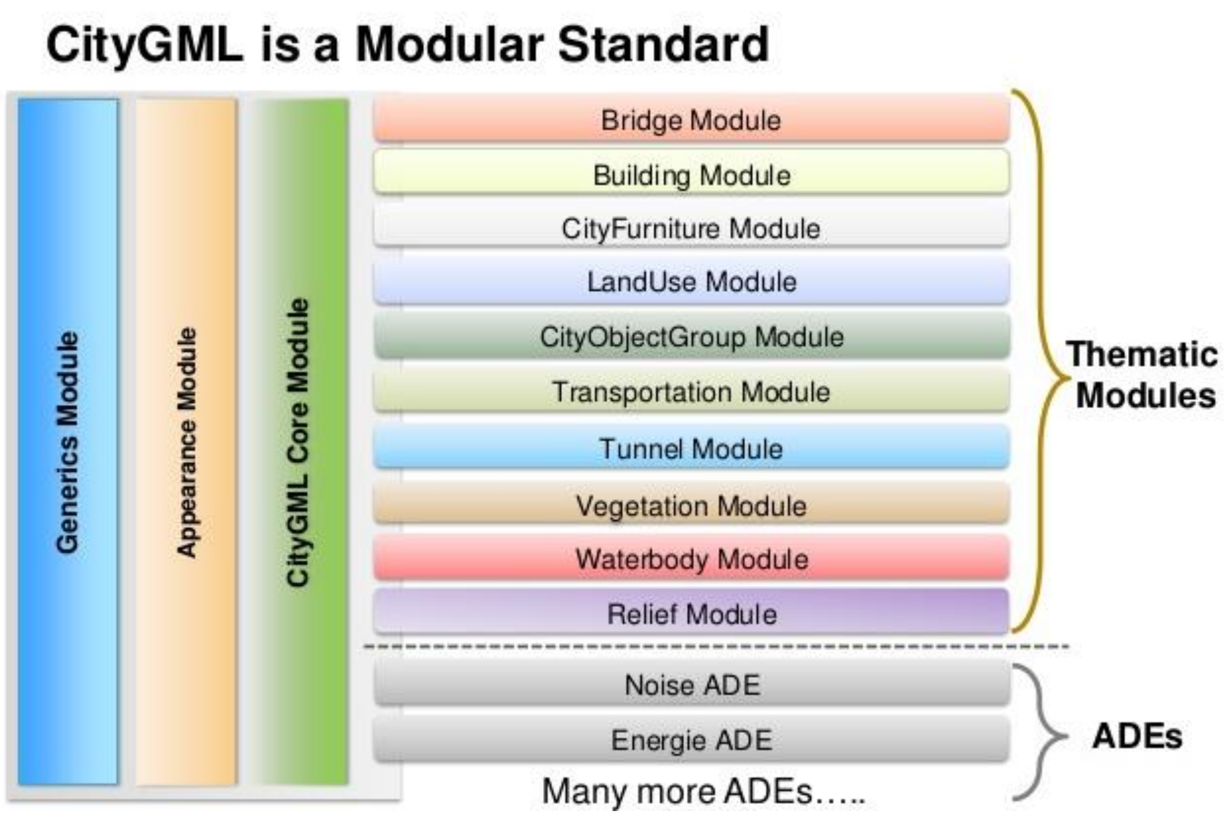


Figure 5: Modular structure of CityGML

2.1.2 Extending CityGML for application needs

As mentioned in the introduction, CityGML is by design an application-independent information model and doesn't include additional information that are required by specific applications (Filip Biljecki, Kumar, & Nagel, 2018). As an example of that case is the application on the energy domain. However, the CityGML data model is designed in a flexible way that enables extensions, according to different needs. These extensions can be materialized in two ways, first, by adding *GenericCityObjects* or extend existing classes properties by *GenericCityAttributes*. This *Generics* module provides a general extension mechanism for CityGML, where arbitrary city objects may be defined according to user-defined criteria and further classified by specific attributes (Gröger et al., 2012). Secondly, by the concept of Application Domain Extensions (ADE). ADE is a mechanism for enriching the data model, while preserving the hierarchy order of classes, and the semantic structure of CityGML. It is a very important component of CityGML, that's why it has been implied from the very early versions (v 0.4). An ADE, similar to CityGML may be specified by UML diagrams, and the accompanying XSD schemas when it comes to the XML-based encoding rules. Some relevant aspects according to Kumar, Ledoux, & Stoter (2016) regarding ADEs are that:

- An ADE requires its own namespace in order to avoid conflicts with the rest of CityGML thematic modules.
- An ADE may extend multiple thematic modules of CityGML at once.
- A dataset may contain more than 1 ADE.
- An ADE can define new spatial representation for the existing CityGML features.
- ADE classes and attributes can have their own code lists.

To support the energy transition process on city scale, bottom-up Urban Energy modelling has experience a steady rise during the last decades (Agugiaro et al., 2018) followed by the need of emerging standards in order to achieve interoperability between different stakeholders. The goal of Energy ADE is to provide a unique and standard model, for all the involved stakeholders (CityGML users, software vendors, scientists and data producers) by following city-wide bottom-up energy assessment, with particular focus on the building sector. Following the philosophy of application independency of CityGML, the Energy ADE aims to be flexible in terms of compatibility with different data quality and levels of detail, and compatibility with different urban energy model complexities. Thus, it comprises 5 main modules and relative classes to provide data to access the energy performance of buildings. Currently Energy ADE, after experiencing a fast development, has reached version 1.0. Version updates are currently paused, to allow the development of applications, focusing on automatic generation of Energy ADE datasets, so that Energy ADE can be further adopted as a standard.

2.1.3 Storing CityGML in a relational database

CityGML can be considered a combination of two things. First, the feature catalogue and data model for the thematic modules discussed in the previous sections. Secondly the aforementioned data model, mapped to an XML-based exchange format using OGC's Geography Markup Language (GML). However, files are not the optimal way for data management. The 3DcityDB is an open source package, consisting of a database schema with over 60 related tables, which is designed for the commercial SRDBMS Oracle and the open source SRDBMS PostgreSQL, as well as of an interface equipped with tools to import, export, analyse and visualize CityGML-based 3D models https://www.3dcitydb.org/3dcitydb/. As it was mentioned in section 1.2, 3DCityDB was already extended to include some ADEs. Since CityGML is a well-structured, related data model, a database schema facilitates data retrieving or storing-updating. In general PostgreSQL has a number of considerable advantages, over files and Oracle (Shukla et al., 2016) when it comes to the management of spatial data.

First of all, it provides native support on geometrical-topological relationships, with the PostGIS extension. PostGIS can be a very strong tool when retrieving data or apply geometrical and topological operations like buffering and intersections between different geometries for two reasons. The first is that it includes almost every kind of 2-dimensional and 3-dimensional methods (https://postgis.net/docs/manual-1.5/ch08.html) so there is no need for additional libraries and secondly because of the native indexing support. With a simple command (which is already implemented into 3DCityDB geometry tables), PostgreSQL uses an R-Tree index implemented on top of an GiST one, to structure GIS data (http://postgis.refractions.net/, 2019). The R-Tree index splits space and breaks data into rectangles (leafs) and thus speeds up searching, especially when it comes to large datasets, since it doesn't have to go through sequential scanning. (Figure 6).

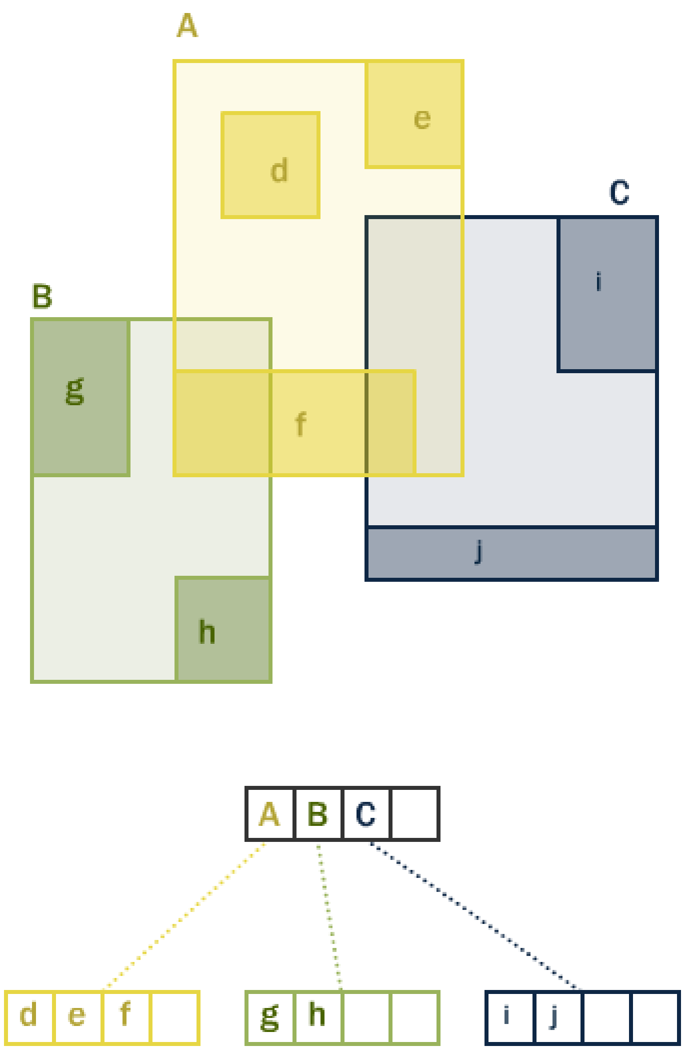
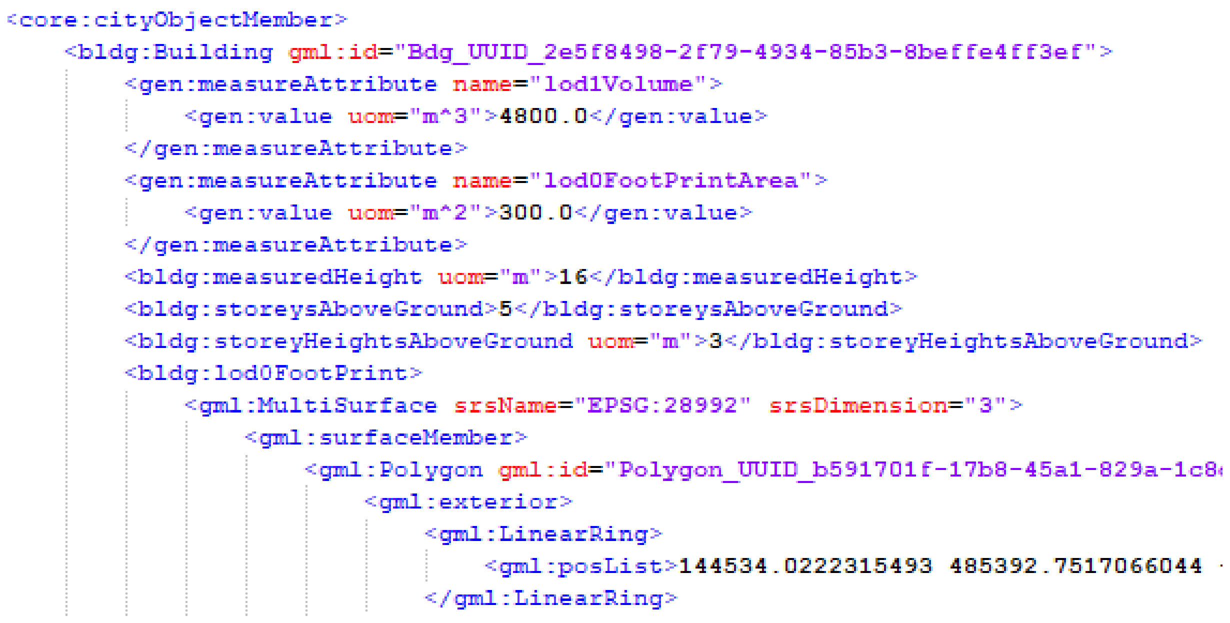


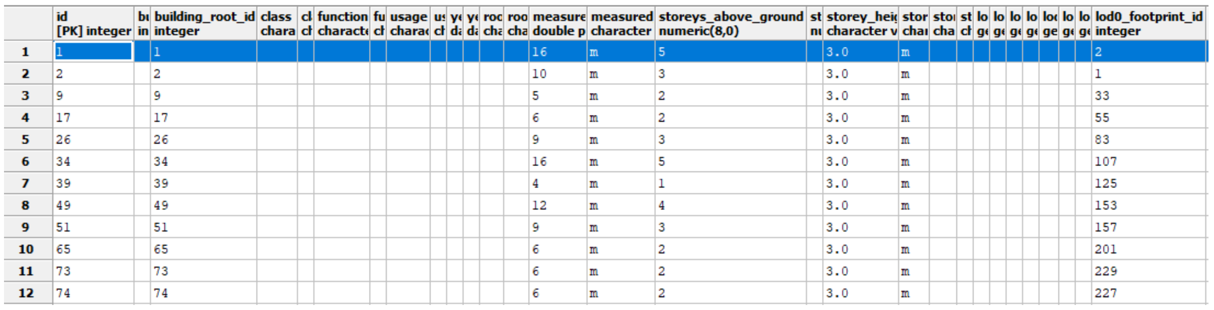
Figure 6: R-Tree index for efficient spatial searching

Furthermore, PostgreSQL is supported with a pre-designed querying language (SQL) for data retrieving. Objects can be filtered during import or export according to spatial regions (bounding box), their object IDs, feature types, names, and levels of detail. At the same time the RDBMS provides easy CRUD (Create, Read, Update, Delete) methods that do not require manually metadata updating. Especially when using the 3DcityDB, where all tables are related, following the CityGML model hierarchy rules, the only concern is to fill or update the pre-designed tables, thus saving a lot of time and effort. At the same time, at least from a personal point of view, the table structure of 3DcityDB is far easier in navigation than the XML format (Figure 7, Figure 8).



```
<core:cityObjectMember>
    <bldg:Building gml:id="Bdg_UUID_2e5f8498-2f79-4934-85b3-8beffe4ff3ef">
        <gen:measureAttribute name="lod1Volume">
            <gen:value uom="m^3">4800.0</gen:value>
        </gen:measureAttribute>
        <gen:measureAttribute name="lod0FootPrintArea">
            <gen:value uom="m^2">300.0</gen:value>
        </gen:measureAttribute>
        <bldg:measuredHeight uom="m">16</bldg:measuredHeight>
        <bldg:storeysAboveGround>5</bldg:storeysAboveGround>
        <bldg:storeyHeightsAboveGround uom="m">3</bldg:storeyHeightsAboveGround>
        <bldg:lod0FootPrint>
            <gml:MultiSurface srsName="EPSG:28992" srsDimension="3">
                <gml:surfaceMember>
                    <gml:Polygon gml:id="Polygon_UUID_b591701f-17b8-45a1-829a-1c8
                        <gml:exterior>
                            <gml:LinearRing>
                                <gml:posList>144534.0222315493 485392.7517066044 ·
                            </gml:LinearRing>
```

Figure 7: CityGML XML encoding of class Building



	id [PK] integer	bu in	building_root_id integer	class chara	cl ch	function characte	fu ch	usage charac	us ch	ye da	ye da	roo cha	roo cha	measure double p	measured character	storeys_above_ground numeric(8,0)	st nu	storey_hei character v	stor char	stor cha	st ch	lo ge	lo ge	lo ge	lo ge	loc ge	lo ge	lo ge	lod0_footprint_id integer
1	1		1											16	m	5		3.0	m										2
2	2		2											10	m	3		3.0	m										1
3	9		9											5	m	2		3.0	m										33
4	17		17											6	m	2		3.0	m										55
5	26		26											9	m	3		3.0	m										83
6	34		34											16	m	5		3.0	m										107
7	39		39											4	m	1		3.0	m										125
8	49		49											12	m	4		3.0	m										153
9	51		51											9	m	3		3.0	m										157
10	65		65											6	m	2		3.0	m										201
11	73		73											6	m	2		3.0	m										229
12	74		74											6	m	2		3.0	m										227

Figure 8: 3DcityDB table of class Building

2.2 Energy and microclimatic simulations

The modelling of urban thermal conditions can be performed at different scales, namely the micro-scale and the meso-scale, depending on the desired parameters and the area under investigation. The tools that support the calculation of building energy demand by detailed building energy simulations are called Building Energy Simulation (BES) while software that can simulate climate effects at the district of block scale by calculating the interactions between buildings, vegetation, water bodies and construction materials are called microclimatic simulation software. The emerging need to control buildings' thermal loads resulted in an increase of the number of available BES and microclimate software.

2.2.1 The ENVI-met software and its outputs

ENVi-met (Bruse, M et al 1998) is a computation fluid dynamics (CFD) based microclimatic simulation software that differently from others, calculates the influence of various parameters to return a wide variety of results. Most microscale models focus on a single aspect of the microclimate failing to achieve all the following, that belong to ENVI-met specifications.

- An adequate resolution (less than 10 meters)
- Accurate modelling of surfaces of all types, and of different material parameters
- Simulation of the physical and physiological properties of plants
- Prognostic and transient calculation of the weather processes

ENVI-met requires only 1 input file format (.SIM, text simulation file) which is a, ASCII text-based file that includes a 3D model, the meteorological parameters on its boundaries, and additional information regarding the simulation, such as its duration, roughness value etc. The 3D model file is a raster model which is encoded in .INX format. An INX file is a text document in XML (https://fileinfo.com/extension/indd, 2018). This model includes topography, buildings, vegetation and surface materials depicted in layers. ENVI-met is using the same 3D model data model and encoding through the years. More information about the INX file and the required weather parameters that compose the simulation file will be provided in chapter 3.

ENVI-met simulation generates a huge amount of hourly data which is organized in different files and folders. The size of the simulation outputs for a large model (250, 250, 40) for a twenty-four-hour day circle can exceed 100Gigabytes of space. In general, there are two types of generated outputs. First, pairs of XML encoded metadata, and data stored in binary files (EDX, EDT, https://fileinfo.com/extension/edx, 2018), and secondly additional data in ASCII format. The ED pairs are used to store voxel – based parameters and are used in the Atmosphere and Soils folders, while ASCII outputs are incorporated to store aggregated data on objects like buildings or 3D trees. The following list summarizes the content of the folders. (Atmosphere, Buildings, Inflow, Pollutants, Radiation, Receptors, Soil, Solar Access, Surface, Vegetation). The list of most important ENVI-met different outputs can be found in Appendix3. Outputs can be visualized, in both 2D and 3D view, by the Leonardo software, which is distributed along with ENVI-met (Figure 9).

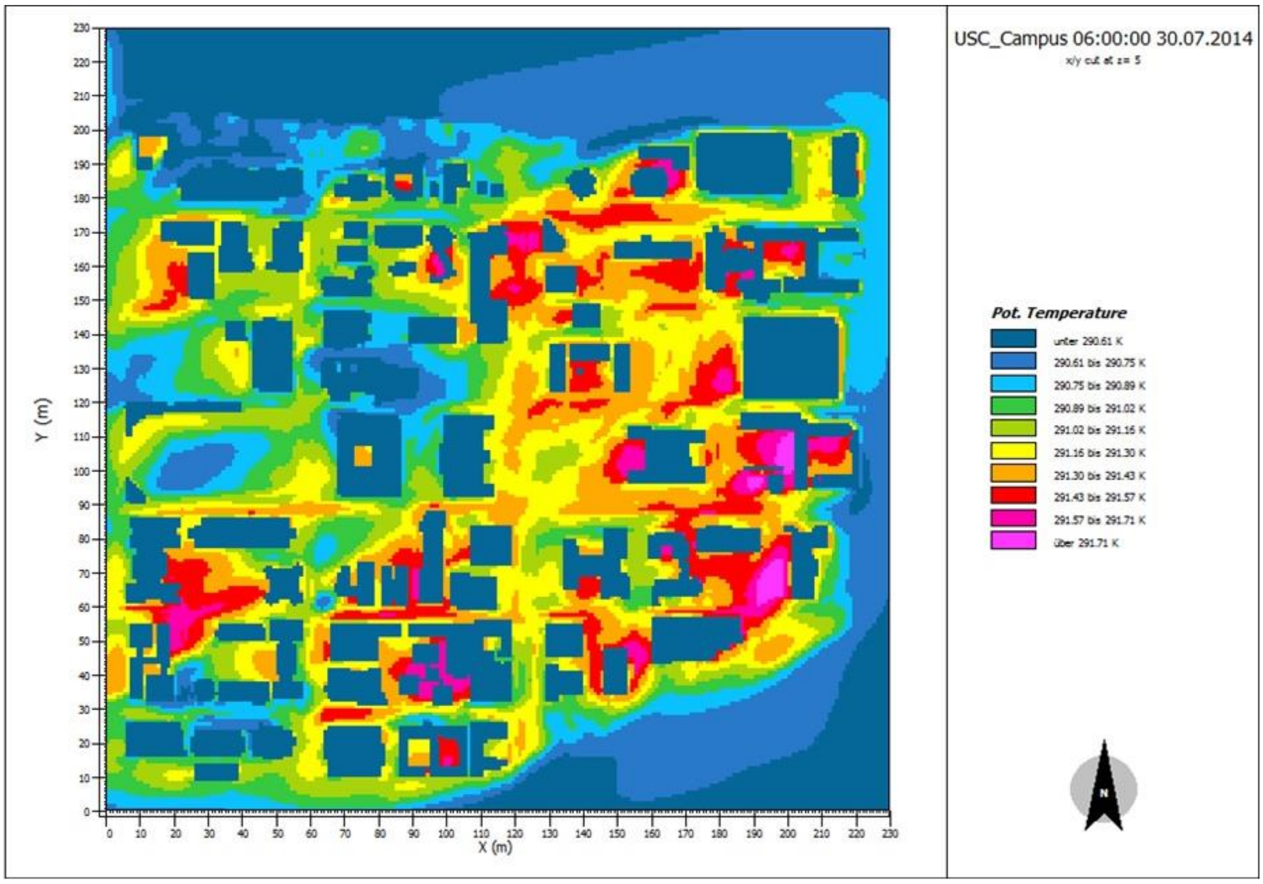


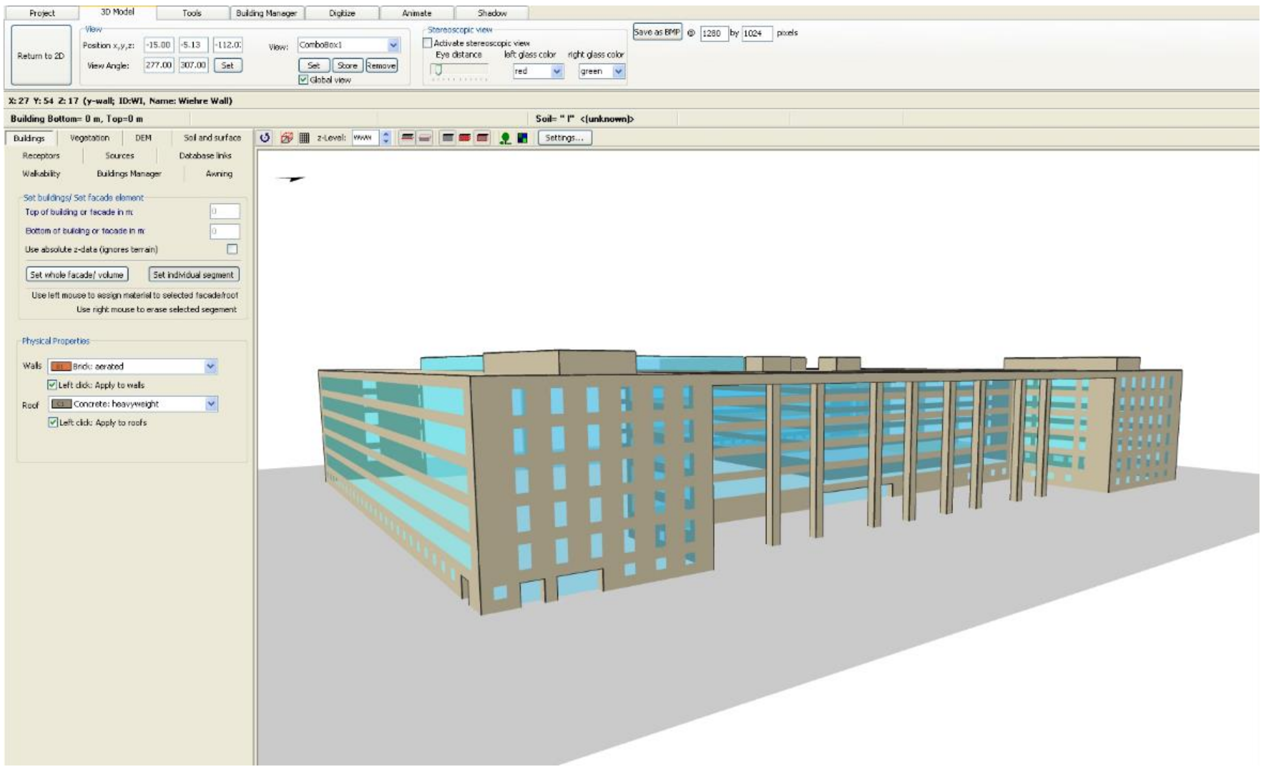
Figure 9: Leonardo interface, visualization of air temperature values in a 2-dimensional level , Source (https://civil808.com/college/architect/)

2.2.2 ENVI-met updates and vector data support

By 2010 ENVI-met managed to become one of the most popular tools for outdoor simulations, and has been used for many different types of research (see for example (Bruce, M. et al, 1988 ; Simon, H., 2016 ; Koerniawan, D. et al, 2015)). During the later years ENVI-met experienced a number of significant updates, on the calculation speed of the interface, and on the available design options. The former improvement allows users to simulate relatively large areas in a higher resolution on personal, multi-core hardware systems.

The most important update concerns the SPACES editor and the nature of the 3D model. Until ENVI-met 3.0 version the design of the 3D model was limited to extruded, prismatic buildings, and the whole SPACES editor did not include a 3-dimensional viewer. The area input file was limited to a 2.5 dimensional one. However, in ENVI-met 4.0 version SPACES editor was updated to support full 3D design. Together with this update a new prognostic 7 node façade model for the construction materials was introduced. In order to parametrize the structure of building facades and roofs, the new model allows the creation of complex materials, composed of different layer properties, including green facades

(Helge Simon & Bruse, 2015). The new full 3D editor allows the assignment of façade material on every building cell (Figure 10), and ENVi-met is now able to model more detailed buildings with detailed material allocation and calculate wall and indoor surfaces temperatures more precisely (Figure 10). The third major update is referred as 'full forcing' and introduced an option to input weather hourly values as boundary climate conditions for air temperature, wind speed and direction, as well as for relative humidity and radiation budget.



Position x,y,z: -15.00 -5.13 -112.0
View Angle: 277.00 307.00
☑ Global view
☐ Activate stereoscopic view
Eye distance
left glass color: red
right glass color: green
@ 1280 by 1024 pixels

X: 27 Y: 54 Z: 17 (y-wall; ID:WI, Name: Wiehre Wall)
Building Bottom= 0 m, Top=0 m
Soil= " I" <(unknown)>

Top of building or facade in m: 0
Bottom of building or facade in m: 0
Use absolute z-data (ignores terrain) ☐
Use left mouse to assign material to selected facade/roof
Use right mouse to erase selected segement

Walls: Brick: aerated
☑ Left click: Apply to walls
Roof: Concrete: heavyweight
☑ Left click: Apply to roofs

Figure 10: Full 3D SPACES editor, detailed material representation (Source: Huttner, 2012)

But despite all the latest improvements, the design of 3D models in SPACES is still time consuming and doesn't provide flexibility in terms of automatic conversion to different resolutions. For this reason, the need of using vector-based formats in order to create the voxel-based geometry of INX file emerged.

Currently there are two available tools that have already utilized that methodology. The first is a Rhino Grasshopper plugin, included in Ladybug tools (https://www.ladybug.tools/, 2018), and developed by Antonello Di Nuzio (https://www.envi-met.com/first-envi-met-plugin-for-rhino-grasshopper/,2019). This newly developed tool enables the conversion of Rhino 3D geometry to ENVI-met model areas (Figure 11). Furthermore, by using additional parameters, it allows users to generate an ENVI-met workspace and build the whole simulation (.SIM) input file of ENVI-met directly. Apart from generating ENVI-met input, it provides and additional option to visualize the outputs in Rhino's 3D viewer. This tool has 3 main limitations. First, Rhino is required as an intermediate platform, which is not an open-source software, and users – apart from architects and designers - might not be familiar with it. Secondly, geometry still needs to be created by the user, or at least converted from an existing format (i.e. Sketchup) to Rhino geometry. Last, there is no documentation concerning the possibility of importing DEM information, or creating full 3D INX files, by using this tool.

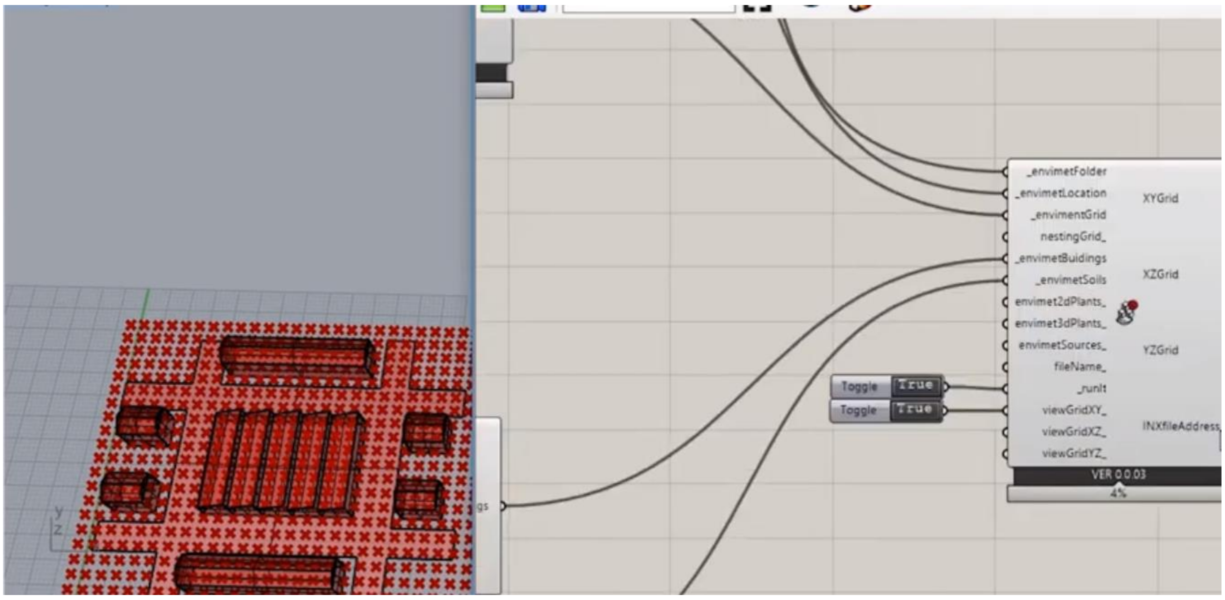


Figure 11: Rhino plugin for converting 3D Geometry to ENVI-met area input/ simulation file (Source : https://www.envi-met.com/learning-support, 2019)

The second tool that is able to transform vector-based geometries to INX file is the newly developed MONDE editor. This editor is currently packed with ENVI-met, (starting by version ENVI-met 4.3, released in December 2018) and provides an alternative option to the widely used SPACES editor. This editor enables the creation of layers of GIS geometry primitives, like nodes, lines and polygons to represent vegetation soils and buildings. Furthermore, it allows importing shapefiles and Open Street Maps (OSM) layers to use them as input for INX files. OSM is a 2-dimensional free online GIS map that also supports data, gathered by individual contribution from volunteers. Geometry in OSM is stored in the World Geodetic System (WGS84). Features, like in GIS, carry attributes (for instance, buildings have a height attribute). ENVI-met and as a result MONDE, support only metric systems. MONDE is programmed to convert vector data from the Universal Transverse Mercator (UTM) reference system, so a translation from WGS84 is previously required. The three main limitations of Monde are first that the editor is really complex (Figure 12), and the documentation is limited (https://www.envi-met.com/learning-support). Secondly data in Open Street Maps is not always complete. For example, building heights might be missing, and in that case the assignment of the attribute has to be done heuristically by the user. Finally, there is no indication yet on the way to apply surface materials, and if it can be done in an automatic or at least a semi-automatic way.

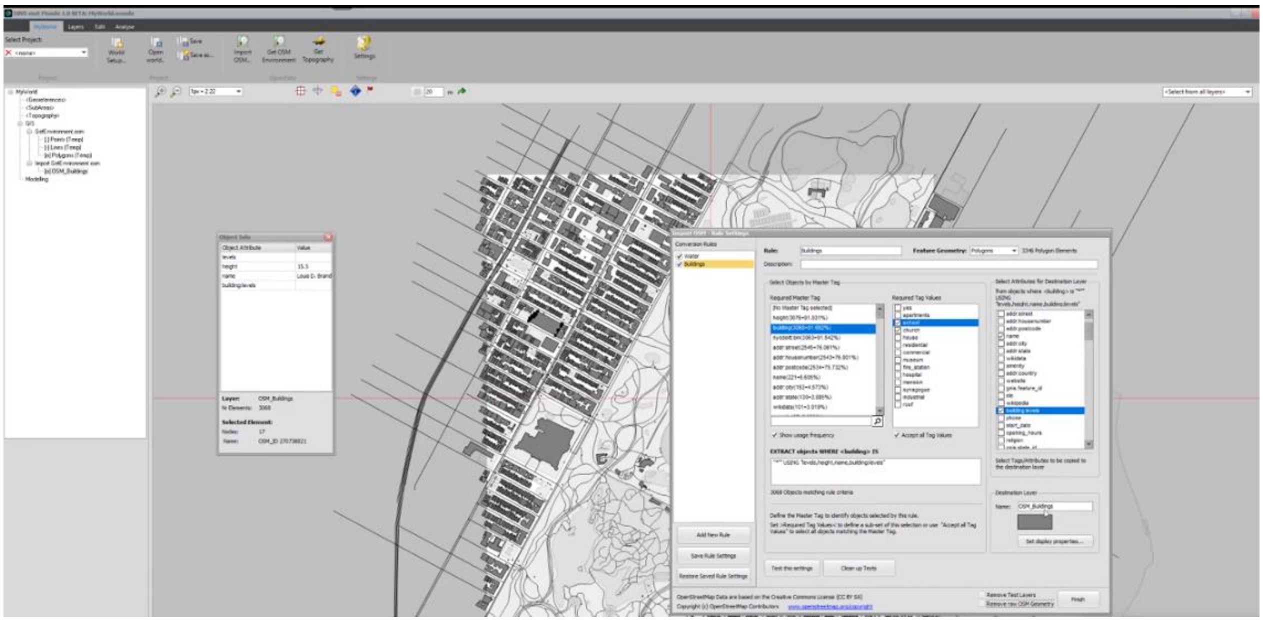


Figure 12: MONDE editor, loading OSM data, (Source : https://www.envi-met.com/learning-support/expert-lessons/, 2019)

2.2.3 ENVI-met level of detail and main limitations

ENVI-met, as mentioned in the section 2.2.1, is calculating numerous environmental parameters, so it is a computational expensive software. Plus, it requires a large grid, including the additional urban context around the area of interest, in order to simulate the microclimatic interactions with higher accuracy. Even with today's powerful processors, and developed downscaling approaches (H. Simon, Kropp, Sohni, & Bruse, 2018) the simulation of high resolution buildings in district scale is time costly and usually impossible. Additionally, each feature in ENVI-met is constricted to the raster format so cells can be either occupied by a feature or 'free'. Surfaces that are not parallel with the grid therefore, are converted to jagged borders and this is a major limitation of this grid approach, since inclined surfaces like roofs can't be modelled efficiently. In the design part, this distortion has to be limited as much as possible. Thus, despite the support of fully 3D area input files, which is indeed a significant step towards future development, when it comes to district scale simulations, this 3D format can't be fully exploited.

For this reason, and due to the limited and scattered documentation over the full 3D INX format, the effort of generating a full 3D area input file was abandoned. However, a material dissemination algorithm which identifies neighbour voxels, developed to handle the material data model peculiarities, has been implemented in SclGModeLer. Despite the lack of testing, it is believed that it will potentially work with uniform building materials and prismatic buildings (in a full 3D format), if that data is available.

Additional limitations of ENVI-met can be summarized on the following bullets

- ENVI-met do not take into account building interior information or occupancy behaviour.
- ENVI-met can neither simulate precipitation nor temperatures below the freezing point
- ENVI-met cannot simulate different underground profiles and thermal sources.

2.3 Related work and coupling between different data models

When it comes to modelling of the built environment, some (open) standards exist. The two most common ones are IFC (Industrial Foundation Classes) and gbXML (green building XML) and they focus on single-building environment and can have a resolution that surpasses even LoD4 in CityGML. A review of software that perform simulations on building scale (BES software) or wider scale is given by (Allegrini et al., 2015). However, moving from a single building approach to district scale, and keeping the bottom-up approach is still a big challenge for the reasons already mentioned. CityGML can be the missing piece that integrates the spatial relationships into a single data model and prevents mistakes caused by heterogeneous data sources. One of the main obstacles is that CityGML data is available only in specific large cities. Furthermore, due to the novelty of Energy ADE, simulation tools that support automated exchange of data do not exist, yet. Nevertheless, CityGML and Energy ADE has been already used in case studies, supported by developed interfaces.

2.3.1 CityGML and BES

An early approach in the city of Trento, Italy, where building energy demand was calculated is described by Agugiaro (Giorgio Agugiaro et al., 2015). It is a project that used CityGML as input for energy simulation and provided comparison values with other common approaches. First, a simplified mathematical model was implemented to estimate annual and monthly building energy demand. Next, more detailed information from an enriched CityGML file with properties like wall materials, thermal zones and building’s age, was fed to the BES tool EnergyPlus, and sub hourly values were calculated. In both cases, three different scenarios were simulated. The two major indicated drawbacks were the lack of more energy related attributes in the CityGML file (Energy ADE was under development at the time when the paper was written) and the complete negligence of microclimatic data.

2.3.2 Energy ADE and BES

The extension of CityGML with Energy ADE brought a growing interest towards exploiting and storing relevant data. Despite the lack of simulating tools that natively support the Energy ADE format, CityGML datasets augmented by means of Energy ADE were used in a number of studies. KIT University and HFT Stuttgart University conducted studies aiming to bring Energy ADE closer to simulation tools (Giorgio Agugiaro et al., 2018). Both generated and validated CityGML models, enriched by Energy ADE data. Classes like *ThermalZones, ThermalBoundaries, ThermalOpenings, UsageZones* and *WeatherStation* were filled with data and were used for the calculation of further energy-related data. KIT developed the

IFCExplorer bidirectional interface that maps a CityGML Energy ADE model to Energy Plus internal data model and returns Energy ADE data back to a CityGML file. HFT developed the platform SimStadt that reads enriched CityGML models, generates energy attributes and offers visualization options, as well as extension plugins. They both focus on urban scale; however, they do not try to integrate the effects of/on microclimate.

2.3.3 ENVI-met and energy modelling

Concerning microclimate simulations and ENVI-met software, many studies have covered the validation of their outputs by using real urban cases (Toparlar et al., 2017). Due to the achieving popularity of CFD studies on the urban microclimate, English published papers have been gathered and categorized by Toparlar et al. (2017). ENVI-met first original documentation was published by Bruse and Fleer back in 1998 and from that time, from around 180 approximate CFD investigated studies, around half of them have used ENVI-met. It also indicates that building-related parameters like energy consumption have an increasing popularity, even though not very common so some microclimate simulation software, including ENVI-met provide a rough estimation of them. The later years that the significance of using microclimate data as building boundary condition in energy simulations became evident, ENVI-met is usually used to feed BES software with accurate weather data. Yang et al., (2012) and Gobakis et al., (2017), ran detailed building simulations with Energy Plus and ESP-r software respectively, and they both created interfaces to match their geometrical units to ENVI-met grid in order to use microclimatic weather data. Both studies compared the simulation result to those with raw weather data from weather stations and found significant differences varying on different periods of the year. Yang et al, highlighted the need for available microclimatic data, as microclimatic simulation is such a heavy operation that actually limits studies.

Finally (Chatzinikolaou, Chalkias, & Dimopoulou, 2018) adopted a GIS approach to create a 3D spatial database to store and visualize ENVI-met simulation results. Therefore, the authors indicated the absence of semantics, and pointed CityGML as a more suitable way of storing the extracted information.

3. Data requirements

The delivered product of this thesis is a coupling framework between the two data models, implemented via an interface that further allows for user interaction via a GUI. It can be easily divided into two parts. The first part consists of the construction of ENVI-met input area model automatically, by retrieving the required data from a CityGML dataset. The second part consists of the geometrical matching of ENVI-met output geometry with the input CityGML model's reference system in order to transfer data back to the city model. In order to perform the mappings, a research over the data requirements of the targeted models was required as well as familiarisation with the different data model specifications and complexities. This chapter will cover the ENVI-met software mandatory requirements in detail, and analyse CityGML, both as a data model and as a DataBase framework.

3.1 ENVI-met functionality and inputs

To determine the data requirements of the ENVI-met software, first it is important to understand its functionality and its different input files. As a microclimatic simulation model, ENVI-met aims to provide a detailed representation of the weather parameters on district scale. The two factors that determine the microclimate are: (i) the weather on the lower tropospheric layers, or in other words meso-scale processes, and (ii) the energy exchanges between soil, water, plants, construction materials. The later, have an important influence on the very lower tropospheric layers, which can intensify the meso-scale processes or counteract them. Thus, ENVI-met needs to be informed about both the meteorological parameters, and the urban environment composition in terms of physical and material attributes. More precisely, ENVI-met is a computation fluid dynamics tool, which uses raster (grid) formats for its area inputs and its outputs, with a suitable resolution for the district representation. Grid cells interact with each other, to simulate the energy exchange. The basic algorithms used for these calculations are described in (Huttner, 2012). ENVI-met tries to model the lower tropospheric layers with the aid of three interrelated sub-models. The first one is a 1-dimentional vertical model which contains vertical profiles of different meteorological parameters, with simpler words the weather information. This model is used to provide the area of interest with boundary conditions and is extended from surface level, up to a height of 2500 meters above the ground (Huttner, 2012). Secondly a 3-dimensional soil model, which starts from the soil surface level and reaches a depth of 1,75 meters. The role of this model is to estimate the temperature and humidity flux in the soil, and between soil and lower air masses. These two grid models are telescopic and not equidistant, which means that the dimension of each cell increases along its direction. This is because they 'embrace' the third model, which is a 3-Dimentional model of the area of interest. So, as they extend away from it, their impact on it fades away, especially in the limited time span of the simulations, which is usually less than three days.

The last model is the aforementioned 3-dimensional, orthogonal grid model. Buildings, trees and the DEM are represented in this model, and ENVI-met provide outputs only for this model. This model is usually generated manually via the built-in 'SPACES' editor, while ENVI-met utilizes a default model for the soil and the vertical profiles. It is often referred as the area input file and is stored in an XML based INX format. This model is the urban 3D model input of ENVI-met, and the model (file) that is mapped by the CityGML urban data in this project.

One of the characteristics of the area input model is that it can theoretically be limitless on the number of cells on the horizontal directions and limited to 100 cells on its vertical. However, since ENVI-met simulation time is exponential to the area input model's size (simulating grids of horizontal extent of more than 200*200 cells can take up to several days) even with multi-core support, and the current processors, the horizontal size should be strictly controlled. Its vertical dimension also determines the lowest part of the 1-dimensional weather model. On top of the 3D model, the 1-dimensional model is extended up to 2500 meters and splits the H-Z distance (Figure 13) in 15 telescopic cells. The horizontal resolution of this model is constant for all grid cells, but on the vertical dimension this is not always the case. Three types of vertical grid are available which will be discussed on a later section.

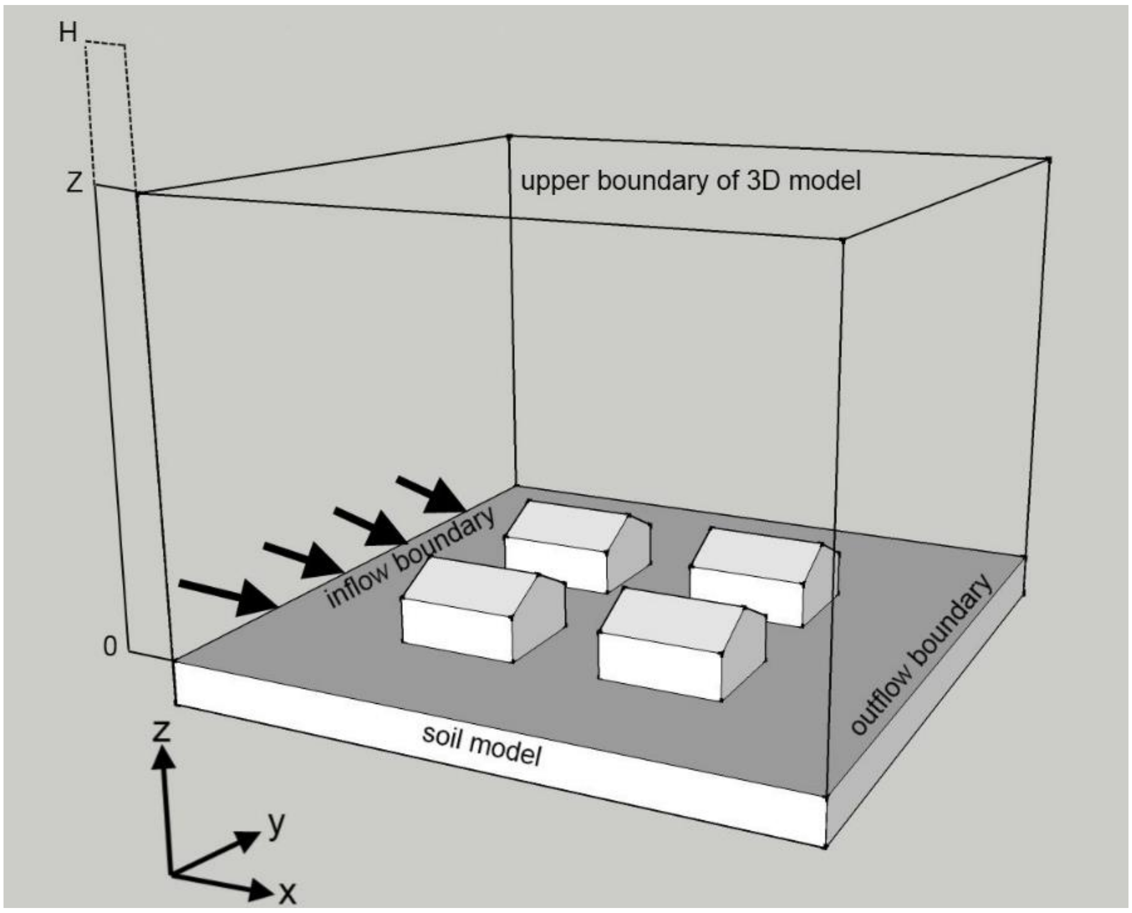
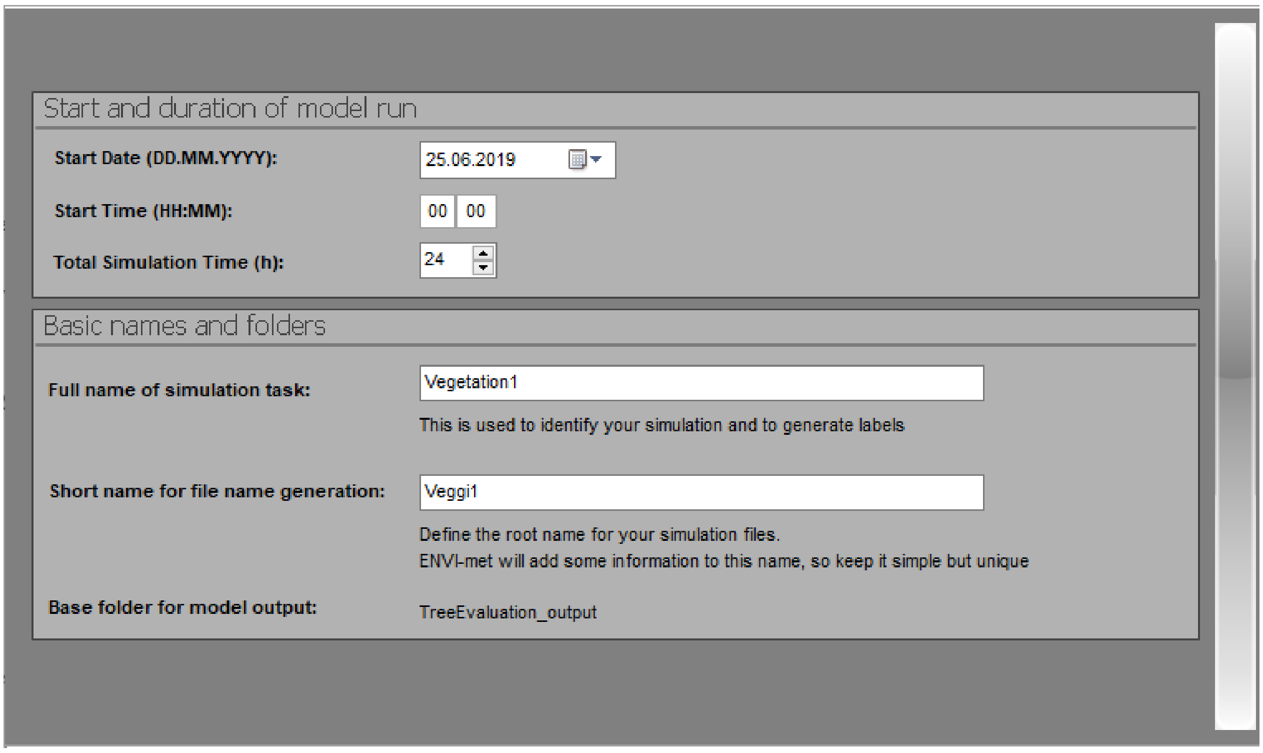


Figure 13: The three different grids used by ENVI-met, area input file is depicted on the 3D model, (Source Huttner, 2012)

3.2 Weather data and settings requirements

ENVI-met needs the meteorological parameters at its boundaries, or in other words the boundary conditions so that it can initialize the vertical weather model and start the interaction with the 3-dimentional area model. The required weather data consists of the air temperature, relative humidity and wind speed near the surface, the wind direction and the specific humidity at 2500 meters beyond the surface (Figure 15). Furthermore, it requires the definition of the date, which is going to be simulated, as well as the starting time, the duration of the simulations and a roughness value (Figure 14). Together with the latitude and longitude which is stored on the area input file ENVI-met can initialize the sun path. The weather parameters as well as the remaining settings can be edited inside ENVI-met during the construction of the simulation file. The available options for inserting the weather parameters are three. The first, allows editing of a higher and a lower temperature, while wind speed and relative humidity values are constant. The second option is called *Simple Forcing*, and the user can create a graph of hourly temperatures and relative humidity values which will be applied to the simulation file. In the winter 2019 version, ENVI-met introduced the latest and most detailed version, which enables the dynamic manipulation of the meteorological input data, and it is called *Full Forcing*. With Full Forcing the user can define a diurnal cycle of vertical boundary conditions for air temperature, relative humidity, wind speed, wind direction and (if available) radiation components. (https://www.envi-met.com/learning-support) The full forcing file (.fox) is an encrypted ASCII file which can be created in the Forcing Manager window of ENVI-met, by an input csv file with the values of the aforementioned parameters, on a half-hour step (https://www.youtube.com/watch?v=KxHnVJFMVcc). This file can be used as an input for the weather data instead of storing weather parameters in the simulation file.



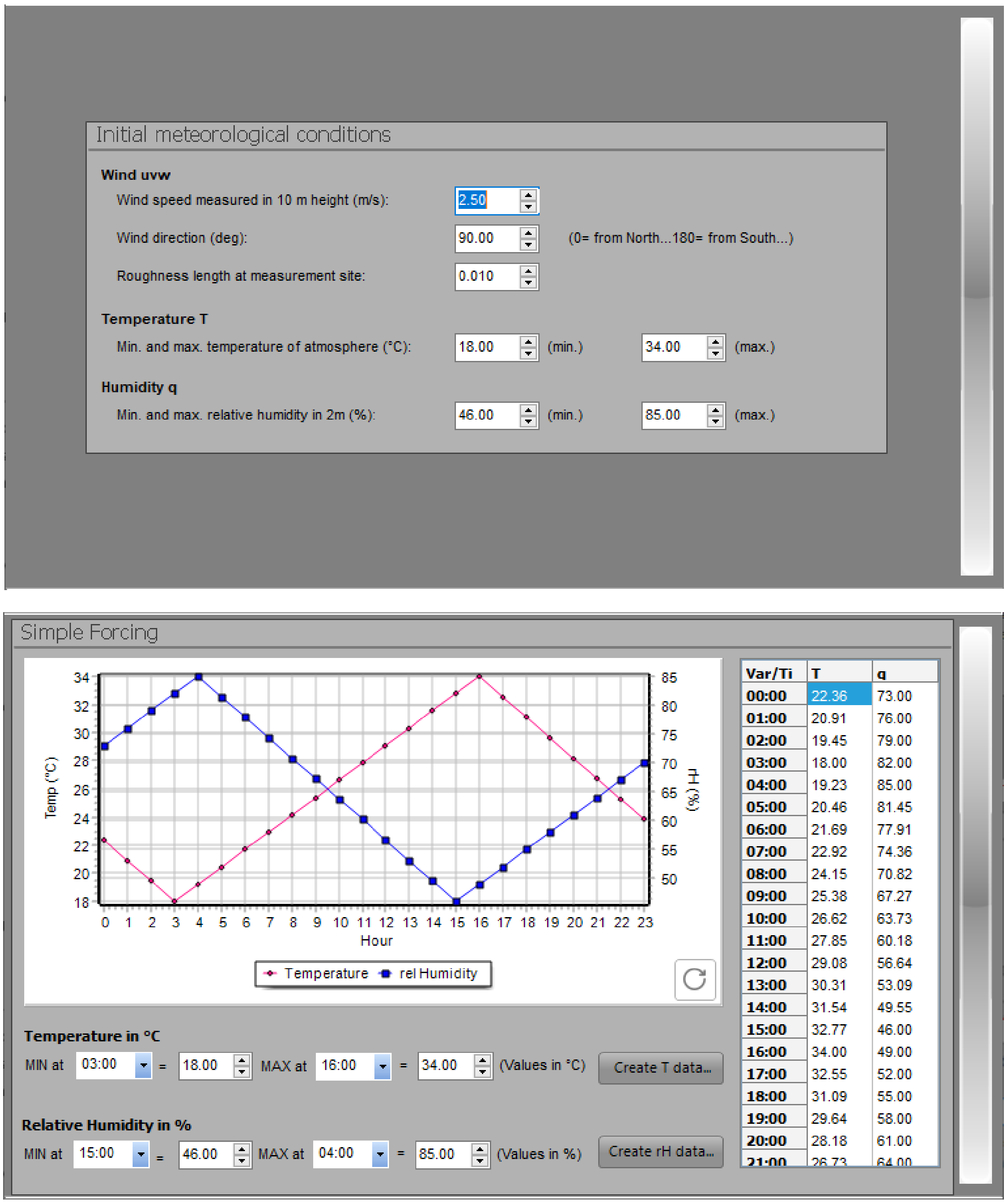
Start and duration of model run

Start Date (DD.MM.YYYY):	25.06.2019
Start Time (HH:MM):	00 00
Total Simulation Time (h):	24

Basic names and folders

Full name of simulation task:	Vegetation1 This is used to identify your simulation and to generate labels
Short name for file name generation:	Veggi1 Define the root name for your simulation files. ENVI-met will add some information to this name, so keep it simple but unique
Base folder for model output:	TreeEvaluation_output

Figure 14: Selection of basic parameters in ENVI-met



Initial meteorological conditions

Wind uvw

Wind speed measured in 10 m height (m/s):	2.50	
Wind direction (deg):	90.00	(0= from North...180= from South...)
Roughness length at measurement site:	0.010	

Temperature T

Min. and max. temperature of atmosphere (°C):	18.00	(min.)	34.00	(max.)

Humidity q

Min. and max. relative humidity in 2m (%):	46.00	(min.)	85.00	(max.)

Simple Forcing

Temperature in °C

MIN at 03:00 = 18.00 MAX at 16:00 = 34.00 (Values in °C) Create T data...

Relative Humidity in %

MIN at 15:00 = 46.00 MAX at 04:00 = 85.00 (Values in %) Create rH data...

Var/Ti	T	q
00:00	22.36	73.00
01:00	20.91	76.00
02:00	19.45	79.00
03:00	18.00	82.00
04:00	19.23	85.00
05:00	20.46	81.45
06:00	21.69	77.91
07:00	22.92	74.36
08:00	24.15	70.82
09:00	25.38	67.27
10:00	26.62	63.73
11:00	27.85	60.18
12:00	29.08	56.64
13:00	30.31	53.09
14:00	31.54	49.55
15:00	32.77	46.00
16:00	34.00	49.00
17:00	32.55	52.00
18:00	31.09	55.00
19:00	29.64	58.00
20:00	28.18	61.00
21:00	26.73	64.00

Figure 15: Simple Forcing graph of weather parameters in ENVI-met

3.3 Area input file requirements

In this section, the architecture of INX format- area input file will be thoroughly discussed The INX file is based in XML; thus, it will be broken down to XML tag level to locate the data needs and the utilized data formats. The area input file holds both metadata of the area, and the urban data. The metadata contains useful information about the location of the model and about the grid type and resolution. Metadata helps ENVI-met initialize the weather profiles. The urban data can be furtherly categorized in three different groups, i) the mandatory 2.5-dimensional tags, ii) the optional 2.5-dimensional tags and iii) the 3-dimensional tags (used to describe buildings, materials and DEM in 3D INX files).

Features stored in the area input file include position and height of buildings, position and type of plants, distribution of surface materials and soils, position of sources and position of receptors (Figure 16).

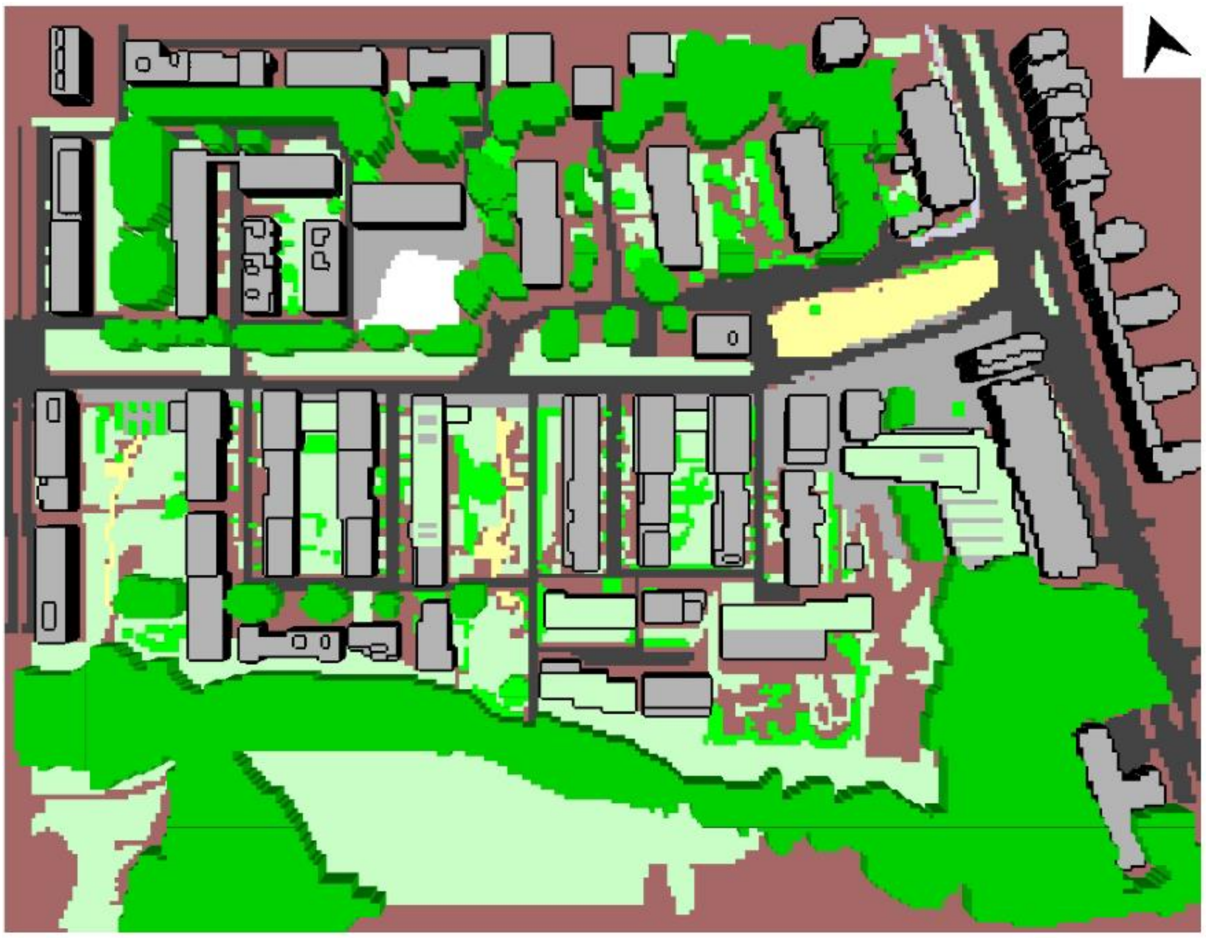
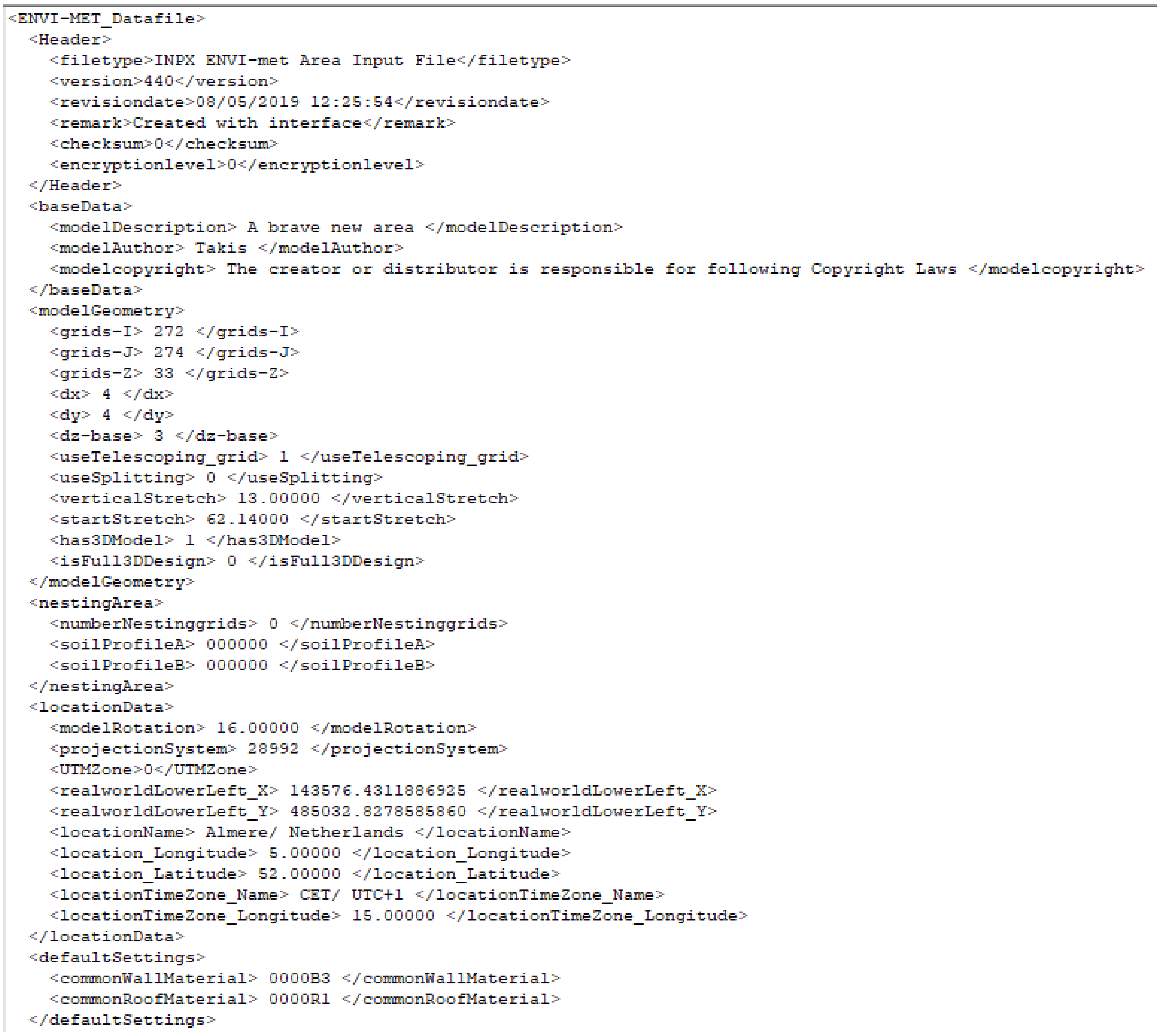


Figure 16: Model area displayed in SPACES editor. Buildings (grey), vegetation(green) and surface material colour coded (Source: Huttner, 2012)

Despite the fact that the documentation on the INX file is outdated - and therefore limited – it is an ASCII file which can be interpreted. To understand its architecture, several different INX files were constructed via the SPACES editor, with different parameters and combination of obstacles, and were analysed in a text editor. Furthermore, area input files, generated by SclGModeLer were always opened with SPACES, and saved again, so that potential changes could be identified. In this way a better understanding of the data format could be achieved.

3.3.1 Settings and metadata requirements

Metadata of the area input file is of equally or even higher importance than the resolution and accuracy of urban data. When ENVI-met starts a simulation, it goes through the metadata tags to understand the location of the area in the world. In this way it can initialize the sun paths (and consequently the shadowing), radiation levels and the 1-dimensional weather grid. Furthermore, the grid information is held in the metadata tags. Grid resolution, number of cells in each dimension and the type of vertical grid is displayed here. Finally, it includes information that can be used in additional applications, such as position of the grid on a metric reference system. This section will provide the data of each of the metadata tags. They can be distinguished on Figure 17.



```
<ENVI-MET_Datafile>
  <Header>
    <filetype>INPX ENVI-met Area Input File</filetype>
    <version>440</version>
    <revisiondate>08/05/2019 12:25:54</revisiondate>
    <remark>Created with interface</remark>
    <checksum>0</checksum>
    <encryptionlevel>0</encryptionlevel>
  </Header>
  <baseData>
    <modelDescription> A brave new area </modelDescription>
    <modelAuthor> Takis </modelAuthor>
    <modelcopyright> The creator or distributor is responsible for following Copyright Laws </modelcopyright>
  </baseData>
  <modelGeometry>
    <grids-I> 272 </grids-I>
    <grids-J> 274 </grids-J>
    <grids-Z> 33 </grids-Z>
    <dx> 4 </dx>
    <dy> 4 </dy>
    <dz-base> 3 </dz-base>
    <useTelescoping_grid> 1 </useTelescoping_grid>
    <useSplitting> 0 </useSplitting>
    <verticalStretch> 13.00000 </verticalStretch>
    <startStretch> 62.14000 </startStretch>
    <has3DModel> 1 </has3DModel>
    <isFull3DDesign> 0 </isFull3DDesign>
  </modelGeometry>
  <nestingArea>
    <numberNestinggrids> 0 </numberNestinggrids>
    <soilProfileA> 000000 </soilProfileA>
    <soilProfileB> 000000 </soilProfileB>
  </nestingArea>
  <locationData>
    <modelRotation> 16.00000 </modelRotation>
    <projectionSystem> 28992 </projectionSystem>
    <UTMZone>0</UTMZone>
    <realworldLowerLeft_X> 143576.4311886925 </realworldLowerLeft_X>
    <realworldLowerLeft_Y> 485032.8278585860 </realworldLowerLeft_Y>
    <locationName> Almere/ Netherlands </locationName>
    <location_Longitude> 5.00000 </location_Longitude>
    <location_Latitude> 52.00000 </location_Latitude>
    <locationTimeZone_Name> CET/ UTC+1 </locationTimeZone_Name>
    <locationTimeZone_Longitude> 15.00000 </locationTimeZone_Longitude>
  </locationData>
  <defaultSettings>
    <commonWallMaterial> 0000B3 </commonWallMaterial>
    <commonRoofMaterial> 0000R1 </commonRoofMaterial>
  </defaultSettings>
```

Figure 17: Metadata tags of (INX) area input file

Header Tag

In this tag, metadata about the file itself is provided. Namely, data concerning the file format, the version of ENVI-met, the creation date, a remark of the editing tool used and finally two tags about a checksum and an encryption level which were left to '0' by default. This information is automatically written upon the creation of each model via SPACES.

Base data tag

Here we can find user defined information, like a string description of the model. Also, the license type is nested in this tag.

Model Geometry tag

This tag initializes the 3-dimensional model's grid. As mentioned earlier in this chapter, for this model, ENVI-met uses an orthogonal grid with a constant resolution along each of the two-horizontal axes (x, y). However, for the vertical axis ENVI-met provides a set of options to the user, and the type of vertical grid shall be defined in this tag. The reason behind the vertical grid variation is that quite often, ENVI-met is used for the estimation of close-to-surface parameters, like for example shadowing or human thermal comfort. As a result, attention is shifted to the lower grid cells where higher resolution is needed. To tackle this, ENVI-met provides three additional grid types, displayed on the figure 18.

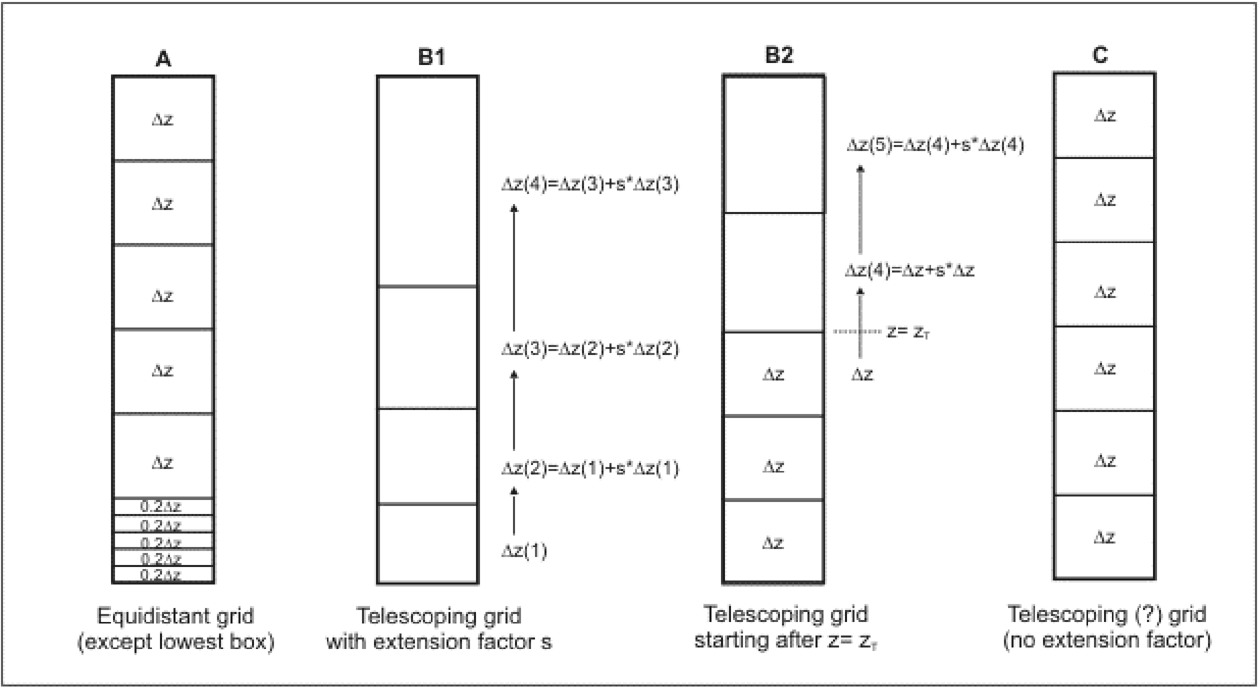

A
Δz
0.2Δz
Equidistant grid (except lowest box)
B1
Δz(4)=Δz(3)+s*Δz(3)
Δz(3)=Δz(2)+s*Δz(2)
Δz(2)=Δz(1)+s*Δz(1)
Δz(1)
Telescoping grid with extension factor s
B2
Δz(5)=Δz(4)+s*Δz(4)
Δz(4)=Δz+s*Δz
z= z_T
Δz
Telescoping grid starting after z= z_T
C
Δz
Telescoping (?) grid (no extension factor)

Figure 18: The available vertical grids in ENVI-met, by left to right (Equidistant grid, Telescoping grid with extension factor, Telescoping grid with extension factor starting from z, Telescoping grid with no extension factor)

The first type of vertical grid is an equidistant grid, used as the default in ENVI-met SPACES editor. It is a hybrid-equidistant grid in practice, where the lowest cell is divided into 5 cells with resolution equal to $\Delta z/5$. This grid type provides high resolution on the lowest part of the model. An additional advantage is that it offers more modelling options when coming to small size features like grass for example. As mentioned before, a cell in ENVI-met can be either fully occupied by a feature or empty. Thus, grass or small size plants and bushes demand high resolution in order to be modelled efficiently, otherwise they are completely neglected. The limitation of this grid approach is that it is not suitable for modelling districts with very high buildings, as the available cell number is limited to 100. Also, in case of a model with an incorporated DEM, the split lower cell is occupied by the DEM, which is not a desired outcome. To set up this grid type, the type of the grid is required and the constant vertical resolution Δz. ENVI-met then proceeds automatically to the adjustment.

The second vertical grid type is the so-called telescoping grid. This grid follows the same logic with the 1-dimensional weather grid. This grid is initialized by setting the size of the lower cell and a telescoping factor s. The size of every cell above the base cell is then increasing, by following a simple exponential algorithm which is displayed on the previous figure. This telescopic grid allows to cover much more height in a compact model. It should be used when modelling a district with high objects, such as skyscrapers, and the processes on the upper parts of the model are of less interest.

The last grid is a modified telescopic grid that the cell expansion starts above a reference height. The latest is initialized by setting the size of the base cell, the telescoping factor and the reference absolute height.

In the model geometry tags the displayed parameters are the number of cells in each direction, the resolution along x and y axis and the size dz of the lowest cells. These are user input and can be edited directly in the Spaces editor. The grid type information is displayed in the subtags '*Use telescoping grid*', '*Use splitting*', '*Vertical stretch*', and '*Startstretch*' and they can take Boolean values. Additionally, this tag holds the file's type information (if it is 2.5D or full 3D).

<u>Nesting Area tags</u>

ENVI-met is not working reliably near its borders, especially where obstacles exist. The reason is that the turbulence cannot be simulated, resulting to Errors. The nesting area is a method used in order to move the borders of the model away from the objects. However, this approach is lately questioned and replaced by extending the main 3-dimensional model by a number of empty cells, or at least with cells with DEM data.

<u>Location Data tags</u>

ENVI-met uses a grid with local Cartesian coordinates. The lower left cell is registered as the 0,0,0 and the higher right as nx-1, ny-1, nz-1. However, ENVI-met simulates a real place with real world coordinates. This is important since the sun path and the solar radiation intensity are influenced by the latitude. Furthermore, correlation to the real-world coordinates is useful for further applications, so this metadata enables the reverse translating of the area model, back to its real-world coordinates. Also, Furthermore, ENVI-met provides a rotation option which is achieved by rotating the model instead of the grid. The rotation is useful in order to align the main building orientation to the grid, reducing the number of jagged borders, as much as possible.

The sub-tags of the location data tag are the <modelRotation> which can only take integer values, a projection system indication and the coordinates on the beginning of the grid in a metric system, the relative to Coordinated Universal Time TimeZone, and four tags that define the location in terms of name, longitude, latitude and time zone name.

3.3.2 2.5D Area input file mandatory requirements

After reading the metadata, ENVI-met needs the features, and their material parameters, which influence the microclimate of the area. This data is captured on the remaining tags, which can be divided in the three sections which were earlier mentioned. The features that can be stored in the following tags are categorized to buildings, trees, soils and topography (dem). In the mandatory section as well as the optional section the data representation type used in most cases is referred as matrix-data. This data type illustrates a 2D raster which separates pixels by commas (Figure 19). It is edited as a sequence of the depicted information, a reference to the data type and the number of cells in both horizontal dimensions

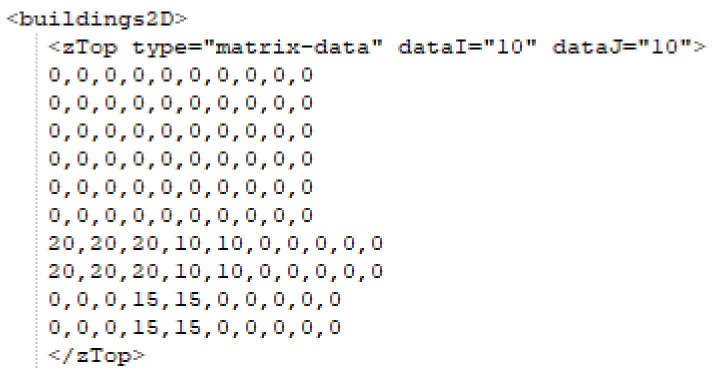
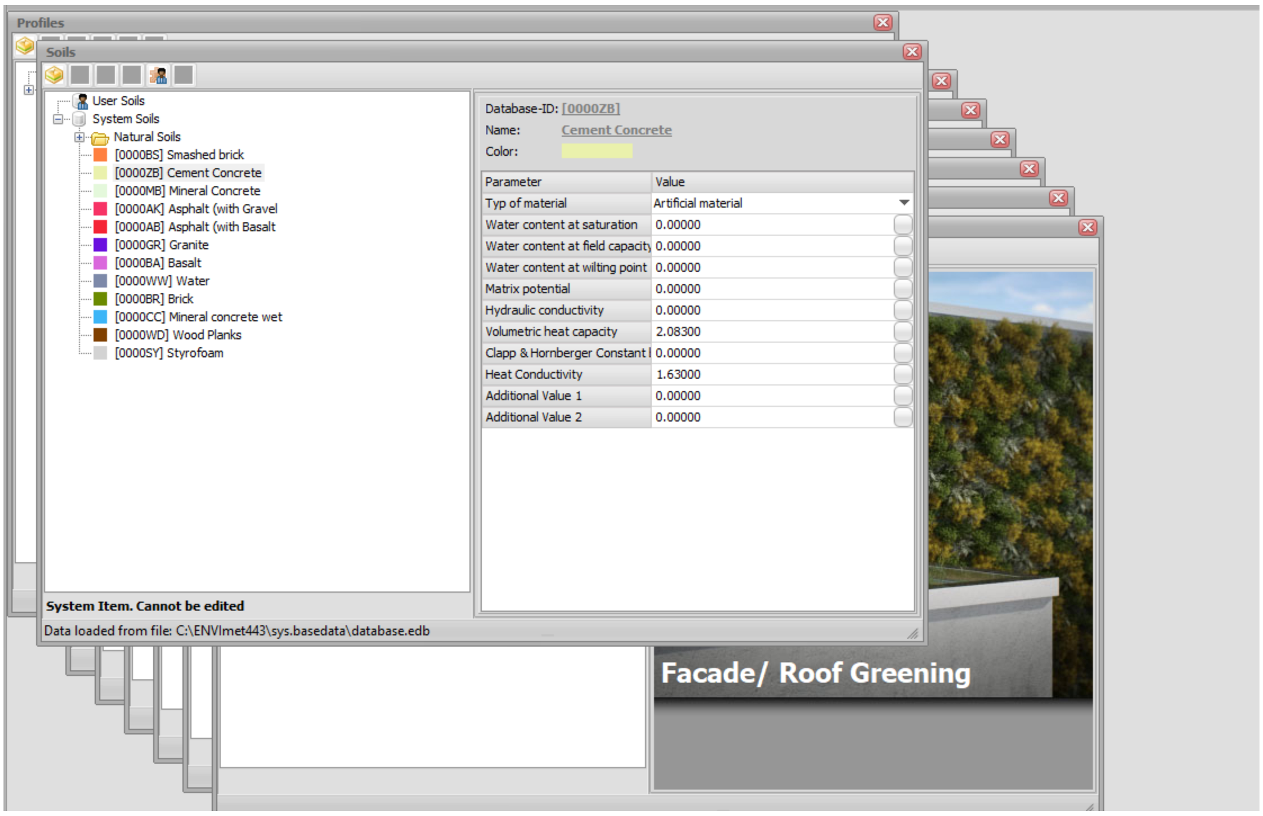

```
<buildings2D>
   <zTop type="matrix-data" dataI="10" dataJ="10">
   0,0,0,0,0,0,0,0,0,0
   0,0,0,0,0,0,0,0,0,0
   0,0,0,0,0,0,0,0,0,0
   0,0,0,0,0,0,0,0,0,0
   0,0,0,0,0,0,0,0,0,0
   0,0,0,0,0,0,0,0,0,0
   20,20,20,10,10,0,0,0,0,0
   20,20,20,10,10,0,0,0,0,0
   0,0,0,15,15,0,0,0,0,0
   0,0,0,15,15,0,0,0,0,0
   </zTop>
```

Figure 19: Matrix type data, used in area input file's 2D tags

Before going through each tag, it is important to describe the basic concept that allows the linking between geometry and materials in ENVI-met. ENVI-met incorporates a number of built-in relational databases of manually pre-built materials that are used in the area input file. These materials can be either soils, water, grass, tree foliage, or building construction materials. They are one of the most important features in ENVI-met, since they give an identity on the occupied voxels. Each material has a number of attributes that influence their behaviour, for instance an albedo value, or for trees a leaf area distribution (LAD) value http://www.envi-met.info/doku.php?id=filereference:dbssystem. All the material properties are stored in a database. The link is applied by using a 6-digit codespace (Figure 20), which is different for each material. Consequently, the database can be considered as a third input. The user can also enrich ENVI-met databases, by designing custom materials. In this case a reference to the updated database, or to a user-created database is needed.



User Soils
System Soils
Natural Soils
- 🟧 [0000BS] Smashed brick
- 🟨 [0000ZB] Cement Concrete
- ⬜ [0000MB] Mineral Concrete
- 🟥 [0000AK] Asphalt (with Gravel
- 🟥 [0000AB] Asphalt (with Basalt
- 🟪 [0000GR] Granite
- 🟪 [0000BA] Basalt
- 🟦 [0000WW] Water
- 🟩 [0000BR] Brick
- 🟦 [0000CC] Mineral concrete wet
- 🟫 [0000WD] Wood Planks
- ⬜ [0000SY] Styrofoam

Database-ID: [0000ZB]
Name: Cement Concrete
Color: 🟨

Parameter	Value
Typ of material	Artificial material
Water content at saturation	0.00000
Water content at field capacity	0.00000
Water content at wilting point	0.00000
Matrix potential	0.00000
Hydraulic conductivity	0.00000
Volumetric heat capacity	2.08300
Clapp & Hornberger Constant	0.00000
Heat Conductivity	1.63000
Additional Value 1	0.00000
Additional Value 2	0.00000

System Item. Cannot be edited
Data loaded from file: C:\ENVImet443\sys.basedata\database.edb

Figure 20: ENVI-met databases, icon display soil 6-digid codespaces

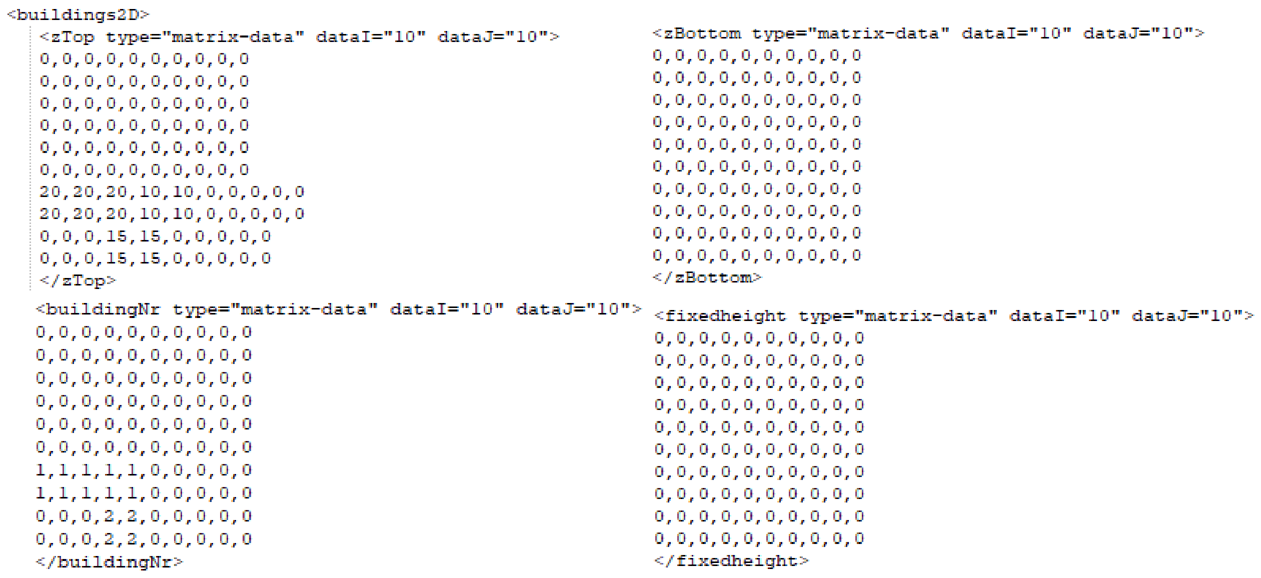
Buildings_2D tags

This tag holds all the required information about buildings on a 2.5-dimensional INX file. As mentioned earlier, buildings in this format are created by extruding the footprint up to their height. This is the first tag that uses the matrix-data type. So, it is a 2-dimensional representation of the building features. Furthermore, an id is assigned to every cell which is used to merge building columns to one building entity (Figure 21). Based on this information, ENVI-met can aggregate output values to building level. The building tag has four sub-tags. For each column that is occupied by a building feature, the corresponding matrix position receives a value, different than zero which indicates a column without building.

The first two matrices, feature the height on the bottom of the building and the height at the top of the building (<zBottom>, <zTop>). These values are relative to the DEM, which means that a building with a 0 <zBottom> is placed at the top of the DEM. An important fact is that the values indicate meters and not voxel numbers, and they must be integers. ENVI-met translate the metric data automatically to voxel occupancy, by transferring the values whenever needed. For instance, in a simple telescoping grid without extension factor, a building with 15 <zTop> value in a 2-meter z axis resolution will occupy 8 voxels. The same building in an equidistant grid will occupy 8+4 voxels.

The third matrix (<buildingNR>) contains the id assigned on every building entity. When designing the area model in the SPACES editor, this id is generated automatically with a default incremental order starting by 1. Every time a new cell position, which is not tangent to a building cell, is filled, it is assigned the next id number. However, disjoint cells can belong to the same building entity. The only limitation concerning ids is that they have to be integers and every building column should have an id.

The final matrix <fixedheight> is set empty by default in 2.5 area input models. Its role has not yet been researched.



```
<buildings2D>
   <zTop type="matrix-data" dataI="10" dataJ="10">
   0,0,0,0,0,0,0,0,0,0
   0,0,0,0,0,0,0,0,0,0
   0,0,0,0,0,0,0,0,0,0
   0,0,0,0,0,0,0,0,0,0
   0,0,0,0,0,0,0,0,0,0
   0,0,0,0,0,0,0,0,0,0
   20,20,20,10,10,0,0,0,0,0
   20,20,20,10,10,0,0,0,0,0
   0,0,0,15,15,0,0,0,0,0
   0,0,0,15,15,0,0,0,0,0
   </zTop>
```

```
<zBottom type="matrix-data" dataI="10" dataJ="10">
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
</zBottom>
```

```
  <buildingNr type="matrix-data" dataI="10" dataJ="10">
  0,0,0,0,0,0,0,0,0,0
  0,0,0,0,0,0,0,0,0,0
  0,0,0,0,0,0,0,0,0,0
  0,0,0,0,0,0,0,0,0,0
  0,0,0,0,0,0,0,0,0,0
  0,0,0,0,0,0,0,0,0,0
  1,1,1,1,1,0,0,0,0,0
  1,1,1,1,1,0,0,0,0,0
  0,0,0,2,2,0,0,0,0,0
  0,0,0,2,2,0,0,0,0,0
  </buildingNr>
```

```
<fixedheight type="matrix-data" dataI="10" dataJ="10">
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0
</fixedheight>
```

Figure 21: The four different building tags. Upper left (Building Height), Upper right (Building bottom) Lower left (Building id) Lower right (Building fixed number)

Simple Plants 2D tag

The second tag of the Main Section is used to depict vegetation as a simple plant object. Simple plants are a set of vertically occupied voxels (Figure 22). The number of voxels is related to the height of the vegetation object. The voxels also have plant foliage parameters like albedo, transmittance, leaf type, leaf occupancy etc. Furthermore, they interact with the 1-dimensional soil grid, since they contain root parameters. Simple plants are the first and simpler representation of vegetation that ENVI-met implements. Grass can also be modelled in this way as it is displayed in Figure 22. However, attention should be paid when using this type of vegetation, as it is independent of horizontal resolution. Since the representations will just occupy a single column, the method can't be effectively used to model tree entities. For their placement on the 3-dimensional grid, the matrix-data type is incorporated, in a way similar to the 2D buildings tag. Trees of this type are automatically projected on top of the DEM, if it exists, and cannot overlay building cells. ENVI-met will not handle a building – tree overlay, and it will return an Error before the simulation begins. Simple trees are depicted in the matrix by their representing codespace in the database.

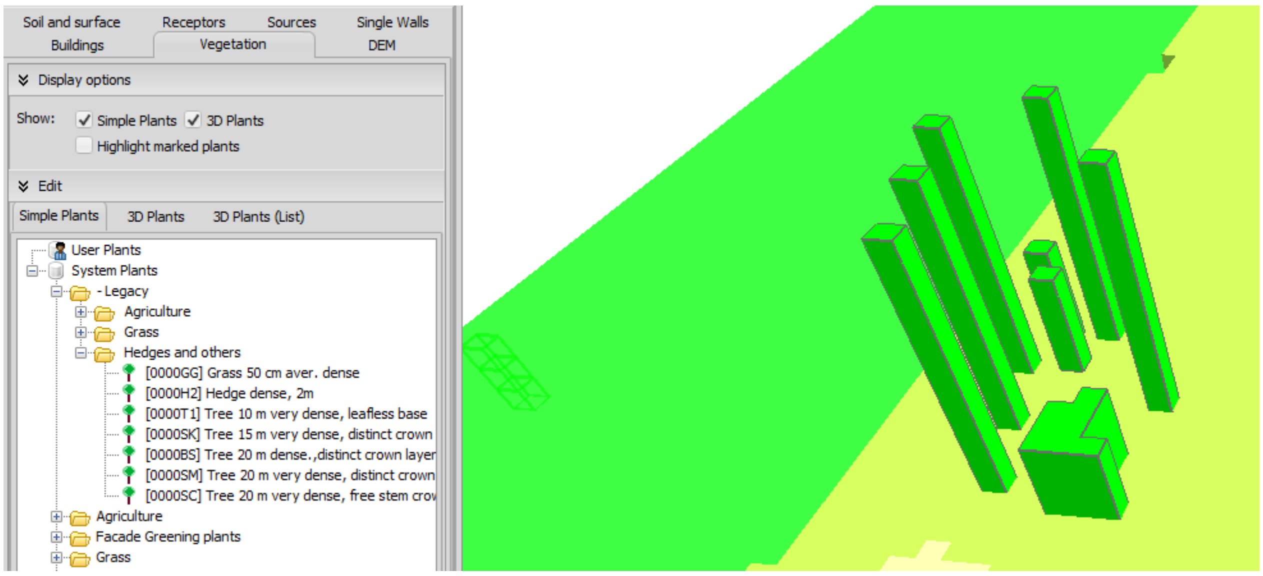


Figure 22: Simple trees in ENVI-met SPACES 3D viewer, they occupy a single cell column

Soils 2D tag

The <Soils2D> tag is using a matrix data type, which captures the material of the surface, on top of the DEM. Different soils types are depicted by their codespace link to the profiles database (Figure 20) and the assignment of a single material in every cell (even where buildings and vegetation is placed) is obligatory. When designing in the SPACES editor, the grid is initialized with a default profile, which can be overwritten. The different material assemblance in ENVI-met is used to design urban elements such as roads, pavements, water bodies etc.

DEM 2D tag

The <dem> tag captures the base altitude value of the area, as well as the height discrepancies. In other words, the digital elevation model. It contains two subtags. The first one is the <DEMReference> and accepts both positive and negative float values. This value sets the reference height of the base voxels of the grid in meters. The second tag, the <terrainheight> is of matrix data type. Every space must contain one value, which is the relative height to the <DEMReference> value. These values must be zero or positive integers, and again in a way similar to buildingheights, correspond to metric values and not number of voxels. ENVI-met takes care of the voxel occupancy by ceiling this metric value. When the equidistant grid with the split lower voxel is used along with a DEM, the splitting is applied to the lower voxel of the grid, and does not follow the DEM. It is evident that this combination should be avoided, since the augmented resolution on the output atmospheric parameters is therefore lost.

3D TREES tags

The 3D tree tag is used to position 3D tree objects on the model. Every 3D tree is an independent entity inspired by a full-grown real tree species. 3D trees were implemented in ENVI-met on a later version and they approximate the tree behaviour with higher accuracy than a simple plant. It has a full 3-dimensional shape (Figure 23), in contrast with simple plants which are the product of extrusion, but can be included in the 2.5 area input file. The 3D trees are stored in a built-in relational database called Albedo, which provide a dynamic design interface. A 3D tree also holds an id value, mainly focusing on tree behaviour simulations.

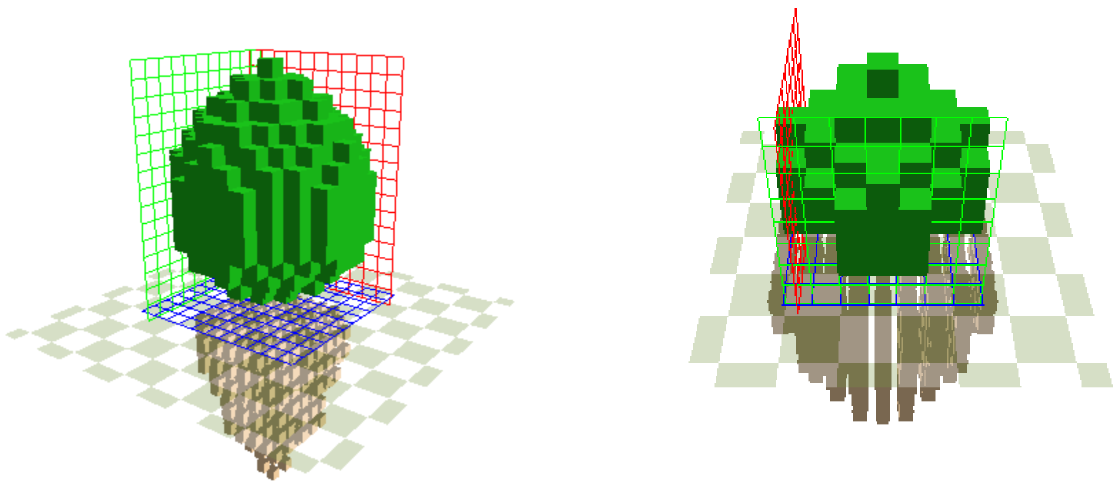


Figure 23: Same 3D tree object in two different resolutions, Albero interface

3.3.3 Area input file optional requirements

This section describes area input model tags that are optional to perform simulations in ENVI-met. In this thesis they were not relevant, and this section will be limited to a small description of the following:

Sources 2D tag

This section defines the position of different sources in the model domain. It uses again a matrix data type, similar to the plants. If there is a particle source in the area, it is referred to its relative cell by its codespace in the database.

Receptors tag

The concept of receptors allows to collect data for selected points in the model area in a compact way without browsing through several files to find the required information.

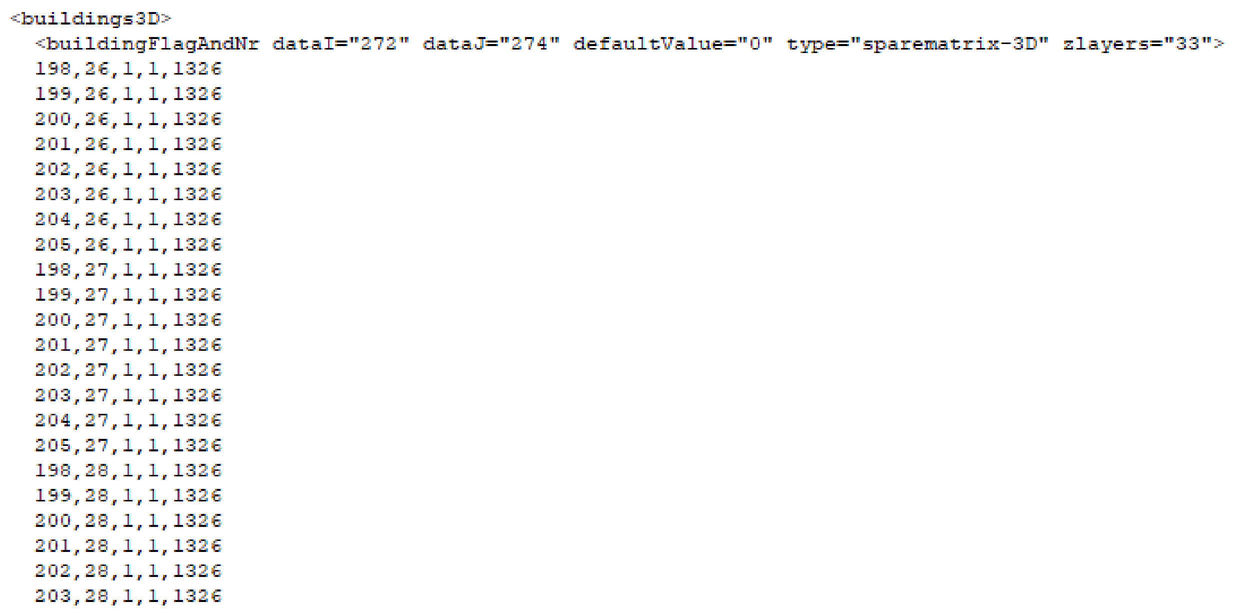
For each receptor, a snapshot and a time series file are created. The snapshot files contain the data at the receptor at a given time e.g. a vertical profile in case of the atmospheric file. The time series files contain the simulation data at the receptor for the complete simulation duration.

Database Link Points and Database Link Areas

These database sections have no function in ENVI-met. It is linked with ENVI-met supporting software to link certain grid points and areas with database entries.

3.3.4 3D Area input file requirements

In the 3D section the data representation used is defined as a sparematrix-3D. This data type is a tag which contain only registered cells by their location on the 3-dimensional grid as a tuple (finite ordered list of elements), where the bottom left cell is registered as 0,0,0. Tuples can be appended by extra attributes which will be explained in the related sub-section (Figure 24).



```
<buildings3D>
  <buildingFlagAndNr dataI="272" dataJ="274" defaultValue="0" type="sparematrix-3D" zlayers="33">
  198,26,1,1,1326
  199,26,1,1,1326
  200,26,1,1,1326
  201,26,1,1,1326
  202,26,1,1,1326
  203,26,1,1,1326
  204,26,1,1,1326
  205,26,1,1,1326
  198,27,1,1,1326
  199,27,1,1,1326
  200,27,1,1,1326
  201,27,1,1,1326
  202,27,1,1,1326
  203,27,1,1,1326
  204,27,1,1,1326
  205,27,1,1,1326
  198,28,1,1,1326
  199,28,1,1,1326
  200,28,1,1,1326
  201,28,1,1,1326
  202,28,1,1,1326
  203,28,1,1,1326
```

Figure 24: Sparematrix-3D data type

Buildings3D

<Buildings3D> is a tag that contains all the cells occupied by buildings. In few words, the voxels are written as (x,y,z,id) tuples where x,y,z indicates the absolute position of the cell on the grid, starting counting from 0, and the id refers to the building entity it belongs. The cells are ordered by default by the z, y and then x value. This data representation is referred in the INX file as sparematrix-3D.

<u>DEM3D</u>

The <DEM3D> tag holds all the cells that are occupied by the DEM. It uses the same sparematrix-3D data representation as the Buildings3D tag. The number of cells that are occupied by the DEM can be vast, especially in large grids, so it is suggested that the reference height of the model should coincide with the lowest DEM height value.

<u>WallDB</u>

The last tag of the INX file, the <WallDB> holds the building materials dissemination over building facades. Construction materials should be applied only on the outer surface of buildings (in the facades that are directly visible in the 3D editor when using SPACES). To put it simply, a material should be applied between neighbour joint cells, where the one is occupied by building and the other is empty. The material information in the area input file is registered in the cell level, by using a codespace link to the materials database. Again, the sparematrix-3D representation is used, and every cell that contains at least 1 material must be registered. A cell can host up to three materials, one on its rear left face, one on its front face, and one on its bottom face (Figure 25). This format though is not suitable for pairing with geometrical algorithms, since materials can be applied to non-building voxels as well (roof façades or building right facades).

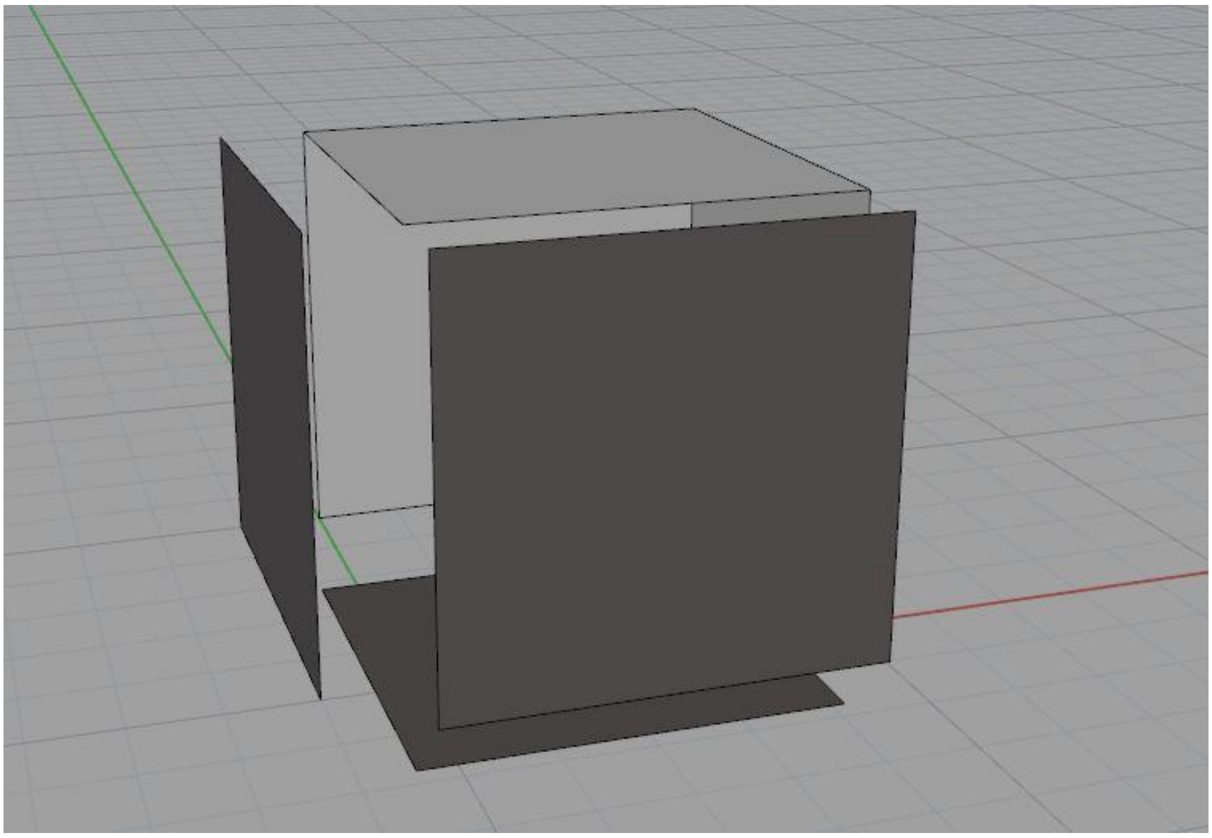


Figure 25: Material application on voxel level. Every voxel can take a value for its left, southern or base surface

3.4 Overview of CityGML and Energy ADE data model

In this section the CityGML + Energy ADE data models will be outlined. Focus won't be shifted over a specific thematic module; a thorough description of the semantic data of each module will be rather provided, since the purpose is to locate data, which can potentially fulfil the requirements of ENVI-met. However, first, it is important to provide a more detailed description of the CityGML data model, so that interrelations between objects will be pointed out. The information that will be provided in this section will further aid to follow sections 5.1 and 5.2, where a methodology to create a CityGML dataset from raw data files will be described.

In CityGML the most important features in an urban environment are divided in separate thematic modules. The goal of this modelling is to reach a high grade of semantic interoperability, between different applications. (Gröger et al., 2012) . By following this logic, CityGML has become a hub of information which can be easily accessed (Gartner et al., 2009). Features belonging to a class contain spatial and non-spatial attributes, which are mapped according to GML3 feature properties. Concerning the geospatial domain, as mentioned in section 2.1.1, the geometries of objects are represented as boundary representations. Furthermore, geometries can be combined to form aggregations, complexes and composites of the primitives (feature26). For example, a building that is composed of building parts can have the *Composite Solid* geometry type, or a road can be an aggregation of traffic and auxiliary traffic areas.

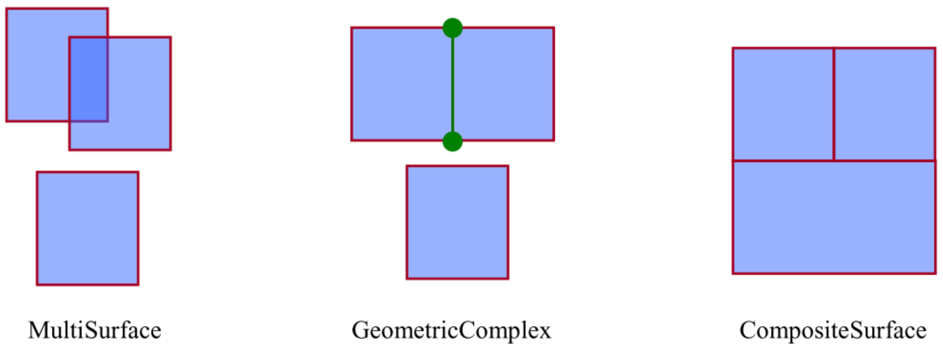


Figure 26: Combined geometries in CityGML

Apart from the geometric representation of a class, spatial characteristics are displayed as attributes. This is to highlight some important characteristics of an object, as is the height of a building, the length of a bridge, the crown diameter of a tree, etc.

The semantic model of CityGML employs the ISO TC211 family standards for the modelling of geographic features. The base class of every object is the abstract class *_CityObject* which is sub class to the class *Feature*. *_CityObject* sub classes are grouped in the different thematic modules (Figure 27) mentioned in the beginning of this section. Their principles, subclasses and attributes are depicted in their representing UML diagrams. This object-oriented modelling facilitates the maintenance of hierarchical order when modelling a complex thematic feature, like a building or a tunnel (Gartner et al., 2009).

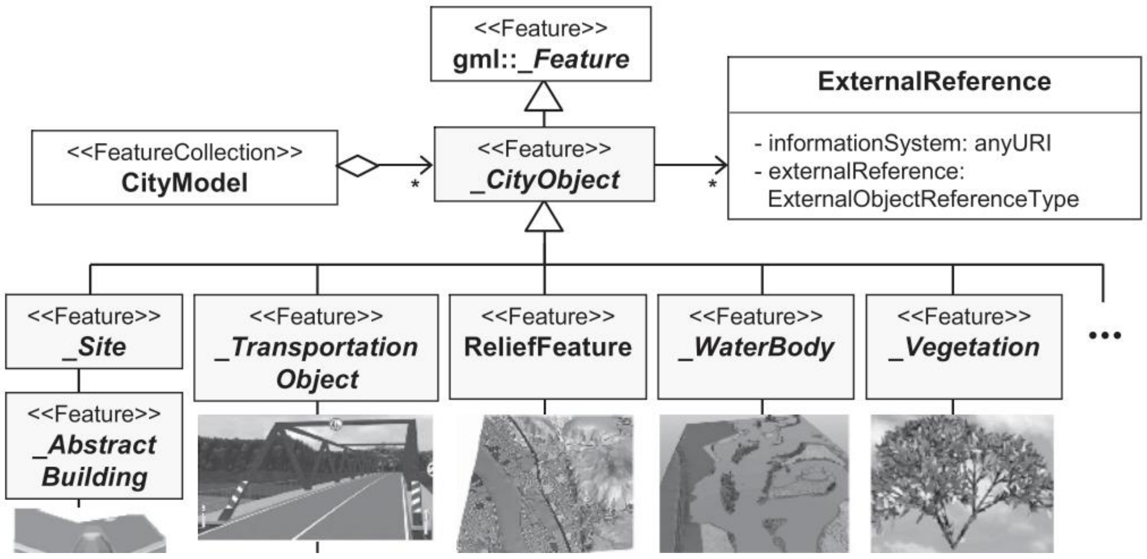


Figure 27: CityGML thematic modules and base class _CityObject (Source: Gartner et al., 2009)

CityGML is scalable, it differentiates five different level of details (LODs) starting from LOD0 and going up to LOD4. LODs in CityGML usually prescribe an upper limit of additional subclasses and do not necessarily result in higher geometrical complexity, nor have standard specifications (F. Biljecki et al., 2013). For instance, a building modelled according to LOD1 has a flat roof and a building modelled according to LOD2 cannot have windows or openings. However, it is not stated that a building modelled according to LOD3 must mandate the specified semantics (Biljecki et al. 2016). Furthermore, if a feature is modelled on a high LOD, it does not have to necessarily contain the lower ones. From that perspective the concept of LODs and the whole CityGML is quite flexible. In general terms, there are not qualitative standards for a CityGML dataset. It is considered 'rich' if the features are modelled in a high LOD and depending the amount of stored data in the data model (Biljecki, 2017).

3.4.1 CityGML Thematic Modules

Core Module

CityGML is partitioned into different thematic modules. It has a *Core* Module where the base class of every city object (*_CityObject)* is defined. The respective UML diagram for the *Core* module can be found on Appendix 2. Each *_CityObject* class has the attributes *CreationDate* and *TerminationDate*. Furthermore, the attributes *RelativeToTerrain* and *RelativeToWater* which provide a base knowledge for basic topological queries. In general, each feature has the attributes *Function* and *Usage*, which provide a further description of the role of the feature. These attributes can take different values according to the class they belong to. Since _*CityObject* is the super class of all ten thematic modules, they inherit those attributes. (Gröger et al., 2012)

In addition to the thematic content, the core model also provides the concept of implicit geometries, which is a different way GML3 approaches geometric representations. This type of geometry is important in CityGML and used for keeping the dataset relatively compact in cases of multiple similar objects that are repeated (like trees or traffic lights). An implicit geometry is a geometric object where the shape is stored only once as a prototypical geometry. Then it is re-used or referenced many times, wherever the corresponding feature occurs in the 3D city model. Each occurrence is represented by a link to the prototypic shape geometry (in a local cartesian coordinate system), by a 4*4 transformation matrix that is used to rotate it and scale it to the actual object's dimensions, and by an anchor point denoting the base point of the object in the world coordinate reference system (Gröger et al., 2012).

<u>Extending CityGML – Generics Module</u>

In section 2.1.2 the concept of ADEs for extending CityGML to application focus issues was discussed. An alternative way of extending CityGML is to use the concept of *Generics* Module. *Generics* objects and attributes facilitates the further storage of objects that are not explicitly modelled by any of CityGML thematic modules. This is enables through the *GenericCityObject* and *GenericAttribute* themes. *GenericAttributes* can be associated to any city object, they must have a unique name which can be freely chosen and may consist of many data types (string, integer, double, etc.). However if an ADE which incorporates such an object is already available, it should be preferred since it is already formally standardized and can be validated against the xsd schema (Giorgio Agugiaro & Kumar, 2018).

<u>Building Module</u>

Buildings are considered the most elaborate features in CityGML since they make the bulk of actual 3D objects in semantic 3D models, and the majority of applications focuses on them. The pivot class is the *Abstract Building* which is child of the thematical module _*Site*. Its child classes are the *Building* and *BuildingPart*. Building aggregations to represent large scale sites are possible through the concept of CityGML *CityObjectGroup*. In the building module, the structured rules mentioned before about the LOD division can be clearly observed. They are illustrated in Figure 28.

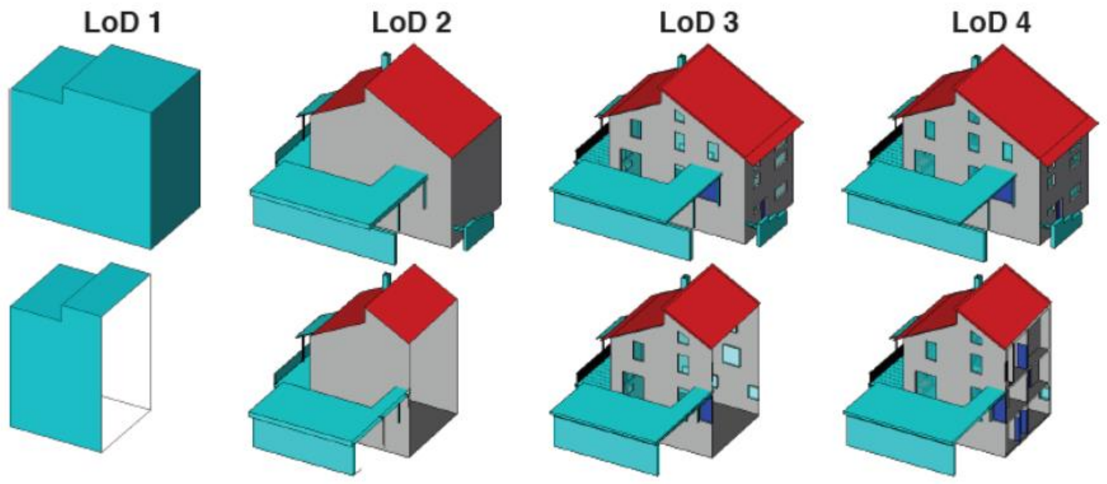


Figure 28: Level of Detail (LOD) for the Building module, Source: (Biljecki et al., 2013)

The coarsest LOD, LOD0 consist of horizontal 3-dimensional surfaces of the footprint of the building and/or the roof area. LOD1 is the simple blocks model comprising prismatic buildings with flat roof structures. It can be represented by either a _*Solid* or by boundary surfaces. Starting from LOD2 though, both properties can be modelled for every building. In LOD2 a building can have distinct roof structures and thematically differentiated boundary surfaces for the walls and ground level. LOD3 denotes architectural models with detailed wall and roof structures, potentially including doors and windows classes. LOD4 results by LOD3 model by adding interior structures for buildings. The UML model of the building class is depicted on the Appendix1.

Useful data concerning the construction of an ENVI-met INX file can be retrieved directly from the building geometry and by the *measured height* attribute. In a district scale resolution for which SclGModeLer is currently designed, the building representation on the INX file is similar to LOD1 (prismatic models without complex roof). Inclined roofs could be modelled, but the result would be a jagged border approximation which is not even guaranteed that would provide more accurate results, so it was not included in the methodology. Consequently, the LOD0 footprint geometry and the *measured height* attribute (or if both geometries are available), or the LOD1 geometry can be the only required data for filling the <building2D> tags of the base and max height.

<u>Tunnel module</u>

Tunnel module doesn't contain any useful data for this project, since ENVI-met does not allow piercings on the DEM.

<u>Bridge module</u>

Bridge module was not examined either, since 3D features of this type are not supported in the ENVI-met INX model.

<u>Water bodies (*Water Object Module)*</u>

Water is a very important part of the urban environment, and its effect in the microclimate is of high scale. For that reason, urbanization processes and cities were aimed to be built near sea, rivers or lakes. CityGML implies the thematic module *Water Object* to model water volumes which contain the subclass *Water Body* that indicates the water existence and *WaterBoundarySurface* which is part of the *Water Body*'s exterior shell. In LOD0 and LOD1 rivers are modelled by curves and seas and lakes by surfaces in LOD0 or solids in LOD1. They represent a low level of detail, but a high level of generalization. In higher LODs, the *WaterBoundarySurface* class is implied, which can be distinguished to *WaterSurface, WaterClosureSurface*, and *WaterGroundSurface*. The UML diagram of the WaterObject thematic feature is available on the appendix. ENVI-met does not provide a variety of water features it rather models water as a <Deep Water> element, so further semantic classification of water masses is not required. For rasterizing water features their geometry should be a 3D horizontal surface. In case of rivers, modelled as curves (i.e. polylines), they could be augmented by a *GenericAttribute* that indicates its width.

<u>City Furniture Module</u>

The *City Furniture Module* is used to illustrate the existence of immovable urban objects like traffic lights, traffic signs, benches, bus stops etc. These objects are used mostly for visualization, or as supporting for the transportation module (Gröger et al., 2012). Their size and parameters in considered of minor significance in ENVI-met and the modelling of this type of objects is not supported in the INX file.

<u>Transportation Module</u>

In CityGML, roads are modelled by using the *Transportation* module. In contrast to buildings, roads are not modelled by using volumes, and can therefore be modelled using 3-dimensional surfaces. Roads can be modelled by using networks of nodes and lines, since most applied to navigation applications. But higher application, like for examples serious games or driving simulation, areal representations are used. The main class is *TransportationComplex*, which represents different type of transportation lines, like a path, a road, a train rail etc. A *TransportationComplex* can compose of the classes (parts) *TrafficArea* and *AuxiliaryTrafficArea.* Different LOD representation of a *TransportationComplex* can be seen in Figure 29. As we can see the definition for LOD 2-4 isn't clear and the depth of detail depends of the complexity of each road feature. Both *TrafficArea* and *AuxiliaryTrafficArea* classes contain a *SurfaceMaterial* attribute which is pretty interesting for this project, since it can feed a large part of the <soils> tag on ENVI-met INX file. For this purpose, a high LOD model can satisfy ENVI-met resolutions of around 1-3 meters which is an average pavement or bike path width. After all, in ENVI-met as far as outdoor ground surfaces are concerned, the coherence of geometry is not as important as the depiction of the surface material.

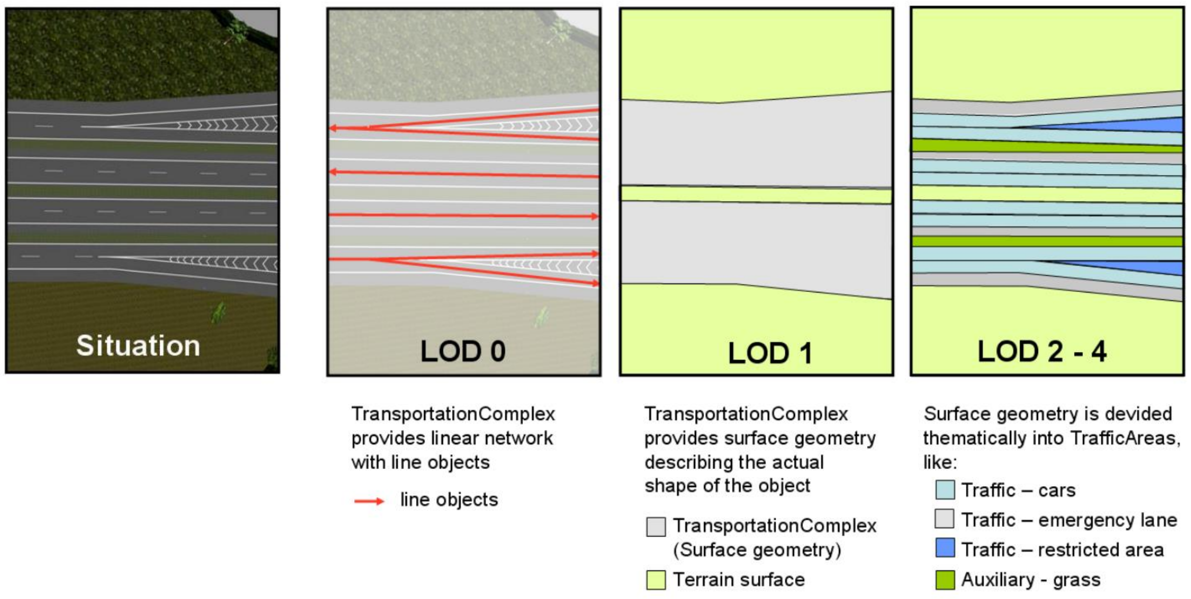

Situation
LOD 0
LOD 1
LOD 2 - 4
TransportationComplex provides linear network with line objects
line objects
TransportationComplex provides surface geometry describing the actual shape of the object
TransportationComplex (Surface geometry)
Terrain surface
Surface geometry is devided thematically into TrafficAreas, like:
Traffic – cars
Traffic – emergency lane
Traffic – restricted area
Auxiliary - grass

Figure 29: Different LOD representation of the Transportation module (Source : Gartner et al., 2009)

<u>Vegetation Module</u>

The vegetation features are important in CityGML since it is used by a large number of applications, including fire simulation, deforestation rate and of course microclimatic simulations (Gartner et al., 2009). The *Vegetation* module distinguishes between two different type of vegetation features. The first is called *SolitaryVegetationObject* and refers to single vegetation objects like trees. For this type of vegetation features can be represented by absolute coordinate GML3 geometries or by the concept of *Implicit Geometry*, discussed on the *Core Module* earlier. Different prototypical geometries can be used for different LODs in the last case, strict rules on the number of surfaces (as long as they don't result in a model of arbitrary size). *SolitaryVegetationObject* features can have the attributes, *Class* which indicates whether the object is a tree, or a bush or whatever, *Species*, *Height, trunkDiameter, crownDiameter.* The second feature is called PlantCover and is used for representing continuous vegetation areas of a similar constitution (i.e. a Forest or Grassland). The geometry representation of a PlantCover feature may be a *MultiSurface* or a *MultiSolid* and is LOD independent. *PlantCover* features may have the attributes *class* which represents its plant community type, and the attribute *averageHeight*.

<u>Relief Module</u>

In CityGML the Digital Elevation Model is provided via the thematic module *Relief*. The main class is called *ReliefFeature* and is composed by at least one *ReliefComponent* classes which can vary between four data types. A *ReliefComponent* can be either represented by a Triangulate Irregular Network (TIN), hence it inherits the name *TINRelief*, by Breaklines and then we are talking about *BreakLineRelief*, or by MassPoint and *MassPointRelief*, and last by a regular raster, therefore the *RasterRelief*. A *ReliefFeature* can consist of different types of *ReliefComponents* of different LODs and every *ReliefComponent* must have a geometry that defines its extent. Due to this uniqueness other thematic features like buildings of higher LOD can be placed on a lower LOD DEM. The DEM can be adjusted to features by utilizing the concept of *TerrainIntersectionCurves* which are used to make holes in the *ReliefModule* to fit to the features' footprint geometry.

The data that can be retrieved from the *ReliefModule* is the altitude of the terrain which is translated to INX file's <dem2D> and <dem3D> tag.

Land Use Module

The LandUse module is used to describe surfaces which are subject to specific and use, like streets or settlement areas. It can also describe larger areas of land cover types, for instance forest, grasslands, solid, sand etc. A *LandUse* object can be modelled according to all LODs and is represented as a *MultiSurface* geometry type. The surface geometry of *LandUse* must be 3D like all geometries in CityGML and it can either follow the terrain (*ReliefFeature)* or not. From the *LandUse* module, useful data can be retrieved that can be used to fill INX file's <soils2D> tag. However, when comparing its attributes with the *Transportation* module for example, we can see that a *SurfaceMaterial* attribute is missing. Although by the *Class, Function* and *Usage* attributes, a general idea of the surface material can be derived (Tables 1, 2), there is nothing to indicate the exact material properties. After examining ENVI-met requirements and highlighting the importance of the surface material distribution on the microclimate studies, it was decided to augment the *LandUse* module with a *GenericAttribute* of surface material, following the concept of the *TransportationModule.* Then, by increasing the resolution of *LandUse* to district scale and covering the whole extent of the model *with LandUse* polygons, it would be possible to fill the whole <soils2D> tag only with data, derived by the *LandUse* module. In other words, a high detailed *LandUse* module could integrate some of the 2-Dimensional spatio-semantic information, usually stored in the *Transportation, WaterBodies* and *Vegetation* modules.

Table 1: LandUse class attribute codespaces

1000	Settlement Area	3000	Vegetation
1100	Undeveloped Area	4000	Water
2000	Traffic		

Table 2: LandUse function attribute codespaces

1010	Residential	2050	Track
1020	Industry and Business	2060	Square
1030	Mixed use	3010	Grassland
1040	Special Function Area	3020	Agriculture
1050	Monument	3030	Forest
1060	Dump	3040	Grove
1070	Mining	3050	Heath
1110	Park	3060	Moor
1120	Cemetary	3070	Marsh
1130	Sports, leisure and recreation	3080	Untilled land
1140	Open pit, quarry	4010	River
2010	Road	4020	Standing Waterbody
2020	Railway	4030	Harbour
2030	Airfield	4040	Sea
2040	Shipping		

3.4.2 Using Energy ADE

In the introduction Energy ADE was indicated as a potential additional (to CityGML's classes and attributes) hub of data for microclimatic simulations. Energy ADE adopts the ADE mechanism to extend CityGML with energy-related classes and attributes. It is both modelled with UML diagrams and as an XSD schema, and like CityGML it has a modular structure which facilitates the data storage. Furthermore, the Energy ADE data model can be used for the direct storage of ENVI-met outputs as long as correspondences to Energy ADE classes and attributes exist, but also if the accuracy of ENVI-met satisfies those classes and attributes specifications. The main reason behind the latest, is that Energy ADE also incorporates aspects of buildings, which ENVI-met completely neglects as, for instance, occupancy and electric systems. ENVI-met accuracy on the estimation of building energy demand from this aspect can be considered questionable for further applications where Energy ADE is used. On the other hand, the integration of detailed weather parameters like temperature, humidity, radiation intensity, to the model, (aggregated on building level) can be very useful for further assessment of its energy performance.

The Energy ADE data model contains an Energy ADE Core which augments CityGML *Building* and *BuildingPart* classes, and additionally provides classes for the four central main modules of Energy ADE. These classes are the *BuildingPhysics, OccupantBehaviour, EnergySystems* and *MaterialandConstruction.* (Figure 30) (Agugiaro et al., 2018)).

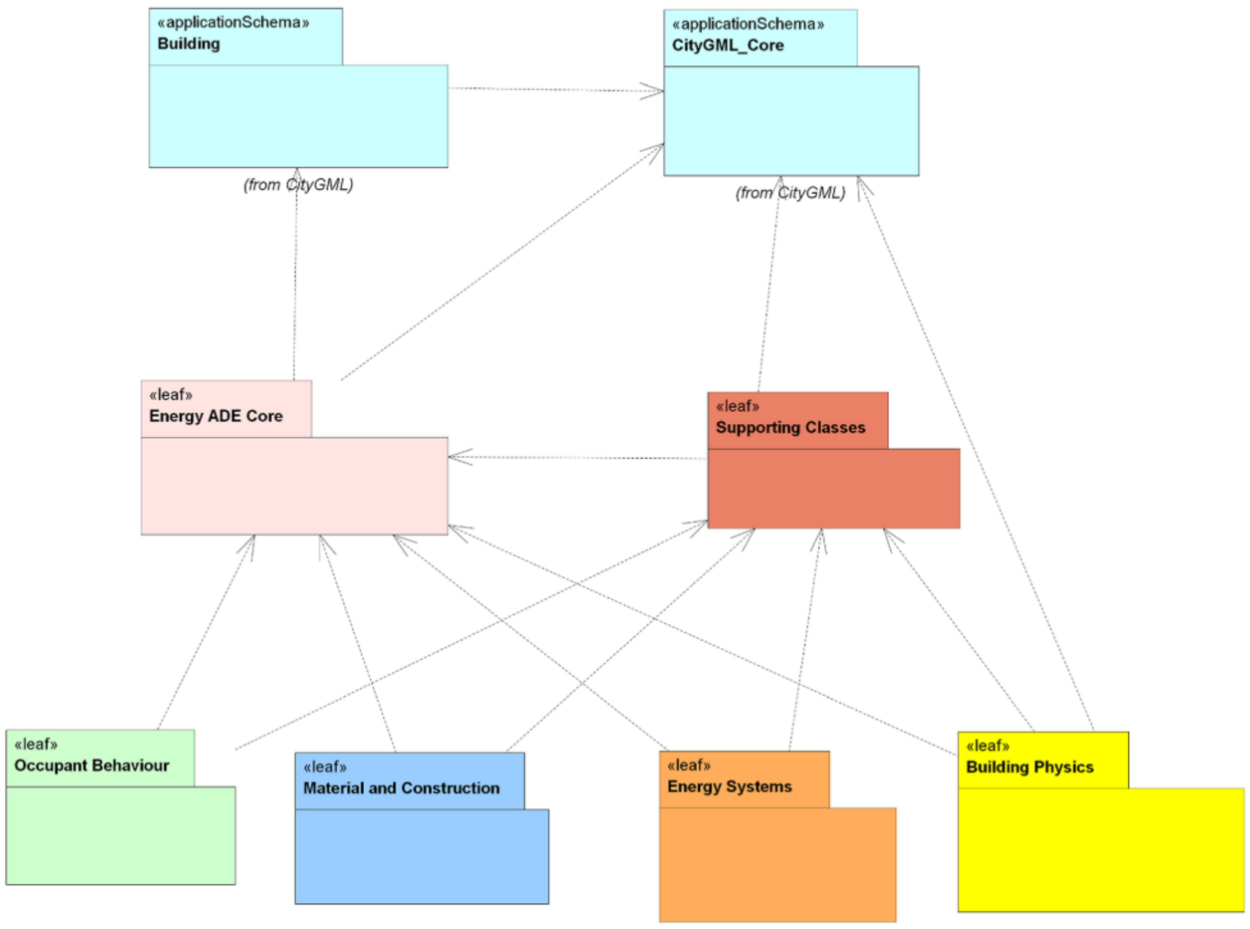


Figure 30: Colour-coded modular structure of the Energy ADE UML diagram. Source: Agugiaro et al., 2018

From a first look, the classes *Energy Systems* and *Occupant Behaviour* are irrelevant when it comes to the scope of this project and they won't be further discussed.

<u>Materials and Construction Class</u>

The material and construction thematic division of Energy ADE physically characterises a building construction depending on its materials and their thermal properties. In other words, it provides information about the physical and optical parameters for groups of building elements like the roof, windows, facades etc. These parameters can include U-values, density, specific heat and can describe a multi-layered material in the concept of a higher LOD. The material and construction class can hold useful data, which can fill the ENVI-met's Area Input File <walls3D> tag which is the building material sector. The material concept is similar to ENVi-met's which supports multi-layered materials in its latest versions.

<u>Building Physics Class</u>

The *Building Physics* module denotes the thermal behaviour of building volumes like rooms and is used to assist the estimation of need it heating/cooling. The three main classes are the class *Thermal Zone* which is associated with the CityGML class *Room*, the class *Thermal Boundary* which is related with the *Building* boundary surface and finally the class *Thermal Opening* which can be related to the class *Window* or *Doors.* A *Thermal Zone* should be always bounded by *Thermal Boundaries* in order to be separated from other classes of the same type in case we are talking for two rooms for example. Those boundaries can in their turn contain *Thermal Openings.* This class specifications do not meet ENVI-met data requirements since ENVI-met doesn't include any house interior information.

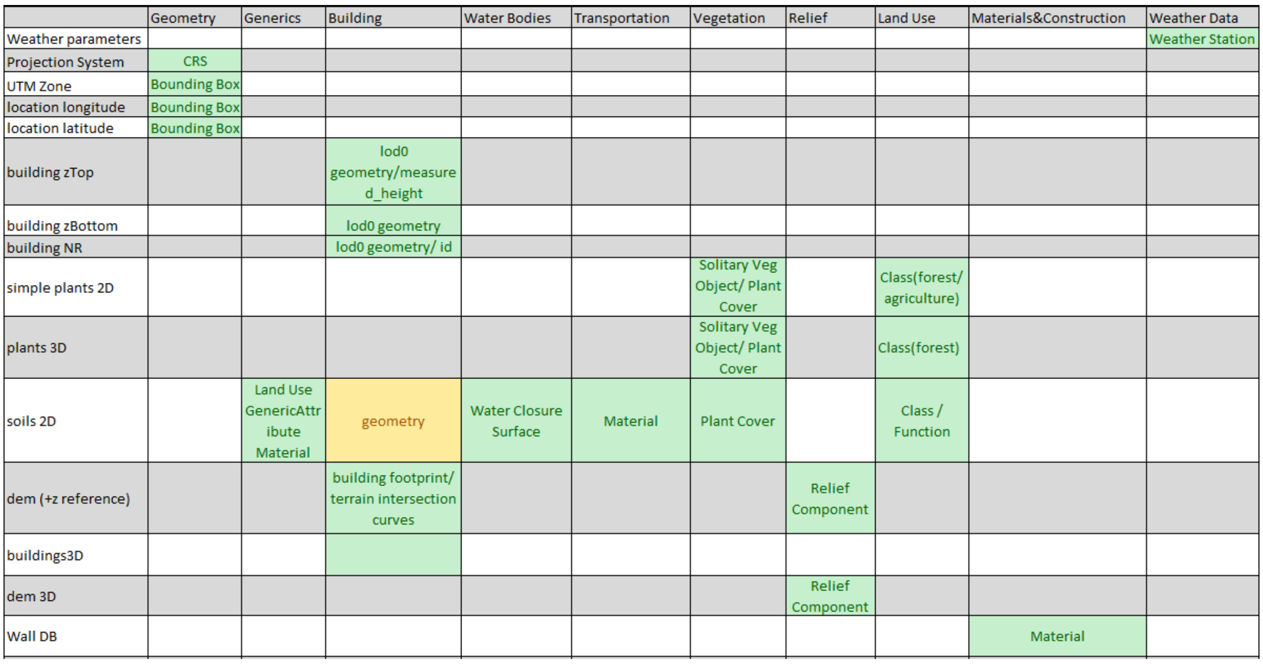
<u>Supporting Classes – Time Series -Schedules – Weather Data</u>

In Energy studies time is an important factor that influences many dependent parameters. For example, weather parameters depend on the seasons and time of the day. The same applies to the heating and cooling demands of buildings. For that reason, the thematic models of Energy ADE use a number of supporting classes, one of which is *Time Series. Time Series* can be divided in regular and irregular. In a regular *Time Series* class, the temporal information is defined by a beginning timestamp and a constant incremental value. Irregular time series refer to a counted parameter without any ordered time step. In this case, each occurrence must be accompanied by its timestamp. All types of time series contain metadata attributes called *TimeValuesProperties*, which picture the described parameter (*thematic description)*, the acquisition method (*acquisition-method)*, and the interpolation type.

The relation weatherData enables to connect any city object with time series of meteorological or radiation parameters (WeatherData). This time-related weather data can be the required input of ENVI-met meteorological parameters if exists. Also, this time series type could be used as a storage option for ENVI-met meteorological outputs (atmosphere). In general time series is a handy data model to store any kind of ENVI-met outputs since it incorporates the same time-based mechanism (ENVI-met produces a number of outputs for each hour).

3.5 Mapping between ENVI-met and CityGML data

In this section the input model's classes were analysed and the question 'Can ENVI-met requirements be met by CityGML and the Energy ADE?' was answered. The outcome of the research was that all the important semantic - spatial requirements of ENVI-met's Area Input File can be fulfilled by the CityGML data model. Furthermore, Energy ADE can assist by providing additional data which can enrich the Area Input File by means of materials, but also the *Weather_Station* class can potentially satisfy ENVI-met weather requirements. Figure 31 provides a better illustration of the correspondences between the two data models. Due to the lack of available data for generating an augmented – by means of Energy ADE - dataset, Energy ADE is not supported as input in the first phase of SclGModeLer. However as mentioned in the beginning of this section and will be also discussed on later chapter the methodology to retrieve and map Energy ADE data would be identical.



	Geometry	Generics	Building	Water Bodies	Transportation	Vegetation	Relief	Land Use	Materials&Construction	Weather Data
Weather parameters										Weather Station
Projection System	CRS									
UTM Zone	Bounding Box									
location longitude	Bounding Box									
location latitude	Bounding Box									
building zTop			lod0 geometry/measured_height							
building zBottom			lod0 geometry							
building NR			lod0 geometry/ id							
simple plants 2D						Solitary Veg Object/ Plant Cover		Class(forest/ agriculture)		
plants 3D						Solitary Veg Object/ Plant Cover		Class(forest)		
soils 2D		Land Use GenericAttribute Material	geometry	Water Closure Surface	Material	Plant Cover		Class / Function		
dem (+z reference)			building footprint/ terrain intersection curves				Relief Component			
buildings3D										
dem 3D							Relief Component			
Wall DB									Material	

Figure 31: CityGML and Energy ADE classes that satisfy ENVI-met area input file's requirements

4. Coupling framework and settings

This chapter outlines the main steps towards the development of the bi-directional data mapping. In other words, it will answer the sub-questions “How to perform the mapping in terms of data model” and “How to implement the interface that reads/writes data”.

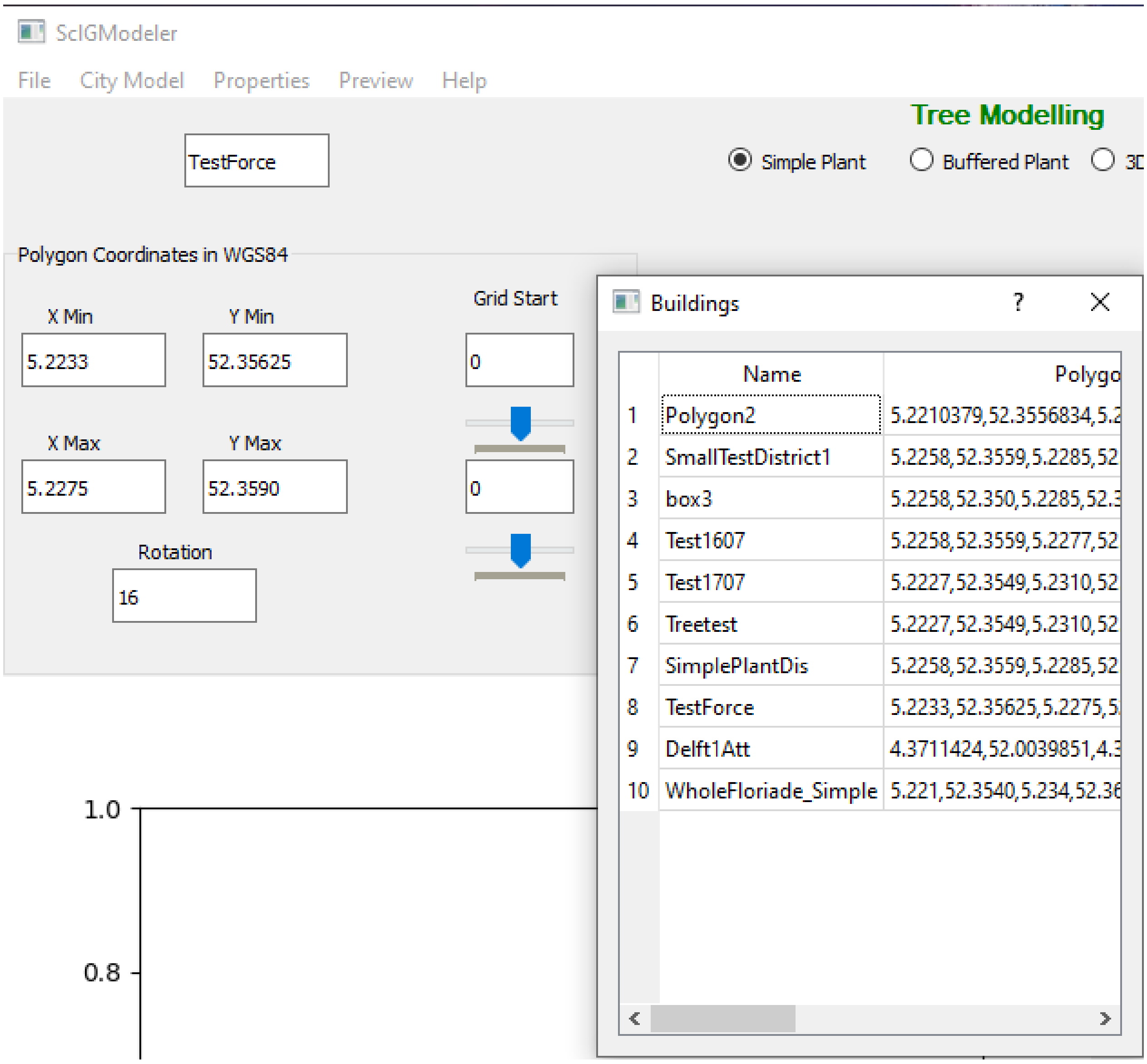
In chapter, 3 ENVI-met and CityGML + Energy ADE data models were discussed thoroughly. As mentioned in section 3.3 the 3DcityDB relational structure of the object oriented CityGML model was used as the input source for the first phase of the mapping as well as for data storage.

The following sections will describe the conceptual and programming framework which is required to realize the data model transformation. The aim is to develop and explain the methodology behind the function of SclGModeLer. The subject of microclimatic simulation that is addressed in this thesis requires a more practical approach, rather than implementing the methodology in a programming environment. Therefore, this work focuses on the creation of a GUI, that would be easily used by both ENVI-met users and CityGML / energy domain experts.

4.1 Interface concept and user inputs

The need for the creation of a GUI for managing the directories and workspaces and inserting the required user inputs, becomes evident during the process that creates the area input files. The role of the implemented methodology is not limited in reading a file and providing an output. It rather aims to equip the user with dynamic control of the spatial parameters of a large city model, without pausing the code.

The user, through this CityGML – ENVI-met coupling approach can create models of districts, out of the CityGML dataset, by just importing a cutting rectangle bounding box (cutting polygon) as it is displayed in figure 32. In order to form a cutting polygon, the interface requires a minimum and maximum point coordinates. The user is able to save a cutting polygon in order to reuse it. The stored polygon can later be reloaded through a separate Dialog window. After the cutting polygon is set, the user defines the horizontal resolution of the grid. This particular sequence in which the parameters are defined provides additional flexibility since from one input, the output models can vary both in resolution and in spatial extent. In particular cases were an area has to be split in smaller ones, to avoid extremely costly simulations, this approach is even more flexible, since it does not need new inputs or input processing, but just the translation of the cutting polygon.



TestForce

Polygon Coordinates in WGS84

X Min	Y Min
5.2233	52.35625

X Max	Y Max
5.2275	52.3590

Rotation: 16

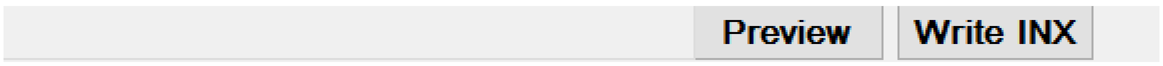
Grid Start: 0, 0

Simple Plant [✓]

	Name	Polygo
1	Polygon2	5.2210379,52.3556834,5.2
2	SmallTestDistrict1	5.2258,52.3559,5.2285,52
3	box3	5.2258,52.350,5.2285,52.3
4	Test1607	5.2258,52.3559,5.2277,52
5	Test1707	5.2227,52.3549,5.2310,52
6	Treetest	5.2227,52.3549,5.2310,52
7	SimplePlantDis	5.2258,52.3559,5.2285,52
8	TestForce	5.2233,52.35625,5.2275,5.
9	Delft1Att	4.3711424,52.0039851,4.3
10	WholeFloriade_Simple	5.221,52.3540,5.234,52.36

Figure 32: SclGModeLer Interface, Selecting previously saved cutting polygons

One additional aspect of SclGModeLer is that it allows the user to visualise the imported data before proceeding to the mapping. More specifically, the user is provided with a preview of how the data will be mapped before the actual process takes place (Figure 33).



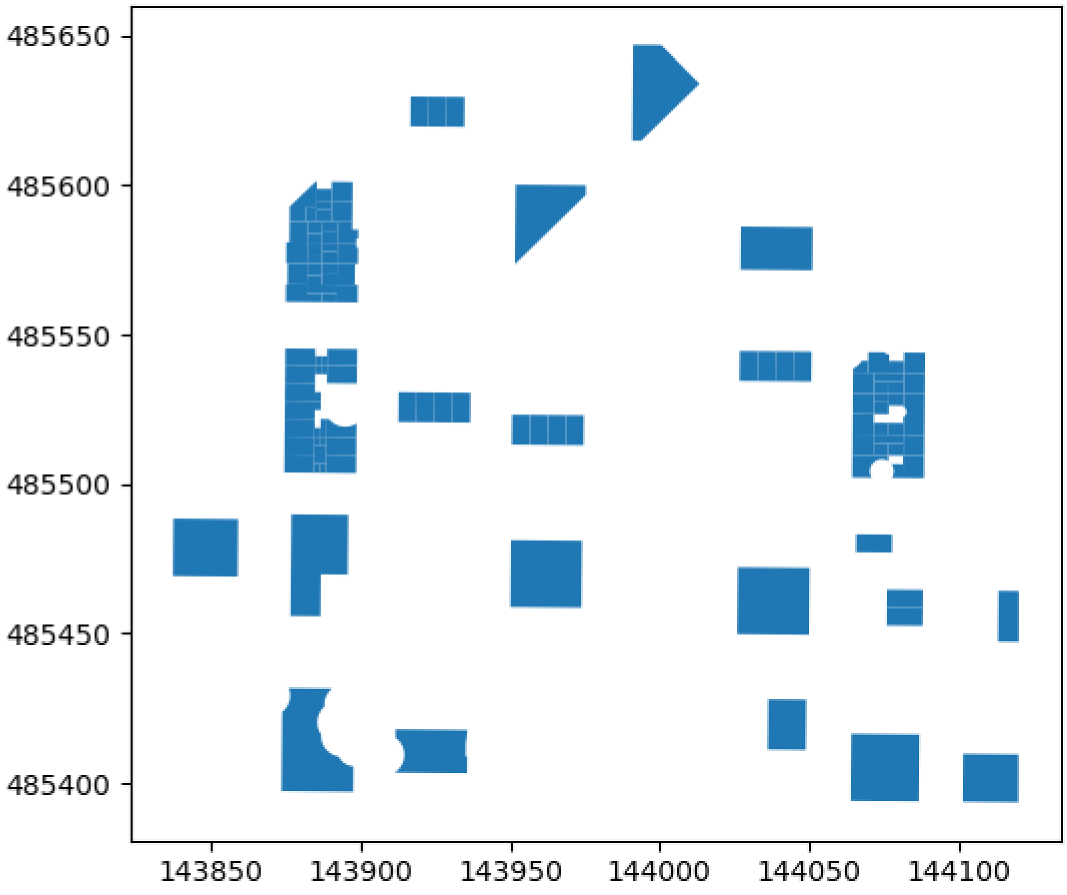


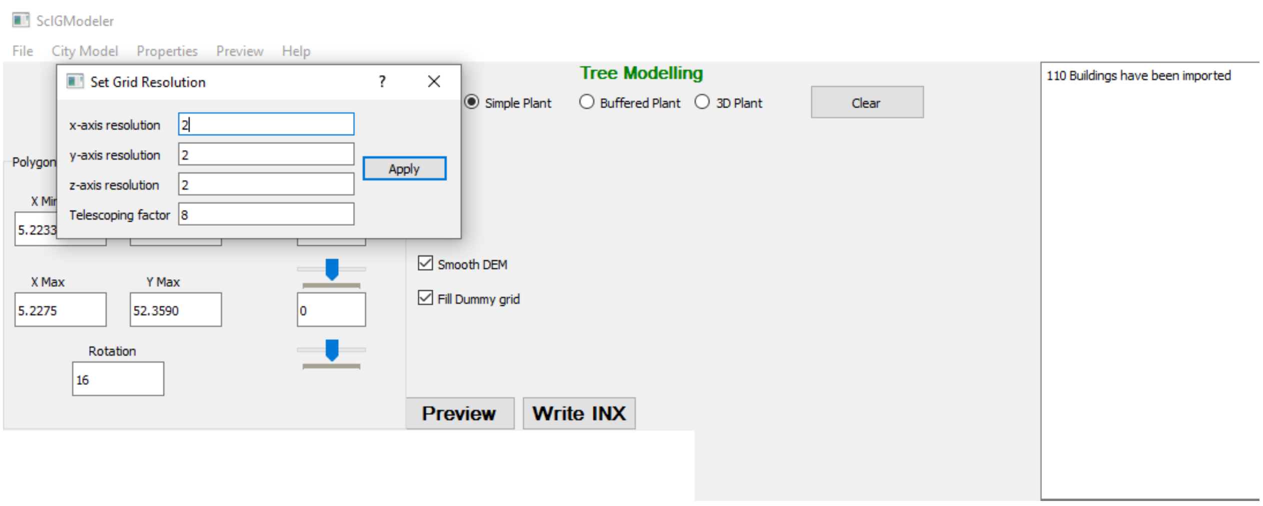
Figure 33: SclGModeLer Figure Canvas, Plotting data before proceeding to actual mapping

The selection of the cutting polygons substitutes the manual selection or creation of geometry (that takes place on MONDE and Rhino plugin respectively). Furthermore, in SclGModeLer the user is in control of the grid initialization parameters, as it is also the case in SPACEs, MONDE and Rhino. However, in this approach, the user inputs are slightly modified: For example, the number of grid cells is calculated automatically when the input extent is provided, and the resolution is set. The vertical grid is set by default to a telescoping grid, and the extension (telescoping) factor starting from the height of the highest object. The default soil materials are pre-selected to the SPACEs defaults but can be manipulated as well. What is also different, in this case, is that metadata of the area is already contained on the CityGML dataset (and thus it does not need to be provided).

More specifically, as long as the user already has a 3DCityDB instance -filled with the required CityGML data-, the inputs to set up SclGModeLer are the following:

- Database Credentials ('Name', 'password', 'port', 'User')
- Cutting polygon (minimum, maximum coordinates in WGS84)
- Model rotation (degrees, counterclockwise)
- Model translation over the grid (from -2 to 2 meters)
- Tree type selection (For implicit vegetation objects – will be discussed in section 4.2.2)
- Default wall materials
- Default roof materials
- Resolution (in x and y axis)
- DEM smoothing and Captured Grid handling (on by default)

The number of required inputs further emphasizes the need for an interface to be flexible (figure 34).



x-axis resolution: 2
y-axis resolution: 2
z-axis resolution: 2
Telescoping factor: 8

Simple Plant [✓] | Buffered Plant | 3D Plant

110 Buildings have been imported

Polygon
X Min: 5.2233
X Max: 5.2275
Y Max: 52.3590
0
Rotation: 16

☑ Smooth DEM
☑ Fill Dummy grid

Figure 34: SclGModeLer interface, toolbar and user input options (displayed: grid parameters)

4.2 From CityGML to ENVI-met. Conceptual framework and requirements.

This section will describe the conceptual framework behind the implementation of the methodology for the creation of the first phase's code. The main aims here are the successful conversion of data from vector to raster format a well as the creation of a robust interface that will process every dataset and map the existing data to area input files. Moreover, this section will address the main goals of the interface, as far as the quality of the constructed area input file is concerned. Finally, it will divide between the user-defined inputs and the automatic data flow.

4.2.1 The challenges of rasterization and robustness

The goal of the first phase is to transfer the continuous, vector-based urban data to ENVI-met discrete 3-dimensional grid. This will be achieved by locating and retrieving the required data based on the CityGML semantics, by its corresponding tables, and afterwards transform it based on its geometry and/or spatial attributes. Since the targeted format is a 2.5 INX file, the approach to be followed should be a 2-dimensional rasterization for urban features, and a 3-dimensional voxelization for the DEM.

The first step, when conducting a microclimate simulation with a raster-based software, is to define the area (spatial extent) to be modelled and setting the spatial resolution of the grid. The second step is to convert the vector geometries of urban features (points, lines polygons, solids) to the raster format. The rasterization process works in the following way. The cells of the grid that are either covered by a polygon, for more than 50% of their area, or contain a point (which represents an entity), are assigned with a respective value of the representing feature.

To achieve proper rasterization results, one should address a number of challenges. The first one has to do with the variety of different outputs, depending on the cell resolution as well as the position and rotation of the vector geometry over the raster cells. This becomes evident when we look at examples of different rasterization results including polygon geometries like buildings. For example, figures 35, 36 show two different rasterization results, of the architecture building of the TU Delft campus. Both outputs are the product of the same, robust over different resolution, rasterization algorithm. However, the rasterized model in Figure 35, despite its coarser resolution, comes much closer to the real form of the building. This indicates that higher resolution does not necessarily lead to accurate representations. For instance, the building in figure 36 is subjected to a jagged border rasterization, a concept that was discussed in section 2.2.3 and was mentioned that should be avoided.

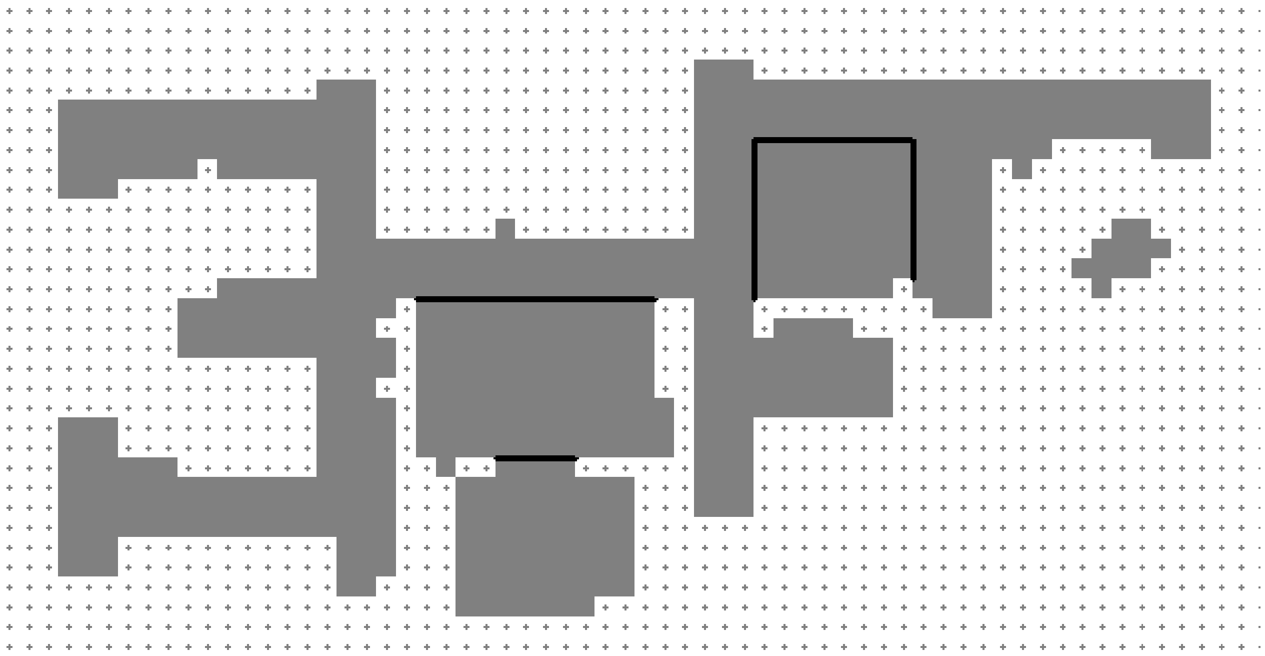


Figure 35: Architecture building in SPACES 2D view (1)

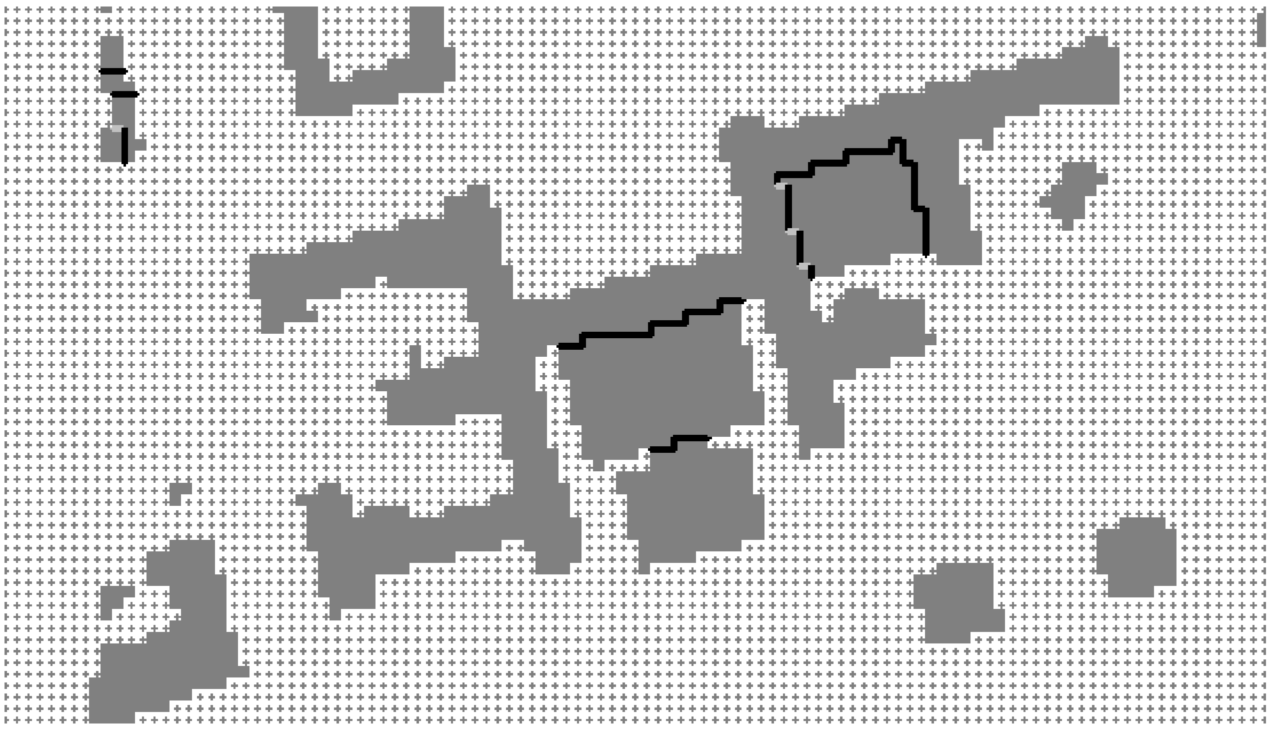


Figure 36: Architecture Building in SPACES 2D view (2)

To deal with jagged borders ENVI-met provides a rotation option that practically rotates the sun path and vector weather parameters, like wind direction, instead of the grid itself, so that it keeps the reference of the north (rotation option that was listed in section 4.1). In this way, the orientation of the model is preserved. Thus, ENVI-met users have to find the main orientation of the buildings, rotate the area so it is mostly parallel to the grid and consequently design their model. Afterwards the rotation is applied as a parameter and is written into the respective INX file metadata tag.

However, in an automatic rasterization of a large model, even with a proper rotation value, the desired rasterization is difficult to achieve. This challenge is not a serious problem for manual design users, since they care more about the shape of the objects, than their exact spacing. For this reason, one of SclGModeLer concerns is to allow the user to slightly move the vector geometry, after it is rotated and placed over the grid, and experiment with the results without having to change other parameters. This experimentation process has a great potential in combining accurate rasterization results without altering the shape of the polygons.

Another peculiarity of the rasterization process is met, when dealing with points which represent features, like CityGML solitary vegetation object anchor point. In this case, a feature is rasterized as a single cell and as a result the accuracy of the representation depends on the resolution (Figure 37). This becomes a problem when it comes to rasterizing objects, such as trees and vegetation, which are resolution dependent. For example, in an area input file, a tree is represented by a point which, through the rasterization process, will add a value to the corresponding cell of the grid. However, depending on the size of the cells, the representation of the tree in the model can acquire sizes that differ radically from the actual size of the vegetation object. In order to address this challenge, a major goal of the interface is to be robust over different resolutions.

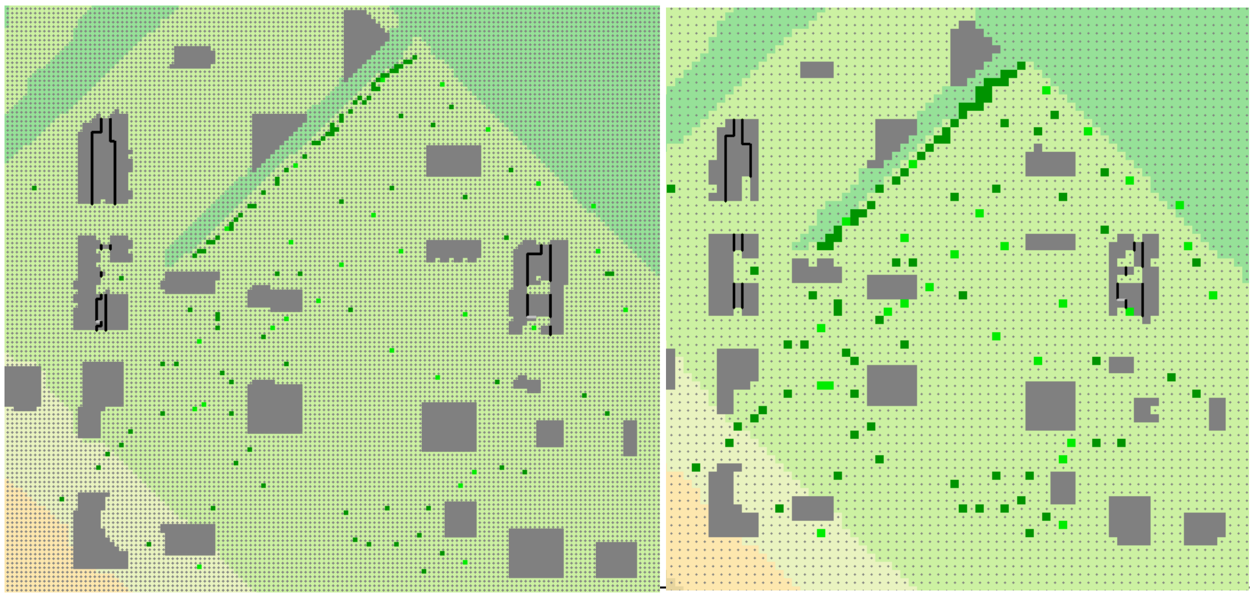


Figure 37: Floriade District area input file displayed in SPACES 2D viewer, simple trees are resolution dependent

An additional challenge for achieving proper rasterization results, is to be aware of the logical relations between features. For example, a tree should not be placed on a building cell, nor on the water. These logical relationships are usually conceived in a 'good' CityGML dataset, however if this is not the case, or if the relationships are disrupted as a result of the rasterization process, the interface should apply corrections whenever needed. Furthermore, the interface needs to ensure that the terrain bellow a building entity is flat, in order to correspond to ENVI-met's building columns. (which follow the DEM by default) In some cases this option is already provided by the use of the Terrain Intersection Curves concept in CityGML. However, when not provided, it should be achieved by the use of the interface.

Different INX files were created via the interface and ran through the whole simulation extent, so it is believed that the interface outputs meet at least all the important specifications. First the overlapping of different urban elements is addressed. For instance, in the area input file, it is important that-plant cells do not overlap with buildings. More specifically for the use of 3D trees this applies only to the cells of main column (where the tree is placed on the 2D matrix), and the crown radius is allowed to overlap. Additionally, there are some limitations concerning the height of the objects in the ENVI-met model. The total height of the model should be at least twice the height of the tallest object and at least 30 meters height. Finally, 'captured cells' should be avoided. According to ENVI-met a captured grid is a grid that contains at least a free grid cell that is surrounded by building/dem -occupied cells only. This is causing turbulence instability and will probably return unrealistic results in that cell but will not crush the simulations. In this project particularly, where weather parameters were aggregated to building level, the accuracy of the results on every empty cell is a matter of high importance, so the integrity of the values in each cell has to be ensured.

4.2.2 Layer approach in the area input file

The INX file's architecture can be considered as an order of sequential layers. Firstly, the DEM is constructed where all the objects are placed by default. Then the soils are assigned, the trees and the buildings. Inspired by this design a layer approach was followed in the implementation as well, aiming to establish a hierarchical order in editing objects and more importantly materials, in order to improve the quality of the resulting area input model, and to ensure robustness over different CityGML datasets. The interface can parse a CityGML dataset and understand its content by checking whether or not classes are filled with data (this approach can be extended to cover various LODs if they are available). By following this approach two facts are secured: First, the methodology can distinguish between more important features, while reading data from two different tables, intended for the same matrix. For instance, when a *Transportation* object is read, it will update the grass profile that was potentially set by a previous derived *LandUse Agriculture* object, to asphalt. In the same way, if an imported vegetation object is set to a cell occupied by building or water, it will be automatically deleted, or moved to a neighbour cell.

Additionally, when a building is inserted to the model, it will smooth the DEM cells on the respective matrix positions.

SclGModeler has been tested on available CityGML models (other than the Floriade model that was created), which are available on https://3d.bk.tudelft.nl/opendata/3dfier/ and it succeeded in writing valid INX files (Figure 35,36) As long as the required data exists, the interface will update the model, layer by layer, starting from the DEM and the <soils> matrix which is pre-set to a default profile. Thus, even if the data model is empty of data, or the cutting polygon used as an input does not intersect any features, the result will be just an empty INX file.

Figure 38 illustrates the mapped classes and attributes that are used in SclGModeler along with the hierarchy rules (from left to right). The mapping can be updated to include further classes and attributes, following exactly the same principles. As the basic source for the <soils> matrix a 'materials' *Generic Attribute* for the *LandUse _multisurfaces* was created; (its codelist is displayed on figure50) inspired by the *Transportation Module material* attribute. This material was set to a higher hierarchical order than *LandUse* class and function attributes, except if their values are "Forest". It is believed that such an attribute could be applicable and essential for energy and microclimatic simulations, and could be added on the CityGML data model, probably in a higher *LandUse* LOD (urban scale).

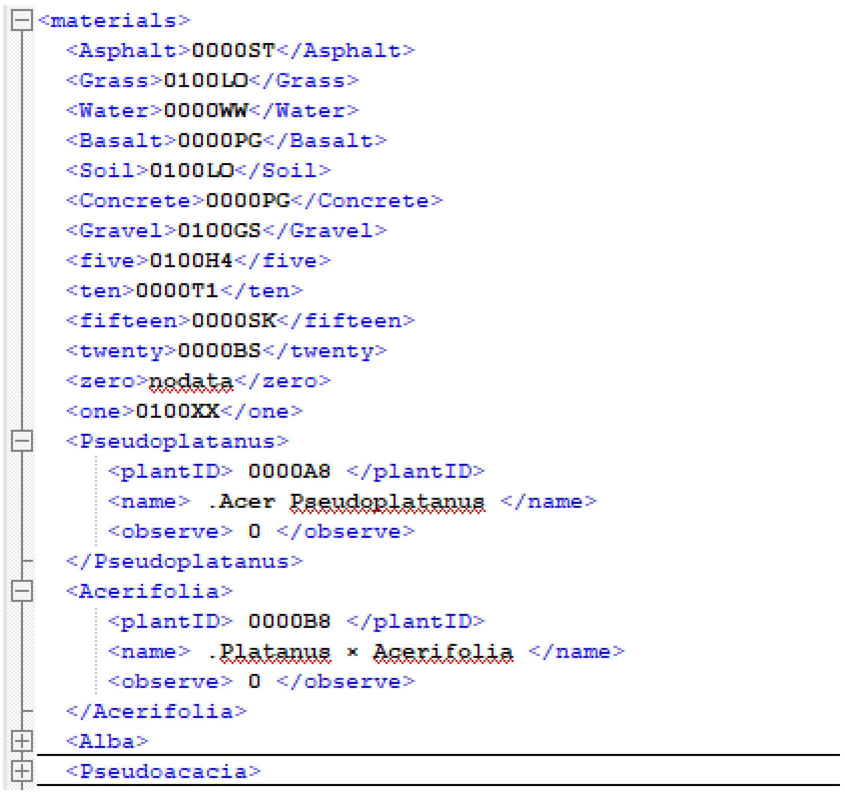
Weather parameters	User input		
Projection System	CRS		
UTM Zone	Bounding Box		
location longitude	Bounding Box		
location latitude	Bounding Box		
building zTop	*Building, lod0_footprint_geometry. measured height*		
building zBottom	*Building, lod0_footprint_geometry*		
building NR	*Building lod0_footprint_geometry*		
simple plants 2D	Vegetation, Solitary Vegetation Object ,height&crown diameter)		
plants 3D	*Plant_Cover, Usage 'Forest'*	Vegetation, Solitary Vegetation Object height&crown diameter)	
soils 2D	*Building* = (default soil)	*Land Use, Genericattribute* (Material)	*Plant_Cover, Usage 'Forest'*
dem (+z reference)	*TIN_Relief*		
buildings3D	(based on buildingzTop and dem)		
dem 3D	(based on dem)		
Wall DB	(Open, code implemented for Energy ADE *Material and Construction, material)*		

Figure 38: Classes and attributes of CityGML that are currently used by SclGModeLer, hierarchy from left to right

4.2.3 Semantic mapping

Section 2.2.2 and 2.2.3 presented the main drawbacks of designing an area input file in a raster-based editor, like SPACES. The most important limitation regarding ENVI-met flexibility is the inability of converting the same model from one resolution to another. In that case the user had to initialize a different grid from the beginning, and then design the area all over again. By using SclGModeLer for the construction of the INX files, like in Rhino Plugin or in ENVI-met Monde, a model can be depicted automatically in different resolutions.

As it was described in the problem statement (section 1.2) this project aims to improve the state of the art, by providing both geometrical and semantic interoperability between the two data models. With only geometric inputs, which MONDE and Rhino plugin use to construct the area input files, the user has to select what kind of object – profile each geometry layer represents. This requires a lot of effort, which could be avoided with the use of an automatic process. This gap can be bridged by using a semantic 3D - model, as the only input, and building a programming environment that not only reads geometry, but also interprets it. This feasible, as long as there are existing correspondences between the codespaces of the two data models. To fill this remaining piece, an XML-encoded database of one-to-one mapping between CityGML and ENVI-met-databases codespaces (Figure 39) was implemented (correspondences database). The database is kept in a separate file so that potential modifications can be applied easier.



```
<materials>
  <Asphalt>0000ST</Asphalt>
  <Grass>0100LO</Grass>
  <Water>0000WW</Water>
  <Basalt>0000PG</Basalt>
  <Soil>0100LO</Soil>
  <Concrete>0000PG</Concrete>
  <Gravel>0100GS</Gravel>
  <five>0100H4</five>
  <ten>0000T1</ten>
  <fifteen>0000SK</fifteen>
  <twenty>0000BS</twenty>
  <zero>nodata</zero>
  <one>0100XX</one>
  <Pseudoplatanus>
     <plantID> 0000A8 </plantID>
     <name> .Acer Pseudoplatanus </name>
     <observe> 0 </observe>
  </Pseudoplatanus>
  <Acerifolia>
     <plantID> 0000B8 </plantID>
     <name> .Platanus × Acerifolia </name>
     <observe> 0 </observe>
  </Acerifolia>
  <Alba>
  <Pseudoacacia>
```

Figure 39: Correspondences database implemented on SclGModeLer

The role of this database will become clearer with the following two examples:

1. The interface reads from the buildings table a building's LOD0 footprint geometry, the *measured_height* attribute and its database id. It performs the rasterization and fills the <buildings2D> matrices with the height and the id values. This is a geometric and straightforward transformation that does not require any semantic mapping.
2. The interface reads from the *LandUse* table, a polygon geometry, and the *GenericAttribute* material which is for instance "Concrete". It performs the rasterization and it is programmed to assign a soil material in the occupied cells. However, ENVI-met has its own database, described in section 3.3.2 which uses a unique codelist as a link. Thus, the interface goes through the correspondences database and translates the attribute information to the soil profile (6-digit codespace).

As a result, with the use of the correspondences database, all the transformations are proceeded automatically, without the involvement of human factor. An exemption is the representation of solitary vegetation objects, where the user has the ability to choose between four different tree types. (section 4.1, tree type user input). (Since ENVI-met also offers two different options for tree representation, the choice was shifted to the user This allows the construction of four different area input files, with a different tree approach, to simulate/ test different scenarios. More specifically, the available options also displayed in Figure 40 are the following:

- No trees
- Simple trees (resolution dependent)
- Simple Buffered trees where neighbour cells covering the crown diameter are converted to simple trees as well (resolution independent)
- 3D trees (resolution independent)

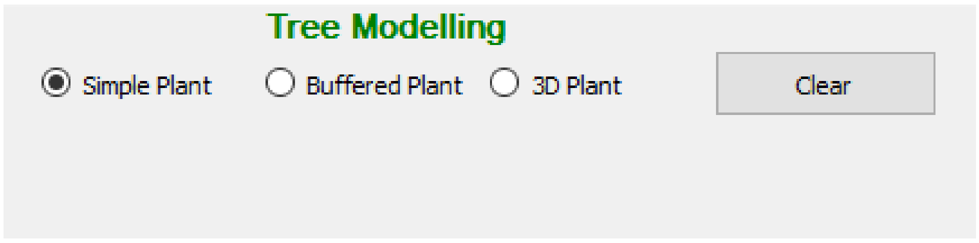


Figure 40: User input for modelling Solitary Vegetation Objects

4.3 Technical Implementation (First Phase)

After the overview of the conceptual approach behind the coding environment, described on the previous section, this section will list the required technical steps to materialize the methodology (technical methodology). The interface works in two phases; the technical steps of the first phase will be explained one by one in this section. Section 4.4 will outline the main steps of the second phase. The queries used for reading data from 3DCityDB are displayed in Appendix 2. All the coding was implemented in Python programming language. The data processing, as well as the rasterizations and the DEM voxelization happened in memory, while the intersections were shifted to the PostgreSQL RDBMS with and SQL queries. For the accomplishment of the methodology, the following libraries and data formats were used: The Psycopg2 library was used to establish connection with the 3DcityDB instances. The PyQT5 library was used to create the GUI. The geopandas library with the geodataframe data structure was used to store the retrieved data to/from the 3DcityDB. The shapely library was used (which is used by the geopandas as well) to transform the cells into rectangle polygon geometry. Numpy and Scipy libraries were used for the DEM voxelization. Finally for the writing of the INX file the xml.etree library was used.

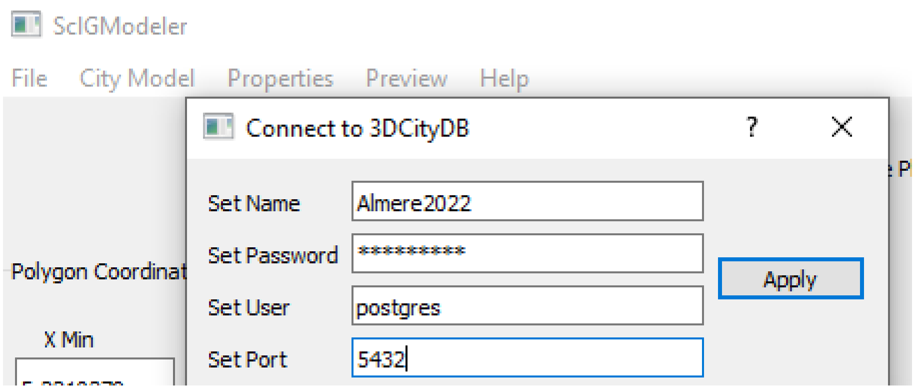
4.3.1 Setting up a PostgreSQL 3DcityDB instance

The first step of the methodology is to access the data stored in a CityGML dataset. In this project, 3DcityDB relational schema was used for managing, retrieving and storing data and this section will outline the requirements of this input framework. Setting up a 3DCityDB instance requires a running installation of a PostgreSQL database server. Currently only PostgreSQL 9.1 of higher with PostGIS extension 2.0 or higher are supported. The 3DCityDB software package contains a set of SQL scripts to create the relational schema (which also includes the Energy ADE tables). The 3DCityDB package, containing the importer – exporter tool, as well as the SQL set, can be downloaded by (https://www.3dcitydb.org/3dcitydb/), together with a tutorial about loading data into a 3DCityDB instance.

The cost of an intermediate layer, being added to the whole process is negligible compared to the advantages of the database approach. After all the end user of the interface doesn't have to be a database expert, since all the querying and table handling is implemented inside the interface. Lastly, the addition of a separate schema to the database instance can be a handful and elegant way of storing intermediate results and useful data and metadata by/for the interface (for example selected rectangles for filtering CityGML Data which can be reused). In this way automatization is supported and protection of interface-generated data is ensured.

4.3.2 Steps to create an area input file from the CityGML data model

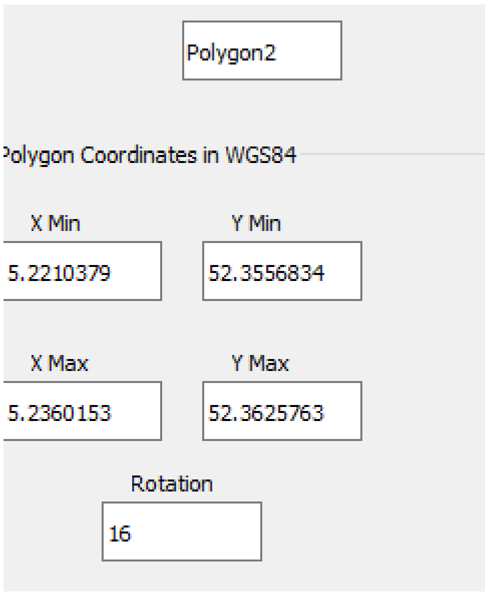
The first phase was implemented in a sequence of short steps. The first one consists of establishing a connection with the Database instance, where the CityGML dataset is stored. As described in section 4.1, this step requires the credentials of the 3DcityDB instance which constitutes the first user input. The credentials are kept in memory and will be used each time data needs to be retrieved from the database instance, either CityGML data, or cutting polygons that were previously stored. In the second phase, the output results will be stored in the same DB instance as Energy ADE attributes (Figure 41).



Set Name: Almere2022
Set Password: **********
Set User: postgres
Set Port: 5432

Figure 41: SclGModeLer interface, database credentials are needed in both phases

The interface pre-requires that the user has already installed 3DcityDB and has successfully imported a CityGML dataset into a 3DcityDB instance. After connecting to 3DcityDB, the Coordinate system of the geometry is retrieved. At this point, the user can import a cutting polygon, from coordinates in WGS 84 World Geodetic System (in degrees) and the interface will translate it to the dataset's coordinates. This polygon must be a bounding box, specified by its minimum and maximum coordinates. The 3DcityDB importer-exporter tool includes a map window that interactively allows the selection of a 2D bounding box, which is used as a spatial filter (Figure 42). So, the coordinates of the cutting polygon can be retrieved by that map window.


Polygon2

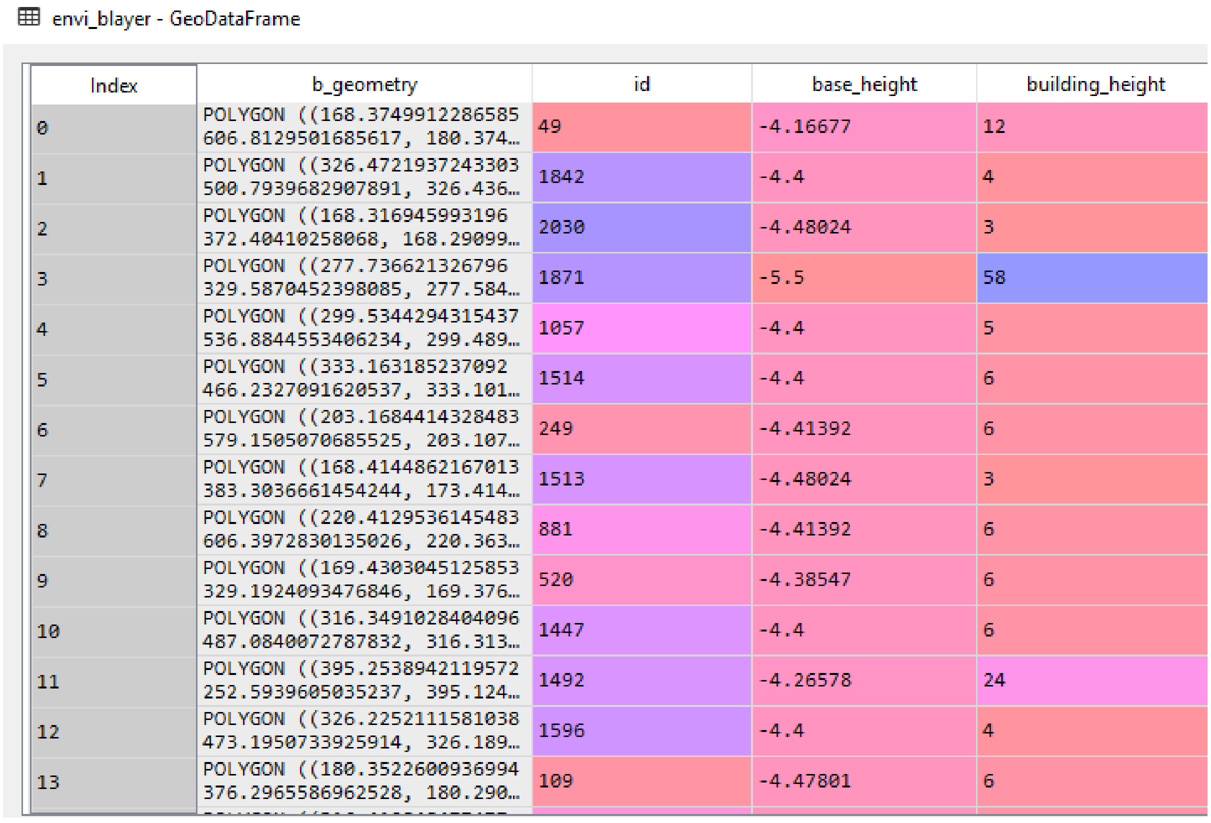
Polygon Coordinates in WGS84

X Min: 5.2210379
Y Min: 52.3556834
X Max: 5.2360153
Y Max: 52.3625763
Rotation: 16

Figure 42: Definition of cutting polygon in SclGModeLer

After importing the cutting polygon, the rotation input must be specified. The polygon is rotated by its minimum coordinates point and is used to filter the CityGML importing features. Immediately, the feature geometries are rotated by the negative rotation. This approach enables the alignment of the important features with the grid lines. At first, only the buildings are retrieved (failed intersections are aborted and buildings with height below 2 meters are filtered) and displayed through the matplotlib Figure Canvas in case the user is unsatisfied with the rotation.

Once the rotation input is verified, the required data (displayed in Figure 40) starts flowing from the DB via queries which are available in Appendix 2, filling the geopandas GeoDataframes data models (Figure 43). In total, four geodataframes are created; containing the *Building*, *Vegetation*, *Land Use* and the *Relief* features. A Geodataframe object is a pandas Dataframe table that has a mandatory column of geometry in a reference system. It is a proper data type to store data retrieved from tables, since it has a similar structure, it supports a direct method to read PostgreSQL with PostGIS tables by translating the geometry to shapely objects and it also provides fast methods for geographic operations, like geometry rotation, x-y translation or translation to different coordinate systems. Other spatial operations in geopandas are performed by shapely (https://shapely.readthedocs.io/en/stable/manual.html). All imported data is filtered by the rotated cutting polygon, except the TIN Relief that is filtered by an 100m buffer of that polygon. The buffered polygon is used to specify the extent of the area input file. In this way objects are restricted to the centre of the model and their required distance from the borders is ensured. Then the number of nested cells (see section 3.3.1) is set to zero.



envi_blayer - GeoDataFrame

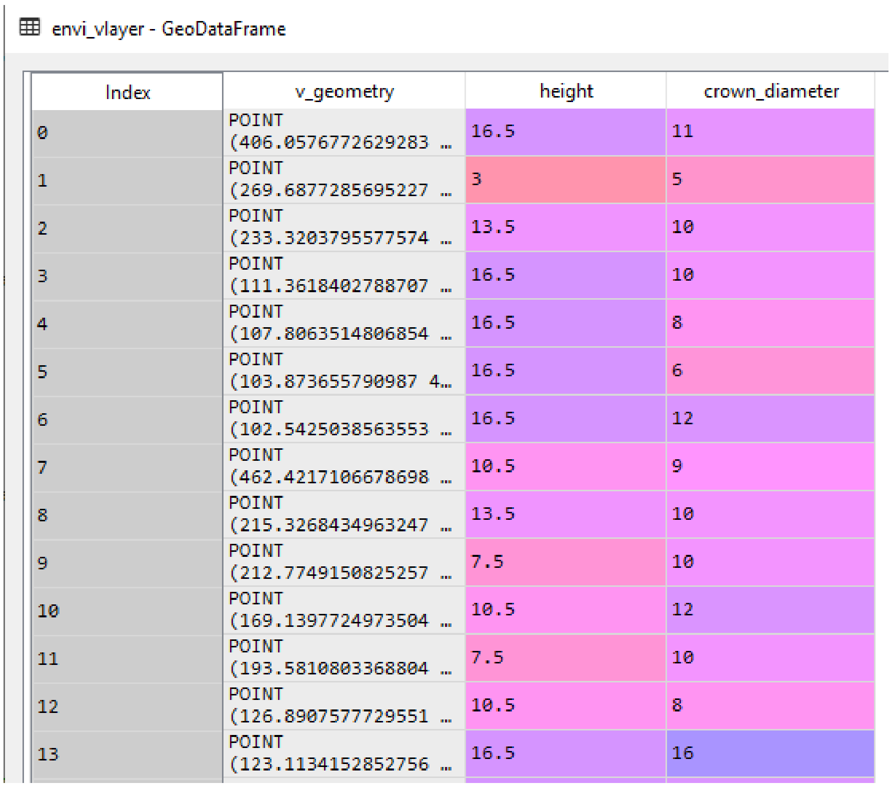
Index	b_geometry	id	base_height	building_height
0	POLYGON ((168.3749912286585 606.8129501685617, 180.374…	49	-4.16677	12
1	POLYGON ((326.4721937243303 500.7939682907891, 326.436…	1842	-4.4	4
2	POLYGON ((168.316945993196 372.40410258068, 168.29099…	2030	-4.48024	3
3	POLYGON ((277.736621326796 329.5870452398085, 277.584…	1871	-5.5	58
4	POLYGON ((299.5344294315437 536.8844553406234, 299.489…	1057	-4.4	5
5	POLYGON ((333.163185237092 466.2327091620537, 333.101…	1514	-4.4	6
6	POLYGON ((203.1684414328483 579.1505070685525, 203.107…	249	-4.41392	6
7	POLYGON ((168.4144862167013 383.3036661454244, 173.414…	1513	-4.48024	3
8	POLYGON ((220.4129536145483 606.3972830135026, 220.363…	881	-4.41392	6
9	POLYGON ((169.4303045125853 329.1924093476846, 169.376…	520	-4.38547	6
10	POLYGON ((316.3491028404096 487.0840072787832, 316.313…	1447	-4.4	6
11	POLYGON ((395.2538942119572 252.5939605035237, 395.124…	1492	-4.26578	24
12	POLYGON ((326.2252111581038 473.1950733925914, 326.189…	1596	-4.4	4
13	POLYGON ((180.3522600936994 376.2965586962528, 180.290…	109	-4.47801	6

Figure 43: Geodataframe in python Spyder IDE, contains building required data

The next step is the 3D grid initialization. The dimensions are calculated by dividing the cutting polygon dimensions with the edited resolution, and the reference height is derived by the minimum z coordinate of the model's 3D Bounding box. Then a 2 Dimensional grid is stored as a normal python dictionary, where the key define the x,y position of every cell starting from (0,0), and the values are set to zero. A Numpy array was examined as an alternative, more structured way of storing the grid, but with this limited number of elements, the dictionary is considered satisfactory. The speed performance of the implementation was tested and models up to 250*250 cells can be created in less than a minute, which is considered negligent, compared to the time of the simulations.

Once the grid is set the geodataframes are rotated back and translated to the grid's reference system (starting from (0,0)). If the user desires a slight movement for a more appropriate rasterization, the input values are added to this translation. The total x, y translation values are stored, along with the rotation, in the INX metadata tags, since their exact value is required for the reverse-transformation in the second phase. Once the two geometries overlay, the features are ready for the rasterization. Before every geodataframe rasterization, a copy of the dictionary grid is generated, following the layer approach discussed in section 4.1.1.

For the point geometries, which is the *Vegetation* geodataframe (Figure 44), the objects are traversed and registered to the corresponding cells as (height, crown diameter) tuple type values. If more than one vegetation objects coincide in a cell, then the highest object prevails.



envi_vlayer - GeoDataFrame

Index	v_geometry	height	crown_diameter
0	POINT (406.0576772629283 …	16.5	11
1	POINT (269.6877285695227 …	3	5
2	POINT (233.3203795577574 …	13.5	10
3	POINT (111.3618402788707 …	16.5	10
4	POINT (107.8063514806854 …	16.5	8
5	POINT (103.873655790987 4…	16.5	6
6	POINT (102.5425038563553 …	16.5	12
7	POINT (462.4217106678698 …	10.5	9
8	POINT (215.3268434963247 …	13.5	10
9	POINT (212.7749150825257 …	7.5	10
10	POINT (169.1397724973504 …	10.5	12
11	POINT (193.5810803368804 …	7.5	10
12	POINT (126.8907577729551 …	10.5	8
13	POINT (123.1134152852756 …	16.5	16

Figure 44: Geodataframe containing tree data

The same applies to the polygon geometries such as the *Land Use* and *Buildings* geodataframes. *Building* cells receive a (*measured_height, id)* value while *LandUse* cells are assigned a (*Material, Class, Function*). Failed intersections will be aborted, returning a warning message. For the TIN voxelization, a TIN interpolation algorithm was used. Since the product of the intersection with a polygon is not a TIN anymore, the vertices of the resulting multisurface were derived, and used as an input. By the vertices a new TIN was constructed with a Scipy, Delaunay triangulation method.

A series of Interrelations between these four grid layers will be applied, after the rasterizations are finished, to adhere the hierarchy order and the rules that were described in section 4.2.2. These four grids can be directly transformed to the required matrix data type to fill the building, soils, vegetation and dem2D tags.

The final step is to create the sparematrix-3D data type to fill the 3D tags. As mentioned in section 3.1, this data representation type requires the absolute position of cells. In case of the 2.5 area input file this is ensured in a relatively simple way, by extruding the buildings from the DEM, up to their height and reverse extruding the dem to zero. The set of cells is then stored as a dictionary, keyed by the 3-dimensional position of the cell. In the case of the <walldb> tag, a material is assigned in a compatible cell façade if it belongs to a building edge. Candidate cells that will host the material information can be identified by checking their adjusting cells and see if they are occupied by buildings or not. The 3 dictionaries, representing buildings, dem and materials require a sorting on the 3 dimensions before written to the required sparematrix-3D tags.

Once all the data has been transformed the INX file can be created with the help of the ElementTree library.

4.4 Technical Implementation (Second Phase)

This section will list the technical steps, required to return ENVI-met outputs back to the input dataset. For this second phase of the interface, the coding was implemented in Python programming language. The outputs of ENVI-met were processed in memory, and finally uploaded to the 3DCityDB instance, where geometrical operations were executed. The following libraries and tools were required. First, the Psycopg2 library was used to establish a connection with the 3DCityDB instance. Furthermore, the pandas library as well as the xarray were used, to read binary outputs and store them in a Dataframe where they could be further processed. For the geometric validation of the reverse-transformation of the geometry, on this second phase, QGIS and FME were used.

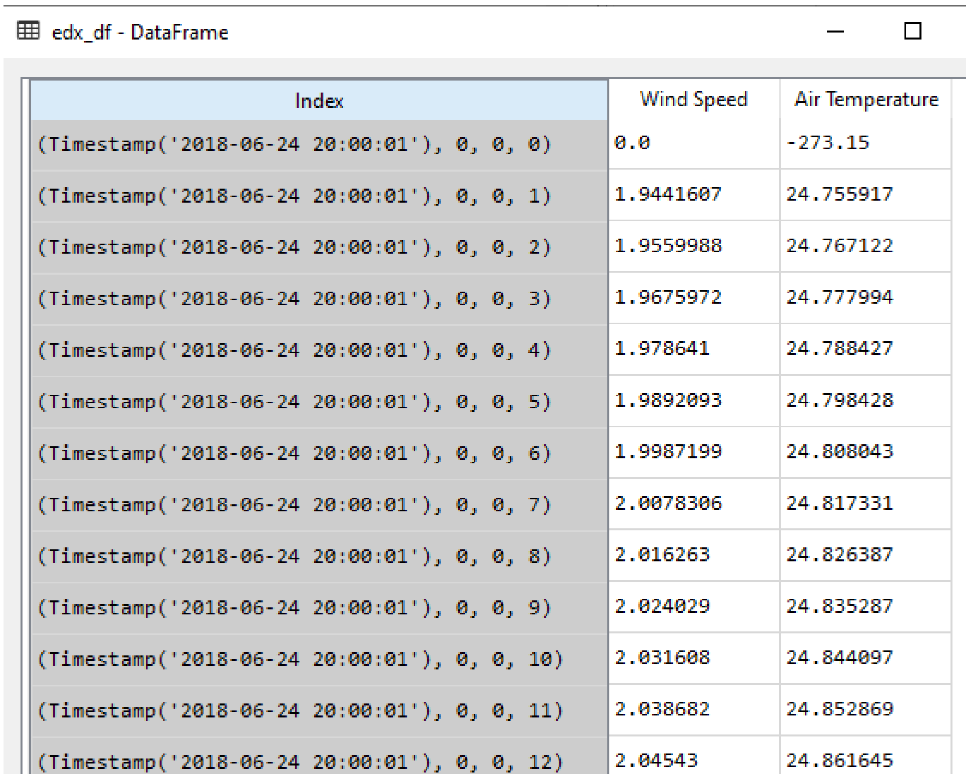
4.4.1 Enriching the model with simulation outputs

This thesis examined the feasibility of creating an automatic flow of data between CityGML + Energy ADE and ENVI-met. For this purpose, the air temperature output values of ENVI-met, included in the Atmosphere folder, were processed and mapped back to CityGML as Energy ADE *Time_Series* class and *Weather_Data* class for building objects. ENVI-met generates outputs for every hour of the simulations, and these two Energy ADE classes are applicable for storing them, since they incorporate the time dimension. The implemented methodology could be applicable to other parameters from the _Atmosphere folder (relative humidity, wind speed and solar radiation can be already mapped with the same approach) or from other folder of ENVI-met outputs. Nevertheless, each parameter requires different processing and interpolation methods to be stored in an efficient and useful way and different classes (out of the ones used), or even different ADEs could be examined. However, that also means the need of additional multiple coding branches, which was impossible during the timeframe of a master thesis.

4.4.2 Steps to map ENVI-met outputs back to CityGML

In the second phase, a methodology to enrich the input CityGML dataset by feeding average Air Temperature values to Energy ADE's *Weather_Data* and *Time_Series* supporting classes was implemented. The input data for this transformation is the EDT - EDX file pairs, stored in the Atmosphere folder. These outputs contain cell values, and not values aggregated to entities like buildings or trees, thus a geometry reverse-transformation is required. The EDT file contains data in a binary format and the EDX file is an XML encoding, which contains metadata on the general structure of data. It also contains the metadata from the area input file that was used for the simulation. Unfortunately, some information like the reference z of the model is not included and the creation of additional metadata is inevitable. SclGModeler creates a metadata text file, which needs to be placed in the Atmosphere folder before using the interface for the second phase.

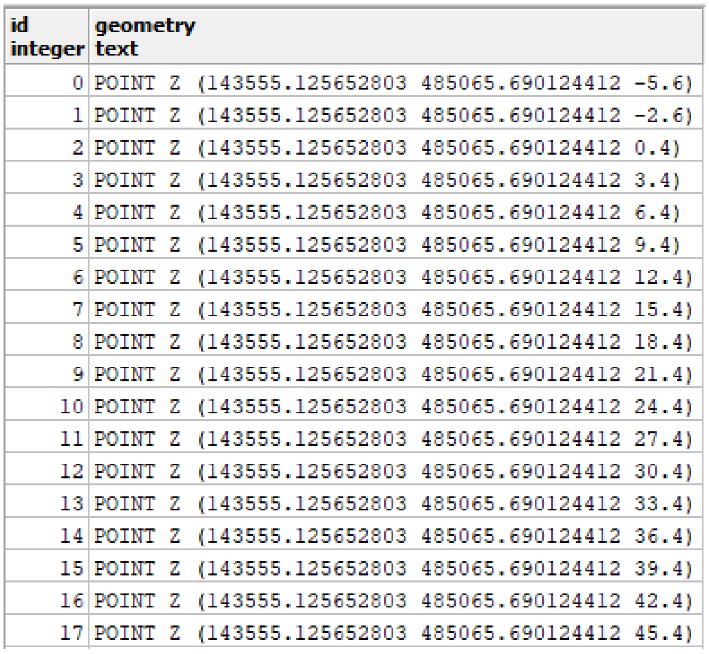
As already mentioned ENVI-met generates a huge amount of output data, so the intention is to shift the process to the RDBMS. In the web page (https://climate-cms.org/2018/04/24/reading-envi-met.html) a function that reads the binary ED pairs and store it to a Pandas DataFrame is provided (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html). The resulting DataFrame is compact, and multi-indexed, in other words its index is a composite key, composed of the timestamp and the position of every cell in the 3 dimensions. The rest of the columns refer to the parameter values (Figure 45).


edx_df - DataFrame

Index	Wind Speed	Air Temperature
(Timestamp('2018-06-24 20:00:01'), 0, 0, 0)	0.0	-273.15
(Timestamp('2018-06-24 20:00:01'), 0, 0, 1)	1.9441607	24.755917
(Timestamp('2018-06-24 20:00:01'), 0, 0, 2)	1.9559988	24.767122
(Timestamp('2018-06-24 20:00:01'), 0, 0, 3)	1.9675972	24.777994
(Timestamp('2018-06-24 20:00:01'), 0, 0, 4)	1.978641	24.788427
(Timestamp('2018-06-24 20:00:01'), 0, 0, 5)	1.9892093	24.798428
(Timestamp('2018-06-24 20:00:01'), 0, 0, 6)	1.9987199	24.808043
(Timestamp('2018-06-24 20:00:01'), 0, 0, 7)	2.0078306	24.817331
(Timestamp('2018-06-24 20:00:01'), 0, 0, 8)	2.016263	24.826387
(Timestamp('2018-06-24 20:00:01'), 0, 0, 9)	2.024029	24.835287
(Timestamp('2018-06-24 20:00:01'), 0, 0, 10)	2.031608	24.844097
(Timestamp('2018-06-24 20:00:01'), 0, 0, 11)	2.038682	24.852869
(Timestamp('2018-06-24 20:00:01'), 0, 0, 12)	2.04543	24.861645

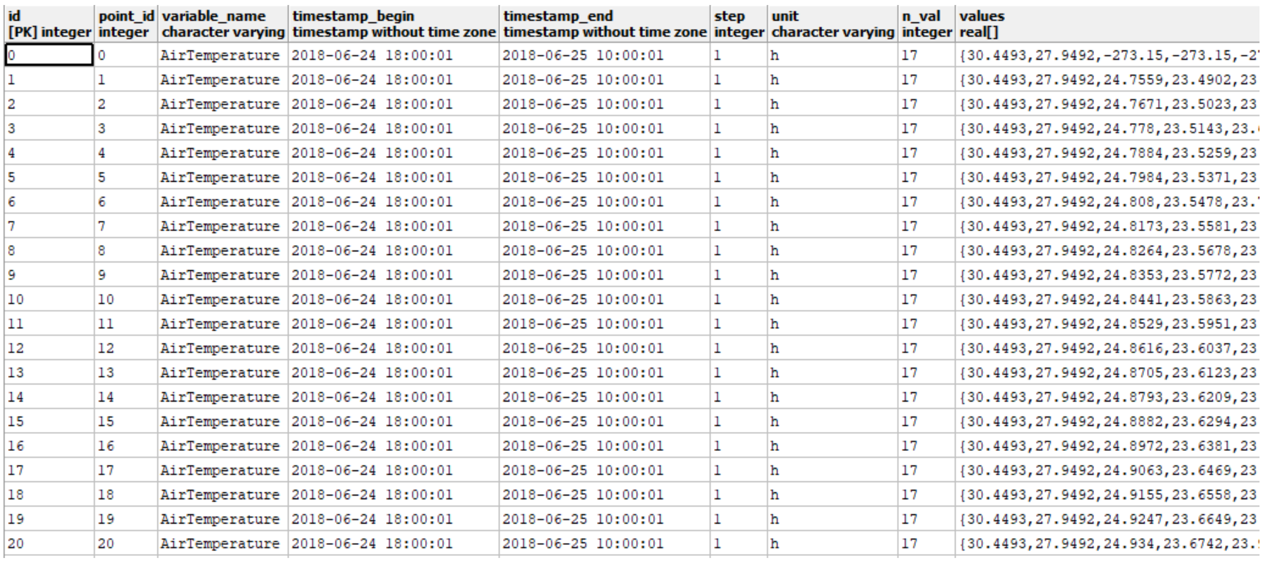
Figure 45: Compact, multi-indexed Dataframe containing ED pair data

The first step of this phase is to transform this DataFrame in a point cloud table with structure similar to the *Time_Series* 3DCityDB table (Figure 64). First the composite key has to be split so the geometry can be extracted as the cell's centroid point. Afterwards, by applying a reverse transformation, it can be translated back to the dataset's coordinate system. As a new DataFrame key, a serial is automatically assigned and a second column (point_id) is created that displays the sequential order of each cell. The remaining columns will contain the timestamp of the first entry, the timestamp of the last entry, the parameters name, an array of hourly values and the total number of entries. The DataFrame transformation is applied in memory and the rest is shifted to the Database. The optimal approach for space and speed efficiency is to split the DataFrame to a geometry table ("geom") and a data table ("inxar") constrained by the point_id index (Figure 46, Figure 47).



id integer	geometry text
0	POINT Z (143555.125652803 485065.690124412 -5.6)
1	POINT Z (143555.125652803 485065.690124412 -2.6)
2	POINT Z (143555.125652803 485065.690124412 0.4)
3	POINT Z (143555.125652803 485065.690124412 3.4)
4	POINT Z (143555.125652803 485065.690124412 6.4)
5	POINT Z (143555.125652803 485065.690124412 9.4)
6	POINT Z (143555.125652803 485065.690124412 12.4)
7	POINT Z (143555.125652803 485065.690124412 15.4)
8	POINT Z (143555.125652803 485065.690124412 18.4)
9	POINT Z (143555.125652803 485065.690124412 21.4)
10	POINT Z (143555.125652803 485065.690124412 24.4)
11	POINT Z (143555.125652803 485065.690124412 27.4)
12	POINT Z (143555.125652803 485065.690124412 30.4)
13	POINT Z (143555.125652803 485065.690124412 33.4)
14	POINT Z (143555.125652803 485065.690124412 36.4)
15	POINT Z (143555.125652803 485065.690124412 39.4)
16	POINT Z (143555.125652803 485065.690124412 42.4)
17	POINT Z (143555.125652803 485065.690124412 45.4)

Figure 46: Point cloud Geometry table, uploaded to a 3DCityDB instance schema



id [PK] integer	point_id integer	variable_name character varying	timestamp_begin timestamp without time zone	timestamp_end timestamp without time zone	step integer	unit character varying	n_val integer	values real[]
0	0	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,-273.15,-273.15,-2
1	1	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.7559,23.4902,23
2	2	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.7671,23.5023,23
3	3	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.778,23.5143,23.
4	4	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.7884,23.5259,23
5	5	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.7984,23.5371,23
6	6	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.808,23.5478,23.
7	7	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.8173,23.5581,23
8	8	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.8264,23.5678,23
9	9	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.8353,23.5772,23
10	10	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.8441,23.5863,23
11	11	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.8529,23.5951,23
12	12	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.8616,23.6037,23
13	13	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.8705,23.6123,23
14	14	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.8793,23.6209,23
15	15	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.8882,23.6294,23
16	16	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.8972,23.6381,23
17	17	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.9063,23.6469,23
18	18	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.9155,23.6558,23
19	19	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.9247,23.6649,23
20	20	AirTemperature	2018-06-24 18:00:01	2018-06-25 10:00:01	1	h	17	{30.4493,27.9492,24.934,23.6742,23.

Figure 47: Point cloud Attribute table, uploaded to a 3DcityDB instance schema

The geometries will be then used to aggregate the points on building level, so that the average hourly air temperature around the building's hull for the simulation period can be finally stored in CityGML. The sequence of queries used to realise the second phase of the methodology can be found at Appendix 2.

5. Proof of concept: Creating CityGML datasets and testing the interface

The case study used in this research is the new Floriade district in the city of Almere, Netherlands. The project, developed by the municipality, is designed around the concept of green village and it will be implemented in two phases. In the first phase, till 2022, the area will host an international agriculture exposition, and after the event it will become a residential district. The boundary dimensions of the district are approximately 300x450 m and a number of green infrastructures will be constructed during the next years. High attention and effort have been dedicated to the design of arboretum, where a number of trees will be planted.

In parallel with the data requirements analysis, a meeting with the municipality of Almere was arranged to check the availability of data and define different simulation scenarios. Since their plan was still at an early stage, a data research had to be conducted in order to determine the case suitability. The outcome was that the final digital model of the urban development had already been designed – so the required infrastructure was already available. Furthermore, a landscape plan was provided with relative georeferenced dwg files.

5.1 Data collection and raw data mapping

Section 4.2.3 (Figure 38) displays the required CityGML classes that needed to be filled with data. Based on this design, a plan of the raw data to be acquired was scheduled. CityGML is a data model which is bounded by strict geometric rules. If a CityGML dataset contain errors, it is most likely that it will cause geometric algorithms to crash when used as input for calculation. This is happening since software assume that their inputs are correct. Such errors can be overlapping polygons, wrong oriented polygons, missing surfaces etc. F. Biljecki et al., (2016) researched the geometric validity on several CityGML datasets and found that geometric errors occur even in the simple LOD1 ones. It becomes evident that geometric validity of the raw datasets used to produce CityGML is required in order to minimize the possibility of occurring error.

This section will present the collected data along with manual pre-processing that was required to reach a desired state. Then the harmonization to CityGML despite being a process that requires precise knowledge, can be executed automatically in FME.

5.1.1 Required data for the Building module

The desired model, at least in terms of building module is of LOD1, which does not include roof, or building overhangs. So, as far as the building module is concerned, the footprints of the buildings along with a corresponding height value is the adequate data. The municipality of Almere provided AutoCAD plans as well as a 3D SketchUp file of the buildings. Unfortunately, the latter was not georeferenced, so the heights of each building were extracted manually and matched to the footprints. The result was a georeferenced building shapefile which contained a Rhino id, and a height attribute. This is the desired input for creating CityGML LOD0 and LOD1 models of buildings. In case SketchUp has to be the design platform, it has to be georeferenced and also each object should be assigned in a different layer. An FME workbench is openly available, as well as a SketchUp plugin for translating 3D models of this type to CityGML models, up to LOD3.

5.1.2 Required data for the Vegetation Module

For the purpose of this thesis, the trees that will be planted on the arboretum were implemented only as solitary vegetation object class while the forest was implemented in the Land Use. A SolitaryVegetationObject may have the attributes class, function, usage, height, truck diameter, crawn diameter. The received tree data consisted of an AutoCAD file of the trees as georeferenced points (Figure 48) with a label pointing to an excel database of the attributes. Furthermore, 3D models of the trees were provided, which were however too complex to be used as the implicit geometry. The database required a processing, in order to translate the name columns from Dutch to English and create a proper csv structure with only useful data (i.e. species, height, crawn diameter, truck diameter) and the different tree geometries that would be used for the different LODs had to be constructed from the beginning.

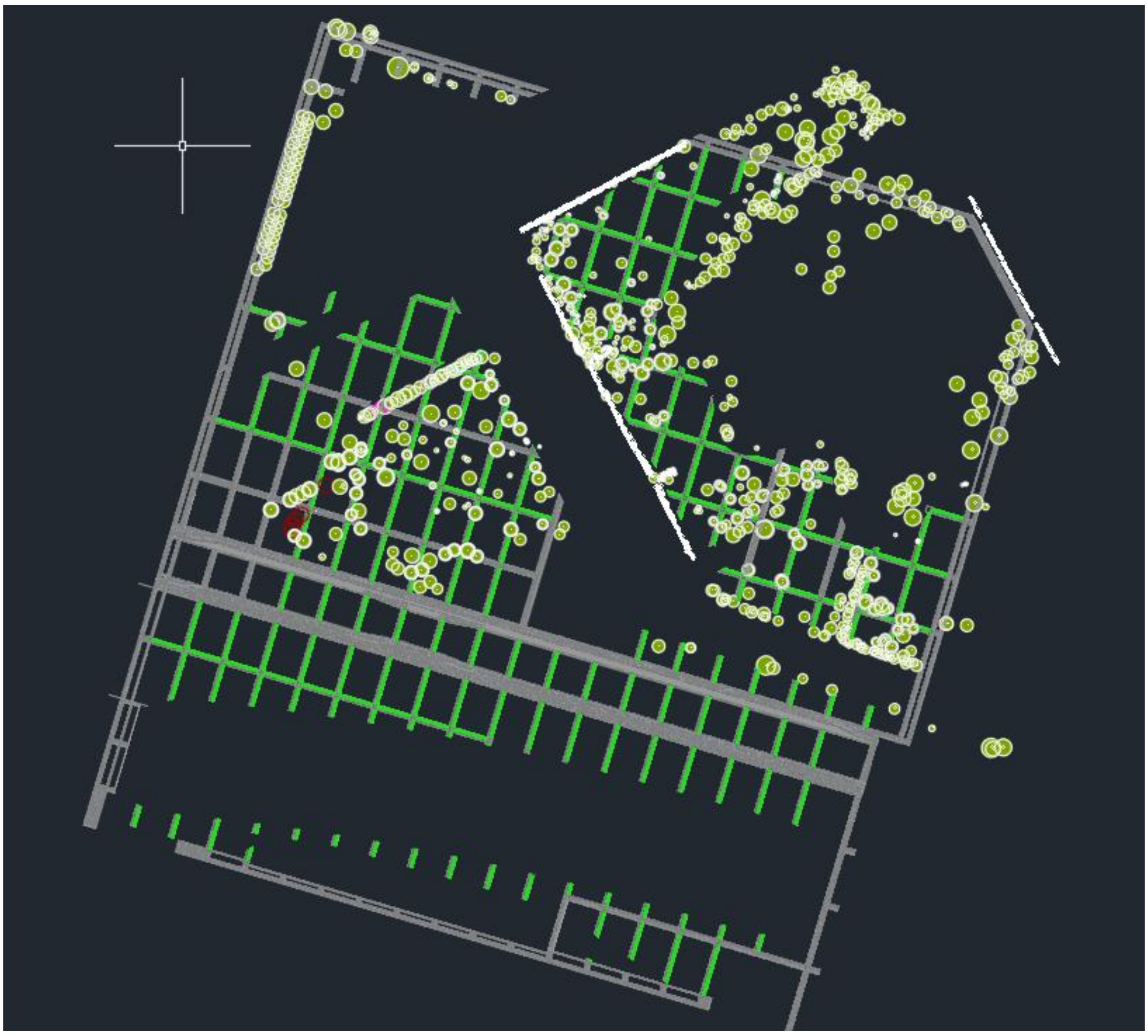


Figure 48: AutoCAD plan of the georeferenced trees on the arboretums

5.1.3 Required data for Relief Module

Concerning the Relief module, the desired implementation was a TIN. A georeferenced DEM raster of the whole extent of the area was provided by Almere, which was adequate and did not require any pre-processing.

5.1.4 Required data for Land Use Module.

Finally, as mentioned in the third chapter, the Land Use module was chosen to be filled in order to provide the required data to the ENVI-met soils tag. The data describing the Land cover consisted of AutoCAD plans were each structured group was designed in a different layer. Out of these plans different CityGML modules could be generated (*WaterBodies, Transportation, LandUse, Vegetation)* however the *LandUse* could potentially satisfy the whole extent of the ENVI-met soils tag. Furthermore, its design is relatively simple, comparing to the remaining CityGML modules. The problem with the raw data in this case was that AutoCAD does not provide topological validity without further topology check. As a result, two major problems emerged. First, primitive polygons contained topological errors such as spikes and holes (donuts), and secondly a lot of polygons did overlap, not only between different layers but also within the same layer. While trying to dissolve and intersect the existing polygons to create one Shapefile with a defined layer hierarchy, the errors were multiplied. As a result, a lot of manual editing was conducted in FME, only to remove the spikes and the self-intersections. Furthermore, the number of depicted layers was vast, and the required fields (roads, paths, forest, water, arboretums, settlement areas) had to be grouped in layers manually. After all the pre-processing a clean shapefile of the different pre-defined layers was created (Figure 49) which would be the ideal input.

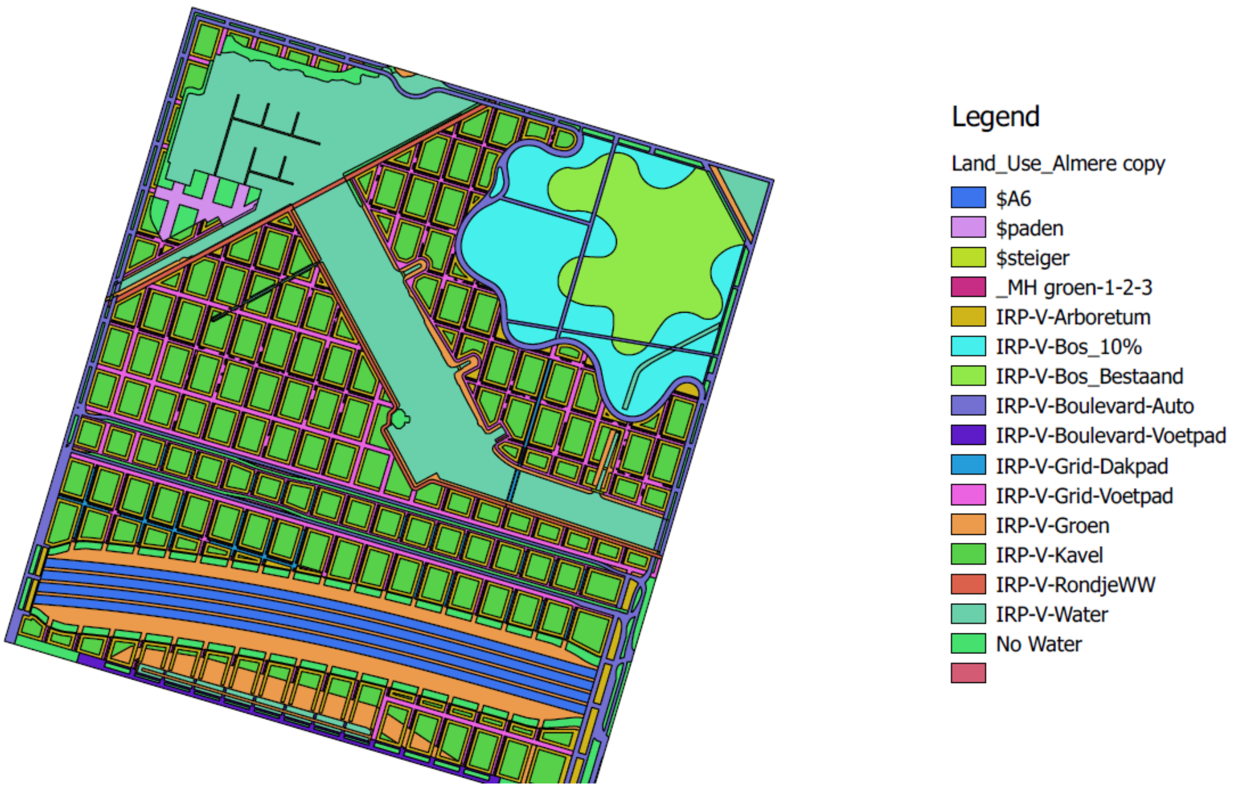


Figure 49: Land Cover shapefile, based on filtered AutoCAD layers

Consequently, four different attributes were created in QGIS, describing the class, usage &function and the *GenericsAttribute (material)* that was later created (Figure 50), based on the layer description.

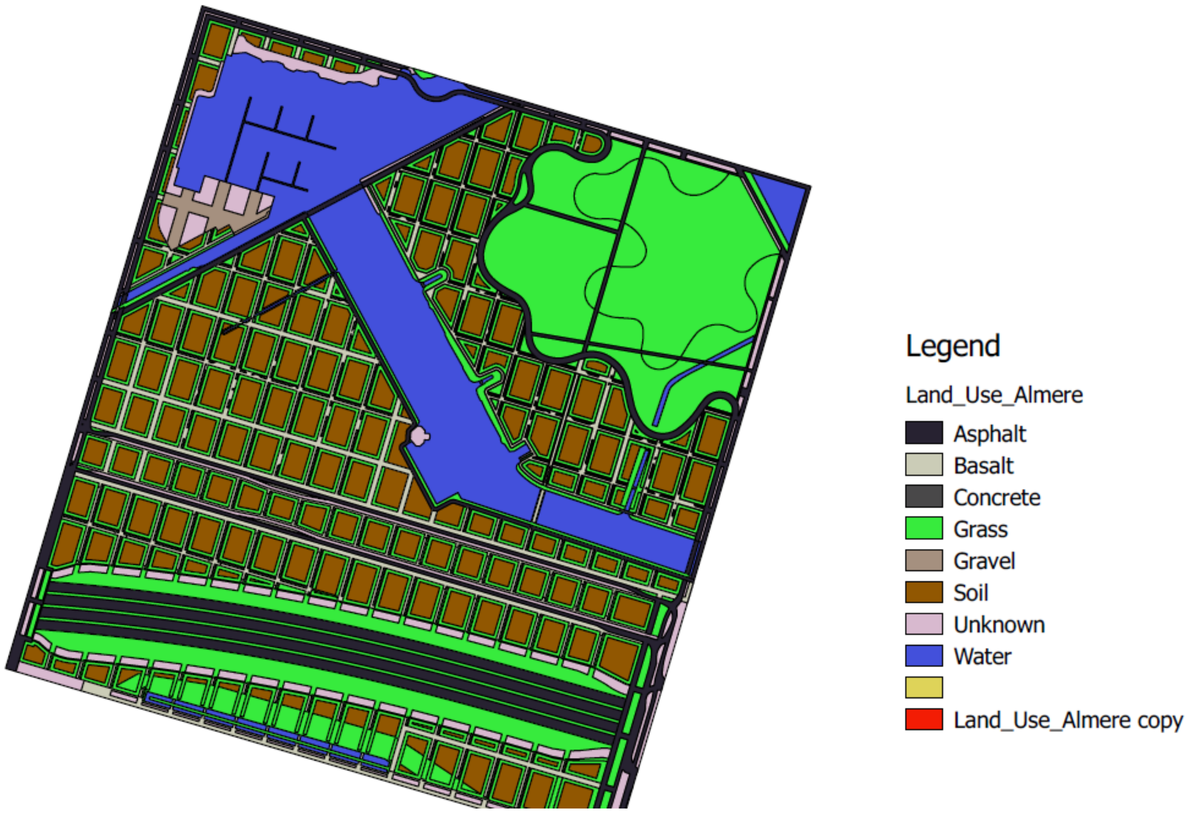


Figure 50: Materials GenericAttribute, Legend displays the codelist

5.2 Conversion to CityGML

For the generation of a CityGML dataset Safe Software’s FME was used. FME is using transformers to change the shape of data and generate additional attributes. It also provides CityGML writers which convert automatically to a CityGML format. Furthermore, FME was used for creating CityGML datasets within the framework of Geomatics courses and is also a suitable tool for data cleaning which was required for the Land Use shapefile either way. For these reasons, FME was preferred for the whole data processing. Harmonizing data into CityGML, from shapefile is a complicated process that require precise knowledge over the semantic codelists and the geometry types used for each class. Hence, referring to the UML diagrams is necessary. However, for LOD1-LOD2 the process is automated, as long as the shapefiles are clean.

Building Module

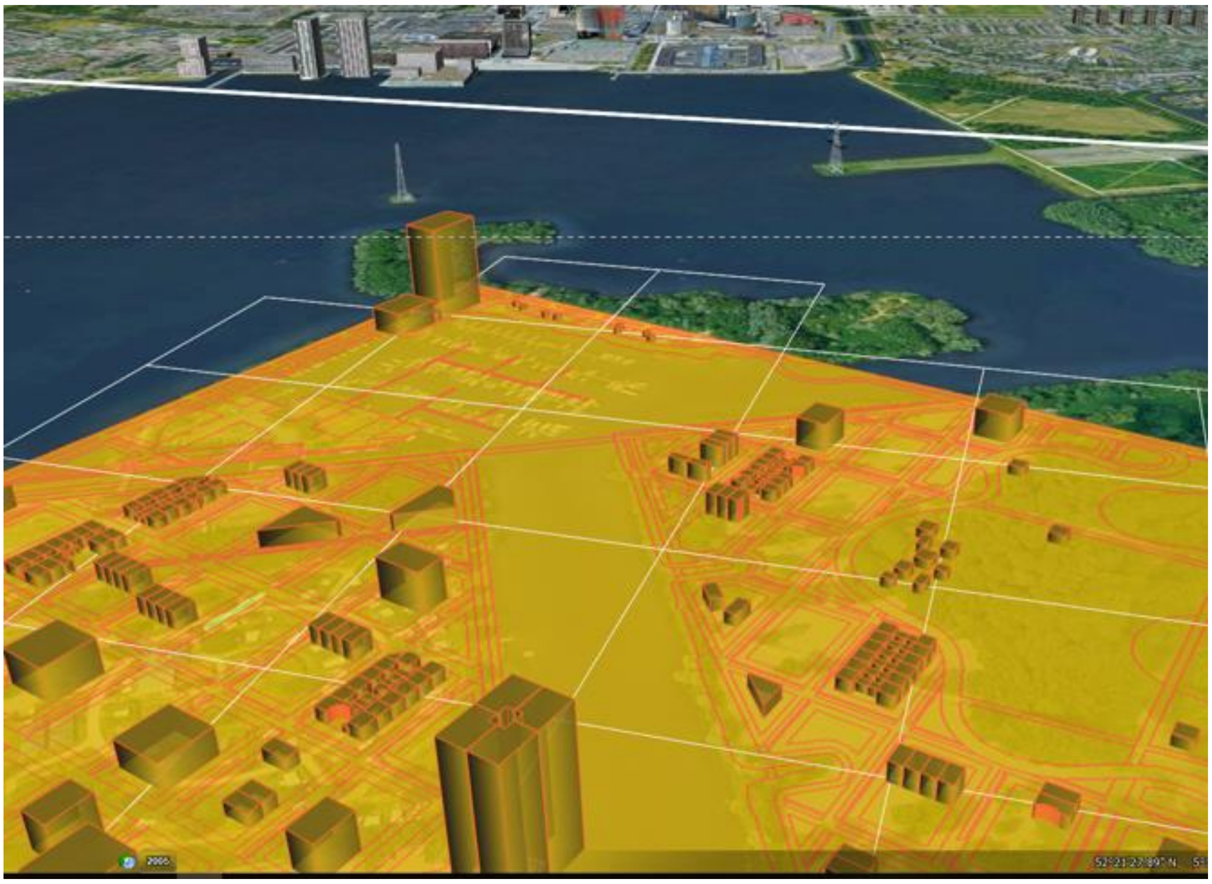


Figure 51: CityGML LOD1 Building model of the Floriade district, exported in KML by 3DcityDB Importer-Exporter and visualized in Google Earth

Starting from the most prominent thematic module of CityGML, the building harmonization is partially automatic for LOD2-3 and fully automatic for LOD1. The pipeline for transforming a 2D building Shapefile to an LOD0 and LOD1 – dataset is described in this section (Figure 51). First an assignment of a height value to the 2D footprint surface is needed. This can be carried over by providing a base height, if it exists as an attribute, otherwise by computing the average height of the DEM with zonal statistics and offset the footprint. Afterwards an extrusion by the building height attribute happened. In the current thesis, every building was modelled as one *AbstractBuilding* object, without including any building parts.

When generating CityGML data with FME, attributes and geometry must be created according to the CityGML rules. The geometry representation of an *AbstractBuilding for LOD2 is shown on Figure 52.*

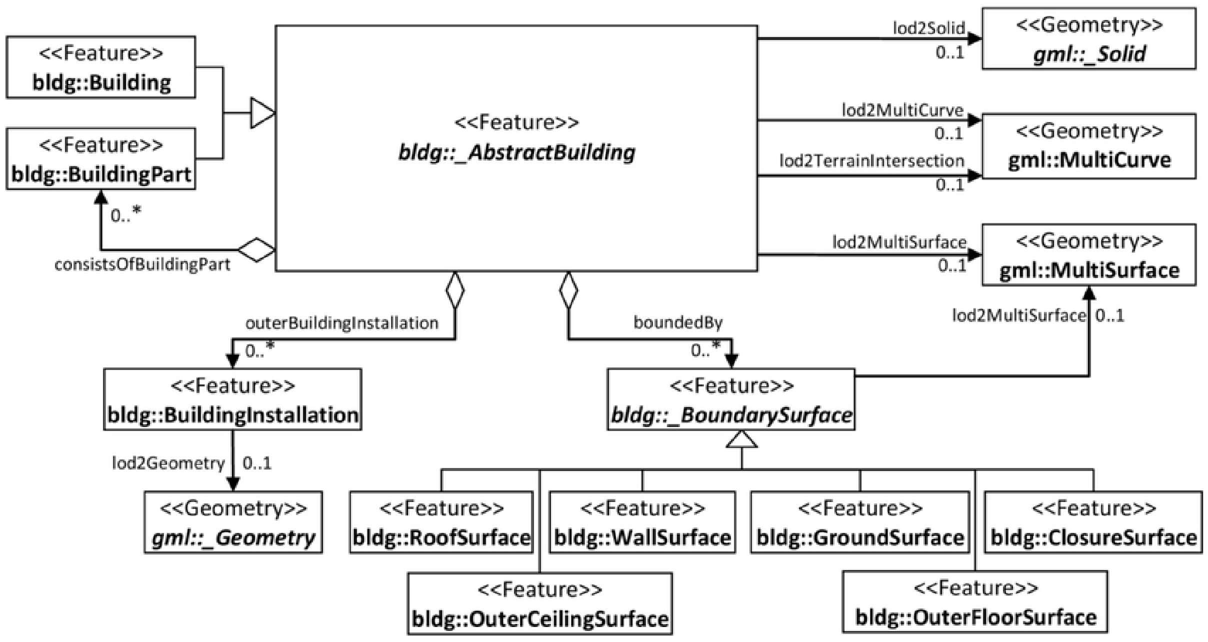


Figure 52: UML diagram of LOD2 building, Source : https://researchgate.net/figure/, 2019

In LOD0 the buildings were represented by a 3D horizontal surface, featuring the footprint. For LOD1 the volumetric part of the exterior building cell was used. The geometry type is represented as gml::Solid type. Finally, an LOD2 model of buildings was created in order to allow temperature aggregations at facade level. The building entities for this level of detail were represented as *_MultiSurface* type facades. Finally, the attributes that were filled with data are the *measuredHeight and Storeys_aboveGround* which occurred after a floor division with the default store height value of 3m. A common error when creating a CityGML dataset is the orientation of the multisurfaces which leads, not only to rendering errors, but more important to unsuitability of the model in calculations. Thus, a prior, to geometry setting, orientation of every face (Figure 53) is highly important to avoid such problems.

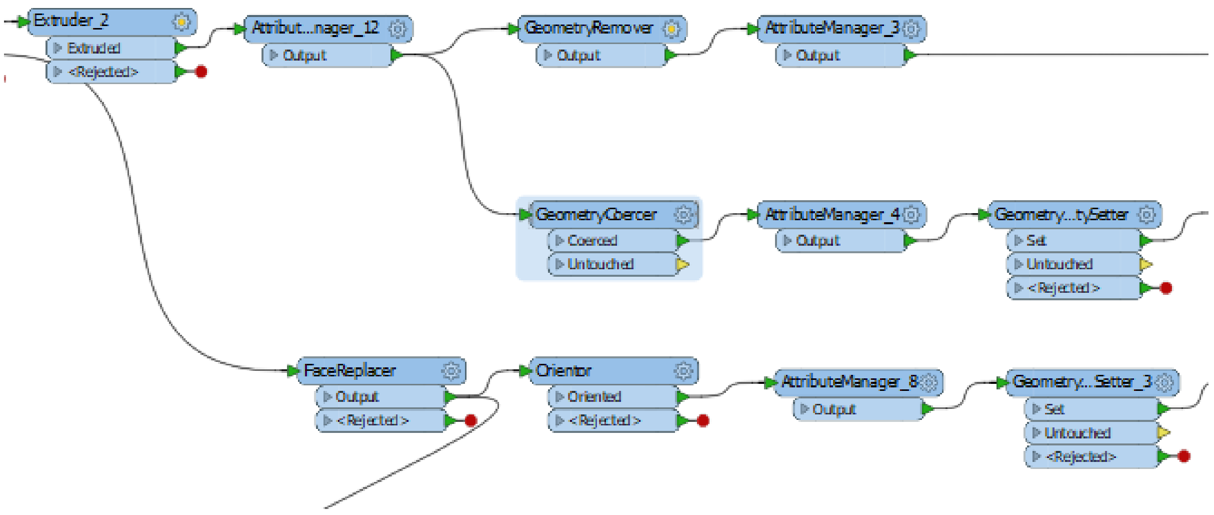


Figure 53: Part of FME workbench which was created for the harmonization of building shapefile to CityGML, face replacer replaces 2D geometry with 3D and orientor verifies the correct orientation of surfaces.

<u>Relief Module</u>

The next module that was modelled was the Relief module, which holds the topographic information. The DEM in CityGML is represented by the relief Feature which consist of 1 or more ReliefComponents. As it was mentioned in a previous chapter, the relief module can be specified by one of the following types. Triangulated irregular network, Raster, Breaklines or Masspoint (Figure 54). In the Floriade model, the terrain was modelled as a TIN without applying Terrain Intersection Curves.

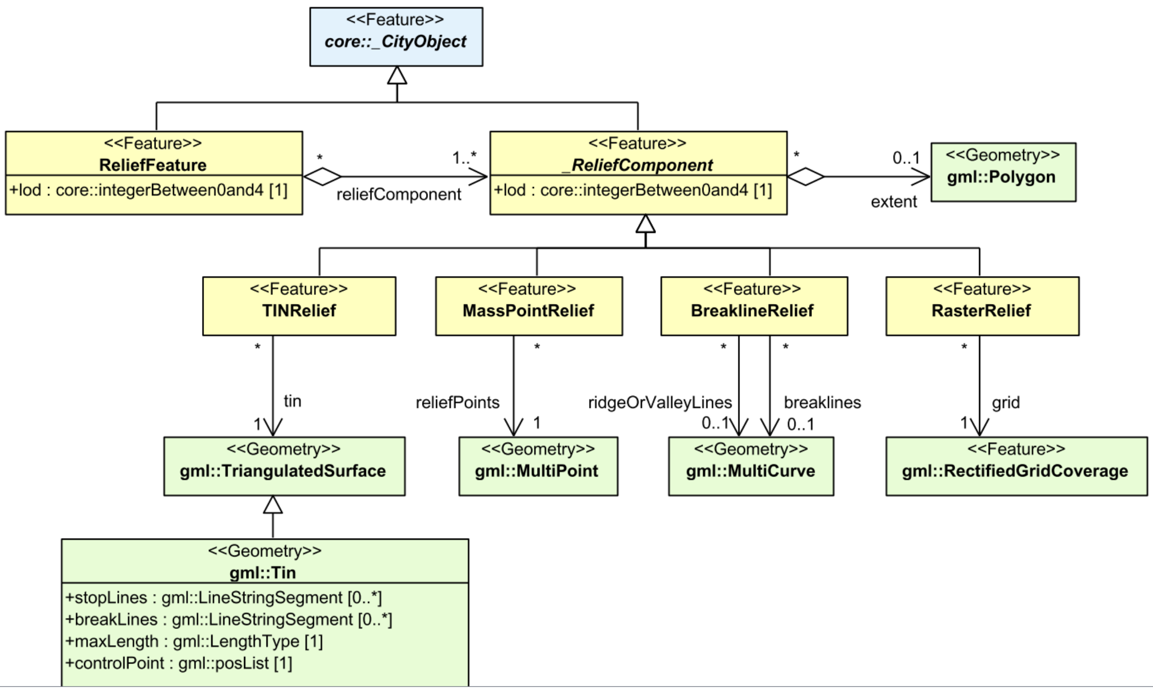


Figure 54: Relief component UML diagram

The detail of the DEM can be flexible and is not blended to strict rules. Relief feature and the relief components must have an LOD attribute which can differ, depending on the needs. In the Floriade model the LOD was set to LOD2 and counts approximately 123.000 triangles for an extent of 2000x1500 m (Figure 55).

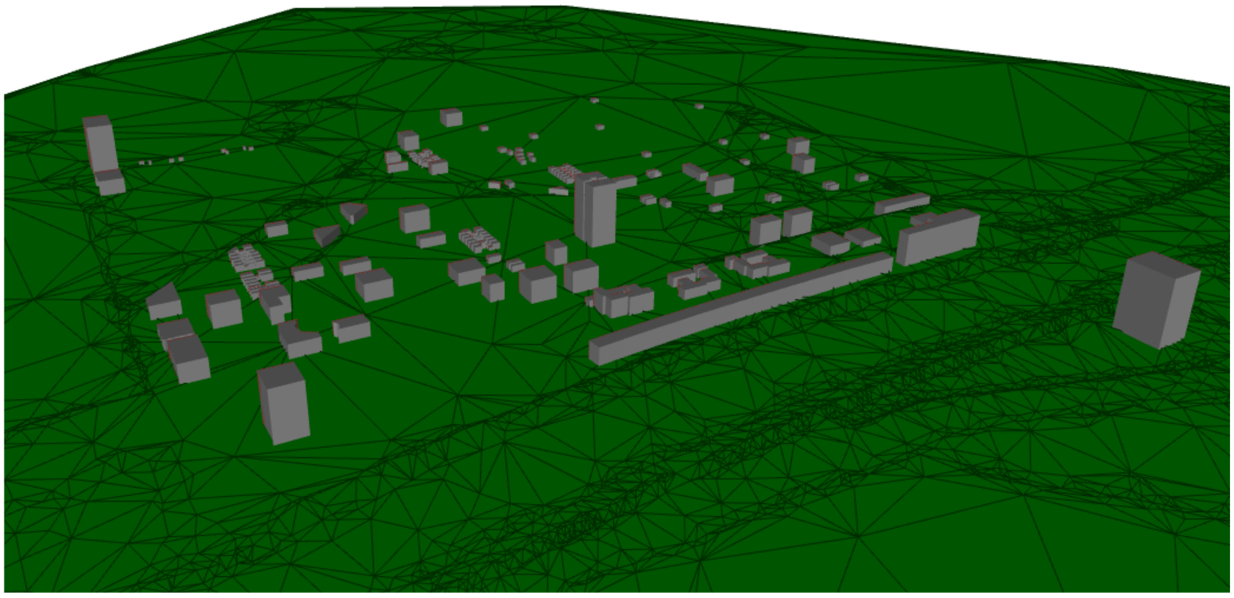


Figure 55: TIN Relief of Floriade CityGML model, visualisation in FME

<u>Land Use Module</u>

The Land Use module was harmonized by the clean Land Cover shapefile. LandUse can be modelled in different LODs, like the relief module explained on the previous subsection, without strict rules. In our model the land Use was implemented as LOD0. The polygons were selected to be flat, instead of following the terrain (Figure 56). First the z coordinate was added, then the geometry was replaced with gml::multisurface and then the attributes were merged. First a csv file was created which expanded the AutoCAD layer name, with the additional attribute columns. Subsequently the attributes were aggregated on each polygon on the layer name field. Attributes that were filled with data are the class, function, usage and the generic attribute material that was discussed in the section 3.4.1.

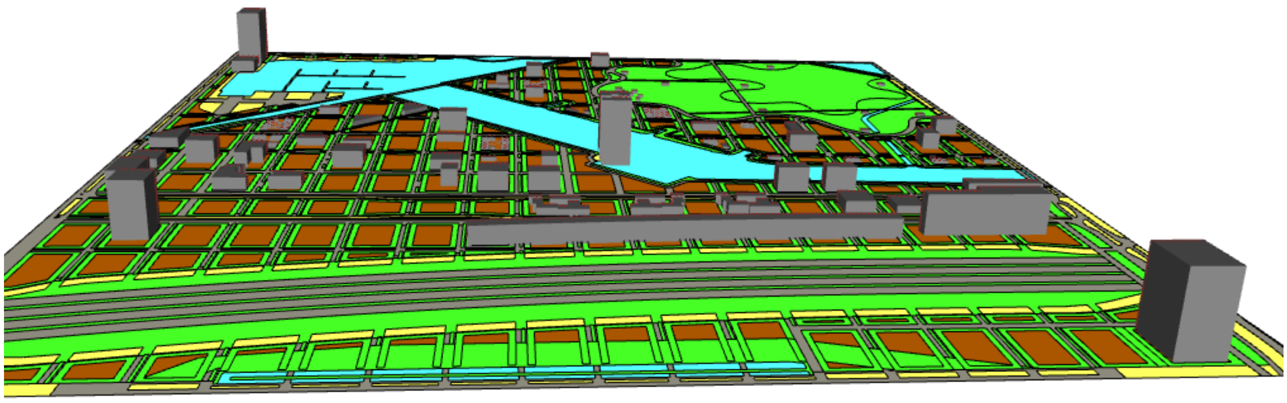


Figure 56: LandUse module of Floriade CityGML model, visualization in FME

<u>Vegetation Module</u>

The last module to be filled with data was the Vegetation. Vegetation module can be displayed as solitary vegetation object, or Plant Cover. In this thesis, the Solitary Vegetation Object was selected for modelling trees in the arboretum, by using implicit geometries. The forest was converted to *LandUse* (Usage = Forest) surfaces. In order to create solitary vegetation objects with implicit geometry concept, a geometric object should be stored once in an absolute coordinate system. Afterwards each occurrence is described by an anchor point, which sets the pivot point of the geometric object, and a transformation matrix, which defines the rotations on the 3 axes and the scale. In the Floriade model, one geometry for each LOD, from LOD1 to LOD3 was used, and the transformation matrix was related to the height attribute and the crown diameter of the tree.

For the harmonization of tree data to CityGML a number of FME transformers were set. The inputs for this workbench were the AutoCAD plan with the points of the trees and their labels, the csv database of the tree species and attributes, and the TIN relief that was constructed earlier. Starting with the AutoCAD dataset, the label was matched to its point with the Labeler transformer, and then the point was immediately draped to the DEM. Afterwards the required attributes were created and filled with data from the csv file, joined on the Label number. In the end the transformation matrix was created, with a scale value related to the tree height and crown diameter (Figure 57). Finally, the attributes that are filled with data are the *species, Height, Truck Diameter, crown diameter.*

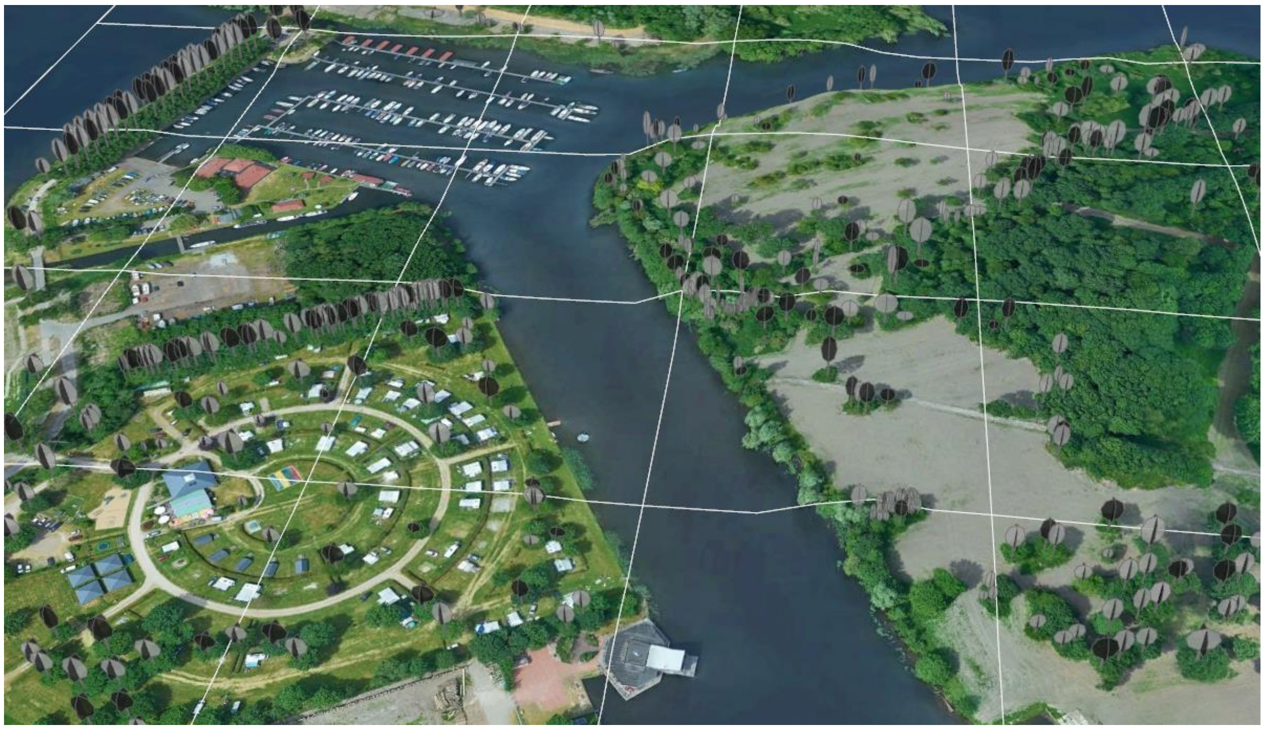


Figure 57: Vegetation module of Floriade CityGML model, exported as KML and visualized in Google Earth

5.3 Scenario definition

The thesis main topic was to couple the two data models, the one of CityGML and ENVI-met. Further geodata was needed in order to test the methodology. Different scenarios were simulated in order to provide an analysis of the microclimate performance of the new district. Scenario methods are commonly used in urban planning practice in order to compare different design solutions and evaluate the impact of design variables. For example, scenarios can be used to simulate the impact of different species of vegetation or to investigate paving materials on local air temperatures and human comfort. The simulations can be performed at different periods of the year, by using sensor data from weather stations. After the meeting with people from the municipality of Almere, we agreed in simulating the effect of trees on the local climatic environment. The Floriade project focuses on the concept of green village, and the number of different tree species that will be planted exceeds 1000. Of course, a manual generalization was required, since ENVI-met build in libraries do not support such a big variety of tree species. Hence, the final simulation would include two main scenarios, i) simulation of the whole area of interest, including all trees (forest and arboretum trees) in the warmest day of the last 3 years and ii) simulation of the same area in the same day but without the trees on the arboretums.

5.4 Methodology application, simulations and results

Currently SclGModeLer is designed to map data according to the rules displayed in section 4.2.2 (Figure38). The designed methodology however shall be applicable for other suitable CityGML classes and attributes such as the *Transportation* class materials, water surfaces, or Energy ADE *Material and* Construction class *Material*, thanks to the layer approach previously described. As mentioned in chapter 4.2.2, SclGModeLer is resilient over different CityGML datasets, and this is the advantage of working with a semantic standard model as an input. However, to cover the full potential of the currently implemented methodology, the aforementioned classes and attributes, including the created *GenericAttribute* must be filled with data.

5.4.1 Area input files

The methodology was tested with different parameters over the CityGML dataset of the Floriade district and responded to every different situation with success (figures). Furthermore, it succeeded in creating INX models from different CityGML datasets, (figure 35, section 4.2.1).

Several INX files of the Floriade district were generated with SclGModeLer, which differ by extent, resolution, rotation and tree types. These INX files are the first successful result of the methodology. They can be modelled only by changing the parameters on SclGModeLer interface (Figure 58, Figure 59).

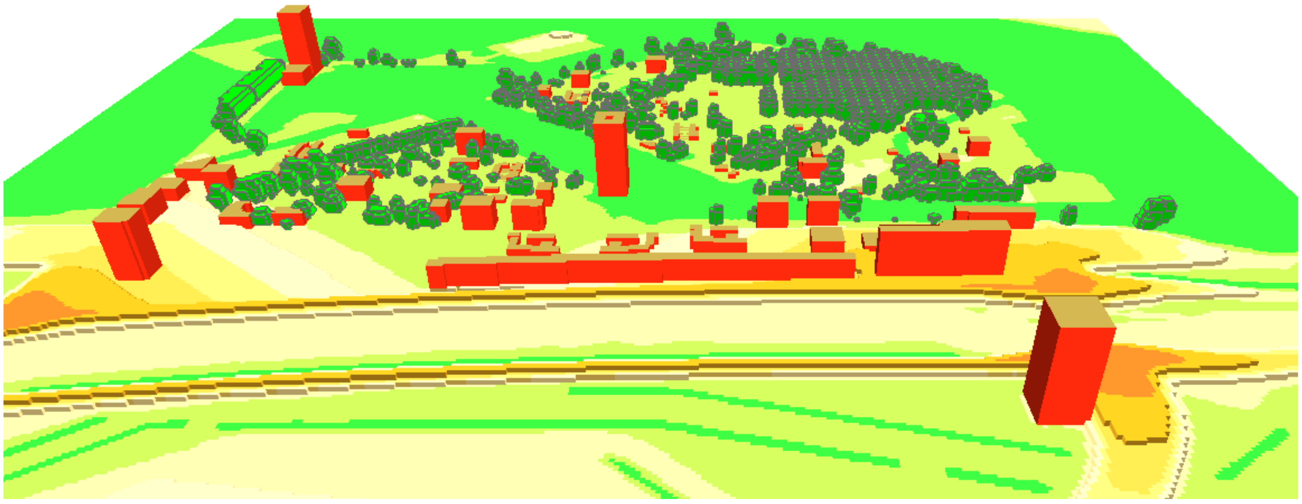


Figure 58: Area input file of Scenario with arboretum trees, displayed in SPACES 3D viewer, focus on DEM. The generation time in personal computer was 3 minutes, 15 seconds while the extraction of the average air temperature at building level took around 1 hour and 20 minutes.

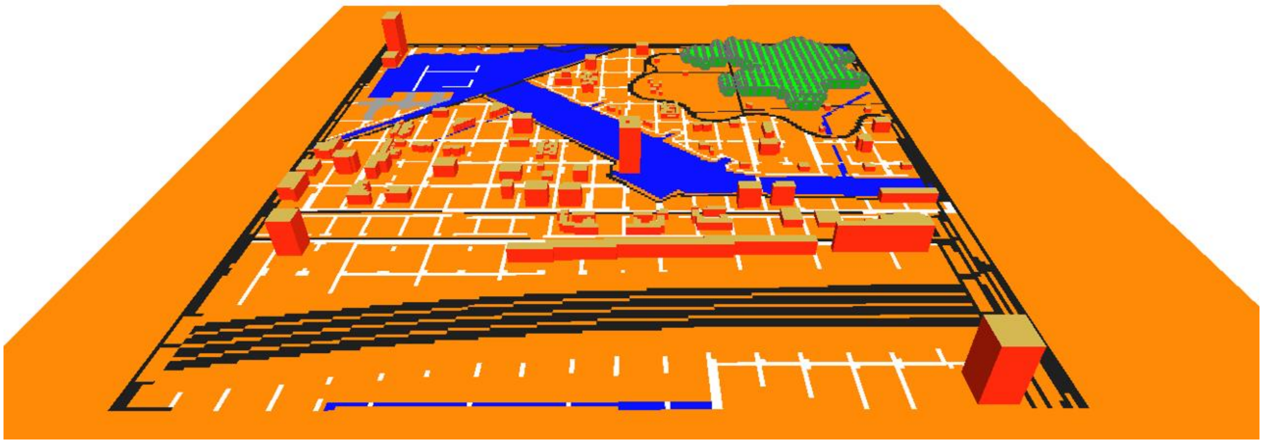
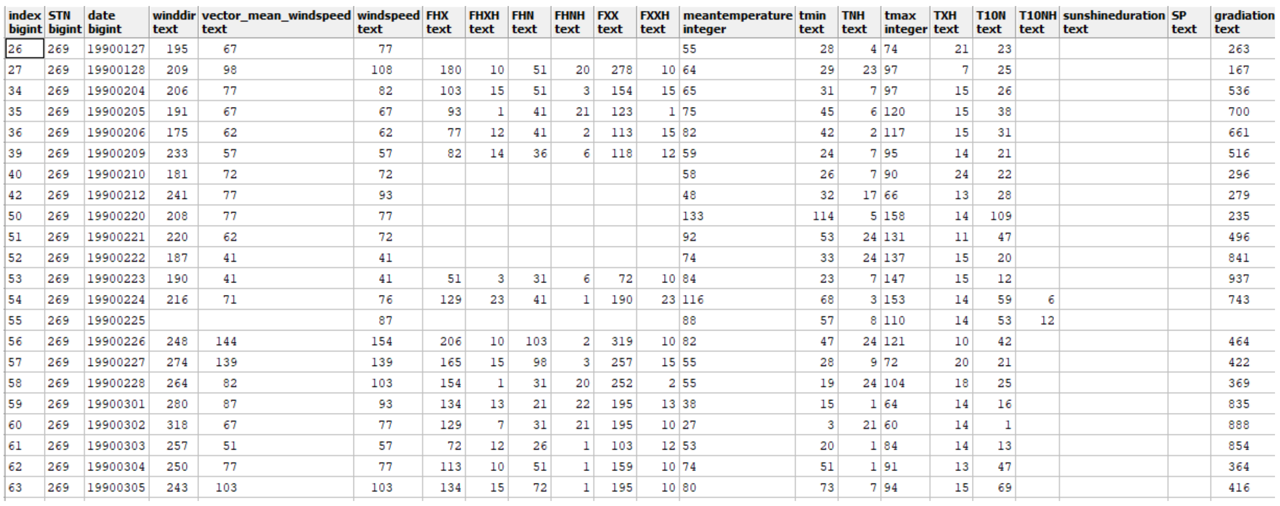


Figure 59: Area input file of scenario without arboretum trees, displayed in SPACES 3D viewer, focus on soils. The generation and temperature extraction times are similar with Figure 58.

While developing the code and creating sample INX files, two visual glitches were discovered in the SPACES interface, one of which was already mentioned in ENVI-met forum, but the other was most probably unknown. The first is that the first-by-left line of the grid was missing the DEM information, and was completely empty. This was however solved by moving the matrix position of the <dem2D> tag inside the INX file by one space. The latter glitch results in the piercing of the DEM by objects, in columns where 1 cell was occupied by the DEM. This glitch however is just a visual one and is not transferred in the outputs.

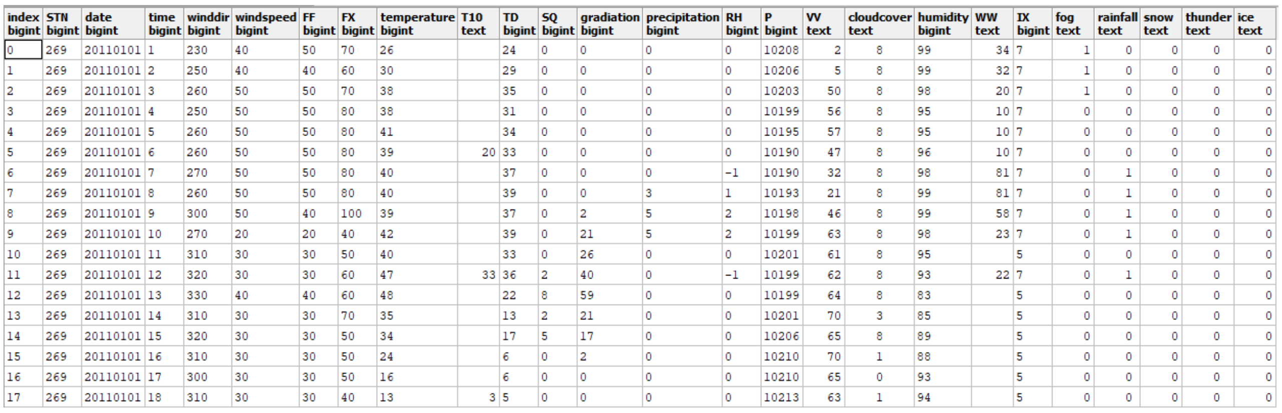
5.4.2 Weather inputs

SclGModeLer output of the first phase is the INX area input file and not the SIM simulation file that includes all the required parameters, along with the weather values, needed to start the simulations. However, this is not considered a major drawback. The creation of the simulation file requires only the definition of an area input file and the required weather values and is carried out automatically by an ENVI-met editor. Selecting the weather parameters from the interface, as user inputs, in order to create simulation files (with simple forcing) directly would be another solution; therefore, would not provide any additional automatization which was the major goal. As discussed on the previous chapters, ENVI-met included more than one way for providing the weather data, and the most detailed method, the full forcing method was used in the following results. The input for creating a Full Forcing (.fox) file is a csv file that depicts half-hourly values for air temperature, relative humidity, wind speed and wind direction. This file can also contain values for direct, diffuse and longwave direction, in relation with cloud cover, but these parameters were not included in the pre-defined scenarios. The fox file is an XML encoded encrypted file (for a 31-hour span of the aforementioned parameters, the result is a fox file of around 1 million lines). A script was created to transform the csv file of the data, derived from the meteorological station to two PostgreSQL tables (Figure 60, Figure 61), that included, translated from Dutch, column names with numeric values on the filtered parameters, so that desired days can be queried.



index bigint	STN bigint	date bigint	winddir text	vector_mean_windspeed text	windspeed text	FHX text	FHXH text	FHN text	FHNH text	FXX text	FXXH text	meantemperature integer	tmin text	TNH text	tmax integer	TXH text	T10N text	T10NH text	sunshineduration text	SP text	gradiation text
26	269	19900127	195	67	77							55	28	4	74	21	23				263
27	269	19900128	209	98	108	180	10	51	20	278	10	64	29	23	97	7	25				167
34	269	19900204	206	77	82	103	15	51	3	154	15	65	31	7	97	15	26				536
35	269	19900205	191	67	67	93	1	41	21	123	1	75	45	6	120	15	38				700
36	269	19900206	175	62	62	77	12	41	2	113	15	82	42	2	117	15	31				661
39	269	19900209	233	57	57	82	14	36	6	118	12	59	24	7	95	14	21				516
40	269	19900210	181	72	72							58	26	7	90	24	22				296
42	269	19900212	241	77	93							48	32	17	66	13	28				279
50	269	19900220	208	77	77							133	114	5	158	14	109				235
51	269	19900221	220	62	72							92	53	24	131	11	47				496
52	269	19900222	187	41	41							74	33	24	137	15	20				841
53	269	19900223	190	41	41	51	3	31	6	72	10	84	23	7	147	15	12				937
54	269	19900224	216	71	76	129	23	41	1	190	23	116	68	3	153	14	59	6			743
55	269	19900225			87							88	57	8	110	14	53	12			
56	269	19900226	248	144	154	206	10	103	2	319	10	82	47	24	121	10	42				464
57	269	19900227	274	139	139	165	15	98	3	257	15	55	28	9	72	20	21				422
58	269	19900228	264	82	103	154	1	31	20	252	2	55	19	24	104	18	25				369
59	269	19900301	280	87	93	134	13	21	22	195	13	38	15	1	64	14	16				835
60	269	19900302	318	67	77	129	7	31	21	195	10	27	3	21	60	14	1				888
61	269	19900303	257	51	57	72	12	26	1	103	12	53	20	1	84	14	13				854
62	269	19900304	250	77	77	113	10	51	1	159	10	74	51	1	91	13	47				364
63	269	19900305	243	103	103	134	15	72	1	195	10	80	73	7	94	15	69				416

Figure 60: Daily meteorological data, derived from Lelystand weather station



index bigint	STN bigint	date bigint	time bigint	winddir bigint	windspeed bigint	FF bigint	FX bigint	temperature bigint	T10 text	TD bigint	SQ bigint	gradiation bigint	precipitation bigint	RH bigint	P bigint	VV text	cloudcover text	humidity bigint	WW text	IX bigint	fog text	rainfall text	snow text	thunder text	ice text
0	269	20110101	1	230	40	50	70	26		24	0	0	0	0	10208	2	8	99	34	7	1	0	0	0	0
1	269	20110101	2	250	40	40	60	30		29	0	0	0	0	10206	5	8	99	32	7	1	0	0	0	0
2	269	20110101	3	260	50	50	70	38		35	0	0	0	0	10203	50	8	98	20	7	1	0	0	0	0
3	269	20110101	4	250	50	50	80	38		31	0	0	0	0	10199	56	8	95	10	7	0	0	0	0	0
4	269	20110101	5	260	50	50	80	41		34	0	0	0	0	10195	57	8	95	10	7	0	0	0	0	0
5	269	20110101	6	260	50	50	80	39	20	33	0	0	0	0	10190	47	8	96	10	7	0	0	0	0	0
6	269	20110101	7	270	50	50	80	40		37	0	0	0	-1	10190	32	8	98	81	7	0	1	0	0	0
7	269	20110101	8	260	50	50	80	40		39	0	0	3	1	10193	21	8	99	81	7	0	1	0	0	0
8	269	20110101	9	300	50	40	100	39		37	0	2	5	2	10198	46	8	99	58	7	0	1	0	0	0
9	269	20110101	10	270	20	20	40	42		39	0	21	5	2	10199	63	8	98	23	7	0	1	0	0	0
10	269	20110101	11	310	30	30	50	40		33	0	26	0	0	10201	61	8	95		5	0	0	0	0	0
11	269	20110101	12	320	30	30	60	47	33	36	2	40	0	-1	10199	62	8	93	22	7	0	1	0	0	0
12	269	20110101	13	330	40	40	60	48		22	8	59	0	0	10199	64	8	83		5	0	0	0	0	0
13	269	20110101	14	310	30	30	70	35		13	2	21	0	0	10201	70	3	85		5	0	0	0	0	0
14	269	20110101	15	320	30	30	50	34		17	5	17	0	0	10206	65	8	89		5	0	0	0	0	0
15	269	20110101	16	310	30	30	50	24		6	0	2	0	0	10210	70	1	88		5	0	0	0	0	0
16	269	20110101	17	300	30	30	50	16		6	0	0	0	0	10210	65	0	93		5	0	0	0	0	0
17	269	20110101	18	310	30	30	40	13	3	5	0	0	0	0	10213	63	1	94		5	0	0	0	0	0

Figure 61: Hourly meteorological data, derived from Lelystand weather station

5.4.3 Running simulations with ENVI-met science version

In total, two sets of simulations were conducted with ENVI-met science-version in two powerful computers (16 processors of 3.4 GHz of speed). For both sets of simulation the simulation span was 31 hours; starting from 17.00 and including the whole day of interest (when running simulations with ENVI-met it is strongly recommended to start the simulation some hours prior to the desired starting time. In this way the model can be calibrated, in order to provide accurate results). The first was a set of 3 test simulations of the same area (southwest district). The resolution used in these simulations was 4*4*3 and the extent of the grid was 122*127*31. The required time for these simulations was 110 hours in total (they ran in single core, simultaneously in 1 computer). In these simulations, trees were modelled according to the three options provided with SclGModeLer (displayed in section 4.2.3). The goal of this set was to examine the influence of the different tree types in ENVI-met outputs as well as the simulation times, and as far as the weather data is concerned, a summer day with average weather values was preferred. The second set included the two different scenarios, described in section 5.3. The forest was modelled as a formation of 3D trees (this is the way that SclGModeLer currently handles the forest) and for the arboretum trees, 3D trees were also preferred, since they are a more advanced method of modelling single trees, and the simulation times didn't differ. The results of the former set of simulations are illustrated in figures (62,63). Furthermore 3 buildings were selected for comparing the three different tree representation results (Figure 64, Figure 65).

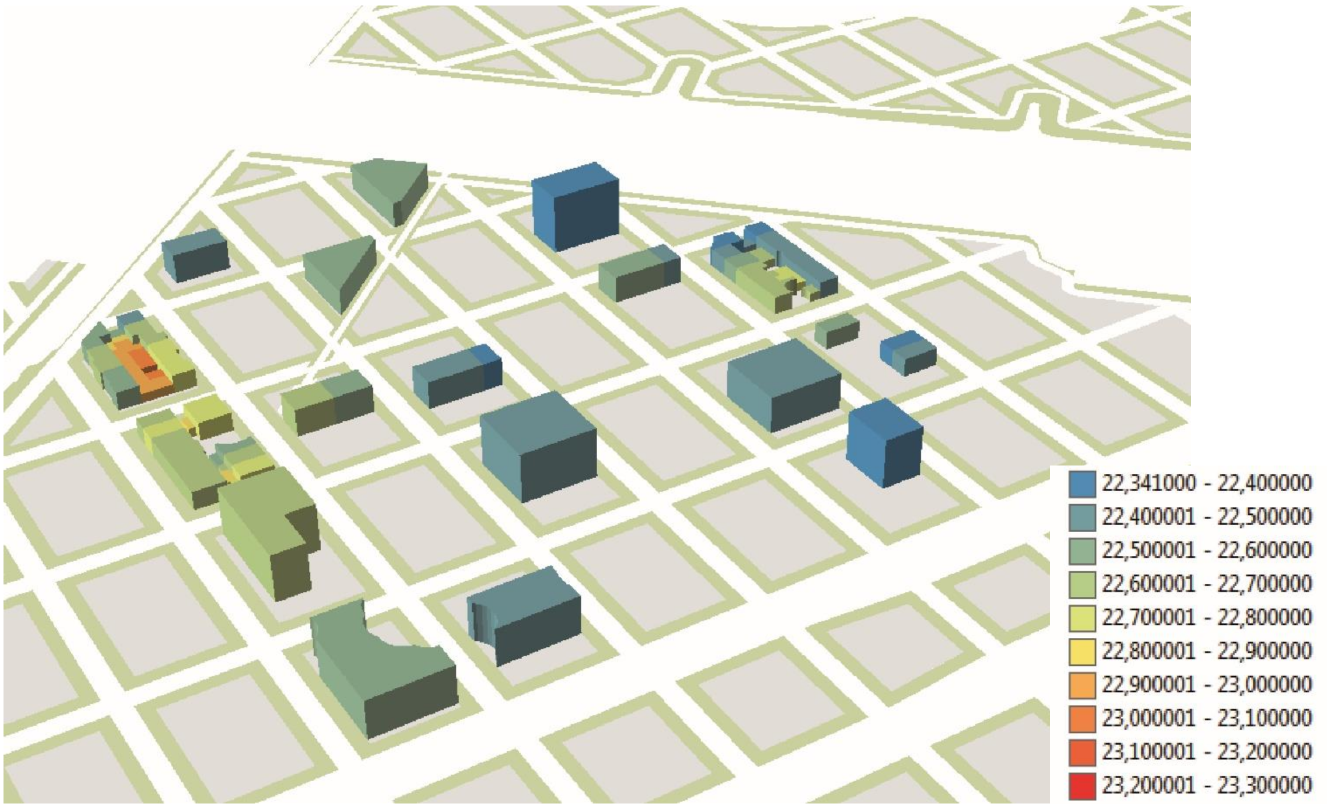


Figure 62: District 3D trees 16:00

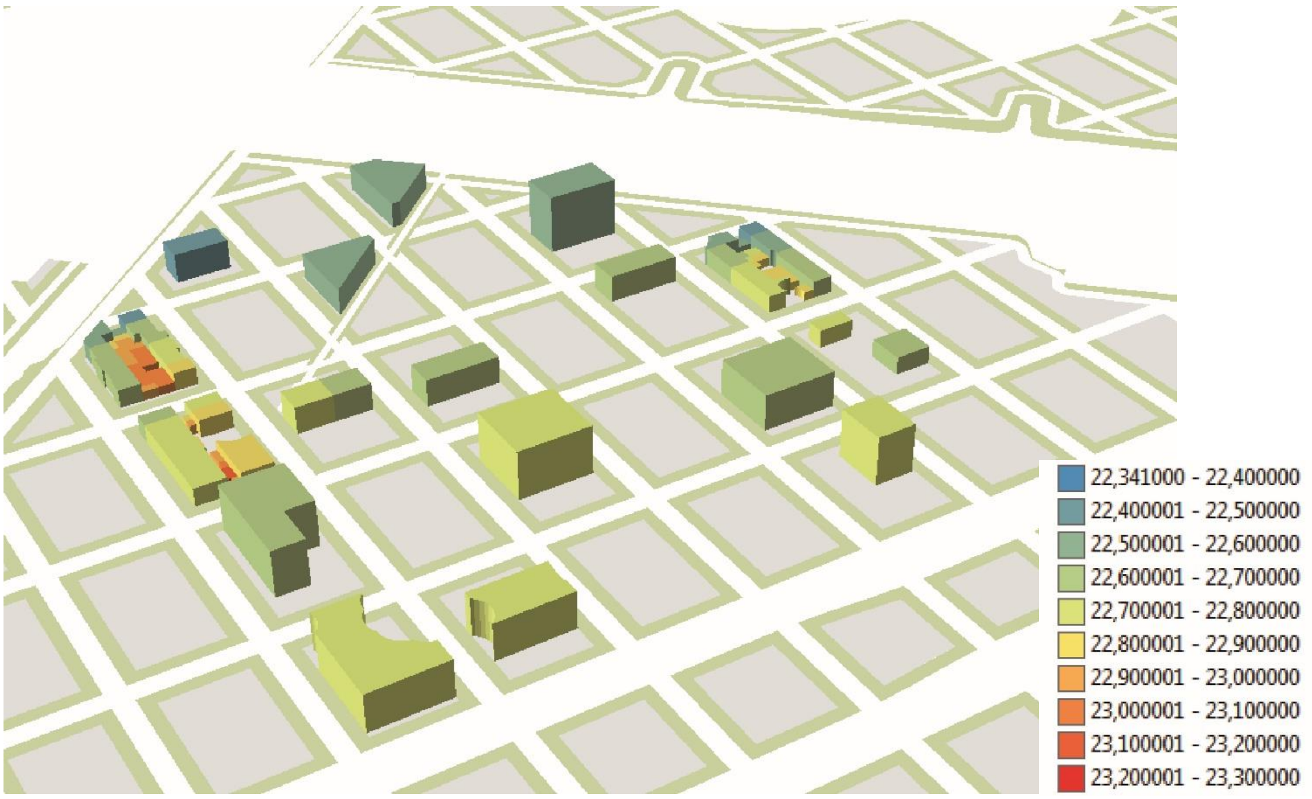


Figure 63: District Simple Trees 16:00

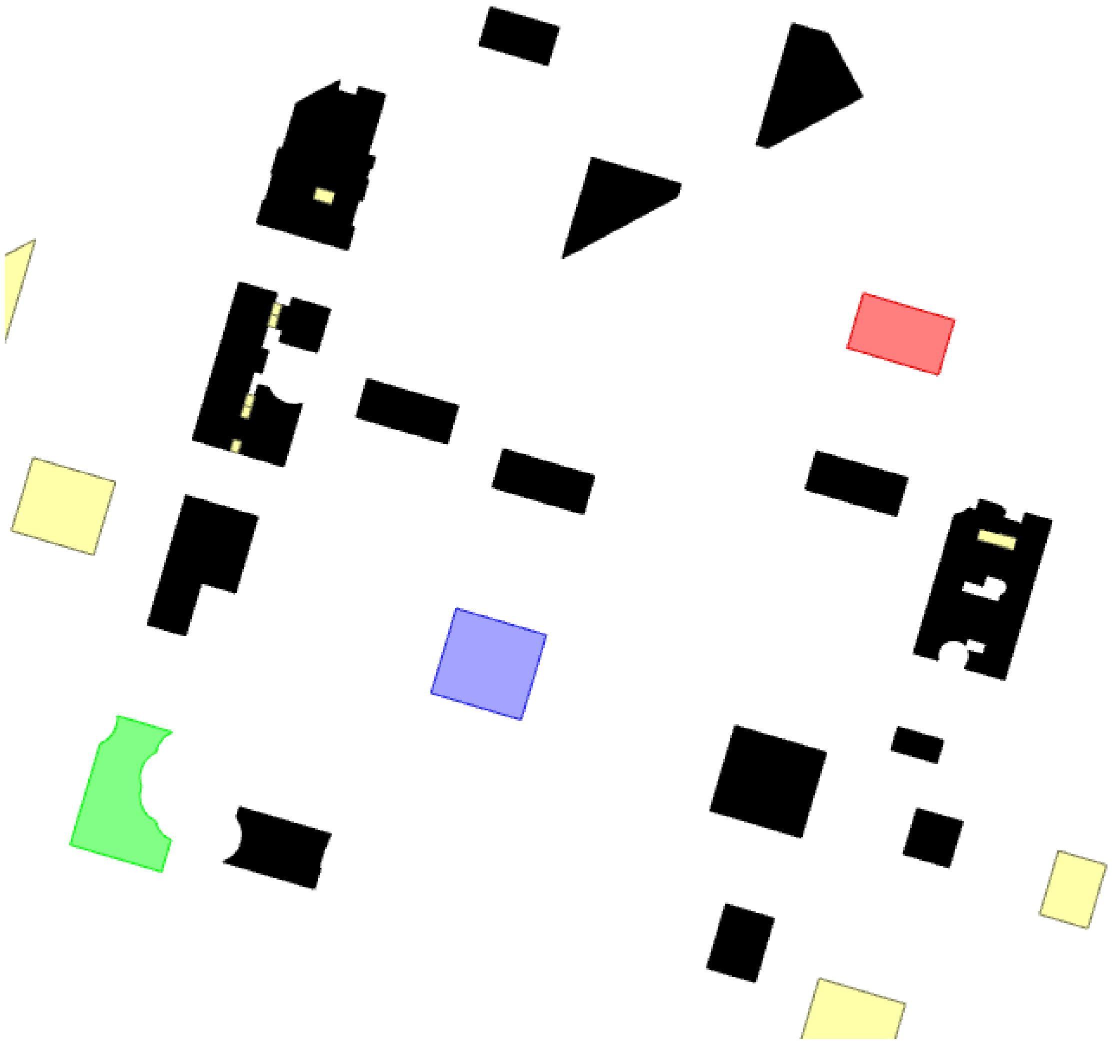
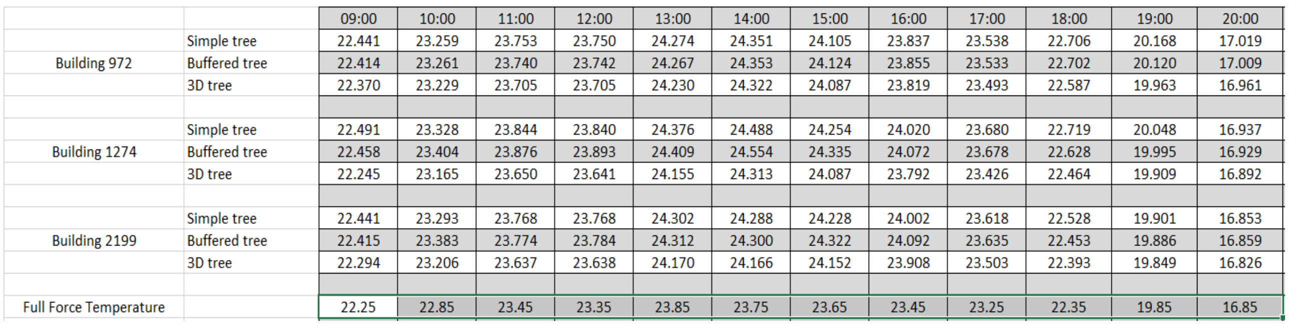


Figure 64: Green building: id 972, Blue building: id 1274, Red building: id 2199, with black the buildings that were assigned an average temperature value



		09:00	10:00	11:00	12:00	13:00	14:00	15:00	16:00	17:00	18:00	19:00	20:00
	Simple tree	22.441	23.259	23.753	23.750	24.274	24.351	24.105	23.837	23.538	22.706	20.168	17.019
Building 972	Buffered tree	22.414	23.261	23.740	23.742	24.267	24.353	24.124	23.855	23.533	22.702	20.120	17.009
	3D tree	22.370	23.229	23.705	23.705	24.230	24.322	24.087	23.819	23.493	22.587	19.963	16.961
	Simple tree	22.491	23.328	23.844	23.840	24.376	24.488	24.254	24.020	23.680	22.719	20.048	16.937
Building 1274	Buffered tree	22.458	23.404	23.876	23.893	24.409	24.554	24.335	24.072	23.678	22.628	19.995	16.929
	3D tree	22.245	23.165	23.650	23.641	24.155	24.313	24.087	23.792	23.426	22.464	19.909	16.892
	Simple tree	22.441	23.293	23.768	23.768	24.302	24.288	24.228	24.002	23.618	22.528	19.901	16.853
Building 2199	Buffered tree	22.415	23.383	23.774	23.784	24.312	24.300	24.322	24.092	23.635	22.453	19.886	16.859
	3D tree	22.294	23.206	23.637	23.638	24.170	24.166	24.152	23.908	23.503	22.393	19.849	16.826
Full Force Temperature		22.25	22.85	23.45	23.35	23.85	23.75	23.65	23.45	23.25	22.35	19.85	16.85

Figure 65: Hourly average values for the three buildings 9:00 - 20:00

The two main scenario simulations ran in multi-core, in two different computers with the same resolution as the first set (4*4*3) and the size of the grid was 272*274*31. The fox file that was used is displayed in figure 66. Some simulation results visualized in QGIS, are displayed in figure 67 and more results can be found in the appendix 4. These simulations took approximately 20 days to go through the 31-hour span.

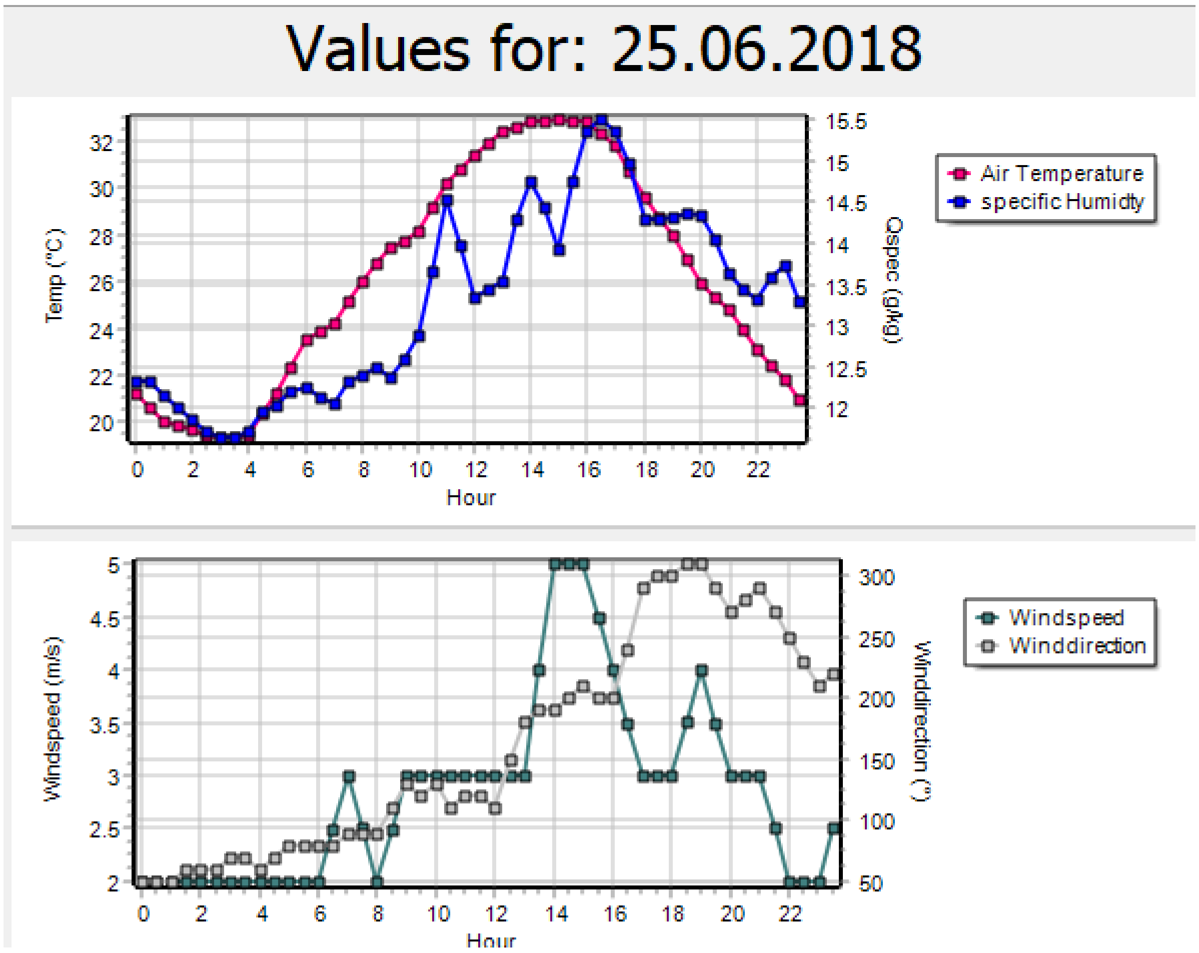


Figure 66: Full Forcing Diagram of the warmest day of the last 3 years

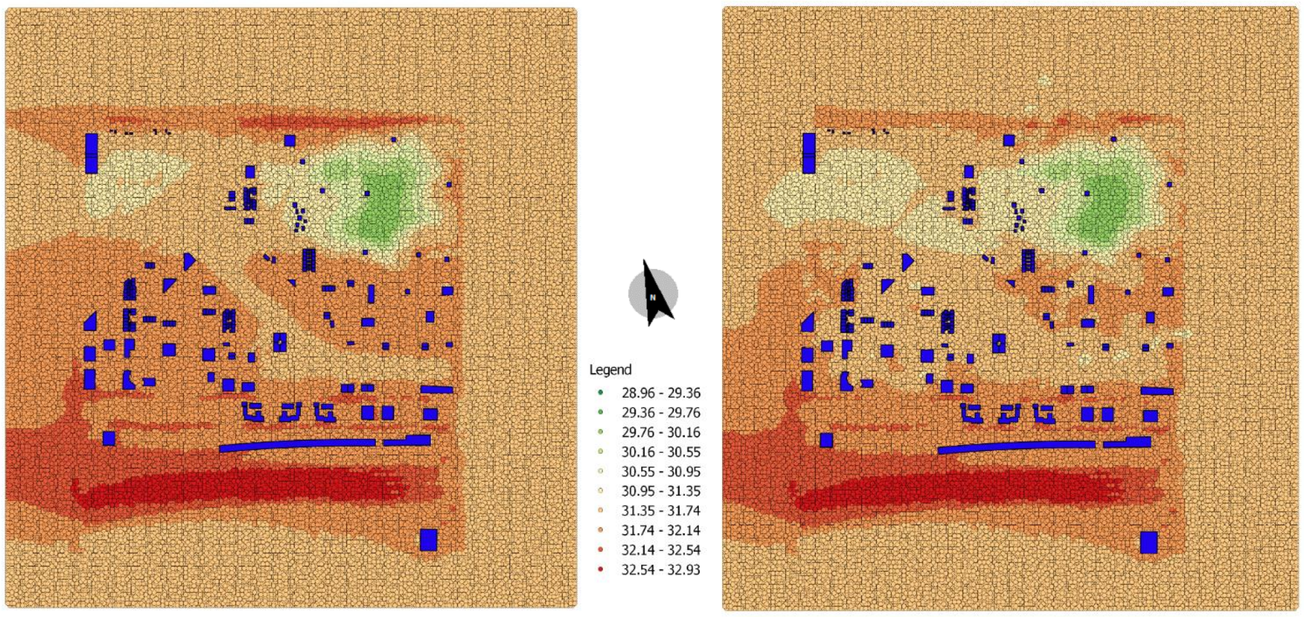
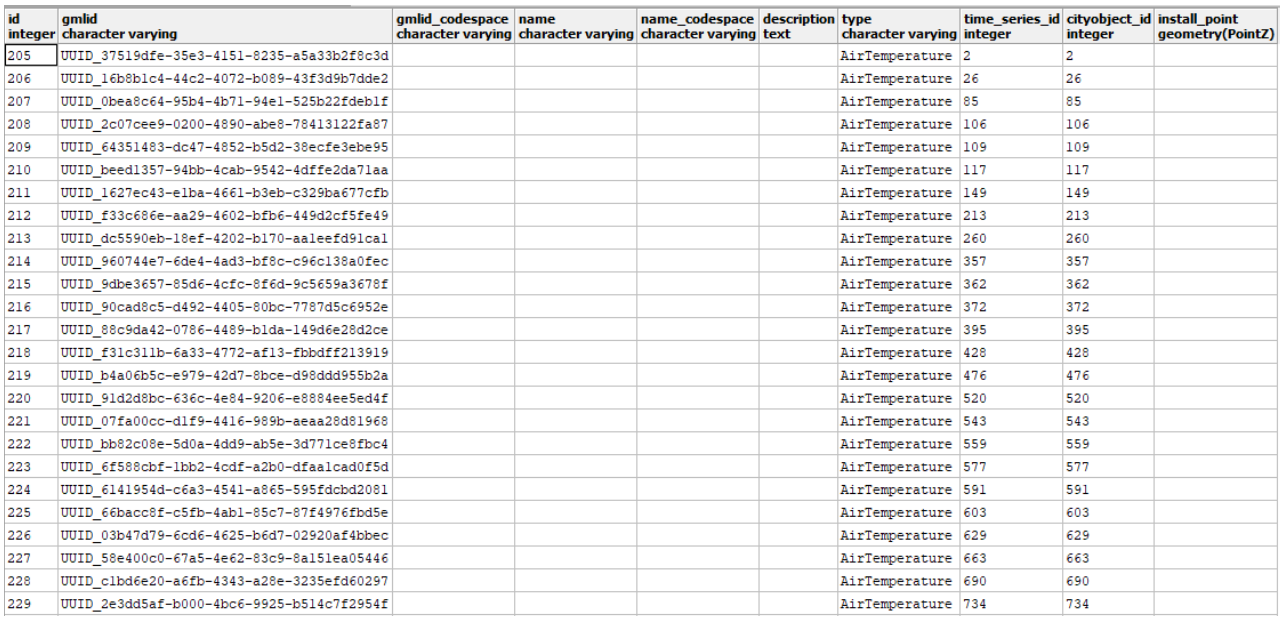


Figure 67: QGIS visualizations (following the terrain at Z = 1.5m, timestamp = 2018-06-25 12:00:01), left : scenario without arboretum trees, right : scenario with arboretum trees. Weather input parameters : Temperature (C) = 32.95, Rel Humidity (%) = 46, Wind Speed(m/sec) = 5, Wind Direction (Degrees) = 210. Reduced temperature are clearly visible above the district.

5.4.4 Storing back the results to the Floriade CityGML dataset

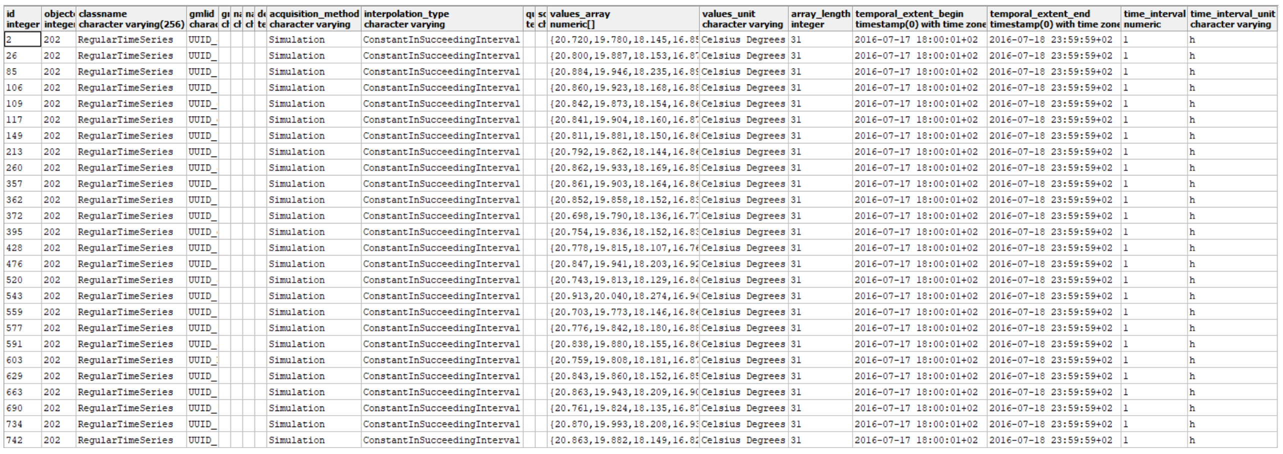
SclGModeLer can extract the geometry of the Atmosphere folder output values, by separating the cells coordinates from the compact, multi-indexed table that was discussed in section 4.3.2. In first place an attributed point cloud is uploaded to the database. Before processing the point cloud to acquire the final results, it is filtered twice, first by identifying default values (assigned by ENVI-met to cells that are covered by buildings or DEM), and secondly by removing the points that intersecting with CityGML building geometries. This point cloud is considered the second result of the interface. This filtering can be also used to validate the geometrical transformation, applied in both phases, as will be displayed in the next section.

Out of this point cloud data, extraction of meaningful results demands different interpolation methods. The same methodology for instance is meaningful, and already applicable to other similar parameters like radiation and humidity. On the other hand, the interpolation of parameters such as the human comfort index, demands point aggregation over different geometries. The interpolated air temperature values were stored back to the input CityGML dataset, as Energy ADE *TimeSeries* and *WeatherStation* supporting classes (Figure 68, Figure 69).



id integer	gmlid character varying	gmlid_codespace character varying	name character varying	name_codespace character varying	description text	type character varying	time_series_id integer	cityobject_id integer	install_point geometry(PointZ)
205	UUID_37519dfe-35e3-4151-8235-a5a33b2f8c3d					AirTemperature	2	2	
206	UUID_16b8b1c4-44c2-4072-b089-43f3d9b7dde2					AirTemperature	26	26	
207	UUID_0bea8c64-95b4-4b71-94e1-525b22fdeb1f					AirTemperature	85	85	
208	UUID_2c07cee9-0200-4890-abe8-78413122fa87					AirTemperature	106	106	
209	UUID_64351483-dc47-4852-b5d2-38ecfe3ebe95					AirTemperature	109	109	
210	UUID_beed1357-94bb-4cab-9542-4dffe2da71aa					AirTemperature	117	117	
211	UUID_1627ec43-e1ba-4661-b3eb-c329ba677cfb					AirTemperature	149	149	
212	UUID_f33c686e-aa29-4602-bfb6-449d2cf5fe49					AirTemperature	213	213	
213	UUID_dc5590eb-18ef-4202-b170-aa1eefd91ca1					AirTemperature	260	260	
214	UUID_960744e7-6de4-4ad3-bf8c-c96c138a0fec					AirTemperature	357	357	
215	UUID_9dbe3657-85d6-4cfc-8f6d-9c5659a3678f					AirTemperature	362	362	
216	UUID_90cad8c5-d492-4405-80bc-7787d5c6952e					AirTemperature	372	372	
217	UUID_88c9da42-0786-4489-b1da-149d6e28d2ce					AirTemperature	395	395	
218	UUID_f31c311b-6a33-4772-af13-fbbdff213919					AirTemperature	428	428	
219	UUID_b4a06b5c-e979-42d7-8bce-d98ddd955b2a					AirTemperature	476	476	
220	UUID_91d2d8bc-636c-4e84-9206-e8884ee5ed4f					AirTemperature	520	520	
221	UUID_07fa00cc-d1f9-4416-989b-aeaa28d81968					AirTemperature	543	543	
222	UUID_bb82c08e-5d0a-4dd9-ab5e-3d771ce8fbc4					AirTemperature	559	559	
223	UUID_6f588cbf-1bb2-4cdf-a2b0-dfaa1cad0f5d					AirTemperature	577	577	
224	UUID_6141954d-c6a3-4541-a865-595fdcbd2081					AirTemperature	591	591	
225	UUID_66bacc8f-c5fb-4ab1-85c7-87f4976fbd5e					AirTemperature	603	603	
226	UUID_03b47d79-6cd6-4625-b6d7-02920af4bbec					AirTemperature	629	629	
227	UUID_58e400c0-67a5-4e62-83c9-8a151ea05446					AirTemperature	663	663	
228	UUID_c1bd6e20-a6fb-4343-a28e-3235efd60297					AirTemperature	690	690	
229	UUID_2e3dd5af-b000-4bc6-9925-b514c7f2954f					AirTemperature	734	734	

Figure 68: Weather Data, 3DCityDB



id integer	objecti integer	classname character varying(256)	gmlid charac	gı ch	na ch	na ch	de te	acquisition_method character varying	interpolation_type character varying	qu te	sc ch	values_array numeric[]	values_unit character varying	array_length integer	temporal_extent_begin timestamp(0) with time zone	temporal_extent_end timestamp(0) with time zone	time_interval numeric	time_interval_unit character varying
2	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.720,19.780,18.145,16.85	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
26	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.800,19.887,18.153,16.87	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
85	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.884,19.946,18.235,16.89	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
106	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.860,19.923,18.168,16.88	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
109	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.842,19.873,18.154,16.86	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
117	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.841,19.904,18.160,16.87	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
149	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.811,19.881,18.150,16.86	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
213	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.792,19.862,18.144,16.86	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
260	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.862,19.933,18.169,16.89	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
357	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.861,19.903,18.164,16.86	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
362	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.852,19.858,18.152,16.83	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
372	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.698,19.790,18.136,16.77	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
395	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.754,19.836,18.152,16.83	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
428	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.778,19.815,18.107,16.76	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
476	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.847,19.941,18.203,16.92	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
520	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.743,19.813,18.129,16.84	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
543	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.913,20.040,18.274,16.94	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
559	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.703,19.773,18.146,16.86	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
577	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.776,19.842,18.180,16.88	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
591	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.838,19.880,18.155,16.86	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
603	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.759,19.808,18.181,16.87	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
629	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.843,19.860,18.152,16.85	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
663	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.863,19.943,18.209,16.90	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
690	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.761,19.824,18.135,16.87	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
734	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.870,19.993,18.208,16.93	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h
742	202	RegularTimeSeries	UUID_					Simulation	ConstantInSucceedingInterval			{20.863,19.882,18.149,16.82	Celsius Degrees	31	2016-07-17 18:00:01+02	2016-07-18 23:59:59+02	1	h

Figure 69: Time series 3DCityDB

5.5 Methodology evaluation and validation

This section will evaluate the methodology, both from time performance and from potentiality of extension, in order to cover a wider spectrum of CityGML classes and attributes. Moreover, it will provide a proof of the validity of the measured temperatures.

5.5.1 Interface performance

As mentioned in section 4.3.1 the computation time for the creation of the INX model is considered satisfactory and negligible compared to ENVi-met simulation time. The two main concerns of the first face are first, that the building plotting should be as fast as possible, near to real time, since the definition of the desired cutting rectangle is subject to multiple tests. This is ensured by performing the intersections directly on the database, so the query filters all, but the required buildings. Secondly the number of DEM cells must be kept minimum, since the writing of a vast number of cells with xml-etree might take up several minutes. For the second phase of SclGModeLer, as it is designed now, the bottleneck appears on the transferring of data from the pandas DataFrame structure to the database, since this is applied by a sequential scanning of the DataFrame, by using a simple loop. An additional option is to use sqlAlchemy library, which was used for storing the weather data to the database. In this case it is believed that the uploading time will be optimized, however the DataFrame will require additional prior formatting. As far as the construction of the DataFrame by the batch reading of the hourly ED tuples is concerned, the performance is considered absolutely satisfactory. The imported data after all consist of millions of cell values, and the extraction of geometry comes down to minute level.

5.5.2 Methodology reflections and current limitations

This section will provide an insight on the methodology. Furthermore, it will outline some first phase issues that are not yet fixed and will present the most important limitations of the result interpolations.

From a first point of view, it is evident that the strength of CityGML, as an input for ENVI-met in this approach, lies on the combination of the different urban features into one single data model. As far as the 3D factor is concerned, the majority of the 3-Dimensional spatial information was extracted mainly by the spatial attributes (except for the DEM). Following a 3-dimensional voxelization approach based on the geometries, for mapping buildings and trees to 2.5D INX files would be meaningless, since for the former, the required action is a much simpler, rasterization and extrusion, and for the latter, tree geometry is generalised to the implicit vegetation one, adjusted by the transformation matrix. On the other hand, the standard architecture of the CityGML data model, and the semantic factor played a major role on reserving the automatization of the first phase. However, it would be wrong to claim that the methodology can substitute the human factor at this moment, since the current approach relies on a *LandUse* module which differs from CityGML's specifications, in terms of the created *GenericAttribute* and its LOD. A *LandUse* module of lower LOD and without the material *GenericAttribute* will not be able to feed the required data for creating an area input file of fine soil resolution, and would rather require, at least, the contribution of a high LOD *Water* and *Transportation* module. Thus, in this scenario, as mentioned in section 3.5.1, the definition of a rich CityGML dataset is related to the assigned application.

A significant problem that the user should be aware of, is the definition of the vertical resolution and it is highly recommended to be set as even number. The reason is that currently all the heights (building and dem) are rounded by a ceiling method twice; first for their registration in the matrix-data tag, and secondly when they are voxelised by ENVI-met) leading to overestimation of heights.

As far as the second phase is concerned, the interpolation approach is related to LOD availability. In case of an available LOD2, average temperature can be aggregated on façade level, which indeed is useful. However aggregating values to features, not included in the first phase (roof facades, balconies), or features lost in the voxelization process is something that should be avoided. A suitable way to solve the latter would be to apply a second feature filtering (out of the reverse-buffered cutting polygon). It can be realized by including a separate table of the rasterized object's (buildings or trees) ids in the generated metadata file.

5.5.3 Implementation quality checks of the interface

In order to provide a proof of the geometric validation of the results, the generated point cloud was filtered by the default air temperature values, assigned by ENVI-met, on occupied cells. Occupied cells in this case refer to cells that are occupied by DEM or buildings, since tree cells are transparent and therefore contain weather parameters. Since the knowledge over ENVI-met default values was missing, all the temperature array duplicates were removed. However, the possibility for ENVI-met to apply default values to those cells should be examined. Furthermore, attribute values of different parameters can be checked, to see if they obtain default values. Slices of points on specific heights, as well as the buildings, used on the first phase, were converted to shapefiles by the interface, and visualized in QGIS Figure 70 shows a horizontal slice of the point cloud, after the first filtering (filtered by just default assigned values) overlaying the building footprints; it is evident that all the geometric transformations applied in both phases are correct since both datasets are perfectly aligned.

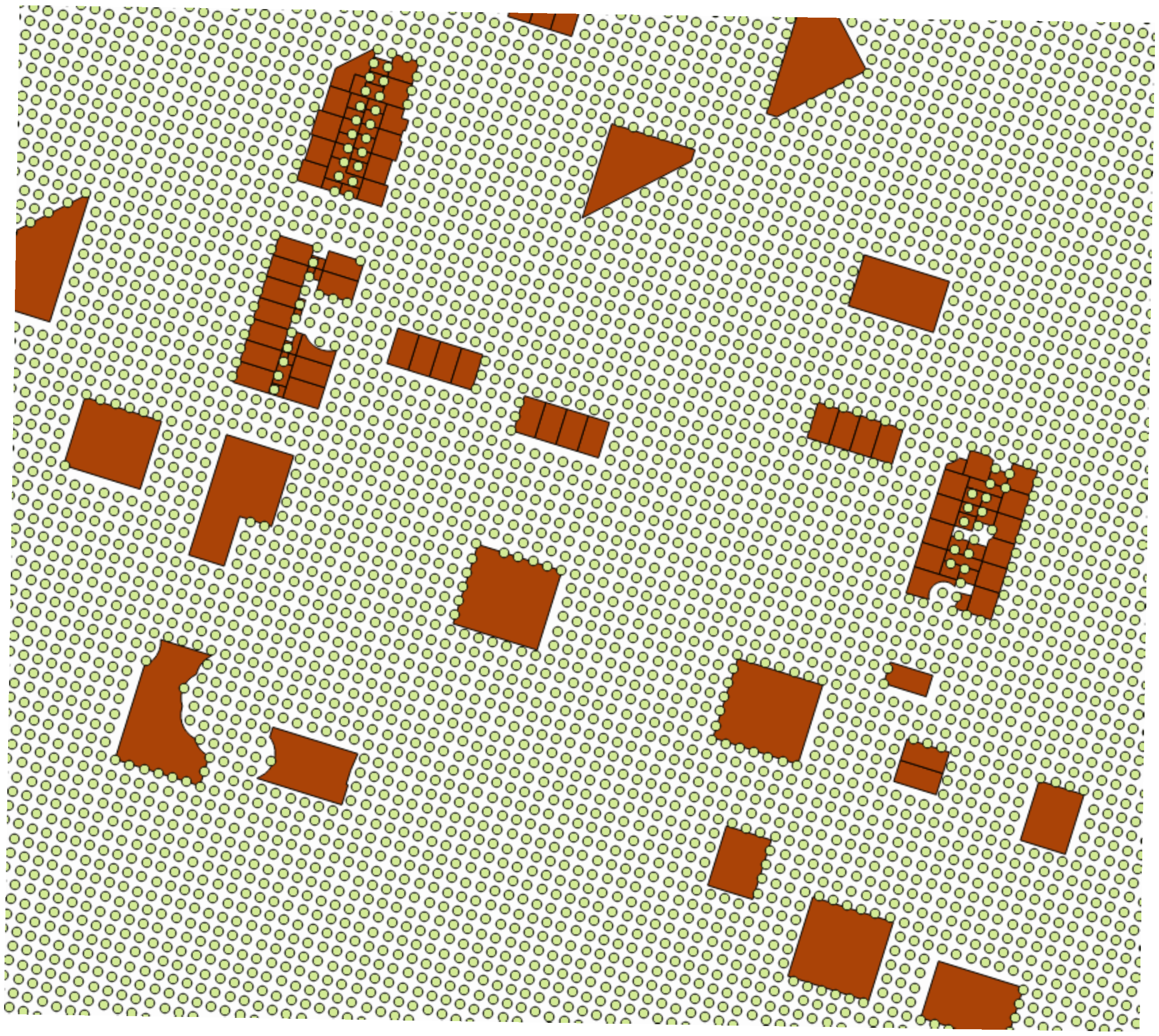


Figure 70: Visualization of horizontal slice of air temperature point cloud in QGIS

The result is that the holes, product of the filtering, fit exactly with the building geometry. Secondly the validity of the definition of the buffered facades that were used to capture points is displayed in Figure 71. Moreover, the points, used for the extraction of the building average temperature were visualized in FME (Figure 72). Finally, a comparison between Leonardo visualizations and the point cloud slices, visualized in QGIS is a rough, but efficient way of validating the results (Figure 73, 74).

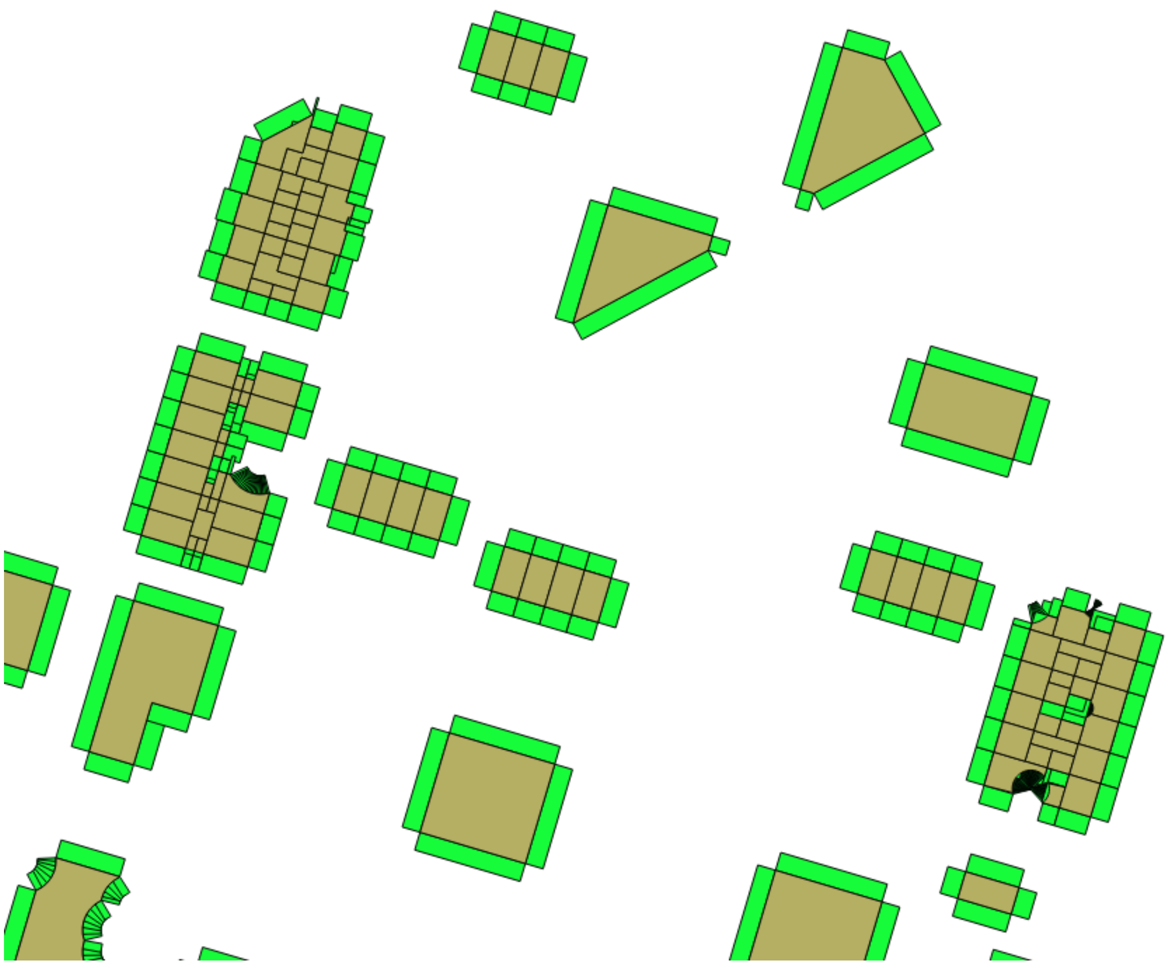


Figure 71: Buffered footprint facades

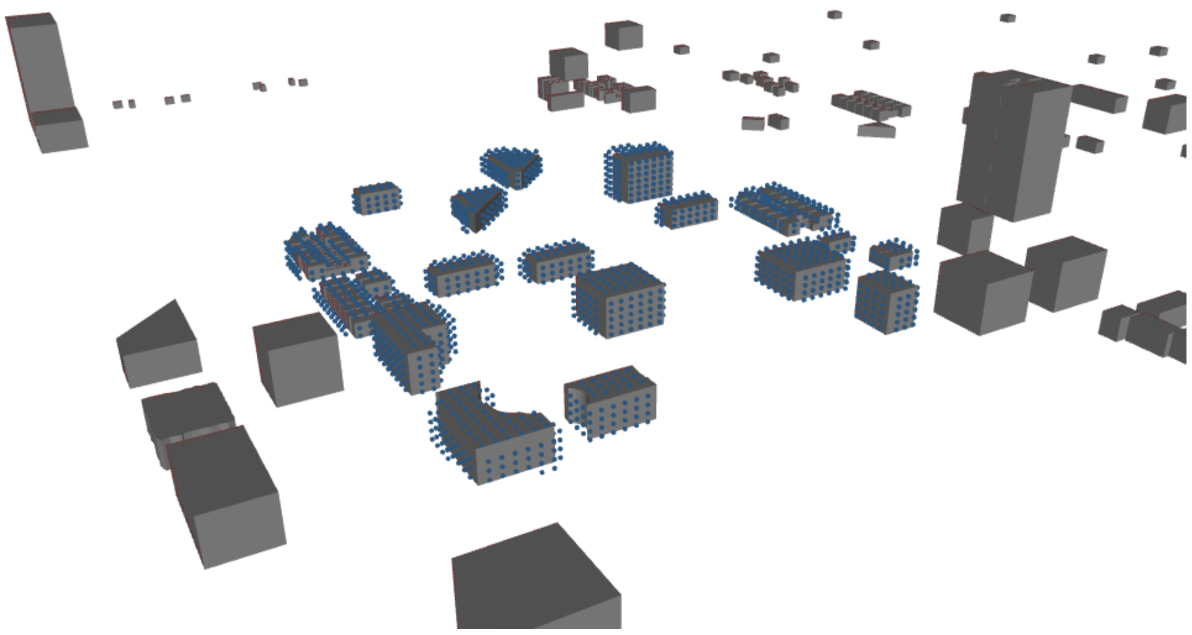


Figure 72: Points used for average temperature extraction

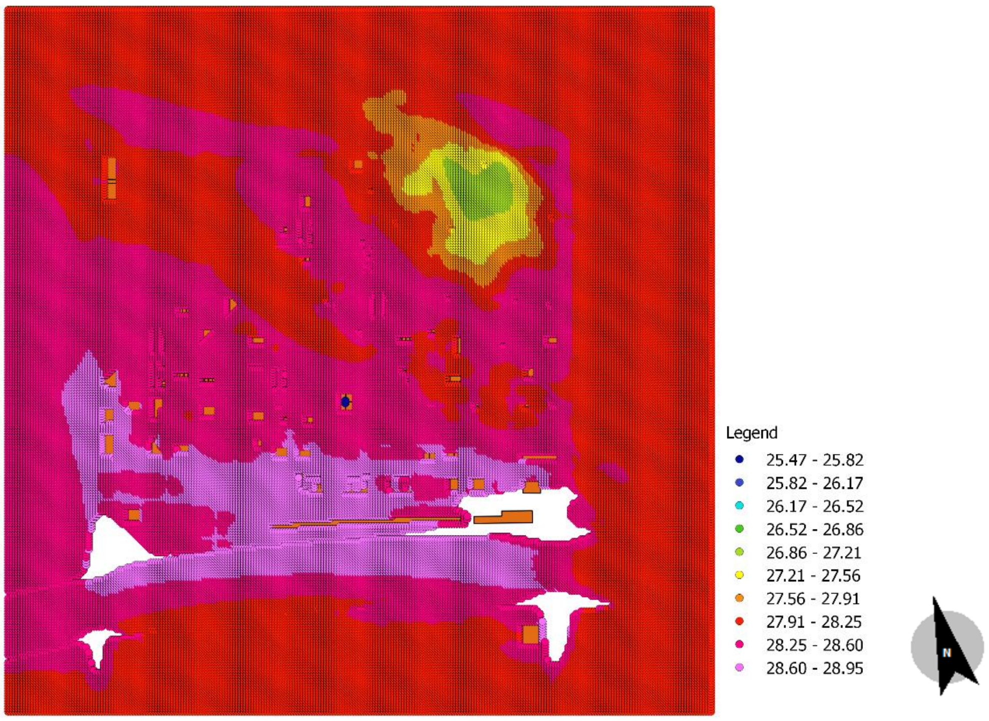


Figure 73: QGIS shapefile of air temperature at 10:00 at Z = 0.4m, scenario with 3D trees

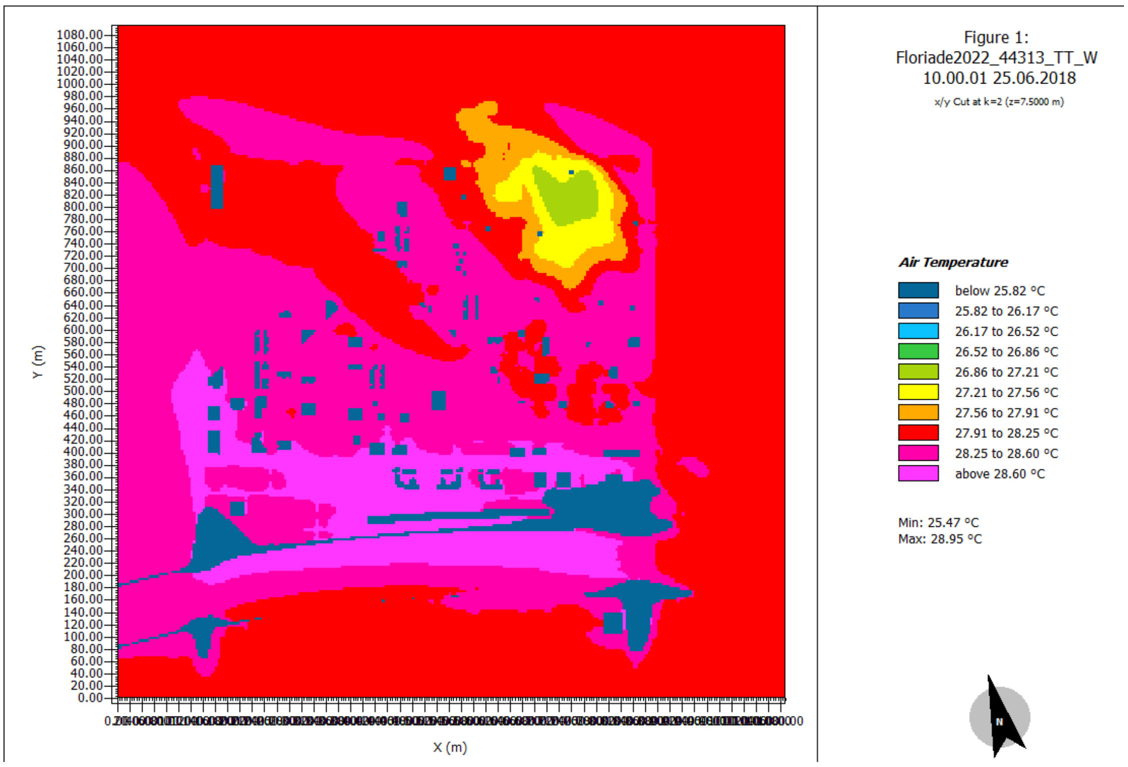


Figure 74: Same visualization on Leonardo

6. Conclusions

The goal of this thesis was to perform microclimatic simulations with the ENVI-met software, by using a semantic 3D model as input and at the same time as the storage hub of the simulation results. As a case study area, the under-development district of Floriade, in Almere was chosen, which is designed around the concept of green village. Instead of following the most common approach and design the area input file that is used as input by ENVI-met, in the built-in editor, a CityGML-based semantic 3D model of the area was created, out of a set of datasets provided by the municipality of Almere. Additionally, an interface was created that can convert the CityGML data model, to the area input file data model and also parse ENVI-met's outputs and return meaningful data as Energy ADE *TimeSeries* and *WeatherData* classes.

This chapter is composed out of the following relevant subsections. In section 6.1, the research questions introduced in section 1.2 are answered. In section 6.2 a summary of the whole contribution of this thesis is discussed.

6.1 Research Questions

During the introduction, a main research question and a number of additional sub questions were asked, that also helped in guiding me throughout the extent of the research. I will start by answering the sub-questions, and later a reflection on the main question will be discussed.

A. <u>Regarding the coupling between CityGML + Energy ADE and ENVI-met</u>

1. Which are the data requirements of ENVI-met?

ENVI-met as a microclimate simulation software requires an accurate estimation of the weather parameters on its borders, and the geometry and physical and physiological characteristics of the district to be modelled, in a raster format model, called area input file. The former includes the diurnal conditions of air-temperature, wind speed and direction, and relative humidity. The latter includes the geometry and surface material of buildings, the topography and type of soils/surface materials including water and the position and type of trees. Moreover ENVI-met contains pre-built databases of trees and soils/material characteristics that can be used on the area input file. User can create own trees and materials, but in that case the new database should be mentioned in the simulation file.

2. Can they be met by CityGML + Energy ADE?

CityGML data model was explained in section 3.5 and the correspondences to ENVI-met required data are displayed on section 3.6. It was found that CityGML alone can cover a sufficient number of ENVI-met requirements, mainly of those concerning the area input file, as long as a correspondence between both data models codelists has been set. Furthermore, an augmented by means of Energy ADE CityGML dataset, can be potentially sufficient for covering all the ENVI-met requirements. It was found that some features, conceived both in CityGML + Energy ADE and ENVI-met were design under the same principles, like for instance the structured materials. In Energy ADE, the materials can be composed of different layers, while ENVI-met new versions include a 7-node model for their representation. Thus, Energy ADE *Materials and Construction* class can potentially be mapped to an ENVI-met materials database, which can be used instead of ENVI-met default database for surface materials. Finally, concerning the full-3D INX file, I didn't try to include it in this thesis, mainly due to lack of documentation. But since it is also a text file, and XML encoded its interpretation would be possible. In that case a voxelization algorithm

3.How to perform the mapping between the two data models, and how to implement the interface that reads and writes data?

As mentioned in sections 3.7 and 4.2, the 3DCityDB was preferred for storing the CityGML based city model. Thus, the interface pre-requires that the user has successfully installed 3DCityDB and has created a database instance for storing the dataset to be mapped. In order to create rich in terms of content INX files, the following classes needs to be filled with data. First, buildings should be mapped according to LOD0, and contain a *measured_height* attribute. As building id, the respective index, on 3DCityDB building table is used, instead of the full CityGML id. Secondly, trees should be modelled as implicit vegetation objects, and contain a height and a crown diameter height. In this way, a tree can be effectively generalized to a 3D tree or a buffered tree (objects that are used by ENVI-met). For modelling water and forests, a *LandUse* module needs to contain the attributes class, and function while for an accurate representation of soils, the *LandUse* module must be augmented with a *GenericAttribute* called material, that can take values according to the codelist displayed on section 3.5.1 (figure…). Finally, for an accurate modelling of the topography, a *TinRelief* is required. In order to return ENVI-met generated data to the input dataset, the 3DCityDB must be extended with the Energy ADE tables. Currently, SclGModeLer can return average air temperature on building level. The method to match ENVI-met output geometry to the input dataset's coordinate system is to apply reverse translation on the cells centroids and return an attributed point cloud back to the database. Furthermore, by working in this thesis project, I found that ids can be preserved when using ENVI-met, so a different way to return generated data could be directly by the matching of ids.

4.What is needed in order to enrich the CityGML model with simulation outputs, should another ADE be needed?

ENVI-met produces vast hourly data, so the required format to store it should be a dynamic class with an efficient way of storing data. Luckily, Energy ADE contains the classes *WeatherData* and *TimeSeries* which are designed to cover the need of storing time-influenced parameters. A regular *TimeSeries* class was filled with air temperature data in this case. The *TimeSeries* class in this case contain the interpolation type (Average), a starting timestamp, a timestep (1 hour), a number of entries and an array of measured values (hourly average values aggregated on buildings). The *WeatherData* class is used to associate a timeseries class to CityGML features; a building can therefore be augmented with a class that includes the name of the measured parameter (Air Temperature), and the respective time series.

The aforementioned classes were adequate for storing weather data, so this thesis did not examine alternative ADEs for this purpose. However, taking into account the vast number of ENVI-met outputs, it is safe to claim that more ADE classes and different ADEs can benefit from a coupling between CityGML and ENVI-met, but as a matter of fact, it is necessary that ENVI-met results meet the ADE's specification.

5.Where do the main difficulties lie?

From my experience by developing the coupling methodology, I would point five main difficulties.

1. **Automated Rasterization**: When designing area input models in a raster-based editor, the user can be focused in preserving objects geometry instead of keeping the exact right distances. This is the recommended way of designing an area input file. When the rasterization is performed automatically, this control over the final product is therefore lost. The interface includes a method to deal with this problem (see section 4.2.1) but it can't be completely fixed.
2. **Vertical profile**: The implemented methodology is currently limited to the creation of INX files of a telescoping vertical grid (see section 3.1.1). This grid was the simplest to begin with. However, an equidistant grid with a lower cell splitting, requires a different mapping approach, at least for the full 3D INX file. Furthermore, features like grass that couldn't be modelled in a telescoping grid, should be included in the mapping.
3. **Façade materials**: This section was not thoroughly investigated, however they way materials are conceived in the INX file is not the desired one. Materials on the left and bottom side of buildings are depicted in building occupied voxels, but for the right and upside this information is depicted on the adjusted empty cells (see Figure 25, section 3.3.4). Thus, any algorithm created for this purpose, should eventually compromise with this peculiarity.
4. **Trees and codelist matching**: A detailed soil depiction in the INX file is always difficult and requires a lot of available information, even when designing the model manually. SclGModeLer, relies on the availability of a high LOD *LandUse* module which is not always met in a CityGML dataset.

Furthermore, trees could be modelled according to their geometry, geometric attributes or species. Currently in SclGModeLer, trees are modelled according to their geometry attributes *height* and *crown diameter*. While this approach ensures the accurate geometry of the tree objects in the INX file, it might miss other foliage parameters, like the LAD index (Leaf Area Density).

5. **ENVI-met returning atmospheric values for occupied cells, difficult formats, not focused to be compatible with applications:** By examining ENVI-met outputs I discovered that it assigns weather parameter values to cells that are already occupied by buildings or the DEM. Moreover, these values do not follow a default pattern, at least for air temperature. The methodology relies on the accuracy of ENVI-met results. As a matter of fact, these cells had to be filtered, and two filtering methods are currently applied (see section 5.7.3). It is believed that a third filtering (before aggregating the points), based on a least square adjustment is required to ensure the validity of the final results. Otherwise other more suitable variable values have to be examined.

B. <u>Regarding the testing of the methodology within the case study in Almere.</u>

1. How to gather, integrate and harmonise the data to prepare the 3D city model of the test area for the status quo situation?

The methodology to create the Floriade CityGML dataset, from the available data is described step by step in sections 5.2 and 5.3. It is important that the data should be first processed in order to be cleaned. Additionally, the attributes must be imported in a csv form. Then in safe software's FME, four workbenches were created (one for the four different thematic modules that were created) that can automatically convert the input data to CityGML.

2. How to define and characterize the other simulation scenarios?

In order to test the tree influence in the district. After a meeting with people from the municipality of Almere, several ideas were discussed, and we reached an agreement that included two simulations. The first would include trees that are about to be planted on the district's arboretums, and the second scenario would be without those trees. For evaluating the impact of the different vegetation types that ENVI-met uses, 3 test simulations were also conducted.

3. How to generate the models of the different scenarios previously defined?

The simulated scenarios did not include any difference concerning the structure domain. In order to simulate scenarios that include or exclude the arboretum trees, solitary vegetation objects were grouped into two *CityObjectGroups* (one including the arboretum trees that are about to be planted, and the other included already existing trees). In 3DCityDB features can be also deleted easily, and one condition can be to delete features belonging to a *CityObjectGroup*.

6.2 Automatization evaluation – state of the art

In the problem statement (section 1.1), the need for storing microclimate data in a standard format, so it can be re-used in future applications was indicated. Furthermore section 2.4 pointed the disadvantages of designing the area input file in the raster-based SPACES editor. Using vector-based inputs for creating the ENVI-met area input file geometry is a method that is already implemented in the Rhino plugin and ENVI-met recently developed editor, MONDE. The hypothesis of this thesis was that a semantic 3D model, based on CityGML would include all, or at least all the important data, required by ENVI-met, so that the user's effort will be minimized. At the same time the input model would be enriched with ENVI-met outputs, since all the required (for the data transformation) geometry translations could be reversed. In this way microclimatic data would be stored and be ready for further application. Besides the storing of microclimatic data, it can also be visualized and processed in GIS platforms, since by this approach, ENVI-met provide outputs with real coordinates, instead of the local Cartesians of the 3D grid.

The methodology points towards two main conclusions. The first one is that semantic 3D models can indeed improve the current state of the art, in terms of automatic ENVI-met area input file generation. In addition, future improvement should consider that a CityGML dataset, augmented by Energy ADE could potentially be translated to a high detailed full 3D area input file, including detailed building material allocation. The design of buildings, trees and soils in SPACES, despite being complicated, can be conducted relatively fast. However, assigning materials to building facades is currently impossible in district scale, since it demands 3-dimensional editing of every single façade cell. (Figure 25)

The second conclusion is that both CityGML + Energy ADE and ENVI-met can benefit by this coupling approach. Beginning with ENVI-met, it becomes compatible with an additional input model, which is also the dominant standard at district scale. Furthermore, with the achieved level of automatization, the saved time and effort can be used to perform simulations of larger domains. To conclude with CityGML, applying the thesis approach pre-requires the availability of CityGML data, thus an additional motivation for harmonizing geospatial data according to CityGML is emerging. Finally, CityGML + Energy ADE status on urban planning will be increased, since it can now include weather data of higher resolution and accuracy, than empirical weather data, interpolated by nearby weather stations.

6.3 Future work

Based on the research overview and the fields that were not examined due to limitation time, a number of future tasks that would complement the methodology is briefly proposed in the following bullets.

- Evaluate the mapping of more CityGML classes : Such as *Transportation, WaterBodies* that – if exist – could be directly mapped. In this way the interface can be direct applicable over more CityGML datasets, that do not nesessairy include a specific *LandUse* module. As mentioned in chapter four, this mapping follows the exact same approach.
- Construct the Full 3D area input file : This task would require a different approach when mapping the *Building* module, and the resolution robustness has to be carefully examined. In this case, a voxelization algorithm is required, instead of a 2D rasterization, as well as a higher available LOD (LOD2). In a Full 3D area input file, detailed building facade materials could be modelled, saving a lot of manual effort (placing materials manually in SPACES is practically impossible). This data could be either retrieved by a LOD3 *Building* module (Windows – Doors), or by Energy ADE *Material and Construction* class.
- Examine and return more attributes : As it was stated in previous chapters returning more attributes should be feasible, however different data requirements (and most probably additional ADEs), and interpolation methods should be examined. Adittional data can be stored back to the model with the same reverse geometry transformation, or by ID mathcing.
- Further test of the Interface

Appendices

Appendix 1: Uml Diagrams

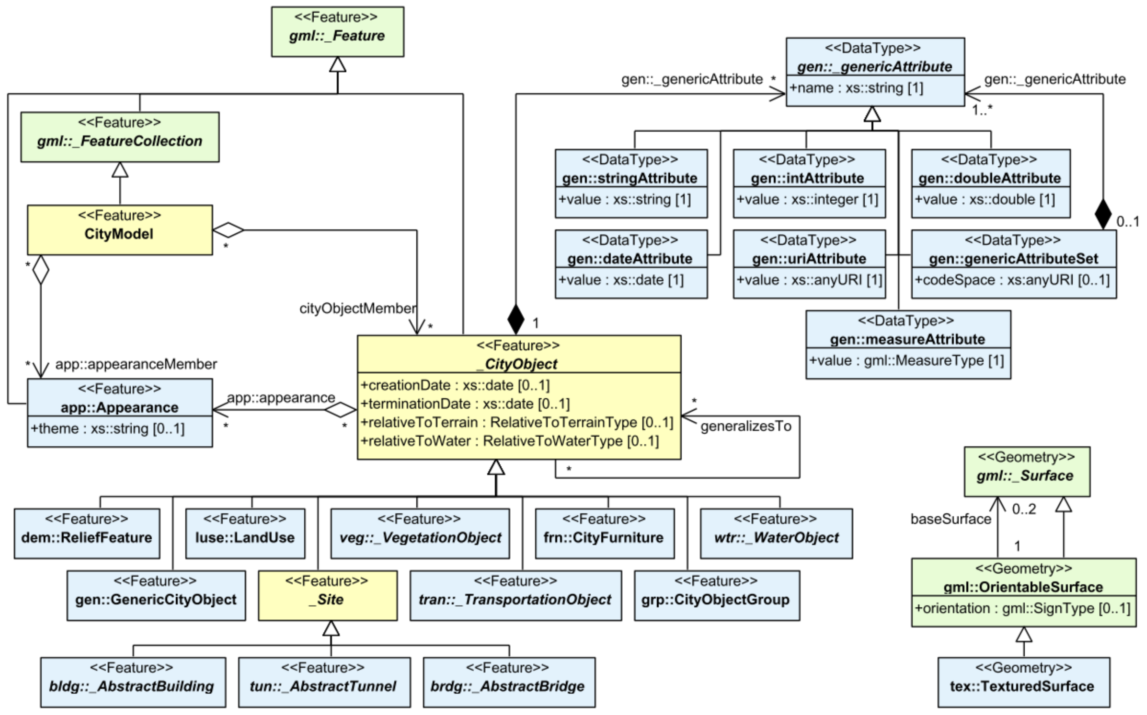

<<Feature>> gml::_Feature
<<Feature>> gml::_FeatureCollection
<<Feature>> CityModel
<<Feature>> app::Appearance
+theme : xs::string [0..1]
app::appearanceMember
cityObjectMember
app::appearance
<<Feature>> _CityObject
+creationDate : xs::date [0..1]
+terminationDate : xs::date [0..1]
+relativeToTerrain : RelativeToTerrainType [0..1]
+relativeToWater : RelativeToWaterType [0..1]
generalizesTo
<<DataType>> gen::_genericAttribute
+name : xs::string [1]
gen::_genericAttribute
<<DataType>> gen::stringAttribute
+value : xs::string [1]
<<DataType>> gen::intAttribute
+value : xs::integer [1]
<<DataType>> gen::doubleAttribute
+value : xs::double [1]
<<DataType>> gen::dateAttribute
+value : xs::date [1]
<<DataType>> gen::uriAttribute
+value : xs::anyURI [1]
<<DataType>> gen::genericAttributeSet
+codeSpace : xs:anyURI [0..1]
<<DataType>> gen::measureAttribute
+value : gml::MeasureType [1]
<<Feature>> dem::ReliefFeature
<<Feature>> luse::LandUse
<<Feature>> veg::_VegetationObject
<<Feature>> frn::CityFurniture
<<Feature>> wtr::_WaterObject
<<Feature>> gen::GenericCityObject
<<Feature>> _Site
<<Feature>> tran::_TransportationObject
<<Feature>> grp::CityObjectGroup
<<Feature>> bldg::_AbstractBuilding
<<Feature>> tun::_AbstractTunnel
<<Feature>> brdg::_AbstractBridge
<<Geometry>> gml::_Surface
baseSurface
<<Geometry>> gml::OrientableSurface
+orientation : gml::SignType [0..1]
<<Geometry>> tex::TexturedSurface

Figure 75: UML Diafram of abstract class _CityObject

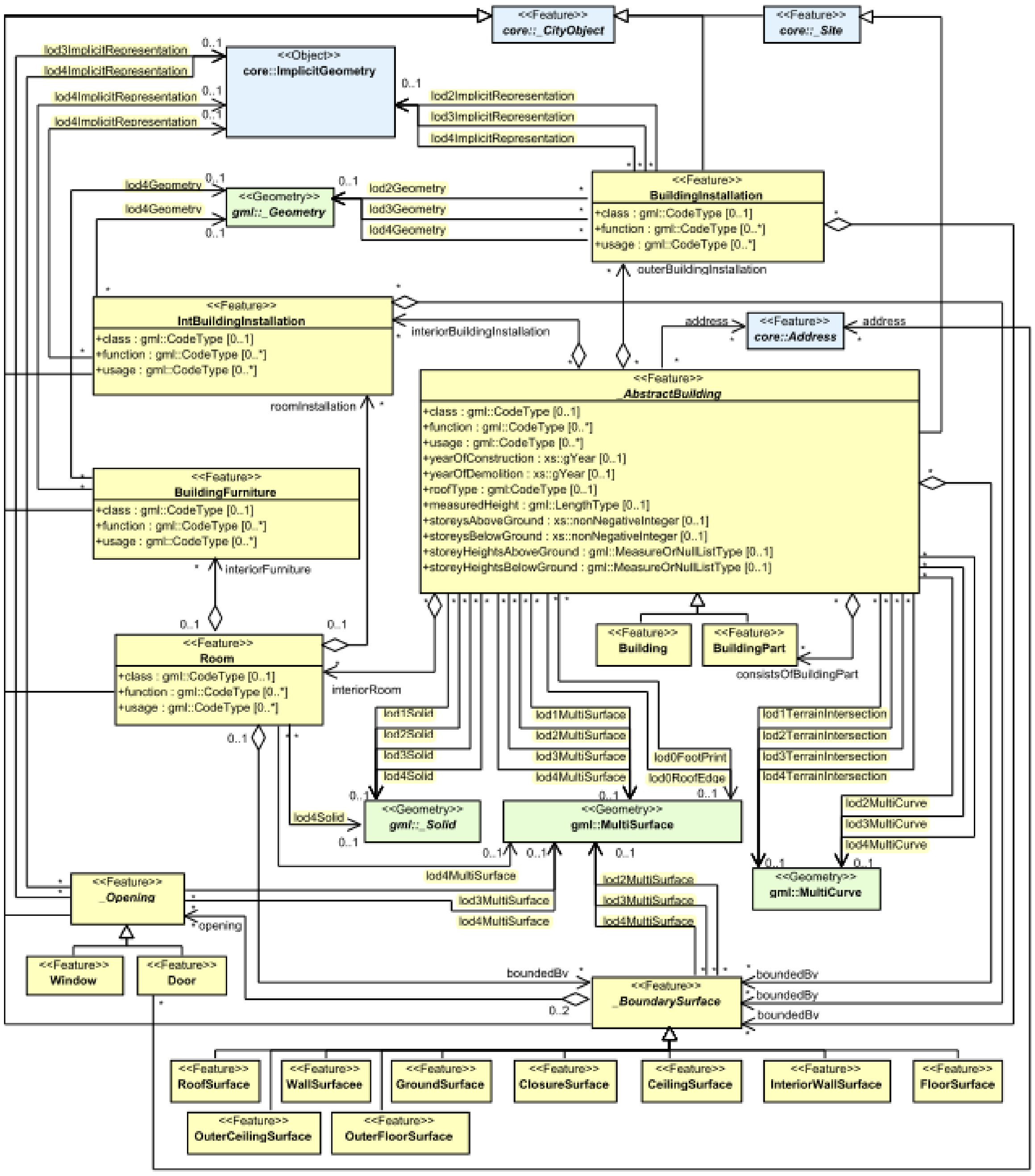

<<Feature>> core::_CityObject
<<Feature>> core::_Site
<<Object>> core::ImplicitGeometry
lod3ImplicitRepresentation
lod4ImplicitRepresentation
lod2ImplicitRepresentation
<<Geometry>> gml::_Geometry
lod2Geometry
lod3Geometry
lod4Geometry
<<Feature>> BuildingInstallation
+class : gml::CodeType [0..1]
+function : gml::CodeType [0..*]
+usage : gml::CodeType [0..*]
outerBuildingInstallation
<<Feature>> IntBuildingInstallation
interiorBuildingInstallation
<<Feature>> core::Address
address
<<Feature>> _AbstractBuilding
+class : gml::CodeType [0..1]
+function : gml::CodeType [0..*]
+usage : gml::CodeType [0..*]
+yearOfConstruction : xs::gYear [0..1]
+yearOfDemolition : xs::gYear [0..1]
+roofType : gml:CodeType [0..1]
+measuredHeight : gml::LengthType [0..1]
+storeysAboveGround : xs::nonNegativeInteger [0..1]
+storeysBelowGround : xs::nonNegativeInteger [0..1]
+storeyHeightsAboveGround : gml::MeasureOrNullListType [0..1]
+storeyHeightsBelowGround : gml::MeasureOrNullListType [0..1]
roomInstallation
<<Feature>> BuildingFurniture
interiorFurniture
<<Feature>> Room
interiorRoom
<<Feature>> Building
<<Feature>> BuildingPart
consistsOfBuildingPart
lod1Solid
lod2Solid
lod3Solid
lod4Solid
lod1MultiSurface
lod2MultiSurface
lod3MultiSurface
lod4MultiSurface
lod0FootPrint
lod0RoofEdge
lod1TerrainIntersection
lod2TerrainIntersection
lod3TerrainIntersection
lod4TerrainIntersection
lod2MultiCurve
lod3MultiCurve
lod4MultiCurve
<<Geometry>> gml::_Solid
<<Geometry>> gml::MultiSurface
<<Geometry>> gml::MultiCurve
<<Feature>> _Opening
opening
<<Feature>> Window
<<Feature>> Door
boundedBy
<<Feature>> _BoundarySurface
<<Feature>> RoofSurface
<<Feature>> WallSurfacee
<<Feature>> GroundSurface
<<Feature>> ClosureSurface
<<Feature>> CeilingSurface
<<Feature>> InteriorWallSurface
<<Feature>> FloorSurface
<<Feature>> OuterCeilingSurface
<<Feature>> OuterFloorSurface

Figure 76: UML diagram of class Building

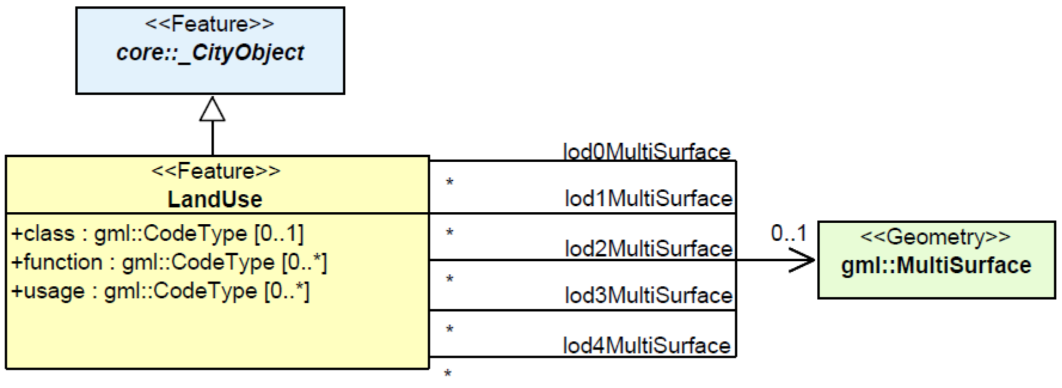


Figure 77: UML diagram of class LandUse

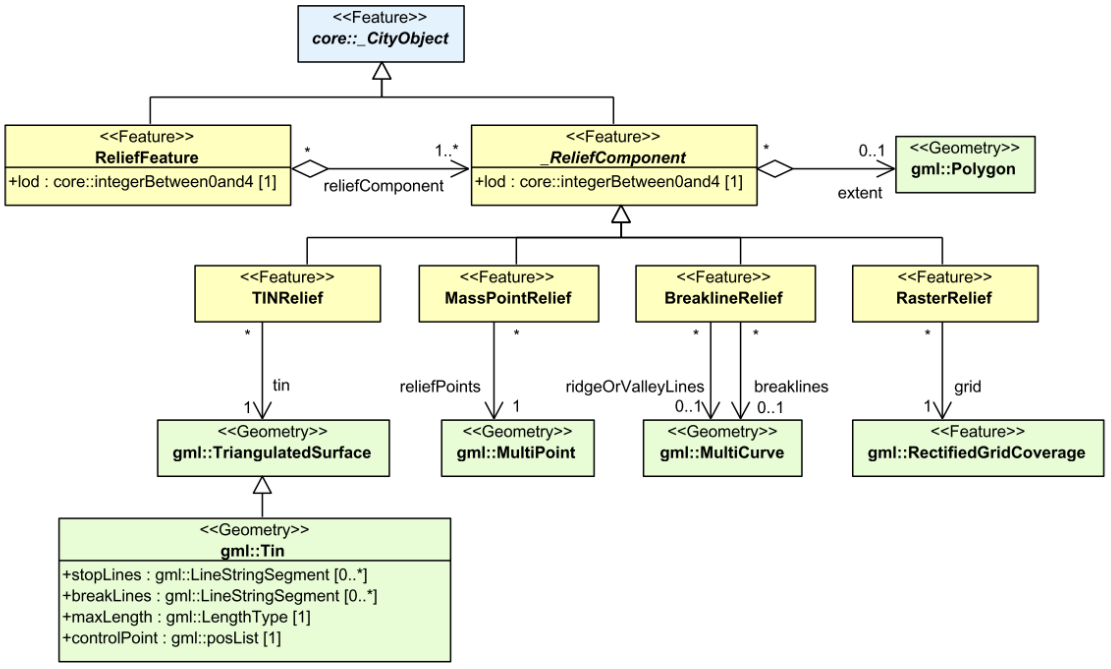


Figure 78: UML diagram of class Relief

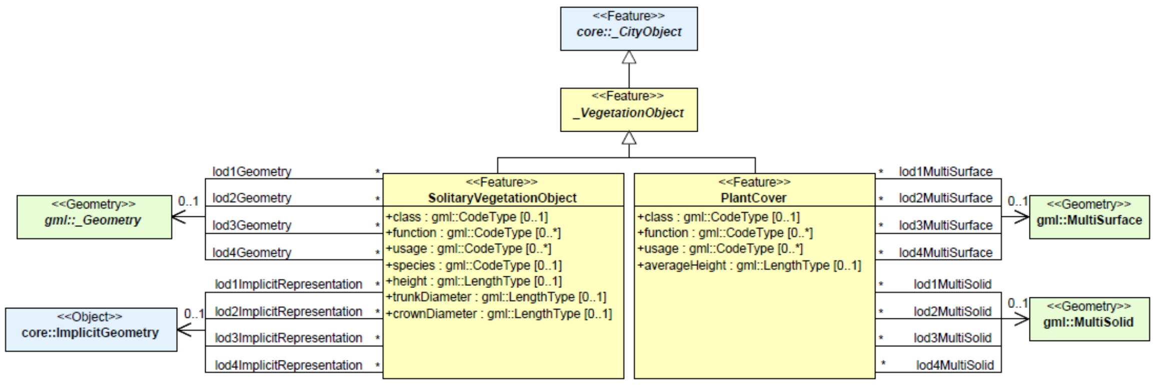

<<Feature>>
core::_CityObject
<<Feature>>
_VegetationObject
<<Feature>>
SolitaryVegetationObject
+class : gml::CodeType [0..1]
+function : gml::CodeType [0..*]
+usage : gml::CodeType [0..*]
+species : gml::CodeType [0..1]
+height : gml::LengthType [0..1]
+trunkDiameter : gml::LengthType [0..1]
+crownDiameter : gml::LengthType [0..1]
<<Feature>>
PlantCover
+class : gml::CodeType [0..1]
+function : gml::CodeType [0..*]
+usage : gml::CodeType [0..*]
+averageHeight : gml::LengthType [0..1]
<<Geometry>>
gml::_Geometry
<<Object>>
core::ImplicitGeometry
<<Geometry>>
gml::MultiSurface
<<Geometry>>
gml::MultiSolid
lod1Geometry
lod2Geometry
lod3Geometry
lod4Geometry
lod1ImplicitRepresentation
lod2ImplicitRepresentation
lod3ImplicitRepresentation
lod4ImplicitRepresentation
lod1MultiSurface
lod2MultiSurface
lod3MultiSurface
lod4MultiSurface
lod1MultiSolid
lod2MultiSolid
lod3MultiSolid
lod4MultiSolid
0..1
*

Figure 79: UML diagram of class VegetationObject

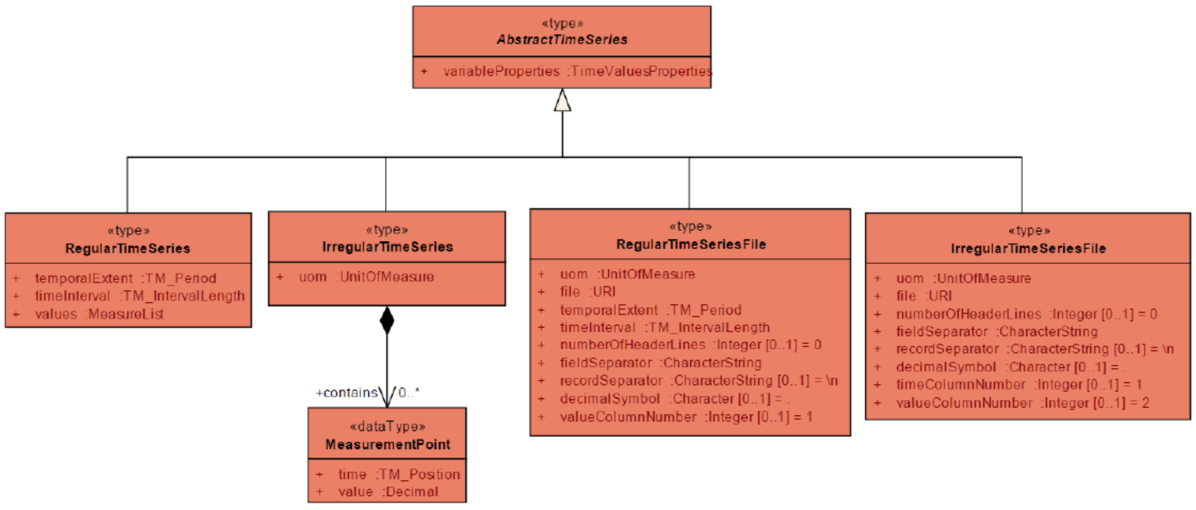

«type»
AbstractTimeSeries
+ variableProperties :TimeValuesProperties
«type»
RegularTimeSeries
+ temporalExtent :TM_Period
+ timeInterval :TM_IntervalLength
+ values :MeasureList
«type»
IrregularTimeSeries
+ uom :UnitOfMeasure
+contains
0..*
«dataType»
MeasurementPoint
+ time :TM_Position
+ value :Decimal
«type»
RegularTimeSeriesFile
+ uom :UnitOfMeasure
+ file :URI
+ temporalExtent :TM_Period
+ timeInterval :TM_IntervalLength
+ numberOfHeaderLines :Integer [0..1] = 0
+ fieldSeparator :CharacterString
+ recordSeparator :CharacterString [0..1] = \n
+ decimalSymbol :Character [0..1] = .
+ valueColumnNumber :Integer [0..1] = 1
«type»
IrregularTimeSeriesFile
+ uom :UnitOfMeasure
+ file :URI
+ numberOfHeaderLines :Integer [0..1] = 0
+ fieldSeparator :CharacterString
+ recordSeparator :CharacterString [0..1] = \n
+ decimalSymbol :Character [0..1] = .
+ timeColumnNumber :Integer [0..1] = 1
+ valueColumnNumber :Integer [0..1] = 2

Figure 80: UML diagram of class TimeSeries

Appendix 2: Queries

Phase 1

Reading Buildings

```
Select ST_Force2D(geometry) as b_geometry, building.id as id, ST_ZMin(geometry) as base_height,
measured_height as building_height
FROM building join surface_geometry on lod0_footprint_id = parent_id
WHERE geometry is not NULL AND ST_Intersects(geometry, ST_PolygonFromText('%s',%s))
GROUP BY building.id, geometry;"% (cutter, co_srid[0])
```

Reading Vegetation

```
Select ST_Force2D(lod1_implicit_ref_point) as v_geometry, height, crown_diameter
FROM solitary_vegetat_object
WHERE ST_Within(lod1_implicit_ref_point, ST_PolygonFromText('%s',%s))" % (cutter, co_srid[0])
```

Reading LandUse

```
SELECT geometry as l_geometry, usage, strval as surface_material
FROM surface_geometry join cityobject_genericattrib on cityobject_genericattrib.cityobject_id =
surface_geometry.cityobject_id join land_use on land_use.id = surface_geometry.cityobject_id
WHERE cityobject_genericattrib.attrname = 'material' AND geometry is not NULL AND
ST_Intersects(geometry, ST_PolygonFromText('%s',%s))
GROUP BY l_geometry, usage, surface_material;" % (cutter, co_srid[0])
```

Reading DEM

```
Select geometry as t_geometry
FROM surface_geometry
WHERE geometry is not NULL AND parent_id = 7265 AND ST_Intersects(geometry,
ST_PolygonFromText('%s',%s))" % (buff_cutter, co_srid[0])
```

Reading Z Values

Select ST_Zmin(ST_3DExtent(geometry)), ST_ZMax(ST_3DExtent(geometry)) from surface_geometry;

Reading coordinate system

Select srid FROM database_srs

Phase 2

Create index on point cloud attribute table(inxar) foreign key

CREATE INDEX geom_id_del33 ON sclgmodeler.inxar(point_id);

Filter ENVI-met values of occupied cells

WITH filtered as (Select

values, count(*)

FROM sclgmodeler.inxar

GROUP BY values

HAVING count(*) > 1)

, filterd as(Select

point_id

FROM sclgmodeler.inxar

WHERE values in (Select values FROM filtered))

DELETE FROM sclgmodeler.geom WHERE id in (SELECT point_id from filterd);

Rotate point cloud

UPDATE sclgmodeler.geom SET point = (Select ST_Rotate(point, %.10f, %.10f, %.10f));' %(-rotation, px, py)

Create spatial index on point cloud geometry table(geom)

CREATE INDEX point_sid33 ON sclgmodeler.geom USING gist(point)

Filter points that intersect with buildings

With delesion as (Select sclgmodeler.geom.id as the_id

FROM building join surface_geometry on lod0_footprint_id = parent_id, sclgmodeler.geom

```
WHERE ST_Intersects(point, geometry) AND ST_Z(point) > ST_ZMin(geometry) AND ST_Z(point) <
ST_ZMin(geometry) + measured_height)

DELETE FROM sclgmodeler.geom WHERE id IN (Select the_id FROM delesion);
```

Catch the first row of points over roofs

```
Create table sclgmodeler.surskit as(Select building.id as building_id, point_id, variable_name,
timestamp_begin, timestamp_end, step, unit, n_val, values

FROM building join surface_geometry on lod0_footprint_id = parent_id, sclgmodeler.geom join
sclgmodeler.inxar on sclgmodeler.inxar.point_id = sclgmodeler.geom.id

WHERE geometry is not NULL AND ST_Within(geometry, ST_Buffer(%s,-10)) AND ST_Intersects(point,
geometry) AND ST_Z(point) > ST_ZMin(geometry) + measured_height AND ST_Z(point) <
ST_ZMin(geometry) + measured_height + %d);" %(cutter, dz)
```

Add the points outside facades by creating buffer of linsestrings of footprint geometry

```
With dysk as(SELECT building_id, measured_height, base_height,
ST_Buffer(ST_MakeLine(sp,ep),%s,'side=right') as geomet

FROM(SELECT building_id, measured_height, base_height, ST_PointN(geom, generate_series(1,
ST_NPoints(geom)-1)) as sp, ST_PointN(geom, generate_series(2, ST_NPoints(geom)  )) as ep

FROM(SELECT building_id, measured_height, base_height, (ST_Dump(ST_Boundary(geom))).geom "

FROM(Select building.id as building_id, measured_height, ST_ZMin(geometry) as base_height, geometry
as geom

FROM building join surface_geometry on lod0_footprint_id = parent_id

WHERE geometry is not NULL AND ST_Within(geometry, ST_Buffer(%s,-10)))

As polygontable)

As linestrings)

As segments)

Insert into sclgmodeler.surskit(Select building_id, point_id, variable_name, timestamp_begin,
timestamp_end, step, unit, n_val, values "

FROM dysk, sclgmodeler.geom join sclgmodeler.inxar on sclgmodeler.inxar.point_id =
sclgmodeler.geom.id "
WHERE ST_Intersects(point, geomet) AND ST_Z(point) > base_height AND ST_Z(point) < base_height +
measured_height);" %(cutter, dx)
```

Aggregate point values on building level

```
Create table sclgmodeler.django_aggate as

With jango as (SELECT building_id, AVG(unnest::decimal(10,4)) AS avg

FROM sclgmodeler.surskit, unnest(values) with ordinality

GROUP BY building_id, ordinality

ORDER BY ordinality)

SELECT building_id, array_agg(avg::decimal(10,3)) as avg_values

FROM jango GROUP BY building_id;"
```

Create a table that is ready to imported as Time Series

```
Create table sclgmodeler.tonrg as

With djaaa as (Select sclgmodeler.django_aggate.building_id, variable_name, timestamp_begin,
timestamp_end, step, unit, n_val, avg_values

FROM sclgmodeler.django_aggate join sclgmodeler.surskit on sclgmodeler.django_aggate.building_id =
sclgmodeler.surskit.building_id)

Select building_id, variable_name, timestamp_begin, timestamp_end, step, unit, n_val, avg_values

FROM djaaa

GROUP BY variable_name, building_id, timestamp_begin, timestamp_end, step, unit, n_val, avg_values

ORDER BY variable_name, building_id;
```

Fill the Time Series table (view)

```
DELETE FROM citydb_view.nrg8_time_series_regular;

INSERT INTO citydb_view.nrg8_time_series_regular (id, acquisition_method, interpolation_type,
values_array, values_unit, array_length, temporal_extent_begin, temporal_extent_end, time_interval,
time_interval_unit)

SELECT building_id, 'Simulation', 'ConstantInSucceedingInterval', avg_values, 'Celsius Degrees', n_val,
timestamp_begin, timestamp_end, step, unit

FROM sclgmodeler.tonrg
```

Fill the Weather data table (view)

```
DELETE FROM citydb_view.nrg8_weather_data;


INSERT INTO citydb_view.nrg8_weather_data (type, time_series_id, cityobject_id)

SELECT variable_name, id, building_id

FROM citydb_view.nrg8_time_series_regular join sclgmodeler.tonrg on
citydb_view.nrg8_time_series_regular.id = sclgmodeler.tonrg.building_id
```

Appendix 3: ENVI-met outputs

<u>Atmosphere Folder</u>

Flow u (m/s)

Flow v (m/s)

Flow w (m/s)

Wind Speed (m/s)

Wind Speed Change (%)

Wind Direction (deg)

Pressure Perturbation (Diff)

Air Temperature (°C)

Air Temperature Delta (K)

Air Temperature Change (K/h)

Spec. Humidity (g/kg)

Relative Humidity (%)

TKE (m^2/m^3)

Dissipation (m^3/m^3)

Vertical Exchange Coef. Impuls (m^2/s)

Horizontal Exchange Coef. Impuls (m^2/s)

Vegetation LAD (m^2/m^3)

Direct Sw Radiation (W/m^2)

Diffuse Sw Radiation (W/m^2)

Reflected Sw Radiation (W/m^2)

Temperature Flux (K*m/s)

Vapour Flux (g/kg*m/s)

Water on Leafes (g/m^2)

Leaf Temperature (°C)

Local Mixing Length (m)

Mean Radiant Temp. (°C)

TKE normalised 1D ( )

Dissipation normalised 1D ( )

Km normalised 1D ( )

TKE Mechanical Turbulence Prod. ( )

Stomata Resistance (s/m)

CO2 (mg/m3)

CO2 (ppm)

Plant CO2 Flux (mg/m^{2}s)

Div Rlw Temp change (K/h)

Building Number ()

Static Building Folder

Albedo of facade ()

Transmission of facade ()

Emissivity of facade ()

View factor sky ()

View factor soil ()

View factor bldg ()

View factor veg ()

Building number ()

Facade/ Roof Greening LAI (m^3/m^3)

Facade/ Roof Greening Thickness (cm)

Dynamic Building Folder

Wall shading flag ()

Wall: Temperature Node 1/ outside (°C)

Wall: Temperature Node 2 (°C)

Wall: Temperature Node 3 (°C)

Wall: Temperature Node 4 (°C)

Wall: Temperature Node 5 (°C)

Wall: Temperature Node 6 (°C)

Wall: Temperature Node 7/ inside (°C)

Building: Sum Humidity Flux at facade (g/s*m3)

Wall: Longwave radiation emitted by facade (W/m2)

Wall: Wind Speed in front of facade (m/s)

Wall: Air Temperature in front of facade (°C)

Wall: Shortwave radiation received at facade (W/m2)

Wall: Absorbed direct shortwave radiation (W/m2)

Wall: Incoming longwave radiation (W/m2)

Wall: Reflected shortwave radiation facade (W/m2)

Wall: Sensible Heat transmission coefficient outside (W/m2K)

Wall: Longwave Energy Balance (W/m2)

N.N.,Building: Temperature of building (inside) (°C)

Building: Reflected shortwave radiation (W/m2)

Building: Longwave radiation emitted (W/m2)

Greening: Temperature Leafs (°C)

Greening: Air Temperature Canopy (°C)

Greening: Air Humidity Canopy (g/kg)

Greening: Longwave radiation emitted (two-side) (W/m2)

Greening: Wind Speed in front of greening (m/s)

Greening: Air Temperature in front of greening (°C)

Greening: Shortwave radiation received at greening (W/m2)

Greening: Incoming longwave radiation (two-side) (W/m2)

Greening: Reflected shortwave radiation (W/m2)

Greening: Transpiration Flux (g/s*m3)

Greening: Stomata Resistance (s/m),Greening: Water access factor ()

Substrate: Temperature Node 1/ outside (°C)

Substrate: Temperature Node 2 (°C)

Substrate: Temperature Node 3 (°C)

Substrate: Temperature Node 4 (°C)

Substrate: Temperature Node 5 (°C)

Substrate: Temperature Node 6 (°C)

Substrate: Temperature Node 7/ inside (°C)

Substrate: Surface humidity (g/kg)

Substrate: Humidity Flux at substrate (g/s*m3)

Substrate: Longwave radiation emitted by substrate (W/m2)

Substrate: Wind Speed in front of substrate (m/s)

Substrate: Air Temperature in front of substrate (°C)

Substrate: Shortwave radiation received at substrate (W/m2)

Substrate: Absorbed direct shortwave radiation (W/m2)

Substrate: Incoming longwave radiation (W/m2)

Substrate: Reflected shortwave radiation substrate (W/m2)

Dynamic Vegetation Folder

Objects ( )

Plant Index ( )

Plant Type ID ()

Flow u (m/s)

Flow v (m/s)

Flow w (m/s)

Wind Speed at Vegetation (m/s)

Wind Speed Change at Vegetation (%)

Wind Direction at Vegetation (deg)

Local Drag at Vegetation (N/m^3)

Horizontal Drag at Vegetation (N)

Horizontal Drag at Vegetation Vertical Sum (N)

Air Temperature at Vegetation (°C)

Spec. Humidity at Vegetation (g/kg)

Relative Humidity at Vegetation (%)

TKE at Vegetation (m^2/m^3)

Vegetation LAD (m^2/m^3)

Direct Sw Radiation (W/m^2)

Diffuse Sw Radiation (W/m^2)

Reflected Sw Radiation (W/m^2)

Mean Radiant Temp. (°C)

Temperature Flux (K*m/s)

Vapour Flux (g/kg*m/s)

Water on Leafes (g/m^2)

Leaf Temperature (°C)

Aerodynamic Resistance (s/m)

Stomata Resistance (s/m)

Plant CO2 Flux (mg/kg*m/s)

Plant Isoprene Flux (mg/cell*h)

PAR (micro mol m-2 s-1)

Appendix 4: Scenario visualizations

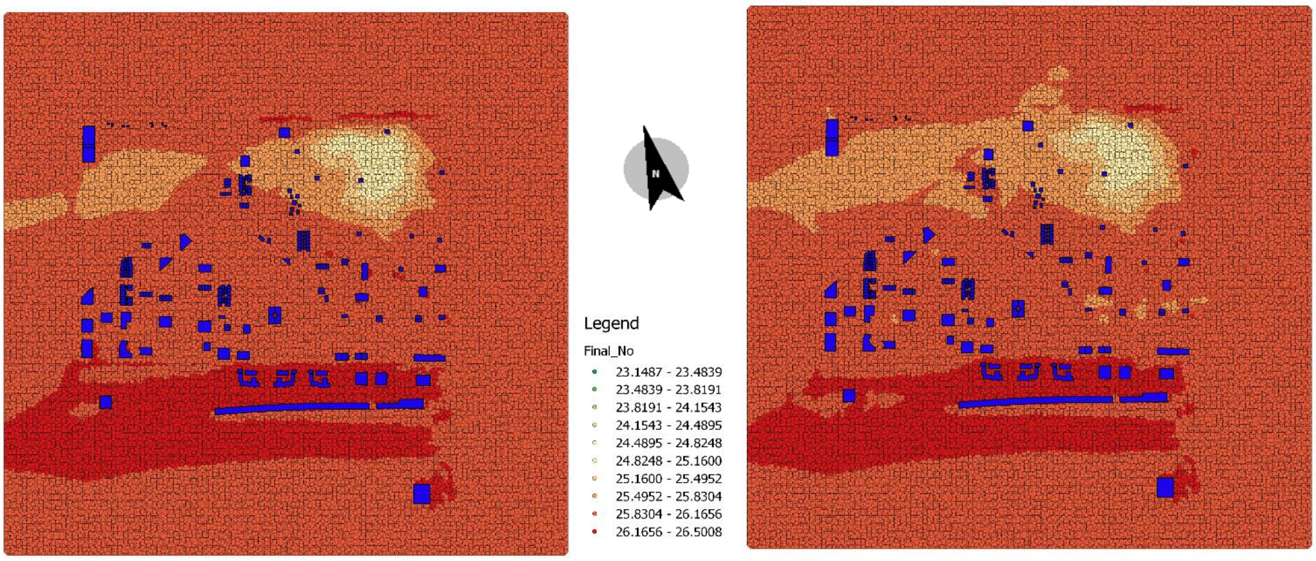


Figure 81: QGIS visualizations (following the terrain at Z = 1.5m, timestamp = 2018-06-25 08:00:01), left : scenario without arboretum trees, right : scenario with arboretum trees. Weather input parameters : Temperature (C) = 26.05, Rel Humidity (%) = 61, Wind Speed(m/sec) = 2, Wind Direction (Degrees) = 90

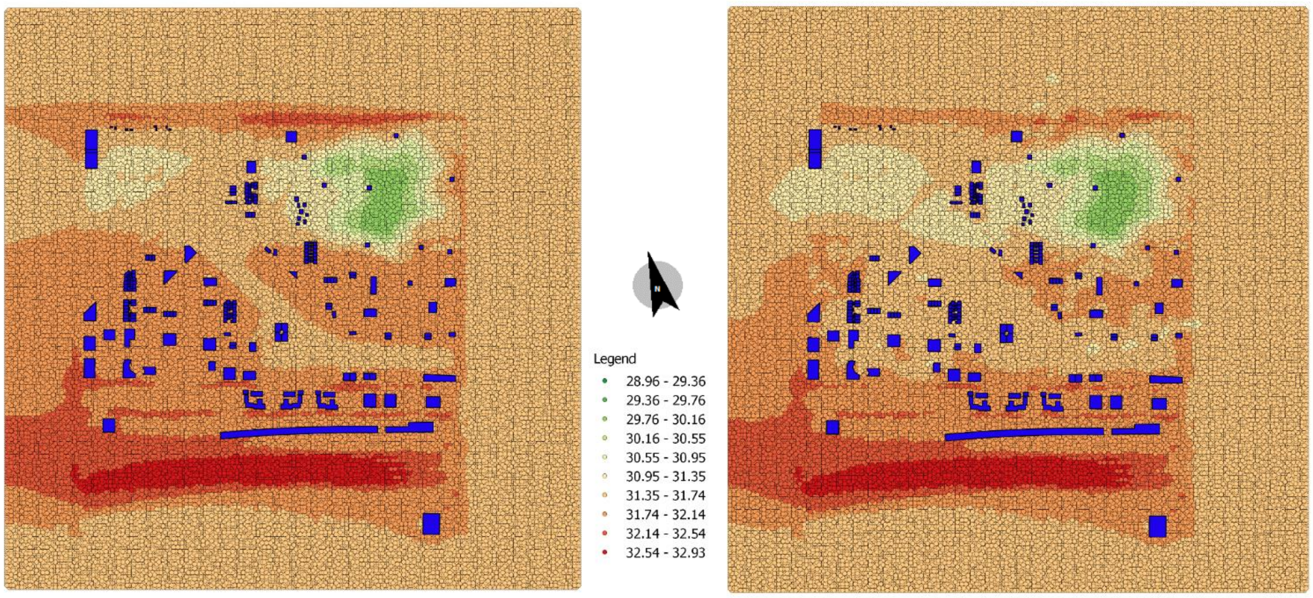


Figure 82: 83QGIS visualizations (following the terrain at Z = 1.5m, timestamp = 2018-06-25 12:00:01), left : scenario without arboretum trees, right : scenario with arboretum trees. Weather input parameters : Temperature (C) = 31.45, Rel Humidity (%) = 48, Wind Speed (m/sec) = 3, Wind Direction (Degrees) = 110

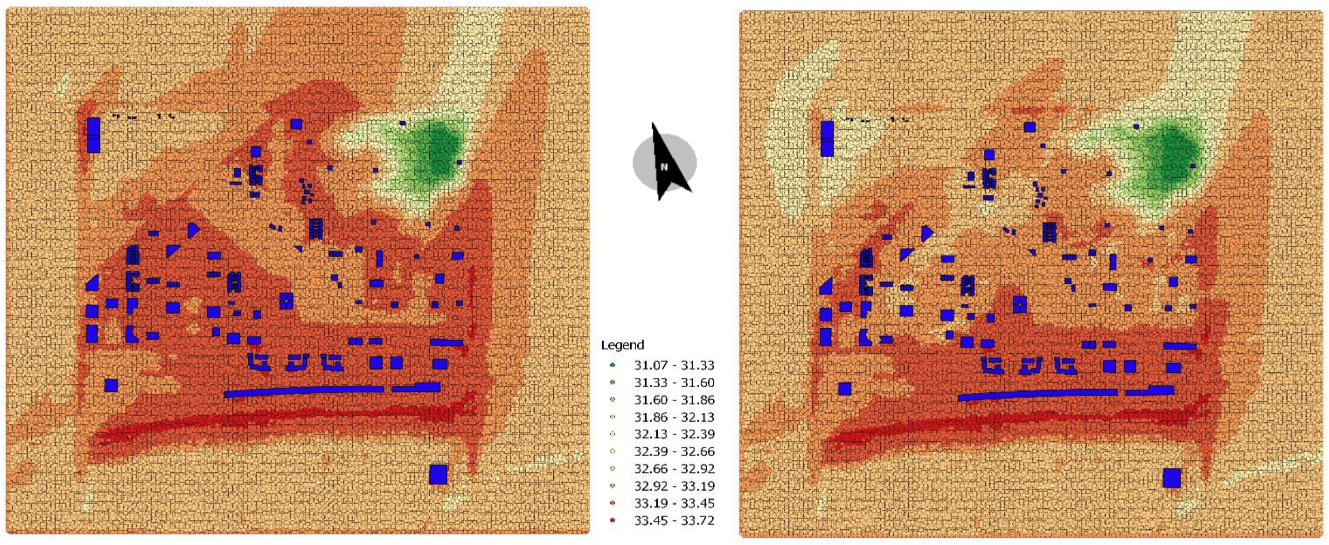


Figure 84: QGIS visualizations (following the terrain at Z = 1.5m, timestamp = 2018-06-25 12:00:01), left : scenario without arboretum trees, right : scenario with arboretum trees. Weather input parameters : Temperature (C) = 32.95, Rel Humidity (%) = 46, Wind Speed(m/sec) = 5, Wind Direction (Degrees) = 210

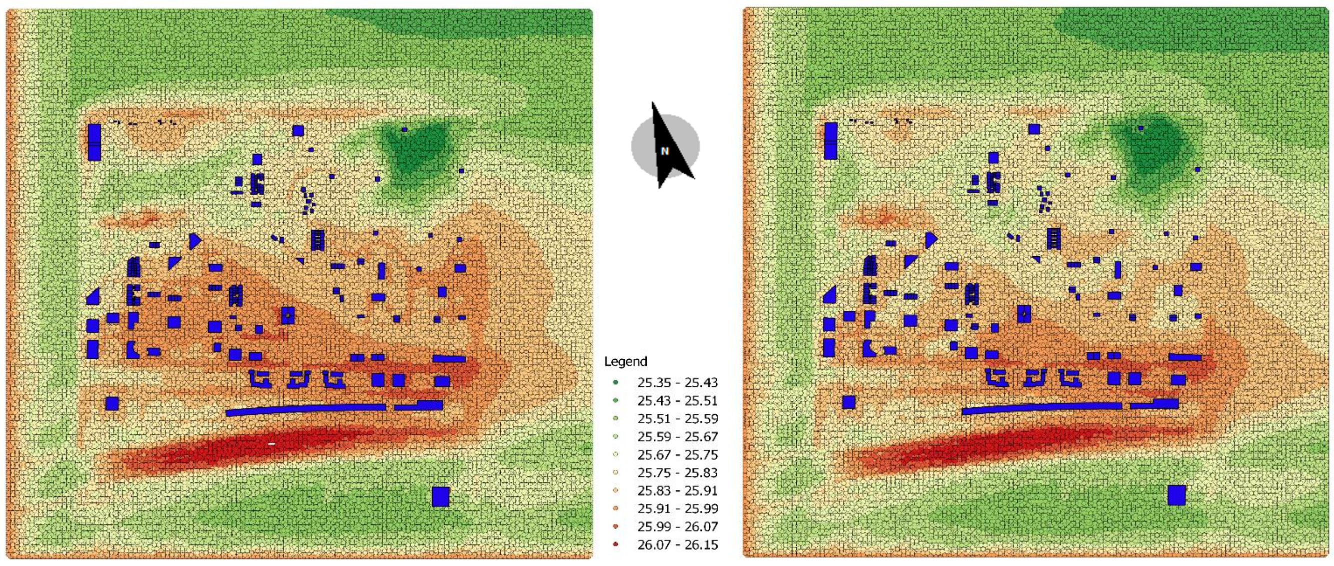


Figure 85: QGIS visualizations (following the terrain at Z = 1.5m, timestamp = 2018-06-25 12:00:01), left : scenario without arboretum trees, right : scenario with arboretum trees. Weather input parameters : Temperature (C) = 25.95, Rel Humidity (%) = 71, Wind Speed (m/sec) = 3, Wind Direction (Degrees) = 270

References

Agugiaro, G. (2016). FIRST STEPS TOWARDS AN INTEGRATED CITYGML-BASED 3D MODEL of VIENNA. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *3*(July), 139–146. https://doi.org/10.5194/isprs-annals-III-4-139-2016

Agugiaro, Giorgio, Benner, J., Cipriano, P., & Nouvel, R. (2018). The Energy Application Domain Extension for CityGML: enhancing interoperability for urban energy simulations. *Open Geospatial Data, Software and Standards*, *3*(1), 2. https://doi.org/10.1186/s40965-018-0042-y

Agugiaro, Giorgio, Hauer, S., & Nadler, F. (2015). Coupling of CityGML-based Semantic City Models with Energy Simulation Tools: some Experiences, *2*(May), 191–200.

Agugiaro, Giorgio, & Kumar, K. (2018). CityGML Energy ADE, (December).

Allegrini, J., Orehounig, K., Mavromatidis, G., Ruesch, F., Dorer, V., & Evins, R. (2015). A review of modelling approaches and tools for the simulation of district-scale energy systems. *Renewable and Sustainable Energy Reviews*, *52*, 1391–1404. https://doi.org/10.1016/j.rser.2015.07.123

Benner, J., Geiger, A., & Häfele Ing Joachim Benner, K.-H. (2016). Virtual 3D City Model Support for Energy Demand Simulations on City Level-The CityGML Energy Extension. *REAL CORP 2016. 21st International Conference on Urban Planning,Regional Development and Information Society*, *2*(June), 777–786. Retrieved from http://www.corp.at

Biljecki, F., Ledoux, H., Du, X., Stoter, J., Soon, K. H., & Khoo, V. H. S. (2016). THE MOST COMMON GEOMETRIC and SEMANTIC ERRORS in CITYGML DATASETS. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *4*(2W1), 13–22. https://doi.org/10.5194/isprs-annals-IV-2-W1-13-2016

Biljecki, F., Zhao, J., Stoter, J., & Ledoux, H. (2013). Revisiting the concept of level of detail in 3D city modelling. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, *2*(2W1), 63–74. https://doi.org/10.5194/isprsannals-II-2-W1-63-2013

Biljecki, Filip, Kumar, K., & Nagel, C. (2018). CityGML Application Domain Extension (ADE): overview of developments. *Open Geospatial Data, Software and Standards*, *3*(1), 13. https://doi.org/10.1186/s40965-018-0055-6

Billen, R., Cutting-Decelle, A.-F., Marina, O., de Almeida, J.-P., M., C., Falquet, G., … Zlatanova, S. (2014). 3D City Models and urban information: Current issues and perspectives. *3D City Models and Urban Information: Current Issues and Perspectives – European COST Action TU0801*, I–118. https://doi.org/10.1051/TU0801/201400001

Chatzinikolaou, E., Chalkias, C., & Dimopoulou, E. (2018). URBAN MICROCLIMATE IMPROVEMENT USING ENVI-MET CLIMATE MODEL, *XLII*(October), 1–5.

Coros, S. (n.d.). Constructive Solid Geometry and Procedural Modeling.

Donny Koerniawan Weijun Gao, M., ムハマド ドニー クルニアワン正会員, & 高 偉俊同. (2015). The Simulation Study of Thermal Comfort in Urban Open Spaces of Commercial Area Using EnviMet Software, (September).

Eriksson, H., Harrie, L., & Paasch, J. M. (2018). What is the need for building parts? - A comparison of CityGML, inspire building and a swedish building standard. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, *42*(4/W10), 27–32. https://doi.org/10.5194/isprs-archives-XLII-4-W10-27-2018

Gaitani, N., Spanou, A., Saliari, M., Synnefa, A., Vassilakopoulou, K., Papadopoulou, K., … Lagoudaki, A. (2011). Improving the microclimate in urban areas: a case study in the centre of Athens. *Building Services Engineering Research and Technology*, *32*(1), 53–71. https://doi.org/10.1177/0143624410394518

Gartner, G., Meng, L., & Peterson, M. P. (2009). *3D Geo-Information Sciences*. *Lecture Notes in Geoinformation and Cartography*.

Gobakis, K., & Kolokotsa, D. (2017). Coupling building energy simulation software with microclimatic simulation for the evaluation of the impact of urban outdoor conditions on the energy consumption and indoor environmental quality. *Energy and Buildings*, *157*, 101–115. https://doi.org/10.1016/j.enbuild.2017.02.020

Gröger, G., Kolbe, T. H., Nagel, C., & Häfele, K. (2012). OpenGIS City Geography Markup Language (CityGML) Encoding Standard. Version 2.0.0, Open Geospatial Consortium, OGC Doc. No. 12-019. https://doi.org/OGC 12-019

Huttner, S. (2012). Further development and application of the 3D microclimate simulation ENVI-met. *Mainz: Johannes Gutenberg-Universitat in Mainz*, 147. Retrieved from http://ubm.opus.hbz-nrw.de/volltexte/2012/3112/

Input, M. E. (2015). 2015 CHINA SUSTAINABLE CITIES REPORT.

Kardinal Jusuf, S., Mousseau, B., Godfroid, G., & Soh Jin Hui, V. (2017). Integrated modeling of CityGML and IFC for city/neighborhood development for urban microclimates analysis. *Energy Procedia*, *122*, 145–150. https://doi.org/10.1016/j.egypro.2017.07.329

Kumar, K., Ledoux, H., & Stoter, J. (2016). A CITYGML EXTENSION for HANDLING VERY LARGE TINS. *ISPRS*

Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, *4*(2W1), 137–143. https://doi.org/10.5194/isprs-annals-IV-2-W1-137-2016

Leao, S., Padgham, L., Perez, P., Nagel, K., & Bazzan, A. (2017). Mohammad-Reza Namazi-Rad Agent Based Modelling of Urban Systems, (February). https://doi.org/10.1007/978-3-319-51957-9

Nouvel, R., Kaden, R., Bahu, J., Kämpf, J. H., Cipriano, P., Lauster, M., … Casper, E. (2015). Genesis of the CityGML Energy ADE. *CISBAT 2015, International Conference on Future Buildings & Districts– Sustainability from Nano to Urban Scale*, 931–936. https://doi.org/10.5075/epfl-cisbat2015-931-936

Reddy, V. S. (2016). Sustainable Construction: Analysis of Its Costs and Financial Benefits. *International Journal of Innovative Research in Engineering & Management*, *3*(6), 522–525. https://doi.org/10.21276/ijirem.2016.3.6.12

Shukla, D., Shivnami, C., & Shah, D. (2016). Comparing Oracle Spatial and Postgres PostGIS. *Computer Science & Electronic Journal*, *7*, 95–100. https://doi.org/10.090592/IJCSC.2016.116

Simon, H., Kropp, T., Sohni, F., & Bruse, M. (2018). Downscaling Climate Models : Running Nested Simulations In The Microclimate Model ENVI-met A Case Study Using WUDAPT2ENVI-met Simulation Data. *Plea 2018*, (December), 6.

Simon, Helge, & Bruse, M. (2015). SIMULATION OF INDOOR CLIMATE WITH FAÇADE DYNAMICS & BUILDING – ATMOSPHERE INTERACTION, (July 2017).

Stadler, A., & Kolbe, T. H. (2007). Spatio-Semantic Coherence in the Integration of 3D City Models. *Proceedings of the 5th International ISPRS Symposium on Spatial Data Quality*, (June), 13–15. Retrieved from http://www.isprs.org/proceedings/XXXVI/2-C43/Session1/paper_Stadler.pdf

Yang, X., Zhao, L., Bruse, M., & Meng, Q. (2012). An integrated simulation method for building energy performance assessment in urban environments. *Energy and Buildings*, *54*, 243–251. https://doi.org/10.1016/j.enbuild.2012.07.042

https://www.afd.fr/en/planet-overheating-lets-take-action-now

https://www.ft.com/content/9d457d54-b272-11e8-8d14-6f049d06439c

https://www.grasshopper3d.com/group/gismo/forum/topics/gismo-ladybug-envi-met-connection

https://www.envi-met.com

https://www.geospatialworld.net