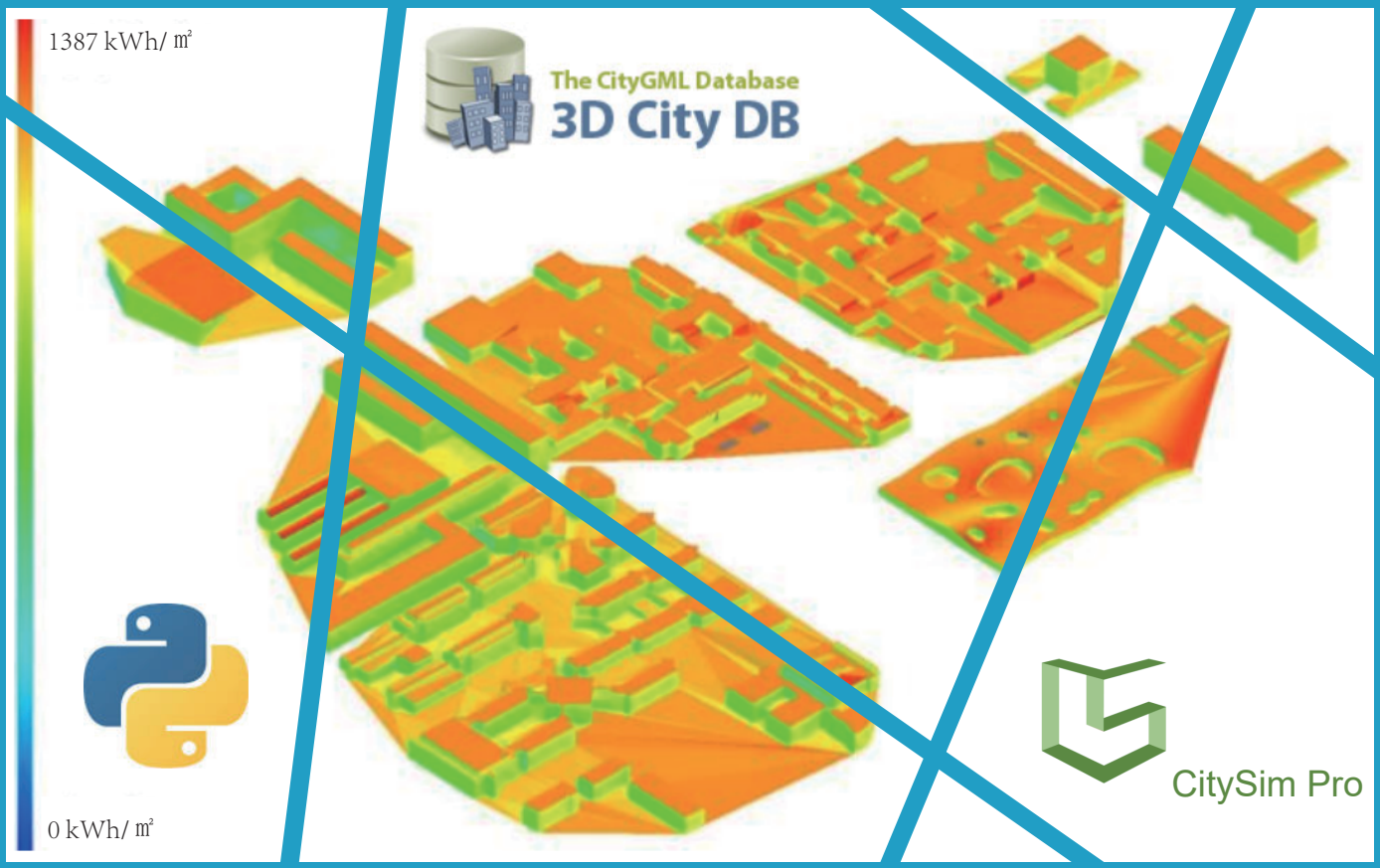
MSc thesis in Geomatics

# Dynamic energy simulations based on the 3D BAG 2.0

Yuzhen Jin

2022




TU Delft

**MSc thesis in Geomatics**

# Dynamic energy simulations based on the 3D BAG 2.0

Yuzhen Jin

June 2022

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of Master of Science in Geomatics

The work in this thesis was carried out in the:


3D geoinformation group
Delft University of Technology

| | |
|---|---|
| Supervisors: | Giorgio Agugiaro |
| | Camilo Alexander Leon Sánchez |
| External Supervisors: | Jérôme Kämpf (Idiap) |
| | Giuseppe Peronato (Idiap) |
| Co-reader: | Azarakhsh Rafiee Voermans |

# Abstract

Issues such as climate change, ecological conservation and sustainable energy have received a great deal of attention in the last decade. Studies have shown that cities are responsible for major energy use and waste emissions. In dealing with the growing environmental problems, people have to look to the cities they live in. Today, urbanisation is still accelerating worldwide which heralds a potentially huge opportunity to improve the environment by increasing the energy sustainability of cities.

To address the energy sustainability of cities, policymakers and urban planners are looking for ways to control energy consumption in buildings. Faced with a large number of urban buildings and complex climate factors, the measurement of building energy consumption has to be done with the help of relevant simulation software. Luckily, software and data formats associated with the calculation of building energy consumption have matured over the years through the efforts of academics and research institutions. This largely helps to solve the complex problem mentioned above. However, these elements still need to be optimized and improved. In this thesis, the research will focus on one of the urban energy simulation software CitySim, the 3D city database 3DCityDB and the 3D city model data format CityGML. Although it is currently possible to rely on these elements for urban energy simulation, the whole simulation process is complex. The main reason for this problem is the number of data extractions, data conversions and data storage required throughout the whole process. Also, the lack of proficiency in data formats, software usage and data storage can be a difficult problem for potential users. Therefore, this research will focus on developing a python-based interface to achieve the goal of well connecting the entire urban energy simulation process.

This approach simplifies the process of urban energy simulation from the preparation of a complete database related to urban energy simulation, to the full process of data extraction, conversion and simulation in python, to the final storage of simulation results back to the database. In addition to this, several user-friendly customised operations are also developed in python. In this way, I hope to help more users to conduct urban energy simulation analysis conveniently.

# Acknowledgements

I would like to thank all the people who have helped me in completing my research.

First of all, a big thank to my first supervisor Giorgio Agugiaro and my second supervisor Camilo Alexander Leon Sánchez, who helped me from the beginning to the end of the research exploration. They provided me with insightful advice, feedback and guidance at every stage of the project's progress. In our weekly meetings, we discuss the direction and process of the research together, and this effective communication has helped me to complete my research step by step. They were always with me when I was in trouble and gave me practical advice. In addition to this, they would give concrete feedback on the milestones of the thesis which help me make a step further. With their knowledge and help, I was able to stay on the right track and eventually complete my research successfully.

Secondly I would like to thank the research collaborators Jérôme Kämpf and Giuseppe Peronato from the Idiap Research Institute in Switzerland. During our many interactions, they provided crucial assistance on the content and direction of the research and the technical issues related to CitySim. Without their technical support, this thesis would not have been successfully completed. They were always patient and specific in their explanations to my questions, which helped me to learn CitySim in depth and thus implement my research ideas.

Finally, I would like to thank my family who have supported and helped me throughout my two years of master study.

In the past two years, life has not been as exciting as it used to be due to the pandemic. But thanks to all the people who were there for me.

# Contents


**1. INTRODUCTION** **1**
- 1.1. Research Question . . . 2
- 1.2. Thesis Outline . . . 3

**2. THEORY BACKGROUND AND RELATED WORK** **5**
- 2.1. Urban Energy Simulation Data Model . . . 5
  - 2.1.1. CityGML . . . 5
  - 2.1.2. CityGML Energy ADE Extension . . . 7
- 2.2. Urban Energy Simulation Tools . . . 7
  - 2.2.1. CitySim . . . 8
  - 2.2.2. CitySim data model and CityGML with Energy ADE data model . . . 8
  - 2.2.3. 3D city database . . . 11
  - 2.2.4. Weather data . . . 12
  - 2.2.5. Horizon data . . . 13
- 2.3. Case Study . . . 13
  - 2.3.1. CitySim simulation: the case study of alt-wiedikon a neighbourhood of Zürich city . . . 13
  - 2.3.2. Machine learning for the energy of buildings on a GIS tool design project 2021 . . . 14

**3. METHODOLOGY** **17**
- 3.1. Research Approach . . . 17
  - 3.1.1. Data inspection . . . 18
  - 3.1.2. Management . . . 18
  - 3.1.3. Data extraction . . . 19
  - 3.1.4. Spatial And Non-spatial Data Pre-processing Functions . . . 19
  - 3.1.5. Mapping . . . 19
  - 3.1.6. Simulation and result storage . . . 20
- 3.2. Management . . . 20
- 3.3. Spatial and non-spatial data pre-processing functions . . . 20
  - 3.3.1. Study area selection . . . 21
  - 3.3.2. Terrain processing . . . 21
  - 3.3.3. Building surface processing . . . 23

**4. DATA PREPARATION** **25**
- 4.1. Data Overview . . . 25
- 4.2. Geometry Data . . . 26
- 4.3. physics parameters Libraries . . . 26
  - 4.3.1. Building physics . . . 26
  - 4.3.2. Additional physics parameters . . . 28
- 4.4. Weather Data . . . 28

**5. PYTHON IMPLEMENTATION** **33**
5.1. Data Extraction And Pre-processing . . . 33
5.1.1. Connect to the database . . . 33
5.1.2. Geometry data extraction . . . 34
5.1.3. Building attributes extraction . . . 35
5.1.4. Physics parameters extraction . . . 36
5.2. Shading Surfaces . . . 36
5.3. Terrain Processing . . . 37
5.4. Party Wall Processing . . . 40
5.5. Building Surface Processing . . . 41
5.6. Write CitySim Input XML File . . . 41
5.6.1. Write the composite part . . . 42
5.6.2. Write the profile part . . . 42
5.6.3. Write the building part . . . 43
5.6.4. Write the shading surface part . . . 44
5.6.5. Write the tree part . . . 44
5.6.6. Write the terrain part . . . 45
5.7. CitySim Simulation And Result Storage . . . 46
5.7.1. Call CitySim and run the simulation . . . 46
5.7.2. Simulation result storage . . . 47

**6. RESULT ANALYSIS, REFLECTION AND FUTURE WORK** **49**
6.1. Testing The Python Interface . . . 49
6.2. Guidelines For The Python Interface . . . 50
6.3. Reflection And Future Work . . . 50

**A. PHYSICS LIBRARY DATA SHEET IN THE DATABASE** **53**

**B. WEATHER LIBRARY DATA SHEET IN THE DATABASE** **57**

# List of Figures


1.1. The current urban energy simulation process with CitySim and 3DCityDB . . . 2
1.2. The proposed urban energy simulation process with CitySim and 3DCityDB . . 2

2.1. Thematic modules and associated feature types including the LoDs in which they are defined [OGC, 2010]. . . . 6
2.2. The color-coded modular structure of the Energy ADE UML diagram [Agugiaro et al., 2018] . . . 7
2.3. CitySim workflow . . . 8
2.4. The idealized data structure of CitySim (a), and CityGML Energy ADE (b) [Coccolo and Kämpf, 2015] . . . 9
2.5. Example of mapping an inheritance hierarchy onto one table [Yao et al., 2018] . 11
2.6. A glimpse of 3DCityDB table *surface_geometry* with data of alderaan.gml . . . . 12
2.7. The intended Energy ADE table *ng_timeseries* for simulation result storage . . . 12
2.8. Schema of the designed CitySim database [Perez et al., 2011] . . . 14
2.9. Scheme of complete tasks for data collection and organization (Sélène Ledain and Maxwell Bergström, 2021) . . . 15
2.10. Urban energy simulation workflow (Sélène Ledain and Maxwell Bergström, 2021) . . . 15

3.1. Methodology of the research . . . 18
3.2. Data enrichment workflow . . . 19
3.3. Data storage structure design methodology . . . 20
3.4. Workflow of the study area selection . . . 21
3.5. The difference of input geometry and simulated geometry . . . 22
3.6. Workflow of terrain processing . . . 22
3.7. Building surface will be simulated or not . . . 23
3.8. Re-triangulation of concave surfaces . . . 23
3.9. Workflow of building surface projection . . . 24

4.1. 3D view of alderaan.gml shown in FZKViewer . . . 27
4.2. The component structure of TABULA.xml . . . 27
4.3. Parse TABULA.xml into tables to be stored in database . . . 28
4.4. The links between building geometry and building physics . . . 29
4.5. The overview of parsing and storing other physics parameters . . . 29
4.6. The .cli file generated from EPW file in python . . . 30
4.7. Parse EPW files into tables to be stored in database . . . 31
4.8. The complete database for python interface development . . . 31

5.1. The overview of data extraction and pre-processing . . . 33
5.2. Example of establishing a connection between the database and the python environment . . . 34

5.3. Geometries shown in CitySim GUI; Left one: the original geometry bounding box as xmin 0 ymin -30 xmax 70 ymax 15; Right one: the selected geometry bounding box as xmin 0 ymin -20 xmax 70 ymax 15, buffer as 20m . . . . . 34
5.4. The building geometry dataframe . . . . . 35
5.5. The building list dataframe . . . . . 36
5.6. The final building list and building physics parametersframe . . . . . 37
5.7. The overview of shading surface processing . . . . . 37
5.8. The processing result shown in CitySim GUI, bounding box as xmin 0 ymin -30 xmax 70 ymax 15, buffer 50m . . . . . 38
5.9. The processed terrain shown in CitySim GUI . . . . . 39
5.10. The processed terrain with the rest geometries shown in CitySim GUI . . . . . 39
5.11. The party walls' relationship stored in table *generalization* . . . . . 40
5.12. The alderaan.gml party wall processing result shwon in FZKViewer . . . . . 40
5.13. The aldraan.gml building surface processing outcome shown in CitySim GUI . 41
5.14. The dataframe of composite information . . . . . 42
5.15. The dataframe of occupancy profile information . . . . . 43
5.16. Running the simulation in CitySim Pro GUI . . . . . 46
5.17. CitySim simulation result TH.tsv in Excel window . . . . . 47
5.18. Part of the UML diagram of the Energy ADE Core . . . . . 47
5.19. Table *ng_regulartimeseries* in database contains the Alderaan simulation result . 48

6.1. The CitySim simulation result of short wave irradiance of RijssenHolten.gml viewed in CitySim Pro GUI . . . . . 49
6.2. Table *ng_regulartimeseries* in database contains the RijssenHolten simulation result 50

# List of Tables

2.1. CitySim input climate file elements . . . . . 13

4.1. Data preparation overview . . . . . 25
4.2. Attriibutes of Alderaan.gml that are important for CitySim . . . . . 26
4.3. The mapping between EPW file and .cli file . . . . . 30

A.1. The physics library data sheet in the database . . . . . 56

B.1. The weather library data sheet in the database . . . . . 58

# List of Algorithms

# Acronyms

**ADE** Application Domain Extensions . . . . . 7
**3DCityDB** 3D City Database . . . . . 1
**XML** Extensible Markup Language . . . . . 6
**GML** Geography Markup Language . . . . . 6
**UML** Unified Modeling Language . . . . . 47
**SQL** Structured Query Language . . . . . 14
**RDBMS** Relational Database Management System . . . . . 11
**LoD** Level of Detail . . . . . 1
**EPW** EnergyPlus Weather Format . . . . . 28
**TIN** triangular irregular network . . . . . 21

# 1. INTRODUCTION

The world's urbanization is accelerating. According to the United Nations [Nations et al., 2018], as of 2018, fifty-five percent of the world population lives in urban areas, and by mid-century, this percentage will expand to roughly two-thirds (sixty-eight percent). More and more people are living in cities, which poses a greater challenge to urban-related environmental issues. In fact, nowadays Cities count for more than seventy-five percent of primary energy use and more than eight percent of greenhouse gas emissions, which shows a tremendous opportunity for urban buildings to enhance the energy sustainability of cities [Nutkiewicz et al., 2018].

As a result, urban planners and policymakers are facing challenges in controlling the amount of shelters growth and their associated energy consumption [Coccolo and Kämpf, 2015]. Faced with a large number of urban buildings and complex climate factors, the measurement of building energy consumption has to be done with the help of relevant simulation software. Fortunately, in order to support energy transition processes at the urban scale, there has been a continuous development of Urban Energy Modelling in the last decade: many actors (e.g. international research centers, private sector, universities) have been developing relevant algorithms and software to provide new digital methods for energy planning and decision support [Agugiaro et al., 2018]. Those are powerful tools for calculating urban energy consumption which largely helps to solve the complex problem mentioned above. However, these elements still need to be optimized and improved. This project will focus on one of the topics of urban energy simulation. To start with, the current urban energy simulation process needs to be introduced.

Recently, the 3D geoinformation at TU Delft released the 3D BAG 2.0, a dataset containing the 3D representation of buildings in several Level of Detail (LoD) of the whole Netherlands. This dataset can be used for applications that are related to building energy use, such as wind environment simulation, noise pollution measurement, etc. With complete urban data and constantly updated maintenance, this dataset is an ideal data source for urban energy simulations. To better manage large city model from 3D BAG 2.0, the manipulation is done by the open source 3D City Database (3DCityDB) which is able to store, represent, and manage virtual 3D city models on top of a standard spatial relational database.

CitySim is one of the urban energy simulation tools, which allows quantifying the heating and cooling energy demand at the single building or urban scale with specific input physics and geometry data [Robinson, 2012]. with the peculiar characteristic of considering the simulation scene as an urban environment whose individual elements interact with each other, CitySim is the ideal software for conducting urban energy simulations. It is important to mention that CitySim has its own unique input and output data format.

The current urban energy simulation process related to the above elements is cumbersome: first, the building data (from 3D BAG 2.0) are extracted from the 3DCityDB. Then those data are transformed into CitySim format as the input file for simulation. After loading into the CitySim Pro software, users need to set the simulation parameters before running the simulation. When the simulation is complete, the result files are not stored back to the 3DCityDB

due to the different data formats. The lack of direct connection between 3DCityDB and CitySim results in relatively complex operations for data retrieval, transformation, and storage. The current process is shown in Figure 1.1.

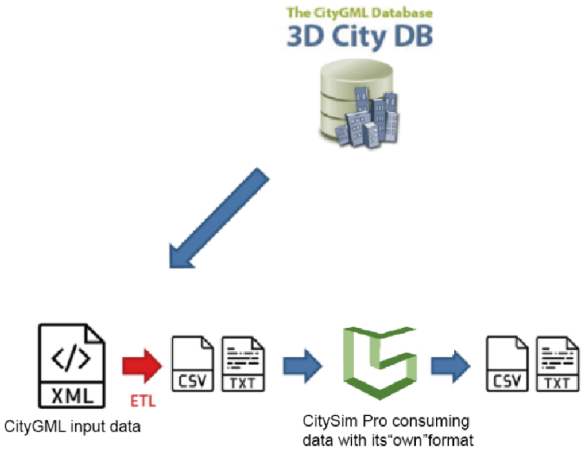



Figure 1.1.: The current urban energy simulation process with CitySim and 3DCityDB

To address the issue mentioned above, the aim of this thesis is to link the 3DCityDB and CitySim in order to allow for a seamless flow of information and perform energy simulations in CitySim based on 3D BAG 2.0. More specifically, the thesis will focus on developing a Python-based bidirectional interface to feed and retrieve data between the 3DCityDB and CitySim (see Figure 1.2).


Figure 1.2.: The proposed urban energy simulation process with CitySim and 3DCityDB

## 1.1. Research Question

The main research question of this project is defined as follow:

- *To what extent is it possible to link the 3DCityDB and CitySim, for a seamless flow of information between the two platforms to perform urban energy simulations?*

To achieve the goal of this project, several relevant sub-questions need to be answered:

- *What information is needed for CitySim urban energy simulation?*

- *Where and how to collect and store the data needed for CitySim simulation?*
- *What is the difference of data formats between CityGML and CitySim? What are the mapping rules between those two?*
- *How to extract information from the 3DCityDB and transformed into the CitySim input file?*
- *To clearly structure the overall data transformation, how to enrich the CitySim input file step by step from basic information to complex information (e.g. from basic LoD2 geometries to geometries with enriched energy simulation-related information)?*
- *How to define and design the database schema in 3DCityDB for storing libraries of energy simulation-related parameters and profiles?*
- *After the simulation, which results need to be stored back into the database, and how to store them back? Should certain data be aggregated?*
- *Does CitySim mechanism require pre-processing of geometry? How to handle this in Python environment?*
- *Whether it is possible to design functions to fulfill the different needs of users (e.g. interest area selection, counting the number of buildings)?*

## 1.2. Thesis Outline

This thesis is organised as follows:

- Chapter 2 introduces the concepts of urban energy simulation data model and tools. Two case studies related to CitySim urban energy simulation will also be presented.
- Chapter 3 describes the research methodology of this project. An overall research methodology covering the whole process will be introduced first. Then, detail discussion will address on the data management and spatial and non-spatial data pre-processing functions.
- Chapter 4 introduces the detail work of data preparation. Starting with an overview of what data is needed for CitySim urban energy simulation to detail work of collecting and storage relates to geometry data, physics parameters and weather data.
- Chapter 5 describes each step of the development of the python-based bidirectional interface.
- Chapter 6 gives a testing of the python interface and provides a guideline for prospective users. Also, the reflection and potential improvements that could be done for future work are presented.

# 2. THEORY BACKGROUND AND RELATED WORK

A python-based interface will be developed to enable an seamless flow of information through the urban energy simulation process. Before the implementation of the python interface, all the data relate to urban energy simulation will be well organized and stored in the database. Then in the python environment, the data needed for CitySim urban energy simulation will be extracted and pre-processed until it is complete for generating the CitySim input data file. After running the simulation in python console, the simulation results will be converted and stored back to the database. Before starting the project, it is very important to understand the theory and relationships among the elements involved. Those topics are related to the urban energy simulation data model and urban energy simulation tools. In this chapter, the detail theory background and several case studies will be discussed.

## 2.1. Urban Energy Simulation Data Model

Throughout the urban energy simulation process, from the initial data storage to the continuous data format conversion, several theories relate to data model are involved. In this subsection, the detail theory of urban energy simulation data model will be discussed.

### 2.1.1. CityGML

For varies application of 3D city models, not only are the geometrical and graphical aspects relevant, but also the semantics of objects [Gröger and Plümer, 2012]. To illustrate the problem in more detail, in urban energy simulation, the function of buildings plays an important role, since buildings with different function have different energy demand. Besides, calculating the heat transfer of walls, roofs, and floors are likely to be different. This places high demands on the urban energy simulation data model. CityGML gives a solution to those problems.

CityGML is from the international standard of the Open Geospatial Consortium (OGC) for representing and exchanging of 3D objects related to city since 2008. It defines how to represent the geometrical, semantic, and visual aspects of 3D city models, such as buildings and their components (building parts, walls, roofs, floors, windows, etc.), terrain, vegetation, transportation, water bodies and furniture (see figure 2.1). What's more, the representation of those objects' relationship is also include, like the relationship of a wall to the floor it bounded. These thematic definitions address the problem of differentiate semantic objects and facilitate data integration [Gröger and Plümer, 2012]. It also defines the standard of representing different LoD for the 3D objects, which can be used in various situation and purpose. In urban energy simulations, for example, detailed LoD2 building models containing thematic surfaces

(roofs, walls and floors) are used for energy calculation, while basic LoD1 building models containing geometry solid are more suitable as shading surfaces.

For the representation of 3D objects, CityGML uses a standardized model that is provided by the Geography Markup Language (GML) [OGC, 2010] which is based on the markup language Extensible Markup Language (XML). This makes CityGML fits perfectly in exchanging and modifying spatial data. CityGML has been accepted well by the software industry: tools from nearly all notable companies provide CityGML interfaces [OGC, 2010]. This allows the CityGML data model to be well adapted to almost all urban energy simulation tools.

| Thematic module | Feature types | Feature types (cont'd.) |
|---|---|---|
| Core | _CityObject (LoD0–4) | |
| Relief | ReliefFeature (LoD0–4) | TINRelief (LoD0–4) |
| | MassPointRelief (Lo 0–4) | BreaklineRelief (LoD0–4) |
| | RasterRelief (LoD0–4) | |
| Building | Building (LoD0–4) | BuildingPart (LoD0–4) |
| | BoundarySurface (LoD2–4) | BuildingFurniture (LoD4) |
| | Opening (Door, Window) (LoD3–4) | Room (LoD4) |
| | BuildingInstallation (LoD2–4) | IntBuildingInstallation (LoD4) |
| Tunnel | Tunnel (LoD1–4) | TunnelPart (LoD1–4) |
| | BoundarySurface (LoD2–4) | TunnelFurniture (LoD4) |
| | Opening (door, window) (LoD3–4) | HollowSpace (LoD4) |
| | TunnelInstallation (LoD2–4) | IntTunnelInstallation (LoD4) |
| Bridge | Bridge (LoD1–4) | BridgePart (LoD1–4) |
| | BoundarySurface (LoD2–4) | BridgeFurniture (LoD4) |
| | Opening (door, window) (LoD3–4) | BridgeRoom (LoD4) |
| | BridgeInstallation (LoD2–4) | BridgeConstructionElement (LOD1–4) |
| | IntBridgeInstallation (LoD4) | |
| Transportation | Road (LOD0–4) | Railway (LOD0–4) |
| | Track (LOD0–4) | Square (LOD0–4) |
| | TrafficArea (LOD2–4) | AuxiliaryTrafficArea (LOD2–4) |
| Water Body | WaterBody (LoD0–4) | WaterClosureSurface (LoD2–4) |
| | WaterGroundSurface (LoD2–4) | WaterSurface (LoD2–4) |
| Vegetation | SolitaryVegetationObject (LoD0–4) | PlantCover (LoD0–4) |
| CityFurniture | CityFurniture (LoD1–4) | |
| LandUse | LandUse (LoD1–4) | |
| Group | CityObjectGroup (LoD0–4) | |
| Generics | GenericCityObject (LoD0–4) | _GenericAttribute (LoD0–4) |

Figure 2.1.: Thematic modules and associated feature types including the LoDs in which they are defined [OGC, 2010].

### 2.1.2. CityGML Energy ADE Extension

In the energy sector, however, CityGML is not sufficient to cover all energy-related information. To address this issue, Nouvel et al. [2015] presented an extension of CityGML called CityGML Energy Application Domain Extensions (ADE) Extension. The overall goal of CityGML Energy ADE Extention is to tackle the existing data interoperability issues when dealing with energy-related applications at the urban scale. This interface is able to increase the number of energy-relevant properties that the city model offers (like physical materials, thermal zones, boundaries, and building occupant behaviour illustrated in Figure 2.2), thus makes CityGML well adapted [Sola et al., 2020].

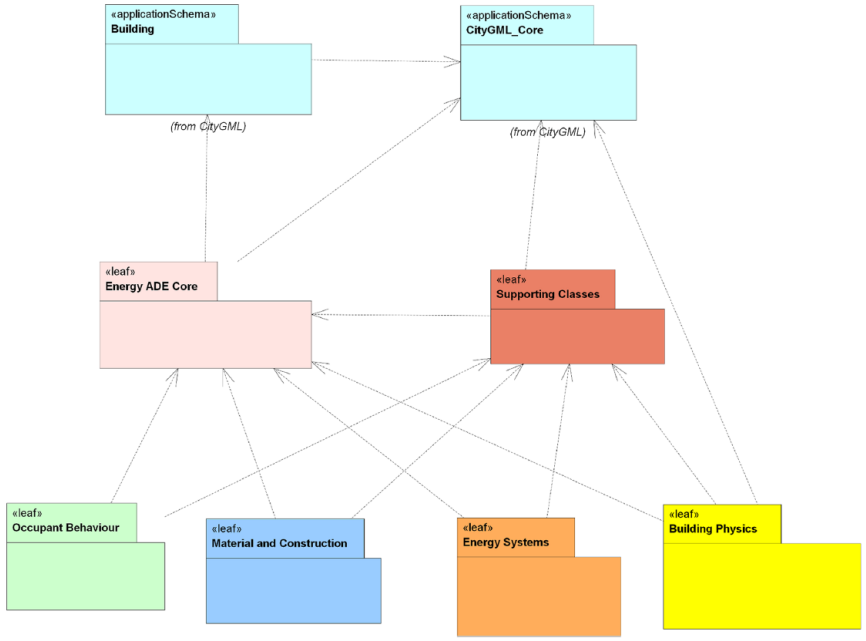



Figure 2.2.: The color-coded modular structure of the Energy ADE UML diagram [Agugiaro et al., 2018]

The detail introduction and description of the latest release of the Energy ADE Extension (v. 1.0) is given by Agugiaro et al. [2018]. The paper presents a number of software as well as concrete case studies where Energy ADE plays a role in facilitating data interoperability for energy-related applications, albeit with varying degrees of integration. In this research, the use of CityGML Energy ADE Extention well supports the simulation result storage. For example, the extended tables *ng_building*, *ng_timeserises* and *ng_regulartimeseries* are used for storing the heat and cooling energy demands.

## 2.2. Urban Energy Simulation Tools

As mentioned above, with wide range of adaptability, a great number of urban energy simulation tools have been developed based on the open semantic 3D city model CityGML [Gröger

and Plümer, 2012]. The aim of those tools is to be the basis of computing energy related information (such as solar irradiance of building surfaces and heat and cooling demands of single building) and help design and operation of building and district energy systems [Sola et al., 2020]. Each tools has its own characteristics and is suitable for different scenarios. Those tools include such as CitySim, SynCity, LakeSim and IDEAS. In this research, the exploration and processing will be established on CitySim.

### 2.2.1. CitySim

CitySim belongs to the family of energy simulation software with the peculiar characteristic of considering the simulation scene as an urban environment whose individual elements interact with each other. CitySim was initially developed as a command-line solver and tested at the Solar Energy and Building Physics Laboratory (LESO-PB) of EPFL. After years of satisfactory results, the energy consulting company kaemco developed a graphical user interface called CitySim Pro, which is a more user-friendly version. CitySim is based on algorithms providing good balance among simulation requirements, simulation speed and simulation accuracy at urban scale [Peronato et al., 2017]. This makes CitySim the ideal software to fulfill the intended urban energy simulation scenario. CitySim takes two main data files to compose the simulation scene, one is the analysed location weather data and the other is geometric and physics information of the buildings [Mutani et al., 2018]. The workflow of CitySim is shown in figure 2.3.


Figure 2.3.: CitySim workflow

### 2.2.2. CitySim data model and CityGML with Energy ADE data model

CitySim has its own data model which is different from the CityGML with Energy ADE data model. In order to perform energy simulation based on the existing CityGML with Energy ADE data model, the characteristics and relationship between those two data formats need to be explained.

The relationship between CitySim data model and CityGML with Energy ADE data model is given by Coccolo and Kämpf [2015]. CitySim data model is based on XML format. The structure is tree-based with objects included in other objects. For example, every building in the scene defined with energy system for heating and cooling purposes and building thermal zone which contains the information about volume, temperature for heating/cooling and infiltration, etc. The building thermal zone contains further information such as occupation (number of occupants and profile), description of walls, roofs and floors (geometrical and

physical information). Whereas, the CityGML with Energy ADE data model is structured by four interrelated modules which are building, zone and boundaries, construction and layers, occupancy module and energy system module. Those modules can present individually or linked with other through reference. The structure of those two data model is shown in 2.4.

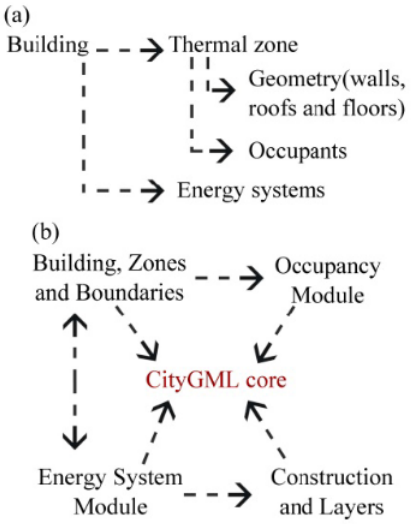



Figure 2.4.: The idealized data structure of CitySim (a), and CityGML Energy ADE (b) [Coccolo and Kämpf, 2015]

With above analysis, it can be concluded that CitySim and CityGML data model are able to have common data organized in different structures. Containing the same information, the CitySim data model is structured as tree-based whereas CityGML data model is organized in independent modules. To illustrate with concrete cases, below is a XML file containing a LoD2 building written by the software CitySim. The tree-based structure XML has its vertex in the geometry, which belongs to thermal zone together with occupants information. While the thermal zone and energy systems information are inside the building tag.

```
<Building id="0" key="id_building_01" Vi="1250.0" Ninf="0.1" BlindsLambda="0.2"
BlindsIrradianceCutOff="100" Simulate="true">
<HeatTank V="0.01" phi="20.0" rho="1000.0" Cp="4180.0" Tmin="20.0" Tmax="35.0"/>
<CoolTank V="0.01" phi="20.0" rho="1000.0" Cp="4180.0" Tmin="5.0" Tmax="20.0"/>
<HeatSource beginDay="258" endDay="135">
<Boiler name = "spaceX" Pmax="10000000.0" eta_th="0.95"/>
</HeatSource>
<CoolSource beginDay="136" endDay="257">
<HeatPump Pmax="10000000.0" eta_tech="0.3" Ttarget="5.0" Tsource="air"/>
</CoolSource>
<Zone id="0" volume="1000.0" psi="0" Tmin="20" Tmax="26" groundFloor="true"
nightVentilationBegin="0" nightVentilationEnd="0">
<Occupants n="5.0" sensibleHeat="90" sensibleHeatRadiantFraction="0.6" latentHeat="0"
type="1"/>
<Wall id="0" key="id_building_1_polygon_4" type="204.0" ShortWaveReflectance="0.3"
GlazingRatio="0.17" GlazingGValue="0.6" GlazingUValue="3.7" OpenableRatio="0">
<V0 x="-40.0" y="40.0" z="0.0"/>
<V1 x="-40.0" y="40.0" z="10.0"/>
<V2 x="-35.0" y="40.0" z="15.0"/>
```

```
<V3 x="-30.0" y="40.0" z="10.0"/>
<V4 x="-30.0" y="40.0" z="0.0"/>
</Wall>
......
<Floor id="2" key="id_building_1_polygon_3" type="132.0" ShortWaveReflectance="0.0"
GlazingRatio="0.0" GlazingGValue="0" GlazingUValue="0" OpenableRatio="0">
<V0 x="-40.0" y="30.0" z="0.0"/>
<V1 x="-40.0" y="40.0" z="0.0"/>
<V2 x="-30.0" y="40.0" z="0.0"/>
<V3 x="-30.0" y="30.0" z="0.0"/>
</Floor>
<Roof id="3" key="id_building_1_polygon_2" type="166.0" ShortWaveReflectance="0.2"
GlazingRatio="0.0" GlazingGValue="0.6" GlazingUValue="3.7" OpenableRatio="0" kFactor="0">
<V0 x="-35.0" y="30.0" z="15.0"/>
<V1 x="-30.0" y="30.0" z="10.0"/>
<V2 x="-30.0" y="40.0" z="10.0"/>
<V3 x="-35.0" y="40.0" z="15.0"/>
</Roof>
</Zone>
</Building>
```

On the contrary, the CityGML data model is written in modular structure based on four interconnected models (Building, Zones and Boundaries, Construction and Layers, Occupancy Module and Energy System Module).

```
<core:cityObjectMember>
<bldg:Building gml:id="id_building_01">
<bldg:lod2Solid>
<gml:Solid>
<gml:exterior>
<gml:CompositeSurface>
<gml:surfaceMember>
<gml:Polygon gml:id="b0_p_w_0">
<gml:exterior>
<gml:LinearRing>
<gml:posList>
-40 40 0
-40 40 10
-35 40 15
-30 40 10
-30 40 0
-40 40 0
</gml:posList>
</gml:LinearRing>
</gml:exterior>
</gml:Polygon>
</gml:surfaceMember>
......
<energy:volume>
<energy:VolumeType>
<energy:type>grossVolume</energy:type>
<energy:value uom="m3">1000</energy:value>
</energy:VolumeType>
</energy:volume>
......
<energy:thermalZone>
......
<energy:usageZone>
......
```

In this research, the transformation between those two data formats consists the core of the

information flow during the energy simulation process. The conversions between those two types of data models will be achieved in Python environment.

### 2.2.3. 3D city database

As mentioned in Section 2.1, CityGML data model is widely used in representing 3D city objects which benefits application and users a lot. However, there are also shortcomings exist.

Typically, a CityGML data file contains all the building data within the scope of the required analysis. This results in a large file size of the 3D models of cities and single CityGML file for a bigger city or region can have from tens over hundreds gigabytes in size. This brings high demand on the need for efficient tools to query, visualize, and update the 3D city model [Yao et al., 2018].

3DCityDB addresses the challenge above. 3DCityDB is a free and open-source database platform. As a spatial Relational Database Management System (RDBMS), 3DCityDB perfectly suit the needs of storing, representing, and managing virtual 3D city models. It can decode CityGML and map its data model from inheritance hierarchy onto several tables (see figure 2.5). This database mapping schema is the core component of the 3DCityDB [Yao et al., 2018]. For example, the 3DCityDB supports the PostgreSQL with the PostGIS extension, after the installation in pgAdmin, there is a *citydb* schema which contains tens of tables ready for storing the parsed CityGML information such as table *building*, *citymodel*, *cityobject*, *surface_geometry* and so on. The database schema makes the parsed CityGML data well distinguished and connect. Worth to mention, the importing and exporting of CityGML file can be done by software *citydb* importer/exporter.

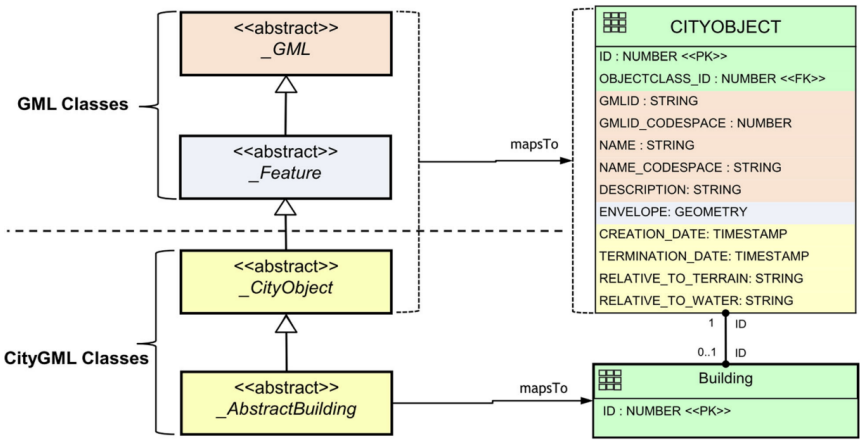



Figure 2.5.: Example of mapping an inheritance hierarchy onto one table [Yao et al., 2018]

In order to handle CityGML with Energy ADE data model, through research work being carried out recent years, the 3DCityDB extension for the Energy ADE is also released to the public. As a powerful tool chain, the Energy ADE extension extend the 3DCityDB with tens of tables specifically for dealing with energy related information. The combination of 3DCityDB

and Energy ADE extension makes the ideal database for handling data related to urban energy domain. For example, the figure 2.6 shows the geometry of a CityGML data file stored in 3DCityDB table *surface_geometry* and the figure 2.7 shows the intended Energy ADE table *ng_timeseries* for storing the simulation result.

Query Editor Query History

```
1  SELECT * FROM citydb.surface_geometry
2  ORDER BY id ASC
```

Data Output Explain Messages Notifications

| | id [PK] bigint | gmlid character varying (256) | gmlid_codespace character varying (1000) | parent_id bigint | root_id bigint | is_solid numeric | is_composite numeric | is_triangulated numeric | is_xlin numer |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | id_lod0_MultiSurf_28 | [null] | [null] | 1 | 0 | 0 | 0 | |
| 2 | 2 | id_lod0_Polygon_28 | [null] | 1 | 1 | 0 | 0 | 0 | |
| 3 | 4 | id_lod0_MultiSurf_29 | [null] | [null] | 4 | 0 | 0 | 0 | |
| 4 | 5 | id_lod0_Polygon_29 | [null] | 4 | 4 | 0 | 0 | 0 | |
| 5 | 7 | id_lod1_Solid_28 | [null] | [null] | 7 | 1 | 0 | 0 | |
| 6 | 8 | id_lod1_Solid_29 | [null] | [null] | 8 | 1 | 0 | 0 | |
| 7 | 9 | id_lod1_CompSurf_28 | [null] | 7 | 7 | 0 | 1 | 0 | |
| 8 | 10 | id_lod1_Polygon_49 | [null] | 9 | 7 | 0 | 0 | 0 | |
| 9 | 11 | id_lod1_Polygon_50 | [null] | 9 | 7 | 0 | 0 | 0 | |
| 10 | 12 | id_lod1_CompSurf_29 | [null] | 8 | 8 | 0 | 1 | 0 | |
| 11 | 13 | id_lod1_Polygon_51 | [null] | 9 | 7 | 0 | 0 | 0 | |

Figure 2.6.: A glimpse of 3DCityDB table *surface_geometry* with data of alderaan.gml

```
1  SELECT * FROM citydb.ng_timeseries
2  ORDER BY id ASC
```

Data Output Explain Messages Notifications

| id [PK] bigint | objectclass_id integer | timevaluesprop_acquisitionme character varying (1000) | timevaluesprop_interpolation character varying (1000) | timevaluesprop_qualitydescri character varying (1000) | timevaluesprop_thematicdescr character varying (1000) | timevaluespropertiest_source character varying (1000) |
|---|---|---|---|---|---|---|

Figure 2.7.: The intended Energy ADE table *ng_timeseries* for simulation result storage

Further information about 3DCityDB can be read in Agugiaro et al. [2018]. In this research, the 3DCityDB (v. 5.0.0), Energy ADE (v. 2.0) and PostgreSQL (V. 13) are being used.

### 2.2.4. Weather data

As mentioned before, CitySim takes location weather data as an input file for energy simulation. The analysed location weather data have to be hourly resolution and contains information such as solar irradiance, wind speed and wind direction which are stored in a .cli file. Those elements are shown in Table 2.1.

| | CLI file requirement | UOM |
|---|---|---|
| d | Day | |
| m | Month | |
| h | Hour | |
| G_Dh | Diffuse horizontal irradiance | W/m2 |

| | CLI file requirement | UOM |
|---|---|---|
| G_Bn | Beam (solar) normal irradiance | W/m2 |
| Ta | Air temperature | °C |
| Ts | Ground surface temperature | °C |
| FF | Wind Speed | m/s |
| DD | Wind Direction | ° |
| RH | Relative Humidity | % |
| RR | Precipitation | mm |
| N | Nebulosity | Octas |

Table 2.1.: CitySim input climate file elements

As introduced by Mutani et al. [2018], there are several tools able to create climate data file of a specific location in the world (e.g. Meteonorm). In this research two ways of generating .cli weather file are developed. One is from open source website `https://climate.onebuilding.org/default.html` which contains climate data designed specifically to support building simulations. Another is from available weather data stored in database. The detail implementation is illustrated in Section 4.4.

### 2.2.5. Horizon data

Apart from weather data, CitySim also needs the horizon information as the starting input data which gives the shape of the horizon according to the direction or simulation scene. As it is beyond the scope of this research, it will not be covered in this thesis.

## 2.3. Case Study

This section will discuss two case studies related to CitySim urban energy simulations. The first one shows the urban energy simulation processing workflow of a neighborhood in ZÜRICH. The second one shows a similar research goal as proposed in this research which I take inspiration and partially continues their methodology.

### 2.3.1. CitySim simulation: the case study of alt-wiedikon a neighbourhood of Zürich city

In order to perform energy simulation of a district, first, varies data needed for CitySim are collected from different sources. Then a simulation model for CitySim needs to be generated from those collected data. The problem is organising those data into a coherent simulation model is a highly time-consuming task. This research presents the creation of a database model for storing urban energy information using PostgreSQL and its link to the CitySim (see figure 2.8). With this designed database, the information are well organized and the creation of the CitySim input file and results storage are made automatically by JAVA interface.

The ideal of storing and managing the data in database and the use of JAVA interface making data handling automated to reduce data conversion are same as proposed in this research.

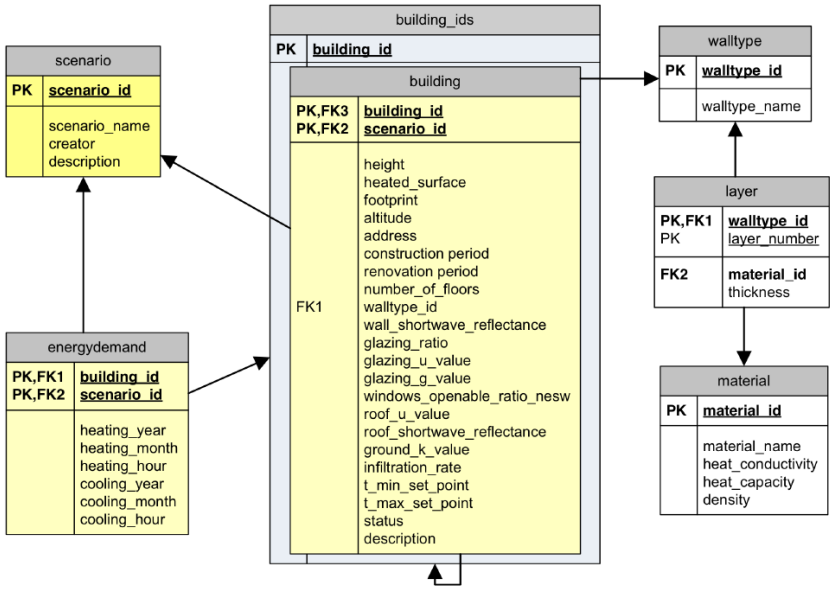



Figure 2.8.: Schema of the designed CitySim database [Perez et al., 2011]

However, since the database is specifically tailored to CitySim urban energy simulation, the method doesn't adapted to other energy application since the schema of the database doesn't cover all energy related information. For a large city instead of a neighborhood, the data collection time consuming would be another issue.

### 2.3.2. Machine learning for the energy of buildings on a GIS tool design project 2021

In order to predict the energy demand of residential buildings with different hourly resolutions, preliminary work of the same research goal mentioned in Chapter 1 was made by Sélène Ledain and Maxwell Bergström from EPFL in 2021. The complete task plan for data collection and organization is shown in Figure 2.9. The goal of this project is to collect data on buildings (located in Monthey, Switzerland) which are connected to the SATOM network to perform energy simulations in the CitySim in order to evaluate the temporal granularity of the energy measurements.

The 3D building models are stored in 3DCityDB based on the CityGML 2.0 structure. The physics parameters are collected from RegBL of the Federal Statistical Office by web scraping. Those physics parameters are linked to buildings by attribute *year of construction* (it is possible to relate the year to the materials and properties of the building). After collection, those data are inserted into the database for storage by Structured Query Language (SQL) queries (represented by the green arrows in Figure 2.10). Besides, the information about energy consumption measurements (which reach at best a three-hour accuracy) collected from open source website are also stored in the database. Once the data preparation is complete, the data are extracted by SQL queries to generate a CitySim input file (represented by the

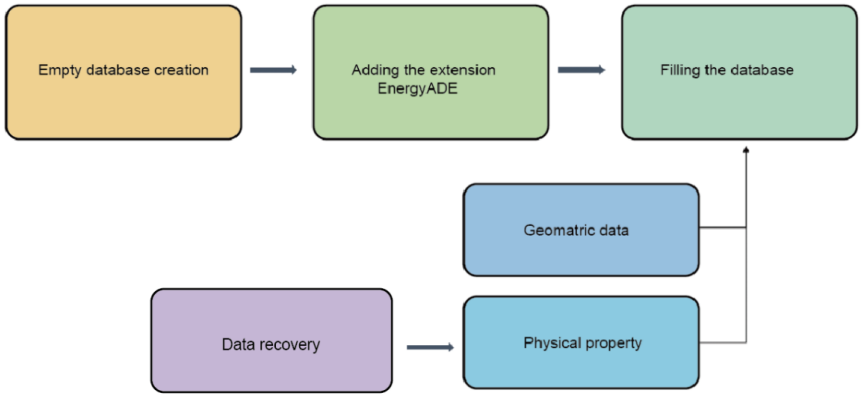



Figure 2.9.: Scheme of complete tasks for data collection and organization (Sélène Ledain and Maxwell Bergström, 2021)

yellow arrows in Figure 2.10) for simulation. After the simulation, several needed simulation results are stored back into the database. All steps mentioned above are achieved by a python-based interface.

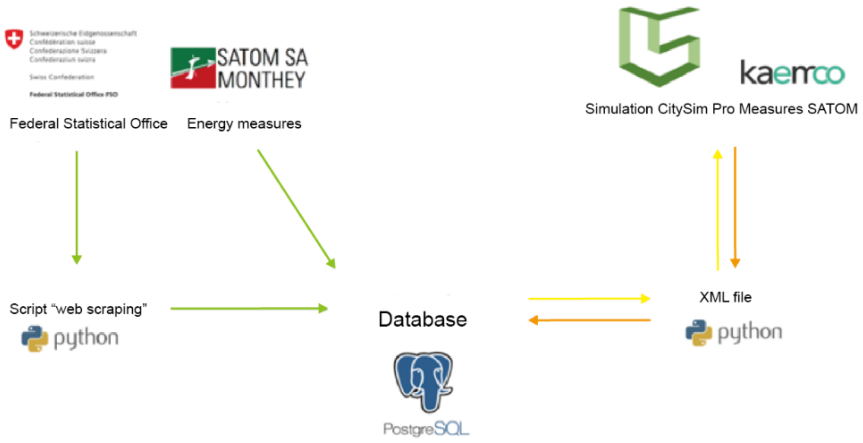



Figure 2.10.: Urban energy simulation workflow (Sélène Ledain and Maxwell Bergström, 2021)

From a database containing energy-related data to CitySim simulation and results storage back, the ideal of creating an seamless information flow was proposed. However, the data collected are incomplete due to the different sources of collection for different data. The missing of certain information makes the program not well performed. Thus, a stable and complete data source could make the project much more easier. In addition, the program was not fully developed due to lack of time. From the two cases discussed above, they both have advantages and shortcomings that can be learned and addressed in this study. With a complete data source (3D BAG 2.0), a data management platform (3DCityDB), and a simulation tool (CitySim), the above problems will be solved.

# 3. METHODOLOGY

In this chapter, the research methods will be discussed. Firstly, a research approach covering the whole project process will be presented. This includes data inspection, management, data extraction, spatial and non-spatial data pre-processing functions, mapping and simulation and results storage.

## 3.1. Research Approach

In order to answer the main and sub research questions, a research methodology covering the entire project process needs to be designed. This research methodology includes the following points:

- **Data inspection**: examine the existing data and the data needed for CitySim urban energy simulation and compare it to identify what data preparation is required.
- **Management**: for missing data, data collection is performed. This includes data relates to specific building entity and the data library covering all geometries. If the data storage schema is not contained in 3DCityDB, a schema design is carried out so that the relevant data can be stored with the rest all in one database.
- **Data extraction**: in python environment, the data required for the urban energy simulation is extracted from the database, including geometry data, physics parameters and weather data. The data is converted into dataframes and merged step by step to generate a complete input file for CitySim simulation.
- **spatial and non-spatial data pre-processing functions**: in python environment, several spatial and non-spatial data pre-processing functions related to CitySim urban energy simulation are carried out.
- **Mapping**: once the extraction of the data and associated processing are complete, the data is translated into CitySim XML input format in python.
- **Simulation and result storage**: in python environment, call CitySim to run the energy simulation, and after the simulation is complete, store the simulation results back into the database based on 3DCityDB Energy ADE 2.0 schema.

The overview of the workflow is shown in the figure 3.1. The six main steps are discussed further in subsections.

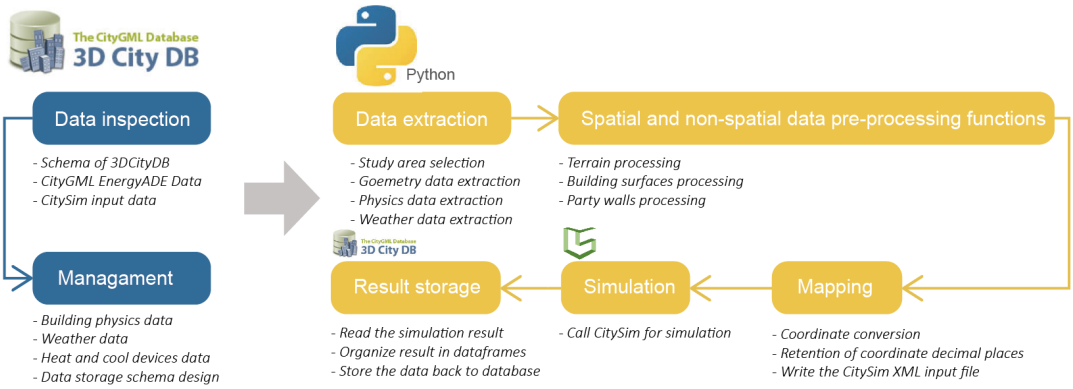



Figure 3.1.: Methodology of the research

### 3.1.1. Data inspection

In the data inspection session, it is first necessary to identify what input data is required by the CitySim urban energy simulation and then to see what the CityGML data stored in 3DCityDB consists of. Based on Mutani et al. [2018], it is possible to know the input files required for CitySim simulations. As mentioned in Section 2.2.1, CitySim takes two main information to compose the simulation scene, one is the analysed location weather data and the other is geometric and physics information of the buildings. After comparison, it is confirmed that apart from geometric data stored 3DCityDB, the physics parameters and weather data need to be collected. In addition to this, it is important to understand the data storage structure of 3DCityDB as it relates to how data is extracted from multiple tables.

### 3.1.2. Management

Data collection is required for the missing data. One way of collecting data is from open source websites. The other way is to construct synthetic data. As the aim of this project is to build a python-based interface that covers the whole process of CitySim urban energy simulation, the synthetic data will not affect the adaptation of the program to the real dataset. There are two datasets that will be used in this research. One is the test dataset Alderaan.gml to build the python interface and the other is a real dataset RijssenHolten.gml from 3D BAG 2.0 for testing the interface adaptability. Apart from the CityGML data that are decoded and well stored in 3DCityDB, for the rest data collected (i.e. physics parameters and weather data), a storage schema targeting those data is designed in order to build a physics and weather data library which covering all potential need in future simulation. Worth to mention, since the 3DCityDB extension for energy ADE is used for storing information of each buildings in database, it is not able to store the physics and weather information library which covers all circumstance (i.e. the physics parameters libraries are about all buildings in the Netherland). Detail information regarding data collection and preparation will be given in Section 3.2 and Chapter 4.

### 3.1.3. Data extraction

The extraction of the data from the database containing complete information required for CitySim simulation will be done in python. The data to be extracted includes geometry data, physics parameters and weather data. Those data are converted into pandas.DataFrame and merged from basic LoD2 geometries to enriched energy simulation-related data. The workflow is shown in figure 3.2. The purpose of this is to check the integrity of the data at each level and to enable the user to extract data of varying precision. The implementation details will be given in Section 5.1.

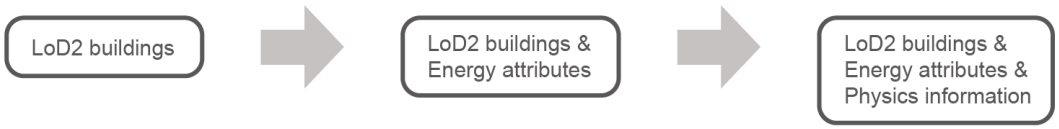



Figure 3.2.: Data enrichment workflow

### 3.1.4. Spatial And Non-spatial Data Pre-processing Functions

As mentioned in Section 2.3, the problem of CitySim data manipulation still exists which affect the consuming time such as data collection and simulation calculation. Besides, CitySim has special requirements for the input geometry which needs to do several pre-processing before simulation. As the data related to the urban energy simulation will all be extracted by the python interface, it makes it possible to design spatial and non-spatial functions based in the python environment. These can include functions which helps different purposes of data collection such as extracting data inside the study area, displaying the number of buildings selected, setting the buffer of buildings to include the surrounding trees, terrain and shading surfaces, coordinate shifting, decimal place managing of coordinate, etc. In addition to this, functions that make the geometry pre-processing of buildings and terrain will be developed according to the input data requirements of CitySim. Based on the python environment, those spatial and non-spatial data pre-processing can be done. Detail information regarding spatial and non-spatial data pre-processing functions will be given in Section 3.3 and Chapter 5.

### 3.1.5. Mapping

As mentioned in Section 2.2.2 CitySim and CityGML data models organize data in different structures and the mapping between those two formats will be established. The data extracted and processed in python will be mapped to the CitySim input data structure (i.e. the data organized in dataframes will be written as an XML file according to the formatting requirements). It is worth noting that the output data is subject to coordinate conversion and the retention of specific decimal places, which is intended to reduce the effect of loss of significance. This can help reduce the XML file size and reduce the simulation time. Detail information regarding mapping will be given in Section 5.6.

### 3.1.6. Simulation and result storage

With a complete CitySim input data, CitySim is called in python to perform energy simulation. After the simulation, the needed data are extracted from the result files and reorganised into dataframes. Finally those simulation results are stored back in 3DCityDB according to the Energy ADE (v. 2.0) structure. Detail information regarding simulation and result storage will be given in Section 5.7.

## 3.2. Management

In the data management section, the data storage schema design needs to be further clarified which are not covered by 3DCityDB structure.

The 3DCityDB (v. 5.0.0) and Energy ADE (v. 2.0) architecture includes storage of 3D models of cities, trees, terrain etc. as well as energy-related information. However, the architecture is not able to store the data libraries for building physics, terrain physics, tree physics, weather data, heating and cooling devices information required for CitySim simulation. To well organize those data in a library for future use, a data storage schema needs to be designed. To enable data to be stored in a clear storage structure for data extraction, the design principle is: one category information is stored in a schema, and the information is split by content and stored in multiple tables. Data in different tables are associated with each other based on their primary key or foreign key or attribute. The data storage structure design methodology is shown in figure 3.3. Detail implementation will be given in Chapter 4.

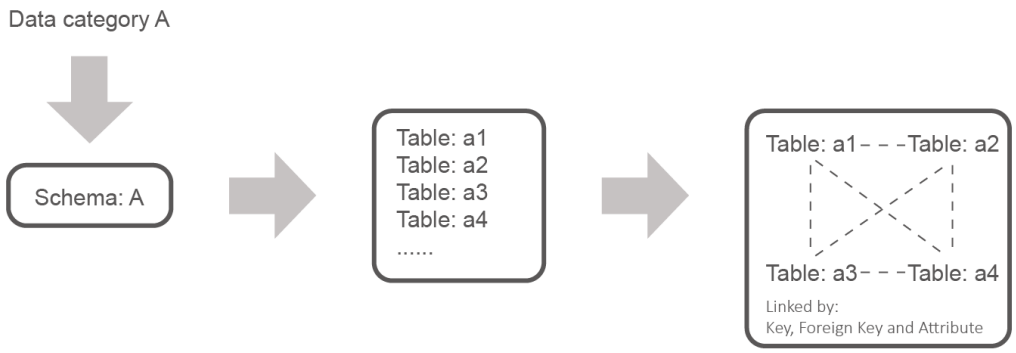



Figure 3.3.: Data storage structure design methodology

## 3.3. Spatial and non-spatial data pre-processing functions

In this section, one spatial and two non-spatial data pre-processing functions are further discussed.

### 3.3.1. Study area selection

CitySim input geometry data for urban energy simulations only requires information about the building to be simulated and its surrounding information (e.g. trees, terrain and shading buildings), so it is often sufficient to provide the data required for a specific area to be simulated. For database that may contain 3D model for the whole city, if only certain area's energy simulation is needed, then extracting the data for the whole city for simulation is unnecessary and computational expensive. This not only increases the energy calculation time, but also causes problems for simulation result storage (e.g. extracting certain values from a large amount of data is complicated).

In order to solve the above problems, the function study area selection will be developed in this project. The function can only extract the geometry data from the database within the user input bounding box, including buildings to be simulated (LoD2), buildings as shading objects (LoD1), trees and terrain. It is Besides, based on the bounding box, this function also sets the buffer for including buildings as shading surfaces (LoD1), trees and terrain around the simulated building that affect the energy calculation. The workflow of study area selection is shown in figure 3.4. Detail implementation regarding study area selection will be given in Section 5.1.2.

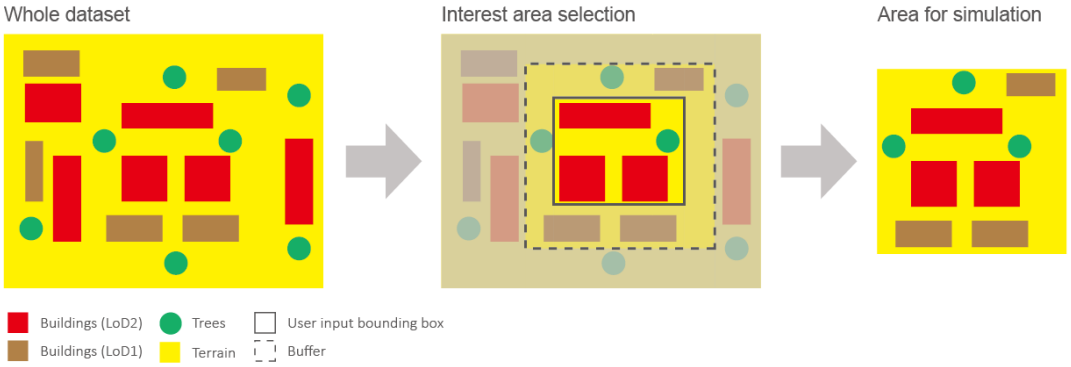



Figure 3.4.: Workflow of the study area selection

### 3.3.2. Terrain processing

The geometry data of the terrain is constructed according to triangular irregular network (TIN). In CitySim mechanism, the terrain triangle will not be simulated (e.g. to get the solar irradiance result) if its centre point is under the building being simulated. This can result in inaccurate energy simulation result due to the lack of complete terrain geometries. The issue is illustrated in figure 3.5.

To solve this problem, the terrain geometry needs to be pre-processed before simulation. The first step is to project the 3D terrain and building footprints onto the 2D X-Y plane, the second step is to cut the building footprints out of the terrain, the third step is to re-triangulates the cut terrain and the fourth step is to project the terrain back into the 2.5D representation. The workflow of terrain processing is shown in figure 3.6. Detail implementation regarding terrain processing will be given in Section 5.3.

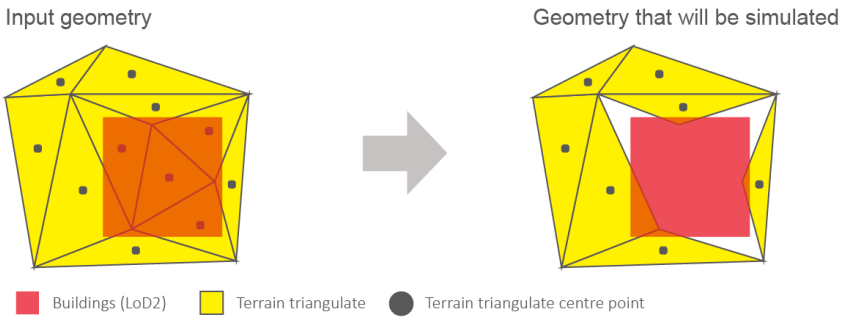



Figure 3.5.: The difference of input geometry and simulated geometry

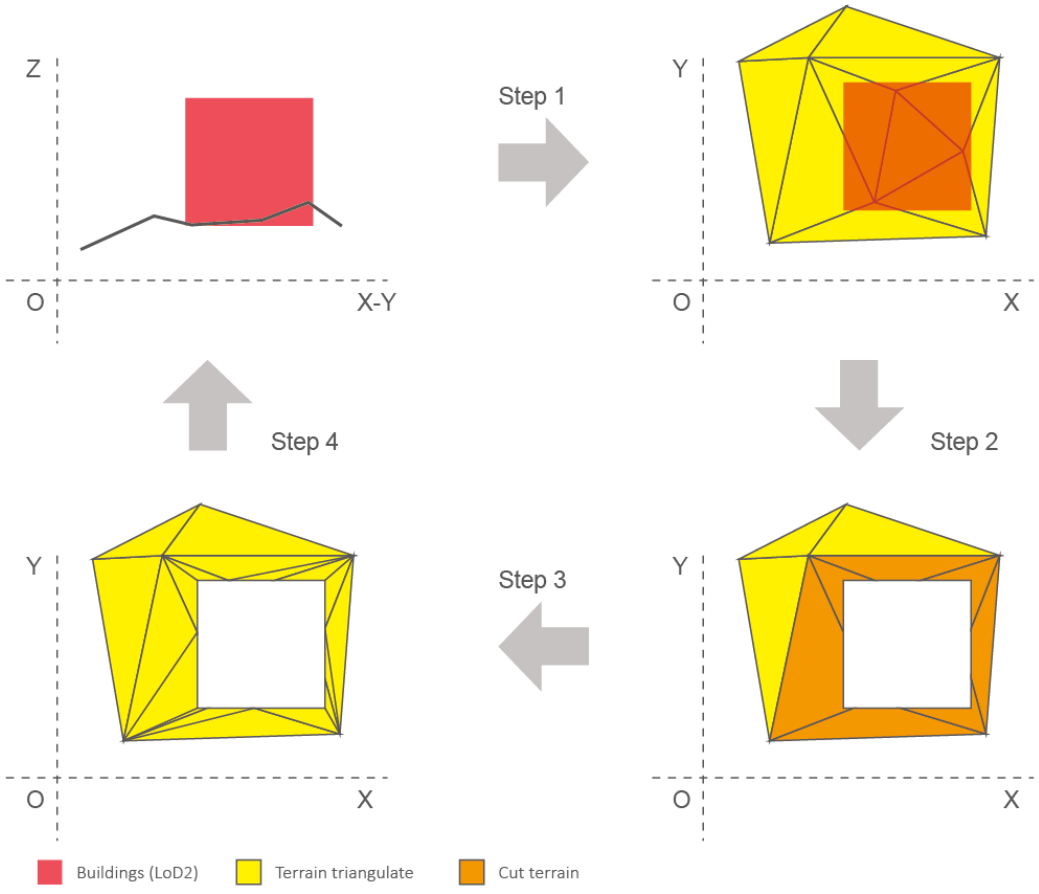



Figure 3.6.: Workflow of terrain processing

### 3.3.3. Building surface processing

The LoD2 building geometry is made up of surfaces (polygons), including walls, roofs and floors. In CitySim mechanism, if the centre point of a surface does not lie on that surface, then the surface related energy simulation will not by carried out (e.g. solar irradiance). This mostly occurs with concave (non-convex) surfaces. See figure 3.7.

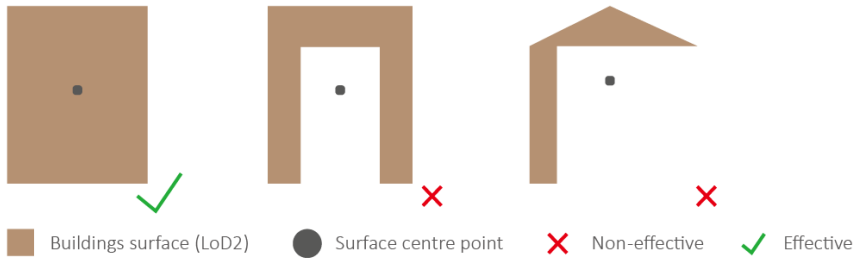



Figure 3.7.: Building surface will be simulated or not

It should be clear that not all concave surfaces have a centre point that lies inside the surface. The function is designed to process re-triangulation of all concave surfaces instead of only targeting surfaces whose center points do not lie on itself. The division of a surface into triangular surfaces ensures that the centre of the surface lies on itself. The workflow of re-triangulation of concave surfaces is shown in figure 3.8.

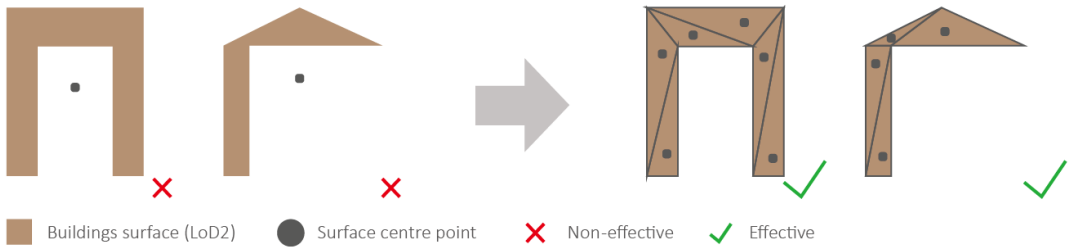



Figure 3.8.: Re-triangulation of concave surfaces

The re-triangulation of concave surfaces is handled in a similar way to the re-triangulation of terrain mentioned in Section 3.3.2. The first step is to check whether the building surface is concave or not, and for concave surfaces continue to determine whether the surface is perpendicular to the X-Y plane; if not, operations such as projection can be carried out in accordance with the processing of the terrain. If the concave surface is perpendicular to X-Y plane, the surface can not be project to X-Y plane. Then the x-axis and y-axis lengths of the surface bounding box are used to determine whether the surface should be projected onto the X-Z or Y-Z plane to get the 2D projection for further processing. The building surface projection workflow is illustrated in figure 3.9. It should be noted that the reason for comparing the x-axis and y-axis lengths of the surface bounding box is to ensure that there is a larger 2D surface after projection and that a larger 2D surface improves the accuracy of the further re-triangulation algorithm. Detail implementation regarding building surface processing will be given in Section 5.5.

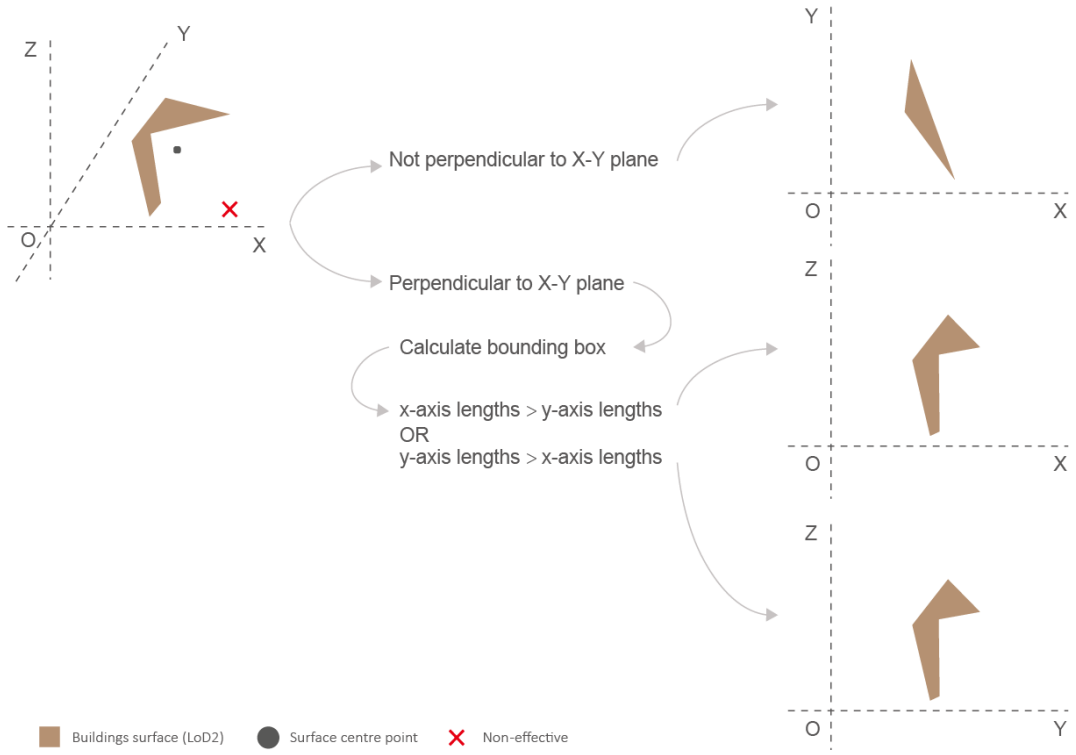



Figure 3.9.: Workflow of building surface projection

# 4. DATA PREPARATION

In this chapter, the detail information regarding data preparation is described. From an overview of the required data to specific geometry data, physics parameters and weather data including collection method and storage processing will be presented. The data preparation content covered in this chapter is the basis for implementing the python-based bidirectional interface.

## 4.1. Data Overview

By comparing existing data with the data required for CitySim energy simulations, the overall picture of work needs to be done for data preparation is clear. Table 4.1 shows the CitySim simulation required data, the existing data, the data source and the storage method. The next sections describe detailed procedure of preparing those data.

| Required data | Existing data | Data source | Storage method |
|---|---|---|---|
| building geometry | Alderaan.gml | Testing model | Through 3DCityDB importer/exporter |
| building energy related information | Alderaan.gml | Testing model | Through 3DCityDB importer/exporter |
| building physics | Dutch_building _physics.xml | TABULA | Storage schema design and through pgAdmin |
| tree geometry | Alderaan_Trees.gml | Testing model | Through 3DCityDB importer/exporter |
| tree physics | NaN | CitySim example parameters | Storage schema design and through pgAdmin |
| terrain geometry | Alderaan_DTM.gml | Testing model | Through 3DCityDB importer/exporter |
| terrain physics | NaN | CitySim example parameters | Storage schema design and through pgAdmin |
| weather | NaN | Climate.OneBuilding.Org | Storage schema design and through pgAdmin |
| heat and cool information | NaN | CitySim example parameters | Storage schema design and through pgAdmin |

Table 4.1.: Data preparation overview

## 4.2. Geometry Data

The alderaan.gml is a CityGML 2.0 test dataset. This file contains a collection of building geometry and their generic attributes (including energy related information, see Table 4.2). The building geometry is divided into two parts, the first part contains 11 buildings. Each building contains 1 LoD0 footprint, 1 LoD1 prismatic solid, a set of LoD2 thematic surfaces. The second part contains a group of additional ancillary buildings. Each ancillary building contains 1 LoD0 footprint and 1 LoD1 prismatic solid. Those ancillary buildings will be input as shading surfaces for energy simulation. In order to adapt the program to multi-part building circumstance, one of the first part 11 buildings is a multi-part building which consists of 2 building parts. In addition, to store the information about building surfaces topology (adjacency), the existing *generalisation* property of CityGML is used innovatively (which is not originally meant for). Those topology information will be used in party wall processing (see Section 5.4). The alderaan.gml is shown in figure 4.1. Trees and terrain of alderaan are also CityGML 2.0 test datasets. The trees are modelled in LoD1, LoD2 and LoD3 using implicit geometries and terrain are modelled as tiled TIN. Those three CityGML files are imported into database via software 3DCityDB importer/exporter. The geometry and attributes are stored in 3DCityDB schema.

| attributes | UOM | description |
|---|---|---|
| lod2_volume | $m^3$ | Involved in citysim energy simulation calculations |
| num_residents | | Involved in citysim energy simulation calculations |
| building_type | | Link the building physic.<br>e.g. SFH (single family house) |
| function | | Link the building physics and filter out non-residential buildings<br>e.g. residential building |
| year_of_construction | | Link the building physics, e.g. 1955 |

Table 4.2.: Attriibutes of Alderaan.gml that are important for CitySim

## 4.3. physics parameters Libraries

CitySim for energy simulation calculations requires input files containing physics information. The physics information makes up the content of the *Composite*, *OccupancyDayProfile*, *OccupancyYearProfile*, *Building*, *ShadingSurface*, *Trees* and *GroundSurface* in the CitySim input XML file. As the preparation of the physics parameters of the building is the most complex, it will be described in detail.

### 4.3.1. Building physics

The existing building physics library data is extracted from TABULA which contains Dutch building physics library. It is originally created for SimStadt. The content contained in this file is composed as shown in the figure 4.2.

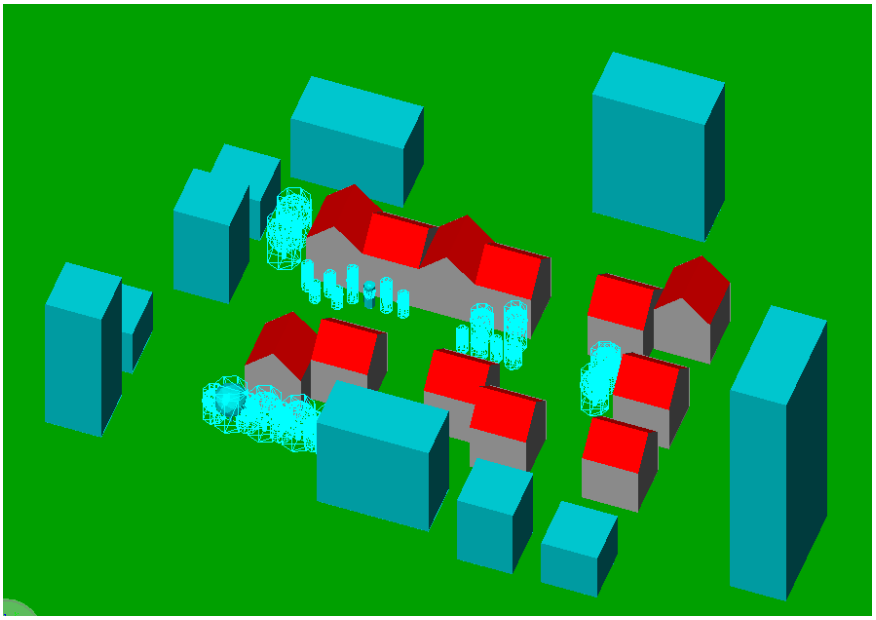


Figure 4.1.: 3D view of alderaan.gml shown in FZKViewer

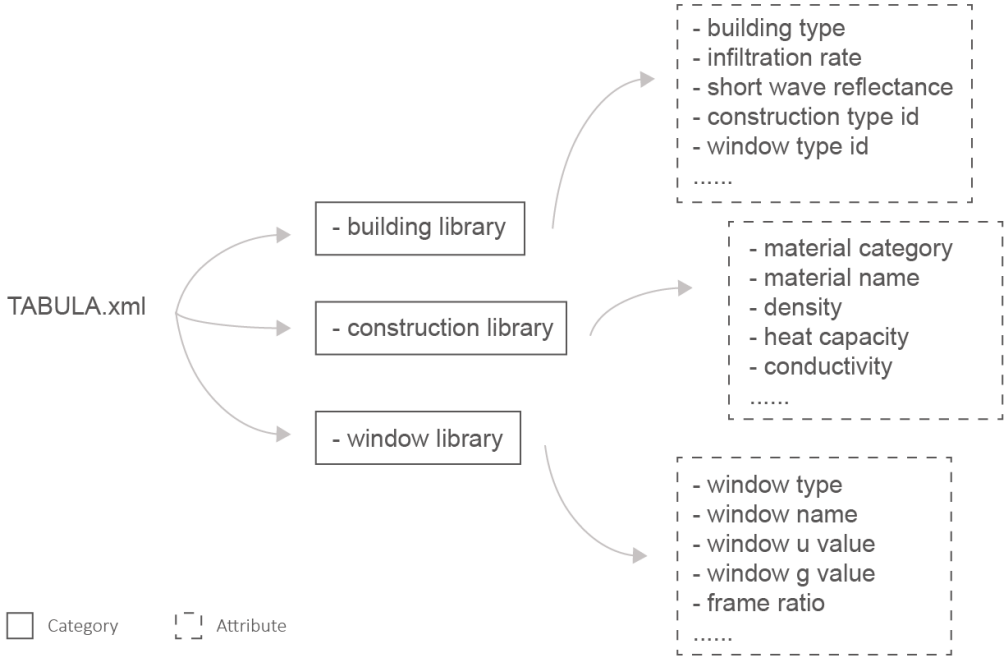



Figure 4.2.: The component structure of TABULA.xml

As 3DCityDB does not contain a storage schema for building physics library, the goal is to design a storage schema based on TABULA.xml content in the database. Also, this physics library can be stored in a separate online sever that can be queried from different applications. By analysing the data content, the TABULA.xml is parsed and stored in several tables (see figure 4.3). The physics parameters in those tables are related to each other by *id* attribute and also linked to the building geometry data by *building type*, *building function* and *year of construction* attributes. The links between those information are shown in the figure 4.4.

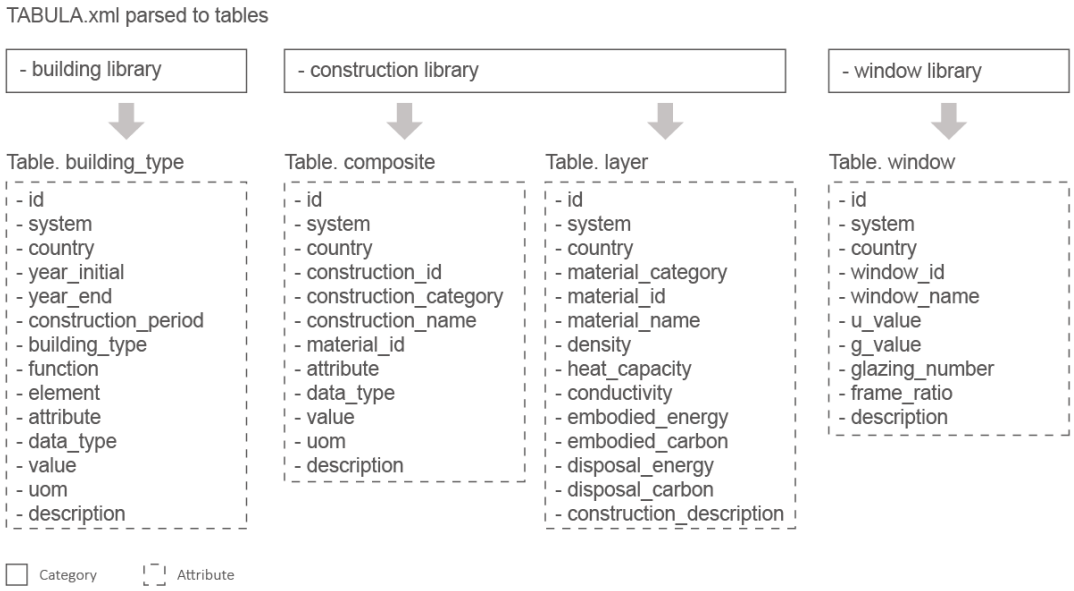



Figure 4.3.: Parse TABULA.xml into tables to be stored in database

### 4.3.2. Additional physics parameters

In addition to building physics parameters, terrain physics, tree physics and heating and cooling information are also required in CitySim energy simulations. As there is no existing file of those data, in this research the relevant parameters in CitySim example input files were extracted and stored via pgAdmin in designed storage schema.

The storage schema design follows the same principle as the building physics library storage schema, i.e. the information is split by content, stored in multiple tables and associated based on *id* attribute. The overview of parsing and storing those physics parameters is shown in figure 4.5.

## 4.4. Weather Data

Climate data are collected from open source websites such as `https://climate.onebuilding.org/`. Files containing hourly resolution of climate data are available in EnergyPlus Weather Format (EPW) format. The EPW and .cli (the CitySim input weather file) files are in different

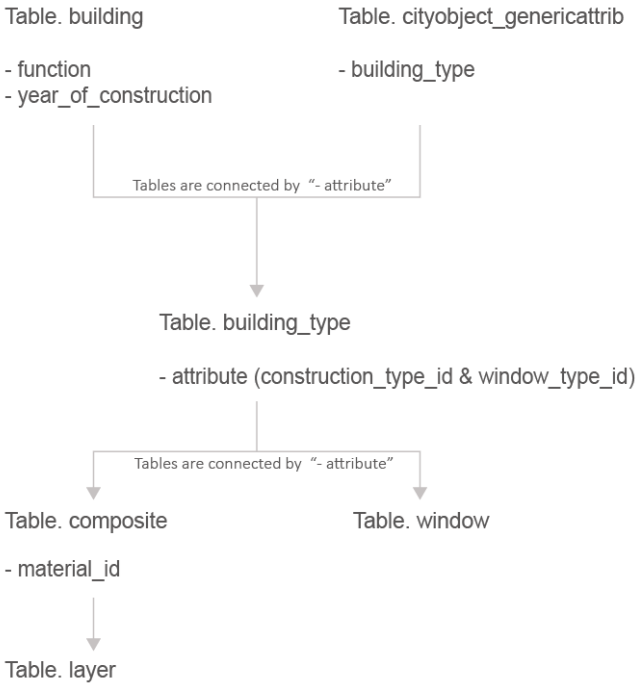



Figure 4.4.: The links between building geometry and building physics

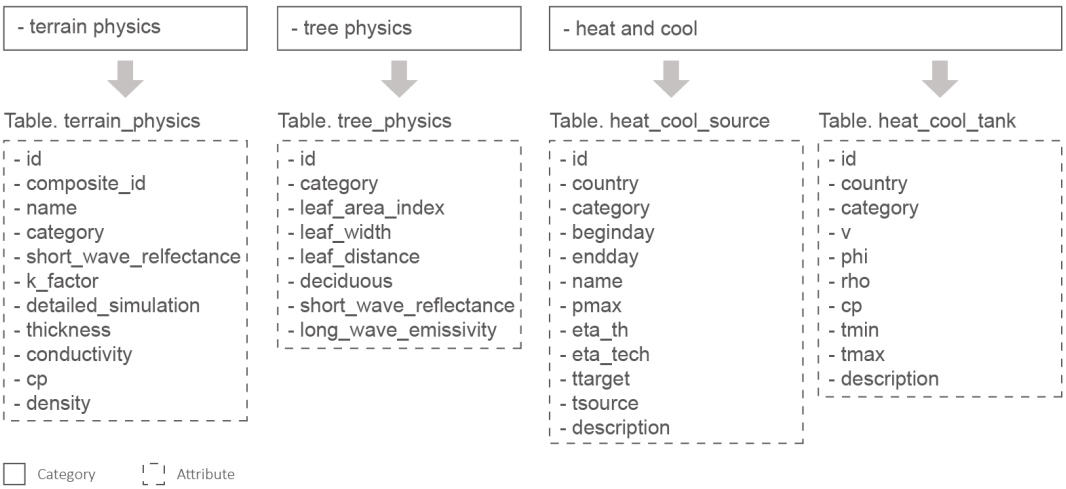



Figure 4.5.: The overview of parsing and storing other physics parameters

formats and do not contain the same attributes, but the EPW file contains all the information needed for generating a .cli file (see Table 4.3).

| | CLI file requirement | uom | EPW available | uom |
|---|---|---|---|---|
| d | Day | | Date – Day | |
| m | Month | | Date – Month | |
| h | Hour | | Time – Hour | |
| G_Dh | Diffuse horizontal irradiance | W/m2 | Diffuse Horizontal Radiation | Wh/m2 |
| G_Bn | Beam (solar) normal irradiance | W/m2 | Direct Normal Radiation | Wh/m2 |
| Ta | Air temperature | °C | Dry Bulb Temperature | °C |
| Ts | Ground surface temperature | °C | Dry Bulb Temperature | °C |
| FF | Wind Speed | m/s | Wind Speed | m/s |
| DD | Wind Direction | ° | Wind Direction | ° |
| RH | Relative Humidity | % | Relative Humidity | % |
| RR | Precipitation | mm | Liquid Precipitation Depth | mm |
| N | Nebulosity | Octas | Total Sky Cover | tenths |

Table 4.3.: The mapping between EPW file and .cli file

A python-based program is developed in order to extract data from EPW files and produce the .cli file. The process of the program is to read the EPW file, extract the required attribute information and map it to generate the .cli file. The .cli file content generated from EPW file in python is shown in figure 4.6).

A situation is also considered in the research where EPW files can be stored in database, and a python-based program is also developed for extracting the climate information from the database and generating the .cli file. The storage of a large number of EPW files in a database makes it possible to construct a climate database that is easy to organize and editing the climate information for urban energy simulation. Therefore, a storage schema for EPW file needs to be designed. The overview of parsing and storing EPW data in database is shown in figure 4.7.

| | dm | m | h | G_Dh | G_Bn | Ta | Ts | FF | DD | RH | RR | N |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 1 | 1 | 0 | 0 | 5.0 | 5.0 | 5.0 | 171 | 84 | 0.0 | 0 |
| **1** | 1 | 1 | 2 | 0 | 0 | 5.0 | 5.0 | 4.0 | 178 | 83 | 0.0 | 1 |
| **2** | 1 | 1 | 3 | 0 | 0 | 5.4 | 5.4 | 4.0 | 186 | 83 | 0.0 | 7 |
| **3** | 1 | 1 | 4 | 0 | 0 | 6.7 | 6.7 | 4.0 | 199 | 80 | 0.0 | 7 |
| **4** | 1 | 1 | 5 | 0 | 0 | 6.5 | 6.5 | 4.0 | 214 | 82 | 0.0 | 7 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **8755** | 365 | 12 | 20 | 0 | 0 | 6.9 | 6.9 | 6.0 | 220 | 86 | 0.0 | 8 |
| **8756** | 365 | 12 | 21 | 0 | 0 | 7.3 | 7.3 | 6.0 | 220 | 84 | 0.0 | 8 |
| **8757** | 365 | 12 | 22 | 0 | 0 | 7.8 | 7.8 | 6.0 | 220 | 84 | 0.0 | 8 |
| **8758** | 365 | 12 | 23 | 0 | 0 | 8.2 | 8.2 | 7.0 | 220 | 82 | 0.0 | 8 |
| **8759** | 365 | 12 | 24 | 0 | 0 | 8.4 | 8.4 | 7.0 | 220 | 83 | 0.0 | 8 |

8760 rows × 12 columns

Figure 4.6.: The .cli file generated from EPW file in python

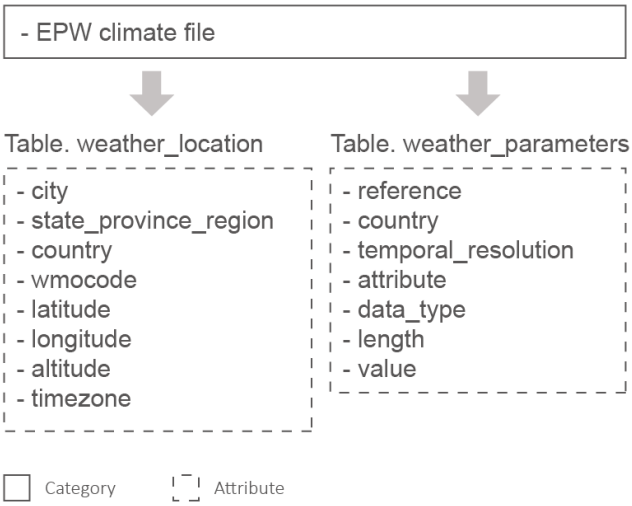



Figure 4.7.: Parse EPW files into tables to be stored in database

After the data preparation process, the database contains the 3DCityDB (which include all the geometry data and associated attributes), physics library and weather data library (see 4.8). This complete database will be used as the data source of python interface.

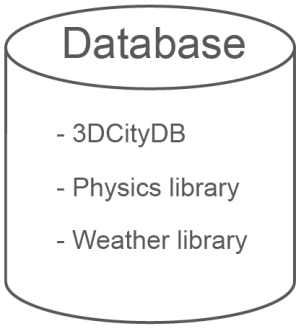



Figure 4.8.: The complete database for python interface development

# 5. PYTHON IMPLEMENTATION

With a complete database, next step is to develop the entire process in the python environment. This chapter will describe how the python-based bidirectional interface is developed to meet the goal of feeding and retrieving data between the 3DCityDB and CitySim.

## 5.1. Data Extraction And Pre-processing

After connecting the python environment to the database, the user can enter the preferred bounding box of the study area. The data inside the study area for CitySim urban energy simulation are read with python library *geopanda*. Those data include geometry data (buildings, trees and terrain), physics parameters and several building attributes (e.g. building function and building type). By merging the building geometry with attributes, the physics information can be fused with the building geometry. Those data are organized and stored in several dataframes for later use (i.e. spatial and non-spatial functions). The result of data extraction and pre-processing operations are two dataframes, one contains the geometry data of buildings, tree and terrain (for writing the geometry part of CitySim input file) and the other contains the physics and attributes information of buildings that are intended to be simulated (for writing the rest parameters part of CitySim input file). The overview of data extraction and pre-processing is shown in figure 5.1.

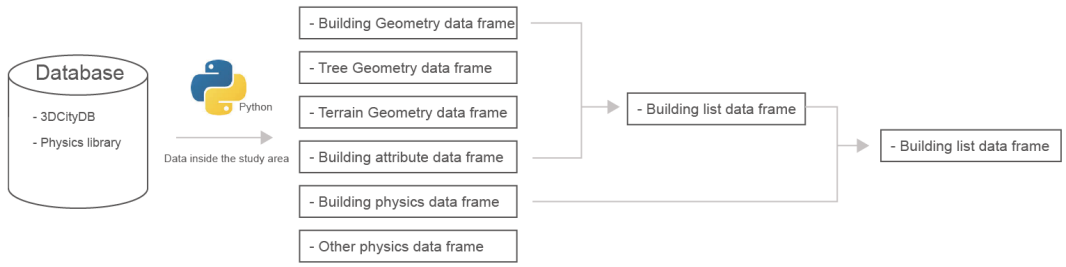



Figure 5.1.: The overview of data extraction and pre-processing

### 5.1.1. Connect to the database

The connection between the database and the python environment is established by the python library *sqlalchemy*. This library can create the database engine for extracting data from the connected database in python environment. Users need to type in the connection information and in this step, the file path of the climate file is also recorded. The example of how to connect a database to python environment is shown in figure 5.2.


Figure 5.2.: Example of establishing a connection between the database and the python environment

### 5.1.2. Geometry data extraction

Once connected to the database, the bounding box of the study area entered by the user (i.e. x_min, y_min, x_max, y_max, buffer) will be used for filtering geometries inside (by postgis function *ST_MakeEnvelope*). Combined with the python library *geopandas*, it is able to get the geometry dataframes by cross link multiple tables in 3DCityDB (e.g. the geometry envelope is from table *surface_geometry* and the gmlid is from table *cityobject*). By extending the study area with the input buffer, the trees and terrain around the buildings which affect the energy simulation results are also selected. In this way, the geometry data inside the study area can be obtained for CitySim energy simulation. For instance, the figure 5.3 shows the original geometry and the geometry after processing the study area selection.

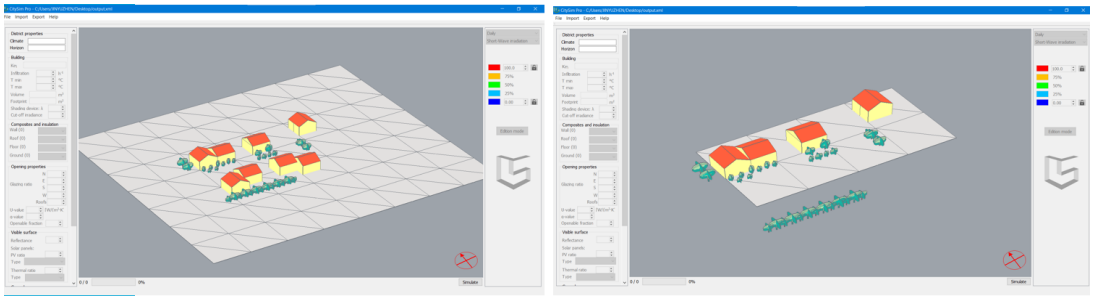


Figure 5.3.: Geometries shown in CitySim GUI; Left one: the original geometry bounding box as xmin 0 ymin -30 xmax 70 ymax 15; Right one: the selected geometry bounding box as xmin 0 ymin -20 xmax 70 ymax 15, buffer as 20m

In the processing mentioned above, linking several tables and selecting required data are made by SQL queries. For example, the extraction of building geometry is from varies tables such as *thematic_surface*, *surface_geometry* and *cityobject*, the detail query is as follows:

```
geometry_envelope = gpd.read_postgis(
"SELECT thematic_surface.objectclass_id,
thematic_surface.building_id, thematic_surface.lod2_multi_surface_id,
surface_geometry.gmlid, surface_geometry.geometry, cityobject.gmlid as
parent_gmlid, surface_geometry.cityobject_id
FROM thematic_surface
```

```
LEFT JOIN surface_geometry ON
thematic_surface.lod2_multi_surface_id = surface_geometry.parent_id
LEFT JOIN cityobject ON
thematic_surface.building_id = cityobject.id
WHERE surface_geometry.geometry && st_makeenvelope(%s,%s,%s,%s,%s)
ORDER BY parent_gmlid",
%(x_min_selection, y_min_selection, x_max_selection, y_max_selection, EPSG),
db_engine, geom_col = "geometry")
```

After processing via python libraries and SQL queries, the building, tree (tree geometry is stored as implicit geometries which are extracted differently from the building and terrain) and terrain geometry data inside the study area are well stored in dataframes. A glimpse of building geometry dataframes is shown in figure 5.4. It is worth mentioning that the python interface also shows the number of buildings selected inside the study area in case the user want to redefine the area to select different amount of buildings.

| | objectclass_id | building_id | lod2_multi_surface_id | gmlid | geometry | parent_gmlid | cityobject_id |
|---|---|---|---|---|---|---|---|
| 0 | 34 | 41 | 281 | id_building_1_polygon_4 | POLYGON Z ((0.00000 10.00000 0.00000, 0.00000 ... | id_building_01 | 50 |
| 1 | 36 | 41 | 275 | id_building_1_polygon_cs1 | POLYGON Z ((10.00000 10.00000 10.00000, 10.000... | id_building_01 | 48 |
| 2 | 35 | 41 | 269 | id_building_1_polygon_3 | POLYGON Z ((0.00000 0.00000 0.00000, 0.00000 1... | id_building_01 | 46 |
| 3 | 33 | 41 | 263 | id_building_1_polygon_2 | POLYGON Z ((5.00000 0.00000 15.00000, 10.00000... | id_building_01 | 44 |
| 4 | 33 | 41 | 260 | id_building_1_polygon_1 | POLYGON Z ((0.00000 0.00000 10.00000, 5.00000 ... | id_building_01 | 43 |

Figure 5.4.: The building geometry dataframe

### 5.1.3. Building attributes extraction

In addition to the geometry data, several attributes related to the building geometry are also needed. The purposes of those attributes information are:

- Filter out non-residential buildings that are not intended for energy simulation (by attribute *building function*).
- Provide information needed for the CitySim energy simulation (by attribute *lod2 volume* and *number of residents*).
- As the key to link the building geometry and the building physics parameters (by attribute *building type* and *year of construction*).

The number of inhabitants per building is also required in the CitySim energy simulation which will be calculated accompany with the occupancy profile (one of the physics parameters) for energy demand. However, 3D BAG 2.0 does not contain those information. In such case, assumptions need to be made from the information already available. For example, with the available information about building volume and storey height, it is able to calculate the floor area; by multiplying the floor area by the number of residents per square meter (with assumption), the number of inhabitants can be got. Although, it is not as accurate as reality, it provides a way to make assumptions for such situation. The function to calculate the number of residents of the building is defined as below:

```
def calculate_num_residents(lod2_volume,storeyHeightsAboveGround):
    floor_area = lod2_volume/storeyHeightsAboveGround
    density = 1/20 #1 person per 20 square meters
    num_residents = round(floor_area * density)
    return num_residents
```

By extraction and pre-processing presented above, a dataframe containing information about the building that will be simulated can be obtained. (see figure 5.5). This tables will be enriched with physics parameters discussed in the next subsection.

| | index | building_id | parent_gmlid | num_residents | lod2_volume | function | year_of_construction | building_type |
|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 41 | id_building_01 | 5.0 | 1250.0 | residential building | 1955.0 | SFH |
| **1** | 7 | 59 | id_building_02 | 21.0 | 1250.0 | residential building | 1955.0 | SFH |
| **3** | 23 | 24 | id_building_04 | 21.0 | 1250.0 | residential building | 1955.0 | MFH |
| **5** | 39 | 42 | id_building_06 | 6.0 | 1250.0 | residential building | 1997.0 | AB |
| **6** | 47 | 45 | id_building_07 | 110.0 | 1250.0 | residential building | 2005.0 | AB |
| **7** | 55 | 23 | id_building_08 | 5.0 | 1250.0 | residential building | 1920.0 | AB |
| **8** | 63 | 92 | id_building_11 | 12.0 | 1250.0 | residential building | 1920.0 | MFH |
| **9** | 70 | 83 | id_building_12 | 34.0 | 1250.0 | residential building | 1964.0 | MFH |
| **10** | 77 | 101 | id_buildingpart_09 | 20.0 | 1250.0 | residential building | 1965.0 | AB |
| **11** | 85 | 110 | id_buildingpart_10 | 25.0 | 1250.0 | residential building | 1940.0 | AB |

Figure 5.5.: The building list dataframe

### 5.1.4. Physics parameters extraction

The physics parameters are stored in the physics library in the database. Those information are extracted from five tables *building_type*, *composite*, *layer*, *profile* and *window*, which are integrated into a physics parameters dataframe in python. Those information are then fused with buildings according to the attribute *year_of_construction*, *building_type* and *building_function* (they have a mapping relationship), and finally integrated into the building list dataframe (see figure 5.6). As the types and categories of trees and terrain are not differentiated further in this study, the physics parameters on trees and terrain do not need to be integrated with the specific geometry entities.

## 5.2. Shading Surfaces

The shading surface section in CitySim input XML file is used for storing all shading objects. Since the non-residential buildings and all other ancillary buildings play a role as shading objects in the 3D urban scene, those buildings can be converted to shading surfaces for simulation. It is worth mentioning that shading objects only play a role in influencing the sunlight on the simulated building, so it is not necessary to use a high level of detail model for these objects (e.g. use LoD1 instead of LoD2). By modelling lower lever of detail models as shading surfaces, it can simplify the CitySim input XML file and reduce the simulation time. As

| | index | building_id | parent_gmlid | num_residents | lod2_volume | function | year_of_construction | building_type | system | country | year_initial | year_end | outwalls_constructiontypeid | outwalls_windowtype |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 41 | id_building_01 | 5.0 | 1250.0 | residential building | 1955.0 | SFH | TABULA | NL | 0 | 1964 | 204.0 | |
| 6 | 7 | 59 | id_building_02 | 21.0 | 1250.0 | residential building | 1955.0 | SFH | TABULA | NL | 0 | 1964 | 204.0 | |
| 12 | 23 | 24 | id_building_04 | 21.0 | 1250.0 | residential building | 1955.0 | MFH | TABULA | NL | 0 | 1964 | 204.0 | |
| 21 | 39 | 42 | id_building_06 | 6.0 | 1250.0 | residential building | 1997.0 | AB | TABULA | NL | 1992 | 2005 | 201.0 | |
| 27 | 47 | 45 | id_building_07 | 110.0 | 1250.0 | residential building | 2005.0 | AB | TABULA | NL | 1992 | 2005 | 201.0 | |
| 30 | 55 | 23 | id_building_08 | 5.0 | 1250.0 | residential building | 1920.0 | AB | TABULA | NL | 0 | 1964 | 204.0 | |
| 36 | 63 | 92 | id_building_11 | 12.0 | 1250.0 | residential building | 1920.0 | MFH | TABULA | NL | 0 | 1964 | 204.0 | |
| 42 | 70 | 83 | id_building_12 | 34.0 | 1250.0 | residential building | 1964.0 | MFH | TABULA | NL | 0 | 1964 | 204.0 | |
| 49 | 77 | 101 | id_buildingpart_09 | 20.0 | 1250.0 | residential building | 1965.0 | AB | TABULA | NL | 1965 | 1974 | 203.0 | |
| 54 | 85 | 110 | id_buildingpart_10 | 25.0 | 1250.0 | residential building | 1940.0 | AB | TABULA | NL | 0 | 1964 | 204.0 | |

Figure 5.6.: The final building list and building physics parametersframe

mentioned before, alderaan.gml contains a group of ancillary buildings which are modelled in LoD1. Those buildings inside the study area (plus the buffer) will be modelled as shading surfaces. Besides, for building modelled in LoD2 which are not selected for simulation (the non-residential buildings), their LoD1 geometry will be extracted and written as shading surfaces. The overall workflow of shading surfaces implementation is shown in figure 5.7 and processing out come is shown in figure 5.8.

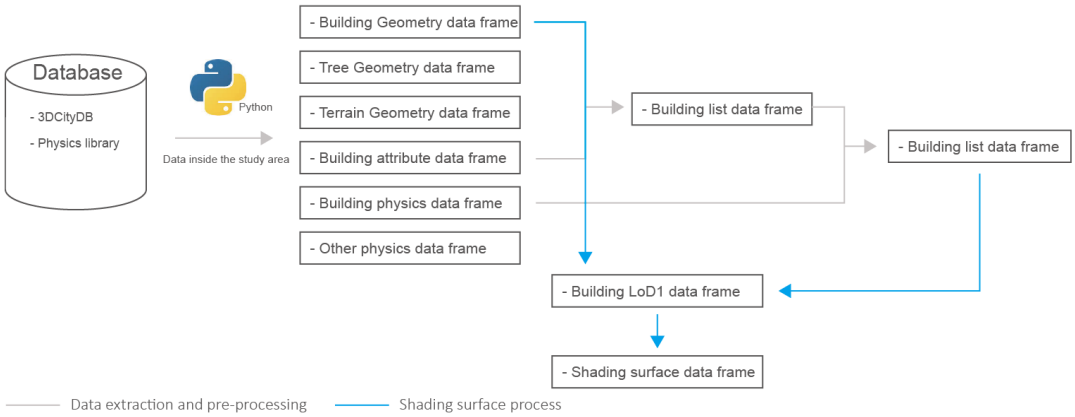



Figure 5.7.: The overview of shading surface processing

## 5.3. Terrain Processing

The terrain geometry data consists of a 3D triangular network, as mentioned in Section 3.3.2, the python interface is implemented as follows:

- **Extract the building footprint**. According to the CityGML mapping structure of 3DCityDB, the thematic surfaces of the building footprint has a distinct object class id (35). Based

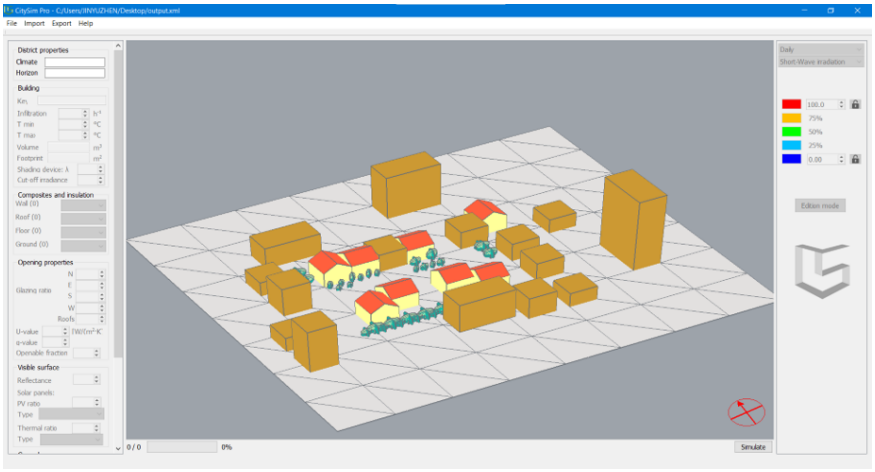


Figure 5.8.: The processing result shown in CitySim GUI, bounding box as xmin 0 ymin -30 xmax 70 ymax 15, buffer 50m

on this information, first, the simulated building footprint geometry will be selected from the geometry dataframe, and then those 3D building footprints are projected into 2D planes for next spatial processing.

- **Find the intersect building footprints and dissolve them into one polygon**. To avoid duplication of intersecting building footprints effecting the terrain processing, it is necessary to integrate the intersecting building footprints in advance. The check and processing of intersection of building footprints are made by python library *shapely*.

- **Project the 3D terrain geometry to 2D**. As mentioned before, the python library such as *trimesh* and *shapely* can only work with 2D objects. Thus, before the terrain processing, the 3D terrain geometry needs to be projected to 2D.

- **Crop and re-triangulate the terrain**. In this step, first, find the terrain that intersects the building footprints, then cut off the overlay part on the terrain geometry and re-triangulate the rest. It is worth noting that the re-triangulation can make a terrain surface to multiple triangle surfaces. Those surfaces are labeled with the same *id* as their 'parent'. In this way, the simulation results for each terrain fragment can still be got (by merging those terrain pieces) and stored in the database.

- **Transform the 2D terrain dataframe to 3D terrain dataframe**. As mentioned above, every processed terrain surface is labeled with same *id* as its 'parent' by the one to one relationship. In this step, first, identify the 3D terrain surface corresponding to the 2D terrain surface. Then, the z-value of the 2D terrain coordinates can be calculated by the 3D plane formula formed from the 3D terrain.

After the terrain processing, every terrain TIN surface will be simulated in CitySim which contributes to a better simulation result. The processed terrain is shown in figure 5.9 and 5.10.

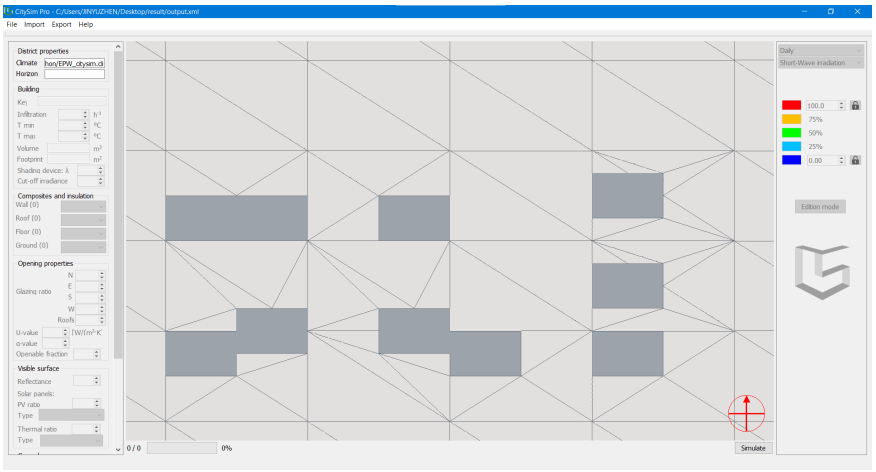


Figure 5.9.: The processed terrain shown in CitySim GUI

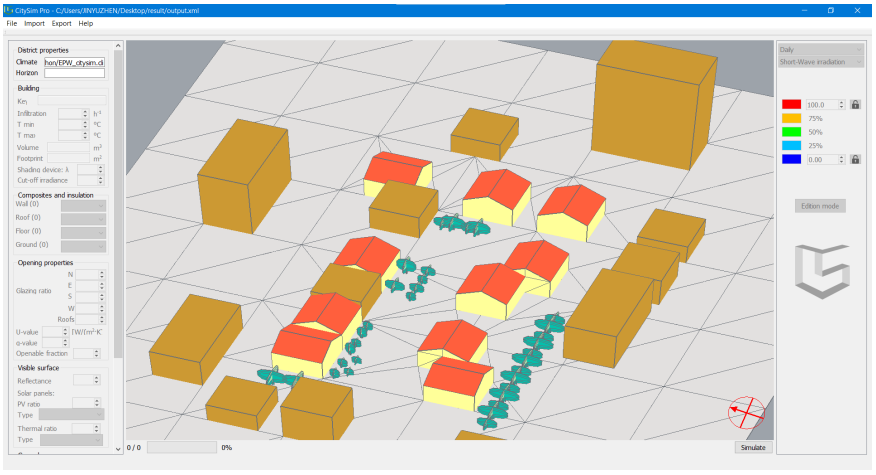


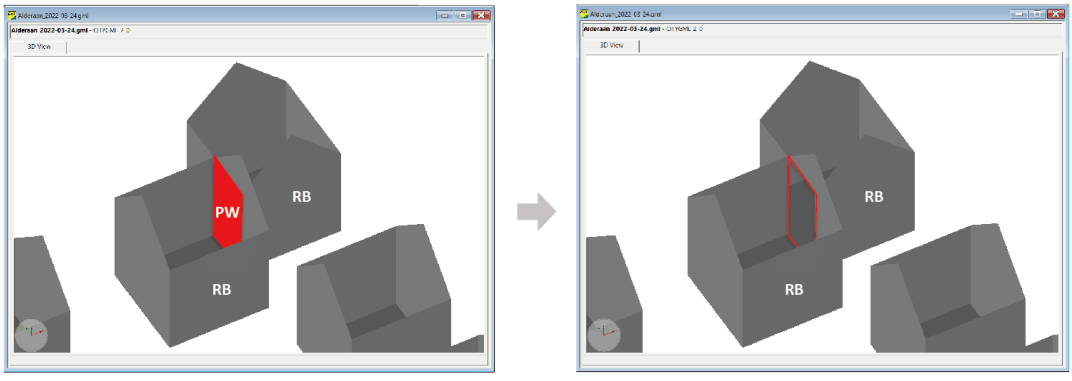
Figure 5.10.: The processed terrain with the rest geometries shown in CitySim GUI

## 5.4. Party Wall Processing

Since there is little heat transfer through the party wall of two adjacent buildings. To reduce the input file size and calculation time consuming, those walls can be identified and removed before the CitySim energy simulation. As mentioned in Section 4.2, the part wall information are stored in table *generalization* (stored as a one to one relationship, see figure 5.11). The first step is to determine the building adjacency. Those information can be obtained by merging the database table *generalization* with the geometry dataframe. After filtering out the party wall belonging to residential building, a geometry dataframe without party wall can be obtained. The figure 5.12 shows the aldraan.gml party walls before and after processing.

| | cityobject_id [PK] bigint | generalizes_to_id [PK] bigint |
|---|---|---|
| 1 | 15 | 59 |
| 2 | 16 | 42 |
| 3 | 18 | 24 |
| 4 | 33 | 45 |
| 5 | 34 | 6 |
| 6 | 50 | 59 |
| 7 | 53 | 7 |
| 8 | 71 | 23 |
| 9 | 74 | 6 |
| 10 | 75 | 41 |
| 11 | 106 | 110 |
| 12 | 115 | 101 |

Figure 5.11.: The party walls' relationship stored in table *generalization*




RB, residnetial building PW, party wall

Figure 5.12.: The alderaan.gml party wall processing result shwon in FZKViewer

## 5.5. Building Surface Processing

To take each surfaces valid for CitySim simulation, the concave building surfaces need to be re-triangulated, especially after the removal of party walls, where additional concave surfaces may exist. As mentioned in Section 3.3.3, the python interface is implemented as follows:

- **Check if the surface geometry is convex or not**. Based on python function, determine whether the building surfaces geometry are convex or not.
- **For concave surfaces, check if the surface is perpendicular to X-Y plane**. First, find the three points on the surface polygon that can form a plane formula, and then determine whether the plane is perpendicular to the X-Y plane.
- **Determine the projection direction to get the 2D surface**. While the concave surface is perpendicular to the X-Y plane, calculate the surface bounding box length on x-axis and y-axis. If the x-axis is larger than y-axis, then project the surface to X-Z plane, otherwise project the surface to Y-Z plane. For concave surfaces that are not perpendicular to the X-Y plane, project the surface to X-Y plane.
- **Re-triangulate the concave surface and transform the 2D surfaces back to 3D**. The implementation of re-triangulating and transforming back to 3D is the same as terrain processing.

With the building surface processing, every surface will be valid in the CitySim simulation. The re-triangulated building surfaces is highlighted shown in figure 5.13.


Figure 5.13.: The aldraan.gml building surface processing outcome shown in CitySim GUI

## 5.6. Write CitySim Input XML File

The writing of CitySim input XML file from the prepared dataframes above needs to strictly follow the required format structure. If the output file is missing required elements or con-

structed differently, the CitySim would report the reading error. To start with the header section, the file path of the climate file, the simulation period and general information need to be included. For example, they can be written as:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<CitySim name="alderaan">
<Simulation beginMonth="1" endMonth="12" beginDay="1" endDay="31"/>
<Climate location="C:/Climate_file/alderaan.cli"/>
```

### 5.6.1. Write the composite part

The composite part consists of construction material's physics information. The material of the simulated buildings, trees and terrain will all be written in this section. For each material, its output structure includes the *id*, *name*, and *tcategory*, followed by different layer information which includes attributes such as *thickness*, *conductivity*, *Cp* and *Density*, etc. Before writing this part, a dataframe for containing composite information is generated from physics parameters. From figure 5.14 can see three types of wall materials (wall 1.61, wall 0.36 and wall 1.45) with their layer information are well prepared. With this dataframe, for instance,

| | construction_id | construction_name | construction_category | id | system | country | material_id | attribute | data_type | value | uom | description | density | heat_capacity | conductiv |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 204.0 | Wall_1,61 | outWall | 280 | TABULA | NL | 0 | thickness | float | 0.200 | None | None | 2000.0 | 840.0 | 0.9 |
| 1 | 204.0 | Wall_1,61 | outWall | 281 | TABULA | NL | 23 | thickness | float | 0.011 | None | None | 105.0 | 1800.0 | 0.0 |
| 2 | 204.0 | Wall_1,61 | outWall | 282 | TABULA | NL | 62 | thickness | float | 0.000 | None | None | 1329.0 | 1014.0 | 0.7 |
| 3 | 201.0 | Wall_0,36 | outWall | 271 | TABULA | NL | 0 | thickness | float | 0.200 | None | None | 2000.0 | 840.0 | 0.9 |
| 4 | 201.0 | Wall_0,36 | outWall | 272 | TABULA | NL | 23 | thickness | float | 0.108 | None | None | 105.0 | 1800.0 | 0.0 |
| 5 | 201.0 | Wall_0,36 | outWall | 273 | TABULA | NL | 62 | thickness | float | 0.000 | None | None | 1329.0 | 1014.0 | 0.7 |
| 6 | 203.0 | Wall_1,45 | outWall | 277 | TABULA | NL | 0 | thickness | float | 0.200 | None | None | 2000.0 | 840.0 | 0.9 |
| 7 | 203.0 | Wall_1,45 | outWall | 278 | TABULA | NL | 23 | thickness | float | 0.014 | None | None | 105.0 | 1800.0 | 0.0 |
| 8 | 203.0 | Wall_1,45 | outWall | 279 | TABULA | NL | 62 | thickness | float | 0.000 | None | None | 1329.0 | 1014.0 | 0.7 |

Figure 5.14.: The dataframe of composite information

the composite section of one category material (wall 1.61) is written as:

```
<Composite id="204.0" name="Wall_1,61" category="outWall">
<Layer Thickness="0.2" Conductivity="0.96" Cp="840.0" Density="2000.0"
NRE="0" GWP="0" UBP="0"/>
<Layer Thickness="0.011" Conductivity="0.045" Cp="1800.0" Density="105.0"
NRE="0" GWP="0" UBP="0"/>
<Layer Thickness="0.0" Conductivity="0.79" Cp="1014.0" Density="1329.0"
NRE="0" GWP="0" UBP="0"/>
</Composite>
```

### 5.6.2. Write the profile part

In the profile part, the building occupancy information will be written. The occupancy day profile and year profile are constructed similarly and written in the section of *OccupancyDayProfile* and *OccupancyYearProfile* in CitySim XML input file. For occupancy day profile, 24 records for 24 hour need to be output. For occupancy year profile, 365 records for a 365 days need to be output. Those records are extracted from the database and stored in a profile dataframe in *array* type (see figure 5.15). By extracting those values from the array, the record can be written as follows:

| | profile_id | id | system | country | profile_type | element | function | data_type | length | value | description |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | TABULA | NL | occupancy | day | residential building | array | 24 | [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.8, 0.6, 0.4, ... | occupancy day profile of residential buildings |
| 1 | 1 | 1 | TABULA | NL | occupancy | year | residential building | array | 365 | [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, ... | occupancy year profile of residential buildings |

Figure 5.15.: The dataframe of occupancy profile information

```
<OccupancyDayProfile id="1" name="occupancy_day_profile_of_residential_buildings"
p1="1.0" p2="1.0" p3="1.0" p4="1.0" p5="1.0"
p6="1.0" p7="0.8" p8="0.6" p9="0.4" p10="0.4"
p11="0.4" p12="0.6" p13="0.8" p14="0.6" p15="0.4"
p16="0.4" p17="0.6" p18="0.8" p19="0.8" p20="0.8"
p21="0.8" p22="1.0" p23="1.0" p24="1.0" />

<OccupancyYearProfile id="1" name="occupancy_year_profile_of_residential_buildings"
d1="1.0" d2="1.0" d3="1.0" d4="1.0" d5="1.0"
d6="1.0" d7="1.0" d8="1.0" d9="1.0" d10="1.0"
d11="1.0" d12="1.0" d13="1.0" d14="1.0" d15="1.0"
d16="1.0" d17="1.0" d18="1.0" d19="1.0" d20="1.0"
d21="1.0" d22="1.0" d23="1.0" ..... d365="1.0" />
```

### 5.6.3. Write the building part

The building information is written in section *Building* in CitySim XML input file. This part consists of the building energy related attributes, heat and cool physics parameters, building physics parameters and building geometry. Apart from the information already stored in building list dataframe, the building physics information is linked to the building geometry by *gmlid*. The LoD2 building geometries are written by thematic surfaces polygon coordinates. Every vertexes coordinates of polygon will be written in one line. When writing coordinates, in order to reduce the loss of significance, the program will first ask the users to enter the number of decimal places they want to keep, then translate the coordinates according to the coordinates recorded of the bounding box. In addition, the type of the thematic surfaces is also required in writing this part. This is achieved by checking the geometry attribute *objectclass_id*. The *objectclass_id* 33, 34 (36) and 35 refers to the type Roof, Wall (Party Wall) and Floor. For examply, one LoD2 building geometry and related physics parameters are written as follow:

```
<Building id="0" key="id_building_01" Vi="1250.0" Ninf="0.1" BlindsLambda="0.2"
BlindsIrradianceCutOff="100" Simulate="true">
<HeatTank V="0.01" phi="20.0" rho="1000.0" Cp="4180.0" Tmin="20.0" Tmax="35.0"/>
<CoolTank V="0.01" phi="20.0" rho="1000.0" Cp="4180.0" Tmin="5.0" Tmax="20.0"/>
<HeatSource beginDay="258" endDay="135">
<Boiler name = "spaceX" Pmax="10000000.0" eta_th="0.95"/>
</HeatSource>
<CoolSource beginDay="136" endDay="257">
<HeatPump Pmax="10000000.0" eta_tech="0.3" Ttarget="5.0" Tsource="air"/>
</CoolSource>
<Zone id="0" volume="1000.0" psi="0" Tmin="20" Tmax="26" groundFloor="true"
nightVentilationBegin="0" nightVentilationEnd="0">
<Occupants n="5.0" sensibleHeat="90" sensibleHeatRadiantFraction="0.6"
latentHeat="0" type="1"/>
<Wall id="0" key="id_building_1_polygon_4" type="204.0" ShortWaveReflectance="0.3"
GlazingRatio="0.17" GlazingGValue="0.6" GlazingUValue="3.7" OpenableRatio="0">
<V0 x="-40.0" y="40.0" z="0.0"/>
<V1 x="-40.0" y="40.0" z="10.0"/>
```

```
<V2 x="-35.0" y="40.0" z="15.0"/>
<V3 x="-30.0" y="40.0" z="10.0"/>
<V4 x="-30.0" y="40.0" z="0.0"/>
</Wall>
......
<Roof id="3" key="id_building_1_polygon_2" type="166.0" ShortWaveReflectance="0.2"
GlazingRatio="0.0" GlazingGValue="0.6" GlazingUValue="3.7" OpenableRatio="0" kFactor="0">
<V0 x="-35.0" y="30.0" z="15.0"/>
<V1 x="-30.0" y="30.0" z="10.0"/>
<V2 x="-30.0" y="40.0" z="10.0"/>
<V3 x="-35.0" y="40.0" z="15.0"/>
</Roof>
......
<Floor id="2" key="id_building_1_polygon_3" type="132.0" ShortWaveReflectance="0.0"
GlazingRatio="0.0" GlazingGValue="0" GlazingUValue="0" OpenableRatio="0">
<V0 x="-40.0" y="30.0" z="0.0"/>
<V1 x="-40.0" y="40.0" z="0.0"/>
<V2 x="-30.0" y="40.0" z="0.0"/>
<V3 x="-30.0" y="30.0" z="0.0"/>
</Floor>
......
</Zone>
```

### 5.6.4. Write the shading surface part

The shading surface information is written in tag *ShadingSurface* in CitySim XML input file. The writing of the shading surface geometry is similar to the building part. The LoD1 geometry will be written in the same structure as the thematic surfaces. The same number of decimal places and coordinate transformation as building part will be applied. For example, one LoD1 building as shading surfaces can be written as follow:

```
<ShadingSurface>
<Surface id="0" ShortWaveReflectance="0.2">
<V0 x="-20.0" y="-20.0" z="5.0"/>
<V1 x="0.0" y="-20.0" z="5.0"/>
<V2 x="0.0" y="-10.0" z="5.0"/>
<V3 x="-20.0" y="-10.0" z="5.0"/>
</Surface>
......
</ShadingSurface>
```

### 5.6.5. Write the tree part

The tree data is written in tag *Trees* in CitySim XML input file. Every tree geometry component (polygons) and physics parameters will be written. Those polygons are labeled as *leaf*. Worth mentioning, the attribute *id* written in this part is important for associating simulation results with identities in database (the *id* will also be recorded in the simulation results). The same number of decimal places and coordinate transformation as building part will be applied. The tree part is written as follow:

```
<Trees>
<Tree id="240" name="Maple" leafAreaIndex="3.0" leafWidth="0.1" leafDistance="1.0"
deciduous="false">
<Leaf id="240" ShortWaveReflectance="0.3" LongWaveEmissivity="0.95">
<V0 x="14.791" y="20.0" z="3.283"/>
```

```
<V1 x="14.791" y="20.0" z="0.0"/>
<V2 x="15.209" y="20.0" z="0.0"/>
<V3 x="15.209" y="20.0" z="3.283"/>
<V4 x="15.0" y="20.0" z="3.2"/>
</Leaf>
<Leaf id="240" ShortWaveReflectance="0.3" LongWaveEmissivity="0.95">
<V0 x="15.0" y="19.791" z="3.283"/>
<V1 x="15.0" y="19.791" z="0.0"/>
<V2 x="15.0" y="20.209" z="0.0"/>
<V3 x="15.0" y="20.209" z="3.283"/>
<V4 x="15.0" y="20.0" z="3.2"/>
</Leaf>
<Leaf id="240" ShortWaveReflectance="0.3" LongWaveEmissivity="0.95">
<V0 x="15.0" y="19.791" z="3.283"/>
<V1 x="15.0" y="20.0" z="3.2"/>
<V2 x="15.0" y="20.209" z="3.283"/>
<V3 x="15.0" y="21.771" z="3.903"/>
<V4 x="15.0" y="22.505" z="5.6"/>
<V5 x="15.0" y="21.771" z="7.297"/>
<V6 x="15.0" y="20.0" z="8.0"/>
<V7 x="15.0" y="18.229" z="7.297"/>
<V8 x="15.0" y="17.495" z="5.6"/>
<V9 x="15.0" y="18.229" z="3.903"/>
</Leaf>
<Leaf id="240" ShortWaveReflectance="0.3" LongWaveEmissivity="0.95">
<V0 x="14.791" y="20.0" z="3.283"/>
<V1 x="15.0" y="20.0" z="3.2"/>
<V2 x="15.209" y="20.0" z="3.283"/>
<V3 x="16.771" y="20.0" z="3.903"/>
<V4 x="17.505" y="20.0" z="5.6"/>
<V5 x="16.771" y="20.0" z="7.297"/>
<V6 x="15.0" y="20.0" z="8.0"/>
<V7 x="13.229" y="20.0" z="7.297"/>
<V8 x="12.495" y="20.0" z="5.6"/>
<V9 x="13.229" y="20.0" z="3.903"/>
</Leaf>
</Tree>
......
</Trees>
```

### 5.6.6. Write the terrain part

The terrain information is written in tag *GroundSurface*. Every terrain triangular polygon and their associate physics information are recorded. The attributes *id* and *key* will be stored in simulation results for associating simulation results with identities in database. The same number of decimal places and coordinate transformation as building part will be applied. The terrain part is written as follow:

```
<GroundSurface>
<Ground id="677" key="ID_d5a47bcc-398d-4d4a-8b28-a5d1bbbbadf1"
ShortWaveReflectance="0.3" type="999" kFactor="0.7" detailedSimulation="true">
<V0 x="20.0" y="30.0" z="0.0"/>
<V1 x="20.0" y="50.0" z="0.0"/>
<V2 x="0.0" y="50.0" z="0.0"/>
</Ground>
......
</GroundSurface>
```

By the writing above, a complete CitySim input XML file is ready. In order to keep all files organized, this file is best to be stored in a folder together with the simulation results later on. Apart from running the simulation in python environment, the input XML file can also optionally be loaded into CitySim Pro GUI and the user can start the simulation manually using the CitySim Pro GUI (see figure 5.16).

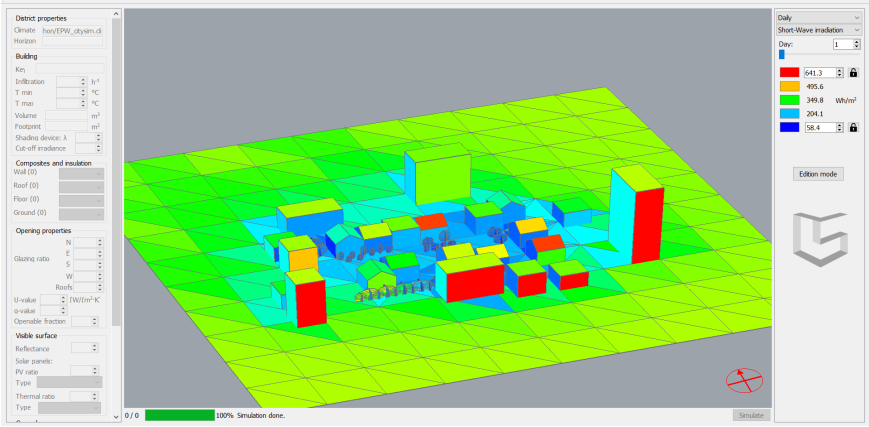


Figure 5.16.: Running the simulation in CitySim Pro GUI

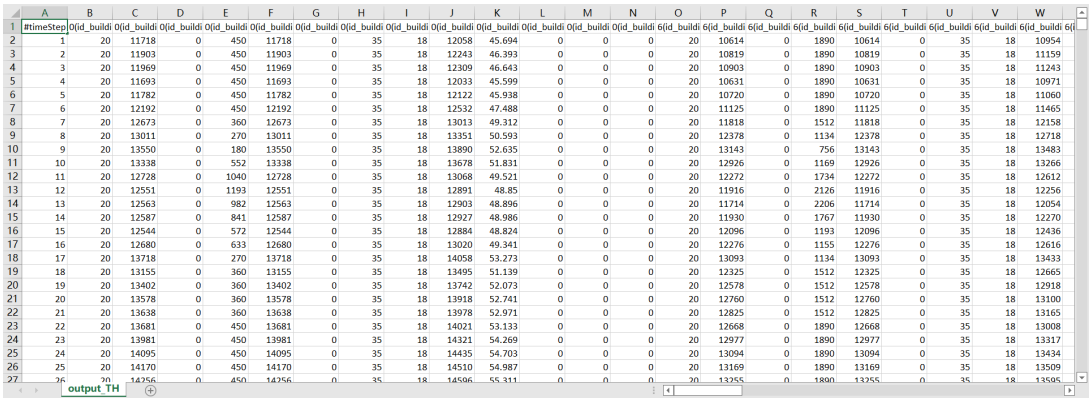
## 5.7. CitySim Simulation And Result Storage

In this section, the detail implementation of calling CitySim to perform energy simulation and simulation result manipulation in python envronment will be discussed.

### 5.7.1. Call CitySim and run the simulation

The calling of CitySim is made by python library *subprocess*. The subprocess module allows the python program to spawn new processes, connect to their input/output pipes, and obtain the return codes. The user needs to type in the file path of CitySim solver and the XML input file. Then the simulation will start automatically. The CitySim simulation progress will be shown in the python console. Once the simulation is finished, the result .tsv files containing all the energy related data produced by CitySim simulation are stored in the same path as the XML input file. Among those result files, one file containing the information we need and for result storage illustration is the '_TH' file (see figure 5.17).

The '_TH' file contains twelve simulated parameters correspond to each building (or layer). Those are *Internal temperature Ta*, *Heating*, *Cooling*, *Qi*, *Qs*, *VdotVent*, *HeatStockTemperature*, *ColdStockTemperature*, *MachinePower*, *FuelConsumption*, *ElectricConsumption*, *SolarThermalProduction*. For each parameters there are 8760 hourly records corresponding to 8760 hours during a typical year. If users need the monthly or yearly result, the data need to be summed based on time period. For illustration purpose, in this research the result storage will focus on *Qs* data representing the hourly total heating and cooling demand for each building.



| #timeStep | 0(id_buildi | 0(id_buildi | 0(id_buildi | 0(id_buildi | 0(id_buildi | 0(id_buildi | 0(id_buildi | 0(id_buildi | 0(id_buildi | 0(id_buildi | 0(id_buildi | 0(id_buildi | 0(id_buildi | 6(id_buildi | 6(id_buildi | 6(id_buildi | 6(id_buildi | 6(id_buildi | 6(id_buildi | 6(id_buildi | 6(id_buildi | 6(id_buildi |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 20 | 11718 | 0 | 450 | 11718 | 0 | 35 | 18 | 12058 | 45.694 | 0 | 0 | 0 | 20 | 10614 | 0 | 1890 | 10614 | 0 | 35 | 18 | 10954 |
| 2 | 20 | 11903 | 0 | 450 | 11903 | 0 | 35 | 18 | 12243 | 46.393 | 0 | 0 | 0 | 20 | 10819 | 0 | 1890 | 10819 | 0 | 35 | 18 | 11159 |
| 3 | 20 | 11969 | 0 | 450 | 11969 | 0 | 35 | 18 | 12309 | 46.643 | 0 | 0 | 0 | 20 | 10903 | 0 | 1890 | 10903 | 0 | 35 | 18 | 11243 |
| 4 | 20 | 11693 | 0 | 450 | 11693 | 0 | 35 | 18 | 12033 | 45.599 | 0 | 0 | 0 | 20 | 10631 | 0 | 1890 | 10631 | 0 | 35 | 18 | 10971 |
| 5 | 20 | 11782 | 0 | 450 | 11782 | 0 | 35 | 18 | 12122 | 45.938 | 0 | 0 | 0 | 20 | 10720 | 0 | 1890 | 10720 | 0 | 35 | 18 | 11060 |
| 6 | 20 | 12192 | 0 | 450 | 12192 | 0 | 35 | 18 | 12532 | 47.488 | 0 | 0 | 0 | 20 | 11125 | 0 | 1890 | 11125 | 0 | 35 | 18 | 11465 |
| 7 | 20 | 12673 | 0 | 360 | 12673 | 0 | 35 | 18 | 13013 | 49.312 | 0 | 0 | 0 | 20 | 11818 | 0 | 1512 | 11818 | 0 | 35 | 18 | 12158 |
| 8 | 20 | 13011 | 0 | 270 | 13011 | 0 | 35 | 18 | 13351 | 50.593 | 0 | 0 | 0 | 20 | 12378 | 0 | 1134 | 12378 | 0 | 35 | 18 | 12718 |
| 9 | 20 | 13550 | 0 | 180 | 13550 | 0 | 35 | 18 | 13890 | 52.635 | 0 | 0 | 0 | 20 | 13143 | 0 | 756 | 13143 | 0 | 35 | 18 | 13483 |
| 10 | 20 | 13338 | 0 | 552 | 13338 | 0 | 35 | 18 | 13678 | 51.831 | 0 | 0 | 0 | 20 | 12926 | 0 | 1169 | 12926 | 0 | 35 | 18 | 13266 |
| 11 | 20 | 12728 | 0 | 1040 | 12728 | 0 | 35 | 18 | 13068 | 49.521 | 0 | 0 | 0 | 20 | 12272 | 0 | 1734 | 12272 | 0 | 35 | 18 | 12612 |
| 12 | 20 | 12551 | 0 | 1193 | 12551 | 0 | 35 | 18 | 12891 | 48.85 | 0 | 0 | 0 | 20 | 11916 | 0 | 2126 | 11916 | 0 | 35 | 18 | 12256 |
| 13 | 20 | 12563 | 0 | 982 | 12563 | 0 | 35 | 18 | 12903 | 48.896 | 0 | 0 | 0 | 20 | 11714 | 0 | 2206 | 11714 | 0 | 35 | 18 | 12054 |
| 14 | 20 | 12587 | 0 | 841 | 12587 | 0 | 35 | 18 | 12927 | 48.986 | 0 | 0 | 0 | 20 | 11930 | 0 | 1767 | 11930 | 0 | 35 | 18 | 12270 |
| 15 | 20 | 12544 | 0 | 572 | 12544 | 0 | 35 | 18 | 12884 | 48.824 | 0 | 0 | 0 | 20 | 12096 | 0 | 1193 | 12096 | 0 | 35 | 18 | 12436 |
| 16 | 20 | 12680 | 0 | 633 | 12680 | 0 | 35 | 18 | 13020 | 49.341 | 0 | 0 | 0 | 20 | 12276 | 0 | 1155 | 12276 | 0 | 35 | 18 | 12616 |
| 17 | 20 | 13718 | 0 | 270 | 13718 | 0 | 35 | 18 | 14058 | 53.273 | 0 | 0 | 0 | 20 | 13093 | 0 | 1134 | 13093 | 0 | 35 | 18 | 13433 |
| 18 | 20 | 13155 | 0 | 360 | 13155 | 0 | 35 | 18 | 13495 | 51.139 | 0 | 0 | 0 | 20 | 12325 | 0 | 1512 | 12325 | 0 | 35 | 18 | 12665 |
| 19 | 20 | 13402 | 0 | 360 | 13402 | 0 | 35 | 18 | 13742 | 52.073 | 0 | 0 | 0 | 20 | 12578 | 0 | 1512 | 12578 | 0 | 35 | 18 | 12918 |
| 20 | 20 | 13578 | 0 | 360 | 13578 | 0 | 35 | 18 | 13918 | 52.741 | 0 | 0 | 0 | 20 | 12760 | 0 | 1512 | 12760 | 0 | 35 | 18 | 13100 |
| 21 | 20 | 13638 | 0 | 360 | 13638 | 0 | 35 | 18 | 13978 | 52.971 | 0 | 0 | 0 | 20 | 12825 | 0 | 1512 | 12825 | 0 | 35 | 18 | 13165 |
| 22 | 20 | 13681 | 0 | 450 | 13681 | 0 | 35 | 18 | 14021 | 53.133 | 0 | 0 | 0 | 20 | 12668 | 0 | 1890 | 12668 | 0 | 35 | 18 | 13008 |
| 23 | 20 | 13981 | 0 | 450 | 13981 | 0 | 35 | 18 | 14321 | 54.269 | 0 | 0 | 0 | 20 | 12977 | 0 | 1890 | 12977 | 0 | 35 | 18 | 13317 |
| 24 | 20 | 14095 | 0 | 450 | 14095 | 0 | 35 | 18 | 14435 | 54.703 | 0 | 0 | 0 | 20 | 13094 | 0 | 1890 | 13094 | 0 | 35 | 18 | 13434 |
| 25 | 20 | 14170 | 0 | 450 | 14170 | 0 | 35 | 18 | 14510 | 54.987 | 0 | 0 | 0 | 20 | 13169 | 0 | 1890 | 13169 | 0 | 35 | 18 | 13509 |
| 26 | 20 | 14256 | 0 | 450 | 14256 | 0 | 35 | 18 | 14596 | 55.311 | 0 | 0 | 0 | 20 | 13255 | 0 | 1890 | 13255 | 0 | 35 | 18 | 13595 |

output_TH

Figure 5.17.: CitySim simulation result TH.tsv in Excel window

## 5.7.2. Simulation result storage

The simulation result storage is based on the Energy ADE 2.0 storage schema. According to the Energy ADE Core Unified Modeling Language (UML) diagram, the energy demand of each building can be stored which is structured as feature *EnergyDemand* (see figure 5.18).

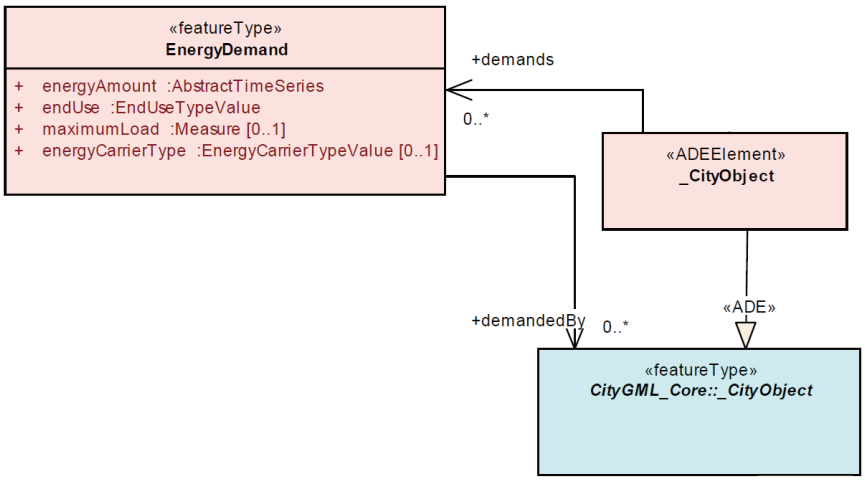



Figure 5.18.: Part of the UML diagram of the Energy ADE Core

As mentioned in Section 5.7.1, the '_TH' file contains 8760 hourly records of total heating and cooling demand (*Qs* value) for each simulated building. The result storage workflow is carried out as follows:

- **Extract the hourly *Qs* value from the '_TH' file**. Read the file in python environment and filter out the *Qs* data for each simulated building.
- **Calculate the monthly and yearly *Qs* value**. For potential needs of monthly and yearly *Qs* data, the calculation is made in python. The result will contain 12 monthly *Qs* value and 1 yearly *Qs* value for each simulated building.

- **Organize data into dataframes**. As mentioned above, the simulation result can be stored based on feature *EnergyDemand*. There are five tables in database (from Energy ADE 2.0) relating to this storage schema which are *cityobject*, *ng_timeseries*, *ng_regulartimeseries*, *ng_cityobject* and *ng_energydemand*. In this step, 5 dataframes containing *Qs* realted information are created with the same structure of those 5 tables.
- **Insert the dataframes into database**. Insert the 5 dataframes into the corresponding 5 tables in 3DCityDB.

By the above operation, the hourly, monthly and yearly *Qs* value of each simulated building is stored in the database. Figure 5.19 shows the table *ng_regulartimeseries* in database which contains the result *Qs* value of 10 simulated buildings in Alderaan.gml.

| | id [PK] bigint | timeinterval numeric | timeinterval_factor integer | timeinterval_radix integer | timeinterval_unit character varying (1000) | timeperiodprop_beginposition timestamp with time zone | timeperiodproper_endposition timestamp with time zone | values_ text |
|---|---|---|---|---|---|---|---|---|
| 1 | 275 | 1 | [null] | [null] | hour | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 11716 11900 11966 11691 11780 12189 12670 1 |
| 2 | 277 | 1 | [null] | [null] | month | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 10878.657 8812.929 8924.511 4681.693 1639.64 |
| 3 | 279 | 1 | [null] | [null] | year | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 61056.352 |
| 4 | 281 | 1 | [null] | [null] | hour | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 10614 10819 10903 10631 10720 11124 11817 1 |
| 5 | 283 | 1 | [null] | [null] | month | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 10327.964 8231.185 8260.245 4126.751 1369.68 |
| 6 | 285 | 1 | [null] | [null] | year | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 56093.055 |
| 7 | 287 | 1 | [null] | [null] | hour | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 13724 13946 13979 13496 13622 14250 15123 1 |
| 8 | 289 | 1 | [null] | [null] | month | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 12867.419 10303.628 10352.846 5296.734 1799. |
| 9 | 291 | 1 | [null] | [null] | year | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 70396.426 |
| 10 | 293 | 1 | [null] | [null] | hour | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 7145 7189 7083 6622 6712 7216 7725 7984 850 |

Figure 5.19.: Table *ng_regulartimeseries* in database contains the Alderaan simulation result

To store the simulation results of other categories, the same workflow mentioned above can be applied. For example, the solar irradiance simulation result (in the '_SW' file ) of each surfaces can also be stored in the database based on the Energy ADE 2.0 storage schema. Different from the energy demand like *Qs* value, the storage of solar irradiance result is structured as feature *WeatherData* and stored in tables *cityobject*, *ng_timeseries*, *ng_regulartimeseries*, *ng_cityobject* and *ng_weatherdata*. Detail implementation see the python script in Section 6.2.

# 6. RESULT ANALYSIS, REFLECTION AND FUTURE WORK

## 6.1. Testing The Python Interface

The python interface is developed based on the test dataset Alderaan.gml. To examine the adaptability of the interface to dataset from 3D BAG 2.0, the RijssenHolten.gml (from 3D BAG 2.0) is used for testing. After processing in the python interface, it proved that it can be run without problems. However, due to the large number of vertexes on building footprints, the terrain geometry pre-processing doesn't function well. This is due to the fact that in a real dataset, there will be more complex vertexes on the building footprints.Functions related to spatial processing can cause erroneous processing results when dealing with close and dense vertexes due to insufficient processing precision. In this case, in order to generate a valid input file and obtain relatively accurate simulation results, the python interface gives user the option to skip the terrain geometry pre-processing. Instead, the original terrain is written in CitySim XML input file for simulation. After the simulation, the result data is also well stored back in the database. One of the energy simulation result of RijssenHolten.gml (study area as xmin 231952 ymin 479844 xmax 232266 ymax 479944, buffer 20m) viewed in CitySim Pro is shown in figure 6.1 and 6.2.

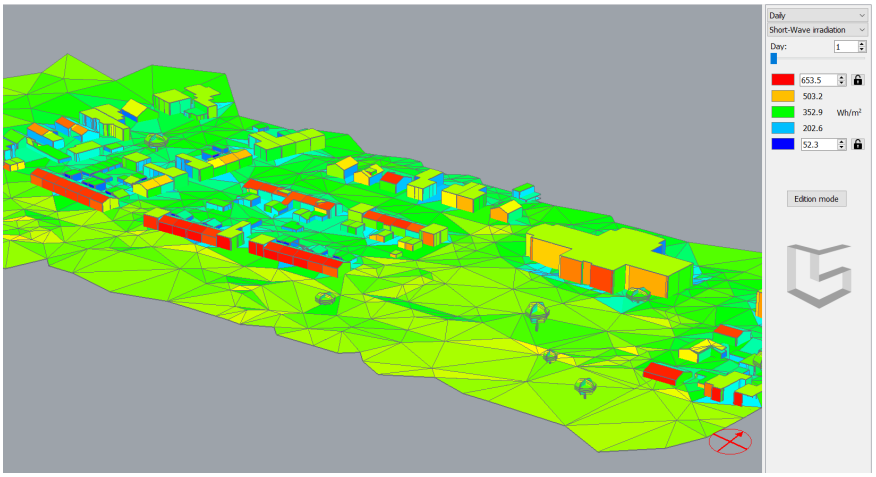



Figure 6.1.: The CitySim simulation result of short wave irradiance of RijssenHolten.gml viewed in CitySim Pro GUI

| | id [PK] bigint | timeinterval numeric | timeinterval_factor integer | timeinterval_radix integer | timeinterval_unit character varying (1000) | timeperiodprop_beginposition timestamp with time zone | timeperiodproper_endposition timestamp with time zone | values_ text |
|---|---|---|---|---|---|---|---|---|
| 1 | 142833 | 1 | [null] | [null] | hour | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 5814 5889 5867 5602 5662 5982 6334 6544 69 |
| 2 | 142835 | 1 | [null] | [null] | month | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 5298.71 4289.198 4353.088 2347.889 817.633 |
| 3 | 142837 | 1 | [null] | [null] | year | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 29514.317 |
| 4 | 142839 | 1 | [null] | [null] | hour | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 5783 5857 5833 5570 5631 5948 6298 6507 68 |
| 5 | 142841 | 1 | [null] | [null] | month | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 5276.545 4272.766 4335.081 2335.581 817.05 |
| 6 | 142843 | 1 | [null] | [null] | year | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 29396.608 |
| 7 | 142845 | 1 | [null] | [null] | hour | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 5821 5894 5869 5602 5663 5984 6336 6546 69 |
| 8 | 142847 | 1 | [null] | [null] | month | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 5305.105 4296.968 4359.176 2352.559 824.423 |
| 9 | 142849 | 1 | [null] | [null] | year | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 29575.321 |
| 10 | 142851 | 1 | [null] | [null] | hour | 2022-01-01 00:00:00+08 | 2022-12-31 11:59:59+08 | 5744 5819 5798 5536 5597 5912 6261 6472 68 |

Figure 6.2.: Table *ng_regulartimeseries* in database contains the RijssenHolten simulation result

## 6.2. Guidelines For The Python Interface

The script of the python interface along with physics library and weather library backup files are posted on GitHub.

`https://github.com/yuzhenjin3000/Dynamic-energy-simulations-based-on-the-3D-BAG-2.0.git`.

Notice:

- This python interface is developed with Python 3 and adapted to the CitySim (v. 22.05.2022) 3DCityDB (v. 5.0.0), Energy ADE (v. 2.0) and PostgreSQL (V. 13).
- The physics library and weather library storage schemas along with the data can be directly installed by importing the SQL files into the database.
- The geometry GML file can be imported into the database via software 3DCityDB importer/exporter.

## 6.3. Reflection And Future Work

As environmental issues receive more and more attention, it makes more sense to contribute in this area. In this research, the development of python-based interface well achieved the goal of linking the 3DCityDB and CitySim based on 3D BAG 2.0 in order to allow for a seamless flow of information and perform energy simulation. In general, the development of python-based bidirectional interface successfully manage the flow of information between 3DCityDB and CitySim. Based on 3D BAG 2.0, it is able to extract any study area in the Netherland for conducting energy simulation via the python interface. The design concepts and python interface development details of data managing or mapping (like the storage schema design, spatial and non-spatial functions and writing CitySim input XML file) could be useful for other researchers who would like to explore further. Compared to the current way of urban energy simulation (with CitySim and 3DCityDB), the python interface not only automates the data feeding and retrieving process, but also able to store the energy simulation results back into the database. The design of physics and weather library storage schema makes physics parameters and weather data along with geometry data (stored in 3DCityDB) stored in one database for efficient management and future use.

During the processing of spatial geometry, varies python libraries are used, such as *trimesh* and *shapely*. Since those libraries are suitable for handling 2D geometries, several projections from 3D (which is actually 2.5D) to 2D are carried out in geometry pre-processing (i.e. the terrain and building surfaces pre-processing). Besides, because those libraries cannot handle

surfaces with inner rings, the inner rings (i.e. projections of building footprints) are treated as polygons and removed in geometry pre-processing. Those features limit the adaptability of the python interface when dealing with real 3D geometries. And also increase the complexity of spatial processing which increases the running time of the program. Those problems could be addressed by using more powerful geometry processing libraries, such as CGAL which provides more geometric algorithms (it is in the form of a C++ library). Besides, the spatial handling process and the data organization can also be simplified to increase the overall performance in the future.

When testing the python interface adaptability to the real 3D BAG 2.0 dataset, the complex geometries resulting in failure of processing is also an issue to be fixed. When dealing with tight vertexes, the triangulation of cut terrain surfaces causes computational errors. Also, tiny surfaces are not covered when removing the overlapped surfaces. It is not the problem of 3D BAG 2.0 dataset (all geometry in 3D BAG 2.0 are valid). When dealing with real complex geometries, those issues always happen. To fix this problem, a pre-processing of all geometries could be carried out to clean up and simplify the objects. By removing tiny surfaces and tight vertexes, it reduces the probability of computational errors and also it will not affect the simulation results.

For potential users who are not familiar with python, a graphical user interface could be developed in the future. Through a user-friendly graphical interface, users can obtain energy simulation results by simply entering the database name, weather file path, study area bounding box, preferred coordinate decimal places, and the result data they want to store.

The CitySim and 3DCityDB as well as all other tools and data formats included in this study, are constantly being developed and optimized. The developed python interface should also be constant update to be well adapted for those elements. The research carried out in the thesis is meant for giving an idea of how those elements could be improved to meet the demand of potential users for urban energy simulation. If it can give a hint to the CitySim developer or someone who is working on urban energy simulation domain, the purpose of the research is achieved. Hopefully, more and more people will find it easy to implement urban energy simulations, which helps them better control and address the environmental issues.

# A. PHYSICS LIBRARY DATA SHEET IN THE DATABASE

Table A.1

| Schema | Table | Attribute | Type | Description | Example |
|---|---|---|---|---|---|
| physics | building_type | id | int | The index of each row | 0 |
| | | system | str | The system where the building physics information from | TABULA |
| | | country | str | Countries to which building physics applies | NL |
| | | year_initial | int | Starting year of building construction | 1965 |
| | | year_end | int | End year of building construction | 1974 |
| | | construction_period | numrange | The period of building construction | [1965,1974] |
| | | building_type | str | The type of the building (e.g. SFH, single family house) | SFH |
| | | function | str | The function of the building | residential |
| | | element | str | The condition of refurbishment | Medium -Refurbishment |
| | | attribute | str | All attributes relate to building physics | outWalls_shortWave -Reflectance |
| | | data_type | str | The data type of attributes (e.g. for outWalls_shortWaveReflectance) | float |
| | | value | float | The value of attributes (e.g. for outWalls_shortWaveReflectance) | 0.3 |
| | | uom | str | The unit of measure of attributes | - |
| | | description | str | The description of each row | - |
| | composite | id | int | The index of each row | 0 |
| | | system | str | The system where the building physics information from | TABULA |
| | | country | str | Countries to which building physics applies | NL |
| | | construction_id | int | Link to the table building_type attributes '_constructionTypeId' | 0 |
| | | construction_category | str | The construction object | outWall |
| | | construction_name | str | The construction material name | Aerated concrete-25cm |
| | | material_id | str | The id of construction material | 62 |
| | | attritube | str | The thickness of construction material | thickness |

| Schema | Table | Attribute | Type | Description | Example |
|---|---|---|---|---|---|
| | | data_type | str | The data type of attributes (e.g. for thickness) | float |
| | | value | float | The value of attributes (e.g. for thickness) | 0.01 |
| | | uom | str | The unit of measure of attributes | - |
| | | description | str | The description of each row | - |
| | layer | id | int | The index of each row | 0 |
| | | system | str | The system where the building physics information from | TABULA |
| | | country | str | Countries to which building physics applies | NL |
| | | material_category | str | The category of construction material | Brick |
| | | material_id | int | The id of construction material | 62 |
| | | material_name | str | The name of the material | Solid brick |
| | | density | float | The attribute relates to construction material | 2000 |
| | | heat_capacity | float | The attribute relates to construction material | 840 |
| | | conductivity | float | The attribute relates to construction material | 0.96 |
| | | embodied_energy | float | The attribute relates to construction material | 0.8 |
| | | embodied_carbon | float | The attribute relates to construction material | 0.25 |
| | | disposal_energy | float | The attribute relates to construction material | 0.11 |
| | | disposal_carbon | float | The attribute relates to construction material | 0.02 |
| | | construction_description | str | The description of construction material | - |
| | | disposal_description | str | The description of disposal | - |
| | profile | id | int | The index of each row | 0 |
| | | system | str | The system where the building physics information from | TABULA |
| | | country | str | Countries to which building physics applies | NL |
| | | profile_id | int | The index of each profile information | 1 |
| | | profile_type | str | The type of the profile | occupancy |
| | | element | str | The category of profile (e.g. day or year) | day |
| | | function | str | The function of buildings that the profile applies to | residential building |
| | | data_type | str | The data type of the profile | array |
| | | length | int | The length of data (e.g. 24 for hourly day record, 365 for daily year record) | 24 |

| Schema | Table | Attribute | Type | Description | Example |
|---|---|---|---|---|---|
| | | value | double precision[] | The value of the profile | {1,1,.....,1} |
| | | description | str | The description of the profile | - |
| | window | id | int | The index of each row | 0 |
| | | system | str | The system where the building physics information from | TABULA |
| | | country | str | Countries to which building physics applies | NL |
| | | window_id | int | Link to the table building_type attributes '_windowTypeId' | 0 |
| | | window_name | str | The name of the window | Window_1,8 |
| | | u_value | float | The attribute relates to window material | 1.8 |
| | | g_value | float | The attribute relates to construction material | 0.6 |
| | | glazing_number | float | The attribute relates to construction material | 2 |
| | | frame_ratio | float | The attribute relates to construction material | 0.3 |
| | | description | str | The description of the window material | - |
| | terrain_physics | id | int | The index of each row | 0 |
| | | composite_id | int | The composite id linked to table composite | 999 |
| | | name | str | The name of the terrain | Clay soil |
| | | category | str | The thematic catrgory of the terrain | groundsurface |
| | | short_wave_reflectance | float | The attribute relates to terrain material | 0.3 |
| | | k_factor | float | The attribute relates to terrain material | 0.7 |
| | | detailed_simulation | str | The parameter relates to CitySim simulation | ture |
| | | thickness | float | The attribute relates to terrain material | 3.82 |
| | | conductivity | float | The attribute relates to terrain material | 0.25 |
| | | cp | float | The attribute relates to terrain material | 890 |
| | | density | float | The attribute relates to terrain material | 1600 |
| | tree_physics | id | int | The index of each row | 0 |
| | | category | str | The typy of the tree | Maple |
| | | leaf_area_index | float | The attribute relates to tree physics parameter | 3 |
| | | leaf_width | float | The attribute relates to tree physics parameter | 0.1 |
| | | leaf_distance | float | The attribute relates to tree physics parameter | 1 |
| | | deciuous | str | The parameter relates to CitySim simulation | false |
| | | short_wave_reflectance | float | The attribute relates to tree physics parameter | 0.3 |

| Schema | Table | Attribute | Type | Description | Example |
|---|---|---|---|---|---|
| | | long_wave _emissivity | float | The attribute relates to tree physics parameter | 0.95 |
| | heat_cool _source | id | int | The index of each row | 0 |
| | | country | str | Countries to which heat or cool source applies | NL |
| | | category | str | The heat source or the cool source | HeatSource |
| | | beginday | int | The start day of heat or cooling in a year | 258 |
| | | endday | int | The end day of heat or cooling in a year | 135 |
| | | name | str | The name of the heat or cool source | spaceX |
| | | pmax | float | The attribute relates to heat or cool source | 100000 |
| | | eta_th | float | The attribute relates to heat or cool source | 0.95 |
| | | eta_tech | float | The attribute relates to heat or cool source | 0.3 |
| | | ttarget | float | The attribute relates to heat or cool source | 5 |
| | | tsource | str | The attribute relates to heat or cool source | air |
| | | description | str | The description of heat or cool source | - |
| | heat_cool _tank | id | int | The index of each row | 0 |
| | | country | str | Countries to which heat or cool tank applies | NL |
| | | category | str | The heat tank or cool tank | HeatTank |
| | | v | float | The attribute relates to heat or cool tank | 0.01 |
| | | phi | float | The attribute relates to heat or cool tank | 20 |
| | | rho | float | The attribute relates to heat or cool tank | 1000 |
| | | cp | float | The attribute relates to heat or cool tank | 4180 |
| | | tmin | float | The attribute relates to heat or cool tank | 20 |
| | | tmax | float | The attribute relates to heat or cool tank | 35 |
| | | description | str | The description of heat or cool tank | - |

Table A.1.: The physics library data sheet in the database

# B. WEATHER LIBRARY DATA SHEET IN THE DATABASE

Table B.1

| Schema | Table | Attribute | Type | Description | Example |
|---|---|---|---|---|---|
| weather | weather_location | city | str | The city of where the weather station located | Twenthe Enschede AP |
| | | state_province_region | str | The state of where the weather station located | OV |
| | | country | str | The country of where the weather station located | NL |
| | | wmocode | int | The WMO station number | 62900 |
| | | latitude | float | The latitude of the weather station | 52.2731 |
| | | longitude | float | The longitude of the weather station | 6.8908 |
| | | altitude | float | The altitude of the weather station | 34.6 |
| | | timezone | str | The timezone of where the weather station located | 1 |
| | weather_parameters | reference | str | The city where the weather station located is used as reference | Twenthe Enschede AP |
| | | country | str | The country where the weather station located | NL |
| | | temporal_resolution | str | The temporal resolution of the weather parameters | hour |
| | | attribute | str | The name of weather parameters | Diffuse Horizontal Radiation |
| | | data_type | str | The data type of weather parameters | float |

| Schema | Table | Attribute | Type | Description | Example |
|---|---|---|---|---|---|
| | | length | int | The length of the weather parameters data (the data are stored in list, there are 8760 hourly values in one year) | 8760 |
| | | value | double precision [] | The value of the weather parameters data | {5,6,...,7} |
| | | uom | str | The units of measure | Wh/m2 |
| | | description | str | The description of each row weather parameters | - |

Table B.1.: The weather library data sheet in the database

# Bibliography


Agugiaro, G., Benner, J., Cipriano, P., and Nouvel, R. (2018). The energy application domain extension for citygml: enhancing interoperability for urban energy simulations. *Open Geospatial Data, Software and Standards*, 3.

Coccolo, S. and Kämpf, J. (2015). Urban energy simulation based on a new data model paradigm: The citygml application domain extension energy. a case study in the epfl campus of lausanne.

Gröger, G. and Plümer, L. (2012). Citygml - interoperable semantic 3d city models.

Mutani, G., Coccolo, S., Jérôme, K., and Bilardo, M. (2018). Citysim guide: Urban energy modelling.

Nations, U., of Economic, D., Affairs, S., and Division, P. (2018). World urbanization prospects the 2018 revision.

Nouvel, R., Mastrucci, A., Leopold, U., Baume, O., Coors, V., and Eicker, U. (2015). Combining gis-based statistical and engineering urban heat consumption models: Towards a new framework for multi-scale policy support. *Energy and Buildings*, 107.

Nutkiewicz, A., Yang, Z., and Jain, R. K. (2018). Data-driven urban energy simulation (dues): A framework for integrating engineering simulation and machine learning methods in a multi-scale urban energy modeling workflow. *Applied Energy*, 225:1176–1189.

OGC (2010). Ogc® geography markup language (gml) — extended schemas and encoding rules. *OpenGIS Recommendation Paper*.

Perez, D., Kämpf, J., and Wilke, U. (2011). Citysim simulation: the case study of alt-wiedikon, a neighbourhood of zürich city. *Proceedings of CISBAT 2011 - CleanTech for Sustainable Buildings*.

Peronato, G., Kämpf, J. H., Rey, E., and Andersen, M. (2017). Integrating urban energy simulation in a parametric environment: A grasshopper interface for citysim. volume 2.

Robinson, D. (2012). *Computer modelling for sustainable urban design: Physical principles, methods and applications*.

Sola, A., Corchero, C., Salom, J., and Sanmarti, M. (2020). Multi-domain urban-scale energy modelling tools: A review.

Yao, Z., Nagel, C., Kunde, F., Hudra, G., Willkomm, P., Donaubauer, A., Adolphi, T., and Kolbe, T. H. (2018). 3dcitydb - a 3d geodatabase solution for the management, analysis, and visualization of semantic 3d city models based on citygml. *Open Geospatial Data, Software and Standards*, 3.

## Colophon

This document was typeset using LaTeX, using the KOMA-Script class `scrbook`. The main font is Palatino.