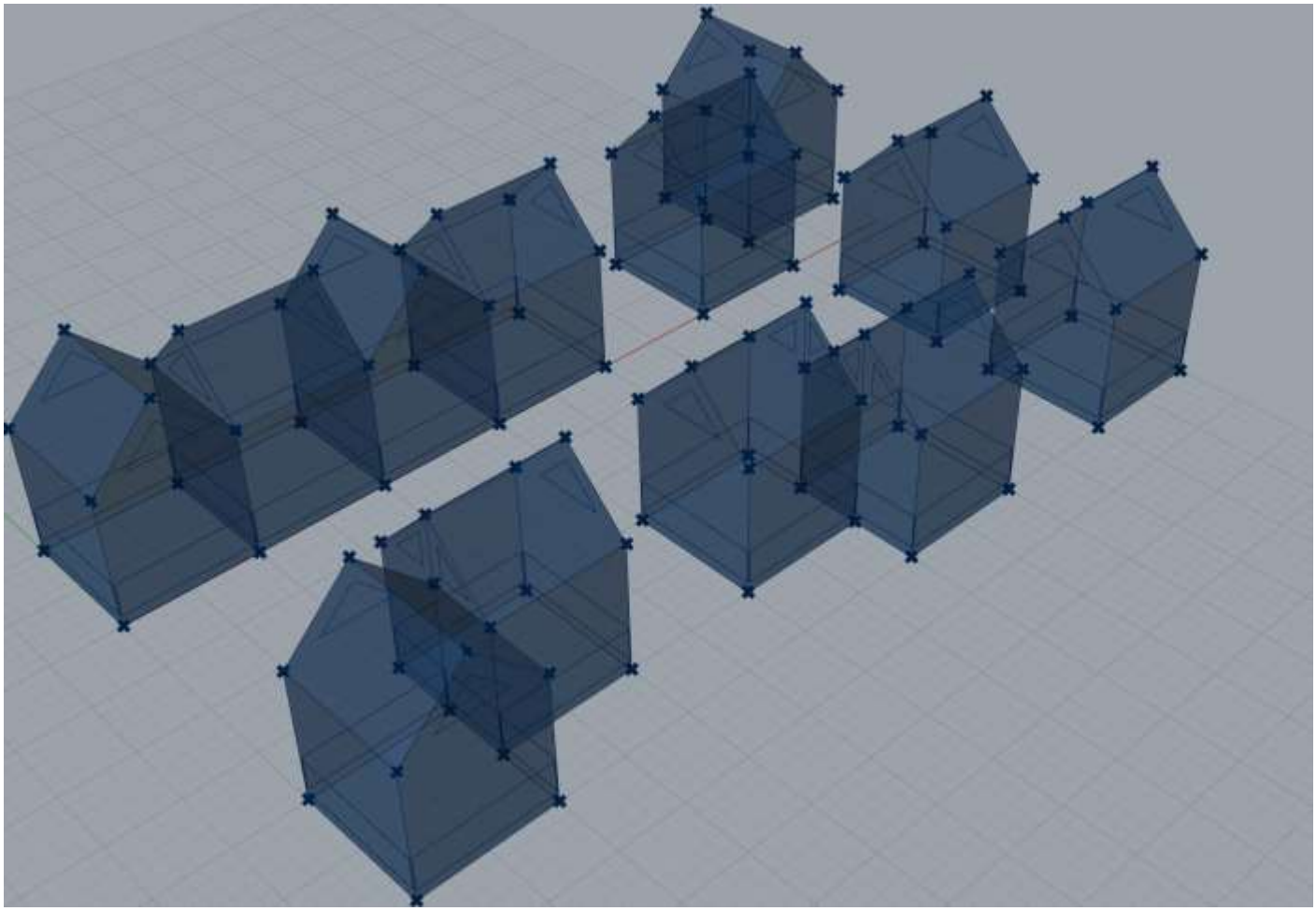
Xin, Wang

Using CityGML EnergyADE Data in Honeybee

MSc thesis in Geomatics for the Built Environment



TU Delft

Using CityGML EnergyADE Data in Honeybee

MSc thesis in Geomatics for the Built Environment

By

Xin, Wang

in partial fulfilment of the requirements for the degree of

Master of Science
in Geomatics

at the Delft University of Technology,
to be defended publicly on Tuesday January 28th, 2020 at 10:30 AM.

Supervisor:	Giorgio Agugiaro, Prof.dr. JE Stoter	
Thesis committee:	Giorgio Agugiaro,	TU Delft
	Prof.dr. JE Stoter,	TU Delft
	Dr.ir. MGAD Harteveld,	TU Delft

This thesis is confidential and cannot be made public until January 28th, 2020.

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Contents

Abstract

Urban energy simulation is becoming more and more important in various areas like urban planning, architecture design and city management. It provides quantified insights for architects and governors to deliver energy-efficient approaches. There are already quite a few energy simulation software or engines on the market. Among these tools, Ladybug family, a series of open-source python packages have its advantages: easy to use, high level of customization and low cost of adoption. It could be run in Rhino Grasshopper, a visual programming interface widely used by architecture industry. Taking 3D geometry created in Rhino and local weather data, Ladybug tools (Ladybug and Honeybee) prepare simulation recipes to run energy simulation with validated software engines like EnergyPlus and OpenStudio.

With all these advantages, when using Ladybug and Honeybee for urban energy simulation, there are two major flaws: it is tedious to build 3D models of all building blocks in Rhino one by one and many key parameters required by energy simulation have to be entered manually. This geometry creation and parameters entering process could be avoided when using CityGML data as input, as CityGML with EnergyADE data already has 3D geometry and energy-related attributes of city within its data model.

In this research, a mapping table between required simulation parameters of Ladybug tool - Honeybee and CityGML with EnergyADE data model is created. Based on this mapping relationship, by following a database approach, all information stored in CityGML with EnergyADE data is retrieved and stored in tables of 3DCityDB and later queried in Rhino Grasshopper and used in data mapping and processing workflow. Energy simulation results could be saved back to database too.

It is concluded that using CityGML with EnergyADE data as input for Honeybee tools is applicable as there is a sufficient mapping relationship between their data models. However, as Honeybee has certain restrictions on input geometry (shared surface areas should be independent surfaces and surfaces should be convex etc.) and it runs energy simulation of buildings not simultaneously but one by one, it is not efficient to use Honeybee for large scale urban energy simulation.

Acknowledgements

I would like to take this chance to appreciate all the support and help offered by my supervisors, family and friends. Special thanks to my first mentor Giorgio Agugiaro for his great patience in hours of tutorial and trouble shooting. It is not possible for me to deliver this research without his guidance and hard work. Also, I would like to express my gratitude for the constructive advices offered by Prof.dr. JE Stoter and Dr.ir. MGAD Harteveld.

During this long journey, I feel so lucky always having support and accompany from my family and friends. My parents and brother cured my depression with love and walked me through the darkness of my life. My friends motivated me with their integrity, kindness and courage, leading me away from reckless decisions and silly mistakes.

Last but not least, I would like to take this chance to mourn the passing of Kobe Bryant, a genius with great achievements, a fighter who never settles and continuously strives to get better, a spiritual mentor and idol of my life. A true hero like him never dies. His legacy will live forever.

List of Figures

List of Tables

1 Introduction

Urban energy simulation is getting more and more important in areas like urban planning, urban or landscape design and urban study. By combining 3D models of urban objects such as buildings, terrain, vegetation and water body with local climate data, practitioners as well as researchers could optimize designing ideas or test theories in the aspects of energy consumption, thermal comfort, day lighting and natural ventilation. Simulation results could not only be used to justify design approach in early stage , but also play an important role in later project operation and management process.

CityGML, as an open standard data model for 3D city developed by Open Geospatial Consortium (OGC), is one ideal input data for above-mentioned simulation. It not only contains geometry information, but also includes texture, semantic attributes and topological relations of city objects. Together with its Energy Application Domain Extension (Energy ADE), which covers even more topics like building physics, material & construction, energy system and occupant behavior [1], it could provide all needed information for urban energy simulation.

There are quite a few simulation programs on the market. This study mainly focusses on Honeybee, one of ladybug tools mainly used for environment energy simulation. As comparing to other software, it has three major advantages: ease of use, low cost of adoption and high level of customization. All tools are open-source python packages with comprehensive comments, allowing entry-level programmers to follow and understand; They could also be integrated into Rhino with Grasshopper, a visual programming environment normally used in architecture and urban planning which provides instantaneous feedback on design modification; Last but not least, codes or algorithms of each tool could be modified or adjusted by advanced users to fit their needs.

Figure 1 below shows traditional workflow of ladybug tools. Two types of input data are taken: 3D models from Rhino with Grasshopper and weather data from Energy Plus Weather (.EPW) file. 3D models in Rhino only provide basic geometry information and .EPW file, which is simple ascii format contains only necessary climatic information such as location, temperature, wind speed of surroundings in hourly resolution (see Figure 2). Depending on what simulation we are going to run (thermal, airflow or light simulation), there are always some key attributes concerning study objects required during simulation process.

For example, if we use Dragonfly, another Ladybug tools to do an urban heat island simulation, plenty of parameters still need to be filled in or chosen from a list during building typology definition process (Figure 3). These include building type, building age, building height, number of stories, floor area, facade area, glazing ratio, material and so on.

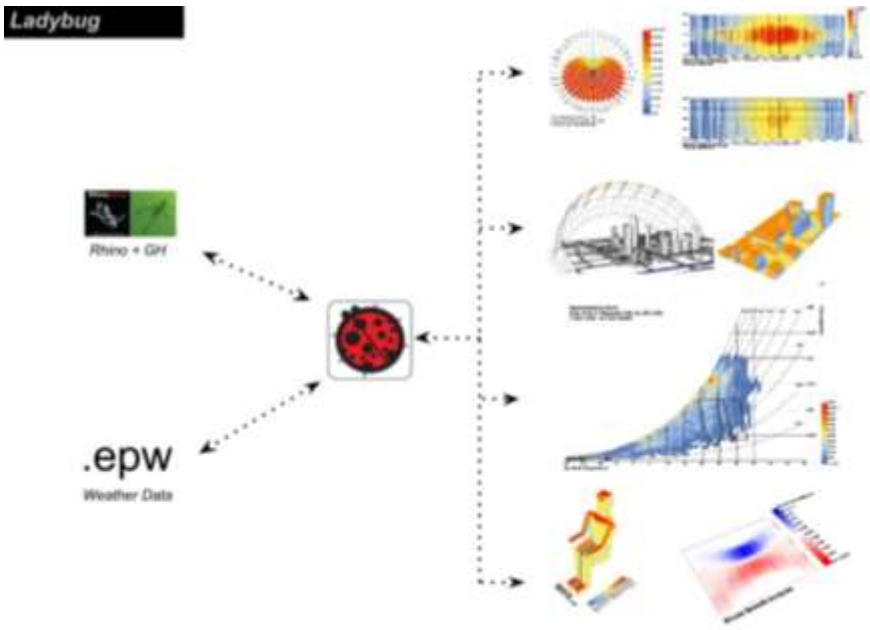


Figure 1: Ladybug Workflow. Source: Ladybug Primer.

Source: https://legacy.gitbook.com/book/mostapharoudsari/ladybug-primer/details [2]

1.1. Scientific relevance

Besides 3D models (geometry information) and .EPW file (local weather data), to use Ladybug tools for urban energy simulation, specific properties are needed. 3D models in Rhino with Grasshopper do not come with these attributes. Entering these data manually is prone to error and may not be feasible when there are too many objects. Also, assigning same values of certain attributes to a group of buildings may over-simplify the situation and lead to inaccurate simulation results.

Some, if not all these information is already there in CityGML with its Energy ADE data schema. However, Ladybug tools, like most of simulation software out there, do not take CityGML as data input option. Even if they did, it is not clear where these parameters are stored as there is no mapping between each other.

To better utilize CityGML data in urban energy simulation, a link between CityGML and simulation software Honeybee is needed. By mapping required parameters needed for urban energy simulation to CityGML with Energy ADE data schema, geometry as well as other important information could be retrieved, stored and fed to simulation engines.

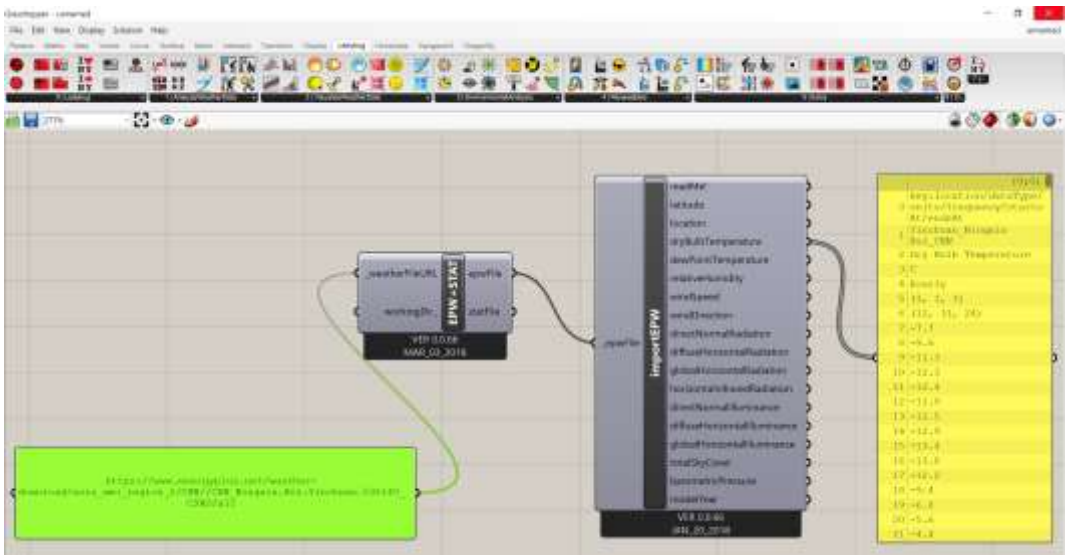


Figure 2: Content of .EPW file.

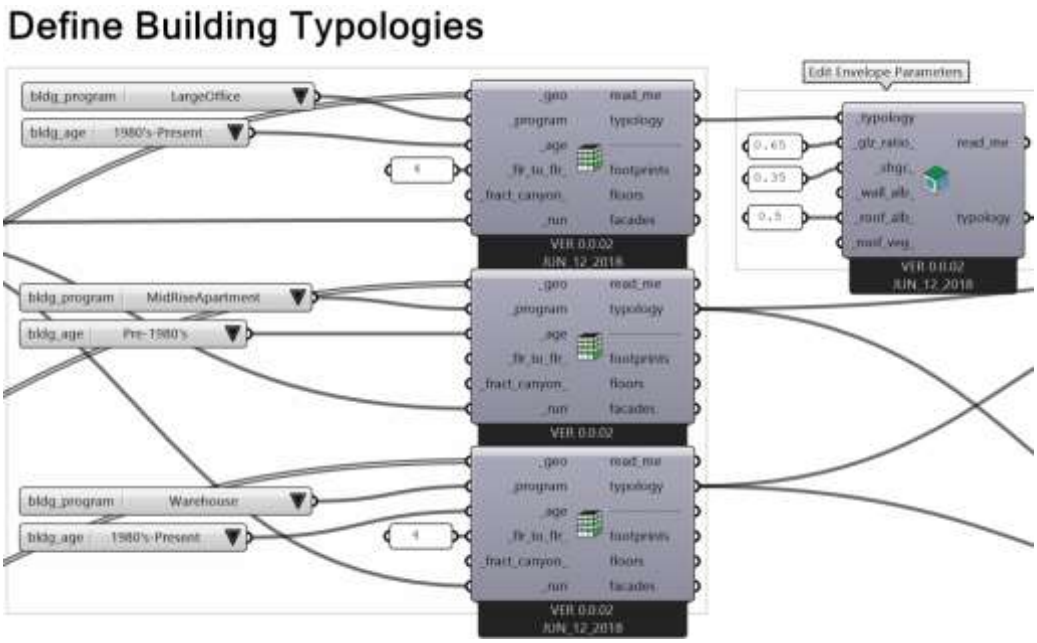


Figure 3: Dragonfly application example.

Source: http://hydrashare.github.io [3]

2 Related Work

2.1. Urban Environment Simulation Data Model & Preparation Approach

Perez, Diane, and Darren Robinson have carried out a research on urban environmental simulation process emphasizing input data [4]. In their opinion, importance of input data is always overlooked. However, for well tested physically based simulation models, limited degree of detail as well as the use of default data is the main source of uncertainty. If user expect correspondence between simulation results and real-world scenario, input data becomes as important as simulation tools.

In their opinion, good input data should be “Ontology”, A formal description of a given knowledge domain, including precise definitions of entities in the domain and logical explanations of their relationships [5]. For CityGML, it defines classes and relations for the city objects, including geometry, semantical and spatial information. In this way, it becomes an important example for “ontology” and could be used as a shared basis for data exchange.

They also discuss the traditional data preparation process for urban environment simulation, which could be rather haphazard: It is not possible or time-consuming to collect all useful data. Even if all data from different sources could be collected, it involves in diverse format and incompatibilities between each other (conflicting values, different units or time series).To save time in data preparation, maximize the value of available data and increase the longevity of data model, they propose a more “sustainable and holistic” approach (Figure 4). A spatial and temporal database is introduced to organize data all together.

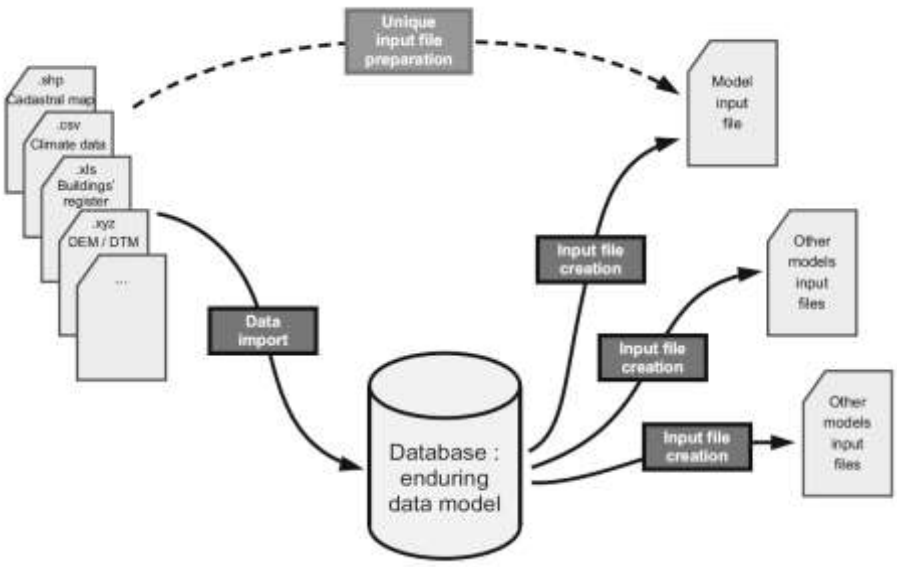


Figure 4: Organized management of data (continuous line) versus traditional direct input file preparation (dotted line).

2.2. Urban Environment Simulation Methodology

Urban energy simulation methodologies could be categorized as macro-simulation, micro-simulation and hybrid of the two. Macro-simulation approach abstracts a whole system as a set of interrelated stocks, flows and feedback mechanisms of resources between these stocks [6].

For macro-simulation approach, system dynamic modelling [7] is used. Whole city as a system is regarded as aggregation of housing, fossil fuels while the ambient environment as a source of renewable energy. Flows of energy, material and associated feedback mechanisms are simulated. Examples of macro-simulation approach include energy modelling of city of Basel, Switzerland [8] and whole Norway [9]. These simulations are based on aggregated statistical data and provide overview information at a large scale. But they are limited to offer more specific knowledge;

For micro-simulation approach, each individual building is physically presented in its spatial context. During the simulation process, unique scene description for each building needs to be prepared. Thus, a significant amount of data is needed. As there are much more information provided, very detailed simulation programs like EnergyPlus, which is designed for modelling single building could be used.

The middle ground of these two approaches is the methodology which simulate only representative building from various classes of building typology. Later, simulation results could be extrapolated to the whole area. In this way, data volume is reduced dramatically and with proper definition of different building typology classes, simulation results could be valuable for both small and large scale. A good example is Urban Weather Generator developed by Building Technology Program, MIT available at https://github.com/ladybug-tools/uwg [10].

2.3. Implementing CityGML in Urban Environment Simulation

SimStadt, a modular platform utilizes LOD1 and LOD2 CityGML data for urban scale thermal and energy demand simulation [11]. The platform consists of preprocessing workflow which extracts key parameters (geometry and semantics) from CityGML model. Besides information retrieval, this process could also fix and enrich CityGML models by using CityDoctor [12]. SimStadt team then uses retrieved information to formulate their own input data model (Figure 5) for simulation process. Figure 6 shows the whole workflow of SimStadt platform.

Sameh Zakhary and etc. design a workflow for simulating arbitrary numbers of building simultaneously [13] by using CitySim, a dedicated urban energy micro-simulation software [14]. They also use 3D City DB for import and export of CityGML files. The main contribution of their study is changing original high -level architecture of CitySim, which is only suited for a standalone simulation engine and could not integrate with other tools to a more open structure. Figure 7 shows the proposed workflow of their study.

Dr. Giorgio Agugiaro and his students have also tried connecting CityGML-based semantic city models to energy simulation engine, namely EnergyPlus. In his research, two approaches are taken separately to estimate heating energy demand of buildings: one is Italian national standard based and the other is by using EnergyPlus with retrieved parameters from CityGML data. Both methods compute three different refurbishment scenarios and later approach presents the initial ways of coupling CityGML data with

simulation engines. Results of his study suggest that both approaches are complementary, not alternatively [15].

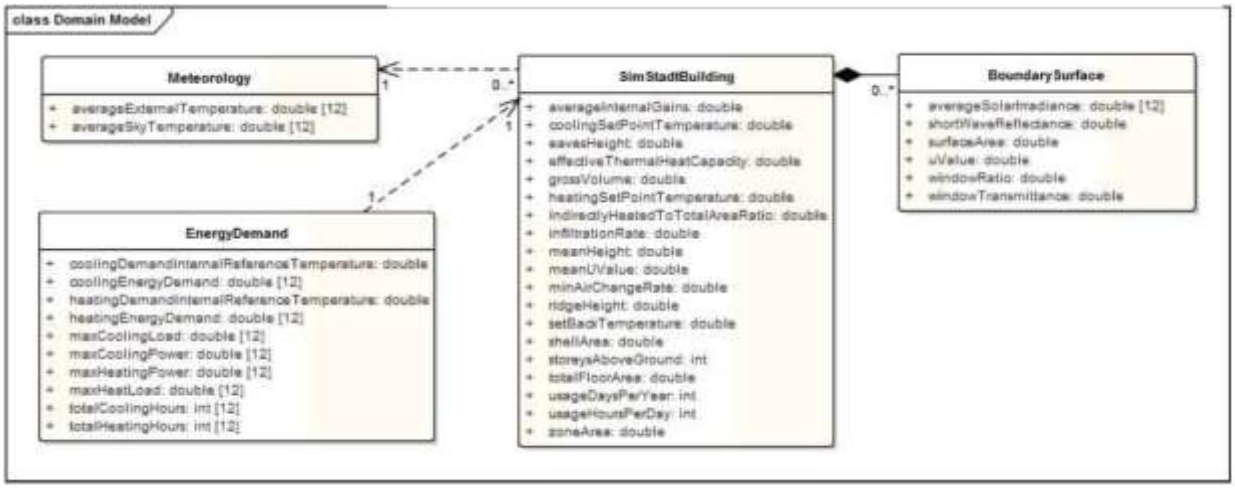


Figure 5: SimStadt input data model for monthly energy balance simulation.

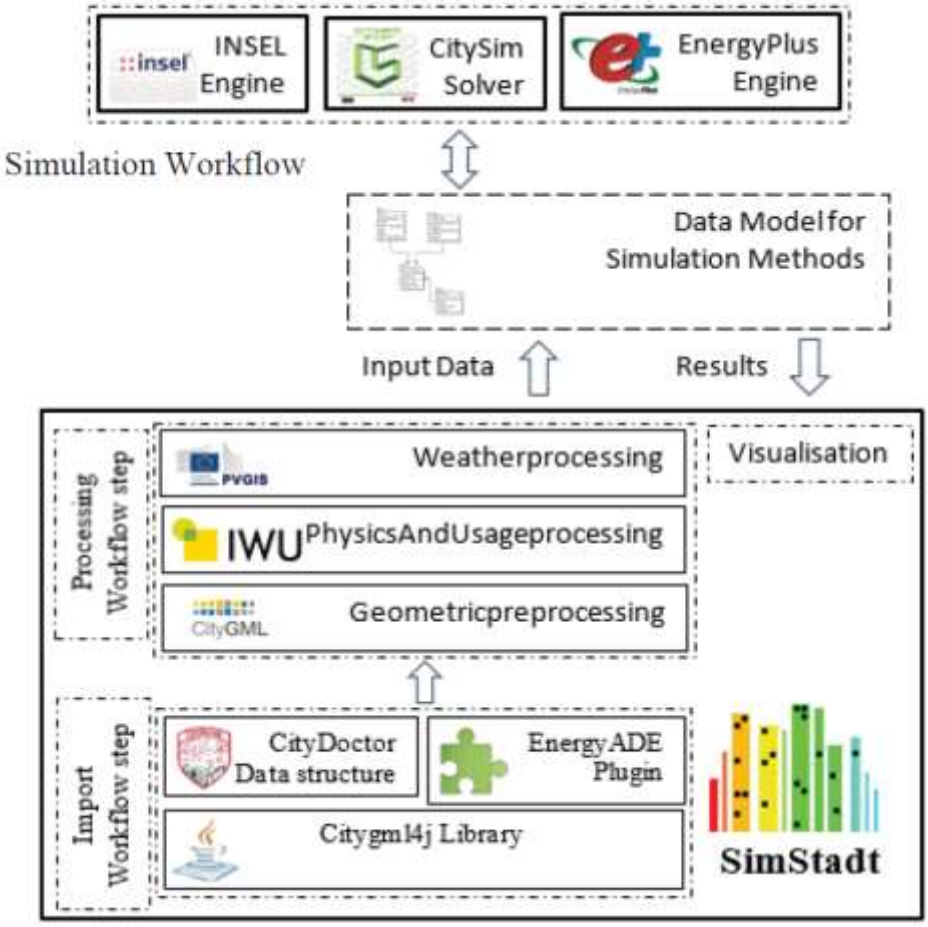


Figure 6: : SimStadt workflow.

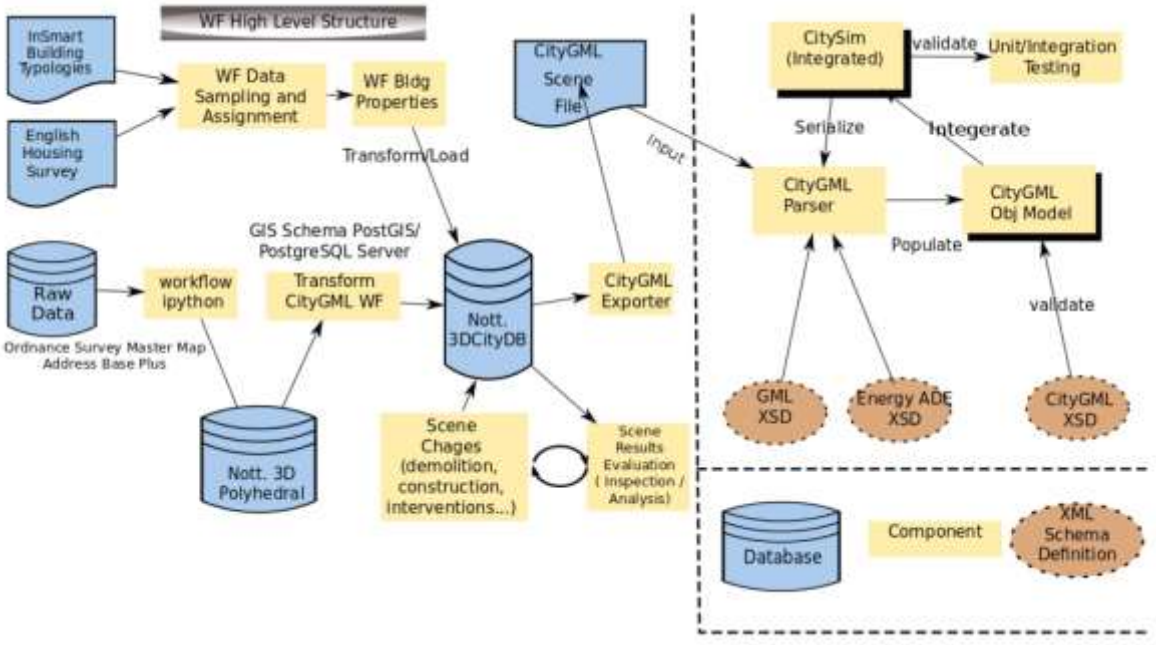


Figure 7: Proposed workflow of urban scene simulation with CitySim.

2.4. CitySim Features and Workflow

Extra attention needs to be paid to CitySim, an urban scale energy simulation tool. It was first developed as a command-line based solver inside the Solar Energy and Building Physics Laboratory (LESO-PB) of EPFL. It takes specifications of input buildings with weather data in XML file and outputs simulation results of energy consumption as text files. Later, a graphic user interface named CitySimPro is distributed by kaemco. This research mainly focusses on GUI or CitySimPro. Comparing to other energy simulation tool like EnergyPlus, CitySim is faster and requires less parameters as it contains models and algorithms that are intended for urban-scale applications [16]. This makes it an ideal tool for urban scale energy simulation. Figure below shows the typical workflow of CitySim tool. CitySim takes two main inputs to composite simulation scene, one is analyzed location weather data and the other is geometric and physics information of the buildings [17].

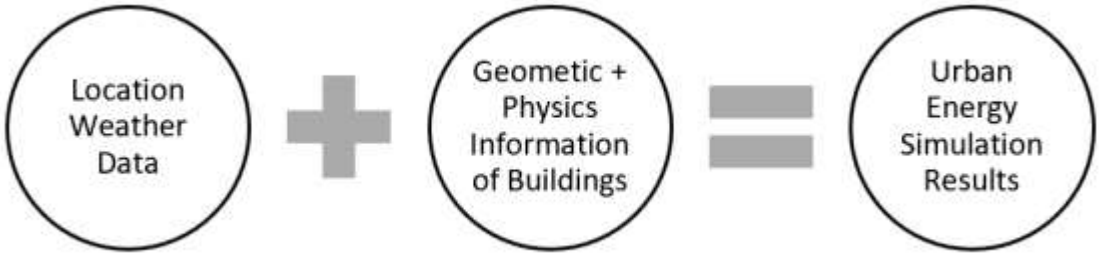


Figure 8: Workflow of CitySim.

Location specific weather data should be hourly resolution and contains following climatic elements as in Figure 9. Besides those climatic elements, CitySim also requires horizon information i.e. far field obstruction (such as mountains) of the skyline [18]. These two types of weather data are usually provided by Meteonorm tool in .cli and .hor files.

Physics information of buildings is key parameters required by algorithms and models to run the energy simulation. It consists of following aspects: Building Characteristics (attributes of whole building), Composites and Insulations (properties of envelope, including predefined composite types in CitySim library and insulation thickness), Opening Properties (information of windows), Visible Surfaces (characteristics of PV or thermal panels and ground reflectance), and Occupants (occupants' information and profile). Next part of the report will clarify more on above-mentioned parameters.

d	Day	
m	Month	
h	Hour	
G_Dh	Diffuse horizontal irradiance	W/m²
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

Figure 9: Required parameters of CitySim.

Another thing worth mentioning is that, CitySim has a built-in library of construction material and pre-defined composites (similar to 'construction' used in EnergyPlus). All these data are in .xml file format and could easily be customized. For each file, power generation systems are specified at the beginning and followed by construction periods properties and occupancy default profiles. From line 138, all material types in 49 sub-categories are included (concrete, wood, metal and etc.). From line 626, for four element class of building (wall, roof, floor and ground), there are pre-defined composites [19].

Figure below shows an example of input xml file with customized composite and its layer information.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<CitySim name="test">
    <Simulation beginMonth="1" endMonth="12" beginDay="1" endDay="31"/>
    <Climate location="Unknown" city="Unknown"/>
    <District>
        <FarFieldObstructions>
        <Composite id="21" name="CitySim Asphalt road" category="Ground">
        <Composite id="4" name="LesosaiBTK W 1" category="Wall">
        <Composite id="10" name="LesosaiBTK B 1" category="Floor">
        <Composite id="12" name="LesosaiBTK D 1" category="Roof">
        <Composite id="1" name="MLP02" category="Wall">
            <Layer Thickness="0.0200" Conductivity="0.9000" Cp="1000" Density="1800" nre="0" gwp="0" ubp="0"/>
            <Layer Thickness="0.5000" Conductivity="0.7200" Cp="1000" Density="1000" nre="0" gwp="0" ubp="0"/>
            <Layer Thickness="0.0150" Conductivity="0.7000" Cp="1000" Density="1400" nre="0" gwp="0" ubp="0"/>
        </Composite>
        <Composite id="2" name="MCV01" category="Wall">
        <Composite id="3" name="MCV02" category="Wall">
        <Composite id="5" name="SOL03" category="Floor">
        <Composite id="6" name="COP01" category="Roof">
```

Figure 10: Customized input xml file.

3 Methodology Concept Design

The approach to answer above mentioned research questions consists of following major steps as shown in figure below:

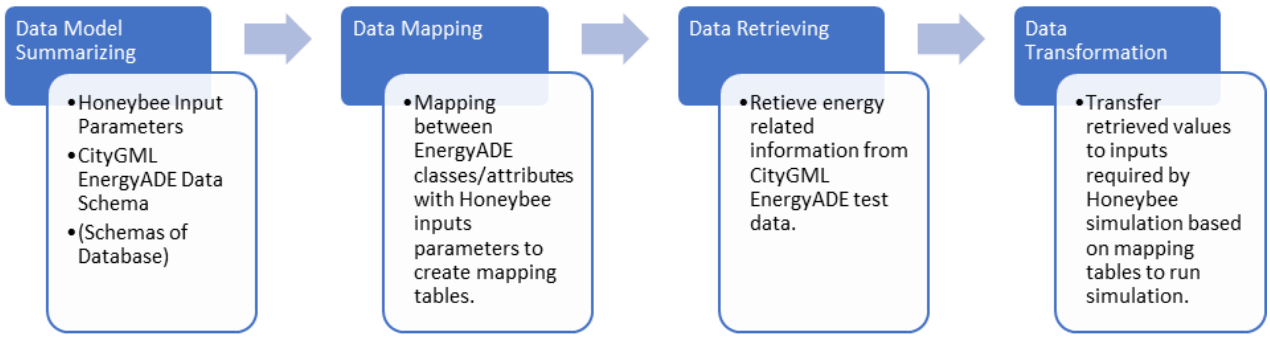


Figure 11: Methodology Roadmap of the Research.

3.1. Data Model Summarizing

There is comprehensive specification documentation and UML diagrams of CityGML with EnergyADE data structure. The main task is to figure out attributes of which classes from CityGML EnergyADE are linked to required inputs of Honeybee energy simulation process and what transformation operations, both in geometry or in semantics are needed to convert those attributes to corresponding inputs. For this research, it is more like a one-way mapping, from schema of Honeybee parameters to model to CityGML EnergyADE. In this process, which attributes from which classes of which part of CityGML EnergyADE data model (EnergyADE core, Building Physics, Material, Schedule, Occupant Behaviors and etc.) should be included is totally decided through understanding Honeybee energy simulation process.

3.1.1 Energy Modelling Concepts of Honeybee and CityGML EnergyADE

Understanding key concepts and processes of Honeybee energy simulation is the start point. From here, the knowledge of relevance between classes of EnergyADE schema and Honeybee parameters would be gained. This also prevents us from running into hundreds of different attributes/parameters from both data models while losing the bigger picture.

Honeybee or EnergyPlus models the heat flow of input scenario (building models) with thermal zones. Each thermal zone represents a 'thermally distinct space [20] in which the temperature of the air is the same. For each building block, it could be treated as a single thermal zone (single-zone approach) or a collection of multiple zones. In latter situation, different parts of buildings like basement, attic, living room, and kitchen have different thermal dynamic characteristics. In Honeybee, each thermal zone is represented by a 3D zone geometry with multiple attributes. The 3D zone geometries are usually created by Rhino and imported into Grasshopper interface. Attribute which are usually thermal or optical related (like surface types, boundary conditions and so on) are then assigned to surfaces or zone itself by Honeybee components. 3D zones need to be closed and there should be no holes on surfaces geometries. Windows or doors are modelled as sub-elements belonging to certain zone surfaces. In this research, thermal zone (3D zone geometry with attributes) created in Honeybee are referred to as Honeybee Zone (HBZone). Thermal surfaces (surface geometry with attributes) are referred to as Honeybee Surface (HBSurface). Figure below shows the geometry rules of Honeybee/EnergyPlus for energy simulation.

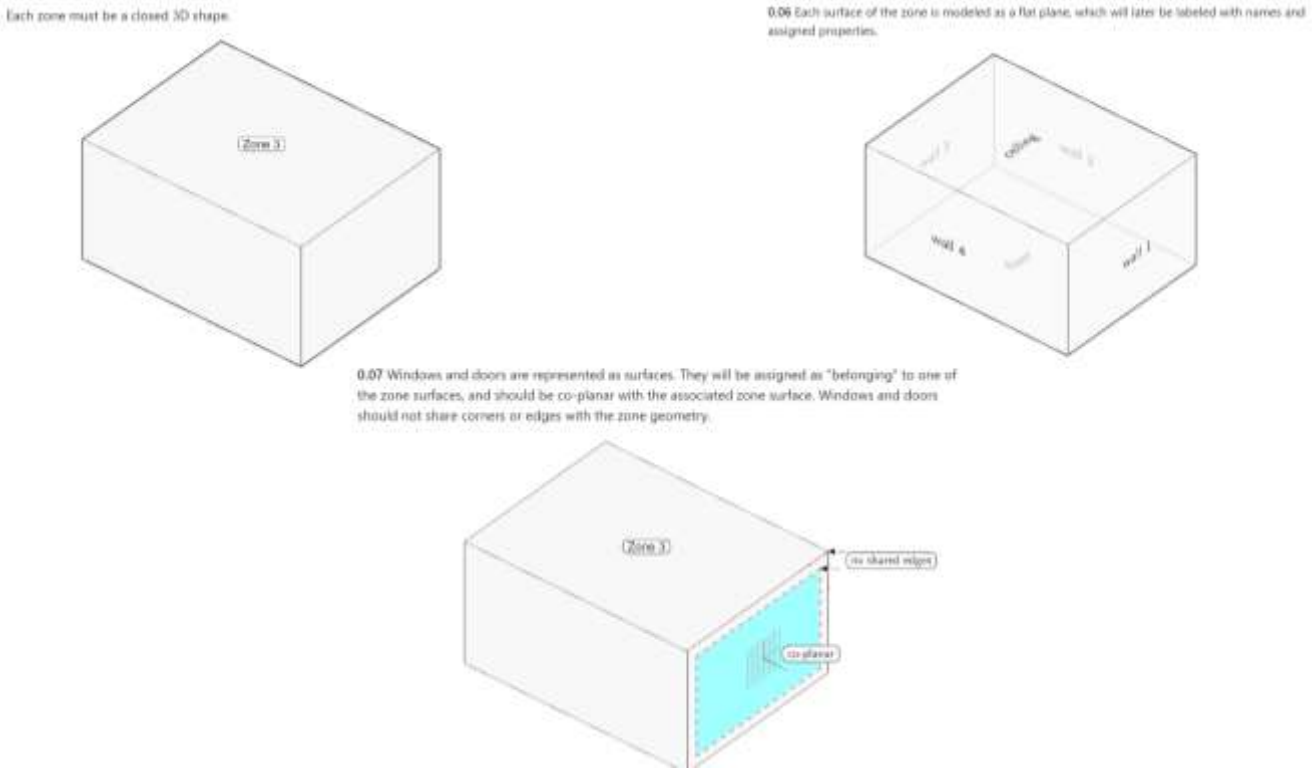


Figure 12: Honeybee Geometry Rules.

Source: https://github.com/mostaphaRoudsari/honeybee/wiki/Modeling-Zone-Geometry [21].

Based on different types of input geometry and components used, Honeybee adopts two approaches to construct HBZones: mass-zone approach and surface-zone approach.

For mass-zone approach, Honeybee takes close 3D masses (closed breps in Rhino) as inputs and by assigning energy-related attributes to masses using components like Masses2Zones, HBZones are created. This approach is straightforward but requires ready-built 3D closed geometries. Another disadvantage is that it offers little flexibility to customize energy-related attributes of each single surface. It might be achieved later by decomposing HBZone to HBSurfaces and change the attributes individually, but it makes the workflow more complicated.
For surface-zone approach, surface geometries are taken as input into Grasshopper interface. By assigning attributes to each individual surface using components like createHBSrfs, HBSurfaces are created. These HBSurfaces could already be used to surface-based simulations like daylight or sheading analysis. Later, HBSurfaces belonging to same HBZone could be aggregated together by using createHBZones component to generate HBZones. This workflow has more steps than mass-zone approach but offers possibility to change/assign attributes of each surface individually.
CityGML with EnergyADE data schema shares a similar modelling methodology: buildings geometries could be modelled as solid or aggregation of (multi-)surfaces. For each building, it could have one or multiple thermal zones which is constructed by thermal boundaries with zone attributes. Thermal boundary is based on surface geometry with surface attributes. Windows or doors are modelled as thermal openings linking to thermal boundaries by foreign keys such as thermal_boundary_id. [22]
By comparing modelling concepts of both schema, surface-zone approach of building up HBZones in Honeybee fits better in this research as attributes retrieved from CityGML EnergyADE data are not only zone related but also surface related. By assigning those attributes to individual surface during process of creating HBSurfaces, EnergyADE data could be better utilized. Also, it is always hard to keep track of individual surface after it is being aggregated to HBZone. It is almost impossible to decompose HBZone to surfaces and then assign mapped values from EnergyADE data.

3.1.2 Module Selection of EnergyADE Data Schema

As mentioned before, CityGML with EnergyADE data schema is a complex system with over hundreds of different classes and attributes categorized in different modules (Figure3). It is impossible to include all modules of this data model in single research. By exploring concepts and workflow of Honeybee energy simulation, only those classes which are essential during surface-zone approach of creating HBZones or energy simulation would be included in mapping process.

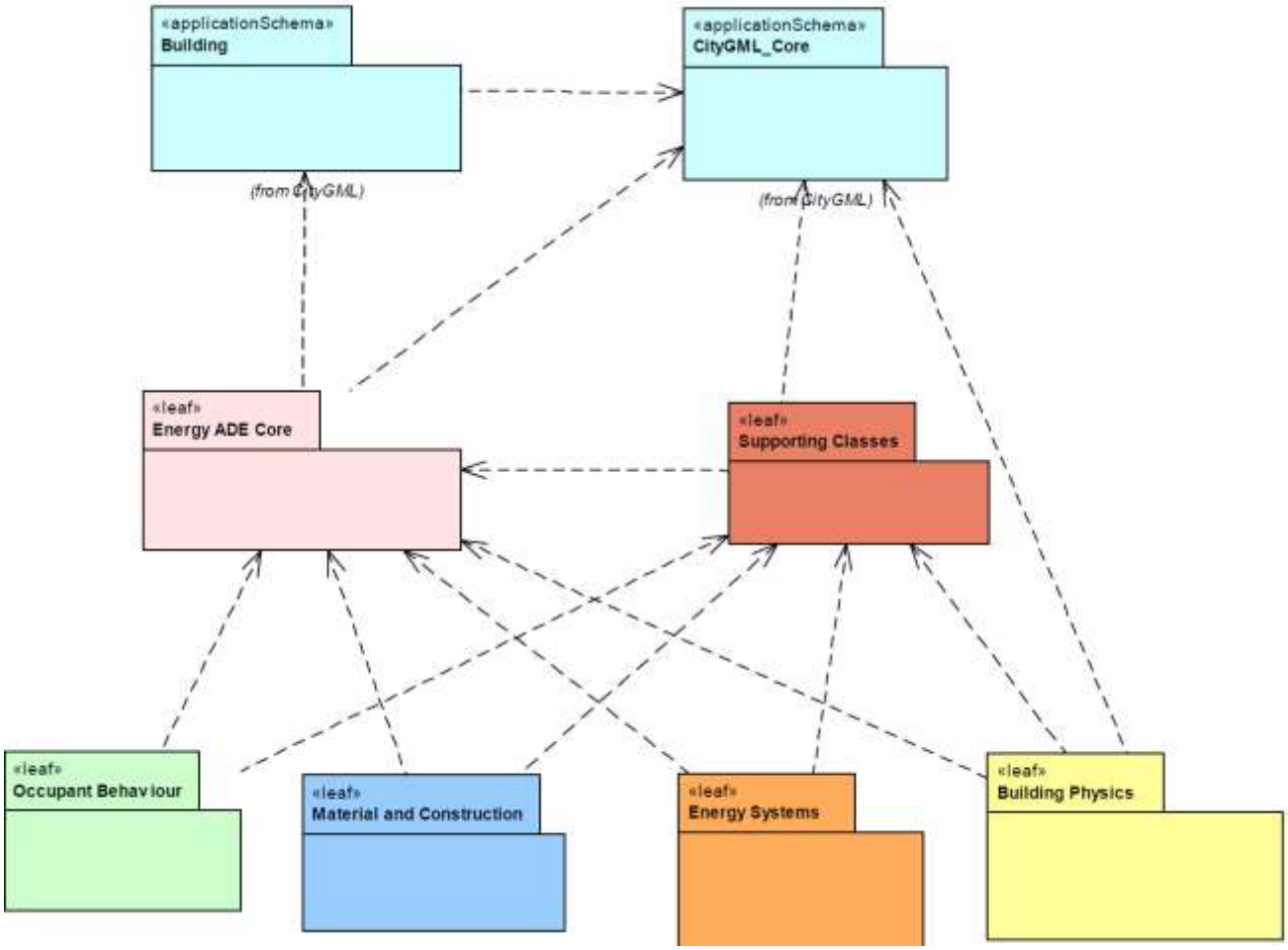


Figure 13: : EnergyADE UML Diagram.

Source: http://www.citygmlwiki.org/images/4/41/KIT-UML-Diagramme-Profil.pdf [23]

In this research, only classes from Building, Energy ADE Core, Building Physics, Occupant Behavior, Material and Constructions are further considered for data mapping process.

3.2 Data Mapping

Unlike CityGML EnergyADE schema which is well presented in UML diagram with comprehensive documentation, information about input parameters of Honeybee are not that well organized. There are two main resources for Honeybee components and input parameters: Honeybee primer [24] which is auto-generated from comments and annotation inside codes of Honeybee components and Honeybee wiki page that is newly launched as tutorial material for users to be familiar with concepts and workflow of Honeybee tool.

By exploring above mentioned material and most importantly, trying out Honeybee components to run simple simulations by myself, two tables (table1 for HBZone and table2 for HBSurface) are created with essential energy-related attributes. Names or attributes as long as short summary of their semantic meanings are included in both table. The idea is that regardless of various input parameters for different components, they are all imported to assign or change some key attributes of HBSurfaces or HBZones. Summarizing these attributes instead of working with all inputs of different components is way more

efficient. Honeybee also provides two very handy components SrfAttributeList and ZoneAttributeList which list above mentioned attributes as outputs (Figure14).

Figure 14: Part of outputs of ZoneAttributeList component.

By checking up UML diagram and if necessary, CityGML specification documentation and codelist library [25], these two tables would be joined with corresponding classes attributes of CityGML and EnergyADE. Also, short summary of their semantic meaning together with notes on data mapping are included.

Table 1: HBSurface Attributes

HBSurface Attributes	Semantic Meaning of Attributes
Radiance Material	The name of a material from default libraries or customized material from material customization components.
EnergyPlus Construction	The name of an EnergyPlus construction from default libraries or customized construction from EnergyPlus construction customization components.
Surface Type	Values indicating different HBSurface types: 0- 'WALL', 0.5-'UndergroundWall', 1- 'ROOF', 1.5-'UndergroundCeiling', 2- 'FLOOR', 2.25- 'UndergroundSlab', 2.5- 'SlabOnGrade', 2.75-'ExposedFloor', 3-

	'CEILING', 4- 'AIRWALL', 5- 'WINDOW', 6- 'SHADING'
Boundary Condition	Different boundary conditions of HBSurface: outdoors, surface, adiabatic, ground.
Shade Material Name	A customized shade material generated by shade material component.
Shading Schedule Name	A schedule which indicates when the shades are raised or lowered.

Table 2: HBZone Attributes

HBZone Attributes	Semantic Meaning of Attributes
Building Program	A built-in library with different types of buildings defined by loads and schedules, such as Office, Retail, MidriseApartment, Warehouse and etc.
Zone Program	Zone program is child class of building program which is usually defined by different zones function. For example, MidriseApartment::Apartment and MidriseApartment::Corridor.
Zone Floor Area	Floor area value of the zone with unit m2.
Zone Volume	Volume value of the zone with unit m3.
Is Conditioned	True or false to decide if HBZone is conditioned with an ideal Air Load System.
Occupancy Schedule	A list of data indicating change of occupancy rate by time.
Lightning Schedule	A schedule that indicates when lightning is on or off.
Equipment Schedule	A schedule that indicates when equipment is on or off.
Equipment Load Per Area	The maximum rate of equipment energy consumption with unit w/m2.
Infiltration Rate Per Area	The desired rate of outside air infiltrated into the HBZone per square meter with unit m3/m2.s (cubit meter per square meter per second).

3.3 Data Retrieving

In this step, CityGML with EnergyADE test data would be prepared with key attributes of classes from above-mentioned modules. When considering what to do with created CityGML test data, there are two potential approaches: File-based approach and database approach.

3.3.1 File-based Approach vs Database Approach

In the context of this research, file-based approach is directly using created test data CityGML file as input to retrieve geometry as well as semantic information and afterwards, feed them to Ladybug tools. Database approach, on the contrary, store all the information from CityGML file in tables of database in the first place and later transfer it to Ladybug tools by using queries.

Workflow of file-based approach consists of following steps (Figure 15): Decoding CityGML file – Storing Information in other files – Reading files in Grasshopper and feeding attributes to Ladybug tools. During this process, based on previous created mapping table between CityGML with EnergyADE data model to input parameters of Ladybug tools, certain attributes transformation (format, units or semantic) could be done in or out of Grasshopper interface.

Figure 15: File-based Approach Workflow.

Workflow of database approach needs several additional steps (Figure 16) : Decoding CityGML File – Mapping CityGML EnergyADE data model with database schema – Writing CityGML EnergyADE features to corresponding database tables – Retrieving information from database in Grasshopper and feed to Ladybug components. Same as file-based approach, transformation of CityGML EnergyADE attributes to meet the parameter requirements of Ladybug tools could be done in Grasshopper or in database.

Figure 16: Database Approach.

There are advantages and disadvantages for both approaches. Based on the following considerations, this research focus only on database approach, not on file-based approach.

A. For file-based approach, there is no ready-made decoder for CityGML with EnergyADE data model. One needs to 'hard-code' everything to deal with complex and verbose structure of CityGML file. This decoding process could be time-consuming and lack of efficiency. For database approach, there are already excellent data integration tools Feature Manipulation Engine (FME) Safe Software and database platform 3DCity Database (3DCityDB) for decoding CityGML and mapping its data model to database schema. The use of these software is more intuitive, convenient and understandable than hundreds lines of code.

B. The outcome program of file-based approach has poor compatibility and expansibility to other applications. In other words, it can only be used to feed data into current version of Ladybug tools (Ladybug

& Honeybee) inside Rhino Grasshopper. The data structure to store all information from CityGML file needs to be designed explicitly for input parameters of these applications beforehand and coded in the program. Every time we would like to test another energy simulation engine, the program has to adjusted. Database approach, on the other side, provides huge flexibility to work with multiple applications even simultaneously. Data is stored as records in tables of database and for majority of modern applications, connecting to existing open-standard database platform like PostgreSQL we used in this research is very handy.

C. It is almost impossible for file-based approach to ensure the quality of input data. The mechanism of this approach is just getting what is inside CityGML file and store the information in another place. What is inside the CityGML file is presumed right. If there are some errors caused by negligence of data provider (same building is input twice or fraction rate is outside the range of 0 to 1), they would be ignored and cause the abnormal results or failure in energy simulation process. Database approach, on the contrary, has overwhelming superiority in ensuring data consistency. By implementing constraints or running redundancy check, only unique features with meaningful attributes that comply with CityGML and EnergyADE standards could be written into database.

D. It is tedious for file-based approach to do data maintenance or update in the future. Especially when there are major changes in CityGML with EnergyADE data model, the codes related to CityGML decoding, data structure designing and information storing needs to be rewritten. Considering CityGML and EnergyADE are relatively young, these changes are almost inevitable and would consume same amount of time, if not even more to adjust the program. For database approach, we only need to create tables for new classes, add some columns for extra attributes or adjust the relations of referenced tables. Mapping process may need to be adjusted too but it is much easier to make minor modifications in workbench of data integration platform (FME) than recoding everything from scratch.

E. It is inconvenient to share information or cooperate during working process in file-based approach. The easy or the only way is to give away copies of data file with retrieved information. Users who have different demands on data input for their work have to figure out by themselves what to keep. For example, one architect would like to run sunlight analysis on roof surface of no.8 building. Instead of getting geometry and attributes of one surface, a huge file with information of 12 buildings and all attributes is provided. It may take a while even to understand the structure of that file. If database approach is implemented, the architect only needs to run a single query to get only record that is a roof surface and belongs to building no.8. Needless to say, database approach also enables operations on data records from multiple users simultaneously. Users' access to database could be regulated and this also eliminate potential risks in data privacy and security.

F. Comparing to file-based approach, database approach is more user-friendly and does not require excellent programming skills. Moreover, it is easier for users to expand the functionality of the approach based on their needs. The starting point of this research is to explore possibilities to utilize CityGML with EnergyADE data in Ladybug tools of Rhino Grasshopper, which is a visual programming interface widely adopted by architects, urban planners or landscape designers who have little programming experience. Working with data integration platform (FME) and database provides them with similar open and straight-

forward user experience. They could also modify FME workbench to decide what to write inside which table of database. This level of customization cannot be matched by file-based approach.
As database approach would be implemented to store information from CityGML EnergyADE test data, a mapping between database schema and EnergyADE data model has to be carried out too. This is explained in implementation part of this research and realized by using FME data integration tool.

3.4 Data Transformation

Based on mapping tables created above, data transformation is carried out later during implementation process within Rhino Grasshopper workflow. This process includes reconstruction of geometries, semantics as well as conversion of formats and units (see notes in mapping tables).

4. Test Data Preparation

Currently there is very little 'real-life' CityGML with EnergyADE data available. As mentioned in previous chapters, researches and use-cases of CityGML EnergyADE data schema are already rare so it is quite challenging to find proper data to build up and test Grasshopper Ladybug tools' implementation process. As a result, a piece of test data has to be prepared beforehand with all necessary energy related information filled in CityGML with EnergyADE data schema.

4.1 Test Data Requirements

Following requirements need to be fulfilled when preparing the test data:
A. The data model should contain not only original CityGML classes and their relationships between each other but also those of Energy Application Domain Extension. In other words, test data should deliver 'full package' of all information from CityGML & EnergyADE data schema.
B. The spatial scale and Level of Detail (LOD) of data should align with research topic, goals and methodology. In this research, the possibility to link CityGML with EnergyADE data to Ladybug tools for urban energy simulation is being explored. Ladybug tools, especially 'energy bug' Honeybee, could utilize LOD2 geometry with a great number of input parameters for single zone energy simulation. However, it cannot handle thousands of zones at the same time. So, the test data should be LOD2 and small in scale and size. And a single LOD2 building is clearly not sufficient for energy modelling of urban level. In this case, a scale of community with dozens of buildings could be optimal.
C. The testing scenarios constructed by the prepared data should be complex enough to include special situations. For example, a building surface polygon with more than 5 vertices might need to be reconstructed differently comparing to those with only 4 or 5 vertices. Also, like mentioned in previous chapters, Honeybee is Energy Zone-based simulation tool and until current process of this research, we consider each building with only one single zone. But what if two buildings share the same wall surface? Will the energy simulation results be different than that of same two separate buildings? More different situations are considered in test data, more meaningful scenarios could be exploited, and more findings would be gained during implementation process..

4.2 Test Data Creation

A piece of test data which perfectly fulfills above-mentioned requirements is prepared by Giorgio Agugiaro. Along with ongoing research process and based on changes in research methodology, test data has been enriched and improved multiple times: It starts with a single LOD2 building with simple geometry, all CityGML features and some attributes from EnergyADE data schema. By using FZKViewer, an open-source software for the visualization of BIM and GIS semantic models, one could have a sense of the scope of test data (Figure 17). Later, more building blocks with additional energy-related information (material, construction and occupants schedule) and complex spatial relationships between each other (shared walls, adjacent buildings) are created and included into the test scenario (Figure 18).

What we have as final test data (Figure 18): A CityGML file which contains 12 LOD2 buildings (each building with one thermal zone) with most of CityGML and EnergyADE attributes. It should be sufficient to work as a start of urban scale energy simulation and would not to be too much for Ladybug tools in Rhino Grasshopper to calculate.

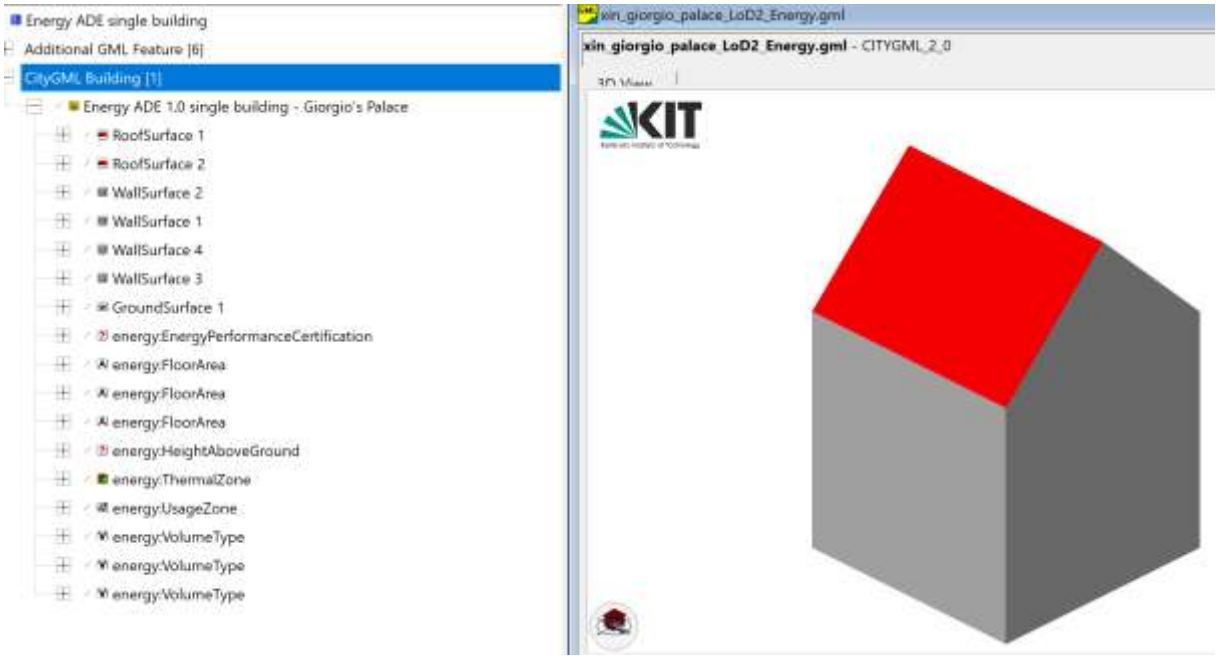


Figure 17: Test Data 1.0. A single LOD2 building with limited EnergyADE attributes.

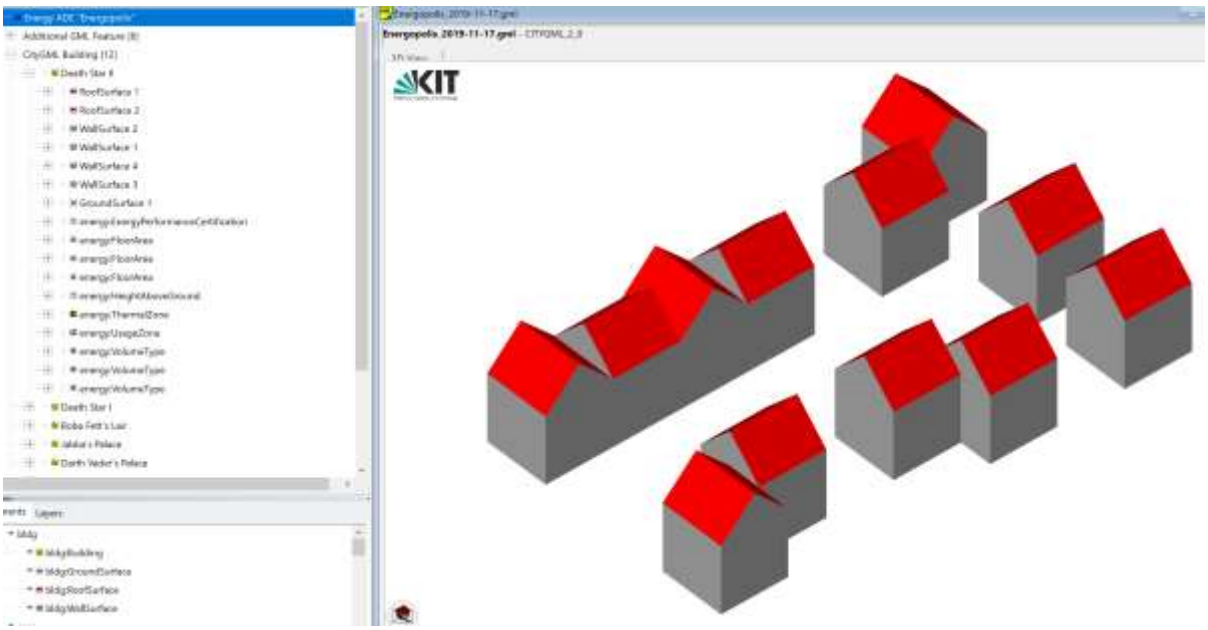


Figure 18: Test Data Final. 12 LOD2 buildings with complex spatial relationship and more EnergyADE Attributes.

4.3 Setup Database Approach

In this research, the commercial data integration platform FME is used for CityGML decoding and data mapping between CityGML file and database. The open-source geo database 3DCityDB is enhanced by 3DCityDB Utility Package, an add-on package developed by Giorgio Agugiaro with extra utilities and extensions to facilitate data management process of 3DCityDB and provides extra schema to store content of EnergyADE.

The implementation process of setting up database approach is like figure below:

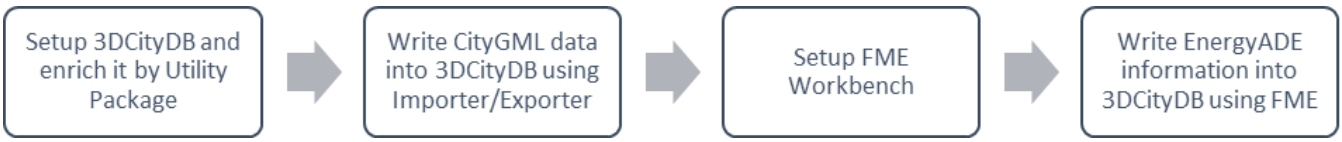


Figure 19: Implementation Process of Setup Database Approach.

4.3.1 3DCityDB Setup

3DCityDB is an open-source package which consists of database schema and multiple tools to import, export, manage, analyze and visualize 3D city models [26]. The database schema complies with CityGML standards (both 1.0 and 2.0) and maps the object-oriented CityGML data model to spatially enhanced relational database management system (SRDBMS). 3DCityDB supports both Oracle (a commercial SRDBMS) and PostgreSQL with spatial extension PostGIS, which is used in this research.
As a package widely used in both academic community and production area, 3DCityDB has a lot of features and advantages:

A. 3DCityDB handles 3D city models of different LOD, both in geometry and semantics. The data schema created in 3DCityDB is 'a complete realization of the CityGML data model of the levels of detail (LOD) 0 to 4.' The way 3DCityDB models city objects allows delicate operation and manipulation on their parts or appliances. For example, roof surface or floor surface or one building object could be selected out and used individually in other applications.

B. 3DCityDB provides a very handy and powerful tool Importer/Exporter to import and export CityGML datasets of both CityGML 2.0 and 1.0 standard with high performance. This tool could process dataset of very large size (over 4 GB) and be able to handle XLinks between CityGML features. It is also multi-threaded which means it could utilize power of multi-processor systems or multi-kernel CPUs to process complex XML structure. Importer/Exporter also provides many useful functions like filtering by bounding box or object IDs. It eases the whole process of writing CityGML data (without EnergyADE part) to database tables.

C. Like mentioned in the first place, 3DCityDB is open-source and platform independent. The whole package, including Importer/Exporter is free and licensed under Apache License, Version 2.0 which allows being included into commercial systems. It can be set up on Oracle or PostgreSQL with PostGIS and run on different operating systems. Moreover, spatial data types implemented in 3DCItyDB are supported by a great number of commercial or open-source GIS software, such as ArcGIS or FME which provide database connection function.

Before setting up 3DCityDB and use it for latter process, it is necessary to understand how CityGML features are mapped into relational database schema and stored in different tables and how their affiliations are managed.

In general, one or multiple classes in CityGML UML diagram are mapped to one table which has identical name to the class name. Each object of class is assigned a unique ID in database tables as well as one and only GMLID from GML datasets. Attributes of these classes are then transferred into columns of corresponding tables with same names. The affiliation between different classes are managed by a table OBJECTCLASS. All class names (column CLASSNAME) of CityGML data model are stored in this table with a unique ID under the column OBJECTCLASS_ID. If one class has parent class, the ID of its parent would be stored under column SUPERCLASS_ID as a foreign key. In this way, for each feature class of CityGML data model, after it is written into tables of 3DCityDB, by checking its OBJECTCLASS_ID and SUPERCLASS_ID, users could know the names of class and its parent class.

Figure 20 is only part of OBJECTCLASS table as an example: Class Building has the OBJECTCLASS_ID of 26. And it has parent class whose OBJECTCLASS_ID is 24, which is _AbstractBuilding (a leading underscore indicates that an abstract class is left out). Thematic surfaces of building with OBJECTCLASS_ID ranging from 30 to 36 are sub-classes of _BuildingBoundarySurface with OBJECTCLASS_ID of 29. And for BuildingInstallation (27), BuildingOpening (37) , BuildingFurniture (40) and BuildingRoom (41), they are all subclasses of _CityObject with OBJECTCLASS_ID of 3.

OBJECTCLASS		
ID	CLASSNAME	SUPERCLASS_ID
0	Undefined	
1	_GML	
2	_Feature	1
3	_CityObject	2
4	LandUse	3
5	GenericCityObject	3
6	_VegetationObject	3
7	SolitaryVegetationObject	6
8	PlantCover	6
9	WaterBody	105
10	_WaterBoundarySurface	3
11	WaterSurface	10
12	WaterGroundSurface	10
13	WaterClosureSurface	10
14	ReliefFeature	3
15	_ReliefComponent	3
16	TINRelief	15
17	MassPointRelief	15
18	BreaklineRelief	15
19	RasterRelief	15
20	_Site	3
21	CityFurniture	3
22	_TransportationObject	3
23	CityObjectGroup	3
24	_AbstractBuilding	20
25	BuildingPart	24
26	Building	24
27	BuildingInstallation	3
28	IntBuildingInstallation	3
29	_BuildingBoundarySurface	3
30	BuildingCeilingSurface	29
31	InteriorBuildingWallSurface	29
32	BuildingFloorSurface	29
33	BuildingRoofSurface	29
34	BuildingWallSurface	29
35	BuildingGroundSurface	29
36	BuildingClosureSurface	29
37	_BuildingOpening	3
38	BuildingWindow	37
39	BuildingDoor	37
40	BuildingFurniture	3
41	BuildingRoom	3

Figure 20: OBJECTCLASS Table 3DCItyDB. [27[

Besides management of classes' affiliation, another thing needs to be noticed is storage of geometry information. Different from UML diagram of CityGML specification, the representation of surface geometry in 3DCityDB is written in table SURFACE_GEOMETRY. Each planer polygon (possibly with holes) is stored under attribute Geometry. And these surfaces could be aggregated to a composite surface or the boundary

of a volumetric object. For each entry in table SURFACE_GEOMETRY, a composite surface or solid object is identified by boolean attributes IS_COMPOSITE and IS_SOLID (0 or 1). And for each entry with unique ID, similar to above mentioned OBJECTCLASS table, it is associated to its parent geometry by foreign keys like PARENT_ID and ROOT_ID. The existence of ROOT_ID eliminate the complexity of recursive queries and improve the query efficiency.

Figure 21 is an example of a volumetric geometry and how it is represented together with all its sub-geometries in SURFACE_GEOMETRY table: This LOD 1 building with GMLID UUID_lod1 is bounded by composite surface with GMLID lod1Surface which consists of 5 single polygons with ID from 3 to 7. Each single polygon is associated with composite geometry by PARENT_ID and they are also related to the LOD1 building object by ROOT_ID. In case the user would like to select all building surfaces of this building, he or she only needs to query those surfaces with same ROOT_ID = 1. There is no need to query the composite surface of building and then trying to find single polygons which belong to composite surface.

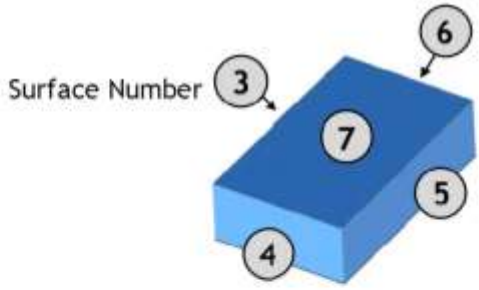


Figure 30: LoD 1 building - closed volume bounded by a CompositeSurface which consists of single polygons

SURFACE_GEOMETRY							
ID	GMLID	PARENT_ ID	ROOT_ ID	IS_ SOLID	IS_ COMPOSITE	GEOMETRY	SOLID_ GEOMETRY
1	UUID_lod1	NULL	1	1	0	NULL	GEOMETRY for Solid
2	lod1Surface	1	1	0	1	NULL	NULL
3	Left1	2	1	0	0	GEOMETRY for surface 3	NULL
4	Front1	2	1	0	0	GEOMETRY for surface 4	NULL
5	Right1	2	1	0	0	GEOMETRY for surface 5	NULL
6	Back1	2	1	0	0	GEOMETRY for surface 6	NULL
7	Roof1	2	1	0	0	GEOMETRY for surface 7	NULL

Figure 21: Geometry Representation in SURFACE_GEOMETRY table of 3DCityDB.

For purpose of this research, it is especially crucial to understand how class Building is mapped into tables of 3DCityDB. 3DCityDB merged all information from three CityGML classes AbstractBuilding, Building and BuildingPart into one single table Building. The subclass relationship between class Building and CityObject is managed by using same ID in both tables. It means that for each entry in table BUILDING, there must be a record in table CITYOBJECT with same ID. Like mentioned before in classes' affiliation management, the table THEMATIC_SURFACE represents the thematic boundary of buildings. The type of boundary surface (CeilingSurface, InteriorWallSurface, FloorSurface and so on) are identified by OBJECTCLASS_ID and the

aggregation relation between thematic surfaces and building is realized by foreign key BUILDING_ID in table THEMATIC_SURFACE which refer to ID of different buildings.

Last thing to pay attention to is predefined sequences of 3DCityDB for users to generate unique integers for primary keys of input records of different tables. The sequence is designed to start from 1 and increment 1 each time as a new sequence number is generated. For example, when a new object of Building class is written into table BUILDING and CITYOBJECT, the ID of this object is generated from sequence CITYOBJECT_SEQ. For writing CityGML part of data into 3DCityDB, this is not that important as it is fully taken care of automatically by Importer/Exporter tool. However, when trying to write EnergyADE part of data into database by using FME platform, IDs of new objects of classes in EnergyADE UML diagram (ThermalZone for instance), needs to be created. And it is very convenient to generate those IDs by sequences than doing it manually each time.

After understanding all above mentioned concepts, setup of 3DCityDB is quite straight forward by following instructions in 3DCityDB manual. One important thing to notice: For this research, PostgreSQL 11 with latest compatible PostGIS extension is installed in the first place. Then 3DCityDB with version number 3.3.0 instead of latest release has to be installed by executing CREATE_DB.sql script. The reason for this is that the 3DCityDB Utility Package used to enrich the 3DCityDB is only tested and compatible with older version (V3.3.0). This could change later by further development of 3DCityDB Utility Package.

After setting up 3DCityDB, CityGML test data created in previous steps could be written into database by using installed Importer/Exporter tool (Figure 22). All data in original CityGML data model will be successfully written into corresponding tables of 3DCityDB but objects and attributes of classes in EnergyADE schema will not be imported. These ‘losing information’ will be later mapped into database schema enriched by 3DCityDB Utility Package by FME data integration tool.

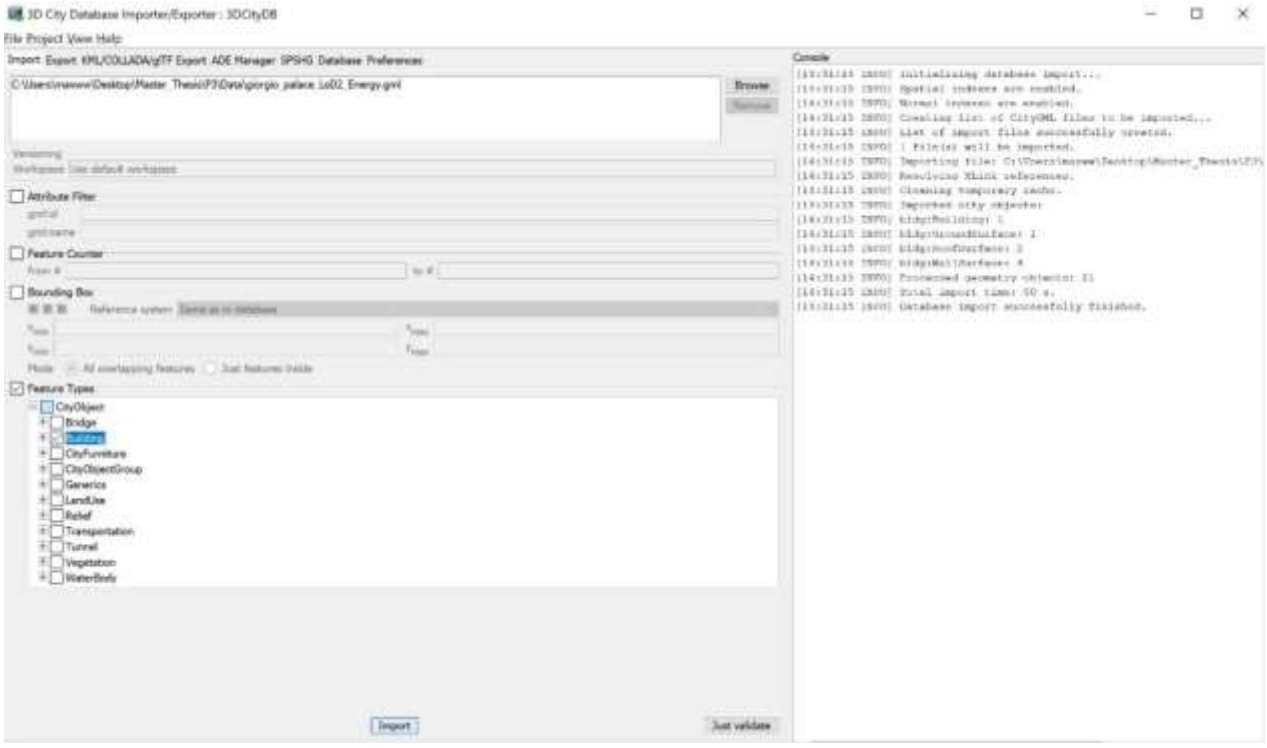


Figure 22: Import CityGML data into 3DCityDB using Importer/Exporter

4.3.2 Install 3DCityDB Utility Package

3DCityDB Utility Package (3DCityDB “Plus”) developed by Giorgio Agugiaro contains additional stored procedures, views and trigger functions to extend the original 3DCityDB [28].

3DCityDB, as discussed above, maps object-oriented CityGML data model to relational database schema. By creating additional tables (like OBJECTCLASS, SURFACE_GEOMETRY tables) and referencing tables by foreign keys (OBJECTCLASS_ID, PARENT_ID, ROOT_ID, BUILDING_ID and etc.), affiliation relationships between different classes and geometries of CityGML model could be managed.

However, this comprehensive design decision also results in inconvenience and complexity in data handling operations. It may not be a problem when user only store CityGML data into 3DCityDB using powerful Importer/Exporter tool. However, as information of one specific class could be stored in multiple linked tables in database, once users would like to carry out further CRUD operations (i.e. select, insert, update and delete) on objects of this class, required SQL statements would be extremely complex. To partially overcome this problem, original 3DCityDB does come with some stored procedures which mainly for ordered and hierarchical deletion process. But it is clearly not enough.

Another problem of 3DCityDB is that, it has not provided direct support of EnergyADE data schema, or generic support of any ADEs. All information of original CityGML data model could be imported into corresponding tables of database directly by using 3DCityDB Importer/Exporter but those classes and attributes of EnergyADE would be left out. The reason is that current database schema is created manually from CityGML data model and it is ‘static’. CityGML ADEs, on the other hand, are not included in the original CityGML data model. To support EnergyADE, 3DCityDB core database schema needs to be extended.

3DCityDB Utility Package has, to some extent, provided solutions to both problems by offering following features:

A. Besides stored procedures for deletion operation from original 3DCityDB, 3DCityDB Utility Package offers insert stored procedures for 3DCityDB tables. These procedures are stored in the citydb_pkg schema and enable user to insert data into tables. The arguments of insert procedures are matched to function parameters by named notation using := and could be written in any order. Figure 23 is an example of implementing insert procedure to write data into building table.

```
SELECT citydb_pkg.insert_building(
id := 5094,
building_root_id := 5094,
class := 'Residential',
storeys_above_ground := 5,
storeys_below_ground := 2
);
```

Figure 23: Insert Stored Procedure of 3DCityDB Utility Package.

B. 3DCityDB install a number of views in the database schema citydb_view [29]. Users could use qualified names which consist of schema name and class name separated by dot to access, insert or delete any database objects which might be stored over multiple tables in the database. It is realized by a number of

aforementioned stored procedures. For example, citydb_view.insert_building(…) procedure will take arguments as all parameters of table CITYOBJECT and BUILDING and insert into both tables separately. And users need not to specify OBJECTCLASS_ID as input parameter in these stored procedures as they will automatically look up OBJECTCLASS_ID and if necessary, OBJECTCLASS_NAME attributes in OBJECTCLASS table based on corresponding views.

C. Trigger functions are added to some views to make them ‘updatable’ (the number of ‘updatable’ views is growing and until now, all views related to energy simulation in this research are made updatable by Giorgio) [30]. That means, when users would like to insert, update or delete any records in multiple tables of database, instead of writing complex SQL scripts to handle them individually, they could directly interact with views. Once views are updated, all records in cascaded tables would be changed accordingly.

D. 3DCityDB Utility Package creates schema explicitly for EnergyADE data model. Tables or views starts with prefix ‘nrg8_’ are those of EnergyADE data and extinguish themselves from original CityGML ones. Parts of names after the prefix are identical or similar to names of corresponding classes. For example, view nrg8_thermal_zone contains all information of class ThermalZone from EnergyADE data model.

Setting up process of 3DCityDB Utility Package is as convenient as that of 3DCityDB. It could be done automatically by running batch file CREATE_DB_citydb_utilities.bat. Before running the batch file, configuration parameters need to be set and corresponding 3DCityDB database instance needs to be selected. Notice again: For current version of 3DCityDB Utility Package, 3DCityDB V3.3, not the latest 3DCityDB version needs to be installed.

B. 3DCityDB install a number of views in the database schema citydb_view. Users could use qualified names which consist of schema name and class name separated by dot to access, insert or delete any database objects which might be stored over multiple tables in the database. It is realized by a number of aforementioned stored procedures. For example, citydb_view.insert_building(…) procedure will take arguments as all parameters of table CITYOBJECT and BUILDING and insert into both tables separately. And users need not to specify OBJECTCLASS_ID as input parameter in these stored procedures as they will automatically look up OBJECTCLASS_ID and if necessary, OBJECTCLASS_NAME attributes in OBJECTCLASS table based on corresponding views.

C. Trigger functions are added to some views to make them ‘updatable’ (the number of ‘updatable’ views is growing and until now, all views related to energy simulation in this research are made updatable by Giorgio). That means, when users would like to insert, update or delete any records in multiple tables of database, instead of writing complex SQL scripts to handle them individually, they could directly interact with views. Once views are updated, all records in cascaded tables would be changed accordingly.

D. 3DCityDB Utility Package creates schema explicitly for EnergyADE data model. Tables or views starts with prefix ‘nrg8_’ are those of EnergyADE data and extinguish themselves from original CityGML ones. Parts of names after the prefix are identical or similar to names of corresponding classes. For example, view nrg8_thermal_zone contains all information of class ThermalZone from EnergyADE data model.

Setting up process of 3DCityDB Utility Package is as convenient as that of 3DCityDB. It could be done automatically by running batch file CREATE_DB_citydb_utilities.bat. Before running the batch file,

configuration parameters need to be set and corresponding 3DCityDB database instance needs to be selected. Notice again: For current version of 3DCityDB Utility Package, 3DCityDB V3.3, not the latest 3DCityDB version needs to be installed.

4.3.3 FME Workbench Setup

Feature Manipulation Engine (FME) is a commercial data integration platform distributed by Safe Software. The whole package consists of FME Desktop (Workbench + Data Inspector), FME Server and FME cloud [31]. Comparing to other data integration or transformation solutions, FME has a number of advantages which make it the perfect tool for writing EnergyADE data into 3DCityDB database. :

A. Like Rhino Grasshopper or ArcGIS Model Builder, it provides a visual programming interface (Figure 24): simply drag-and-drop data and pre-built transformation tools into the workspace and connect them properly to form a customized workflow. During the whole process, no coding work is need and users could focus on what they want from the data.

B. It supports a large number of data formats and applications (over 450, including CityGML and PostgreSQL and PostGIS) and provides a variety of powerful transformers to change the structure, content and characteristics of data. CItyGML data model (1.0 and 2.0), or XML data structure is of course supported and could be decoded effortlessly in FME Workbench.

C. The workflow of data mapping and integration in FME could be automated and event based. Same working procedures could be bounded together, and no more manual effort is needed to perform the repetitive and tedious tasks. Furthermore, users could define scripts to run before the transformation starts so that a high level of customization could also be achieved.

D. FME offers an intuitive data mapping process when writing attributes from data source to destinated data models. A writer could be generated explicitly for certain data schema and by connecting processed data into writer, FME could inform user if certain attributes are well prepared and ready to be integrated (green flags in front of attribute names) or just missing (red flags). Figure 25 as an example.

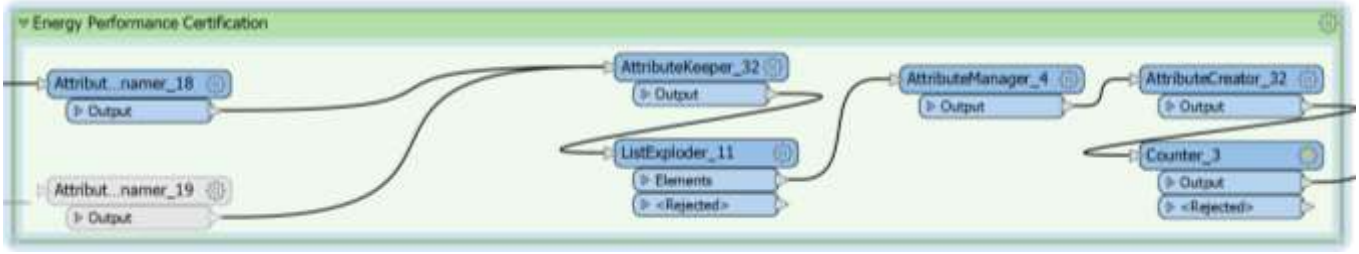


Figure 24: Grasshopper-like Drag-and-drop Interface of FME

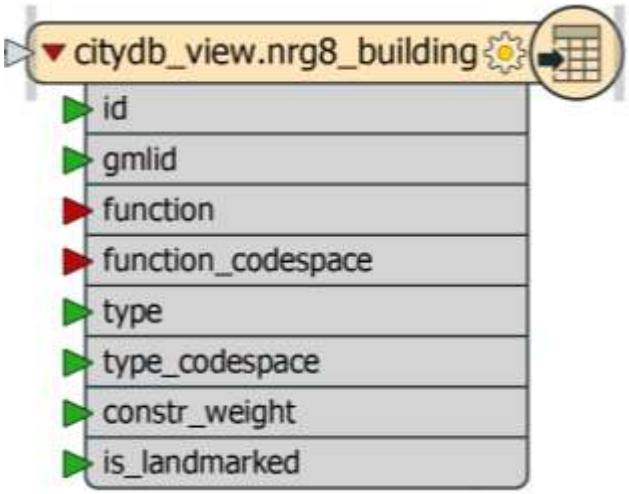


Figure 25: FME Feature Writer with Green/Red Flags.

Though the structure of finished FME workbench from Giorgio seems of great complexity (See Appendix), designing concept of this workflow is simple: Write classes of EnergyADE model with their attributes in hierarchical order (from highest level to lowest) into corresponding updatable views of enriched 3DCityDB and link them with each other by referring to their IDs and parent class IDs as foreign keys.

There are certain key points to be sorted out in order to build FME workbench by following this concept:

A. Generate unique IDs for objects of classes to be written into 3DCityDB. Like mentioned in previous part, 3DCityDB provides sequences which start from 1 and automatically increments 1 when new sequence value is created to generate IDs of imported objects with ease. In this research, the next value of CITYOBJECT_SEQ is used as a global counter in FME to assign a unique ID to each object. Notice that only certain classes (Building, ThermalZone, ThermalBoundary, UsageZone, Facilities and etc.) are sub-classes of City_Object. Other classes like Occupants, Household, Schedule are only features and in principle, they should have their own counters or use sequences from other tables and not consume IDs generated by sequence of CITYOBJECT_CLASS table. However, as in database, it is ok as long as IDs are unique and there is no data redundancy issue, the design decision is made here to use CITYOBJECT_SEQ value as global counters for all classes to simplify the FME workbench, which is already overwhelming.

B. To manage the classes' affiliation relationships, they should be written in hierarchical order from highest level to lowest level. It has to be done in this way as objects from subclasses need to inherit attributes PARENT_ID or BUILDING_ID or Therm_Zone_ID as foreign keys from IDs of objects in parent classes. As mentioned before, unique IDs are generated by values of CITYOBJECT_SEQ when they are written into the database. If subclasses are written first, there are no records in tables or views of parent classes to retrieve those foreign keys. This situation will always cause error messages when trying to write data into updatable views as trigger functions implemented in 3DCityDB Utility Packages would first refer to records of parent class tables. After all objects of a parent class are written into database, objects of its subclasses would be prepared and joined with them by gml_parent_id and their gml_id. IDs of objects from upper classes would be joined to the tables of lower classes as values of PARENT_ID.

C. Carry out INSERT or UPDATE operations directly on corresponding updatable views of classes. Like mentioned before, 3DCityDB Utility Packages enriches 3DCityDB and provides trigger functions for views to make them ‘updatable’. Insert or update records of views could automatically change data stored across the spreaded tables. Also, when deleting records in views, those in cascaded views or tables would be cleaned too. This provides very convenient data management operations for 3DCityDB. When building FME workbench in steps or by try-outs, it is always necessary to delete those records imported into database in previous process. By initiating citydb_pkg.nrg8_delete_building function up front, all records imported into different views or tables would be deleted. Another thing to notice is that, PostgreSQL writer rather than PostGIS writer should be used when writing joined data into database as PostGIS writer does not work with updatable views.
D. 3DCityDB Utility Package combines some code-lists of EnergyADE data model into standalone tables. In this research, two tables have to be filled with data from values in EnergyADE data:
nrg8_dimensional_attrib combines type, value and value unit of height, floor area and volume of buildings and nrg8_optical_properties includes fraction, range, surface side of emissivity, reflectance and transmittance. All these values stored inside these two tables are linked to buildings or constructions based by foreign keys like CITYOBJECT_ID and CONSTR_ID. These values are stored inside CityGML data attributes as lists and need to be ‘exploded’ first and written separately into corresponding views. Figure 26 shows part of table nrg8_dimensional_attrib as an example.

	id [PK] integer	objectclass_id integer	type character varying	value numeric	value_unit character varying	cityobject_id integer
1	5	258	netVolume	1000	m^3	1
2	6	258	grossVolume	1250	m^3	1
3	7	258	energyReferenceVol...	800	m^3	1
4	12	258	netVolume	1000	m^3	2
5	13	258	grossVolume	1250	m^3	2
6	14	258	energyReferenceVol...	800	m^3	2
7	19	258	netVolume	1000	m^3	15
8	20	258	grossVolume	1250	m^3	15
9	21	258	energyReferenceVol...	800	m^3	15
10	26	258	netVolume	1000	m^3	23

Figure 26: Part of nrg8_dimensional_attrib table.

E. After being imported into FME, geometry attributes of CityGML data would be automatically transferred into FME Multi-Surfaces. However, 3DCityDB could only handle geometries in Multi-Polygons. To successfully write geometry attributes into tables of 3DCityDB, after reading CityGML data into FME, geometry attribute (in Multi-Surfaces) need to be deaggregated into FME_Surface, coerced into FME_Polygon, reaggregated and then write into database tables. Remember when writing transformed

geometry attributes into data base tables, UPDATE operation instead of INSERT operation needs to be carried out. Figure 27 shows a demo of FME workflow for geometry transformation process.

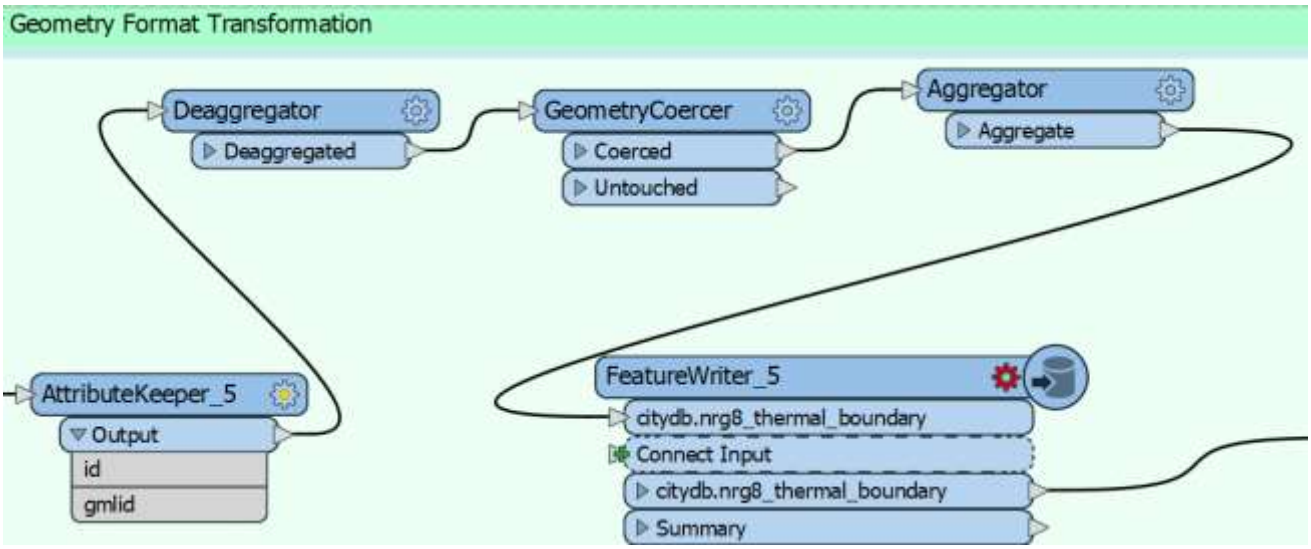


Figure 27: Geometry Transformation from FME_MultiSurface to FME_MultiPolygon.

By running the created FME workbench, EnergyADE part of CityGML test data which are needed for energy simulation process in Grasshopper Ladybug tools would be written into corresponding updatable views and filled across tables.

5. Grasshopper Honeybee Workflow implementation

Process of Grasshopper Honeybee implementation consists of following key procedures (Figure 28):

A. Database Connection. In previous steps of test data preparation, database approach has been adopted and CityGML with EnergyADE data has been imported into 3DCityDB database. To apply the mapping tables between CityGML EnergyADE model and Grasshopper Honeybee parameter inputs, all information of classes and attributes should be able to be retrieved from database in the first place.

B. Data Retrieval and Storage. After establishing database connection, next step is to retrieve required information from tables of 3DCityDB by running queries. Based on unique data structure adopted by Grasshopper and characteristics of its workflow, acquired information may need to be split, aggregated or stored in different ways to fulfill the input requirements of Grasshopper components.

C. Data Mapping. By applying mapping tables created before, CityGML EnergyADE attributes would be mapped to corresponding Honeybee input parameters. This mapping process includes geometry reconstruction (surface-zone approach vs zone-surface approach), semantics transformation and formats/units' conversion. As in Grasshopper, data could not be managed in tables with foreign keys referencing each other like in 3DCityDD, it requires more effort to keep track the order of input parameters. In other words, it is always essential to make sure that all inputs belong to same surface or zone and there is no mismatch with each other.

D. Automation of Grasshopper Workflow. One of the reasons to explore possibility to utilize CityGML EnergyADE data in Grasshopper Ladybug tools is that it contains many useful attributes for energy simulation. By retrieving values of those attributes and mapping them to corresponding input parameters, the tedious process of manually setting these inputs is eliminated and working efficiency is greatly improved. This advantage could not be fully realized if Grasshopper workflow is still 'single-threaded' – each surface or zone has to be treated individually by same flow of Grasshopper components before feeding to simulation engine. So, the automation of Grasshopper workflow has to be done by customizing Ladybug or Honeybee tools in order to take lists of surfaces/zones and export lists of outputs.
E. Energy Simulation. Running energy simulation with mapped inputs from customized workflows works as a test for previous steps. By checking the log (if there are severe errors during simulation process) and result (if wanted simulation outputs such as zone energy use and comfort metrics comply with simulation scenarios), we could see if CityGML EnergyADE data could be truly used in energy simulation of Ladybug tools and troubleshoot the possible flaws of design decision or coding bugs.
F. Simulation Result Visualization. Result visualization as last step is a showcase of powerful visual representation capabilities of Ladybug tools. It also firmly supports one of the initial research objectives: By combining richness of CityGML EnergyADE data together with abundant customization /visualization possibilities of Ladybug tools, energy simulation would be more convenient and user-friendly.

Figure 28: Grasshopper Implementation Process.

5.1 Database Connection

Currently there are two options to connect Grasshopper interface to database: using component slingshot! [32] with psqlODBC - PostgreSQL ODBC driver or utilizing GH Python Remote with Psycopg2 - Python PostgreSQL adapter. Using another component GH_CPython with Psycopg2 is no longer workable as the development of GH_CPython has been stopped since November 2017.
Slingshot! utilize Microsoft ODBC interface to access data of database management system using standard SQL language. Besides installing Slingshot! component itself, user needs to install ODBC driver for corresponding database (PostgreSQL, in this case). Installation and setting-up process of this approach is quite smooth following documentations on both websites. It only requires extra attention to add correct 64-bit or 32-bit system data source under System DSN (Figure 29). The workflow of Slingshot! component in Grasshopper is very straight forward and clean (Figure 30): User creates a CString which contains connection information (driver name, database name, user id, port, password and etc.) and SQL query scripts with panels. After turning on the boolean toggle, all query results as a string with values of different columns separated by comma would be retrieved. Another advantage of Slingshot! is that it works with multiple DBMS (Database Management System) like MySQL, PostgreSQL and even Excel. Besides ODBC driver, it also provides access to databases which support OLE DB interface, a newer and more advanced

standard (https://docs.microsoft.com/en-us/sql/connect/oledb/oledb-driver-for-sql-server?view=sql-server-ver15).

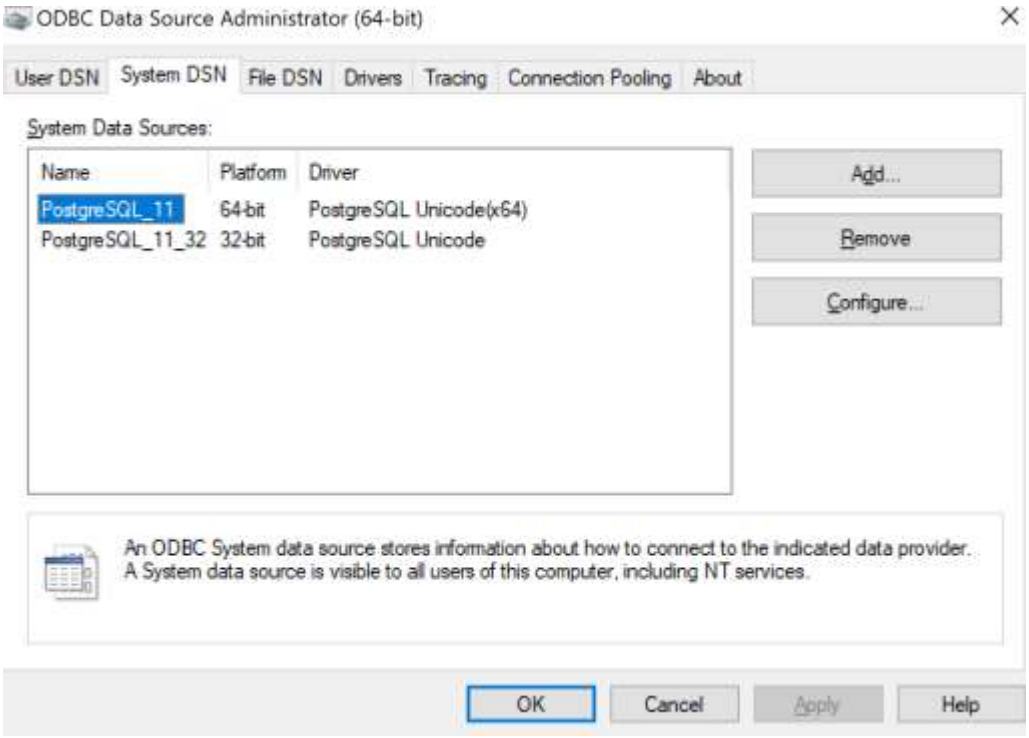


Figure 29: 32-bit and 64-bit of System DSN of ODBC adapter.

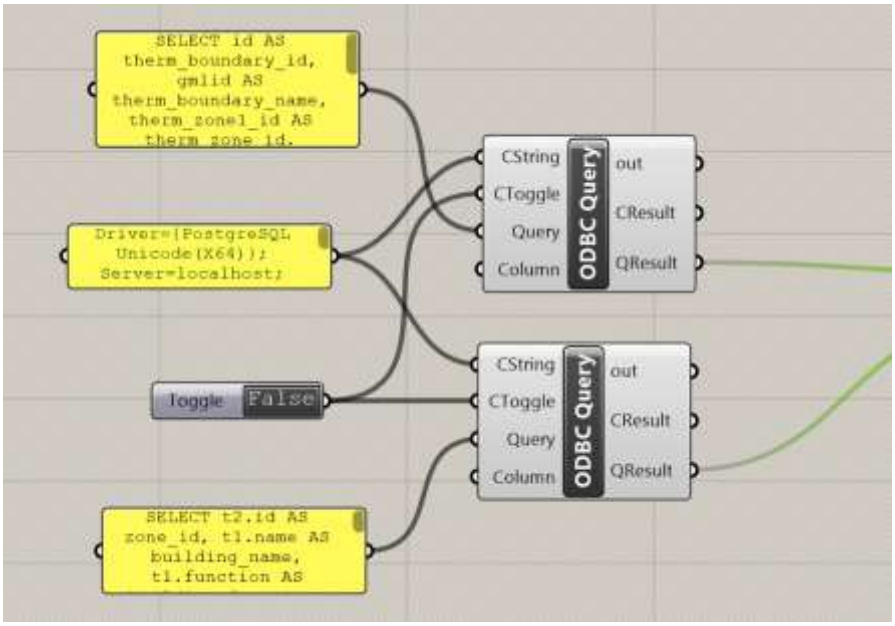


Figure 30: Slingshot! Workflow in Grasshopper.

GH Python Remote component uses Psycopg2, a python package which provides access to PostgreSQL database. Most part of Psycopg2 is implemented in C as libpq wrapper, thus it is very efficient and supports heavy multi-threading tasks. Different from Slingshot!, data retrieved by running SQL queries through Psycopg2 is returned automatically as a list with original PostgreSQL data types, not a string with values of different columns separated by commas. However, with all above-mentioned advantages, setting up Psycopg2 in Rhino Grasshopper is not very convenient. As Rhino implements IronPython which is tightly integrated with .NET framework (https://ironpython.net/), many of locally installed python packages like

NumPy, SciPy are not supported. And unfortunately, Psycopg2 is one of them. That is when GH Python Remote comes out. GH Python Remote connects Rhino Grasshopper to an external instance of python installed locally. In this way, all packages which could be used in python can be used in Rhino Grasshopper. During setting up process of GH Python Remote, user needs to specify location of installed python instance and desired packages/modules (Figure 31). After GH Python Remote is running successfully, these packages (Psycopg2) could be referenced in GHPython components to build up the database connection. Though GH Python Remote offers great flexibility to utilize other python modules in Grasshopper, but in the context of this research, extra setting up procedure of GH Python Remote, installation process and extra time needed to get familiar with Psycopg2 module make GH Python Remote + Psycopg2 a less appealing option.

Two queries (surface_queries and zone_queries) are sent to 3DCityDB database to retrieve geometries and attributes of thermal boundaries and thermal zones of CityGML EnergyADE data. Which attributes to be retrieved and how to store those attributes in proper data structure would be discussed in following sector.

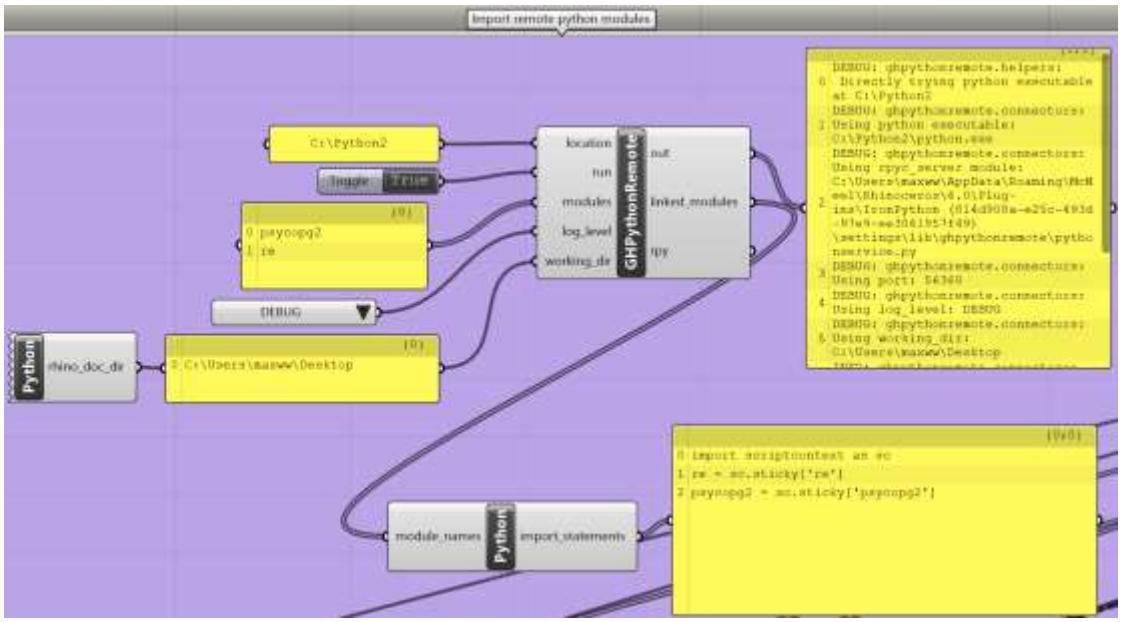


Figure 31: GH Python Remote Workflow.

5.2 Data Retrieval and Storage

Based on Honeybee energy modeling concepts , workflow and previously created mapping tables between CityGML EnergyADE data model and Grasshopper Honeybee input parameters (HBSurf attributes & HBZone attributes), two queries (surface_query and zone_query) are sent to 3DCityDB to retrieve related data for energy simulation (see Appendix for SQL scripts)

5.2.1 Surface_query

For surface_query which is mainly used to retrieve thermal boundary related information, data under following columns of nrg8_thermal_boundary view are acquired: id as HBSurface id, gmlid as HBSurface name, therm_zone1_id as HBZone id, multi_surf_geom as HBSurface geometry, type as HBSurface type.

The reason to use gmlid as name of HBSurface is that gmlid is unique (unlike name, which is only thermal_boundary_1 to thermal_boundary_7 for thermal boundaries across all thermal zones) and provides extra information of parent thermal zone; It has been mentioned in previous chapter of 3DCityDB that all geometry information is stored in table surface_geometry and linked to thermal boundary by thermal boundary id as foreign key. To simplify the SQL script, geometry information is directly retrieved from thermal boundary view under column multi_surf_geom; Type of thermal boundaries would be mapped to types of HBSurfaces.

5.2.2 Zone_query

For zone_query, which is used to retrieve HBZone related information, several views have to be joined together to access data from multiple classes. Data under following columns of view nrg8_thermal_zone are collected: id as HBZone id, infiltr_rate as infiltration rate, infiltr_rate_unit as infiltration_rate_unit, is_cooled and is heated are jointed together with values of following attributes of view nrg8_building: name as HBZone name, function as building function.

As shown in mapping tables, infiltration rate unit stored in 3DCityDB could be different from inputs of Honeybee, transformation needs to be made in later data mapping process; is_cooled and is_heated both indicate that HBZone is conditioned with air flow system as is_conditioned; Values of HBZone programs which is subclass of building programs are most similar to values of building functions.

If one is going to retrieve information of constructions and would like to link them to thermal boundaries and thermal openings, attributes values of view nrg8_construction should be joined together with view nrg8_cityobject_to_constr based on id of construction. View nrg8_cityobject_to_constr works as a reference table between city objects and constructions. They are linked to each other with cityobject_id. By referring foreign key cityobject_id, construction data could be further joined with thermal boundaries and thermal openings.

5.2.3 Data Storage

Unlike in 3DCityDB, in Rhino Grasshopper interface, it is very hard to link certain data to others based on id or any other foreign keys. The simplest and most efficient way to solve data affiliation problem is to use ordered lists to store all attributes and later split into ordered sub-lists. In surface_query, values retrieved are ordered (ascending as default) first by HBZone id and then by HBSurface id as a list. In zone_query, values acquired are sorted by HBZone id. Each sub list contains values of one column and values of element with same index across sub lists represent attributes of one HBSurface or HBZones.

Another thing worth noticing is that, Grasshopper does not handle nested lists (or, lists of lists) for data transfer between different components. Even the output data is set to ‘Graft’ mode, which enables list of data to be exported in different branches, the data in sub lists would only be recognized as IronPyton.Runtime.List object, not as a list of values (Figure 32). This affects mostly transfer of geometries when surfaces constructed by list of points are further aggregated into different lists by zones.

The Grasshopper’s solution to nested lists is Data Tree, an ordered collection of lists [36]. Nested lists are stored hierarchically on different branches with unique paths. Figure 33 shows an example of data tree

representation [37] with depth level 2: Six sub lists are stored on six different branches. On each branch, there is list of points. To query sub lists, users need to refer to path notation, which is consisted of curly brackets and semi-colons, like {0:1} for sub list on this path. To get an element of one sub list, index which should be integer needs to be followed after that path inside square bracket such as {0:5}[1]. Both path and index start with 0.

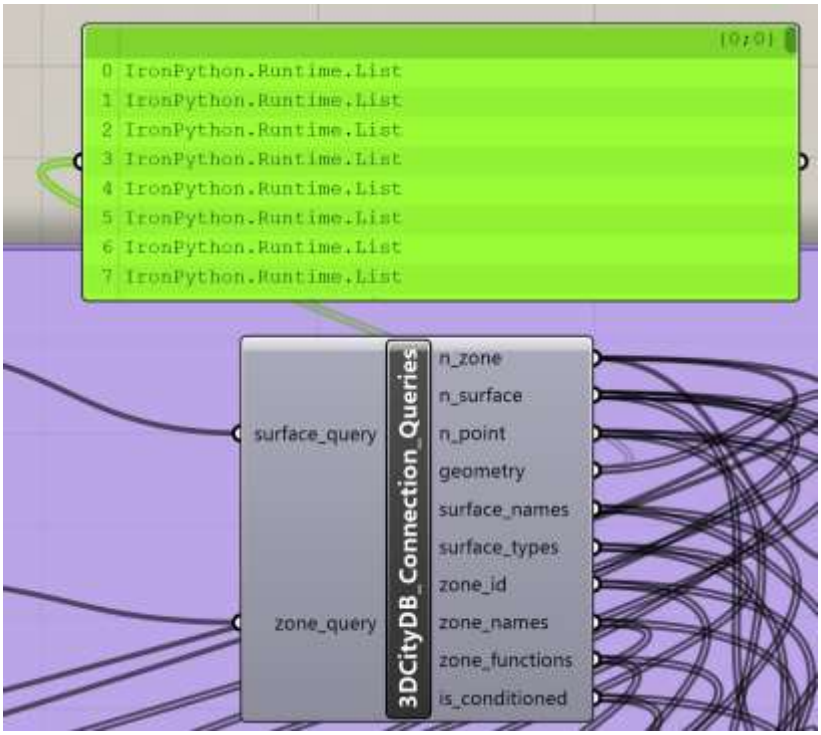


Figure 32: The way Rhino Grasshopper handles nested lists.

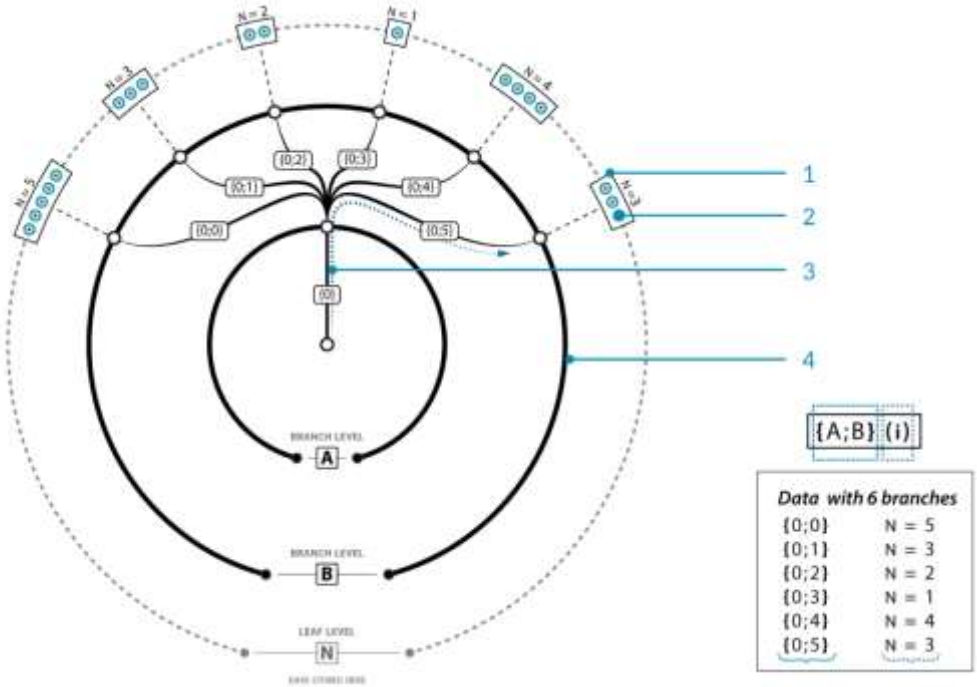


Figure 33: Data Tree Representation.

In data storage process, only geometry attribute needs to be stored in data tree structure. To make sure the elements of lowest-level sub lists are points, not individual x/y/z values, coordinates need to be transferred to point_3d objects using rhino grasshopper syntax method rs.CreatePoint() and then stored into data trees. Rhino Grasshopper website has provided comprehensive knowledge of transformation between nested lists and data trees [38]. A grasshopper component was built based on code provided above to store geometry data inside the nested lists to data tree.

5.3 Data Mapping

Like mentioned above, attribute values retrieved from surface query are mapped to HBSurfaces and those from zone query are mapped to HBZones based on simplified mapping tables below (table 3 and table 4) where semantics meaning of both sides and mapping notes are deleted for display reason. 1-1 mappings are carried out by dictionary. Please reference to Appendix for complete mapping tables.

Table 3: Mapping tables for HBSurface-based approach.

Honeybee HBSurface-based Approach Parameters	CityGML with EnergyADE Attributes
Radiance Material	AbstractMaterial:Gas AbstractMaterial:SolidMaterial
EnergyPlus Construction	AbstractConstruction:Construction
Surface Type	ThermalBoundary::thermalBoundaryType
Boundary Condition	--
Shade Material Name	ThermalOpening::indoorShading ThermalOpening::outdoorShading
Shading Schedule Name	--

Table 4: Mapping tables for HBZone-based approach.

Honeybee HBZone-based Approach Parameters	CityGML with EnergyADE Attributes of Classes
Building Program	_AbstractBuilding::function _AbstractBuilding::type
Zone Program	_AbstractBuilding::function _AbstractBuilding::type
Zone Floor Area	ThermalZone::floorArea
Zone Volume	ThermalZone::volume
Is Conditioned	ThermalZone::isCooled or ThermalZone::isHeated
Occupancy Schedule	UsageZone:Occupants::occupancyRate

Lightning Schedule	LightningFacilities::operationSchedule
Equipment Schedule	ElectricalAppliances::operationSchedule
Equipment Load Per Area	ElectricalAppliances::electricalPower
Infiltration Rate Per Area	ThermalZone::infiltrationRate
Lighting Density Per Area	LightningFacilities::electricalPower
Number of People Per Area	UsageZone:Occupants::numberOfOccupants
Ventilation Per Area	--
Ventilation Per Person	--
Ventilation Schedule	UsageZone::ventilationSchedule
Cooling SetPoint	--
Cooling SetBack	--
Heating SetPoint	--
Heating SetBack	--
Max Humidity SetPoint	--
Min Humidity SetPoint	--

Surface types for HBSurfaces are indicated by following integer-type matches: 0- 'WALL', 0.5- 'UndergroundWall', 1- 'ROOF', 1.5-'UndergroundCeiling', 2- 'FLOOR', 2.25- 'UndergroundSlab', 2.5- 'SlabOnGrade', 2.75-'ExposedFloor', 3- 'CEILING', 4- 'AIRWALL', 5- 'WINDOW', 6- 'SHADING' (Honeybee Premier). During Honeybee energy modelling process, HBSurface types are of great importance as it represents thermal dynamic characteristics of surfaces. Code list of thermal boundary types from EnergyADE data schema are of following values: interiorWall, intermediaryFloor, sharedWall, outerWall, groundSlab, basementCeiling, atticFloor, roof. A mapping decision has been made as following: roof – 1, outerWall – 0, intermediaryFloor – 2, groundSlab – 2.5, basementCeiling – 1.5, atticFloor – 2.75.
Zone programs of HBZones are subclass of building programs, which is consist of ‘Office’, ‘Retail’, ‘MidriseApartment’, ‘Warehouse’ and etc. Values of attribute thermal zone types are obviously categorized in different standard, not function-based rather than structure-based. In this way, building function is used to map with HBSurface types. Code list of CityGML EnergyADE building types could be referred to library online, which covers even more building typologies based on different functions [39]. For test data, 12 buildings are all ‘residential building’ and it is mapped to ‘MidriseApartment::Apartment’. Notation of double colon refers to subclass of building program.
Attribute is_conditioned is a boolean variable indicating if HBZone has air flow equipment. isCooled and isHeated are both boolean variables and if either of them is True, is_conditioned should be true.
Infiltration rate, occupants’ number and electrical power are mapped to corresponding classes of CityGML Energy ADE. User of mapping tables should notice that though these attributes seem to share semantic meanings on both sides, their units might be different, depending on unit values in 3DCityDB tables. Most of

these input parameters of Honeybee are generalized by zone floor area. Attributes units of EnergyADE classes, on the other hand, are often not.

The last major ‘mapping’ process is to reconstruct surface geometries in Rhino Grasshopper. By implementing DataTree.AllData() method, all vertices of surfaces could be retrieved. Like emphasized before, it is important to keep the order of point inputs so that by referring to number of points per surface as index, surface geometries could be reconstructed by using Rhino Grasshopper syntax methods.

Two methods have been tested to reconstruct surface geometries: Mesh approach and (Brep)Surface Approach. Surface geometries could be built up as mesh by using rs.AddMesh(vertices,face_vertices) method. The idea is to import all points of one building all together and by figuring out indexes of different face vertices, generate the mesh for the whole building. One major flaw of this method is that, in Rhino Grasshopper, mesh could only be triangulated or quad. In other words, if a surface has more than four vertices, it has to be split into triangulated or quad-meshes. Surfaces of buildings are split into multiple sub-surfaces and it is difficult to code for all potential situations. As a result, in this research, second approach (Brep) Surface approach is implemented. The workflow of surface approach is as following: First construct closed polyline for each single surface from ordered points and number of points per surface. Later, generate (brep)surface from closed polylines using rs.AddPlanarSrf(polyline) method. During this process, it should be noticed that to construct a closed polyline, start and end point of polyline should be of same coordinates and avoid feeding all closed polylines as a list to rs.AddPlanarSrf method. If a list of closed polylines are used as input, the order of polylines used to generate surface is non-sequential. This means generated surfaces are not ordered ascending firstly by zone _id and then by surface_id. This would cause great problems in creating HBZones.

Figure 34 shows the results of geometry reconstruction for test data. Each surfaces is created as complete (brep)surface which could be directly used to generate HBSurfaces.

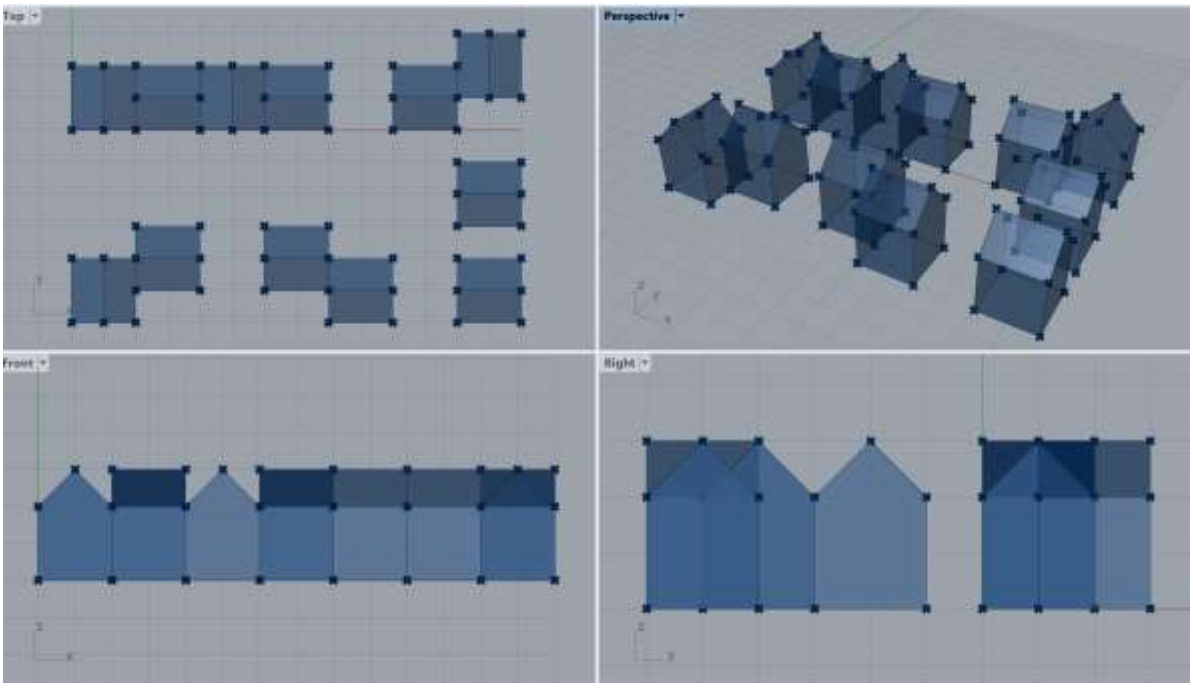


Figure 34: Reconstructed Surface Geometries.

5.4 Honeybee Simulation Workflow Automation

With all geometry and semantic information correctly reconstructed and mapped to required input parameters, the next step is to build the Honeybee simulation workflow. As stated in chapter of Methodology and Implementation Design, a surface-based workflow has been adopted in this research: Surface Geometries with Attributes – HBSurfaces – HBZones with Attributes – Run Simulation. However, traditional surface-based approach of Honeybee simulation is basically surface by surface and zone by zone: User need to generate HBSurfaces one by one and each time with a createHBSrfs component. This workflow indeed offers a great flexibility and possibility of customization for single zone or single building energy simulation as user could assign a variety of parameters to each surface. However, as CityGML EnergyADE data already provides rich information and this research focuses on larger scale energy simulation, the Honeybee surface-based workflow needs to be automated to improve the efficiency and productivity.

Automation of surface-based workflow is mostly done by customizing two very important Honeybee component: createHBSrfs and createHBZones: createHBSrfs takes surface geometry and surface attributes as inputs and generate HBSurfaces. Original data inputs of createHBSrfs are individual geometry and attributes values. Data inputs are now set to ‘List Access’ and the code has been adapted to register list of HBSurfaces; createHBZones components takes list of HBSurfaces which construct single HBZone and multiple attributes values (HBZone program for instance) as inputs to generate the zone. Like createHBSrfs, input data are set to ‘List Access’ and code of component is adjusted to generate a list of HBZones as output. Unlike inputs of createHBSrfs components, inputs lists of createHBZones need to be spliced to sub lists for different zones based on number of surfaces per zone. Number of surfaces per zone could be queried by SQL statement (join and count) or counting the surfaces with same zone id in Grasshopper.

To check if attributes as list inputs are assigned correctly to each HBSurfaces and HBZones, user could utilize Honeybee tools to retrieve attributes values from HBSurfaces or decompose HBZones to different types of surfaces and visualize the result. In this research, getSetHBObjName component is used to retrieve names from generated HBSurface to see if names mapped from gmlid of thermal boundaries have successfully being assigned (Figure 35 - left). To check if types of HBSurfaces or EnergyPlus boundary conditions are correct, one could use component decomposeByType or decomposeBasedOnBoundaryCondition. In this research, every HBZone is decomposed into different types of HBSurfaces with different colors (Figure 35 - right).

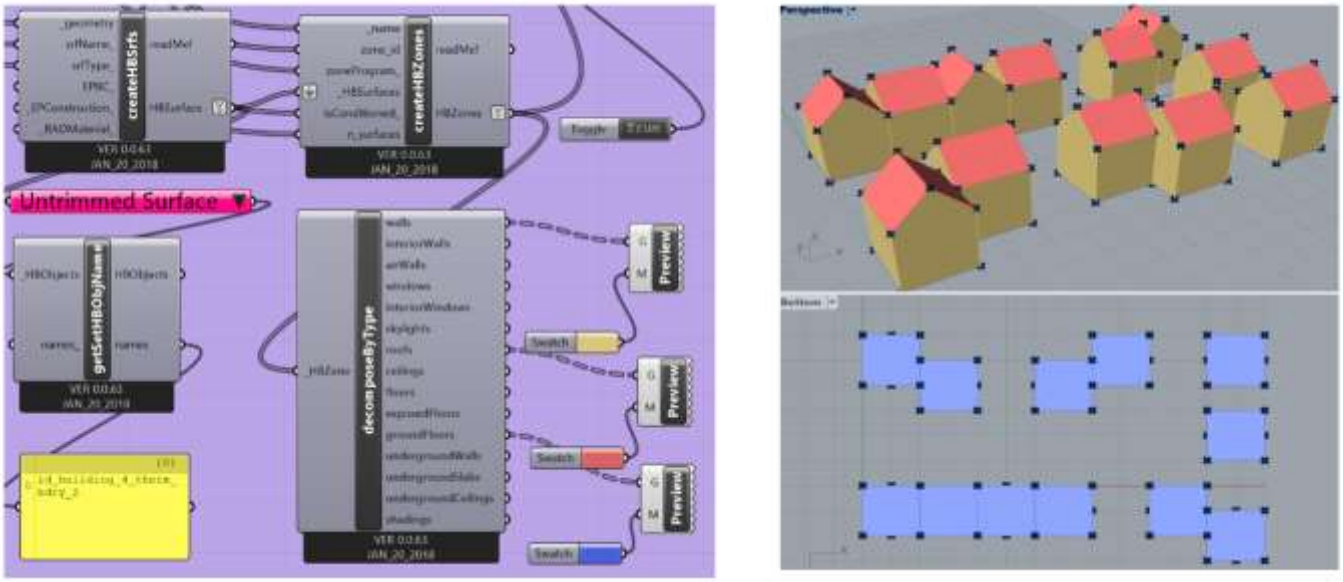


Figure 35: Checking lists inputs are correctly assigned by coloring different types of HBSurfaces.

5.5 Energy Simulation

Generated HBZones are then plugged into runEnergySimulation component as HBZone inputs. An .EPW weather file which offers local metrological information is also compulsory. By using analysisPeriod and EPOutput components, time series for energy simulation and output parameters could be configured. Output simulation results is saved locally as .csv file with data entries. In this research, weather file generated near Schiphol Airport is used and analysis period is set from January to March in hourly step. Outputs relevant to zoneEnergyUse (heating, cooling, electricity) and zoneComfortMetrics (mean air temperature, relative humidity) are set to be generated during simulation process. Figure 36 is part of output results in CSV file.

5.6 Simulation Result Visualization

The last step of result visualization is totally a showcase of powerful data display functions of Ladybug and Honeybee tools. CSV format of output results are difficult to read and use in design stages. As only data visualization components of Ladybug and Honeybee are being tested and no coding work or research activities are included, aiming to generate more intuitive figures for better interpretation of energy simulation results.

	A	B	C	D	E	F	G
1	Date/Time	Calendar Y	GIORGIO'S_	PAOLA'S_P.	GIOVANNI'	AURONTE'S	YODA'S_HL
2	01/01 01:00:00		285589.4	285589.4	285589.4	285589.4	285589.4
3	01/01 02:00:00		285589.4	285589.4	285589.4	285589.4	285589.4
4	01/01 03:00:00		285589.4	285589.4	285589.4	285589.4	285589.4
5	01/01 04:00:00		285589.4	285589.4	285589.4	285589.4	285589.4
6	01/01 05:00:00		797092.8	797092.8	797092.8	797092.8	797092.8
7	01/01 06:00:00		1679436	1679436	1679436	1679436	1679436
8	01/01 07:00:00		1875513	1875513	1875513	1875513	1875513
9	01/01 08:00:00		1675174	1675174	1675174	1675174	1675174
10	01/01 09:00:00		733154.9	733154.9	733154.9	733154.9	733154.9
11	01/01 10:00:00		507240.9	507240.9	507240.9	507240.9	507240.9
12	01/01 11:00:00		507240.9	507240.9	507240.9	507240.9	507240.9
13	01/01 12:00:00		507240.9	507240.9	507240.9	507240.9	507240.9
14	01/01 13:00:00		507240.9	507240.9	507240.9	507240.9	507240.9
15	01/01 14:00:00		507240.9	507240.9	507240.9	507240.9	507240.9
16	01/01 15:00:00		507240.9	507240.9	507240.9	507240.9	507240.9
17	01/01 16:00:00		878080.9	878080.9	878080.9	878080.9	878080.9
18	01/01 17:00:00		1871250	1871250	1871250	1871250	1871250
19	01/01 18:00:00		2625718	2625718	2625718	2625718	2625718
20	01/01 19:00:00		3533636	3533636	3533636	3533636	3533636

Figure 36: Part of Energy Simulation Results in CSV file.

6 Results Analysis, Conclusion and Future Work

6.1 Results Analysis and Conclusion

This research is not focusing on analysis of energy simulation results but on results of file-based approach implementation: Data Mapping, Data Writing into Database, Data Retrieving and Feeding into Ladybug Tools (Figure 37).

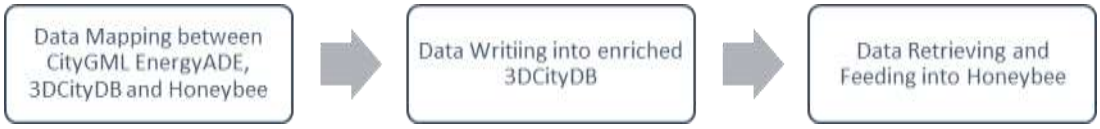


Figure 37: Main Steps of File-based Approach Implementation.

Based on outcomes of above-mentioned procedures (mapping tables, FME workbench, 3DCityDB 'Plus' instance with CityGML EnergyADE data, complete Ladybug & Honeybee energy simulation workflow), following conclusions could be made for current research process:

A. Data models of CityGML with EnergyADE shares great similarities with input parameters of Honeybee (or EnergyPlus). As discussed in previous chapters, they have same material – layer – construction - surface (thermal boundary) – zone (thermal zone) based concepts of energy modeling and a great number of common attributes with alike semantic meanings. They also handle loads and schedules of zones. It does require transformation between class attributes, both geometrically and semantically, in formats or

even units. But as indicated in mapping tables and implemented Rhino Grasshopper workflow, which is automated and runs without problems, these transformations could be carried out without great difficulties in- or outside Grasshopper. There are certain parameters missing in EnergyADE data schema or the parameters are scatter into multiple parts across tables, such as boundary conditions of surfaces, ventilation attributes (ventilation per area or person), cooling/heating/humidity setpoints and setbacks. But these parameters are not compulsory in energy modelling process of Honeybee and some of these would be assigned automatically based on other inputs or results of Honeybee calculations with default values (for example, surfaces with different types would be assigned various EnergyPlus boundary conditions accordingly during simulation process). In general, this research has successfully provided an applicable workflow or solution of utilizing CityGML EnergyADE data in Ladybug tools (Ladybug + Honeybee) that is friendly for common users, like architects, urban planners or landscape designers who have little programming knowledge with both efficiency and productivity. Many design concepts and technical details of data mapping or transformation (like proper Rhino Syntax methods to reconstruct surface geometries) could be useful for other researchers who would like to explore further. The research also indicates that Ladybug tools could supplements traditional ways of using CityGML data with interactive/ intuitive interface and powerful visualization functions;

B. Comparing to file-based approach, which requires hard-decoding of CityGML data structure, retrieving attributes and does mapping internally in program with Honeybee input parameters, database approach has proved itself to be more 'light-weighted', flexible and convenient. Rhino Grasshopper also provides intuitive adapters to DBMS which could be integrated seamlessly into Rhino Grasshopper workflow. Modular design in Rhino Grasshopper implementation workflow which separates functionalities like database connection and queries, data mapping and transformation, Honeybee energy simulation and results visualization. provides great flexibilities to adapt current approach to many other projects. Also, the 'bottom-up' approach of constructing simulation scenarios starting from HBSurfaces then to HBZones fits better in this research than 'top-down' methodology (zones geometries – HBZones – HBSurfaces) as it is harder to match surfaces of HBZones and their properties accordingly after HBZones are created.

C. Under surface-zone approach, original 'single-thread' Honeybee energy simulation workflow could be optimized and automated. By changing inputs from 'Item Access' to 'List Access' and adjusting code of components accordingly, two most important steps – constructing HBSurfaces and HBZones could intake lists of inputs and generate multiple HBSurfaces and HBZones. Branches of workflow would be simplified as there is no need for multiple surface or zone generators. This automation process is still light-coded and easy to implement as codes of Honeybee components are in python with abundant comments. It is also important to notice the difference between IronPython implemented by Rhino Grasshopper and original python we are getting used to as many python packages installed locally, such as NumPy or SciPy cannot be used in GHPython component. Data Tree structure, the way Grasshopper handles nested lists is also another important feature to get used to as 'lists of lists' could not be transformed between different Grasshopper components. Automation of Honeybee simulation workflow complies with objectives of this

research: Facilitate Honeybee energy simulation with richness of CityGML EnergyADE data and improves the efficiency and productivity.

D. During data retrieving, storage and mapping process in Grasshopper workflow, it is always essential to keep the sequential order of different inputs lists. When CityGML with EnergyADE data is imported into 3DCityDB and managed by tables, affiliation between different attributes could be managed by foreign keys referencing to each other. However, in Grasshopper, considering troublesome interaction with data tree structure, it is not always convenient to check keys and attributes stored in nested lists. The easier yet most efficient way to keep the matching relationship between different data inputs is maintaining sequential order of input elements in different lists of attributes. The approach in this research is to order data retrieved by SQL queries ascendingly by zone_id first and then by surface_id. With the help of other important parameters like number of vertices per surface, number of surfaces per zone, it is convenient to reconstruct data structure. To keep lists of data always in order, users may not be tempted to use some Rhino Grasshopper Syntax methods which take entire list as inputs and generate outputs randomly. For example, method AddPlanarSrf(polyline) could take lists of closed polylines and generated corresponding surfaces. However, if polylines are imported as a single list parameter, output results of surfaces would not be in same order. This would make reconstruction of zones from surfaces impossible as it is difficult to match surfaces to corresponding zones. On the contrary, by inputting polylines into that method one by one, such as using a loop, output surfaces would be in same order with inputs and also matches the ordered list of surface number per zone.

E. Both CityGML EnergyADE data schema and Ladybug tools (Ladybug & Honeybee) are pretty young and growing fast in last several years. There would be changes in data modeling concepts, data structure as well as semantic meanings of attributes on both sides. Mapping tables created of surface attributes and zone attributes in this research might be outdated when next edition of CityGML EnergyADE standard is coming out. Also, 3DCityDB utility package has to be updated to map classes and attributes of EnergyADE into corresponding 3DCityDB views or tables. All this does not make this research ‘outdated’. The way to carry out data mapping from understanding energy modelling concepts of Ladybug tools, the idea of hierarchical design of FME workbench and preset operations on cascaded ‘updatable’ views, choices made in options of database connection and data storage/reconstruction approaches for Rhino Grasshopper workflow, automation of Honeybee energy simulation process and ways to customize Honeybee components are valuable experience for further relevant studies.

6. 2 Future Work

Current research could be expanded in following aspects to fully customized potential of using CityGML EnergyADE data in Ladybug tools:

A. Energy modeling of more buildings on larger scale with multiple zones. Input data for this research is consisted of 12 buildings and each one with single thermal/energy zone. Comparing to other options of urban energy modeling like CitySim which could handle thousands of building blocks in CityGML formats,

the boundary or limit of modeling scale of Honeybee needs to be tested. Also, in real-life cases, a building is often consisted of different functional space (living room, bedroom, attic, kitchen and basement) with various loads , structure and construction. Thermal dynamic characteristics of these individual zones are vastly different and modelling building with single zone could be over-generalized in some situations. To solve this problem, multiple thermal zones of buildings need to be imported into Honeybee and aggregated by individual building to carry out energy simulation. An example of multi-zone simulation is as Figure 38 below. As Honeybee is currently running the simulation zone by zone, research needs to explore possibility of aggregation multiple zones as single input for simulation or just merging the simulation the results.

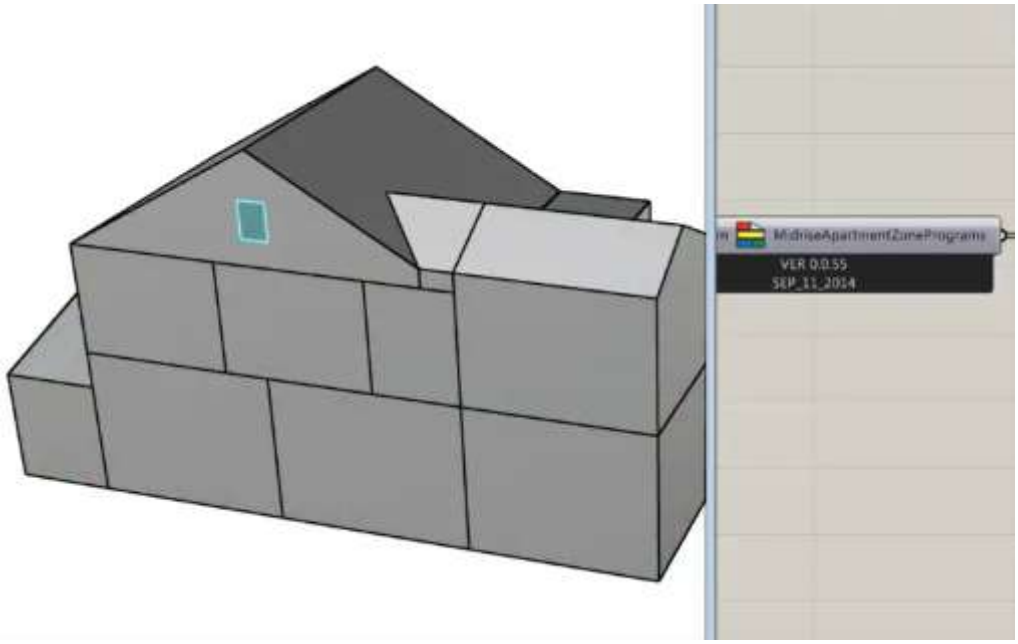


Figure 38: A building with multiple HBZones. [40]

B. Better mapping with material and constructions. As already mentioned in previous chapters, Honeybee mainly uses pre-built libraries (Radiance Material library or EnergyPlus Construction library) to manage material and constructions. EnergyADE data schema, on the other hands, use a combination of multiple thermal and optical properties to define a material. Both Honeybee and EnergyADE combine layers with different material types in a certain order to build the constructions. Unlike Honeybee which stores constructions in a library, EnergyADE link constructions with layers and materials by foreign keys and reference them to corresponding city objects (thermal boundaries and thermal openings) by cityobject_id. One way to map the construction and material data of EnergyADE to Honeybee is to build a customized material or construction with thermal and optical parameters. However, parameters or properties used in Honeybee are different from that implemented in EnergyADE data model. All of these reasons cause great difficulty to map constructions and material from CityGML EnergyADE model to Honeybee inputs. As Radiance Material library and EnergyPlus Construction library are adopted universally across academic community and practical fields it might be better for future EnergyADE schema to include code lists for entries in those libraries.

C. Dealing with complex geometries (like curved surfaces) and spatial relationships (shared walls or adjacent surfaces). For both CityGML specification and Honeybee or EnergyPlus geometry rules, a two-

dimensional geometry primitive, or a surface has to be planar. As a result, complex curved roofs or facades of buildings have to be broken into planar sub-polygons/surfaces. CityGML with EnergyADE data model manages this situation with class multi-surfaces referencing to thermal boundaries with complex geometries and this would increase the depth of nested list geometries by one level and bring challenges to geometry reconstructions in Grasshopper interface. In Grasshopper, there is a workaround to deal with inputs with curved surface geometries by first writing them into IDF files and reimported into Grasshopper canvas. In this way, curved surfaces would be automatically deaggregated into planar surfaces. In this research, only planar surfaces are considered, and no multi-surfaces classes are included. Beside complexity of curved geometry, spatial relationships of some buildings in test data are not handled so well in this research: some pairs of buildings are attached to each other with adjacent surfaces and adjacent surfaces are only partially overlapped. For Honeybee or EnergyPlus energy simulation, shared surfaces must be of same shape and size to ensure principals of energy conservation. However, in CityGML EnergyADE, those extra shared surface geometries are not compulsory. In this way, when imported into Honeybee to run energy simulations, heat transfer between adjacent surfaces would not be simulated correctly. Honeybee provides a solution for this problem. A component named intersectMasses takes adjacent building masses as inputs and automatically generate shared surface geometries on both surfaces. Figure 39 shows the output results after of both masses after running through this component. As in this research, a surface-zone approach is implemented and geometries are directly transfer to HBSurface then to HBZones, those adjacent surfaces remain unsolved. Also in Honeybee, HBSurfaces should be convex and it is another restriction.

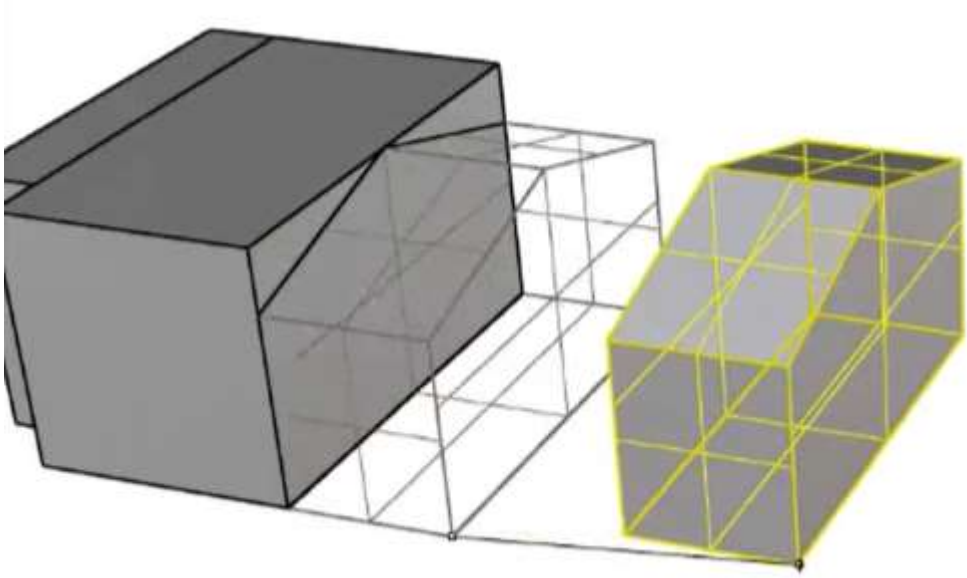


Figure 39: Shared-surface Geometry Automatically Created after running IntersectMasses Component. [41]

D. Handling time-series data of CityGML EnergyADE model and mapping it with schedule in Honeybee. In EnergyADE data schema, information of time-series is saved under column values_array with extra information like time-series unit, time interval, begin and end time of temporal extent. Time-series are referenced by schedules by foreign key time_series_id. Honeybee provides support of customized schedules with different data types too (https://github.com/ladybug-tools/honeybee-legacy/wiki/Custom-Schedules) : fraction schedule with value between 0 to 1 which is always multiplied by maximum load value

to calculate simulated load; on or off schedules with boolean values 0 or 1 showing a device is on or off during certain period and etc. By defining a customized schedule in Honeybee, user may be able to map the schedules with time-series values from EnergyADE to Honeybee schedule inputs. For this research, only time-series with boolean type are included and as time_series_id of schedules are null, a link between schedules and time-series data could not be established.

In general, future work on using EnergyADE data in Rhino Grasshopper would focus on larger scale, more realistic and complex simulation scenarios with considering of occupants and appliances activities.

Bibliography

[1]: Agugiaro, G., Benner, J., Cipriano, P., & Nouvel, R. (2018). The Energy Application Domain Extension for CityGML: enhancing interoperability for urban energy simulations. *Open Geospatial Data, Software and Standards*, *3*(1), 2.

[2]: Ladybug Primer. https://docs.ladybug.tools/ladybug-primer/

[3]: Dragonfly – Urban Weather Generator Workflow Example. http://hydrashare.github.io/hydra/viewer?owner=chriswmackey&fork=hydra_2&id=Urban_Weather_Generator_Workflow&slide=0&scale=1&offset=0,0

[4]: Perez, D., & Robinson, D. (2012). Urban energy flow modelling: A data-aware approach. In *Digital Urban Modeling and Simulation* (pp. 200-220). Springer, Berlin, Heidelberg.

[5]: Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing?. *International journal of human-computer studies*, *43*(5-6), 907-928.

[6] [8]: Filchakova, N., Robinson, D., & Thalmann, P. (2009). A model of whole-city housing stock and its temporal evolution. *Proc. Building Simulation, Glasgow, UK*.

[7]: Forrester, J. W. (1970). Urban dynamics. *IMR; Industrial Management Review (Pre-1986)*, *11*(3), 67.

[9]: Sartori, I., Wachenfeldt, B. J., & Hestnes, A. G. (2009). Energy demand in the Norwegian building stock: Scenarios on potential reduction. *Energy Policy*, *37*(5), 1614-1627.

[10]: Urban Weather Generator https://github.com/ladybug-tools/uwg

[11]: Wate, P., & Coors, V. (2015). 3D data models for urban energy simulation. *Energy Procedia*, *78*, 3372-3377.

[12]: City Doctor https://www.citydoctor.eu

[13]: Zakhary, S., Allen, A., Siebers, P. O., & Robinson, D. (2016). A computational workflow for urban micro-simulation of buildings' energy performance.

[14]: CitySim https://leso.epfl.ch/transfer/software/citysim/

[15]: Agugiaro, G., Hauer, S., & Nadler, F. (2015). Coupling of CityGML-based semantic city models with energy simulation tools: some experiences. In *REAL CORP 2015. PLAN TOGETHER–RIGHT NOW–OVERALL. From Vision to Reality for Vibrant Cities and Regions. Proceedings of 20th International Conference on Urban Planning, Regional Development and Information Society* (pp. 191-200). CORP–Competence Center of Urban and Regional Planning.

[16] [18]: Peronato, G., Kämpf, J. H., Rey, E., & Andersen, M. (2017). *Integrating urban energy simulation in a parametric environment: a Grasshopper interface for CitySim* (No. CONF).

[17] [19]: Mutani, G., Coccolo, S., & Kämpf, J. (2018). CitySim Guide.

[20][21]: Honeybee Wiki https://github.com/ladybug-tools/honeybee-legacy/wiki

[22]: Gröger, G., Kolbe, T. H., Nagel, C., & Häfele, K. H. (2012). OGC city geography markup language (CityGML) encoding standard.

[23]: CityGML EnergyADE UML Diagram http://www.citygmlwiki.org/images/f/fb/UML-Diagrams_EnergyADE.pdf

[24]: Honeybee Premier https://mostapharoudsari.gitbooks.io/honeybee-primer/content/

[25]: CityGML codelist library https://www.sig3d.org/codelists/citygml/2.0/

[26]: 3DCityDB https://www.3dcitydb.org/3dcitydb/

[27]: 3DCityDB V3.3 Documentation https://www.3dcitydb.org/3dcitydb/fileadmin/downloaddata/3DCityDB_Documentation_v3.3.pdf

[28]: 3DCityDB Utility Package https://github.com/gioagu/3dcitydb_ade

[29] [30]: 3DCityDB Utility Package Manual https://github.com/gioagu/3dcitydb_utilities/blob/5d365ba48d27b99b9781227a3b7ed8a0359a29a7/manual/3DCityDB_Utilities_Package_Documentation.pdf

[31]: Feature Manipulation Engine (FME) https://www.safe.com/fme/

[32]: Slingshot! https://www.food4rhino.com/app/slingshot

[33]: psqlODBC driver https://odbc.postgresql.org/

[34]: GHPython Remote https://www.food4rhino.com/app/gh-python-remote

[35]: Psycopg2 https://pypi.org/project/psycopg2/

[36]: Grasshopper Data Tree https://www.grasshopper3d.com/forum/topics/the-why-and-how-of-data-trees

[37]: Grasshopper Data Tree Example https://modelab.gitbooks.io/grasshopper-primer/content/1-foundations/1-5/2_what-is-a-data-tree.html

[38]: Transformation between Data Tree and Nested Lists https://developer.rhino3d.com/guides/rhinopython/grasshopper-datatrees-and-python/

[39]: CityGML EnergyADE code list library https://www.sig3d.org/codelists/citygml/2.0/building/2.0/_AbstractBuilding_function.xml

[40] [41]: Honeybee Energy Modeling Tutorial The Laws of Geometry – Adjacent Surfaces https://www.youtube.com/watch?v=cDvBWDA0aF0&list=PLruLh1AdY-SgW4uDtNSMLeiUmA8YXEHT_&index=10