Student thesis series INES nr 558

Design, implementation and evaluation of a daylight estimation tool using 3D city model data

Peter Nezval

2021
Department of
Physical Geography and Ecosystem Science
Lund University
Sölvegatan 12
S-223 62 Lund
Sweden

Peter Nezval (2021).

Design, implementation, and evaluation of a daylight estimation tool using 3D city model data

Master's degree thesis, 30 credits in *Geomatics*
Department of Physical Geography and Ecosystem Science, Lund University

Level: Master of Science (MSc)

Course duration: *January* 2021 until *September* 2021

Disclaimer

Design, implementation, and evaluation of a daylight estimation tool using 3D city model data

Peter Nezval

Master thesis, 30 credits, in *Geomatics*

Supervisor:

Lars Harrie
Dep. of Physical Geography and Ecosystem Science, Lund University

Exam committee:

David Tenenbaum
Dep. of Physical Geography and Ecosystem Science, Lund University

Karolina Pantazatou
Dep. of Physical Geography and Ecosystem Science, Lund University

Acknowledgments

First of all, I would like to thank my supervisor Lars Harrie. His constructive feedback, support, and deep knowledge within the GIS field has significantly boosted the entire thesis project since the earliest stage. His supportive and professional attitude has been important especially in this pandemic period, when almost all the communication has been carried out online. Also, I would like to thank Per-Ola Olsson for providing help and support with the data in the beginning of the project.

I would also like to send big thanks to Hongchao Fan and Gefei Kong, from The Norwegian University of Science and Technology for providing me with the 3D city model. Also, I would like to thank Fredrik Lindberg for developing UMEP as an open project which can be freely used. This thesis project would be very difficult to accomplish without their support and tools.

Huge thanks go to all my colleagues from FOSSID, which has meanwhile become a part of Snyk, for their support. There are also a few people in the company I would like to particularly mention. Thanks to Jon Aldama for hiring me as a student without many experiences within the industry, for providing me a great flexibility to combine my job with the studies as well as for an opportunity to develop professionally and personally within a highly growing business sector as IT. Thanks to Max Gnipping and Gabriel Manea for being such awesome and tolerant managers, who always allowed me to prioritize work on the thesis project, if needed. Also, a special thanks to Michael Poe for a great company during our "lunchtime running sessions" and for listening to my concerns when I got stuck in an ostensible dead end.

Another important group of people who were very important during the entire thesis project and last two years are my fellow students from the *Geomatics* master program. It was nice to spend so much time with all of you while dealing with study related or personal errands. Although not being a *Geomatics* student, special thanks go to Kerstin for all her help and support and also "for reminding me to celebrate the small wins."

Last thanks are sent to my homeland, to Slovakia. Thanks to my best buddies for always being crazy and raising my spirit anytime I felt down. I am so grateful for our friendships, and especially for the fact that the distance did not weaken them at all. An absolutely colossal thanks goes to my family. Things might have gone up and down, but you have always been there for me. I cannot even describe how thankful I am. Ďakujem za všetko, nesmierne Vás ľúbim!

Abstract

Solar energy is an important component of sustainable urban development. However, it is still not reaching its full potential due to several reasons. One of them is a lack of free tools based on open geodata capable of estimating solar energy (daylight) metrics on building features, e.g., windows, in the early stages of the urban planning process. To fill this gap, a design and implementation of the "Daylight Estimation Tool" is proposed in this thesis project. All inputs required by the tool can be created in free and open-source software. The tool uses two primary data sources – UMEP (Lindberg et al., 2015) energy simulation output files and a 3D city model. The tool performs several operations resulting in the estimation of solar irradiance values for window features, the geometry of which is retrieved from the 3D city model. The output of the tool is an extended city model as well as a report with information about irradiance values on building surfaces. The extension of a 3D city model is done by adding the attributes to the semantic objects and by adding material objects for visualization purposes. The tool is evaluated against the test dataset created by the UMEP simulation tool and online VGI3D platform (Fan et al., 2021). The tool itself is a free and open-source project licensed under an MIT license and available on GitHub.

Keywords: *3D city model, CityJSON, solar energy, free and open-source software, 3D feature extraction, solar urban planning, sustainable urban development*

Abbreviations

ADE - Application domain extension
BIM - Building information model
CAD - Computer aided design
CLI - Command line interface
CRS - Coordinate reference system
DEM - Digital elevation model
DET - Daylight estimation tool
DSM - Digital surface model
EPBD - Energy Performance of Buildings Directive
EPW - EnergyPlus Weather format data
FOSS - Free and open-source software
GDP - Gross domestic product
GIS - Geographic information system
GML - Geographic markup language
GPL - General public license
IEA - International energy agency
ISO - International organization for standardization
JSON - JavaScript Object Notation
LoD - Level of detail
OGC - Open geospatial consortium
OSGeo - Open-source geospatial foundation
OSI - Open-source initiative
UMEP - Urban Multi-Scale Environmental Predictor
UML - Unified modeling language
VGI - Volunteered geographic information
XML - Extensible markup language
XSD - XML schema definition

1 INTRODUCTION

1.1 Background

Urban areas represent a major contributor of greenhouse gas production that is undoubtedly one of the most significant drivers of climate change. In 2015, cities accounted for 70% of worldwide CO2 emissions. At the same time, urban areas covered only a small fragment of the entire global area - 2% in 2015 and 3% in 2020 (Nouvel et al. 2015) - while urban inhabitants represented 55% of the world‘s population in 2018. Projections estimate that the continuous migration of the globe‘s inhabitants from rural to urban areas will continue. By 2050, almost 70% of the global population will be living in cities (United Nations 2018). Consequently, demand for housing will increase, putting the entire building sector under growing pressure. Residential and non-residential buildings coupled with the building sector (construction and operations) already accounted for the largest share of both global final energy use (36%) and energy-related CO2 emissions (39%) in 2018 (IEA et al. 2019). Relevant authorities should take immediate action in recognizing the key trends in urbanization and considering the data about emissions produced by buildings and the building sector. Such action could be in the form of a rapid transition of urban areas towards energy efficiency and their adaptation to challenges created by climate change (Nouvel et al. 2015). This would involve collaboration between all key players in the building sector, such as municipalities, energy suppliers, construction companies, architects, housing companies and private property owners. Their mutual cooperation can result in the creation and implementation of city policies for sustainable development as well as low-carbon urban energy strategies. The European Union recognized the importance of sustainable city development and established a corresponding legal framework. It is framed by two directives – the Energy Performance of Buildings Directive 2010/31/EU (EPBD) and the Energy Efficiency Directive 2012/27/EU, amended in 2018 and 2019 respectively. Despite the changes, the main idea of these directives remained untouched - to raise awareness in Europeans on how to save energy and money while keeping in mind a common long-term strategic goal, namely, to create an energy-efficient and decarbonized building stock by 2050 (European Commission 2021).

There are already existing methods on how to diagnose the energy performance of buildings. Popular solutions for architects and urban planners usually include CAD or BIM platforms in combination with energy simulation software (Jakica, 2017). Zooming out at the city/neighborhood level, GIS and geodata offer suitable tools and datasets to analyze energy interactions between buildings within the local energy system. 3D geospatial technologies usually known as 3D city models are becoming a popular choice for many municipalities and urban planners. 3D city models serve as a tool to estimate the impact of several urban components

(buildings, vegetation, roads, city furniture) on their surroundings, and to visualize urban studies while increasing the quality of the decision-making process (Agius et al. 2018). 3D city models can also be applied in the analysis of solar access and the estimation of solar potential, both very dependent on the surrounding environment.

Solar energy (SE) is one of the cleanest energy sources and one of the main solutions in achieving higher energy efficiency. The optimization and increased effectiveness of SE's usage can decrease energy consumption from the conventional sources (e.g., burning fossil fuels) in residential and non-residential buildings and increase sustainable energy since SE represents a highly renewable source of energy. This increased energy efficiency is usually associated with on-site energy production. This technology, also known as active solar systems, is applied by implementing photovoltaic panels (PV) or solar (ST) thermal panels (Kanters & Wall, 2018). Based on the available data, the potential of urban PV systems can cover about 30% of urban electricity requirements in 2050 globally (Medeiros, 2020).

On the other hand, passive solar systems do not produce any electricity but can still contribute to decreasing the energy demand of a building. According to Cillari et al. 2021, passive solar energy systems can be utilized to save almost 20% of a building's energy demand. In comparison to active solar systems, passive solar systems do not require any additional devices. The implementation of passive solar systems is solely done by solar energy gain through building features such as facades, windows, floors and building elements in the near exterior surroundings and can be used for the likes of solar heating or solar cooling (Daneshvar Tarigh et al. 2012).

Understanding the solar potential and prioritizing daylight access carries additional environmental and financial benefits. Proper daylight access is strongly associated with human health and well-being, economic activity, and social interaction (Kanters et al., 2021).

1.2 Problem statement

Regardless of the apparent advantages in the application of solar energy (daylight) in the urban environment, studies (Kanters & Wall, 2018; Lundgren et al. 2018; Kanters et al., 2021) also show that the utilization of the solar potential is still not prioritized in the urban development process. This does not represent good practice since there is an existing standard with regards to daylight in buildings issued by the EU (EN 17037:2018) and adopted by many European countries, including Sweden. Sweden also set challenging goals towards future sustainable cities. These targets were a reaction to a clear pattern of ongoing urban densification in the country (Kanters et

al. 2021). From a Swedish perspective, this insufficient state can be characterized by three main issues. The first one is associated with the ambiguous means to effectively use solar energy in the actual urban planning process. The other problem refers to the lack of meaningful metrics and threshold values to assess the performance indicators related to solar access. The last issue is a result of inappropriate tools and data to perform the solar access analysis in the early stages of a development project (Kanters et al. 2021).

Jakica (2017) compared several tools capable of estimating solar energy and daylight for buildings. Most of the tools can provide complex daylight parameters including a detailed building design with facade and facade fenestration. The shortcoming of these tools is that they are designed for CAD/BIM data and are thus better suited for individual buildings or a small group of buildings. This indicates another issue since both Kanters & Wall, 2018 and Lundgren et al. 2018 emphasize the need to assess the daylight potential at the city/neighborhood level rather than at the individual building level. This requirement is also in accordance with the EU directive (EPBD), where the benefits of a transition from nearly-zero energy buildings to net-zero energy districts are laid out (EU Science Hub, 2019).

GIS solar assessment tools and methods can cover larger areas and perform considerably well in the estimation of rooftop solar (daylight) potential. Several studies on the estimation of solar potential in urban areas using GIS are of note, e.g., Brito et al. (2019); Chow et al. (2014) or Quan et al. (2015). However, neither of them incorporate facades or facade fenestration and thus fall short of presenting an optimal method for the estimation of active and passive solar systems that are dependent on these features (Daneshvar Tarigh et al. 2012). Additionally, the datasets used in the studies mentioned above were 2D data (footprints geometry, DEM), which do not have the capability to store facade fenestration features. Including facades and fenestration features in the estimation of solar (daylight) potential is especially important in regions with a higher latitude, such as Sweden, where the solar altitude is comparably low (the Southern part around 35 degrees and the Northern part around 22 degrees) at equinox. As a result, a significant amount of solar energy (daylight) is directly hitting the facades and facade features or is reflected from the facades of surrounding buildings (Dubois et al., 2019).

1.3 Aim of the study

Given the pertaining limitations identified in the problem statement, the aim of the study is to investigate current data and the availability of tools suitable for the estimation of solar (daylight)

potential on building envelopes, including facade features such as windows. The first part of the thesis addresses the following research questions:

- *Is there open-source software available that can estimate solar access for buildings on a city level?*
- *What kind of geodata is required to estimate solar (daylight) potential for buildings on the window level for a whole neighborhood?*
- *How can required input data be retrieved or created? More specifically, how can window data be obtained on a city / neighborhood level in order to calculate the solar irradiance on the facades / windows?*

If the already existing tools and data are not appropriate for the estimation of solar (daylight) potential on a window level, a new tool will be developed. The project will then have to answer additional research questions:

- *How should the tool be designed and implemented?*
- *How should the tool be evaluated?*

1.4 Disposition

Since the thesis combines information from different scientific fields, chapter two plays an important role in the document. It contains the theoretical framework that provides the most essential concepts in order to understand the methods used as well as the intersections of the related scientific fields in a multidisciplinary context.

Chapter three relays the design and implementation of the daylight estimation tool. The design and implementation stem from findings in the theoretical background.

Chapter four describes the evaluation of the daylight estimation tool. In order to evaluate the tool, a case study in Lund has been carried out.

Chapter five focuses on the discussion with a critical view including limitations of the design and implementation as well as shortcomings based on the evaluation of the tool within the case study.

Finally, chapter six summarizes the thesis in respect to research questions and contains recommendations for future work within this topic.

2 THEORY

This chapter focuses on describing the essential terms from different scientific fields (GIS, physics, environmental science, computer science, architecture) related to the thesis. Concepts such as sustainable urban development, solar energy and 3D city models are introduced. The first two sections provide a general overview about the concept of sustainable development and solar energy and establish the thesis in a wider context. The following section narrows the focus down to explain more of the technical concepts of the thesis related to 3D city models. After that, the entire section discusses the concepts directly associated with the aim of the study - solar potential estimation on the building's facades, including facade windows. The last chapter introduces the concept of free open-source software because it is becoming a very popular method of software development (including scientific resp. academic software) and many GIS software and solar modeling tools are already released under free and open-source licenses.

2.1 Sustainable urban development

2.1.1 Definition of sustainable development and its brief history

"Sustainable development is the development that meets the needs of the present without compromising the ability of future generations to meet their own needs."

By Gro Harlem Brundtland

The quote of the former Norwegian prime minister is widely used as one of the most precise definitions of the term sustainable development (SD). The definition is more than 30 years old, but the idea of sustainable development has been growing for a longer time. According to Basiago (1999), the first concerns related to sustainable growth were formulated by the English economist Thomas Malthus in his work *An Essay on the Principle of Population* published in 1798. By that time, in the middle of the industrial revolution the problem was not urgent, and the side effects of the rapid transition in manufacturing processes and the population growth were mostly ignored by the economists and politicians. The next related work occurred almost two hundred years later in a book called *The Limits to Growth* in 1972. The collective of authors led by Donella Meadows from Massachusetts Institute of Technology described the concept of sustainability as it is known nowadays. However, the term "sustainable development" was introduced in a collaborative work *World Conservation Strategy* issued by the United Nations Environment Programme and the International Union for the Conservation of Nature in 1980 (Basiago, 1999). Since then, sustainable growth has become one of the main interests of different international organizations. The most significant of the actions to raise awareness about sustainable growth was the Brundtland Commission (World Commission on Environment and Development - WCED) Report where the

definition of SD (used as an introductory quote) was formulated. The other very significant event was the 'Earth Summit' which was the 1992 United Nations Conference on Environment and Development in Rio de Janeiro. At this conference diplomats declared a commitment for SD and accepted the action plan which is known as 'Agenda 21' (Wheeler, 1996).

Wheeler (1996) summarized the several definitions of SD. As the author declared, any of them is perfectly suitable. However, all of them oscillate around three main axes/dimensions of sustainability, specifically the social, the environmental and the economic aspect of sustainability. Social sustainability is an approach to ensure the highest quality of life for every individual living in the society. It is tightly connected to the terms like equality, healthy environment, justice, and human rights (Social Sustainability, 2020). Economic sustainability is most difficult to define because there are different opinions of what economic sustainability is. Pure economic point of view can indicate that an economy is sustainable as the economy grows and capital increases albeit with a negative impact on the different aspects of sustainability (social and environmental). A more complex definition of economic growth can be defined as a set of rules and regulations supporting durable economic growth attempting to eliminate the negative impact on the social and environmental aspects (Economic Sustainability, 2020). A brief explanation of the last aspect can be the following - environmental sustainability means living in accordance with the limited natural resources and utilizing them at an appropriate rate as well as a liability to protect the ecosystem's integrity at the global scale (Circular Ecology, 2020). Thus, sustainability can be defined as the balance between all three essential axes. Omitting any of the essential dimensions of sustainability cannot provide sustainable development.

2.1.2 Sustainable development in the urban environment

Zooming in on the urban context the entire idea of SD becomes clearer. Transition of cities and urban areas into being governed, operated and designed in a sustainable way can be considered as one of the biggest challenges of this generation. All the essential axes of SD are unquestionably connected to the urban environment. As mentioned in the introduction, more than 55% of the world's citizens live in cities today, which emphasizes challenges related to social sustainability. Pradhan and Abdullahi (2017, p. 22-23) mentioned a number of those challenges e.g., accessibility to several public facilities and services (education, healthcare, etc.), job opportunities, protection of human health from pollution and natural hazards, fulfillment of essential requirements (housing, food, water), etc.

Economy is another indisputable factor of urban areas since cities generate more than 80% of global GDP (Urban development, 2020). Fast growth of urban areas can increase the size of economy and capital in the cities; however, this does not have to automatically mean a benefit to society. Economic growth that sacrifices non-renewable natural resources as well as human-made resources without preserving at least a part of them for future generations cannot be considered as sustainable. The same can be applied in terms of society. If the economic growth does not reflect on higher salaries, improving infrastructure and accessibility to facilities then criteria for economic sustainability is not met. The idea of economic sustainability can be considered as a 'profit to action' model, where higher monetary capital is invested back into society as e.g., higher salaries, eco-friendly refurbishment, increased technological efficiency, etc. (Koglin, 2009).

Urban areas are also accountable for 70% of global CO_2 emissions which absolutely highlights the need for environmental sustainability. The European Environment Agency in its report about air quality recognizes five main sectors that produce the most emissions. Namely, transport (both road and non-road), energy supply for residential and non-residential and institutional buildings, all types of industries (manufacturing, extractive, heavy and light), agriculture and waste (EEA, 2020). Thus, implementation of environmental sustainability in an urban context should address the need for better utilization of renewable resources, sustainable utilization of those resources that are not renewable, better public transportation system, low-carbon building materials, properly planned urban sprawl and sustainable waste management (Pradhan and Abdullahi, 2017).

2.1.3 Solar energy (daylight) as part of sustainable development in urban areas

In the introduction, a few applications of solar energy (daylight) in the urban environment have been mentioned e.g., active and passive solar systems or impact of daylight access on human's well-being. However, these are only a few examples and exploiting the full potential of solar energy and daylight is the subject of an extensive ongoing research in this topic. International energy agency (IEA) launched the *Solar Heating and Cooling programme* in 1977 as one of the first programmes of IEA. The work within this programme is done by tasks, which usually are a collaborative effort of experts from member countries, sponsors, and EU. According to the official website[1], the main purpose of this programme is to help in increasing the deployment of solar applications by resolving technical and non-technical obstacles through multidisciplinary international research. Another example of an initiative associated with the application of solar energy in urban areas is the POLIS project. POLIS project joins six European cities (Lisbon, Vitoria, Lyon, Paris, Münich and Malmö) and its aim is to establish a more unified planning and

[1] https://www.iea-shc.org/

legislative practice for solar developments based on assessing the current methods in solar urban planning and bring together the main responsible parties in the process as well as to adopt strategic urban planning and municipal policy initiatives to benefit from solar energy potential (Medeiros, 2020).

Medeiros (2020) also came up with a term of "SUNstainability" which refers to a method for achieving sustainable urban development through the intensive utilization of solar energy. The author identified five main components that SUNstainability is composed of. The first dimension is the solar energy generation capacity, which is important to assess if an area or a city is exposed to a sufficient amount of solar radiation for the application of solar energy systems. The other dimension represents urban planning and governance processes and their appropriate instruments and tools (for urban planners and local governments), which would help with the regulation and stimulation of solar energy. The third dimension is related to the environmental aspect of utilization of solar energy and its improvements to environmental conditions directly and indirectly. The fourth dimension discusses the economic aspect of utilization of solar energy and its impacts on the stimulation of the green economy. The last dimension examines the social aspect of utilization of solar energy because daylight can significantly improve the quality of life and health via proper solar access. Even though all the dimensions are important, the most crucial task before implementing is the analysis of solar energy potential of the area of interest, which depends on the amount of available incoming solar radiation (Medeiros, 2020).

2.2 Solar radiation

Solar radiation is the electromagnetic radiation emitted by the sun, the essential source of energy on the Earth. Since this energy comes directly from the sun it contributes no emission to the atmosphere and capturing this energy is considered as one of the cleanest, highly renewable, and environmentally friendly sources of energy. With the help of technological progress, smart energy policies and appropriate urban planning, solar energy will hopefully represent an adequate share of future energy systems (Freitas et al., 2015).

2.2.1 The Sun and solar radiation

The sun is an extremely hot spherical body with the blackbody temperature 5777 K. It can also be considered as an "endless fusion reactor" where hydrogen is being converted into helium (Duffie and Beckman, 2013). This reaction causes a tremendous release of energy and heat. The process of fusion is called proton-proton chain reaction and is taking place in the sun's core. The simplified principle of the fusion is that 2 protons of hydrogen collide to produce deuterium. The other step

is that deuterium hits another hydrogen proton, and a nucleus of helium-3 is formed. The last step is that 2 nucleases of helium-3 collide to create a stable helium-4 nucleus. The mass of the helium-4 is lesser than the mass of hydrogen protons coming into the reaction, so the loss of mass converts to energy (NASA, 2021).

This energy is then transferred to the sun's surface and radiated into space. The solar energy has to travel one astronomical unit, which represents mean earth-sun distance 1.495×10^{11} m and gets to the atmosphere and hits the Earth's surface. The average amount of solar radiation is 1367 W/m^2. This value is also known as the solar constant. In real conditions, the value of solar radiation that eventually reaches the surface is reduced because of absorption and scattering particles in the atmosphere (Duffie and Beckman, 2013).

For the estimation of solar irradiance (solar radiation received per unit area), several components and parameters have to be taken into account. Solar radiation towards an object on the Earth's surface can be split into three main components: beam radiation (also referred to as the direct radiation), diffuse radiation and reflected radiation (Duffie and Beckman, 2013). Beam radiation is a non-scattered radiation received directly from the sun. Diffuse radiation is also a solar radiation received from the sun, but the direction of radiation has shifted from its original path in the atmosphere. Reflected radiation is radiation that has already hit the object on the Earth's surface, but the radiation has not been absorbed but reflected (Duffie and Beckman, 2013). The value of solar irradiance also depends on the geometrical relationship between the object and the sun. The most important geometrical parameters are zenith angle(θ_z), solar azimuth angle(γ_s), solar altitude angle(α_s), surface azimuth angle(γ), hour angle(ω), latitude(ϕ), declination(δ) and slope(β). The Figure 2.1 shows their mutual relationships (Duffie and Beckman, 2013).

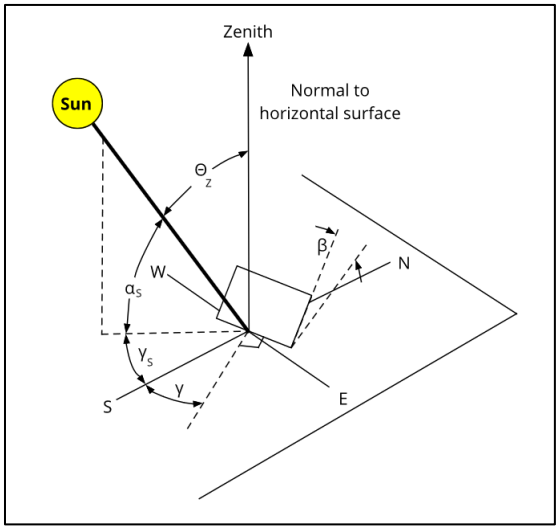


Figure 2.1 *- Solar geometry, (Duffie and Beckman, 2013, p. 13 – adopted by author)*

More detailed description of actual mathematical and geometrical figures can be found in Duffie and Beckman (2013, p. 13). The authors focused on several applications of solar energy, including the estimation of solar irradiance on the tilted and vertical surfaces. These surfaces in the real-world are usually represented by buildings. To run a solar energy simulation, buildings geometry has to be stored in a digital format, suitable for such analysis e.g., a 3D city model.

2.3 3D city model

A 3D city model can be briefly described as a digital representation of urban areas by 3D geospatial data that integrates building models, city furniture, vegetation model, roads as well as land covers, which can serve for various purposes such as visualization, exploration, analysis, and management of urban data (Döllner et al., 2007). However, a critical listener might come up with a question: how can such a complex system as an urban area be modeled? Especially nowadays, when cities represent a complex mixture of all types of relations (spatial, environmental, economic) between all its components (Batty, 2009). Moreover, the important part of city modelling has to also include its dynamics - mutual interactions, flows and synergies as well as long term processes of urban development. One of the answers provided by Kolbe and Gröger (2003) in their paper, where they introduced a scalable spatio-semantic concept of a 3D city model. This study contributed to the development of a data model and XML-based format called CityGML, which became later an international standard by OGC, and it is considered as the most comprehensive standard and exchange data format for 3D city models (Eriksson, 2020).

2.3.1 CityGML

CityGML can be described as an open XML-based multifunctional model that can be used for geospatial data exchange, data storage, database modelling as well as distribution of geographic data. According to the technical specification issued by the Open Geospatial Consortium (OGC), CityGML is an application schema of the GML3 (GML), the extensible international standard for spatial data exchange and encoding. CityGML is thus a recognized encoding standard by both the Open Geospatial Consortium (OGC) and the ISO TC211 (Technical Committee for Geographic information/Geomatics of the International Organization for Standardization) (Gröger et al., 2012).

The special focus during the development of the CityGML model was on its interoperability and compatibility in order to provide as wide usage as possible. The model design is based on the international standard ISO 19107, which defines spatial schemas for storing geographic information as well geometry and topological relationships. They also implemented the Abstract Model of the OpenGIS Consortium, where concepts of spatial features were introduced, into the

model. The definition of features can be formulated as follows: “Features are abstractions of real-world objects resp. phenomena having spatial and non-spatial properties” (Kolbe and Gröger, 2003).

The other fundamental objective of CityGML was to define the main feature classes, attributes and relations corresponding to their real properties. The main principle was based on a feature class classification and decomposition of city objects for both domains - the spatial and semantic. The decomposition can be understood as a vertical hierarchical structure where the upper layer represents a city as a whole and with going down in the hierarchy more detailed city/building objects are defined (Stadler and Kolbe, 2007).

The concept of vertical hierarchical structure can be also replaced by the term called Level of Detail (LoD), which allows a city model to be used in different range of scales. Biljecki (2013) in his dissertation described LoD as a “concept which defines the degree of abstraction of real-world objects, primarily designated to use an optimum of details of modeled objects to the user’s need, and computational and economical aspect”. Besides the computational and economical aspect, another important reasons to scale a city model are data availability and its application domain. A city model of higher level of degree is usually required for visualization purposes, while for urban analysis and urban simulations high graphic details can be omitted, but there is a need for detailed semantic information (Stadler and Kolbe, 2007). There are 5 levels of detail defined. LoD0 is a building footprint where building height can be added as an attribute (2.5D). LoD1 represents the simplest representation of a building, which is a simple rectangular block. The LoD2 adds the shape of roof to the building level, LoD3 also includes the positioning of the facade windows with more detailed building features (attics, overhangs, etc.), LoD4 incorporates the modelling of the indoor space (Nouvel et al. 2013).

The geometry and topology of the model is expressed by 0-d (node), 1-d (edge), 2-d (surface), and 3-d(volume) primitives. The model stores only coordinates of the nodes (the lowest degree primitive), because any higher degree primitives consist only of primitives of lower degree. Thus, the geometry of the model (except of 0-d primitive) is defined implicitly, but topological relationships have to be defined explicitly, e.g., edges have to have defined by a start node and an end node, outer rings and inner rings are used to bound the surfaces and b-rep (boundary representation schema) is used to model the volumetric objects (Kolbe and Gröger, 2003).

The semantic domain of the CityGML model consists of features, such as buildings, city furniture, transportation objects, water bodies and vegetation which, in fact, represent real-world entities.

The model description of features also incorporates attributes, relations, and aggregation hierarchies between features (Gröger et al., 2012). All the classes in the model are derived from the basic class 'Feature,' which is used to describe spatial objects and their aggregations in ISO 19109 and GML3 (Stadler and Kolbe, 2007). The International Standard ISO 19109 establishes rules for creation and documentation of application schemas, also including principles for the definition of features (ISO 19109).

Another important concept of the CityGML data model is its spatio-semantic coherence. According to Stadler and Kolbe (2007, p. 3), "coherence in the geospatial context describes consistent relationships of spatial and semantic entities." These connections are made by associations, which can only be formed if structural similarity between the spatial and semantic domain exists (Stadler and Kolbe, 2007). In other words, the spatio-semantic coherence assures that a user working with a 3D model knows what kind of feature the geometry data represents or in the other way, what is the spatial extent and geometry of the features stored in a 3D city model.

The current version of CityGML is 2.0 but the development of the newer version CityGML 3.0 is almost finished. The conceptual model of CityGML 3.0 began being reviewed in the current year (Kutzner et al., 2020). The new CityGML version will bring several changes and the most dominant idea was to make the model more accessible for more users in different application fields. The most important updates are the revision of the core schema, revision of LoD concept, model's new ability to represent time-dependent properties, city models versioning support or a sharp distinction between conceptual model and data encoding specification. New official data encoding specification will likely also include a relational database schema and CityJSON-based representation (Kutzner et al., 2020).

2.3.2 CityJSON format

CityJSON is a JSON encoding for a subset of the data model in CityGML 2.0, as such it represents a simplified alternative to CityGML encoding. CityJSON has been developed by the 3D geoinformation research group at TU Delft and represents another OGC standard. Ledoux et al. (2019) described CityJSON as a JSON-based exchange format for CityGML data model, which implements the majority of the CityGML data model. One of the biggest differences between CityGML and CityJSON is removing a deep hierarchy of the CityGML data model. The main reason behind developing a new exchange format was to offer an easier way for using 3D city models in practice. JSON is the most popular exchange format on the web, and it is also a widely preferred exchange format among developers. The main drawbacks of CityGML data format were

identified as difficult parsing and information extraction, big file sizes, many possibilities on how to store geometry or extensive use of *XLinks* which are not always properly supported. CityJSON offers more simplicity for programmers including design based on simple data types and data structures natively supported in almost all programming languages without the need for external libraries.

2.3.3 Extension of 3D city model

Even though the CityGML model is a fairly complex and universal data model, it sometimes does not include all necessary information - new object types or object properties necessary for a specific application. The creators of CityGML took into account situations like this and developed important components of CityGML that enables extending the initial scope of the model (Gröger et al., 2012). There are two possible ways to extend a CityGML model.

The first approach refers to generic objects and attributes. This approach can be used to include attributes or features, which are not originally modeled. Extension is made by creating a *GenericCityObject* which obtains *_genericAttribute* (Wate and Saran, 2015).

The second approach is extending the CityGML model via an ADE. According to Biljecki et al. (2018, p.1), "Application Domain Extensions (ADE) is a mechanism for enriching the data model with new feature classes and attributes, while preserving the semantic structure of CityGML". ADE provides two possible ways to extend the data model. One of them defines new application-specific object types as sub-classes of the existing classes in the CityGML. The sub-class is allowed to inherit all the properties and associations of its super classes. The other way extends existing objects by additional attributes. The concept can be explained as an attribute substitution, where application-specific attributes are attached or "hooked" to a feature class by defining an *_GenericApplicationPropertyOf<Featuretypename>*, where the tag *<Featuretypename>* represents the name of the CityGML class that is extended (Wate and Saran, 2015).

The main difference between these two methods is that the ADE method defines new features and their properties and attributes. The extension is defined with an XSD (XML schema definition file), which can be defined directly or firstly modeled with UML (Unified Modelling Language) and subsequently generated from the UML diagram. The generated schema is then imported into the CityGML schema. The generic objects and attributes are defined directly in the existing CityGML schema, so there is no need to model and generate new schemas (Biljecki et al., 2018).

Since CityJSON format is built upon the CityGML data model, the possibility of extension is also similar to a certain extent. However, CityJSON is a little bit less flexible while extending the core schema (e.g., it does not support extension through ADE). CityJSON can be extended in three possible ways. First of them adds new comprehensive attributes to existing City Objects. The second way adds several properties to the document root and the third way creates new city objects including a definition for geometry (CityJSON Specification, 2019).

2.3.4 BIM - Building information model

Building information models (BIMs) represents another way to store detailed building information into one 3D model. According to Autodesk official website[2] the BIM is "*the holistic process of creating and managing information for a built asset, which integrates structured, multi-disciplinary data to produce a digital representation of an asset across its lifecycle, from planning and design to construction and operations*." The multi-disciplinary data can come from different experts at different stages of a project e.g., architects can add the data regarding the design of the building, building engineers can contribute with energy performance data or the contractors with the data regarding actual construction of a project (Lorek, 2021).

Thus, BIM models do not directly represent a city model but can be used as a useful source of relevant data for other city models e.g., in the CityGML format. The need for integration of geodata (e.g., CityGML) and BIM data has been also recognized by *Open Geospatial Consortium (OGC)* and *builduingSMART International (bSI)* that released a joint report that focused on integration between two platforms. The report summarized different barriers that block a better integration between those standards (Mercer, 2020).

The example, how the implementation of the "GeoBIM" approach can look like is described in section 2.4.4 and is related to extraction of window information from BIM to city models stored in CityGML.

[2] https://www.autodesk.com/solutions/bim

2.4 Theoretical framework for the estimation of solar (daylight) potential of facade windows

2.4.1 General purpose for solar modeling

In the end of section 2.1.3, the importance of estimating the solar potential of a specific location before implementing any solar systems is mentioned. The main purpose for that is the high cost associated with installation of active solar systems. Also, another important purpose relates to an actual structure of a building and its spot in the existing city pattern. The orientation of external building facades is an essential factor which determines the degree of exposure to the sun and thus, has a big impact on the efficiency of both active and passive solar design of a building (Medeiros, 2020).

2.4.2 Solar modeling in GIS

One way to quantify the solar potential of a specific location is utilization of solar maps. A solar map can be described as a GIS application, which gives information regarding annual solar irradiance values at a specific area or building envelope (roofs or/and facades). Furthermore, solar maps can also serve to accelerate the process of utilizations of solar energy in urban development projects (Kanters et al. 2014).

Choi et al. (2019) conducted a study related to GIS-based solar radiation modeling. The authors conducted a large review of 92 articles related to the topic and found three key applications of GIS-based solar modeling as described below.

The first application is solar radiation mapping. Solar radiation mapping is important for identification of areas with sufficient solar radiation exposure as well as for predicting the solar radiation distribution in the spatial and temporal domain. The method to estimate solar irradiance values depends on the type of data available for a specific location. The solar radiation values can be obtained by a) direct measurement of solar radiation at a station, b) measurement of other weather parameters, which solar radiation can be calculated from. Standard interpolation methods can be used, if there are enough measured points in the area of interest. For the case that solar radiation needs to be estimated in the areas without measured data, solar models can be used. The input for the solar model is the measured weather data (e.g., temperature, humidity, precipitation, etc.), and also specific geographical and terrain information (e.g., latitude, longitude, aspect, slope).

The second application of GIS-based solar modelling is the site evaluation for solar energy systems. The multi-criteria decision analysis is used for the site evaluation including several different factors (financial, social, environmental, risks) that might affect the site selection besides solar radiation.

The last application is the assessment of solar potential for implementation of solar systems (both active and passive), which plays the key role in energetic and economic efficiency estimation of active solar systems. GIS-based methods are used for two main purposes; a) to estimate physical potential of buildings/building features (by calculating the total solar irradiance values) and b) geographic potential of buildings/building features (by providing information about suitable areas for installation of solar systems). This assessment is a combination of solar radiation modeling (the available tools are described in the following section) in urban areas with spatial analysis tools as feature extractions, different types of overlay operations and visualization of data (Choi et al., 2019).

2.4.3 Free urban solar energy simulation tools

Clearly, an urban building stock is dense and tall, and thus it becomes a natural obstacle for incoming solar radiation, which only emphasizes the importance of modeling local solar energy potential and assessing the effectiveness of the solar systems before they have been installed and deployed (Freitas et al., 2015). There are several tools to model solar energy in the urban context, and three open-source modeling tools are described below.

2.4.3.1 UMEP - The Urban Multi-Scale Environmental Predictor (SEBE module)

The UMEP simulation modelling tool is designed to use 2D input data and produce 3D output. In this fashion, it is considered as a 2.5-dimensional model (Lindberg et al., 2015). The main input for the calculation consists of a Digital Surface Model (DSM) and meteorological data. Other two mandatory raster files - wall heights raster and wall aspect raster can be derived once, when the DSM is created. Therefore, all the required data can be obtained from the model's pre-processor (Lindberg et al., 2015).

In order to perform an accurate solar energy simulation in urban areas, the generation of shadow patterns is an essential task to obtain correct results. The algorithm for estimation of shadow casting in this modelling tool uses the calculation of shadow volumes and this algorithm was developed by Prof. Paul Richens (Lindberg et al., 2015). The main principle is based on the method when the DSM raster is being moved according to the known position of the sun (azimuth) and for each iteration the height is reduced according to the sun elevation angle - sun altitude. A

fragment of the total shadow estimation is calculated per each iteration. In the end of the loop, the whole shadow volume is aggregated and stored as a new raster layer. This layer is later subtracted from the original DSM. A similar algorithm is used for the building walls, but it is extended for an edge detection filter. The algorithm can identify building walls this way (Lindberg et al., 2015). UMEP is also an open-source project available on GitHub[3].

2.4.3.2 Solar3DCity

Solar3DCity is also a free and open source[4] modelling tool developed by the 3D Geoinformation group at the Delft University of Technology. The tool estimates the solar potential on the rooftops of buildings stored in a 3D city model. The model uses a 3D city model in CityGML format as the input. A simplified workflow of the tool can be summarized in four major steps. Firstly, the tool reads the CityGML dataset and identifies the rooftops for each building. The next step is to estimate the parameters required for calculation of solar irradiation. These parameters are the azimuth, tilt and area of rooftops. The third step is actual solar energy estimation using python library *solpy* and EPW weather data (EnergyPlus Weather format). The last step is writing the result back to a CityGML dataset. The tool uses a city model of LOD3 as the input. The limitation of the tool is that it does not perform a shadow estimation (Biljecki et al., 2015).

2.4.3.3 Solar3D

Solar3D represents another open-source[5] tool that can be used for estimation of solar irradiance on 3D surfaces. This tool interactively calculates the solar irradiance in a virtual environment composed of a 3D city model, DEM, and feature layers. To measure the solar irradiance at a point on a virtual 3D surface over a given time interval, the tool uses a cube map to decide whether the point is shaded at each interval for the whole time. Another task performed in this step is the estimation of the surface slope and aspect from the surface normal. The next step is to create a cube map-based panoramic view of the 3D scene at the point. The panoramic view contains sky and non-sky pixels encoded in different color values. After that, the tool decides whether the point is shaded at each time interval based on the pixel intersection in the corresponding cube map created in the previous step.

This tool extends the GRASS GIS *r.sun* solar radiation model by providing additional parameters required for 3D modelling (aspect, slope, shading) as long as 2D solar raster maps usually require only a given day, atmospheric conditions, surface, and latitude. The authors of the tool consider as the main limitation of the tool that it can calculate the solar irradiance only in the points and the

[3] https://github.com/UMEP-dev/UMEP
[4] https://github.com/tudelft3d/Solar3Dcity
[5] https://github.com/jian9695/Solar3D

tool is not able to perform the calculation over large areas. Another drawback of the tool can be that this method is quite quick only for short time intervals (less than an hour) (Liang et al., 2020).

2.4.4 Facade windows in solar modeling

Passive solar systems and their impact on decreased energy demand of a building were mentioned in the introduction. This section provides a brief background for a passive solar building design, and the role of facade windows in it. In the context of the thesis, this section explains why it is important to perform solar modeling on a facade window level and why it is valuable information for architects and urban planners when designing buildings or neighborhoods.

Passive solar design is a method to effectively use solar energy gained through building components (facades, windows, floors). The systems are designed to accumulate, store and redistribute energy, which is transformed into heat. This heat can be used to heat up a building during the winter or system components can reject the heat, and thus cool down a building during the summer. From a scientific point of view, passive solar systems integrate features and methods known from thermodynamics (particularly heat transfer), climatology or fluid mechanics (Sadeghsaberi et al. 2013).

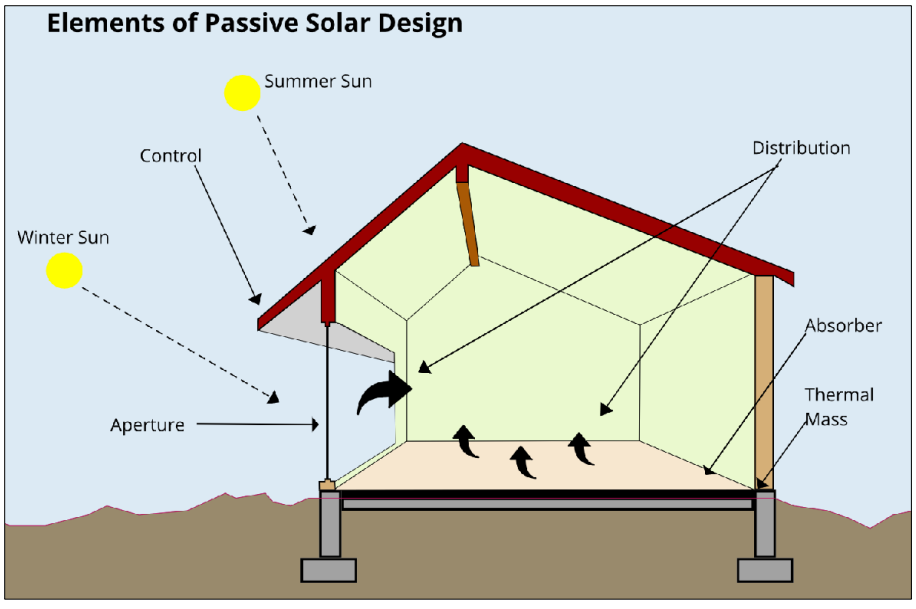


Figure 2.3 *- Passive solar design (Fosdick, 2016) – adopted by author*

A basic passive solar design combines five elements (Figure 2.3). The first one is an aperture (collector)- usually a glazed area located on a facade or roof, which serves as an entrance of solar

energy (daylight) into the building. The second element is an absorber - hard, darkened surface (walls, floors) which absorbs the solar energy as a heat. This heat is then transferred into the third element - the thermal mass, serving as a storage component. The fourth element is represented by a distribution system. The purpose of this element is to redistribute the gained heat stored in the storage component. It is done by natural convection, conduction, or radiation of a material. The last element is a control system, e.g., a roof overhang that creates a shade and controls the system from overheating especially during the summer months (Fosdick, 2016).

Apart from those mandatory elements, there are a few more variables that need to be considered when designing a passive solar system. One of them is site selection - it is very important to secure an unobstructed path for sunlight to obtain a high efficiency of the passive solar system. Another important factor is proper window orientation, where windows are usually oriented to the south to maximize the solar heat gain (U.S. Department of Energy, 2021). Solar heat gain can be described as the increase of thermal energy when an object absorbs incident solar radiation. The amount of solar heat gain is a function of total incident irradiance and solar heat gain coefficient, which represents the ratio between incident solar irradiance and transmitted solar irradiance per surface, e.g., a window (Tamm et al., 2020).

2.4.5 Collecting window information to city models

The BIM model can be a useful source of window information and accurate geometry. Huang et al. (2020) proposed a method of integrating BIM models with CityGML models on a query level and thus, be able to extract information from BIM to be used in CityGML. One of the methods on how to achieve this goal is to use an integrated query approach on knowledge graphs. Knowledge graphs can be described as a multi-relational graph where entities represent nodes and edges serve as relations. Every single edge has a form of triple, and elements featuring the triple are head entity, relation and tail entity. As a result of such an understandable data model, data may be arranged into connected graph structures, allowing multi-source and heterogeneous data to be interconnected and integrated. The other important step was to create ontologies which served as a base for data integration and exchange. The authors first created knowledge graphs from initial datasets, which were interlinked with both the ontological and instance level. Authors then used SPARQL, which is the most common query language being used for knowledge graphs. The SPARQL language has a capability to query across the interlinked data sources and thus retrieve integrated data in the knowledge graphs.

However, this solution assumes that a user has access to BIM models as well as a deep understanding of these models as well as having the ability to perform advanced queries. More user-friendly approach to obtain and collect window information was proposed by Fan et al. 2021 as a web-based interactive platform VGI3D. The design of the platform has been focused on low-cost data requirements, speed of 3D building modelling, high-quality of the modeling result as well as the simplicity and ease of use for either expert or non-expert users. The platform effectively reconstructs a 3D building model in a CityGML format from images, e.g., street view images or VGI (volunteered geographic informa-tion) provided by users.

The workflow of the platform in Fan et al. (2021) can be divided into a few key steps. The first step is the user interaction where the user provides the images and draws the facade boundaries and two additional parameters associated with the building, namely facade orientation and a roof type. The second step is facade features extraction using convolutional neural networks (CNN) for object detection. The detection network has been already pre-trained (900 images) and thus is able to perform semantic segmentation of facade features including facades, roofs, doors or windows. The following step is the location correction and layout correction to fix the perspective distortion from street level images and misalignment of features. Once the facades and facade features have the correct location, they can be modeled into a 3D city model. The platform uses JavaScript library Three.js to build 3D shapes from 2D polygons. The output city model is in CityGML format, and it is available to download from the platform's web location (Fan et al. 2021).

2.5 Free open-source software

The tools described in section 2.4.3 have one common feature that is worth mentioning. All of them are associated with the term "free open-source software (FOSS)". This section briefly explains this term and also provides an overview of applications of free open-source software within GIS.

2.5.1 Free open-source software definition

Many computer users are familiar with the term FOSS. However, it originates from two different premises. While both of the terms usually refer to the same category of the software, the philosophy behind the terms is different. The first term *free software* is based on the ethical principles, which stand for the maximal possible freedom to a user. In this context, free software does not mean "gratis" from the monetary point of view. A software can be qualified as free, when the principles of four essential freedoms of free software are integrated within the software (GNU Project, 2021). On the other hand, the term *open-source* points out more the practical benefits of using free software and its philosophy can be considered as more pragmatic and business oriented. The term

was brought up by *The Open-Source Initiative* (OSI), which has an objective to engage more users and developers to contribute and support software development processes (Open-Source Initiative, 2018). Nevertheless, both of the terms regardless of the philosophy behind allow users to "*run, copy, distribute, study, change and improve the software*". These operations can be also considered as a brief summary of four essential freedoms of free software (GNU Project, 2021).

2.5.2 Free open-source software development

FOSS development differs from conventional software development which is used by most of the software engineers working in companies. Furthermore, there is not even a standardized or proposed project management method on how to run a FOSS project. The development strategy is different for every project but there are visible the features of the principles mentioned in the section above. For example, the code is publicly available, all stages of development are openly visible, but the most significant difference is the leading role of the community of users and developers on how to steer the project (Scacchi et al., 2006).

The infrastructure for a FOSS development project should include several components. The essential is the public source code repository. This usually means a code located on a website where users and contributors can clone it in order to run it or modify it. Another important component is the project documentation. Since there are several contributors, it is important to maintain detailed and accurate documentation during the project. Bug database represents the next part of FOSS infrastructure. Keeping track of bugs is important for continuous improvement of quality of the software and thus, also increasing popularity of the project. The last component is a communication platform enabling social interaction within the community among developers and users. This platform should provide an open and safe place for the discussions, brainstorming or bug reporting (Gabriel & Goldman, 2005).

Nowadays, there are several service providers who can offer all the mentioned infrastructure components within one single workspace. The most popular providers are GitHub[6], GitLab[7] or SourceForge[8]. These platforms often integrate several functionalities as hosting the source code, providing the full version control using Git including the source code management as well as project management tools or team administration tools.

[6] https://github.com/
[7] https://about.gitlab.com/
[8] https://sourceforge.net/

2.5.3 Free open-source software in GIS

The development of free and open-source software is growing very rapidly. According to the 2020 Open-Source Security and Risk Analysis (OSSRA) report, 70% of the entire global code is compromised by open-source software (e.g., using an open-source component). In comparison, in 2015 it was only 36% (TechRepublic, 2020). This growth is affecting all the industries including GIS.

OSGeo, which stands for Open-Source Geospatial Foundation, is the biggest and most important foundation which supports and popularizes collaborative development within the GIS industry. According to the official OSGeo website main (OSGeo, 2019), goals of the foundation are:

- to provide resources (financial, organizational, or legal) for ongoing projects within the foundation
- promote the use of open-source GIS software
- emphasize interoperability among the foundation projects as well as with the open and community standards
- assure innovative and high-tech approach within the foundation projects.

The project portfolio of the foundation is very broad. It covers projects related to content management systems, metadata catalogs, desktop applications, geospatial libraries, spatial databases, or web mapping.

One of the most famous and successful projects is the desktop application QGIS. QGIS is a professional GIS software enabling a user to create, edit or analyze geodata in both raster and vector formats. QGIS is also well integrated with web map services and has multiple integration with other GIS open-source software (e.g., GRASS GIS, PostGIS). Furthermore, QGIS is highly extensible through plugins developed by the users. Plugins can be written either in C++ or Python (QGIS, 2021).

Web mapping projects contribute either as free map servers (MapServer, GeoServer) or as a library for easy visualization of dynamic maps on the website, known as OpenLayers. Moreover, the popular spatial database PostGIS is an OSGeo foundation project too (OSGeo, 2019).

Gradually, there are more free and open-source software related to 3D geospatial data including 3D city models. One of the biggest initiatives to develop free and open-source software related to

3D city models is taken by the 3D geoinformation research group[9] at TU Delft in the Netherlands. This group has already developed several tools for validation, manipulation, or visualization of 3D city models. This research group also run the CityJSON project[10] on GitHub which refers directly to development of tools related to this encoding format of 3D city models. A few tools developed by this group have been also used within this thesis project and they are explained below.

[9] https://github.com/tudelft3d
[10] https://github.com/cityjson

3 DESIGN AND IMPLEMENTATION OF "DAYLIGHT ESTIMATION TOOL"

3.1 Background and tool selection motivation

Given the aim of the study, the first part of which was to investigate available data and tools capable of estimating solar (daylight) potential on the building envelopes, including facade features such as windows, a study of current software and data availability was carried out.

In order to assess the feasibility of the current software it is important to define the requirements that need to be met. The requirements in this particular case were the ability to perform urban solar energy simulation using 3D or 2.5D data, another requirement was the capability to estimate the solar potential on a window level, and the last criterion was that the software should be distributed as open-source. Based on these requirements, several software have been reviewed. Three of them (*UMEP, Solar3DCity, Solar3D*) are described in section 2.4.3. However, none of those software fully met the requirements, especially the criteria related to the capability of estimating solar potential on a window level.

Another task was to analyze the availability of geodata. Since the aim of the study was to perform the estimation of solar potential on the window level, there was a need for a free 3D city model that included window information. The platform VGI3D (Fan et al. 2021), described in section 2.4.5, represents a suitable option for the requirements and scope of this thesis project. The tool is able to reconstruct the 3D city model of LoD3 based on the free and available street-level images, which is in accordance with requirement for free and publicly accessible geodata.

Because of the limitations of current software, which do not provide a sufficient solution for estimation of solar potential on window level, a new tool needs to be designed and developed. Referring to the available software and data, a good solution is to utilize the solar energy simulation tool - UMEP, a 3D city model obtained from VGI3D platform and to develop a "Daylight Estimation Tool" that contains a method to obtain metric parameters that can be useful for urban planners, decision makers and everyone else involved in the urban development process.

The UMEP software was selected due its capabilities to calculate solar irradiance on the rooftops as well as facades and thus allow the developed tool to calculate corresponding values for window features. The other reasons were the detailed documentation and the least demanding requirements for the input data. Moreover, the software is deployed as a QGIS plug-in with a user interface, which makes it relatively easy to use.

3.2 "Daylight Estimation Tool" requirements

User stories is a common method for defining needs used in agile software development. The method uses a simple template e.g. *"As a <role>, I want to <goal>, in order to <benefit>"* to express desired functionality or use case (Lucassen et al., 2016). A few user stories have been defined based on the needs in respect to the research questions.

User stories for this project have been defined as following:

- *As a user, I want to use only free software and publicly accessible geodata.*
- *As a user, I want the tool to be easy to manipulate.*
- *As a user, I want the tool to estimate solar potential parameters on the building envelope, especially on the facade windows.*
- *As a user, I want the tool to store calculated parameters to the 3D city models as attributes.*
- *As a user, I want a tool that has output which is easy to interpret and visualize.*
- *As a user, I want to use a free and open-source project, which I can modify if I need to.*
- *As a developer, I want to use free and open-source software components.*
- *As a developer, I want to use a programming language that has a great number of libraries, including libraries supporting geoprocessing and geometric features.*

In order to fulfill the last requirement, "Daylight Estimation Tool" is developed by using the Python programming language. Python is a high-level, general-purpose, and object-oriented programming language released in 1991 by Guido van Rossum. It is a free open-source project, and its source code is available under the GNU General Public License (GPL). Python is popular for its quite easy syntax, a big community, many libraries and modules and cross platform ability (W3school- Python Introduction).

3.3 "Daylight Estimation Tool" design

The selected software (UMEP and VGI3D) and the user stories establish the basic framework for the architecture and functionality of the daylight estimation tool. The first step is an analysis of outputs of the selected tools and drawing a high-level architecture of the tool. The insight gained in the analysis can represent a means to identify limitations or shortcomings within UMEP and VGI3D in respect to requirements formulated above. It is important to emphasize that limitations and shortcomings do not represent a wrong methodology or inaccurate result of UMEP. They represent only limitations for the method and solution applied in the context of this thesis project.

The precise identification of limitations in combination with requirements will be a crucial factor in order to design the tool correctly.

3.3.1 System architecture and functional analysis

The output of the UMEP energy simulation consists of two files. One of them is a raster image representing the incoming irradiance on the rooftops within the study area. The other file is a text file containing the irradiance values for each pixel which represents a building facade. Since the file contains several values for each pixel in the vertical direction, the data can be extracted and used in a 3D analysis. The output of VGI3D is a 3D City Model in CityGML that can be converted to CityJSON format. This dataset serves as the main source of building and building features geometry.

Based on the requirements, selected tools, and their outputs a high-level architecture (system level) of the method and its desired result can be sketched (Figure 3.1).

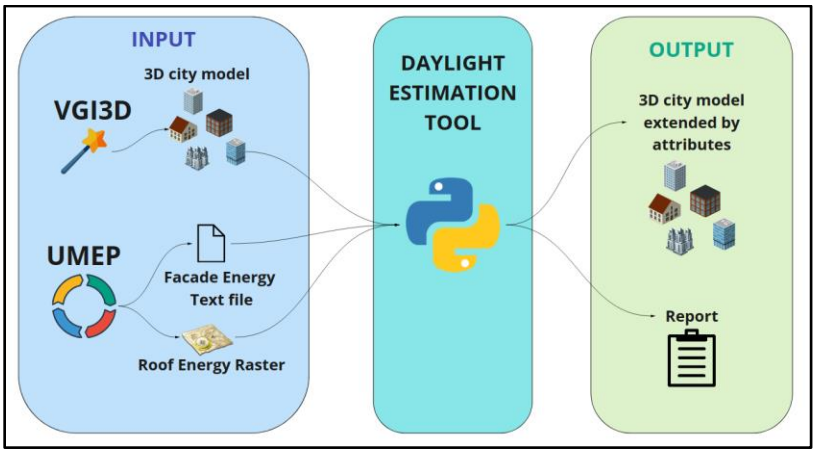


Figure 3.1 *- Solution architecture diagram – figure represents overview of tools used in order to create input data for DET as well as the desired output at high level (each process is explain in detail later)*

Given the requirements, UMEP output has an insufficient format that needs to be further processed within the custom tool. The first issue that needs to be solved is the inclusion of facade windows into the energy estimation. The UMEP simulation does not include windows information. Another shortcoming is related to the output files and their formats. The roof raster is easily interpreted, but the output facade energy text file is very difficult to interpret without further processing. The solution of this shortcoming is related to the requirement about an easily interpretable output with the visualization capabilities. Hence, one of the main challenges is to find a way to identify which of the pixel values in the raster belong to a particular feature in the 3D City model. Another challenge is to extract irradiance values only for window features. This requires a custom approach

in order to analyze raster data (output text file representing the irradiance values on the facades) with the data stored in a 3D model (window features).

3.3.2 Solution architecture

Following the functionality analysis, the solution has to cover several operations which the daylight estimation tool is expected to carry out during the process. Given the availability of current libraries some of the operations can be done by using already existing libraries and some of the functionalities need to be developed entirely from scratch. Thus, the final solution represents a custom application with the integration of a few already existing libraries.

The existing libraries can provide the support for basic operations such as file reading and writing (.txt, .json) or basic mathematical operations. In this particular case, these operations are managed by standard Python libraries *json* and *math*. Since Python is a popular programming language and has a big community of users and contributors, libraries capable of reading, creating, modifying, processing vector features as well as reading, creating raster datasets and performing raster operations are available. For this purpose, Python libraries *Shapely*[11] (creation, manipulation, geo-processing of vector features) and *Rasterio*[12] (creation, manipulation, analysis of raster datasets) are used. Furthermore, the Python library *Numpy*[13] is used for creation and manipulation of arrays and matrices.

For the developed operations and functions, the bulk of the work was done by using the object-oriented programming paradigm. According to Pedamkar (2021) this programming approach has many well-known benefits such as:

- easier code maintenance - using different classes creates the code more modular and thus also easier to modify. Changes in methods of one class does not affect the remaining classes (as long as they are not dependent on each other), respectively changes and updates do not require large-scale changes.
- data redundancy - object oriented approach enables the inheritance of classes so same data do not need to be stored twice
- extensibility of properties/methods - classes can be extended anytime in the process when there is a need for a new functionality or a property. This feature also represents a good practice for future versions of developing software.

[11] https://shapely.readthedocs.io/en/stable/manual.html
[12] https://rasterio.readthedocs.io/en/latest/
[13] https://numpy.org/

In the context of this project, there is one extra reason why it is an optimal solution to use object-oriented programming. It is because of the input data structure - one of the inputs is a 3D city model, which can be simply decomposed to several smaller components (e.g., facades, roof, windows, doors, roads, etc.). Using an object-oriented approach can utilize this decomposition and create corresponding object classes with relevant properties based on semantic surfaces within a 3D city model. This way it is also easier and more comprehensive to identify mutual relationships between classes because they can be described as the abstraction of real-world features. Figure 3.2 shows the designed object classes, which is the daylight estimation tool composed of.

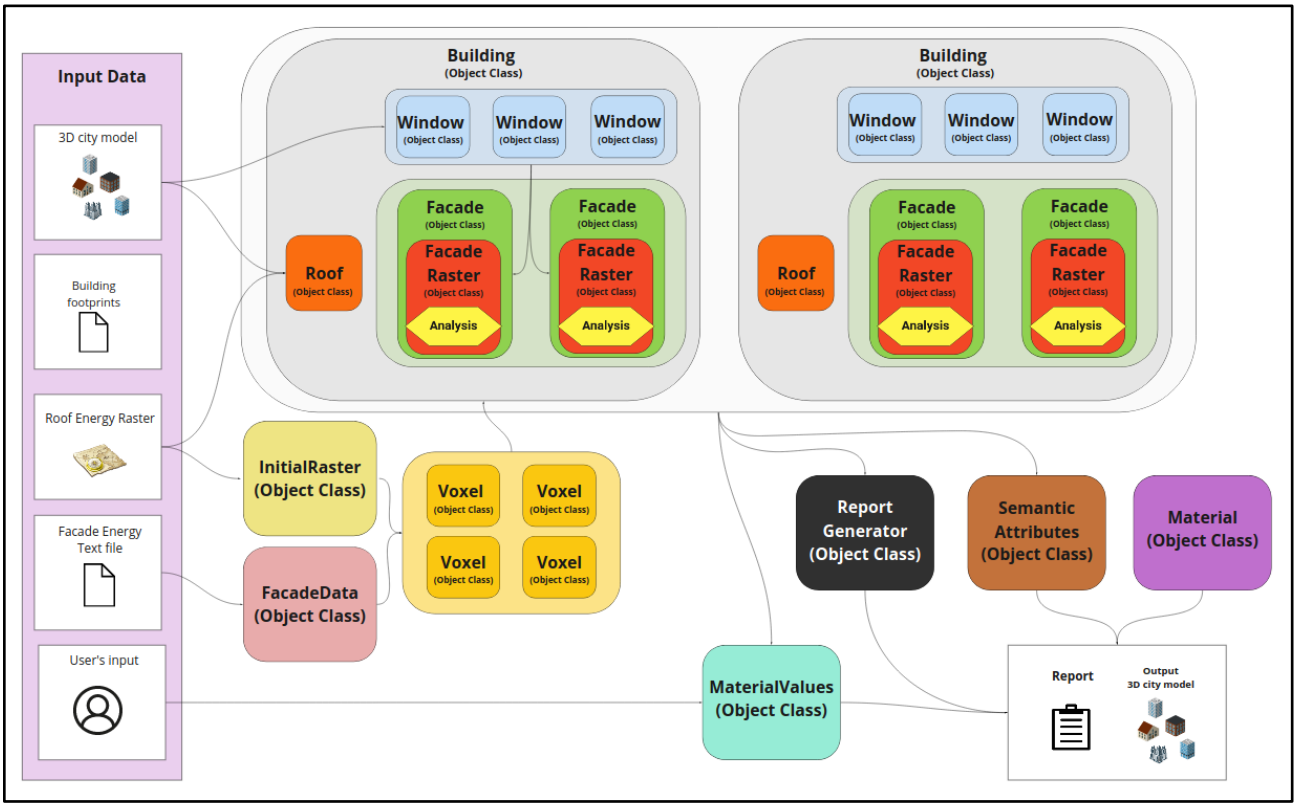


Figure 3.2 *- Solution architecture diagram – this figure shows the object classes used in the solution; the diagram shows the entire process since input data until output creation at component (classes) level*

Briefly, the diagram describes a solution which creates an object for each semantic surface in a 3D city model which is exposed to sun (facades, roofs, windows). These classes are incorporated into an object of the *"Building"* class. Also, there are several classes which do not represent a semantic surface, but they are important for the solution since these object classes contain data used in the analysis. These classes are usually associated with processing the input data, resp. performing an

analysis. This type of object classes is represented by *"InitialRaster", "FacadeData", "Voxel"* and *"FacadeRaster[14]"* classes. The last type of object classes covers the output creation.

3.4 "Daylight Estimation Tool" implementation

The previous section specified the design of the tool at the system and component level. This entire section zooms into the code level. Figure 3.2 shows several object classes which the tool is composed of, and how they are connected to each other. First of all, there is a brief explanation of object classes (the detailed technical specification of object classes used in the main process, their attributes and their methods are provided in Appendix A) for better understanding of the process. Remaining subsections describe the main process and output creation.

3.4.1 A brief object classes description

The purpose of the *"InitialRaster"* object class is to establish the spatial extent for further processing of the facade energy text file and also serves as an input for the estimation of the solar irradiance values on the rooftops. The spatial characteristics are extracted from the raster image representing the incoming irradiance on the rooftops and they represent "a bridge" between the image coordinate system (indices of rows and columns) and the CRS (x and y coordinates).

The *"FacadeData"* object class reads, converts, and stores the data from the facade energy text file. Since each line of the file contains row and column indices for each raster cell representing a building facade, followed by values for irradiance on the building facade, these data are converted and stored into a list of dictionaries where each dictionary has a simple structure: `{row_id (int), column_id (int), irradiance_values [(float)]}`.

The data stored in the *"FacadeData"* object is the input for initialization of another object class - the *"Voxel"* object class. The objects of this class are very significant since they gather complex information about each voxel, including membership to a particular building or facade and also irradiance values. The main purpose of this object class is to provide correct data for a facade raster.

"Building" object class is another essential class entering the process. This object class collects the geometry of city objects stored in the 3D city model. As can be seen in Figure 3.2, a *"Building"* object incorporates all other object classes related to semantic surfaces as its attributes. Thus, this

[14] although, *"FacadeRaster"* is stored within a *"Building"* object it does not represents a semantic building surface

object class can be used as a collection of information about each city object as well as each surface that has been analyzed during the process. *"Building"* object class is also associated with several important methods that are crucial for the result of analysis. For instance, membership identification of *"Voxel"* objects to buildings/facades or creation of *"Facade"* objects are performed by the methods of *"Building"* objects. A list of "*Building"* objects serve as an input for output creation.

As mentioned above, *"Facade"* object class is initialized by one of the *"Building"* object methods. The benefit of this class is very similar to the previous object class. A *"Facade"* object mostly collects all data associated with a particular facade - *"Voxel"* objects, the geometry and dimensions of a facade, etc. A loop which iterates through a certain list is present in almost every method during the process. Hence, having all the necessary information (e.g., list of *"Voxel"* objects, or a facade geometry) extracted for a single facade reduces the number of objects or list elements iterated through, and the code execution can be faster.

"FacadeRaster" object class is a subclass of the "*Facade"* object class. This object class provides methods and functions, which allow the creation of a raster image from data stored inside *"Voxel"* objects. The raster image representing solar irradiance on a building facade is the most important output of the object class methods. This facade irradiance raster serves as one of the inputs for the estimation of solar irradiance on windows.

"Window" object class is designed to store a geometry of facade windows retrieved from the input 3D city model. Based on the geometry and a facade membership the values of solar irradiance are extracted by performing a raster overlay operation.

The last object class associated with the *"Building"* object class is the *"Roof"* object class. *"Roof"* objects are used to extract solar irradiance values on the rooftops of the building objects. The extraction is also made by a raster operation, when a mask representing a geometry of a roof is created and this mask is used in combination with the *"InitialRaster"* object class, which is the source of irradiance data.

Four remaining object classes are associated with the output creation and their brief description is included within section 3.4.3.

3.4.2 Main process

Figure 3.3 shows the process diagram of the tool. The process starts with importing configuration and libraries. Next step is reading input data which is used during initialization of the object classes. First initialized class is the *"InitialRaster"*, followed by the *"FacadeData"* object class. Subsequently, *"Voxel"* object class is created based on the data from the *"FacadeData"* object. Additionally, parameters from the *"InitialRaster"* object are used to define x and y coordinates for each *"Voxel"* object, this process is covered by the *.defineCoordinates* function.

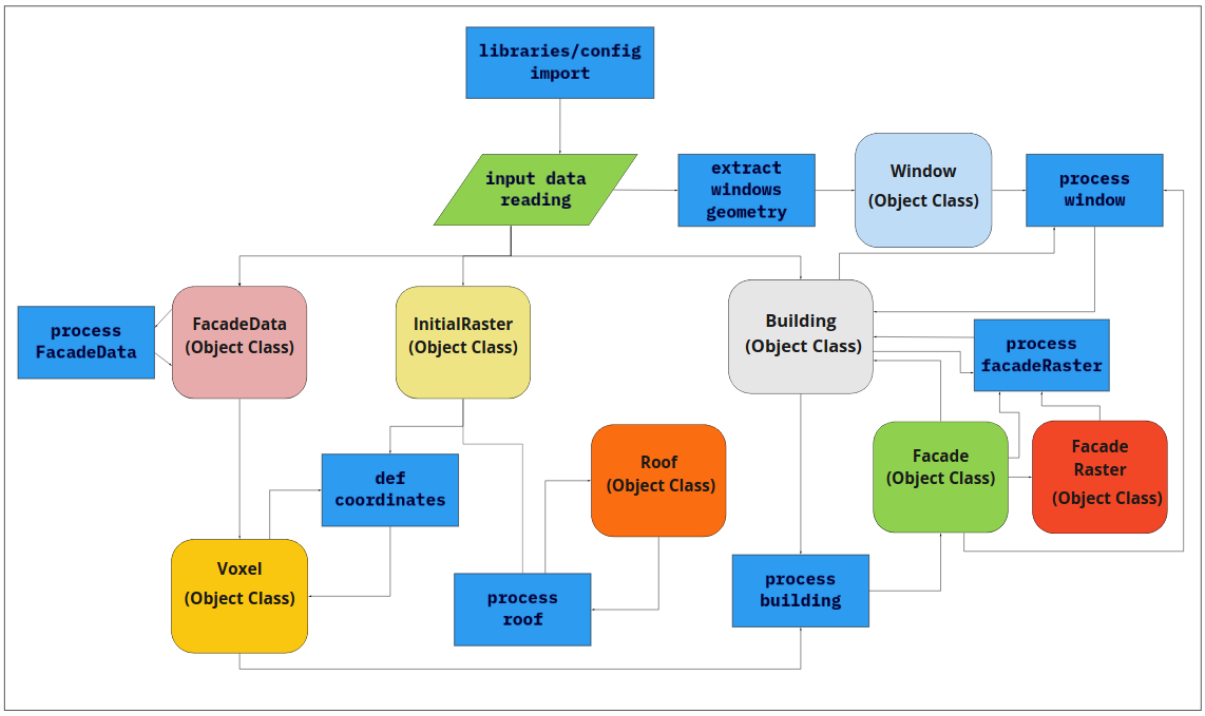


Figure 3.3 *- Main process – a process flow diagram that clarifies the data flow between classes as well as their mutual relationships in terms of which class serves as input for another, etc.*

Remaining input data - a 3D city model together with building footprints are used to initialize the *"Building"* object class. Exact arguments to create a "*Building*" object can be found in Appendix A. "Once the "*Building*" object is created, the function *process_building()* is called which incorporates methods belonging to *"Building"* objects. All methods used in the process_building*()* function are described in detail in Appendix A. One of the class methods run in the function creates a *"Facade"* object for each facade of the building. All *"Facade"* objects are stored into a list, which is a *"Building"* object attribute. Arguments for the function are the *"Building"* object itself and *"Voxel"* objects, which obtain corresponding building and facade index based on the proximity analysis. List of *"Voxel"* objects belonging to a particular building/facade is then stored as another *"Building"/"Facade"* object attribute.

When all *"Facade"* objects are created and *"Voxel"* objects are identified, the *"FacadeRaster"* object class can be initialized. As a subclass of a *"Facade"* object inherits all attributes, which are necessary for creation of a raster representing the irradiance values of a facade, namely the facade dimension and irradiance data stored in voxel objects. The actual facade raster is created while the function *process_facadeRaster()* runs, which uses the same logic (a sequence of object class methods) as the *process_building()* function and it is explained in Appendix A. Created facade raster is stored as an attribute of *"Facade"* object.
Next step is dedicated to the extraction and creation of *"Window"* objects. Firstly, the geometry of all window features related to a particular building are extracted from the 3D city model using the function *extract_window_geometry()*. Secondly, a *"Window"* object is created for each window. The last step is running *process_window(),* which identifies the facade that a *"Window"* object is corresponding to and uses an overlay method to extract the actual irradiance values for a window from the facade raster. These values are stored as attributes of a *"Window"* object. All window objects belonging to a particular building are stored into a list, which is an attribute of *"Building"* object.

The last step is the extraction of irradiance values for the roof surface. After the *"Roof"* object class initialization, the function *process_roof()* is called. This function extracts the geometry of the roof surface from the input 3D city model and uses the *"InitialRaster"* object to extract the irradiance data on the rooftop. Roof objects are also stored as an attribute of *"Building"* objects. All *"Building"* objects are appended to a list which is returned after the entire process has run. Another list of all irradiance values for windows is created and both lists serve as input data for output creation purposes.

3.4.3 Output creation

When the processing module of the tool is done, the output can be created. The output of the tool consists of the updated 3D city model and report in JSON format. The entire process of output creation is shown in Figure 3.4.

Before the particular methods of output creation are described, the motivation behind the selected solution is provided. The goal is to create a framework, which is flexible enough to provide customized results for each user's purposes. There was an assumption made that this tool will be used most likely by urban planners to quickly evaluate the solar (daylight) potential of windows in the early stage of the urban planning process. For this purpose a good means to visualize the

result would be very appreciated. However, visualizing quantitative data from 3D city models is not a trivial task. For example, a 3D city model stored in CityJSON format would need to have rules on how to visualize each solar irradiance value that has been assigned to window features. This would mean a creation of a new material object for each of the rules, and could result in a big extension of the 3D city model. For this reason, four categories of suitability are defined - not suitable, fairly suitable, well suitable and highly suitable. Each category gets a material object containing the rules for visualization (different color). The thresholds that determine a suitability category for each window are given by the user during the output creation process. A hint on how to select reasonable values for each threshold is provided in a form of histogram as well as basic statistics of all irradiance values estimated for window features.

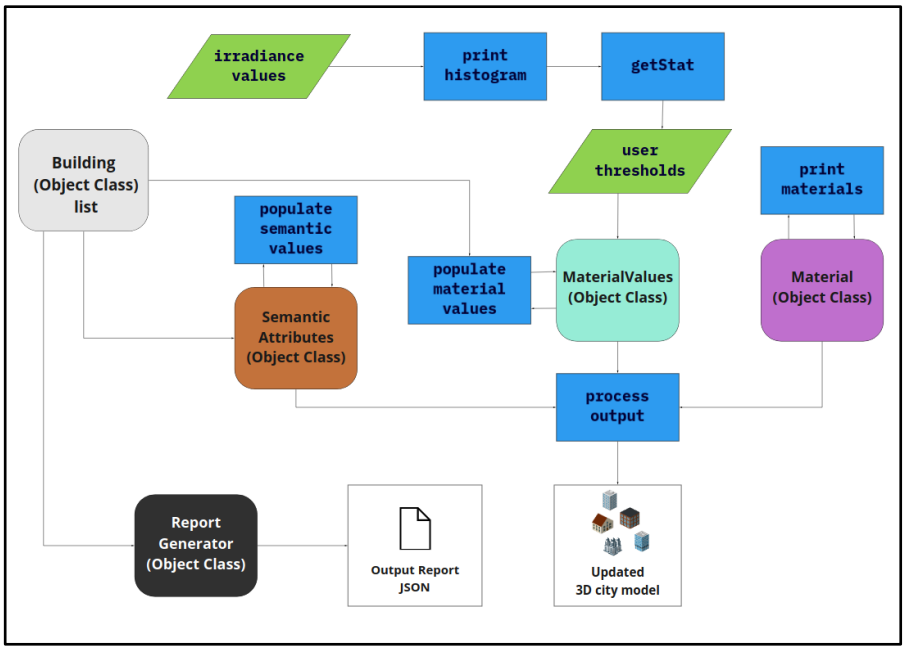


Figure 3.4 *- Output creation process – similarly as Figure 3.3, this figure shows the data flow and classes relationships during the output creation process*

The actual update of a 3D city model stored in the CityJSON format consists of two tasks. Both tasks are strictly performed with respect to the CityJSON specification[15]. The first task is adding complex attributes to the semantic surfaces by extending *surfaces* objects within the *semantics* property. New attributes related to estimated solar irradiance are added for each window surface. This is done by the "*SemanticAttributes"* object class. This object class uses the attribute *surfaces* from the "*Building*" object class to be initialized. The method is based on the identification of surface type from the list of values retrieved from the *semantics* property of the 3D city model. To

[15] https://www.cityjson.org/specs/1.0.1/

clarify this, Figure 3.5 illustrates a possible way of storing surfaces in a 3D model in CityJSON format.

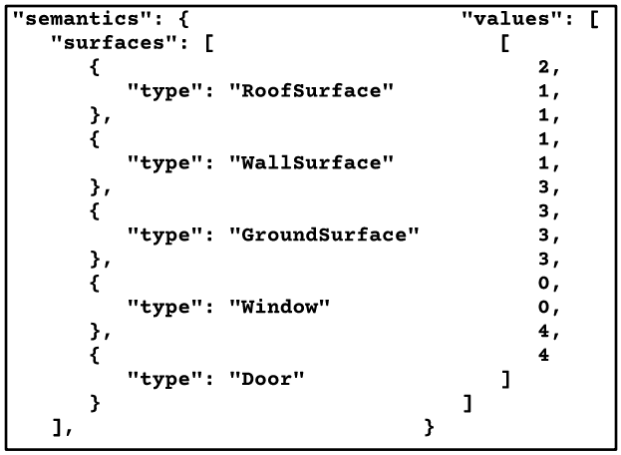


Figure 3.5 *- Surface information stored in CityJSON format*

For instance, if a retrieved value from the "values" list is equal to 1, it means that the examined surface represents a building wall. If the method detects a value indicating a window feature, the object in the "surfaces" list is extended by new attributes related to estimated irradiance values of a window object. The same applies for roof surfaces. Surfaces of other types contain only information about their type without any other attributes. Extended surfaces objects are appended to the 3D city model by the *.processOutput()* function.

The second task is adding new material objects into the *appearance* property and creating an association to these material objects within the *semantics* property (each semantic surface has an assigned index value of a material object defined in *appearance* property). This is done by creating two object classes. One of them is "*Material"* object class which creates valid material objects based on the predefined values - name of a material object (e.g., *irradiance1*) and color codes which are defined in RGB percentage color code. These inputs are wrapped together in a valid structure by the *.printMaterials()* method and appended to the *appearance* property of the 3D city model by the *.processOutput()* function. The *"MaterialValues"* object class is applied to correctly link a material object to a corresponding semantic surface. The method of this object class is called *.populateMaterialValues()* and uses the thresholds defined by the user to sort out windows features obtained from the "*Building"* object class into four categories and assign them the correct material object index value. These values are then turned into a valid structure and appended to the 3D city model by the *.processOutput()* function.

Besides the updated 3D city model, the tool also generates a report where an overview of the irradiance values is per each surface which is possible to estimate. The extracted values irradiance on rooftops, facades and windows are stored in JSON format in a case that a user would need to do any further processing of the file for any purposes, e.g., statistics.

3.5 Distribution of the "Daylight Estimation Tool"

The tool is distributed as a free and open-source software issued under the MIT free software license[16]. This license is categorized as a permissive, which grants any user who uses the software quite flexible opportunities and a wide variety of tasks to perform with the software. Hereby, the license grants anyone to "*use, copy, modify, merge, distribute, publish, sublicense, and sell copies of the software*". Only one restriction associated with this license is a preservation of copyright notice on all future copies of the software (Berman, 2021).

The source code is available on GitHub[17]. GitHub gathers the biggest community of possible contributors to the project because the entire further development and improvements of the tool is meant to be conducted as an open-source project. Several ideas for further development are discussed in chapter five.

[16] https://opensource.org/licenses/MIT

[17] https://github.com/petnez9/daylight_estimation_tool

4 EVALUATION OF "DAYLIGHT ESTIMATION TOOL"

4.1 Background

The following step after the development of the "Daylight Estimation Tool" (DET) was to evaluate the tool in a case study. The tool is evaluated with test data, which have been created using the tools described in section two. Specifically, the 3D city model, which was created as an output of VGI3D tool, and the output of UMEP energy simulation tool. The purpose of the evaluation was not to perform a complex daylight potential simulation for a certain period, nor to assess the performance of an urban energy simulation tool. The purpose of this evaluation was to evaluate the capability of the tool to combine several inputs from heterogeneous sources and formats, correctly match these input data, and extract the values for the features of interest, which were windows features in this case, and finally to create a valid output. The other purpose of the evaluation was to identify the current limitations of the daylight estimation tool, which should be improved in the further development.

4.2 Study area

The study area is located in the neighborhood of Spoletorp in the City of Lund, Skåne, Sweden. The area is in the city center in the built-up area close to the Central Station. Approximate coordinates of the area are 55.709°N and 13.186°E.

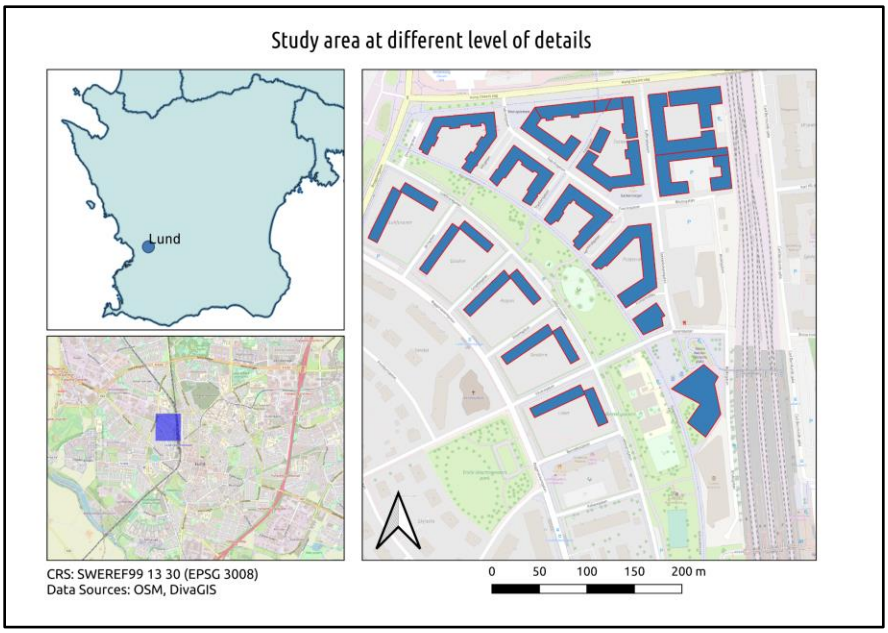


Figure 4.1 *- Detail on the Study Area in Lund*

4.3 Initial datasets and software requirements

In order to perform energy simulations and run the DET, the following datasets are used:

- DEM - digital elevation model of study area downloaded from Lantmäteriet (product name GSD-Elevation data, Grid 2+) with 2 m spatial resolution. The horizontal reference system is SWEREF 99 TM and the vertical reference system is RH 2000. Positional accuracy of the dataset is 0.1m in heights and 0.3m in plane. The dataset is available under the FUK (Forskning, utbildning och kulturverksamhet) License. The DEM raster was resampled to 1m resolution.
- 3D city model of the study area (Figure 4.2) - a model of LOD3 obtained as the result of VGI3D platform[18]. The tool was run by Hongchao Fan and Gefei Kong, NTNU, Norway, who are also the creators of this tool (Fan et al., 2021).

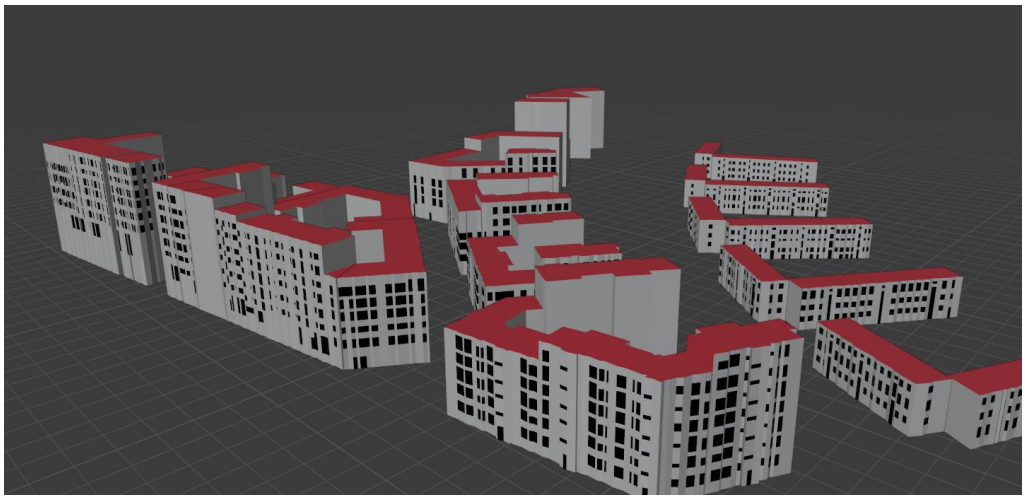


***Figure 4.2** -3D city model of study area*

If a user wants to run the DET on a computer, these software components need to be installed:

- QGIS with UMEP plug-in
- Python version 3 and higher
- Python libraries (*shapely, geopandas, rasterio, matplotlib, numpy)*

4.4 Data preparation

The data preparation does not directly concern the design and implementation of the DET but it is important in order to correct usage of the tool. Thus, the processes creating the required input data are covered in the Appendix B. Briefly explained, input data creation contains two main steps. First step is regarding the manipulation of the initial dataset. The second step is performing the

[18] this dataset has been provided to the author in CityJSON format by the creators of VGI3D platform

UMEP energy simulation. Successful performance of these operations ensures the correct input data for the DET.

The following list of input data served as an input for the DET:

- 3D city model - initial data
- Shape file containing footprints and building heights (used also within the UMEP energy simulation) - created by "pre-processor" toolbox (see Appendix B)
- raster image representing the solar irradiance on the rooftops (output of UMEP energy simulation, see Appendix B)
- text file representing the solar irradiance on the building facades (output of UMEP energy simulation, see Appendix B)

4.5 Running the tool

The first step prior to the DET code execution, is setting up the configuration file. The file is located in the root directory of the tool and is very comprehensive. The configuration consists of providing correct paths to all input data (see the section above) and to output data which are the updated 3D city model and output report.

Once the configuration is set, the user navigates to the directory, where the code is stored, and enters the following command to launch the code execution:

```
python3 det.py
```

The command line interface (CLI) informs the user about the progress, particularly the user gets information which building is processed and the processing time for each building. When all buildings are processed, the user's interaction is required in order to set thresholds for windows solar suitability categories. These values can be decided based on the histogram (shown automatically by the tool) and basic statistical values. The user can enter the values directly in the command line when the prompt is raised. After the user interaction the tool stores the output files based on provided paths in the configuration file.

The figure of different CLI cases during the code execution is shown on the Figure 3.4 below. The top left part illustrates the CLI right after the tool has been launched. The top right corner shows the histogram which is rendered by the *Matplotlib*[19] library. The bottom part shows the end of the process, including the user's interaction part.

[19] https://matplotlib.org/

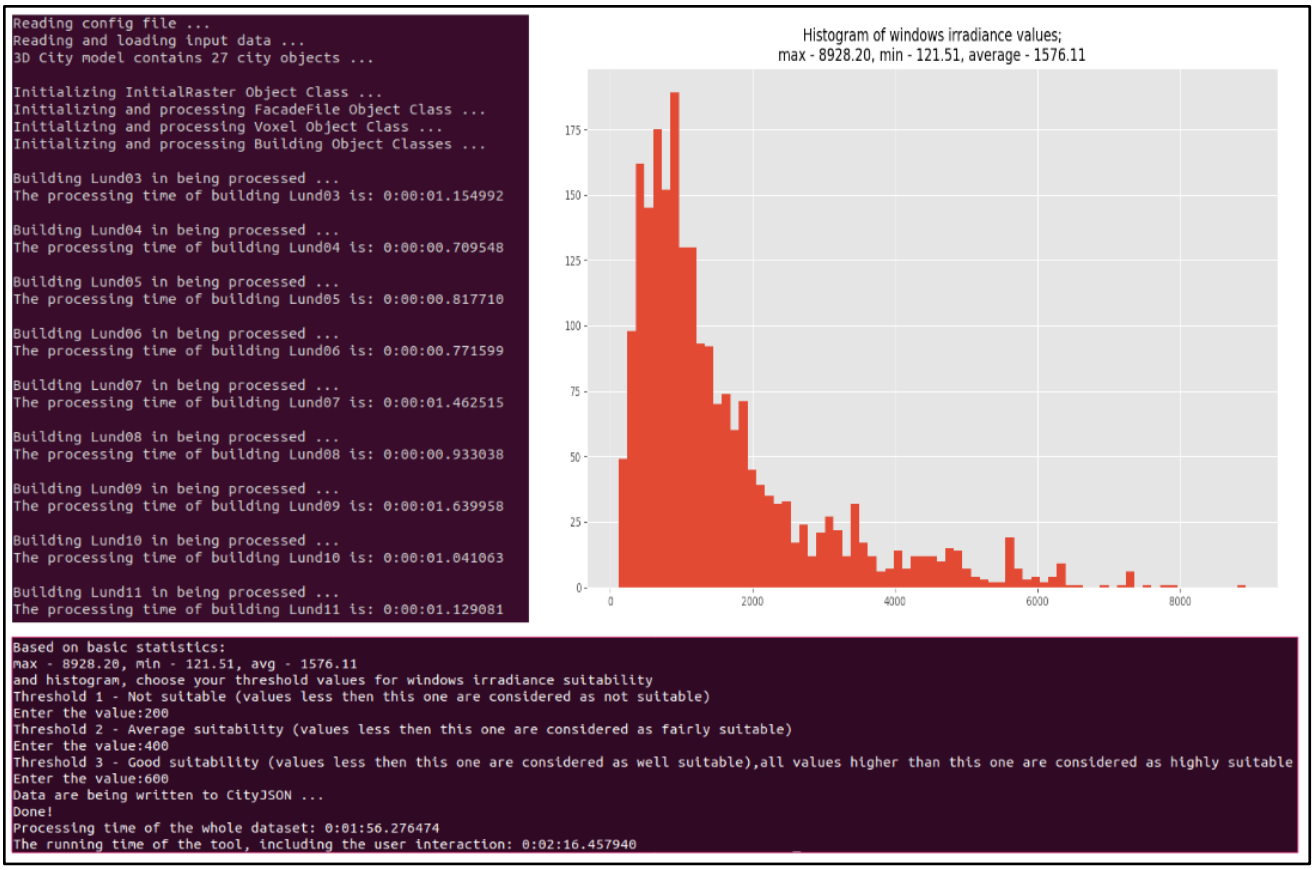


Figure 4.3 *- CLI interface during the code execution – top-left corner shows the CLI interface during processing; top-right corner shows the histogram of irradiance values; bottom of the figure illustrates the output creation part of the process including user interaction*

4.6 Output and output validation

As mentioned before, the output consists of two files - the updated 3D city model and output report. The updated 3D city model contains material objects in the appearance property that can be used for visualization purposes and their values to match the corresponding surface as well as solar irradiance attributes for window features. The report represents an overview of all city objects divided into several categories based on their semantics.

Because several modifications have been made to the 3D city model stored in CityJSON format, it is a good practice to check the validity of the new file. For this purpose, the free open-source command line tool *cjio*[20], developed by the 3D geoinformation research group at TU Delft (authors and developers of the CityJSON encoding standard), have been used. The *cjio* tool provides numerous functions and one of them is the validation against the official specification of

[20] https://github.com/cityjson/cjio

CityJSON. The tool validated the output 3D city model stored in CityJSON format, and the result can be seen on the Figure 4.4.

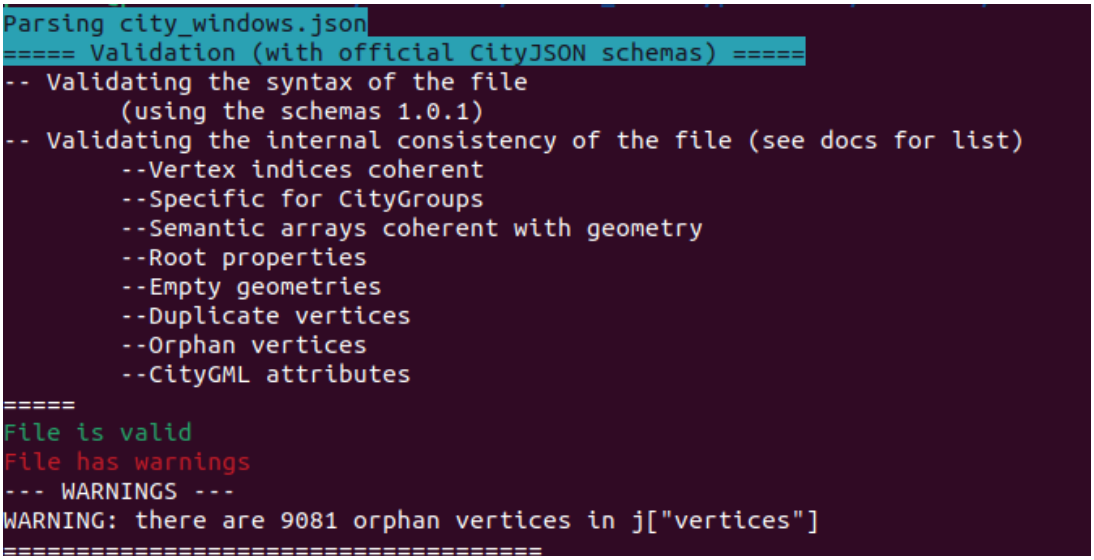


Figure 4.4 *- Validation output provided by command line tool cjio*

The warning flag in the validation output indicates the orphan vertices. This is caused by the fact that the original dataset covered a bigger area, and the number of city objects has been reduced. However, the list of vertices has remained unchanged.

5 DISCUSSION

This section discusses the insights that have been gained during the evaluation of the DET and also compares the evaluation of the tool with user stories that served as defined requirements for the design and implementation.

The tool is able to perform an estimation of solar irradiance on building envelopes, including facade windows. The tool is also fairly easy to manipulate, as a single command executes the entire code. The results are stored in the valid output 3D city model as parameters. From a developer's perspective, only free and open-source components are used during the development. The solution is coded in Python which is ranked as one of the most popular programming languages with a wide range of available libraries.

Moreover, the comprehensive design of the tool based on semantic objects from a 3D city model enables users to easily extend the capability of the tool in the future. Thus, the irradiance on different building features (doors, balconies) can be performed if a dataset contains such information. The extension would require the creation of new object classes, which would work on similar principles as the *"Window"* object class, to provide an example.

5.1 Limitations

A number of limitations have been observed in the case study. They can be classified into four main categories:

- limitations caused by inconsistencies in the input data;
- limitations caused by insufficient design and implementation of the DET;
- limitations due to limited testing datasets;
- limitations caused by the unavailability of free visualization tools for the created output.

The first category mostly refers to errors in the 3D city model. According to Biljecki et al. (2016), errors in city models are more common than usual and only a small share of the models (mostly simple LoD1 models) are without errors. However, it can still cause problems when working with a city model. The geometry of input data has been validated with the val3Dity[18] online validation tool made by the 3D geoinformation research group at TU Delft. The result of the validation showed that only 4 from 27 city objects were valid. The overview of errors in the dataset is shown on Figure 5.1.

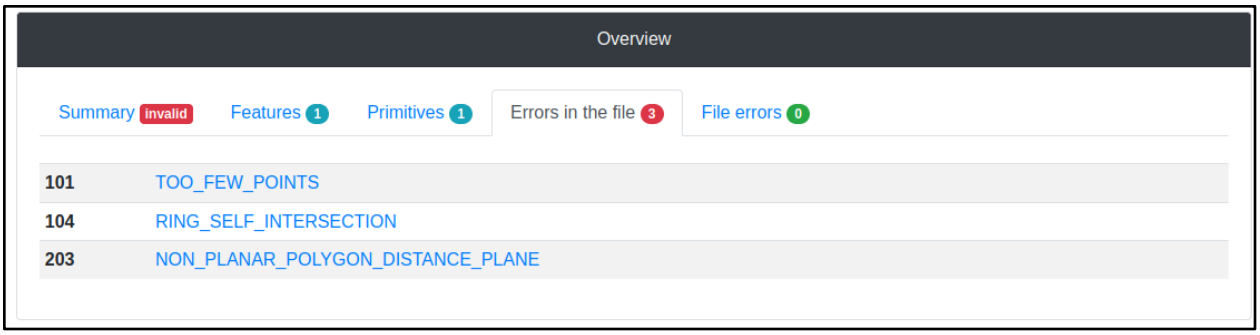


Figure 5.1 *- Errors in the 3D city model provided by val3Dity online validation tool*

A few errors have been handled during the process. However, when it comes to a big file such as the CityJSON, it is nearly impossible to handle every possible source of errors.

Another limitation of the input data was its big fragmentation. The facades containing window features are not created as simple rectangles but are made of many thin strips with the width of a single window. For this purpose, their footprints were simplified (Figure 5.2). These simplified footprints were more suitable for energy simulation purposes as well as for the tool performance. For instance, if the original footprints were used, the city object *Lund03* (a rectangular building) would have 130 facades without much semblance to reality.

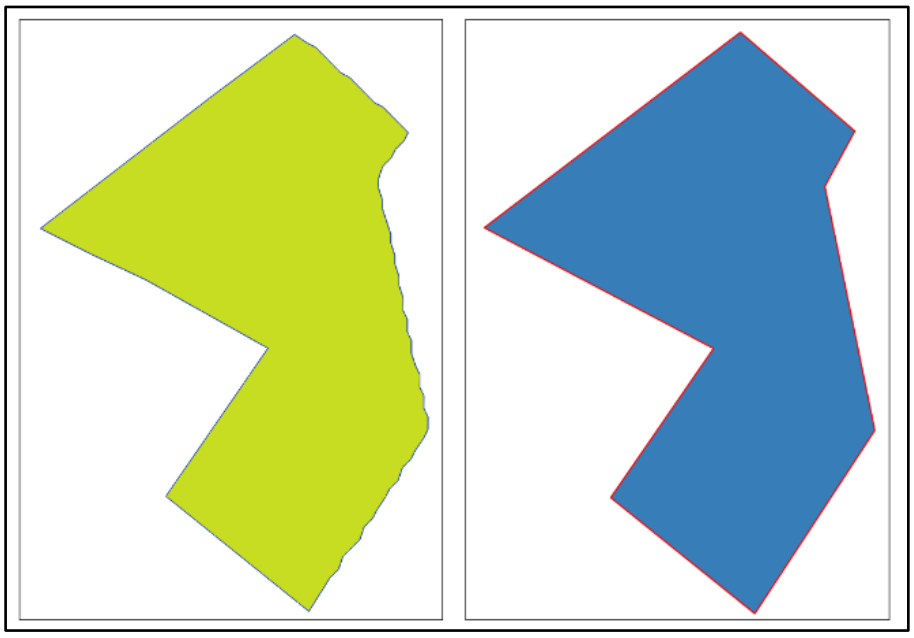


Figure 5.2 *- Fragmented footprints (left) vs simplified footprints (right)*

The second category of limitations relates to the design and implementation of the "Daylight Estimation Tool". The method of assigning corresponding *"Voxel"* objects can result in incorrect

membership for a particular building or facade when two facades are very close to each other. Another potential issue caused by the proximity analysis stems from the eventuality when a certain voxel or window feature does not fall into the buffer area of the facade. This can lead to a loss of information.

Another approach that can be considered as a limitation is the method of estimating solar irradiance values to window features and facades, respectively. The method using the raster overlay technique was applied. However, rasterizing vector data (the positional error arises by default) inherits this error and the estimated value is affected by the error. The possible solution for this case could be a further resampling of the facade raster and using a finer grid for rasterizing the window features.

The last issue regarding design and implementation of the is the user interface. So far, the tool does not have any graphical user interface, and is launched from the command line or from an IDE. Deployment as a web application with a user-friendly graphical interface could help to attract more users and possibly contributors to engage with and help improve this free and open-source project. Another possibility of the tool's deployment, which would also solve the problem with the graphical user interface, is its inclusion with the UMEP plugin in QGIS. UMEP is one of the dependencies since the tool extracts irradiance values from the output created by the UMEP plugin. Therefore, inclusion with UMEP would be a user-friendly solution and both tasks (energy simulation and the estimation of solar irradiance on window features) would be possible to launch from the same graphical user interface.

The third category is caused by the lack of availability of 3D city models including window features. As a consequence, the tool was evaluated against a single dataset. More testing datasets would most likely discover a few more errors and methods to handle them would improve the quality of the tool.

The most significant shortcoming of this thesis project is the last category, or the lack of appropriate free visualization tools. There are several ways to render a city model in the CityJSON format, including some free and publicly available tools and proprietary software. The first group is represented by the online CityJSON viewer "*CityJSON Ninja*[19]" or free and open-source software for 3D graphics called *Blender* in combination with the plug-in "*Up3date*[20]". Both projects are created by the 3D geoinformation research group at TU Delft. Blender creates default materials based on the semantic surfaces and ignores the pre-defined rules for materials appearance. CityJSON Ninja also lacks support for reading the material objects. According to their

GitHub project, this functionality is known and is on the roadmap for the tool's further development. Nevertheless, a new plug-in in QGIS or Blender able to read and render custom defined material objects in the appearance property would be an elegant solution to solve this limitation.

5.2 Utilization of the tool

Limitations notwithstanding, the tool is still able to perform the solar (daylight) estimation, which can be helpful for urban planners who need to obtain metric values of solar potential (or, in this case, solar irradiance) on facade features, e.g., windows in an early phase of a project. The availability of a tool of this kind can be useful for the assessment of passive solar systems efficiency, site selection, finding an optimal orientation of a building or the alignment of window features on a facade.

Moreover, free tools can increase the prioritization of solar energy applications to a greater extent within urban planning, which can result in more sustainable, energy-efficient, environmentally friendly urban areas and ultimately provide a higher standard of living with a healthier environment for everyone.

6 CONCLUSIONS

Current GIS tools and methods can be sufficient for the estimation of solar potential for buildings on a city level. There are also a number of open-source tools that can be freely used. However, they are usually limited to a few applications, such as creating solar maps, site selection and solar potential on rooftops or facades. Neither of these tools are able to perform an estimation of solar potential on facade features, e.g., windows.

Good data is available for the estimation of solar potential on a city or building level but zooming in on the level of building features can cause a number of problems in finding sufficient data.

To fill the gap created by non-existing tools that would be able to estimate solar potential on a window level, the “Daylight estimation tool” was designed and implemented within this thesis project. The tool has been designed with respect to the semantic concept of the 3D city model, outputs of solar energy simulation tool (UMEP) and existing Python libraries that are capable of geoprocessing operations.

The tool has been evaluated against test data. The test data consisted of the UMEP simulation outputs, and the 3D city model created by the open online platform VGI3D. The 3D city model contained 27 buildings within the built-up area in Lund, Sweden. The code execution on test data was successful and resulted in the extraction and estimation of total incident irradiance on window features. Limitations discovered during the evaluation can be categorized into four categories - input data inconsistency, design and implementation shortcomings, limited testing datasets and lack of proper visualization tools.

The tool can be utilized within the early stages of the urban development process. The ability to estimate solar parameters can be beneficial for several urban planning operations, including the assessment of passive solar systems efficiency, site selection, finding an optimal orientation of a building or the alignment of window features on a facade.

Additionally, ideas about the further development of the tool have been discussed. As a free and open-source project available on GitHub, the tool can benefit from a big community of possible contributors.

REFERENCES

Abdullahi, S., & Pradhan, B. (2017). Sustainable Urban Development. In Spatial Modeling and Assessment of Urban Form (1st ed. 2017 ed., pp. 17–34). Springer.

Agius, T., Sabri, S., & Kalantari, M. (2018). Three-Dimensional Rule-Based City Modelling to Support Urban Redevelopment Process. ISPRS International Journal of Geo-Information, 7(10), 413. https://doi.org/10.3390/ijgi7100413

Basiago, A.D.(1999). Economic, social, and environmental sustainability in development theory and urban planning practice. The Environmentalist, 19(2), 145-161.

Batty, M. (2009). Cities as Complex Systems: Scaling, Interaction, Networks, Dynamics and Urban Morphologies. Encyclopedia of Complexity and Systems Science, 1041–1071. https://doi.org/10.1007/978-0-387-30440-3_69

Berman, D. (2021). *What is the MIT License?*. Snyk. https://snyk.io/learn/what-is-mit-license/

Biljecki, F., Heuvelink, G. B., Ledoux, H., & Stoter, J. (2015). Propagation of positional error in 3D GIS: estimation of the solar irradiation of building roofs. International Journal of Geographical Information Science, 29(12), 2269–2294. https://doi.org/10.1080/13658816.2015.1073292

Biljecki, F., Kumar, K., & Nagel, C. (2018). CityGML Application Domain Extension (ADE): overview of developments. Open Geospatial Data, Software and Standards, 3(1), 2–17. https://doi.org/10.1186/s40965-018-0055-6

Biljecki, F., Ledoux, H., Du, X., Stoter, J., Soon, K. H., & Khoo, V. H. S. (2016). THE MOST COMMON GEOMETRIC AND SEMANTIC ERRORS IN CITYGML DATASETS. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, IV-2/W1, 13–22. https://doi.org/10.5194/isprs-annals-iv-2-w1-13-2016

Biljecki, F. (2013). *The concept of level of detail in 3D city models* (GISt Report No. 62). TU Delft. http://www.gdmc.nl/publications/reports/GISt62.pdf

Brito, M. C., Redweik, P., Catita, C., Freitas, S., & Santos, M. (2019). 3D Solar Potential in the Urban Environment: A Case Study in Lisbon. Energies, 12(18), 3457. https://doi.org/10.3390/en12183457

Cillari, G., Fantozzi, F., & Franco, A. (2021). Passive Solar Solutions for Buildings: Criteria and Guidelines for a Synergistic Design. Applied Sciences, 11(1), 376. https://doi.org/10.3390/app11010376

Circular Ecology. (2020, May 17). Sustainability and Sustainable Development. https://circularecology.com/sustainability-and-sustainable-development.html

CityJSON Specification. (2019). *CityJSON Specifications 1.0.1*. https://www.cityjson.org/specs/1.0.1/#extensions

Choi, Y., Suh, J., & Kim, S. M. (2019). GIS-Based Solar Radiation Mapping, Site Evaluation, and Potential Assessment: A Review. Applied Sciences, 9(9), 1960. https://doi.org/10.3390/app9091960

Chow, A., Fung, A., & Li, S. (2014). GIS Modeling of Solar Neighborhood Potential at a Fine Spatiotemporal Resolution. Buildings, 4(2), 195–206. https://doi.org/10.3390/buildings4020195

Daneshvar Tarigh, A., Daneshvar Tarigh, F. & Nikranjbar, A. (2012). A Survey of Energy-Efficient Passive Solar Houses.

Döllner, J., Baumann, K., & Buchholz, H. (2007). Virtual 3D City Models as Foundation of Complex Urban Information Spaces.

Dubois, M.-C., Gentile., N., Laike, N., T., Bournas, I., & Alenius, M., 2019: Daylight and lighting – under a Nordic sky. Studentlitteratur, Lund.

Duffie, J. A., & Beckman, W. A. (2013). Solar Engineering of Thermal Processes (4th ed.). Wiley.
Economic sustainability. (2020, July 11). KTH. https://www.kth.se/en/om/miljo-hallbar-utveckling/utbildning-miljo-hallbar-utveckling/verktygslada/sustainable-development/ekonomisk-hallbarhet-1.431976

EEA (European Environment Agency). (2020). Air quality in Europe - 2020 Report (EEA Report No 9/2020). doi:10.2800/786656

Eriksson, H. (2020). *Harmonisation of 3D geodata : a prerequisite for a digital information flow for applications in the planning and building sector*. Department of Physical Geography and Ecosystem Science, Faculty of Science, Lund University.

EU Science Hub - European Commission. (2019, August 20). *From nearly-zero energy buildings to net-zero energy districts.* https://ec.europa.eu/jrc/en/news/nearly-zero-energy-buildings-net-zero-energy-districts

Fan, H., Kong, G., & Zhang, C. (2021). An Interactive platform for low-cost 3D building modeling from VGI data using convolutional neural network. Big Earth Data, 5(1), 49–65. https://doi.org/10.1080/20964471.2021.1886391

Fosdick, J. D. (2016). Passive Solar Heating. WBDG - Whole Building Design Guide. https://www.wbdg.org/resources/passive-solar-heating

Freitas, S., Catita, C., Redweik, P., & Brito, M. (2015). Modelling solar potential in the urban environment: State-of-the-art review. Renewable and Sustainable Energy Reviews, 41, 915–931. https://doi.org/10.1016/j.rser.2014.08.060

Gabriel, R. P., & Goldman, R. (2005). Innovation Happens Elsewhere: Open Source as Business Strategy. Morgan Kaufmann Publishers.

GNU Project. (2021). *What is Free Software? - GNU Project - Free Software Foundation*. https://www.gnu.org/philosophy/free-sw.html#mission-statement

Gröger, G., Kolbe, T.H., Nagel, C., Häfele, K.H. (2012). OGC City Geography Markup Language (CityGML) Encoding Standard. OGC Doc. No. OGC 12-019.

Gröger, G., & Plümer, L. (2012). CityGML – Interoperable semantic 3D city models. ISPRS Journal of Photogrammetry and Remote Sensing, 71, 12–33. https://doi.org/10.1016/j.isprsjprs.2012.04.004

Huang, W., Olsson, P. O., Kanters, J., & Harrie, L. (2020). Reconciling City Models With BIM in Knowledge Graphs: A Feasibility Study of Data Integration for Solar Energy Simulation. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, VI-4/W1, 93–99. https://doi.org/10.5194/isprs-annals-vi-4-w1-2020-93-2020

Kanters, J., Gentile, N., & Bernardo, R. (2021). Planning for solar access in Sweden: routines, metrics, and tools. Urban, Planning and Transport Research, 9(1), 348–368. https://doi.org/10.1080/21650020.2021.1944293

Kanters, J., Wall, M., & Kjellsson, E. (2014). The Solar Map as a Knowledge Base for Solar Energy Use. Energy Procedia, 48, 1597–1606. https://doi.org/10.1016/j.egypro.2014.02.180

Koglin, T. (2009). Sustainable development in general and urban context : a literature review. Department of Technology and Society, Lund University.

Kolbe, T.H., & Gröger, G. (2003). Towards unified 3D city models.

Kutzner, T., Chaturvedi, K., & Kolbe, T. H. (2020). CityGML 3.0: New Functions Open Up New Applications. PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science, 88(1), 43–61. https://doi.org/10.1007/s41064-020-00095-z

Ledoux, H., Arroyo Ohori, K., Kumar, K., Dukai B., Labetski, A., Vitalis, S. (2019). CityJSON: A compact and easy-to-use encoding of the CityGML data model. Open Geospatial Data, Software and Standards, 4:4

Liang, J., Gong, J., Xie, X., & Sun, J. (2020). Solar3D: An Open-Source Tool for Estimating Solar Radiation in Urban Environments. *ISPRS International Journal of Geo-Information*, *9*(9), 524. MDPI AG. http://dx.doi.org/10.3390/ijgi9090524

Lindberg, F., Jonsson, P., Honjo, T., & Wästberg, D. (2015). Solar energy on building envelopes – 3D modelling in a 2D environment. Solar Energy, 115, 369–378. https://doi.org/10.1016/j.solener.2015.03.001

Lorek, S. (2021). *What is BIM?* Trimble. https://constructible.trimble.com/construction-industry/what-is-bim-building-information-modeling

Lucassen, G., Dalpiaz, F., Werf, J. M. E. M. V. D., & Brinkkemper, S. (2016). The Use and Effectiveness of User Stories in Practice. Requirements Engineering: Foundation for Software Quality, 205–222. https://doi.org/10.1007/978-3-319-30282-9_14

Lundgren, M., & Dahlberg, J. (Eds.). (2018). Approaches, Methods and Tools for Solar Energy in Urban Planning. IEA SHC Task 51 Solar Energy in Urban Planning. https://doi.org/10.18777/ieashc-task51-2018-0004

Medeiros, E. (2020). Urban SUNstainability: a multi-dimensional policy evaluation framework proposal. Cidades, Comunidades e Território. 40. 117-133. https://doi.org/10.15847/cct.jun2020.040.art01

Mercer, A. (2020). *buildingSMART International (bSI) and Open Geospatial Consortium (OGC) Release BIM and GIS Integration Paper*. BuildingSMART International. https://www.buildingsmart.org/buildingsmart-international-bsi-and-open-geospatial-consortium-ogc-release-bim-and-gis-integration-paper/

NASA - Solar Physics. (2021). NASA/Marshall Solar Physics. https://solarscience.msfc.nasa.gov/interior.shtml

Nouvel, R., Mastrucci, A., Leopold, U., Baume, O., Coors, V., & Eicker, U. (2015). Combining GIS-based statistical and engineering urban heat consumption models: Towards a new framework for multi-scale policy support. Energy & Buildings, 107, 204–212. https://doi-org.ludwig.lub.lu.se/10.1016/j.enbuild.2015.08.021

Open Source Initiative. (2018). *History of the OSI*. https://opensource.org/history

OSGeo. (2019, August 21). *About*. https://www.osgeo.org/about/

Pedamkar, P. (2021, April 9). Advantages of OOP. EDUCBA. https://www.educba.com/advantages-of-oop/

QGIS. (2021). *Discover QGIS*. https://www.qgis.org/en/site/about/index.html

Quan, S. J., Li, Q., Augenbroe, G., Brown, J., & Yang, P. P. J. (2015). A GIS-based Energy Balance Modeling System for Urban Solar Buildings. Energy Procedia, 75, 2946–2952. https://doi.org/10.1016/j.egypro.2015.07.598

Sadeghsaberi, J., Zarei, S., & Hemmati, S. (2013). Passive solar building design.

Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S., & Lakhani, K. (2006). Understanding Free/Open Source Software Development Processes. Software Process: Improvement and Practice, 11(2), 95–105. https://doi.org/10.1002/spip.255

Social Sustainability. (2020, May 12). Smart City Sweden. https://smartcitysweden.com/focus-areas/social-sustainability/ (last accessed: 20.02.2021)

Solar Energy - Introduction to SEBE — UMEP Tutorial documentation. (2018). Solar Energy - Introduction to SEBE. https://tutorial-docs.readthedocs.io/en/latest/Tutorials/SEBE.html#sebe

Stadler, A., & Kolbe, T.H. (2007). Spatio-semantic coherence in the integration of 3D city models.

Tamm, M., Macià Cid, J., Capdevila Paramio, R., Farnós Baulenas, J., Thalfeldt, M., & Kurnitski, J. (2020). Development of a Reduced Order Model of Solar Heat Gains Prediction. Energies, 13(23), 6316. https://doi.org/10.3390/en13236316

TechRepublic. (2020). *2020 Open Source Security and Risk Analysis Report.* https://www.techrepublic.com/resource-library/whitepapers/2020-open-source-security-and-risk-analysis-report/

U.S. Department of Energy. (2021). *Passive Solar Home Design.* https://www.energy.gov/energysaver/passive-solar-home-design

Urban Development. (2020, April 20). World Bank. https://www.worldbank.org/en/topic/urbandevelopment/overview

Wate, P., & Saran, S. (2015). Implementation of CityGML energy application domain extension (ADE) for integration of urban solar potential indicators using object-oriented modelling approach. Geocarto International, 30(10), 1144–1162. https://doi.org/10.1080/10106049.2015.1034192

Wheeler, S. (1996). *Sustainable Urban Development: A Literature Review and Analysis*. Retrieved from https://escholarship.org/uc/item/6mx0n01x

APPENDIX A - Technical documentation of object classes

This section provides a detailed description of the code and solution for each object class used during the main process of DET tool. The description includes input parameters, class attributes and class methods for each class. Furthermore, methods are broken down to small pieces in order to help understand all algorithms used including mathematical and geometric formulas.

"InitialRaster" Object class

Parameters:

- raster image representing the solar irradiance on the rooftops

Attributes:

- x coordinate of top-left corner of the image in spatial reference system (float)
- y coordinate of top-left corner of the image in spatial reference system (float)
- image resolution (int),
- number of rows (int)
- number of columns (int)
- Coordinate Reference System (string)

Methods: -

"FacadeData" Object Class

Parameters:

- text file representing the solar irradiance on the building facades

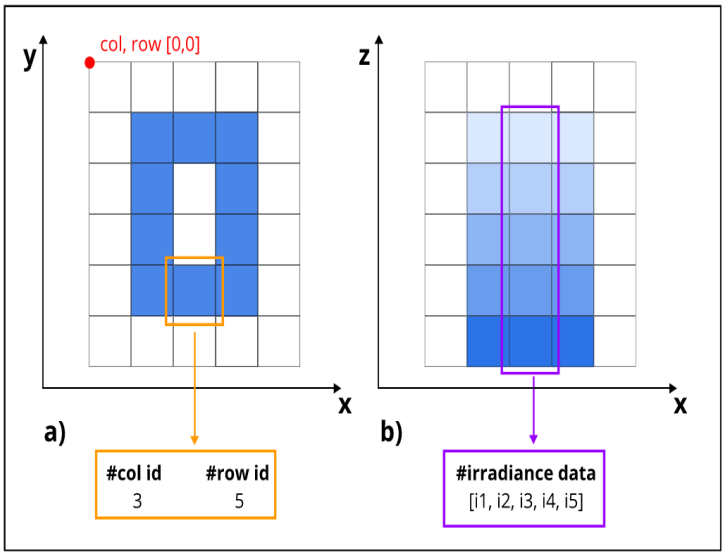


Figure A1 - *The example of a record in the facade energy text file*

Attributes:

- path to the file (string)
- file size (float)
- file data (list of dictionaries)

Methods:

.*processInput():*

- reads data from text file which has structure shown on Figure A1
- parses the data into a dictionary format with following structure `{row_id (int), column_id (int), irradiance_values [(float)]}`
- appends each dictionary into the file data attribute

"Voxel" Object class

Parameters: *FacadeData* Object Class

Attributes:

- voxel index(int)
- column index (int)
- row index (int)
- irradiance values (list of floats)
- building index (int)
- facade index (int)
- facade start node (Boolean)
- facade end node (Boolean)
- x coordinate (float)
- y coordinate (float)

Methods:

.*defineCoordinates(InitialRaster)*

- uses the x and y coordinate of the top-left corner of the image, which is an attribute of the *InitialRaster* object class
- calculates the values of x,y coordinates of centers of each voxel using a simple formula:

$$x_{CRS} = x_{lt\_corner} + col\_id * x\_resolution - x\_resolution/2$$
$$y_{CRS} = y_{lt\_corner} + row\_id * y\_resolution + y\_resolution/2$$

Formula A1 *- Coordinates calculation formula*

"Building" Object class

Parameters:

- building footprints with the height stored as an attribute (shapefile)

- CityObject feature index (retrieved from 3D city model)

Attributes:

- feature index of a city object (string)
- building index (int)
- building footprints (list of floats)
- building height (float)
- list of facades (*Facade* objects)
- building voxel list (*Voxel* objects)
- corner areas (shapely MultiPolygon object); if exception occurs format is list of polygons
- list of corners (shapely point objects)
- list of facade line strings (shapely LineString objects)
- windows geometry (list of *Window* Objects)
- surfaces (dictionary)
- roof (a *Roof* object)

Methods:

.*assignBuildingIndex(voxels)* - since the UMEP pre-processor rasterizes building footprints, the location of voxels objects are not exactly on the building facades but in their close proximity (situation on Figure A2).

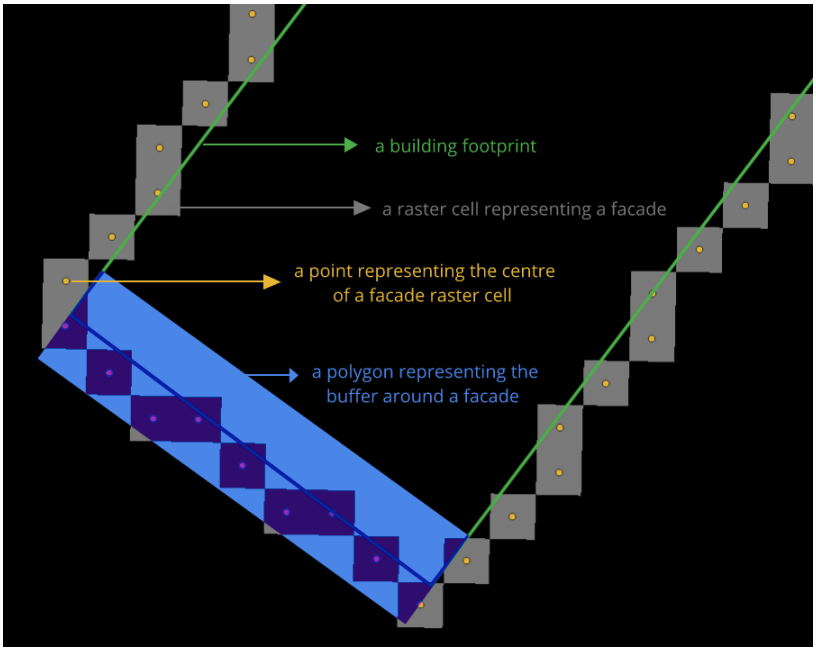


Figure A2 *- Facade raster, building footprints and voxel objects*

To identify all the voxels belonging to a building, the method performs the following operations:

- creates a buffer around the building footprints

- uses the point in polygon method to determine whether a *Voxel* object is situated in the buffer envelope or not
- if the examined *Voxel* object is located in the buffer envelope, corresponding building index is assigned to the *Voxel* object as its building index attribute
- identified *Voxel* objects are appended to the building voxel list attribute

.*getfacadeList()*

- iterates through the list of coordinates representing the footprints
- populates the corner list attribute
- initializes *Facade* objects and based on the recognition of start nodes and the end nodes of each facade - start node of a facade represents the current node (iterable with the index value *i* at any time of the loop) and the end node represents the next node (iterable with the index value *i+1)*
- appends *Facades* objects to the facade list attribute

.*createCornerAreas()* - this method creates a polygon area around each corner of a building. The method uses a simple algorithm that can be represented as following pseudo-code:

```
for each corner
       check side of the next corner
       if right
              → createConvexCornerArea()
       else
              → createConcaveCornerArea()
       self.corner_area = created_polygon
```

Figure A3 shows differences between convex and concave corner areas.

The method performs the following operations:

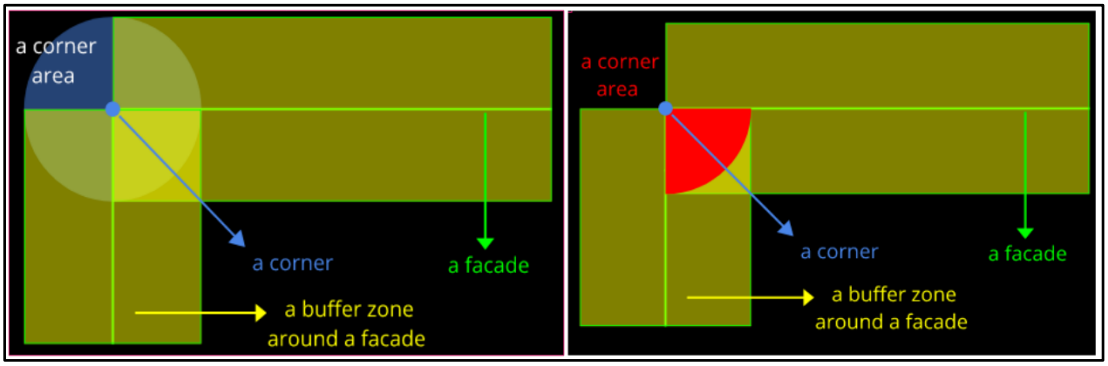
- a side function is utilized to determine whether the next corner lies on the right or left side
- based on the result .*createConvexCornerArea()* or .*createConcaveCornerArea()* method is executed (explained below)
- created polygons stored as the corner areas attribute

.*createConvexCornerArea()*

- creates buffer envelopes around facades that feature the corner
- creates a union of facade buffers
- creates a round buffer around the corner
- subtracts the union of facade buffers from the corner buffer - corner area
- returns resulting corner area
-

.createConcaveCornerArea()

- creates buffer envelopes around facades that feature the corner
- creates a union of facade buffers
- creates a round buffer around the corner
- performs an intersection between the union of facade buffers from the corner buffer - corner area
- returns resulting corner area



A3 *- Convex(left) and concave(right) corners*

.assignFacadeIndex() - this method is very similar to the *assignBuildingIndex()* method, the pseudo code for this method has a following structure:

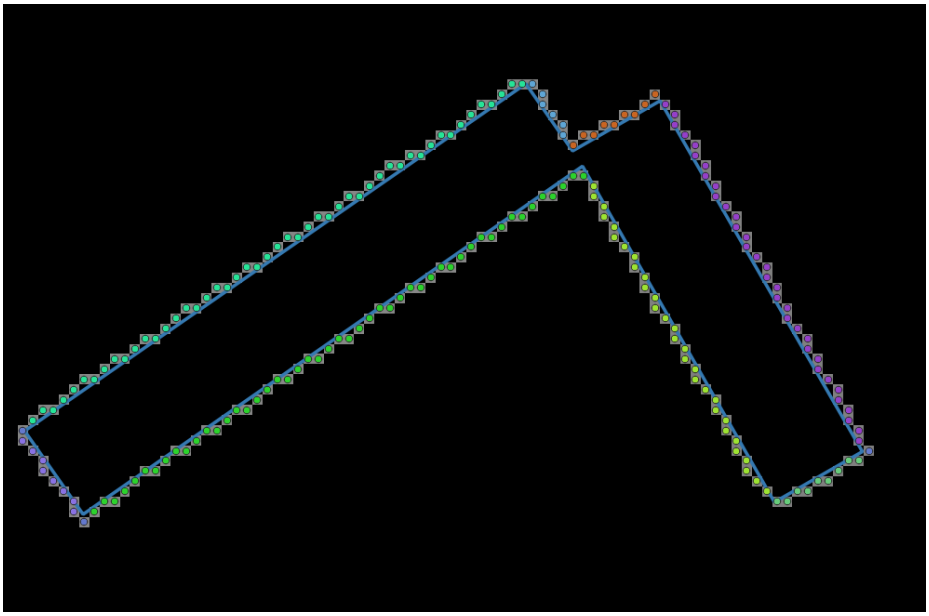
```
Corner voxel list = []
for each facade
       for each voxel in building_voxel_list
              if voxel falls in corner area
                     append to corner voxel list (if it is not already
       there)
              else
                     if voxel falls in facade.buffered_area
                            voxel.facade_id = facade.id
resolve_corners(corner voxel list)
```

The method performs the following operations:

- uses the point in polygon method to determine whether a *Voxel* object is situated in a corner area or not
- if the examined *Voxel* object is located in a corner area, the method appends it to corner voxel list, if it is not already there
- if the examined *Voxel* object is not located in a corner area, the method uses the point in polygon method to determine whether a *Voxel* object is situated in a facade buffer
- if the examined *Voxel* object is located in a facade buffer, the *Voxel* object inherits a corresponding facade index

.resolveCorners(corner_voxel_list) - this method resolves the corner list because the previous method is insufficient for *Voxel* objects located in corner areas.
The method performs the following operations:

- iterates through corner voxel list
- calculates distances to each corner of a building
- identifies facades which create the closest corner
- calculates distances to previously returned facades
- the examined *Voxel* object inherits a facade index of the closest facade



A4 *– Voxel objects categorized by facade id*

"Facade" Object class

Parameters:

- feature index of a city object
- facade index
- facade start node
- facade end node
- building height

Attributes:

- feature index (string)
- facade index(int)
- facade start node (Point geometry)
- facade end node (Point geometry)

- building height (int)
- facade buffer (Polygon geometry)
- facade voxel list (list of *Voxel* objects)
- facade windows (list of *Window* features)
- facade raster (*FacadeRaster* object)
- facade windows raster (Numpy matrix object)
- area (float)
- total irradiance (float)
- total irradiance per area (float)

Methods:

.*getfacadeWindows(Building Object)* - the function returns the list of *Window* objects associated with a particular facade

- iterates through a list of *Windows* objects associated with a particular building
- checks the facade index of a window
- returns a list of *Window* objects where the indices between a window and facade matched

.*getWindowIrradiance()* - the function is used to estimate solar irradiance on window surfaces

- creates a zero matrix with the size matching facade height and facade width
- the distance between an edge of the window feature and the starting node of the facade is calculated - the rounded distance represents the column index (j) within the empty matrix
- assigns row index (i) which is equal to the elevation where the window bottom edge lies
- generates submatrix based on the following formula where n is a window width:

```
submatrix = raster_window[i:i + window.height, j:j + n]
```

- raster cells within the submatrix obtains 1 as their value (value 1 then represents a window feature as long as value 0 means a facade)
- multiplies the submatrix by facade raster
- sums non-zero values and applies ratio correction for the estimated value (ratio is equal to real area of window against rounded area used in submatrix) and stores this value as the *total irradiance* attribute of a *Window* object
- divides the total irradiation per window area and stores this value as the *total irradiance per area* attribute of a *Window* object

.*getFacadeIrradiance()* - this method works on a similar principle as the .*getWindowIrradiance()* method. The difference between them is that this function firstly creates a matrix with ones instead

of zeros, and submatrix is getting value 0 instead of 1creates a zero matrix with the size matching facade height and facade width

- creates a one matrix with the size matching facade height and facade width
- the distance between an edge of the window feature and the starting node of the facade is calculated - the rounded distance represents the column index (j) within the empty matrix
- assigns row index (i) which is equal to the elevation within a facade of window bottom edge
- generates submatrix based on the following formula where n is a window width:

```
submatrix = raster_window[i:i + window.height, j:j + n]
```

- raster cells within the submatrix obtains 0 as their value (value 0 then represents a window feature as long as value 1 means a facade)
- multiplies the submatrix by facade raster
- sums non-zero values and and stores this value as the *total irradiance* attribute
- divides the total irradiation per facade area and stores this value as the *total irradiance per area* attribute

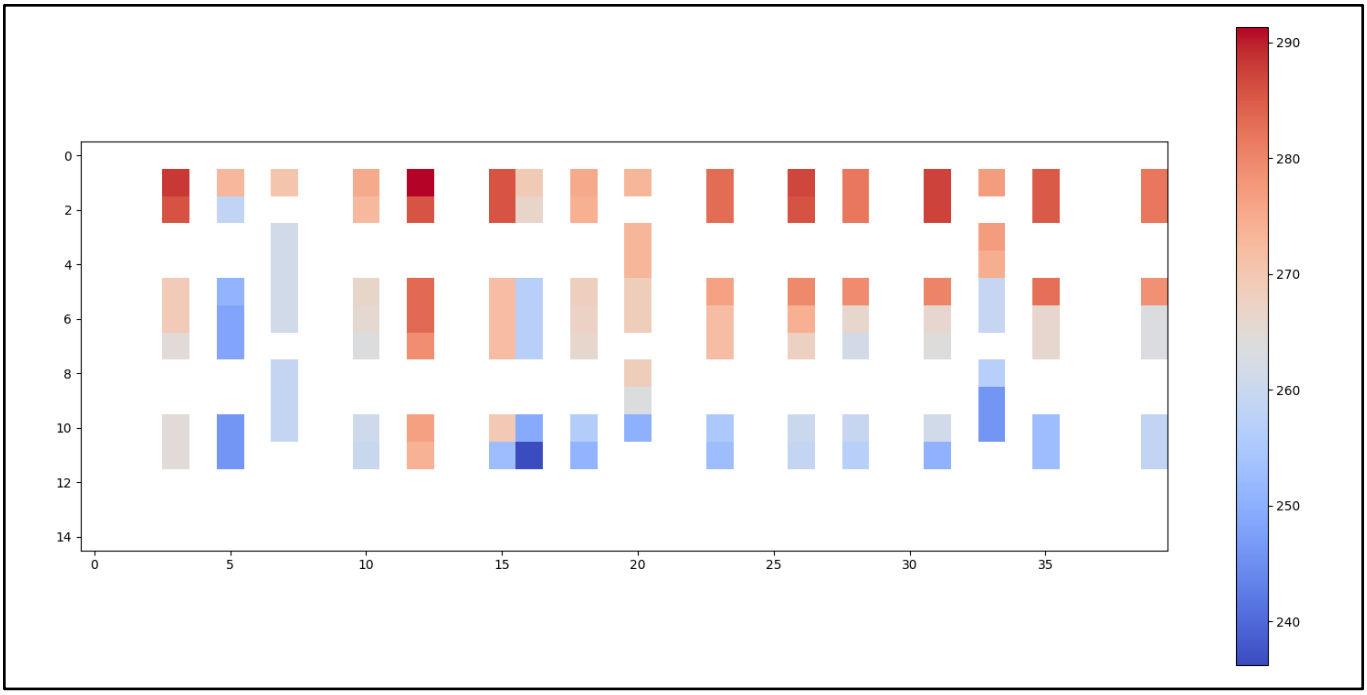


Figure A5 *– Estimated values for windows on a random facade*

“FacadeRaster” Object class

Parameters:

- feature index of a city object
- facade index
- facade start node
- facade end node
- building height

Attributes:

- building height (int)
- facade index(int)
- facade start node (Point geometry)
- facade end node (Point geometry)
- acade voxel list (list of Voxel objects)
- columns (int)
- rows (int)
- resolution (int)
- grid (Numpy matrix object)

Methods:

.*getRasterDimension()*

- calculates distance between facade starting node and facade ending node
- the rounded is assigned as the *columns* attribute
- assigns facade height as the *rows* attribute
- creates an empty matrix with the size of columns and rows
- stores the matrix as the grid attribute

.*getfacadeVoxelList(Building Object)* - the function returns the list of *Voxel* objects associated with a particular facade from a list of *Voxel* objects linked to a particular building

.*calculateLineCoords()* - this method adjusts x,y coordinates of *Voxel* objects and relocates them on a line representing a facade, Figure A6 illustrates the situation and the geometric properties of elements during the coordinate adjustment process. The result of this function can be seen on the Figure A7.

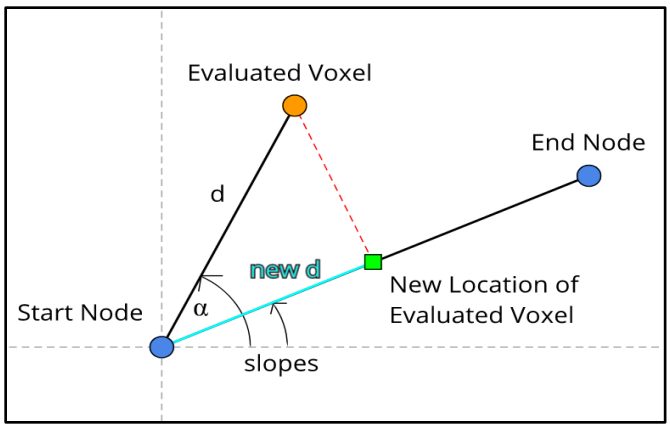


Figure A6 *- Coordinate adjustment process*

The method performs the following operations:

- calculates the quadrant (based on the direction of a facade)
- calculates the bearing of a facade
- calculates the angle α (see Figure A6) as a difference of slopes
- calculates new distance as d multiplied by the $cos(\alpha)$ (see Figure A6)
- calculates coordinate increments for the adjusted position of evaluated voxel based on formulas (signs +/- can change based on the quadrant):

$$x_{adj} = x_{start\_node} + new\_d * cos(bearing)$$
$$y_{adj} = y_{start\_node} + new\_d * sin(bearing)$$

Formula 2 *- Coordinates adjustment calculation formula*

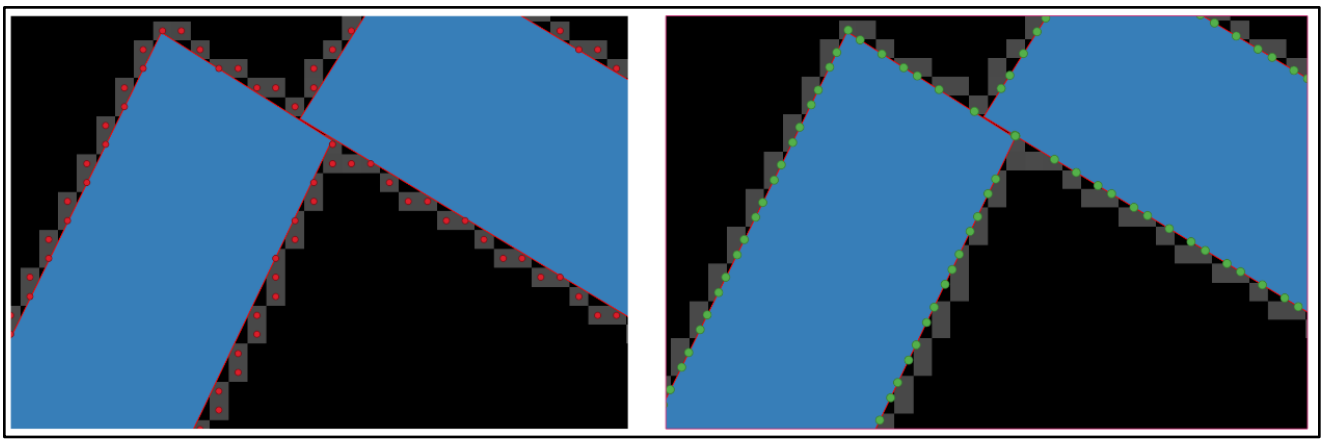


Figure A7 *– Unadjusted coordinates (left) vs. adjusted coordinates (right)*

.populateGrid() - this is a nested function responsible for populating an empty matrix (class attribute grid) with correct irradiance values. Figure A8 shows the diagram of the funcion

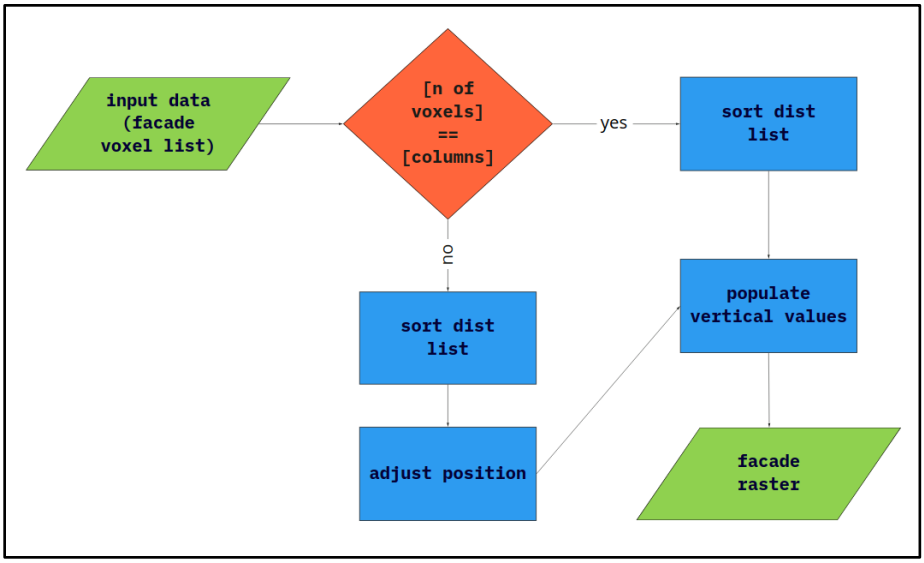


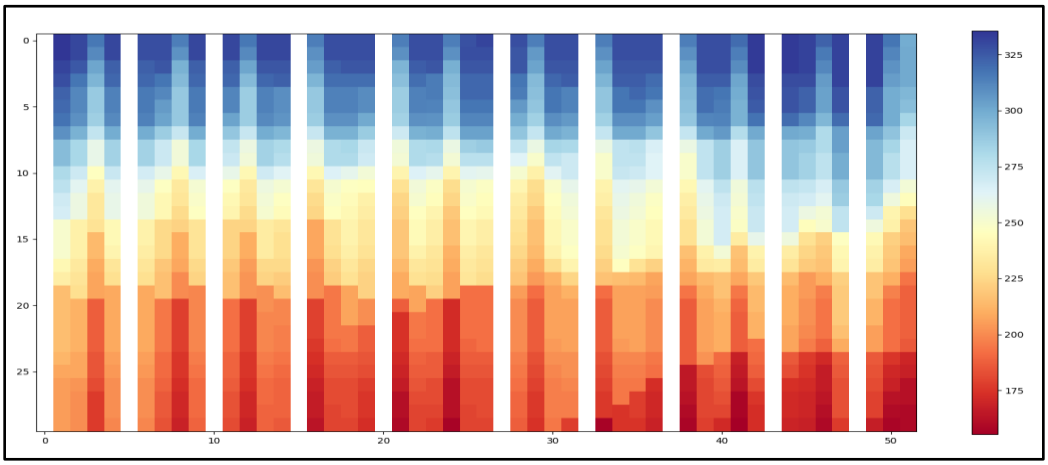
Figure A8 *– populateGrid() function diagram*

The function performs these operations:

- checks if the number of assigned voxels for a facade is equal to the number of columns
  - if yes
    - the function sorts the voxels according to their distance from the starting node
    - the rounded distance indicates a column index which will obtain the corresponding irradiance values from voxel
    - calls the function *.populate_vertical_values()*
  - if no
    - the function sorts the voxels according to their distance from the starting node
    - the rounded distance indicates a column index which will obtain the corresponding irradiance values from voxel
    - if two voxels have the same rounded distance, function *.adjustPosition* is called and finds the closest free index
    - the rounded distance after the adjustment indicates a column index which will obtain the corresponding irradiance values from voxel
    - calls the function *.populate_vertical_values()*

.populateVerticalValues - unpacks the voxel irradiance data stored in the data attribute of *Voxel* object and assigns them to a correct row index. The result of the function is shown on Figure A9.

- checks if number of rows is equal, less or more than the number values in the data attribute of *Voxel* object
    - the number of values is equal to the number of rows
        - the grid value is then equal to a value of element with the inverse index stored in the data attribute of a *Voxel* object[21]
    - the number of values is lower than the number of rows
        - the grid value is then equal to a value of element with the inverse index stored in the data attribute of a *Voxel* object in direction from to the top of the facade - the bottom of facade remains without irradiance value
    - the number of values is greater than the number of rows
        - a sublist with the size equal to the number of rows is created, the sublist is created in direction from the top of the facade so the values representing the bottom of facade are omitted
        - the grid value is then equal to a value of element with the inverse index stored in the data attribute of a *Voxel* object



FigureA9 *– facade raster after .populateGrid function*

.interpolate() - as can be seen on the Figure A9, a few cells have been left unassigned, so they have to be interpolated. The method performs these operations:

- creates a floating window of the size 3x3 cells (floating window is a sliced submatrix from the original matrix stored as the grid parameter)
- the floating window moves over the grid and detects unassigned cells (the value of a central cell is equal to 0)

[21]The index is inverse because the function populates the values from the top of the raster as long as the list of irradiance values starts with bottom values first

- if an unassigned value is detected the mean value is calculated from the remaining values of the floating window and that value is assigned to the central cell.

Figure A10 shows the result of the interpolation on a random facade.

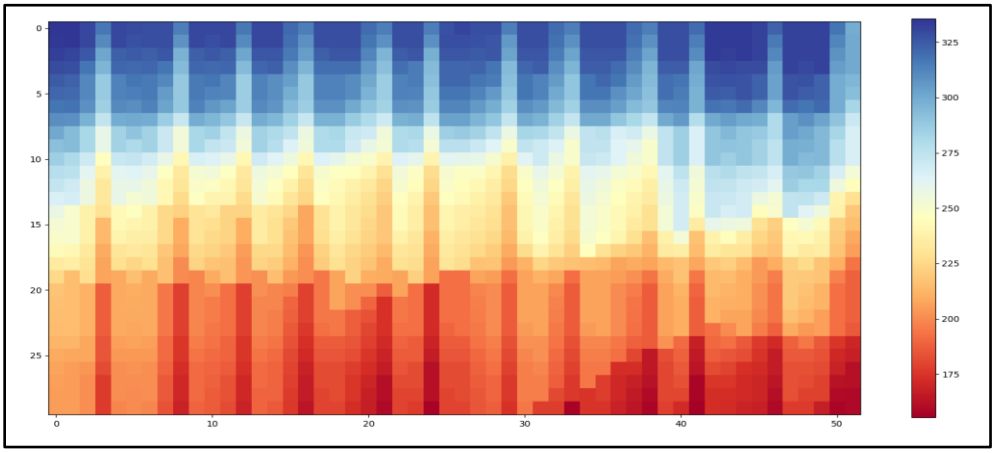


Figure A10 *– facade raster after interpolation*

"Roof" Object class

Parameters:

- feature index of a city object
- *InitialRaster* Object

Attributes:

- feature index (string)
- roof geometry (Polygon)
- total irradiance (float)
- total irradiance per area (float)
- roof vertex geometry (list of dictionaries)
- area (float)
- total irradiance (float)
- total irradiance per area (float)

Methods: *.getRoofGeometry(city model data)* - this method gets the geometry of roof surfaces from the city model. It is done by:

- finds out the index of roof surfaces
- gets the boundaries list from geometry which are associated with roof features
- extracts coordinates from the list of vertices
- returns the geometry as a list of dictionaries and stores them as the roof vertex geometry attribute

.getRoofPolygon() - this method creates a Polygon object based on the geometry retrieved within the previous method

- iterates through the roof vertex geometry attribute
- gets x,y coordinates of each vertex
- appends them to a list
- creates a shapely polygon object from the list of vertices
- the polygon is stored as the roof geometry attribute

.extractRoofIrradiance(InitialRaster.raster) - this method is used to estimate the total irradiance on the rooftops. To make that happen, the method performs these steps:

- applies *rasterio.mask* function to create and extract the values, the input is the roof raster and Polygon object representing the roof surface
- the function is applied to extract only values residing inside of the mask
- the output of the function is an array containing all values extracted by the mask function.
- the total irradiance on the rooftop is then calculated as the sum of the array and stored as the *total irradiance* attribute
- divides the total irradiation per roof area and stores this value as the *total irradiance per area* attribute

---

"Window" Object class

Parameters:

- feature index of a city object
- Retrieved geometry from the 3D city model
- index of a window feature

Attributes:

- geometry [list of coordinate pairs]
- feature index(string)
- window index (int)
- width (float)
- height (int)
- window start x,y ([coordinate pair])
- window end xy ([coordinate pair])
- window_height_index (int)
- facade id (int)
- valid (Boolean)
- total irradiance (float)

- total irradiance per area (float)
- suitability (float)

Methods: ***.****setExtent(facade)* - this method figures out the location and orientation of a window feature within the facade. Following operations are used to get a desired result:

- the method loops over through the geometry attribute
- each vertex stored in the list is examined following way:
    - distance to the facade start node is calculated
        - the vertex which is closest to the start node is stored as window start x,y attribute
        - the vertex which is furthest to the start node is stored as window end x,y attribute
    - max and min values of each vertex z-coordinate are compared
- the result of the function is populating the following attributes: window height index receives z-min value; the window height attribute is getting value which equals to z-max-z-min; the window width attribute is equal to distance between window start x,y and window end x,y attributes

.assignFacade(Facade object, buffer) - the method uses the same principle as the method assigning voxels. Function checks if both points representing the window are located within the buffer area around a facade. If yes, the window object inherits the corresponding facade index.

APPENDIX B - Data preparation description

This section covers two main tasks which have to be performed before the DET is run. First task is regarding the manipulation of the initial dataset. The second step is performing the UMEP energy simulation. Successful performance of these operations ensures the correct input data for the DET.

Initial dataset manipulation

A simple "pre-processor" toolbox, which creates a shape file representing building footprints with the height attribute, was developed. This toolbox extracts the geometry of building footprints from the 3D city model. Subsequently, building height values are assigned to the corresponding polygon footprints feature. The toolbox uses a simple algorithm that iterates through all vertices of each feature and stores the lowest value and the highest value of Z-coordinate for each building. The building height represented a difference between those coordinates.

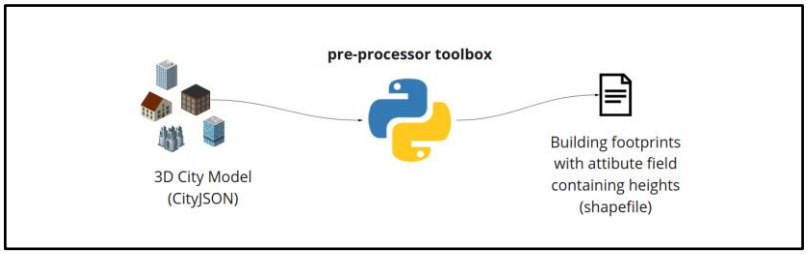


Figure B1 *– Initial data manipulation*

UMEP Energy simulation

The UMEP simulation tool has been introduced in section 2.4.3.1. This section describes all the tasks which have been performed by the tool. The subsection is split into 2 parts, following the architecture of the UMEP tool (figure B2).

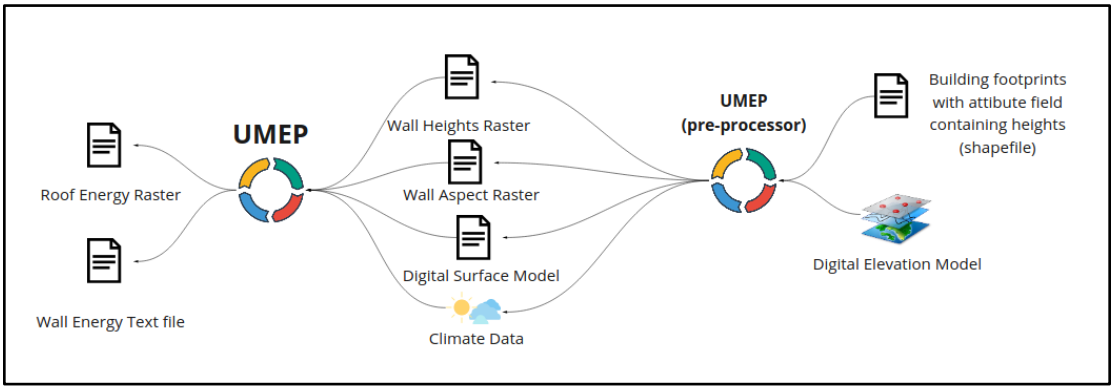


Figure B2 *- UMEP energy simulation process*

UMEP Pre-processing

Data input: DEM, building footprints with the building height stored as an attribute - shapefile

Process: The most important task in this step is creating the DSM (digital surface model). The pre-processor combines these two layers and creates a new raster file, which represents heights of the buildings and surrounding area. Another task is generating wall height raster and wall aspect raster files. Both files are derived from the DSM. The UMEP Pre-processor creates both required files.

The last requirement for the actual simulation is meteorological data. The meteorological data can be downloaded from the Climate Data Store (CDS) of the Copernicus program through the UMEP model's user interface. The only requirement is to set up an account in order to access the data via CDS API. The dataset is retrieved as the ERA5 standard, which provides climate data on an hourly basis. The required parameters in order to calculate solar energy are time parameters (year, day of the year, hour of the day) as well as parameters related to shortwave radiation - incoming shortwave radiation [W m^{-2}], diffuse radiation [W m^{-2}] and beam radiation [W m^{-2}].

Data output: DSM raster, Wall Height raster, Wall Aspect raster, meteorological data

UMEP Energy Simulation

Data input: DSM raster, Wall Height raster, Wall Aspect raster, meteorological data

Process: Since all required input files have been successfully created, it is possible to run the actual solar energy estimation on the rooftops and building walls. The output of the simulation is a raster file showing total irradiance per pixel in kWh and a text file storing the information about total irradiance on the building walls. The file contains row and column numbers for each raster cell representing a building wall (based on the original DSM) following with values for irradiance on the building wall. The values represent voxels and when the value for irradiance reaches 0 it means that the maximum height of the building has been reached.

Data output: Raster image representing the solar irradiance on the rooftops, a text file representing the solar irradiance on the building facades