UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Bohdan Romashchenko

Mapping Solar Potential of Tartu

Master's Thesis (30 ECTS)

Supervisor(s): Pelle Jakovits, PhD

Tartu 2022

Mapping Solar Potential of Tartu

Abstract:
As the photovoltaic (PV) systems become more and more popular and developed there also grows the need for tools that would help to estimate the usability and impact of such systems on a larger scale. The use case addressed in this work is estimation of solar energy production on the city scale. This information could help city governments to easier find ways to increase solar energy production and, hence, improve energy efficiency of their city. Such information could be obtained by analyzing the rooftops of the building in the city and, luckily, datasets with detailed city geometry for Estonian cities are publicly available. With regards to photovoltaic systems, the datasets mentioned above can be used as a starting point to analyze the roofs, extracting the attributes such as area, tilt and location for further processing. In this work the author aimed to create a solution that would allow to estimate the amount of solar power produced by Tartu or any other Estonian city present in the dataset given by Estonian Land Board. As a result, author has created a prototype consisting of a processing pipeline that does the analysis of geometry and estimation of solar power, as well as a web app to visualize the results. Considering previous work, there have been similar studies which are either not publicly available, not suitable or outdated. The solution described in this work uses the latest technology and no paid external services. Additionally, it's designed to work with Estonian data out of the box which was not possible to use with some of the existing tools. Hopefully the developed tool will be further improved and used by local governments to improve the energy efficiency in various cities of Estonia.

Keywords:
Solar Panel, Solar Potential Estimation, Smart City

CERCS:
P170 Computer science, numerical analysis, systems, control

Tartu päikesepotentsiaali kaardistamine

Lühikokkuvõte:
Kuna fotogalvaanilised (PV) süsteemid muutuvad üha populaarsemaks ja arenevad, kasvab ka vajadus lahenduste järele, mis aitaks hinnata selliste süsteemide kasutatavust ja mõju laiemalt. Käesolevas töös käsitletakse kasutusjuhtumina päikeseenergia tootmise potentsiaali hindamist linna mastaabis. See teave võib aidata linnavalitsustel hõlpsamini leida võimalusi päikeseenergia tootmise suurendamiseks ja seeläbi parandada oma linna energiatõhusust. Sellist infot võiks saada linnas asuvate hoonete katuseid analüüsides ja õnneks on Eesti linnade kohta detailse linnageomeetriaga andmestikud avalikult kättesaadavad. Fotogalvaaniliste süsteemide puhul saab ülalnimetatud andmekogusid kasutada katuste analüüsimise lähtepunktina, eraldades edasiseks töötlemiseks atribuudid, nagu katuse pindala, kalle ja asukoht. Käesoleva töö eesmärk on luua lahendus, mis võimaldaks hinnata Tartus või ka tesites Eesti linnades võimalikku toodetavat päikeseenergia kogust. Selle tulemusena on autor loonud prototüübi, mis koosneb andmetöötluskoveierist, mis viib läbi geomeetrilist analüüsi, hindab potentsiaalset toodetaad päikeseenergia mahtu, ja veebirakendusest tulemuste visualiseerimiseks. Varasemalt on sarnaseid uuringuid tehtud, aga nende tulemused ei ole avalikult kättesaadavad, ei ole praktiliselt rakendatavad Tartu linnale või on aegunud. Käesolevas töös kirjeldatud lahendus kasutab uusimat tehnoloogiat ja ei kasuta tasulisi kolmandate osapoolte teenuseid. Lisaks on see loodud töötama Eesti andmetega, mida ei olnud võimalik kasutada mõne olemasoleva tööriistaga ilma neid täiendamata. Loodetavasti täiustatakse väljatöötatud vahendit tulevikus veelgi ja selle võtavad kasutusse kohalikud omavalitsused energiatõhususe parandamiseks erinevates Eesti linnades.

Võtmesõnad:
Päikesepaneel, päikeseenergia potentsiaali hindamine, tark linn

CERCS:
P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Contents

1 Introduction

Solar potential in this study means the amount of energy the building can produce if it had solar panels installed on the rooftop. The knowledge of solar potential of the city would allow the city government to estimate the amount of additional renewable energy that the city could be producing. Also if the general city-wide estimations look promising, it will give a strong foundation to encourage citizens to deploy solar panels at their homes. Even though Estonia does not have many sun hours in general, some surfaces with a particular tilt and azimuth could be a profitable investment. On the city scale it would mean a lot of saved energy.

The aim of this work was to create a tool capable of delivering the detailed information about solar potential of all the buildings, as well as the total estimations for Estonian cities. For easier interpretation, author has also created a web application that demonstrates the results for several Estonian cities. This thesis also describes how 3D data of Estonian cities can be analyzed, which could be interesting for similar use cases.

The project itself was requested by the city of Tartu and inspired by the similar project called SUN-AREA done in the city of Osnabrück [23]. SUN-AREA, based on the description, is done with the help of the ArcGIS Desktop tools which have a paid license. Since it is not code-first nor publicly available it is impossible to directly use its approach for other cities.

Estimating solar potential of the buildings is not trivial, it depends on location, time, surface area, tilt and azimuth, weather, etc. Solar potential directly depends on the amount of the sunlight that the surface can capture and the efficiency of the solar panels installed, which might slightly vary under different conditions. Therefore the outlined factors are important. Accurate 3D data of the cities is also important, as it allows to extract the roof surface parameters.

Based on the research of existing work, similar problems could be solved with the help of the GIS(Geographical Information System) tools, such as ArcGIS, or custom software which is not available for wider use. ArcGIS, for example, offers a powerful toolset, however has several limitations, such as a big amount of manual work to process a single city. Other GIS tools the author studied, also were not fully suitable, as they were either not possible to use programatically, were difficult to integrate or had commercial license. That is why in this work the author has decided to propose an open-source solution suitable for Estonian cities, that can be used easily and efficiently based on the previous work and available tools and validate it.

The biggest inspiration for this work was Solar3DCity project [28] which is a Python Command Line Interface (CLI) tool that is able to calculate solar potential of the buildings on the larger scale. Solar3DCity is open-source and can be modified or extended as needed, as well as capable of processing many cities at once. This is something that the author wanted to achieve as well. Solar3DCity aimed to improve upon SUN-AREA however it is outdated now and it doesn't have the weather data for Estonia, which is vital.

In this work the author has aimed to create a similar solution with a modern technology stack and specifically designed to work with Estonian cities out of the box. However, compared to Solar3DCity, the developed solution also strongly differs in implementation having: roof detection, different solar estimation implementation, different output format and Docker support. Additionally, a web app to visualize the results was developed.

Regarding the structure of this thesis, in section 2, the author describes the general background of the work, what is solar potential and how applicable is the current project. Then, section 3 describes how the 3D data provided by the Land Board can be used to extract the attributes useful for estimating solar potential. In the section 4 the author describes how it is possible to estimate solar potential, given the roof attributes. Next, Section 5 describes how the data processing and solar potential estimation were implemented. Section 6 describes how can the obtained results be visualized, what are the options and limitations for visualizing 3D data. In the section 7 it is described how the developed application was deployed and validated. In the last section there are conclusions and future work.

2 Domain background

To begin with, it is important to understand the needed domain terminology a bit better. This section provides a brief overview about modern solar energy collection in the urban environment.

2.1 Solar potential

Solar irradiance describes the rate of energy that is being accumulated on a surface (power per unit area), expressed in W/m2. It varies depending on the location of the surface, time and date, atmospheric conditions, and other factors [28]. Solar irradiation also has to be integrated over time to have a more easily interpretable result, usually one hour is used as a time frame. Obtained result is then expressed in Wh/m2.

The solar potential in this study is the total amount of solar energy that can been collected on a surface area by means of using solar panels within a year, expressed in kWh/m2/year. It is important to understand that solar panels cannot capture all the incoming energy. Even though the technology is constantly improving, at the moment of writing this work, most solar panels have around 20% efficiency. Additionally, there are always some losses related to wear, weather, etc.

For estimating solar potential of buildings, knowing the precise structure and shape of the roof area is crucial. Solar potential depends on the roof area, tilt and azimuth, because those determine how much direct sunlight can be captured by the panels. Two identical roofs with different angles could have a big difference in solar potential [41]. Then, solar potential also depends on different weather conditions: temperature, pressure, amount of sun-hours, to help capture this information knowing the location of the building is vital.

In theory it should also be possible to do an estimation of solar potential purely based on the area of the whole city, however it simplifies way too many factors. First of all, not all area of the city is suitable for deployment of solar panels - spaces like parking lots and roads are not. Second of all, such approach would simplify the geometry of all the buildings and that would also result in a huge error in estimations. Lastly, roof-integrated solar panels are quite a common technology nowadays and it is definitely a suitable technology to begin the city wide transformations towards renewable energy.

2.2 Modern PV systems

As mentioned in the previous subsection, solar panels cannot capture all the incoming energy. It is important to consider this factor in the developed tool, for example to provide meaningful default values for some parameters.

Based on the recent articles and blogs, the most modern PV systems can achieve the efficiency of about 23%, while in general case the efficiency stays in the 15% - 20% range [42]. However those numbers are calculated under special test conditions - cell

temperature of 25°C, solar irradiance of 1000W/m2 and Air Mass of 1.5 and would differ in the real world use case. Some factors that change the efficiency are:

- Air temperature
- Dust and dirt
- Inverter and cable losses
- Shadows

High temperatures, shadows and dust could greatly decrease the output power and ideally these factors should be also accounted when doing the estimations. Based on the temperature alone, the power output can drop up to 25% down as can be seen in Figure 1. Other losses, due to wear, dirt, cable losses, etc may vary. PVGIS tool [20] sets a default estimation for such losses to 14%.

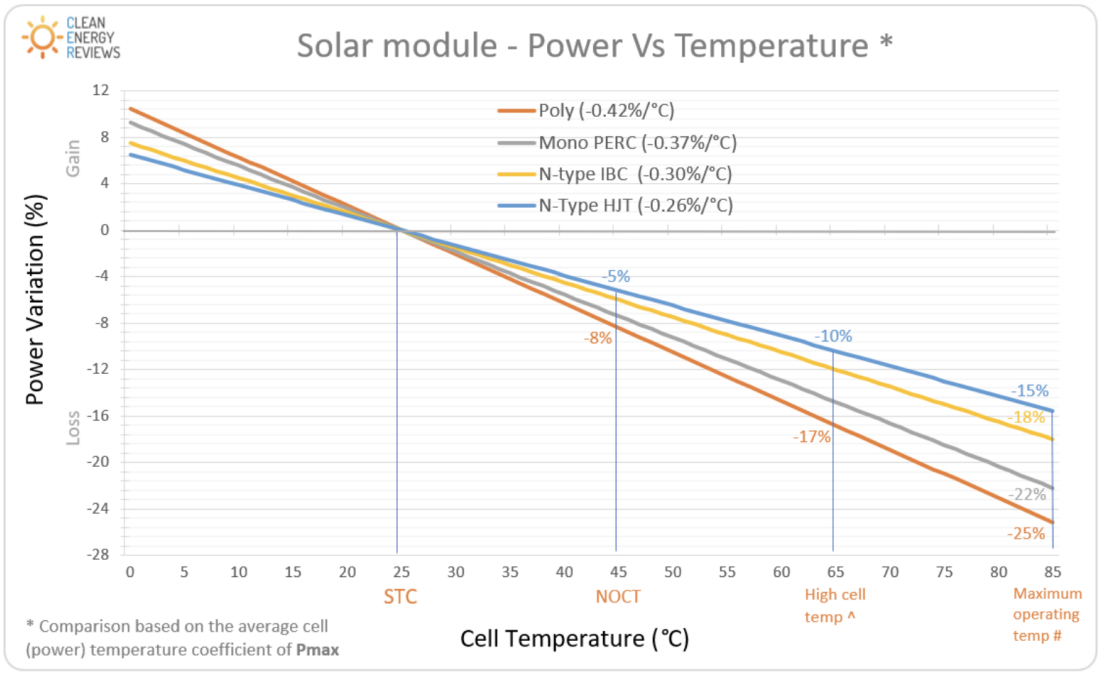


Figure 1. Loss to temperature graph [42]

To simulate these factors when estimating solar potential, the implementation should include configurable parameters for desired PV system efficiency and accounted system losses. While the temperature, amount of sun-hours, and ideally shadows factor should be accounted based on the weather data.

2.3 Solar panels in cities

For the last years, the scientists have been researching the topic of solar potential in the urban context. There have been numerous works related to estimation of solar potential of cities and vehicles pursuing one major goal - accelerating the shift towards renewable energy on the large scale. For example in [33] the authors describe how changing the geometry of the rooftops in Toronto could potentially increase the production of solar power. In 2018 there was a solar potential study conducted in Canberra [30], it described the solar potential of the whole city, most suitable rooftops and potential decrease in CO2 emissions. It was made possible due to APVI Solar Potential Tool [3] developed with Esri ArcGIS similarly to the city of Osnabruck project. Important to note that APVI tool only covers Australia, so it cannot be used for solar potential estimation in Estonia. Similar project were also done in Spain [40], as well as other countries [29]. Even though Estonia is not a very sunny country, the tool developed in scope of this work could give a ground for studies similar to ones described above, allowing to optimally plan construction of new buildings and increase the amount of green energy production where possible on the city or country wide scale.

Combined with other renewable energy sources such as wind and water, the potential benefit for the environment is hard to underestimate. To simplify the transition to such energy source there is a need for various software tools that would help to estimate profitability of various solutions.

2.4 Historical solar power data in Estonia

For better understanding of the topic, as well as, validating future solar potential estimations, it is vital to have data about the solar power production in Estonia. Ideally this data should also feature at least the types of solar panels to be able to find out the installation area and efficiency.

One of data sources is University of Tartu Delta Center which has solar panels installed on the rooftop. The power output of those is monitored as well. Statistics for the past one and a half year, as well as solar panel data sheet were kindly provided by the supervisor of this work. Other than that, the author was not able to find any publicly available historical data from buildings that have solar panels in Estonia.

Considering the country wide statistics, even though the solar power production is growing it is still quite a small fraction of total power produced in the country. According to Estonian Statistikaamet, the amount of solar power produced in 2020 (latest year available) was 122.5 GWh, while the total energy production was 5516 GWh. [14].

3 3D data analysis

The first pillar which made the project possible is access to the 3D data of the cities and counties in Estonia containing the outlines of the buildings provided by Estonian Land Board [1]. The data includes 3 formats: CityGML, Wavefront (obj) and Geodatabse (GDB), available for the whole country with an option to only get the desired city or region. As described on the website, the data for the buildings is created by using a fully automated process that converts the point-cloud dataset combined with the building footprint data into formats mentioned above. The goal of the data analysis is extracting the roof geometries to further compute the solar potential of the buildings.

3.1 Obtaining and storing 3D models of the cities

As mentioned at the beginning of this section, the origin of the data for the city buildings is the point-cloud data representing digital elevation models (DEM). The Estonian Land Board website mentions that it is collected via drones equipped with Light Detection and Ranging (LiDAR) sensors. This point-cloud data can be found on the website [12]. Such approach was also previously described in various research papers. This is a common approach to capture the precise data about the city to support the smart city development [38]. The convertion to CityGML, OBJ and GDB formats unfortunately is not described, aside from the sentence saying that's it's done through an automated process.

3.1.1 Wavefront (OBJ)

OBJ is a simple 3D data format that represents the geometry alone - positions of vertices, normals, texture UV coordinates and faces[19].This format is often used in 3D modelling software. OBJ does not contain any metadata, which would be important when working with buildings. For example, it contains no information about the address or coordinates of the building. Also OBJ by default has no units, unless you write them in a comment, which makes it less convenient to automatically extract them. An example of OBJ file can be seen in Figure 2. There is coordinate system information written in a comment however, there is no building-specific information at all. Because of the drawbacks described this format was discarded.

3.1.2 Geodatabase (GDB)

GDB is a file format that works natively with ArcGIS for managing and storing the data. By definition it's a collection of different geographic datasets [15]. After researching the usability of the data in this format, the author came to the conclusion that it is practically unusable outside of ArcGIS. GDB is a collection of files, rather than as single file, the

```
# Created with FME Version: FME(R) 2021.1.3.0 20211002 - Build 21631 - WIN64
mtllib hooned_lod2-Anija_vald.mtl
# COORDINATE_SYSTEM:  OGC_DEF PROJCS["Estonian Coordinate System of 1997",
GEOGCS["EST97",DATUM["Estonia_1997",SPHEROID["GRS1980",6378137,298.257222101,AUTHORITY["EPSG","7019"]],
AUTHORITY["EPSG","6180"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],
UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AUTHORITY["EPSG","4180"]],
PROJECTION["Lambert_Conformal_Conic_2SP"],PARAMETER["latitude_of_origin",57.5175539305556],
PARAMETER["central_meridian",24],PARAMETER["standard_parallel_1",59.3333333333333],
PARAMETER["standard_parallel_2",58],PARAMETER["false_easting",500000],
PARAMETER["false_northing",6375000],UNIT["metre",1,AUTHORITY["EPSG","9001"]],AUTHORITY["EPSG","3301"]]
# Number of Geometry Coordinates  : 115649
# Number of Texture  Coordinates  : 0
# Number of Normal   Coordinates  : 1
v -7739.534500 10718.890500 -24.277250
v -7739.534500 10718.890500 -27.333550
v -7739.352500 10712.771500 -24.317150
v -7739.352500 10712.771500 -27.333550
v -7733.216500 10719.079500 -23.128950
v -7733.216500 10719.079500 -27.333550
v -7733.020500 10712.494500 -23.171950
v -7733.020500 10712.494500 -27.333550
v -7737.674500 10712.355500 -23.942750
v -7737.674500 10712.355500 -27.333550
v -7739.352500 10712.771500 -23.843750
v -7737.465500 10705.352500 -24.188550
v -7737.465500 10705.352500 -27.333550
v -7742.816500 10705.192500 -24.179750
v -7742.816500 10705.192500 -27.333550
```

Figure 2. OBJ file example

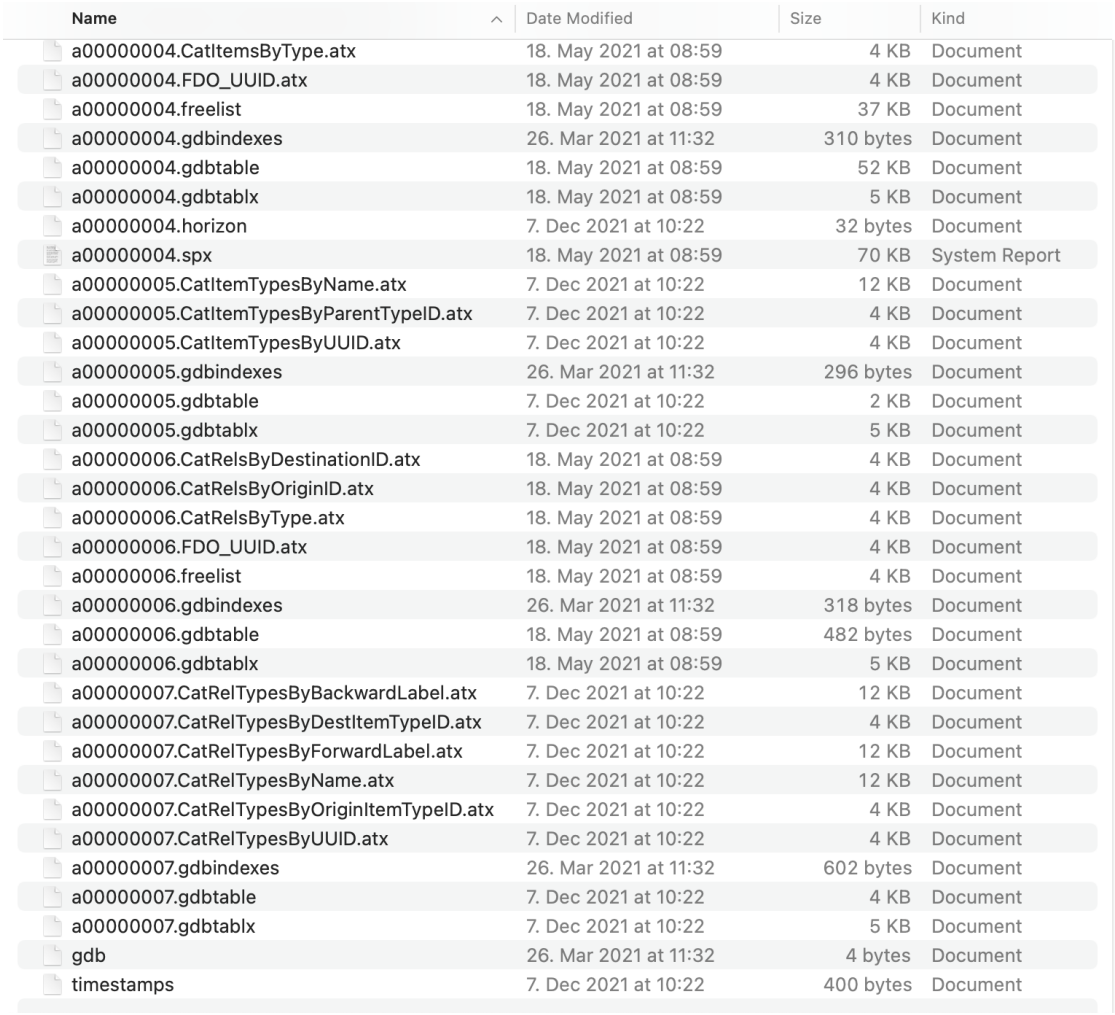
structure can be seen in Figure 3. As the author aimed not to use any paid software for the project, this format was discarded as well.

3.1.3 CityGML

CityGML is a standard specifically designed to store and represent virtual 3D cities, the files themselves can be in GML or XML format. The GML files provided by the Land Board include plenty of metadata for each building in the city, as well as the points for the surfaces and the reference system. Author has determined CityGML as the best format for analysis to be used in the project. It's downside is that there are no tools to easily visualize it, however it is possible to overcome this limitation through convertion to another format. CityGML files from Estonian Land Board are provided in 2 levels of detail (LOD) - LOD1 and LOD2. LOD1 represents simplified buildings only with flat surfaces, while LOD2 buldings can have inclined roofs. CityGML files could also have LOD3 and LOD4 providing more details about exterior and interior, such as windows, doors, etc. An example CityGML file can be seen in Figure 4.

3.2 Roof analysis

Before estimating the solar potential, first step is to understand the CityGML data and extract the information about roofs from it.



Name	Date Modified	Size	Kind
a00000004.CatItemsByType.atx	18. May 2021 at 08:59	4 KB	Document
a00000004.FDO_UUID.atx	18. May 2021 at 08:59	4 KB	Document
a00000004.freelist	18. May 2021 at 08:59	37 KB	Document
a00000004.gdbindexes	26. Mar 2021 at 11:32	310 bytes	Document
a00000004.gdbtable	18. May 2021 at 08:59	52 KB	Document
a00000004.gdbtablx	18. May 2021 at 08:59	5 KB	Document
a00000004.horizon	7. Dec 2021 at 10:22	32 bytes	Document
a00000004.spx	18. May 2021 at 08:59	70 KB	System Report
a00000005.CatItemTypesByName.atx	7. Dec 2021 at 10:22	12 KB	Document
a00000005.CatItemTypesByParentTypeID.atx	7. Dec 2021 at 10:22	4 KB	Document
a00000005.CatItemTypesByUUID.atx	7. Dec 2021 at 10:22	4 KB	Document
a00000005.gdbindexes	26. Mar 2021 at 11:32	296 bytes	Document
a00000005.gdbtable	7. Dec 2021 at 10:22	2 KB	Document
a00000005.gdbtablx	7. Dec 2021 at 10:22	5 KB	Document
a00000006.CatRelsByDestinationID.atx	18. May 2021 at 08:59	4 KB	Document
a00000006.CatRelsByOriginID.atx	18. May 2021 at 08:59	4 KB	Document
a00000006.CatRelsByType.atx	18. May 2021 at 08:59	4 KB	Document
a00000006.FDO_UUID.atx	18. May 2021 at 08:59	4 KB	Document
a00000006.freelist	18. May 2021 at 08:59	4 KB	Document
a00000006.gdbindexes	26. Mar 2021 at 11:32	318 bytes	Document
a00000006.gdbtable	18. May 2021 at 08:59	482 bytes	Document
a00000006.gdbtablx	18. May 2021 at 08:59	5 KB	Document
a00000007.CatRelTypesByBackwardLabel.atx	7. Dec 2021 at 10:22	12 KB	Document
a00000007.CatRelTypesByDestItemTypeID.atx	7. Dec 2021 at 10:22	4 KB	Document
a00000007.CatRelTypesByForwardLabel.atx	7. Dec 2021 at 10:22	12 KB	Document
a00000007.CatRelTypesByName.atx	7. Dec 2021 at 10:22	12 KB	Document
a00000007.CatRelTypesByOriginItemTypeID.atx	7. Dec 2021 at 10:22	4 KB	Document
a00000007.CatRelTypesByUUID.atx	7. Dec 2021 at 10:22	4 KB	Document
a00000007.gdbindexes	26. Mar 2021 at 11:32	602 bytes	Document
a00000007.gdbtable	7. Dec 2021 at 10:22	4 KB	Document
a00000007.gdbtablx	7. Dec 2021 at 10:22	5 KB	Document
gdb	26. Mar 2021 at 11:32	4 bytes	Document
timestamps	7. Dec 2021 at 10:22	400 bytes	Document

Figure 3. GDB format structure example

3.2.1 CityGML roof surfaces extraction

First, it is important to understand how the CityGML files for Estonian cities are structured. The root tag is **core:CityModel** followed by the XML namespace definitions for CityGML, for example **core** points to "http://www.opengis.net/citygml/2.0". Root tag itself only provides the the bounding region for the city and the list of **core:cityObjectMember** tags which represent individual buildings.

From the Figure 7 it is possible to see that in CityGML for each building there is indeed some metadata related to it, as well as the most important **bldg:lod2MultiSurface** tag, which includes all the surfaces that the building has. Each surface has its points separately provided in XYZ coordinates and in EPSG:3301 reference system[13], see

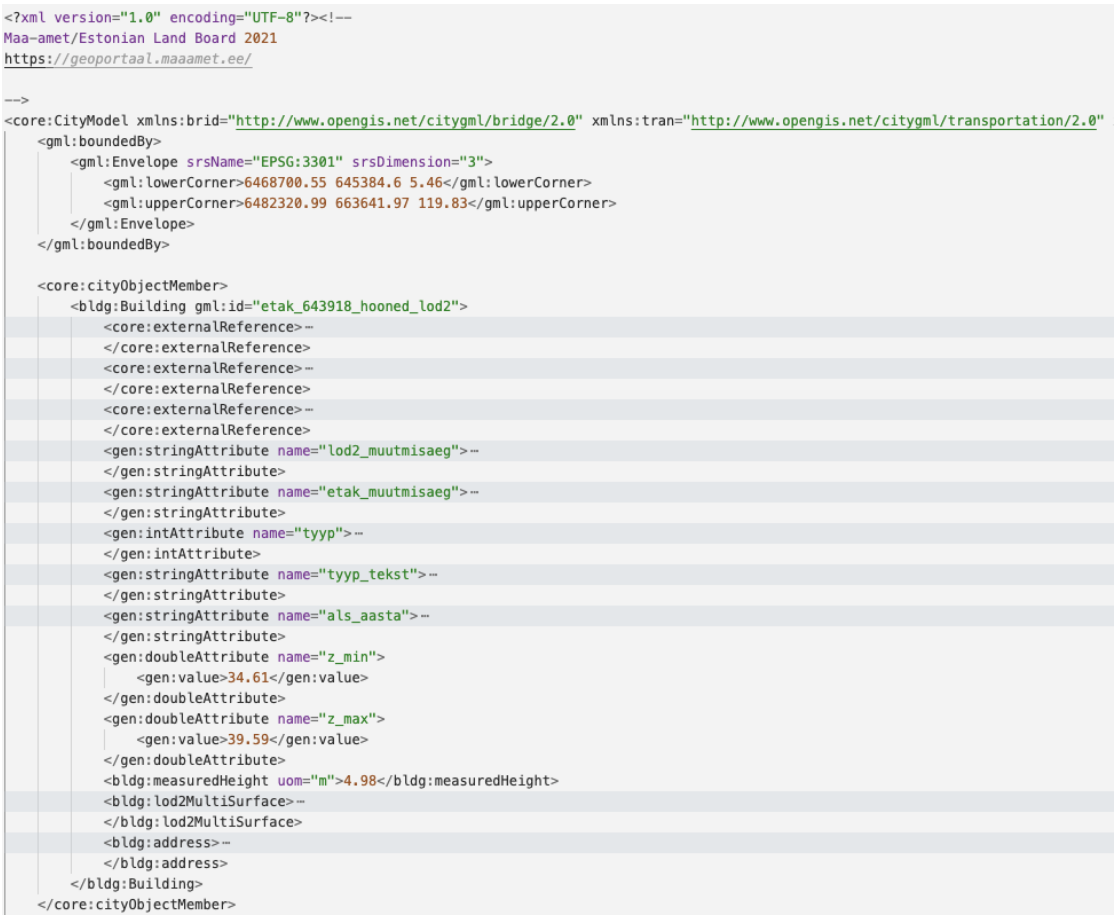


Figure 4. CityGML file example

Figure 8. Unfortunately, at the moment in the dataset provided it is not specified which of the surfaces are the roof of the building. To overcome this issue a roof detection algorithm had to be implemented. With the current implementation, to be considered a roof, a surface has to fulfill two properties:

- Not be a floor surface - a floor surface has all its points located on the min(z) level. Min(z) level in this case is the bottom most point of the building.
- Not be a wall surface. A wall surface has 90°tilt.

3.2.2 Roof area

Once it is possible to detect which surface belongs to the roof, it is possible to calculate the roof area. As mentioned previously, a roof might consist of multiple surfaces which

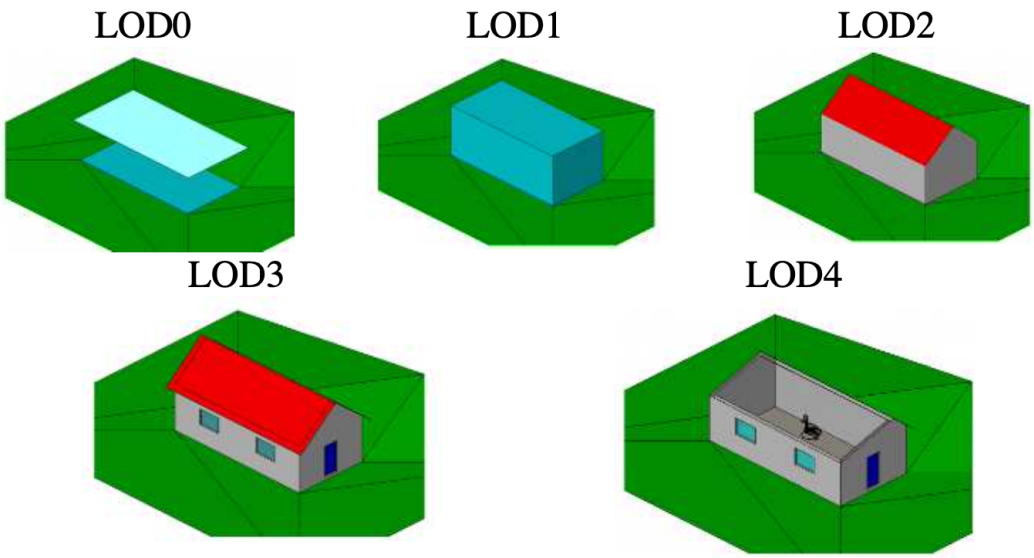


Figure 5. CityGML LODs [39]

```
<gml:boundedBy>
    <gml:Envelope srsName="EPSG:3301" srsDimension="3">
        <gml:lowerCorner>6468700.55 645384.6 5.46</gml:lowerCorner>
        <gml:upperCorner>6482320.99 663641.97 119.83</gml:upperCorner>
    </gml:Envelope>
</gml:boundedBy>
```

Figure 6. cityModel boundedBy tag content example

should be accounted separately, with their own area, tilt and azimuth, to have more precise solar potential estimation. Surface area calculation comes down to determining the area of a 3D polygon. Mathematical explanation and implementation for that was found in an article [5] and used in the pipeline implementation.

3.2.3 Tilt and azimuth

Tilt and azimuth are the characteristics of the surface that are often used for estimating solar potential. Tilt or inclination angle can be defined as an angle between the surface and the horizontal axis. Azimuth or orientation is an angle of the surface relative to the direction of the South. In case of local coordinates azimuth will actually be an angle relative to the Y axis. A visual explanation of these concepts can be seen in Figure 9.

```
<core:cityObjectMember>
    <bldg:Building gml:id="etak_643918_hooned_lod2">
        <core:externalReference>
            <core:informationSystem>https://geoportaal.maaamet.ee/est/Andmed-ja-kaardid/Topograafilised-andmed/Eesti-topograafiline-andmekogu-p79.html</cor
            <core:externalObject>
                <core:name>643918</core:name>
            </core:externalObject>
        </core:externalReference>
        <core:externalReference>
            <core:informationSystem>https://xgis.maaamet.ee/adsavalik/</core:informationSystem>
            <core:externalObject>
                <core:name>ME00749793</core:name>
            </core:externalObject>
        </core:externalReference>
        <core:externalReference>
            <core:informationSystem>https://www.ehr.ee/</core:informationSystem>
            <core:externalObject>
                <core:name>104015773</core:name>
            </core:externalObject>
        </core:externalReference>
        <gen:stringAttribute name="lod2_muutmisaeg">
            <gen:value>2021-03-06</gen:value>
        </gen:stringAttribute>
        <gen:stringAttribute name="etak_muutmisaeg">
            <gen:value>2019-06-24</gen:value>
        </gen:stringAttribute>
        <gen:intAttribute name="tyyp">
            <gen:value>20</gen:value>
        </gen:intAttribute>
        <gen:stringAttribute name="tyyp_tekst">
            <gen:value>Kõrval- või tootmishoone</gen:value>
        </gen:stringAttribute>
        <gen:stringAttribute name="als_aasta">
            <gen:value>2020</gen:value>
        </gen:stringAttribute>
        <gen:doubleAttribute name="z_min">
            <gen:value>34.61</gen:value>
        </gen:doubleAttribute>
        <gen:doubleAttribute name="z_max">
            <gen:value>39.59</gen:value>
        </gen:doubleAttribute>
        <bldg:measuredHeight uom="m">4.98</bldg:measuredHeight>
        <bldg:lod2MultiSurface>⋯
        </bldg:lod2MultiSurface>
        <bldg:address>⋯
        </bldg:address>
    </bldg:Building>
</core:cityObjectMember>
```

Figure 7. cityObjectMember tag content example

3.2.4 Latitude and longitude

To make solar potential estimations more precise, it's vital to know an approximate location. The most common way to store locations is in latitude and longitude. Since the approximate value is enough, author has decided to take the first point of the surface and convert it's coordinates from EPSG:3301 to WGS 84 (latitude and longitude). Having this data allows to use more accurate and relevant weather data.

Figure 8. EPSG:3301 zone [13]

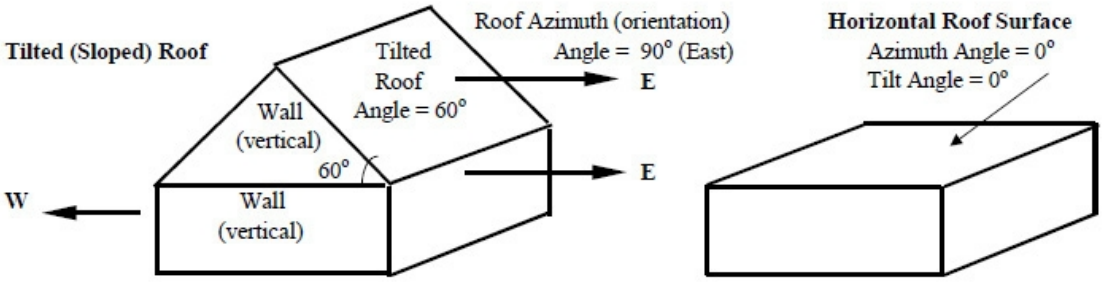


Figure 9. Solar panel tilt and azimuth [43]

4 Solar potential estimation

As described in [34], there are two main approaches to predict solar power output: statistical and physical. While first one focuses on historical data, second one leverages the weather data. Considering there was very little historical data, second approach makes the most sense to utilize in the project. Developing such a prediction system would highly increase the scope of this work, therefor the author has decided to evaluate if it is possible to use an existing API, library or ML model. Then, obtained results can be compared with the historical data from University of Tartu Delta Center.

4.1 GIS software tools

When considering solar irradiation or potential, there are several GIS (Geographic Information System) software tools capable of estimating it based on 3D data. In [32] the author compares 5 different GIS tools: GRASS GIS, ArcGIS, SimStadt,CitySim and Ladybug in terms of input data requirements, usability, and accuracy of the estimated solar irradiation results. This subsection of the thesis is largely based on that work. GIS tools include plenty of functionality aside from solar potential estimations, therefore their usage in the developed tool would lead to increased overhead of the final product. Yet, it is important to evaluate them as they are used in many of the solar potential case studies worldwide, for example in [37], [36] and [44].

4.1.1 GRASS GIS

GRASS GIS is a free GIS software built for vector and raster geospatial data management, geoprocessing, spatial modelling and visualization [16]. It provides the **r.sun** module that computes direct, diffuse and reflection solar irradiation based on provided Digital Elevation Map (DEM). GRASS GIS can be used via UI (User Interface) or CLI, so technically it is possible to integrate it in the pipeline. Important to note that this module does not account for weather data and has no means to import it at the moment of writing this thesis, as well as no functionality related to simulating solar panels. Those limitations directly impact the accuracy, as well as integration time, therefore the author decided not to use this tool.

4.1.2 ArcGIS

ArcGIS [4] offers a big amount of possibilities when it comes to geospatial analysis, visualization and more through online and desktop tools. Estimating solar potential can be done in ArcGIS Pro with Spatial Analyst extension. Here important to note that ArcGIS Pro requires at least 32GB of disk space, as well as available only under paid

license. Such requirements would prevent the developed solution from being lightweight and free, so this tool was discarded as it is against the initially defined goals.

4.1.3 SimStadt

SimStadt is an open-source, Java-based GIS desktop software for urban simulations, analysis and energy potential estimation. SimStand has a built-in workflow to estimate solar potential that requires LOD1 CityGML file and weather data in tmy format as inputs. As a result the workflow produces the output describing the power it is possible to produce, number of usable surfaces and other information. The workflow does not support some solar panel related parameters, such as system loss, as well as the software can only be used through desktop UI, which is a bit limiting, but yet still is a good option to consider.

4.1.4 CitySim

CitySim [9] is a free and open-source for urban energy planning. CitySim is capable of working with CityGML files through the desktop UI. To work with CitySim it is required to provide only 3D data and climate data, while then it is capable to output the solar irradiance data. Partially, the interactions with CitySim can be automated, however CityGML file import can only be done through UI, which is a limitation. During the initial research this tool was overlooked, therefore it was not evaluated properly before the development. In future work this could be done, as CitySim could potentially be a good fit.

4.1.5 Ladybug

Ladybug [17] is a collection of free and open-source applications for environmental design and education. The tools are free but they are usually run in Grasshopper, a visual programming language and environment that runs within the(commercial) software Rhinoceros 3D [32]. Rhinoceros 3D is a CAD software, which requires inputs in DWG format. Ladybug then provides the functionality to estimate solar irradiance of the buildings in the 3D scene. Even though it is possible to use, integrating it seems to require additional work compared to other soltuions, therefore it was discarded for now.

4.2 Libraries and APIs

In this subsection the author describes several libraries or APIs that could be used for handling solar irradiance or power estimations. The target was to find the one that is the easiest to integrate programatically, while having good quality of the results not to extend the scope of the project too much. Compared to GIS tools a library or an API is a

better solution, as it offers programmability by default and should be more lightweight. Therefore, if a suitable solution of such kind exists, it will be preferred.

4.2.1 Solpy

Solpy is a Python library designed to model solar system performance [25]. It was used in the Solar3DCity project and it features various ways to calculate solar irradiance and PV system output. This library does not provide any utilities to work with the weather data, so that has to be handled separately. Same as the Solar3DCity, this project is around 7 years old and it is not maintained anymore. Given that the author's focus was to use modern technologies the idea to use this library was discarded.

4.2.2 Pvlib

Pvlib Python is an open-source tool that provides various functions and classes for simulating the performance of PV energy systems [35], similarly to Solpy. The amount of functionality offered by this library is huge, it is even possible to simulate the power produced by a certain type of solar panel, or alternatively just calculate surface irradiance based on surface attributes and weather data. The weather data could be custom, however the library has a utility to fetch the weather data directly from PVGIS API. The possibilities that are open with this package are very extensive and based on the research it seems like the best Python solution for simulating PV systems.

The only issue with Pvlib is that it is a bit difficult in usage when trying to integrate weather data and solar irradiance predictions, so the author has decided to look for simpler alternatives, while keeping this option for the future work. That difficulty itself is in building the weather prediction model based on the statistical data (2005-2020) provided by PVGIS API and aggregating it by month and year.

4.2.3 Solcast API

Solcast provides a scalable API for solar irradiance and PV power estimations globally [24]. Solcast provides high availability and plenty of features, such as historical and forecast data about weather, irradiance, rooftop solar power and more. However it comes with quite a high price of 370 USD per month + 20 USD for each additional set of 10,000 API requests per month. Given that the estimation pipeline would process tens of thousands of buildings, this API is very cost ineffective to use.

4.2.4 PVGIS API

PVGIS API, which is based on the PVGIS web tool [20] provides an extensive amount of functionality for estimating solar irradiance, PV system output and weather in a specified location. Description of the tool mentions that PVGIS, maintained by the EU Science

Hub, uses high-quality and high-spatial and temporal resolution data of solar radiation obtained from satellite images, as well as ambient temperature and wind speed from climate reanalysis models.

The endpoint which allows to estimate PV system output with just a few parameters allows to fulfil the requirements for the tool the author was looking for. The API allows to pass surface parameters such as tilt and azimuth, as well as PV system parameters like system loss. A big advantage of PVGIS API is that accounts for weather automatically, based on provided location, which simplifies the integration. Even though it would be better to do the estimations locally rather then over the network, in the prototype stage it is fine to sacrifice the efficiency for the sake of a working solution.

Considering the points above, author has decided to select this tool for solar potential estimations. In the future, the usage of Pvlib or GIS software should be more thoroughly explored as it seems promising. Also, important to mention that the PVGIS API is free to use.

To check the accuracy of the estimations, author first decided to use the PVGIS app and compare the yearly results in the app to historical Delta Center data stored in InfluxDB. Delta Center has 732 solar panels with each one having an area of 1.937 m2 and efficiency up to 17.29%. Resulting covered area then is around 1417.88 m2. Instead of area, PVGIS expects the peak power (kWp) which can be calculated as (kWp) = 1 kW/m2 * area * efficiency. So the approximate peak power for current Delta Center PV system should be 241 kWp.

Final comparison can be seen in Table 1. It is possible to see that the values are quite close. The difference could be due to different system loss or various other factors.

Table 1. Historical Delta values vs PVGIS estimations

Type	**Area**	**Efficiency**	**System loss**	**Yearly output**
Historical	1417.88 m2	17%	unknown	192247.183 kWh
PVGIS	1417.88 m2	17%	20%	204219.97 kWh

One more discovered way came from Eesti Energia solar panel solutions offered in Estonia [11]. They offer several propositions which feature peak power of the system and estimated yearly power production. Their comparison to numbers from PVGIS can be seen in Table 2, system loss was not accounted.

As it can be seen, those numbers are quite close as well, especially considering that system loss was not accounted. Eesti Energia, on the other hand, probably included historical system loss. Overall, it is possible to see that the difference in PVGIS estimation and the historical data is in range of the loss deviation. Therefore, when estimating the solar potential for cities, the historical loss could be obtained with the help from solar panel providers for more precise estimations.

Table 2. Eesti Energia estimations vs PVGIS estimations

Peak power	Eesti Energia estimation	PVGIS estimation
5.25 kWp	4990 kWh	5560 kWh
7.5 kWp	7125 kWh	7944 kWh
9.75 kWp	9215 kWh	9800 kWh
12 kWp	11500 kWh	12710 kWh

5 Processing pipeline implementation

This section describes how the steps mentioned in sections 3 and 4 were implemented. To briefly remind, the pipeline should process one or more CityGML files, extract and analyze the roof surfaces of the buildings in the dataset, estimate solar potential for those surfaces and export the estimations.

5.1 Setting up the project

For the processing pipeline implementation author has decided to mainly use Python programming language. Python is often used for scripting, it is easy to use and it has very good support for working with files. Also some code could be reused from Solar3DCity project. The processing pipeline is a CLI tool that accepts different parameters for tuning the calculations.

There were also two external Python modules used: Numpy [18] and Pyproj [21]. First one was used for mathematical operations related to roof area calculations, while the second one was needed to transform coordinates to different cartographic coordinate systems.To have consistent code style, author has used autopep8 [6] package which automatically formats the code according to the standards and set rules.

Based on the previous sections and usability considerations, author has defined the following requirements for the pipeline:

- Should be able to process one or more CityGML files stored locally
- Should have configurable parameters related to solar potential estimations
- Should run end to end, no manual steps in the middle of the processing

Those requirements resulted in the chosen pipeline architecture in Figure 10 and parameters in Table 3. Default values for parameters related to solar potential estimations were based on the domain background described in section 2.

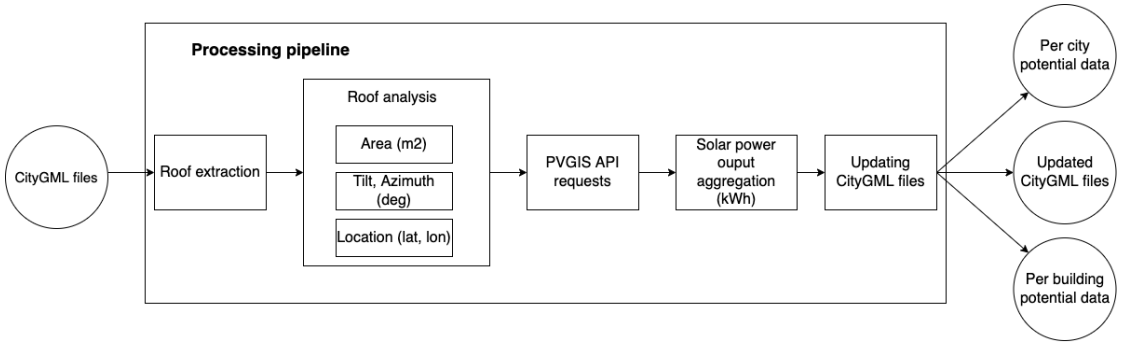


Figure 10. Processing pipeline

Table 3. Pipeline parameters

Name	Description	Default value
lod	LOD of the buildings (LOD1 and LOD2 supported)	2
filename	a specific file to process (process all if not specified)	none
pv-efficiency	efficiency of the PV system	0.20
pv-loss	losses in cables, power inverters, dirt, etc	0.14
roof-coverage	fraction of a roof that can be covered with solar panels	0.90

5.2 Implementing roof extraction

When the pipeline starts it reads the contents of the data directory with CityGML files and sequentially processes one or all files there. To open the CityGML files the built-in **xml** Python module can be used. With its help it's possible to get the root XML tag of the file as a Python object and begin querying for needed information. The most important information to obtain for each building is its identificator and a list of the surfaces. First one is used to provide per building outputs which can later be tracked back to actual buildings in the dataset. Based on the list of surfaces, the pipeline analyzes which ones belong to the roof, then calculates their area, tilt, azimuth and extracts their longitude and longitude. Those values are stored in an in-memory dictionary, which is later populated with solar potential data. Roof areas are also written back to CityGML files.

Functions for surface tilt and azimuth calculations were taken from the Solar3DCity project. Calculating both is based on the surface normal and its projections. An angle between the projection on the horizontal plane and the negative X (south) direction is considered an azimuth. Similarly, tilt can be found through an angle between normal's Z (up) component and distance between normal's X and Y components.

5.3 Implementing the solar potential estimation

As the PVGIS API was chosen for doing the solar potential estimations, the author had to integrate it with the 3D data processing part of the pipeline. Since the API has the rate limit of 30 requests per second, a straightforward solution is to send requests in batches with a pause after each batch. The important part is that once the request resolves, the response with solar power data has to be mapped back to the building id where the roof surface is located. After trying a few different ways to implement the logic mentioned above in Python, the author has decided to delegate this part of the pipeline to Node.js, because JavaScript as a language makes working with asynchronous requests much simpler than Python. To integrate these two different technologies, the author has decided to exchange data using temporary files - Python pipeline stores the data needed to make the requests in a JSON file. Then, Python pipeline starts the Node.js script, which reads the data from JSON file, makes the requests, and stores the results in another JSON file which is then read back by Python pipeline. While this might not seem like an efficient approach, it does not cause any performance issues, temporary files require not more than 100 MB of memory and the time used for reading/writing is very small compared to overall processing duration.

Once all the requests are done, the results are stored in the attribute map created by the first part of the pipeline. So now, in addition to roof data, for each building id there is also a list of solar power predictions for each surface. Finally, some data aggregation is done to have monthly and yearly estimations per building and per city. Per building estimations are needed to accurately visualize the solar potential on the map, highlighting the buildings that are the most useful. Per city estimations are needed to give an overview of the total city solar potential.

5.4 Pipeline output

After the processing is done, the pipeline outputs an updated CityGML file and two most important files: **city-pv.json** and **city-attributes.json.** First one contains aggregated yearly and monthly solar potential for the whole processed file (city). It is meant for quick analytics and as a light-weight source of information that can be embedded into applications. Second file containes very detailed information about every building in the city including each roofs area, tilt, orientation as well as solar power estimations and is meant for more in-depth analytics or custom integrations that could be done in the future. The link to examples of those files can be found in Appendix I, as well as the source code of the project.

6 Visualization

When looking into different ways to visualize the results, author has considered different options. While the processing pipeline is meant to be run very rarely, the visualization should be available at any time. The author has decided to create a web app, because it is a universal solution that can be accessed from any device. The app should display the buildings in 3D alongside the solar power statistics for the city.

6.1 Visualizing CityGML file

Even though CityGML is a convenient and descriptive way to store 3D data of the cities, it cannot be visualized on the web directly [27]. To overcome this issue, the files need to be converted to another format beforehand.

6.1.1 OBJ

In the Solar3DCity project, which is frequently mentioned in this work, the author has used his custom software to convert processed CityGML files to OBJ format. While this approach is valid, the software used by the author of Solar3DCity is deprecated by himself in favor of using CityJSON. There are other solutions available, which however, would cause the loss of attributes, simply because OBJ file does not store any metadata. For example, if one would want to paint the building in different color based on the roof area stored in CityGML file, it would not be possible.

6.1.2 CityJSON

CityJSON is a relatively new alternative to CityGML, which is being actively developed by the 3D Geoinformation research group at TU Delft. CityJSON provides more compact files compared to CityGML and more tooling for different use cases. The research group claims that the CityGML files can be easily converted with the help of the **citygml-tools** package provided by them and then viewed in the web viewer. Because the web viewer is open-sourced, it would be relatively easy to create a similar web application with enhanced functionality for solar potential visualization.

After trying out the convertion to CityJSON, the author noticed that the buildings are not displayed at all, most likely due to a silent format convertion error. As this approach is quite new, there could be various issues at this stage, therefore author has decided to evaluate using 3D tiles.

CityJSON shows great potential and seems to be the future for storing 3D city data. The author belives that the tooling built around it will make it a lot more convenient to use than CityGML.

6.1.3 3D tiles and Cesium

One common way to visualize CityGML is through converting it to 3D tiles format. 3D Tiles is designed for streaming and rendering massive 3D geospatial content such as Photogrammetry, 3D Buildings, BIM/CAD, Instanced Features, and Point Clouds [2]. It has a hierarchical structure and allows for optimized rendering while still preserving building-related metadata that can be accessed in the application.

There are not many ways to convert CityGML to 3D tiles described. The one featured the most often is by using the FME Workbench software, which is a desktop application. It has different pricing categories and is available for free for students, however the license for it is quite costly. Additionally, it's not possible to use it from CLI, therefore it would add a manual step to the processing pipeline.

Ideally, the convertion to 3D tiles should be done within the processing pipeline. The author was able to find one package called **city-gml-to-3d-tiles** [8] written in JavaScript that does the job. Same as with PVGIS API requests, the Node.js script can be called from Python code. The output can then be visualized on a map by using Cesium JavaScript library.

Cesium is the open platform for software applications designed working with 3D data[7]. It offers various libraries and APIs that make it easier to create 3D maps, which one can enrich with custom 3D data such as buildings. Client side functionality can be used via CesiumJS library. Adding custom 3D tiles data can be done very easily, as can be seen in Figure 11. Added buildings are also interactable and stylable, which are two very useful features. First one would allow to get some building specific information on click, while the second one would allow to color the buildings based on their solar potential.

```
var viewer = new Cesium.Viewer('cesiumContainer');
var tileset = viewer.scene.primitives.add(new Cesium.Cesium3DTileset({
    url : '/path/to/3d/tileset'
}));
```

Figure 11. Visualizing 3D tiles via CesiumJS [31]

Cesium also offers Cesium ION service which is free for non-commercial purposes. It allows to upload the 3D data in various formats (including CityGML) which can later be easily used in a Cesium application. Unfortunately, there were unknown errors without answers when trying to upload CityGML files produced by the Land Board there, therefore the previously described step of converting to 3D tiles was neccessary.

Therefore, the chosen approach was to use the **city-gml-to-3d-tiles** library to convert the pre-processed CityGML files to 3D tiles before visualizing them with CesiumJS inside of a web application. Out of the box it did not support EPSG:3301 reference system, so the author made a fork of the repository and introduced a small fix there. The link to the fork can be found in Appendix I. Once the files are converted, Cesium allows to create a 3D map in a few lines of code and load the 3D data directly onto the map. An example of a map with 3D tiles data built with Cesium can be seen in Figure 12.

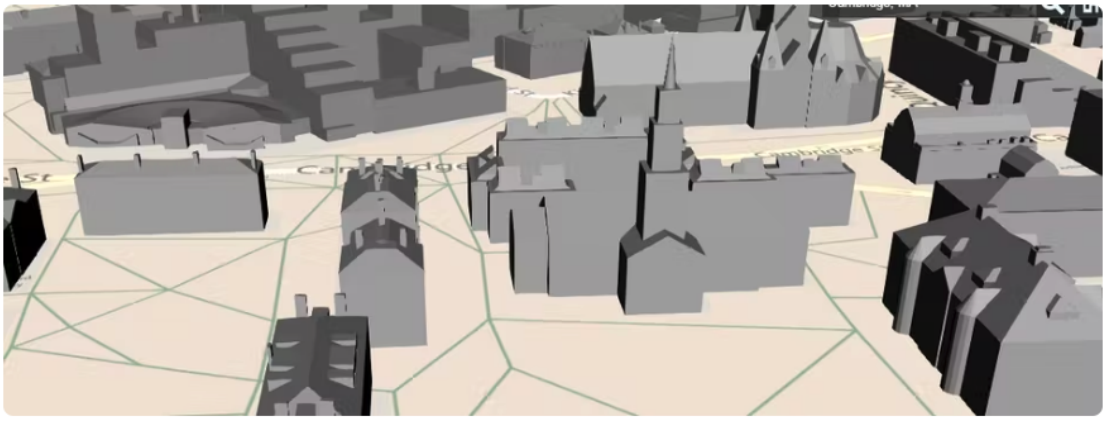


Figure 12. 3D buildings in Cambridge [31]

And the final processing pipeline can be seen below in Figure 13. Also two more parameters were added to the pipeline that can be seen in Table 4. 2D map optimization is needed to support flat maps, by default the 3D data has some minimum and maximum Z value for each building. Output format currently supports only one value, however since the project could be improved in the future this gives a foundation for other potential formats. Alternatively, conversion might not be needed at all if the user only needs the estimations data. Also it is important to specify higher RAM limit for Node.js which defaults to 512MB. Experiments have shown that at least 5GB of RAM is needed to convert Tartu CityGML to 3D tiles when using the **city-gml-to-3d-tiles** library.

Table 4. Pipeline visualization related parameters

Name	**Description**	**Default value**
optimize-2d-map	optimize the buildings to have the floor at z = 0	True
output-format	output type to convert CityGML files to	tiles
node-ram-limit	RAM limit for Node.js process	5500

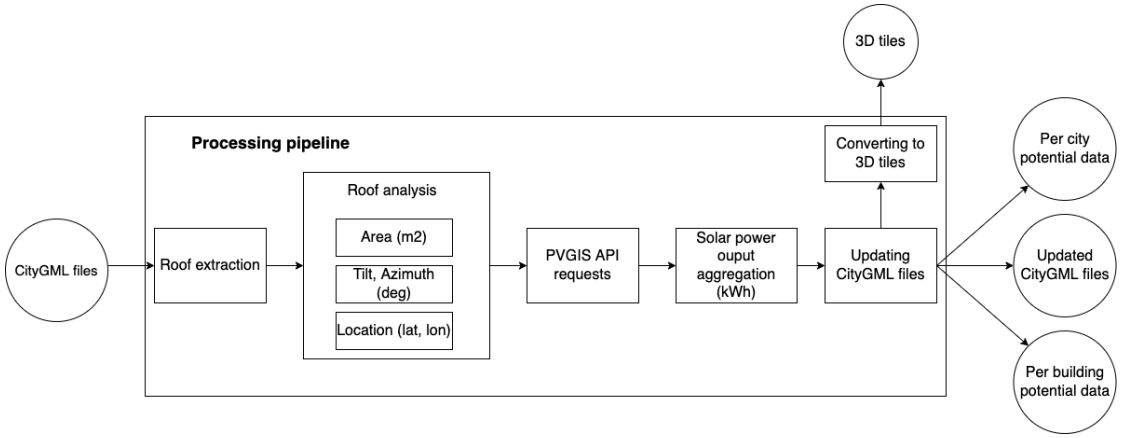


Figure 13. Final processing pipeline

6.2 Developing a web app

In the modern era there is a wide choice of frameworks and libraries to use for developing web applications. They mainly share the same concepts and differ in the implementation and various details. Considering the amount of code to write for the application was expected to be small, the author has decided to use the library he is the most proficient with - React. React is a JavaScript/TypeScript library which helps to build declarative, component based user interfaces [22].

Instead of using Webpack, which is the most common tool for building, the author has decided to use Vite [26]. Vite is extremely easy to configure, it is loads much faster during local development and has a lot of flexibility when in comes to production builds. It also supports various frontend frameworks out of the box, such as: React, Vue, Angular, etc.

When the application starts, it reads the 3D tiles data provided from the pipeline stored locally. Then, the application has a main page which allows to select the city to visualize. This was done to allow the application to scale easier and provide information about multiple cities in one location. Even though this approach is not ideal, it allows to add new cities quite easily, because it is only needed to process desired new cities and then restart the web app. Considering the expected traffic is very low, as the application is not meant to be for public use yet, this approach was considered sufficient for now, while in the future it would be better to decouple the web application and the data, removing the need to restart the server.

Once the city is selected in the application, there are two main components - **Map** and **Dashboard**. Map is located on the left side of the screen and it contains the Cesium map supplied with the 3D tiles dataset of the chosen city. Dashboard is located on the right side of the screen and it shows various statistics related to solar power production

for chosen city, it takes the data from the JSON file that is produced by the processing pipeline. An example of the developed app can be seen in Figure 14, dashboard and map components separately can be seen in Figures 15 ad 16 respectively.

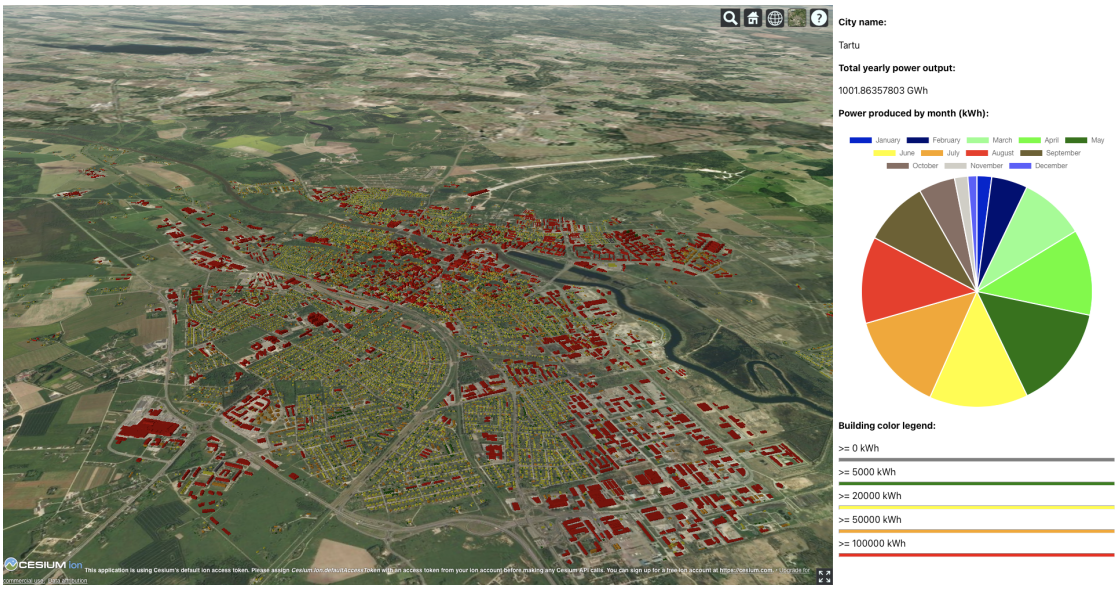


Figure 14. Tartu solar potential visualization example

Because 3D tiles format does not allow to give different colors to surfaces of one building, instead the buildings have different colors based on their overall solar potential. In the future one could attempt to improve the **citygml-to-3dtiles** library to include this feature. Alternatively, other visualization options could be considered. When a building is clicked, it is possible to see more details which are added during the processing, see Figure 16 for the reference. First seven fields are inferred from the CityGML data, while the fields starting from **area** are the custom ones added during the processing. Their explanations can be seen in Table 5. The units are: m2 for area fields and kWh for power fields.

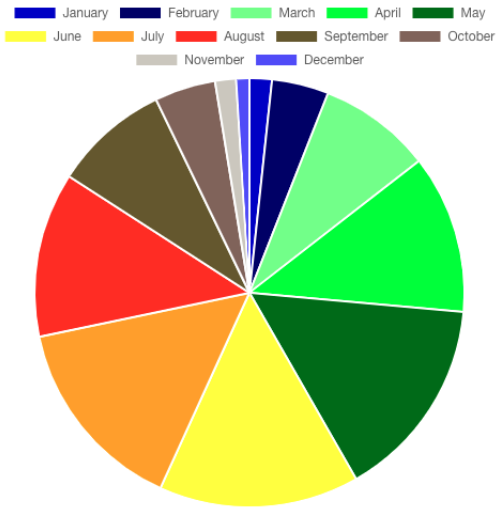
From file:

tartu.gml

Total yearly power output:

886.3223 GWh

Power produced by month (kWh):



Building color legend:

>= 0 kWh

>= 5000 kWh

>= 20000 kWh

>= 50000 kWh

>= 100000 kWh

Figure 15. Dashboard component

Table 5. Building features

Name	Description
area	total area of detected roof surfaces
north-area	area of roof surfaces facing North
south-area	area of roof surfaces facing South
west-area	area of roof surfaces facing West
east-area	area of roof surfaces facing East
flat-area	area of roof surfaces with tilt <= 10°
etak_id	ETAK ID of the building, not inferred automatically
power	total solar power that can be generated from the rooftop of this building
north-power	solar power that can be generated from the rooftop surfaces facing North
south-power	solar power that can be generated from the rooftop surfaces facing South
west-power	solar power that can be generated from the rooftop surfaces facing West
east-power	solar power that can be generated from the rooftop surfaces facing East
optimized-power	solar power that can be generated from the flat surfaces (optimized panels)

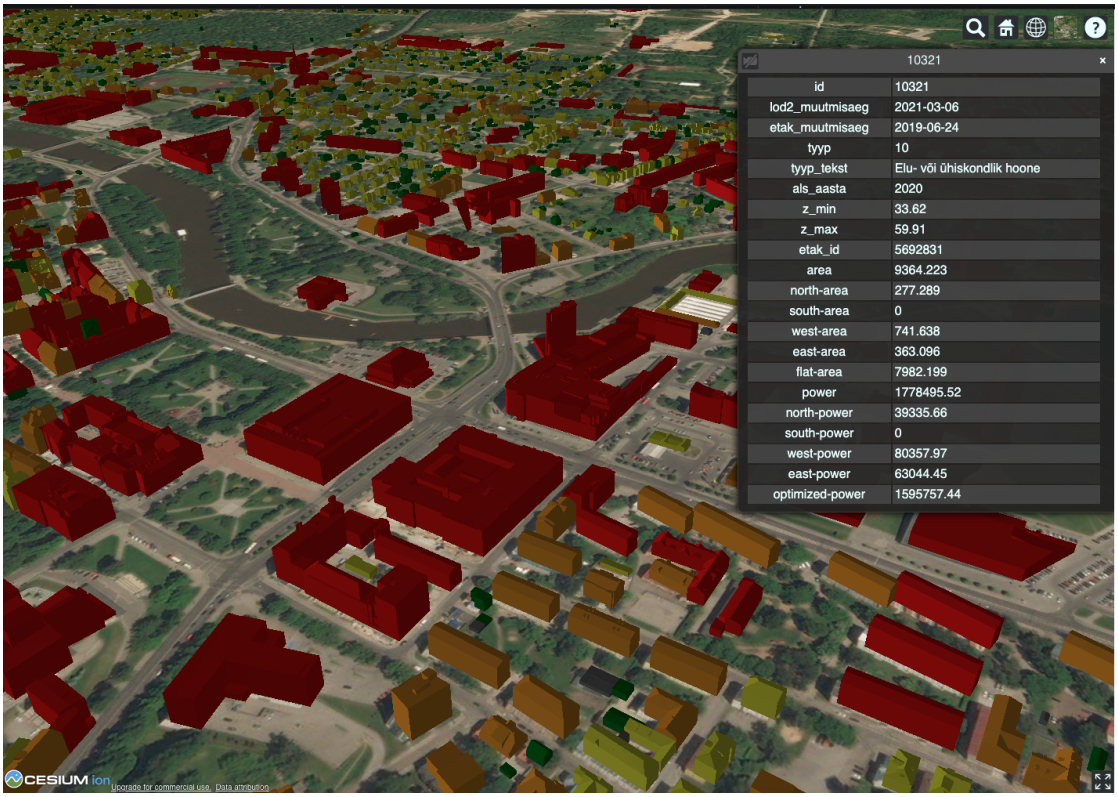


Figure 16. Map component when a building is clicked

7 Deployment and validation

To be able to demonstrate the visualized solar potential of the cities easily, the application has to be deployed to a server, so that it can be globally accessible. In the current project iteration, the outputs from the processing pipeline and the web app need to be on the same machine in order for web app to access the data. Hence, the author has decided to use one virtual machine for both processing and a web server. Later, it would be more hardware efficient to have processing pipeline and web server decoupled due to those having different hardware requirements. While processing pipeline needs more RAM, web server can run on quite a small virtual machine.

Though there are lots of cloud providers with their own pros and cons (AWS, Google Cloud Platform, Azure and many more), the author has used OpenStack because it is provided as a free service for students by the University of Tartu HPC center. At this stage it was not so important to consider different options, because the virtual machine (VM) is only needed for demonstrating the application. The requirements for the VM are explained in the next subsection. Since OpenStack could be a paid service, in the future various cloud providers, as well as deployment options (e.g cloud function for the pipeline) could be considered.

Once the application is accessible online it is also much easier to evaluate it and collect feedback.

7.1 Setting up the virtual machine

OpenStack provides a vast variety of VM configurations available with support of different operating systems. Experimentally it was discovered that the machines with 2GB, 4GB and even 8GB cannot handle the whole processing pipeline and the process gets killed due to too high RAM consumption (local development was done with 16GB RAM). Therefore, the author has chosen the least powerful machine with 16GB of RAM which is m2.xsmall. It has 4 VCPU, 16GB RAM and 20GB disk which is more than enough for processing of the city of Tartu. In terms of CPU there are no strict requirements as the pipeline is most composed of sequentially processed blocks. Regarding the disk size, around 1GB of memory is needed for all the outputs and temporary files when processing the Tartu city dataset, original CityGML file is around 200MB. Therefore, the amount of disk available on this machine is likely enough to store the results for most parts of Estonia. Ubuntu 20.04 was used as the OS for the machine, because it is a stable general purpose Linux distribution which is very commonly used on both desktop and server.

To configure the machine fast and install all needed dependencies author has written a separate script, that can be seen in Figure 17. Alternatively, the author has also added the option to build and run both the pipeline and web application via Docker. This approach is described in subsection 7.4.

scripts > linux > full-setup.sh

```
1
2   echo "installing Node.js 16"
3   curl -sL https://deb.nodesource.com/setup_16.x | sudo bash -
4   sudo apt -y install nodejs
5
6   echo "installing python 3.9"
7   sudo apt install python3.9-venv
8   sudo apt-get -y install python3-pip
9
10  python3.9 -m venv venv
11
12  echo "Installing python deps"
13  source venv/bin/activate
14  pip3.9 install -r pipeline/requirements.txt
15  pip3.9 install autopep8
16
17  echo "Installing react app deps"
18  mkdir web-app/public
19  cd web-app
20  npm i
21  cd ..
22
23  echo "Installing Node.js deps"
24  cd pipeline/node_scripts
25  npm i
26
27  echo "install screen"
28  sudo apt install screen
```

Figure 17. A script to install the dependencies

One last step before running the project is to download the CityGML files to the VM. For that the author has used the **scp** command which transfers the files from one machine to another. Another way that it could be done in the future is to create a script that would fetch the files directly from the Land Board website and, perhaps, even embed it in the pipeline.

7.2 Pipeline processing time

On the m2.xsmall instance mentioned in the section above, the processing of Tartu CityGML file takes 80 minutes. The biggest part of time is used for requests to PVGIS API - 65 minutes a because those are throttled to avoid rate limiting. Considering the pipeline is not meant to be run often the result is satisfactory. If the solar potential estimations were done locally, for example via Pvlib mentioned before, then the pipeline's performance would drastically improve. As also mentioned previously, the usage of more VCPUs should not give a big performance difference, therefore such experiments were not conducted.

Also important to mention that Tartu has 20896 buildings in the dataset and the amount of roof surfaces detected equals 76830. The information in this subsection is taken from the logs of the pipeline which are very detailed to allow further performance analysis.

7.3 Running the application

Once the VM was setup and pipeline was run to analyze target cities, the web app can be started. At this stage, the VM has to be configured to accept requests on the port where the app runs. After that the application can be accessed on the network. As mentioned previously, before starting the app a script to detect cities which were processed is run and therefore once new cities are analyzed, the app has to be restarted. Combined script to run the app is located in the repository and can be seen in Figure 18.

scripts > web.sh

```
1  echo "Starting the web app"
2  cd web-app
3  npm i
4  npm run preprocess
5  npm run start
```

Figure 18. A script to run the web app

7.4 Containerization

To simplify the process of configuring environment and running the pipeline and web application, the author has used Docker [10]. Docker removes the need to configure local environment and allows to run applications easily on any operating system (OS). Instead of the local setup, the applications run in isolated containers with specified dependencies which are installed during build.

To containerize the pipeline author has created a Dockerfile where the source code and data are copied and required dependencies are installed. For the pipeline to be able to write the data back to host machine from Docker, author has used mount binding. This is necessary so that web app would be able to consume the outputs of the pipeline. The Dockerfile for the pipeline can be seen in Figure 19 and the commands examples to build and run the pipeline are:

- `docker build -t pipeline -f Dockerfile.pipeline .`
- `docker run -v <full-path-to-repo>/data:/app/data pipeline – <pipeline-param> <some-value>`

Dockerfile.pipeline > ...

```
1   FROM python:3.9
2
3   WORKDIR /app
4   COPY /data /app/data
5   COPY /pipeline /app/pipeline
6
7   RUN apt-get update;
8
9   # Install Node
10  RUN curl -sL https://deb.nodesource.com/setup_16.x | bash - && apt -y install nodejs;
11
12  # Python deps
13  RUN pip install -r pipeline/requirements.txt;
14
15  # Node deps
16  RUN cd pipeline/node_scripts && npm i;
17
18
19  ENTRYPOINT ["python", "pipeline/main.py"]
```

Figure 19. Dockerfile for pipeline

Containerizing the web app was done in a similar way. Source code and data folders are copied when the image is built. One noticeable difference is a port in the container has to be mapped to the host port when running the container, so that the application

is accessible. The Dockerfile for the web application can be seen in Figure 20 and the commands to build and run the pipeline are:

- `docker build -t web -f Dockerfile.web .`
- `docker run -p 3000:3000 web`

Dockerfile.web > ...

```
1   FROM node:16
2
3   WORKDIR /app
4   COPY /web-app /app/web-app
5   COPY /data /app/data
6
7   EXPOSE 3000
8
9   RUN apt-get update;
10
11  # Install Node
12  RUN curl -sL https://deb.nodesource.com/setup_16.x | bash - && apt -y install nodejs;
13
14  WORKDIR /app/web-app
15
16  # Node deps
17  RUN npm i;
18
19
20  CMD ["npm", "run", "start-full"]
```

Figure 20. Dockerfile for web application

8 Conclusion

As stated at the beginning of the thesis, the aim was to create a tool capable of outputting the solar potential of Estonian cities. As a result, a processing pipeline that estimates the solar potential of buildings based on the 3D city data was developed. The pipeline outputs both raw data in JSON format with estimations for the whole city as well as each individual roof surface. Additionally, the author has developed a web application capable of visualizing the results.

The developed pipeline can be run for any Estonian city present in the data provided by the Land Board, empowering further research and analysis on how to improve the power efficiency of various cities. The data produced by the pipeline can be used as a basis for various data science applications, as well as for embedding it in new and already existing web applications.

The developed web application can be used for demonstration purposes and visual analysis. From there it is possible to identify the buildings with the highest solar potential, as well as see some statistics for the whole city.

Also, the solar power estimations made for Tartu can be compared to the total energy consumption of Estonia mentioned in section 2.4. To recall, the amount of energy consumed in 2020 equals 5516 GWh. Based on the estimation from the developed application, a total of 886 GWh of solar energy can be produced from buildings of Tartu city alone which is about 16% of the total consumption. Here it is important to note the parameters used: PV system efficiency 20%, PV system losses 14% and solar panel roof coverage 80%, which might be not fully realistic. Taking into account, that solar energy is only one potential source of renewable energy this result seems quite promising. Using the developed tool for doing the estimations of other cities and counties in Estonia could bring considerable numbers that should be used as a motivation for the further improvements in green energy transition.

Compared to existing options such as ArcGIS Pro and Solar3DCity, developed solution has several benefits:

- It is based on the modern technology stack and can be easily run via Docker
- Weather is taken into account for locations anywhere in the world due to PVGIS API integration
- Provided as a free open-source solution under MIT license, can be easily improved and extended
- Multiple cities can be processed at once with no intermediate manual steps required

Even though the final solution is not perfect, the goals of this thesis can be considered achieved, as the solution to estimate and visualize solar potentials of the cities was developed. In the current state it can be already used within Estonia. However, with the

improvements defined in section 8.2 it would be possible to make it much performant and easier to use.

8.1 Current issues

Even though the solution is fully working, there are areas to improve in terms of performance, cost / resource utilization and maintenance and usability:

- Relatively slow pipeline processing time because rate limited API is used
- A powerful VM which mainly runs a simple server, because it is coupled with the pipeline and the data
- More manual steps needed to add new cities to a running web application than desired

8.2 Suggested improvements

To address the issues outlined above, author suggests the following things.

For performance:

- Attempt to replace PVGIS API with Pvlib or a suitable GIS tool to do the estimations locally, should drastically improve performance of the pipeline

For resource/cost efficiency:

- **citygml-to-3dtiles** should be replaced or improved to decrease the RAM requirements
- File server should introduced for storing outputs
- Pipeline should be run on-demand, download needed CityGML file and send the output to a file server
- Web application should be running on appropriate hardware (2-4GB of RAM should be enough)

For usability:

- Pipeline can fetch the specified CityGML file(s) from the web
- Pipeline can be easily triggered without a need to ssh to machine, cloud function seems like a good option to consider
- Web application should be constantly running and should fetch the data from the file server on reload

8.3 Future work

In addition to improvement suggested above, there are various opportunities for future work. One could re-attempt converting CityGML to CityJSON and develop a project with tools for CityJSON which are more wide and are actively developed. Work in that direction could lead to more easily customizable visualization apps, built on top of open sourced CityJSON visualization tools and web components. Further more, it is possible to use the already developed app to analyze cities in Estonia and find buildings with high solar potential and report them to the local governments. Lastly, it is possible to analyze the current geometry of the rooftops and find out if it is possible to suggest improvements for future construction works that would increase total solar potential of the city as it was researched in Toronto [33].

References

[1] 3d data of estonian cities. https://geoportaal.maaamet.ee/eng/Download-3D-data-p837.html. Accessed: 14.05.2022.

[2] 3d tiles format description. https://www.ogc.org/standards/3DTiles. Accessed: 30.05.2022.

[3] Apvi solar potential tool. http://pv-map.apvi.org.au/potential. Accessed: 10.06.2022.

[4] Arcgis. https://www.esri.com/en-us/arcgis/about-arcgis/overview. Accessed: 02.05.2022.

[5] Area of triangles and polygons. https://web.archive.org/web/20210418090316/http://geomalgorithms.com/a01-_area.html. Accessed: 15.06.2022.

[6] Autopep8. https://pypi.org/project/autopep8/. Accessed: 15.05.2022.

[7] Cesium. https://cesium.com/. Accessed: 31.05.2022.

[8] Citygml-to-3d-tiles package. https://github.com/njam/citygml-to-3dtiles. Accessed: 31.05.2022.

[9] Citysim. https://www.epfl.ch/labs/leso/transfer/software/citysim/. Accessed: 02.05.2022.

[10] Docker. https://www.docker.com/. Accessed: 25.07.2022.

[11] Eesti energia solar panels. https://www.energia.ee/en/era/taastuvenergia/paikesepaneelid?orderDetails=1&tabgroup_2=seadmed-paigaldus. Accessed: 25.06.2022.

[12] Elevation data. https://geoportaal.maaamet.ee/eng/Spatial-Data/Elevation-data-p308.html. Accessed: 25.07.2022.

[13] Epsg:3301 definition. https://epsg.io/3301. Accessed: 18.05.2022.

[14] Estonian energy statistics. https://www.stat.ee/en/find-statistics/statistics-theme/energy-and-transport/energy. Accessed: 25.06.2022.

[15] Geodatabase format description. https://desktop.arcgis.com/en/arcmap/latest/manage-data/geodatabases/what-is-a-geodatabase.html. Accessed: 18.05.2022.

[16] Grass gis. https://grass.osgeo.org/learn/overview/. Accessed: 02.05.2022.

[17] Ladybug. https://www.ladybug.tools/. Accessed: 02.05.2022.

[18] Numpy. https://numpy.org/. Accessed: 20.05.2022.

[19] Obj format description. https://docs.fileformat.com/3d/obj/. Accessed: 18.05.2022.

[20] Pvgis tool. https://re.jrc.ec.europa.eu/pvg_tools/en/. Accessed: 30.05.2022.

[21] Pyproj. https://pypi.org/project/pyproj/. Accessed: 20.05.2022.

[22] React. https://reactjs.org/. Accessed: 30.05.2022.

[23] Solar potential of osnabrück roof areas. https://www.osnabrueck.de/solardachpotenzial/solarpotenzial-osnabruecker-dachflaechen. Accessed: 12.05.2022.

[24] Solcast api. https://solcast.com/solar-data-api/api/. Accessed: 15.06.2022.

[25] Solpy. https://github.com/nrcharles/solpy. Accessed: 15.06.2022.

[26] Vite. https://vitejs.dev/. Accessed: 30.05.2022.

[27] Web-based 3d data visualization with citygml city models. https://towardsdatascience.com/web-based-3d-data-visualization-with-ciytgml-city-models-f0796b37e9f5. Accessed: 30.05.2022.

[28] Filip Biljecki, Gerard B M Heuvelink, Hugo Ledoux, and Jantien Stoter. 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, December 2015.

[29] Estefanía Caamaño-Martín, Ester Higueras Garcia, Javier Neila, I. Useros, D. Masa-Bote, Fiorella Tortora, S. Díaz-Palacios, X. Marrero, A. Alonso, A. Saade, M. Jedliczka, C. Miquel, M. l'Epine, E. Willdbrett, E. Kjellsson, A. Cornander, and J. Fernandes. Solar potential calculation at city and district levels. volume 155, pages 675–685, 05 2012.

[30] Jessie Copper, Mike Roberts, and Anna Bruce. Spatial analysis of solar potential in canberra, 02 2018.

[31] Patrick Cozzi. Introducing 3d tiles. https://cesium.com/blog/2015/08/10/introducing-3d-tiles/. Accessed: 01.06.2022.

[32] D. Giannelli, Camilo León-Sánchez, and Giorgio Agugiaro. Comparison and evaluation of different gis software tools to estimate solar irradiation. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, V-4-2022:275–282, 05 2022.

[33] Javeriya Hasan, Miljana Horvat, Charles Riddell, and Ruotao Wang. The impact of roof morphology on solar potential: making toronto suburbs solar ready. *Journal of Physics: Conference Series*, 2042:012101, 11 2021.

[34] Mehdi Hatamian. Predicting location-based green energy availability in smart buildings. Master's thesis, University of Tartu, 2022.

[35] William F. Holmgren, Clifford W. Hansen, and Mark A. Mikofski. pvlib python: a python package for modeling solar energy systems. *Journal of Open Source Software*, 3(29):884, 2018.

[36] Xiaoxun Huang. Estimation of rooftop solar power potential by comparing solar radiation data and remote sensing data—a case study in aichi, japan. *Remote Sensing*, 14, 04 2022.

[37] Shweta Khandelwal, Asfa Siddiqui, Sardar Patil, Sonal Khobrgade, and Harpreet Singh. An estimation of solar energy potential using point solar radiation tool in arc-gis: A case study of college of engineering, pune. *International Journal of Applied Research 2015; 1(9): 890-897*, 9:890–897, 08 2015.

[38] Jaromir Landa, David Procházka, and Jiri Stastny. Point cloud processing for smart systems. *Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis*, 61:2415–2421, 12 2013.

[39] Iñaki Prieto, José Izkara, and Francisco Delgado del Hoyo. From point cloud to web 3d through citygml. pages 405–412, 09 2012.

[40] Elia Quirós Rosado, Mar Pozo, and José-María Ceballos. Solar potential of rooftops in cáceres city, spain. *Journal of Maps*, 14:44–51, 01 2018.

[41] Teresa Santos, Nuno Gomes, Sergio Freire, Luís Silveira Santos, and José Tenedório. Applications of solar mapping in the urban environment. *Applied Geography*, 51:48–57, 07 2014.

[42] Jason Svarc. Most efficient solar panels. https://www.cleanenergyreviews.info/blog/most-efficient-solar-panels. Accessed: 29.05.2022.

[43] Varkie C. Thomas. Azimuth angles of building surfaces. https://energy-models.com/azimuth-angles-building-surfaces. Accessed: 25.07.2022.

[44] Vishnu Mahesh Vivek Nanda, Laura Tateosian, and Perver Korca. Gis-based estimation of seasonal solar energy potential for parking lots and roads. 12 2020.

Appendix

I. Code

Source code is publicly available under MIT license on GitHub at `https://github.com/boroma4/Mapping-Solar-Potential-of-Tartu` . The repository also contains a brief manual about how to get started. A fork of the city-gml-to-3dtiles can be found at `https://github.com/boroma4/citygml-to-3dtiles`. Example output data can be seen at `https://github.com/boroma4/solar-potential-output-example`.

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Bohdan Romashchenko**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Mapping Solar Potential of Tartu,

   supervised by Pelle Jakovits.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Bohdan Romashchenko
07.08.2022