Efficient Solar Potential Estimation of 3D BAG; a large 3D City Model of The Netherlands

P2 Report

Robin
Hurkmans

student #4370511
1st supervisor: Stelios Vitalis
2nd supervisor: Hugo Ledoux

Date P2: 2021-06-09

June 7, 2021

1 Introduction

Solar energy is the sustainable and renewable energy source with the most promising future prospects to meet global energy demand (Kabir et al., 2018). Attaching solar panels to building roofs is one such way to generate solar energy and is already being applied extensively in practice. Whether individual building roofs are suitable for solar panels depends on various factors, such as the type of the roof (Song et al., 2018), the orientation of the roof surface and whether the roof is obstructed by another object, for instance a building or a tree (Pinde and Rich, 2002). So, correctly oriented building roofs are ideal locations for solar panels because the generated energy can directly be applied to the household and as roofs are elevated, they have a higher chance of avoiding shadows cast on the panels than locations on ground-level. Building facades can also be suitable locations for solar panels. Although they generate less solar energy than roofs, the area to attach solar panels onto is larger, yielding sufficient solar energy (Jaugsch and Löwner, 2016).

To determine the solar potential of buildings a solar irradiance model is needed. Modules of well-known GIS packages such as Solar Radiation Tools (former Solar Analyst) in ArcGIS (ArcGIS, 2016) and r.sun in GRASS GIS (Hofierka and Suri, 2007) offer solid and fast solar irradiance modelling tools for raster data like digital elevation models (DEMs). However, raster data does not model the 3D urban environment accurate enough. This is specifically noticeable for vertical surfaces such as facades, as DEMs model the environment in at most 2.5D (Redweik et al., 2012). Another downside is that raster data is captured with a certain spatial resolution, causing loss of information with low resolutions. The higher the resolution, the longer the process takes.

To capture the 3D urban environment with more spatial detail, buildings can be stored as vector data in 3D meshes. As a benefit, vector data needs less computer memory than raster data. Moreover, vector data is more versatile than raster data and it is built up of geometric primitives like polygons and lines making vector data suitable to accurately represent 3D building meshes. However, techniques that take only 3D vector data as input for solar irradiation modelling are not widely implemented or available yet. The ones that are described or implemented are generally not fast enough for large scale data sets (Hofierka and Zlocha, 2012; Liang et al., 2015; Wieland et al., 2015). This is mostly due to the necessary and compute-intensive shadow casting step.

Nowadays, more and more 3D city models are openly available. An example of this is the massive 3D model of all the buildings in The Netherlands. It is openly available as the data set 3D BAG (3D Geoinformation Research Group, TU Delft, 2021), where BAG stands for 'Basisregistratie Adressen en Gebouwen', meaning Register of Buildings and Addresses. This data set contains a representation of all the buildings as vector data in a level of detail (LoD) of at most 2.2, which includes vertical walls and sloped roof surfaces.

The goal of this thesis is to implement a fast solar irradiation model, but with realistic outcomes, for large 3D city models by taking the 3D BAG data set as a use case. As the 3D BAG is a large vector data set, determining whether a building is put in shadow by neighbouring objects is a compute-intensive process. To overcome these time-related challenges, efficient approaches within the field of ray tracing to perform shadow casting are needed and simplifications in the solar irradiation model need to be made.

2 Related work

Computing solar irradiation can be done for a complete geographic area, regardless of the contents of this area. An example is a mountainous landscape, where for each cell in the raster data set (typically a DEM) the solar irradiation is computed. For this thesis I am not interested in doing computations for all contents in an area, but only for specific parts of the data set, namely the building roofs and walls in urban data sets.

When presenting the existing methods for computing the solar irradiation of a building, one can distinguish the methods on its input data type, namely raster and vector data. These data types both need a different approach when computing solar irradiation. Table 1 gives an overview of some of the existing tools and methods for solar irradiation modelling addressed for this thesis. For each tool or method it states the publication year, the type of input data for the solar irradiation method, a key feature and the citation. Freitas et al. (2015) also performed a state-of-the-art review of available solar radiation models.

2.1 Raster data

One of the first solar irradiation models was SolarFlux (Hetrick et al., 1993). This method used simple formulas and was unfortunately not suitable for large area raster maps.

A first approach in implementing solar irradiation models with faster speeds, higher accuracy and open source availability was done by Pinde and Rich (2002). They created the tool called Solar Analyst, which is still implemented, although improved, within the ArcGIS software package. Their method takes the topography, elevation and surface orientation of a DEM into account. Obstructions by surrounding topographic features, like hills and trees are also considered, by using a hemispherical viewshed algorithm. The resulting insolation maps include both direct and diffuse radiation. Their resolution back then was 30 meter, which is quite low for building roofs while sufficient for large geographic areas. This work is a good starting point for solar irradiation studies and is therefore cited a lot by other authors as well.

A more recent methodology utilising the Solar Analyst tool as described by Kodysh et al. (2013) also makes use of a hemispherical viewshed. It is raster-based and has a resolution of less than 1 meter, which is necessary to capture all features of a roof. A similar tool as the Solar Analyst was implemented for GRASS GIS, called *r.sun* (Hofierka and Ri, 2002; Hofierka and Suri, 2007). It is also raster-based and widely used nowadays with reliable accuracy.

The way these tools work to compute the solar irradiation for specific buildings is to combine the raster data set with a 2D vector data set of the building footprints. The building footprints are then used as a mask laid over the pixels in the raster data set. Only these pixels are taken into consideration when computing solar irradiation values for the building roofs. As one can imagine, the higher the resolution of the raster grid, the better the geometry of the building roofs can be represented.

The work by Song et al. (2018) focuses more on the various shapes of the rooftops. It states that the solar potential differs per rooftop type and it selects the most suitable roof types to use for solar irradiation computations based on some filters. Stendardo et al. (2020) incorporate the GPU to speed up the shadow casting algorithm.

Tool/Method	Year	Type	Key feature(s)	References
SolarFlux	1993	Raster	One of the first tools	Hetrick et al. (1993)
Solar Analyst	2000	Raster	Widely used, reliable	Pinde and Rich (2002)
GRASS GIS *r.sun*	2002	Raster	Widely used, reliable	Hofierka and Ri (2002); Hofierka and Suri (2007)
Solar Potential on multiple building rooftops	2013	Raster	Uses Solar Analyst. Combines high-resolution DEM with upward-looking hemispherical viewshed	Kodysh et al. (2013)
Solar Potential Remote Sensed Rooftops	2018	Raster	Takes variable rooftops into account	Song et al. (2018)
GPU-Enabled Shadow Casting (Geneva)	2020	Raster	Computationally efficient for high resolution, but shadow casting for rasters	Stendardo et al. (2020)
v.sun	2012	Vector to voxel	Vector-voxel approach, lacks benchmarking	Hofierka and Zlocha (2012)
SORAM	2013	Vector	Focused on diffuse radiation	Erdélyi et al. (2014)
Solar3DCity	2015	Vector	About error propagation, takes CityGML as input but does not consider shadows	Biljecki et al. (2015)
Solar radiation on CityGML building data	2015	Vector	Takes CityGML as input, does consider shadows. Uses point sampling on the surfaces	Wieland et al. (2015)
SURFSUN3D	2015	Vector to raster	Rasterises 3D triangular meshes, but not fast enough for higher LoD	Liang et al. (2014, 2015)
Sparse Voxel Octree (SVO)	2017	Vector to voxel	Extension of SURFSUN3D, sparsely voxelises the 3D triangular meshes in an octree	Liang and Gong (2017)
Skyline-based method	2019	Vector	Computes SVF and SCF. Also uses the GPU to speed up	Calcabrini et al. (2019)
3D shadow cast vector-based model	2020	Vector	Accurate; different projection techniques used based on the tilt of the surfaces	Viana-Fons et al. (2020)
Vertical 3D walls	2016	Raster to 3D points	Uses observer point columns to efficiently ray cast 3D vertical walls	Jaugsch and Löwner (2016)
Solar3D	2020	OAP3D	Very accurate, it uses the cube mapping technique on 3D photographs of buildings	Liang et al. (2020)

Table 1: Overview of various solar radiation modelling tools or methods

2.2 Vector data

Vector data is more accurate than raster data as it is built up of geometric primitives such as polygons and lines, representing the 3D environment with more spatial detail. This makes modelling of for instance vertical facades easier. Moreover, vector data is more efficient to store in computer memory than raster data. However, raster data is simpler and faster to use as the resolution can be determined beforehand. Therefore, the challenge is to find a way to process the vector data computationally efficient to incorporate the higher spatial detail into the resulting solar potential of the individual 3D buildings.

An early approach in using 3D vector data in complex urban environments as input for solar irradiation computations is described by Hofierka and Zlocha (2012). The module *v.sun* is created, which is based on the existing module *r.sun* to compute solar irradiation values. It takes a vector-voxel approach to segment the vector objects into smaller polygons. Then, for each voxel the proposed algorithm is applied: calculating the normal vector, solar ray directions for various time and the shadows cast by neighbouring buildings. However, the performance of the method is not assessed and it is not implemented in the GRASS GIS package as was proposed to be.

A tool taking 3D city models stored in CityGML files as input for solar irradiation computations is Solar3DCity (Biljecki et al., 2015). The tool extracts the roof surfaces for each building and computes their inclination, area and orientation. Afterwards, it estimates the yearly solar irradiation for each surface by using the solpy library (Charles, 2015) and writes the enriched 3D city model back to the CityGML file. A notable shortcoming of this tool is that it lacks an implementation for shadows cast by neighbouring buildings. This is omitted because of computational reasons.

The study described by (Wieland et al., 2015) also takes 3D city models stored in CityGML format as input for solar irradiation computations. The improvement is that it does incorporate the determination of shadows. Their workflow is as follows. For each roof and wall surface of a building they establish a point grid. Then for each point they cast a line towards the direction of the sun for specific points in time. By using an intersection tool in PostGIS they check whether this line intersects with surrounding buildings to determine whether the corresponding point is in shadow or not. Afterwards, the solar irradiance (beam and diffuse) is computed for each surface point. Lastly, the values are integrated over time and area to arrive at monthly values per building. This is a simple and effective way to compute the shadows cast by surrounding building. However, it is quite inefficient because of the high computation costs. This can be explained by the inefficient way the buildings are processed; just brute force.

The work by Liang and Gong (2017) also takes a 3D vector-voxel approach to compute solar irradiation like Hofierka and Zlocha (2012), but this time in combination with a sparse voxel octree (SVO) to store the geometries. The strong point of using an SVO as storage model is that the computation time of shadow casting does not slow down for more complex geometries. Its time complexity stays the same with increasing LoD. Its predecessor is about just rasterising the 3D meshes in 2D (Liang et al., 2014), which does slow down shadow casting with increasing LoD. However, storing a 3D city model into an SVO (Liang and Gong, 2017) is memory inefficient, making it unusable for large 3D city models. Therefore, a better storage model than the SVO is needed as indexing structure to speed up obstruction detection for shadow casting.

The work described by Calcabrini et al. (2019) uses the indicators sky view factor (SVF) and sun coverage factor (SCF) to represent the skyline profile for a building in a 3D city model. The method is implemented for direct, diffuse and reflected solar irradiation making the model quite complex, but accurate. It also makes use of the GPU to speed up computations for large areas.

Viana-Fons et al. (2020) describe an efficient 3D shadow cast vector-based model. They create a 3D city model out of LiDAR data and building footprints. Then they apply their shadow model to the surfaces of a building, distinguishing whether a surface is vertical or not. This is necessary as projection techniques are used for different surfaces to obtain a correct shadow profile. Their resulting values have low errors compared to real data, yielding a high accuracy.

2.3 Other types

The tool Solar3D is mainly focused on using a cube mapping technique to accurately recreate 3D scenes based on oblique airborne photogrammetry-based 3D-city models (OAP3D) (Liang et al., 2020). For this, real life photographs of the scene are needed, which is unfeasible for large data sets. For this to work, the tool also relies on DEMs, DSMs and feature layers as input data. The tool yields a high accuracy.

To enlarge solar energy generation, solar panels can also be installed on the facades of buildings. Jaugsch and Löwner (2016) implemented a method that takes 2D maps with additional height information as input. 3D block models are extruded from this input. They introduced the concept of observer point columns which determines whether a whole vertical part of the facade is illuminated or not. To speed up shadow computations they take the maximum shade length determined by the tallest building. As the building blocks are extruded from 2D data resulting in 3D mesh data, this method is considered as a hybrid approach between raster, points and vector to model the solar potential.

2.4 Simplifications and acceleration structures

Further methods to speed up computation times are simplifications in the solar irradiation models and to apply acceleration structures. A possible simplification is the usage of the characteristic declination of the sun (Brito et al., 2021). This is the declination for the day on which the daily solar irradiation on a horizontal surface is identical to its monthly average value. This value, instead of daily values, can be used to reduce computing demand by a factor of 30. But, accuracy errors are introduced ranging from +5% to +12% depending on the latitude. The higher the latitude, the higher the errors. This should be kept in mind when applying this simplification to the solar irradiation model.

Another way to achieve better computation times is the usage of acceleration structures, indexing the buildings in the 3D city model. This is beneficial when performing shadow casting as obstructing neighbouring buildings can be efficiently found based on the index. The SVO as used by Liang and Gong (2017) is an example of such an acceleration structure. However, the SVO and voxelisation in itself raises some limitations and deficiencies as it is memory inefficient to voxelise a massive 3D city model. The memory usage increases proportionally with the size of the 3D city model. Another downside is that data management is challenging, because the 3D city model loses its spatial relations and semantics after the voxelisation.

Therefore, another acceleration structure is necessary for massive 3D city models. The bounding volume hierarchy (BVH) is a tree structure to spatially organise data by wrapping the geometries into bounding boxes (Wald et al., 2007). Spatial relations and semantics of the geometries are not lost and it is also an efficient way to detect intersections of neighbouring objects when doing ray tracing. The time complexity of BVH is $O(log(n))$. This is achieved by the fact that geometries stored in children nodes do not have to be checked for intersections when the bounding box of the parent node is not intersected by the ray. Moreover, the BVH is already used in many ray tracing algorithms.

The two acceleration structures described can be distinguished by their purpose in two classes: space subdivision methods and object subdivision methods. SVO is a space subdivision method while BVH is an object subdivision method. Space subdivision methods subdivide the 3D space into smaller regions by using planes, where the geometry might belong to multiple regions. Object subdivision methods subdivide the geometry into smaller objects, resulting in tight volumes wrapped around the object (Pharr et al., 2004 - 2021). The two method classes are both successful at speeding up the ray intersection method, hence SVO and BVH are both suitable for shadow casting.

But, BVH seems better for large 3D city models as the spatial relations and semantics of the geometries stay intact. Also, no literature describes the usage of BVH for shadow casting in solar irradiation models yet. Therefore, this is worth researching.

3 Research questions

The main objective of this thesis is to compute the solar potential of buildings in large 3D city models, such as 3D BAG, computationally efficient while still having realistic outcomes. The 3D BAG data set covers all buildings in The Netherlands as vector data. The accompanying main question is:

How can the solar potential of vector buildings in large 3D city models, such as the 3D BAG data set be computed computationally efficient?

Sub-questions to help answering the main question:

- *How can BVH be used as indexing structure to speed up shadow casting computations on the 3D BAG vector data set?*
- *What simplifications in the solar irradiation model can be applied while still having realistic solar potential outcomes?*
- *How can the solar irradiation model be implemented in Python by using open source libraries and open data?*

3.1 Scope

During this thesis the aim is to fully implement one methodology that computes the solar potential for all the buildings in the 3D BAG data set. The main focus will first be to compute the solar potential for roof surfaces only. When testing for neighbouring objects that are possibly casting shadows on the respective roof surfaces, the possible obstructing objects taken into consideration will only be the buildings in the 3D BAG data set. If the process goes smoothly, extending solar potential calculations to vertical wall surfaces and taking other objects such as trees into consideration when testing for obstructions can be integrated. Furthermore, at first, the solar potential will be determined by taking into account only direct radiation, neglecting the effect of diffuse and reflective radiation.

4 Use case

This section describes the use case to which the methodology will be tested. The data set 3D BAG is chosen as use case (3D Geoinformation Research Group, TU Delft, 2021). This open data set covers all the buildings in the Netherlands as 3D building meshes. It covers the whole country at multiple levels of detail (at most LoD 2.2). An example of some buildings, with the faculty of Architecture and the Built Environment of the TU Delft as central point, is shown in figure 1. The data set is fully automatically generated by combining two open data sets, namely the building data from BAG and the elevation data from AHN.

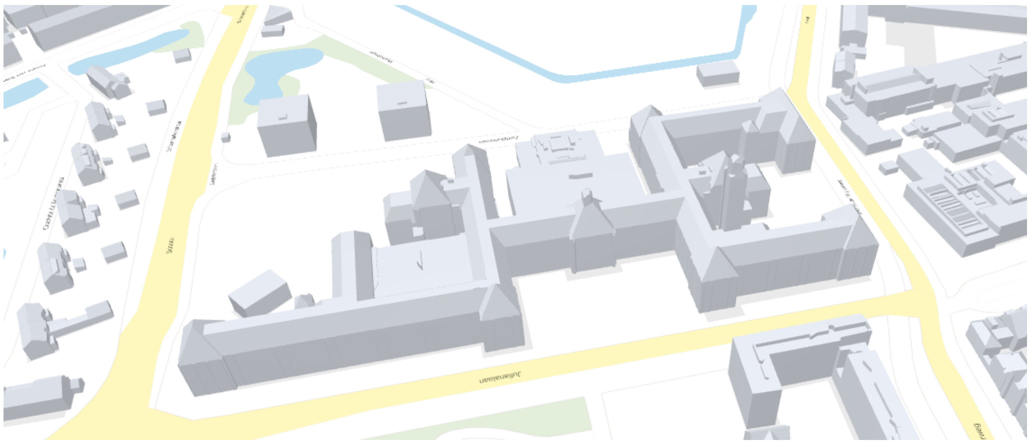
Figure 1: Sample of 3D BAG with LoD 2.2 (taken from the 3D BAG viewer)

The buildings within 3D BAG are individually stored as a 3D mesh at LoD 1.2, LoD 1.3 and LoD 2.2. LoD 2.2 can include sloped building roofs whereas LoD 1.2 and LoD 1.3 only store the buildings as block models with flat roofs. The geometric primitives of the 3D building meshes are triangles and its vertices. For each building a list of attributes is attached. To insure valid geometries, the meshes are run through the program 'val3dity'. In the case a mesh is invalid the error code is included as an attribute. The model also stores semantic surfaces; ground surface, wall surface and roof surface.

Because the data set consist of more than 10 million buildings, the data is subdivided into tiles covering smaller geographic areas, as shown in figure 2. The amount of buildings in a tile are determined by a quad tree data structure. Furthermore, the data set is available in several formats: CityJSON, GeoPackage, Wavefront OBJ, PostgreSQL, WMS and WFS.

As 3D BAG is a large data set covering vector data, it is very suitable to take as use case for my thesis. The fact that the the data set is available in LoD 2.2 and the buildings have semantic surfaces, ensures that sloped roofs are represented correctly, and roof and wall surfaces can easily be distinguished from other surfaces. Moreover, it is advantageous that the data set is available in CityJSON and PostgreSQL. CityJSON is a compact and easy-to-use encoding of the CityGML data model very suitable to use when programming (Ledoux et al., 2019). When dealing with larger parts of the data set it may be beneficial to store it in a database by using PostgreSQL.

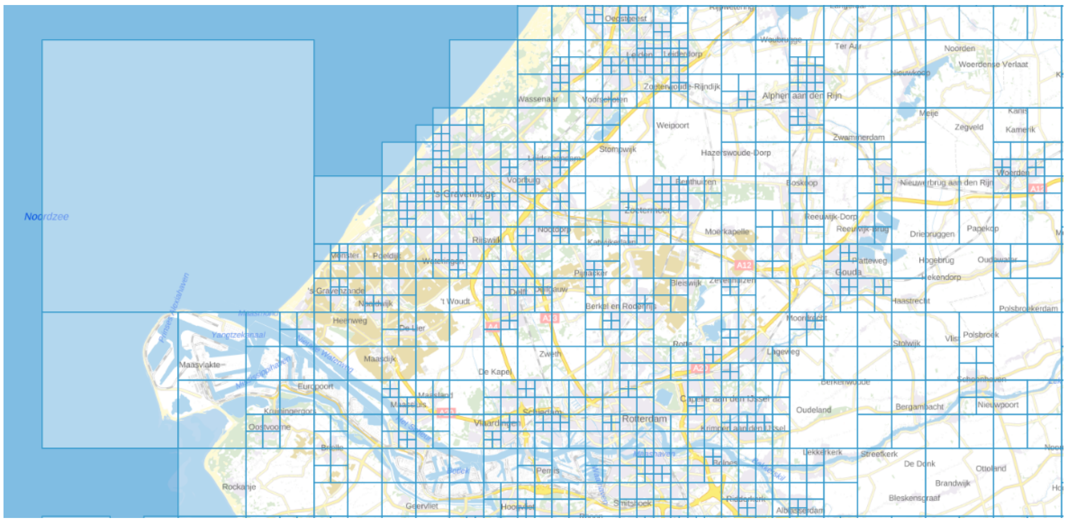
Figure 2: 3D BAG tiles as quad tree

5 Methodology

In short, the methodology can be described as follows. First, building geometries in parts of the 3D BAG data set, stored in CityJSON files, will be loaded in memory. Then, a BVH, wrapped around the building geometries, is formed. Afterwards, each building will be processed in parallel on the CPU where each triangular roof (and wall) surface will be sampled into a grid of points. For each point on the surface, the shadowing is determined by intersection detection in the GPU and the beam solar irradiation is computed accordingly. The resulting values are integrated over time and space. Lastly, the 3D city model is written back to CityJSON, enriched with the solar irradiation value as attribute per geometry. This whole workflow will be implemented in Python and is shown in figure 3.

5.1 Detailed methodology

The proposed methodology will be discussed and justified in more detail below. Its structure can be roughly divided into three parts; preparing input data, processing the building geometries and exporting output data for visualisation.

5.1.1 Preparing input data

The first part of the methodology is to prepare the input data for further processing. The first step is to load the buildings in the 3D BAG data set into memory. The data set is divided in tiles of different geographic extent, so not all data has to be loaded at the same time. At first, the data will be downloaded as CityJSON and parsed. Later on, the data might be downloaded as a PostgreSQL database for more efficient data storage and management if the input grows too large. Once the data is loaded, the building geometries are hierarchically wrapped in axis-aligned bounding boxes (aabb) as bounding volumes to form a BVH tree structure by using the library aabbtree in Python (Hart, 2021). The BVH is chosen as acceleration structure to speed up ray-geometry intersection tests for shadow casting (Wald et al., 2007; Karras, 2012). The BVH will be constructed in a top-down fashion with a degree of 2, yielding a binary tree which is most common practice. This has as downside that the root-to-leaf traversal time takes longer than with non-binary trees, but less time is spend at each tree node to check for bounding box overlaps as the bounding boxes are smaller. An example of a BVH representation of a scene

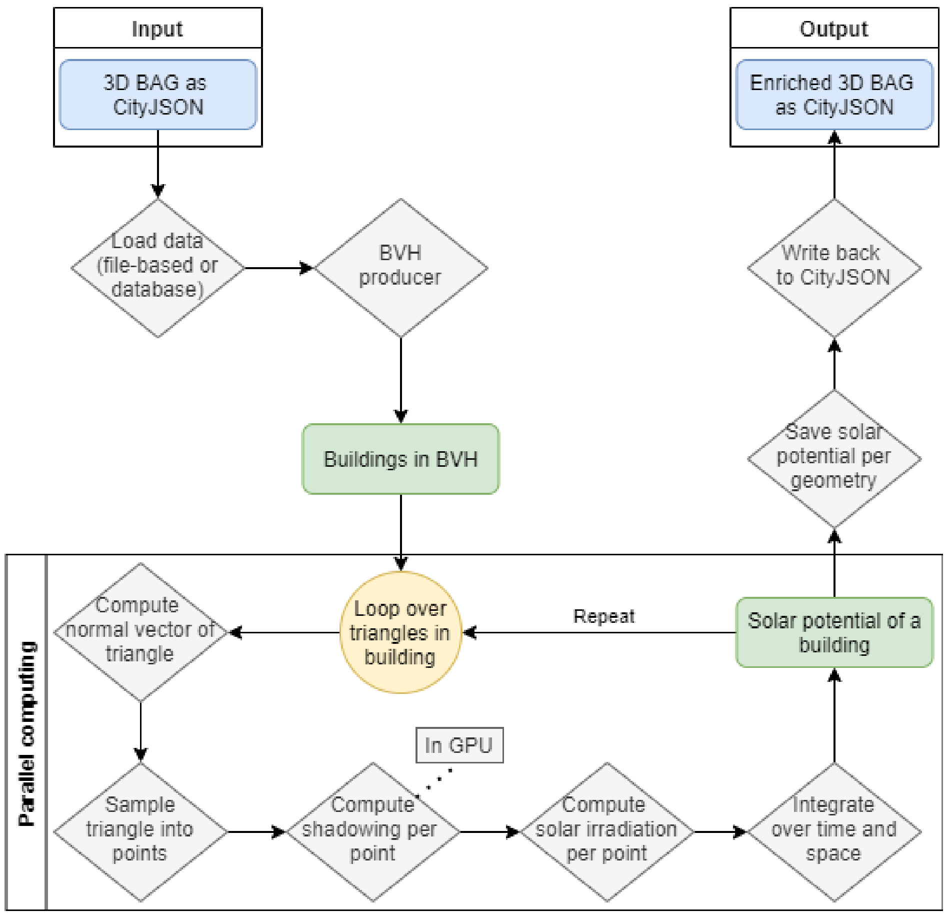
Input
3D BAG as CityJSON
Output
Enriched 3D BAG as CityJSON
Load data (file-based or database)
BVH producer
Write back to CityJSON
Save solar potential per geometry
Buildings in BVH
Parallel computing
Compute normal vector of triangle
Loop over triangles in building
Repeat
Solar potential of a building
In GPU
Sample triangle into points
Compute shadowing per point
Compute solar irradiation per point
Integrate over time and space

Figure 3: Workflow for solar irradiation modelling on 3D BAG

with triangles is shown in figure 4.

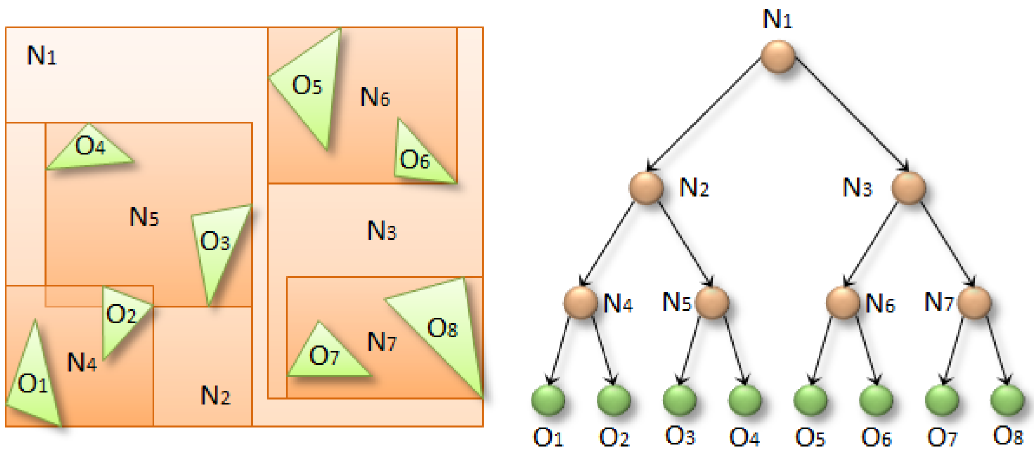

Figure 4: 2D BVH representation of a scene. The orange nodes represent the bounding volumes and the green leaf nodes represent the geometries (Karras, 2012)

5.1.2 Processing building geometries

The second part of the methodology is to process all loaded building geometries to compute their solar irradiation values. Due to the fact that the solar irradiation values of one building do not influence the solar irradiation value of other buildings, each building can be processed in parallel in the CPU to speed up computation times. There are a couple of steps to go through to compute the solar irradiation values, based on the work by Wieland et al. (2015).

First, for all the triangle surfaces, of which the building mesh is built up, the normal vector is computed. Also, each triangle is sampled into a grid of points, with each point having their own *xyz*-coordinate. Sampling is necessary to account for variable solar irradiation values on a triangle, especially on larger triangles. Some parts of the triangle may be blocked by a shadow from a neighbouring object, while other parts are not. An example of sampled surfaces with varying solar irradiation values is shown in figure 5.

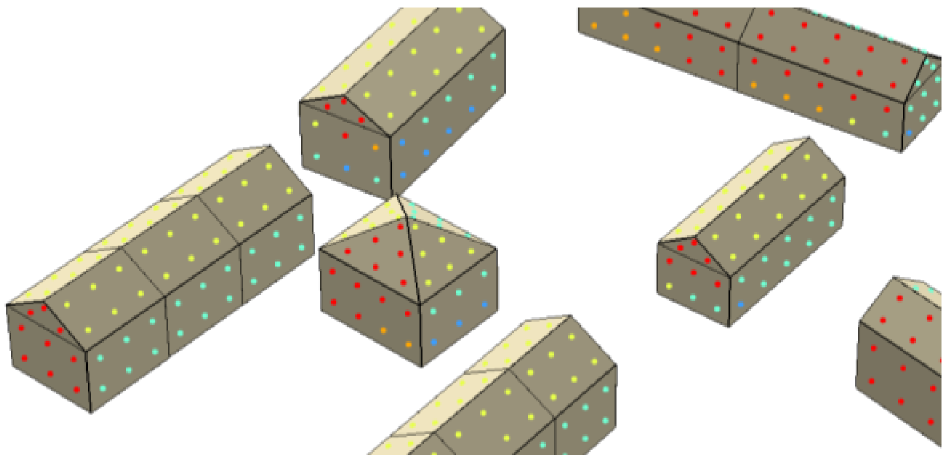

Figure 5: Point sampled building surfaces (Wieland et al., 2015)

Second, for each point, it is determined whether it is illuminated or not, based on the shadows cast by neighbouring buildings. To save processing time, the neighbouring buildings to take into consideration are filtered on their height and distance. Only buildings that are higher than the current building and at a distance smaller than a specified value (say, 300 meter) are taken into consideration. Then, rays originating at each point on the surface are cast in the direction of the sun at a specified time. Due to the fact that building geometries are stored in a BVH, a substantial part of the buildings can be skipped. Only bounding volumes and eventually the geometries inside the bounding volumes that intersect with the ray need to be further checked for intersection. For the intersection check, a fast ray-box intersection method is implemented, like the one described by Williams et al. (2005). As the GPU is fast in processing geometries (Stendardo et al., 2020), the ray casting and intersection check is performed in the GPU, by using a NVIDIA's CUDA for Python (NVIDIA).

Third, for the points on the triangle surface that are illuminated, the beam solar irradiation is computed by using the solarpy library in Python (aqreed, 2019). The library has a function called 'irradiance_on_plane' that takes as input the normal vector of the plane (point on the triangle in this case), the height of the point, a time and the latitude. Its output is the solar irradiance for this point on the triangle in W/m^2. To save computation time, the timestamp to use as input will be the time of the characteristic declination of the sun (Brito et al., 2021). This means that only one value for each month is computed instead of daily values, achieving a performance increase by approximately a factor 30. Now, each point will have its solar irradiation value computed.

5.1.3 Output and visualisation

In the last part of the methodology, the solar irradiation values need to be saved per geometry and integrated over time and space to get $kWh/m^2/year$ as unit of measurement. There are three options to eventually store this value and to export it to CityJSON. Below, these are mentioned, with their pros and cons:

1. *Per point*. This is an accurate representation of the computed values per location. The downside is that it is inefficient to store, because of the many values per triangle and building. Also, a way need to be found to store this both time and space efficiently within a CityJSON file. For this choice, no aggregation over space is needed.

2. *Per triangle*. This is a less accurate representation than per point, but simpler to store in a CityJSON file. For this choice, the values for all the points on the triangle need to be aggregated.

3. *Per building*. This is the least accurate way to represent the computed values, as it is not known which part of the building surfaces has the highest solar potential. It is a too simplified representation, but very easy to extend with an extra attribute. For this choice, the values for all the triangles in the building need to be aggregated.

To extend an object in a CityJSON file with a new attribute, this is easy to do at triangle and building level (Ledoux, 2021a).

To visualise the resulting solar potential values for all the buildings in the enriched CityJSON, a colour scheme will be used to distinguish between high and low solar potential. An obvious colour scheme to choose would be from green to red, where green represent low solar potential and red represents high solar potential. The degree of accuracy depends on the chosen option to store the resulting values. A way to add colour information as appearance to a geometry, would be to use textures or materials in CityJSON. Creating a colour scheme for the geometries within QGIS, would also be an option for a small part of the enriched data set.

5.2 Experiment Design

To test the usability of the implemented methodology, a use case is chosen. The ideal use case would be the whole of The Netherlands as covered in the 3D BAG vector data set, but it is probably more feasible to start with a smaller part because of expected high computation times. Therefore, I have chosen to take a couple of tiles in the province of Zuid-Holland. This region covers densely populated urban areas like The Hague and Rotterdam but also more rural areas in between the urban areas. A challenging pre-processing step necessary to make the use case data suitable for the methodology is stitching the tiles together to be able to take buildings that are stored on other tiles, but within the shadow casting radius, accessible.

But first the methodology will be implemented for one building only. The second step is to take one building with neighbouring buildings to perform shadow casting. The third step is to run the implementation for all the buildings in a whole tile. The last step is to run it for multiple tiles within Zuid-Holland. The ultimate goal would indeed be to run it on all the tiles in the 3D BAG data set.

As already mentioned before, only the solar potential of roof surfaces will first be computed and as neighbouring objects, only buildings are taken into consideration for shadow casting. If the process goes smoothly, the solar potential will also be computed for vertical surfaces and trees can be taken into consideration for shadow casting. The latter requires the integration of the AHN point cloud to identify trees.

5.3 Quality Assessment

After the implementation of the methodology, quality assessment needs to be carried out to check whether the results are reliable and fast enough. The two methods to assess the quality:

1. Ground-truth comparison;
2. Comparing reasonable time complexity and accuracy with other methods;
3. Software benchmarking for different data input sizes and BVH configurations.

For the ground-truth comparison, values from solar radiation measurement stations can be used (EU Science Hub, 2005-2015). These values are in raster format and available for horizontal and inclined surfaces. It covers the area from 60°N to 20°S and from 180°W to 22°30′ E. The size of each grid cell is 0.04°, which is quite large. The solar radiation values of the grid cells will be compared with the values of the corresponding buildings. If this data turns out to be unusable, due to the resolution mismatch, this method will be skipped.

Another way to assess quality is to compare the time complexity and accuracy of the resulting solar potential values of the implemented methodology with the widely used solar irradiation modelling tools in GRASS GIS; *r.sun* and in ArcGIS; Solar Radiation Tools. For an area in the 3D BAG data set, both the proposed methodology and the algorithms in GRASS GIS and ArcGIS will be carried out. As these two latter methods are raster-based, the building footprints of the 3D BAG data are used to mask the raster data. Different resolutions will be chosen to test for accuracy and time complexity.

The third method to assess quality is software benchmarking. The implemented methodology will be run multiple times for varying input size. This will determine the time complexity of the methodology for increasing input size. Another parameter to tweak to compare processing times is the degree of the BVH tree. A higher degree will result in shallower trees, increasing the possibility for overlapping bounding volumes. In worst case, this might reduce the time complexity of the ray casting process.

6 Preliminary results

In order to get familiar with the 3D BAG data set in CityJSON format, one tile within Delft has been parsed in Python. A building with two sloped roof parts in opposite direction, as shown in figure 6 has been chosen to use as experiment to test some Python libraries.

The solarpy library (aqreed, 2019) has been used to compute solar irradiation values in W/m^2 for all roof surface triangles of this building. This is done with a temporal resolution of 15 minutes, just for one day. The input parameters are the normal vector of the roof triangle, which is computed from the vertex coordinates of the triangles, the height of the building and the latitude. The resulting solar irradiation values per time interval can be plotted for the triangles with their normal vectors in the legend. The plot is shown in figure 7. The area under the graph is integrated to get the total solar irradiation of the two roof parts in $kWh/m^2/day$.

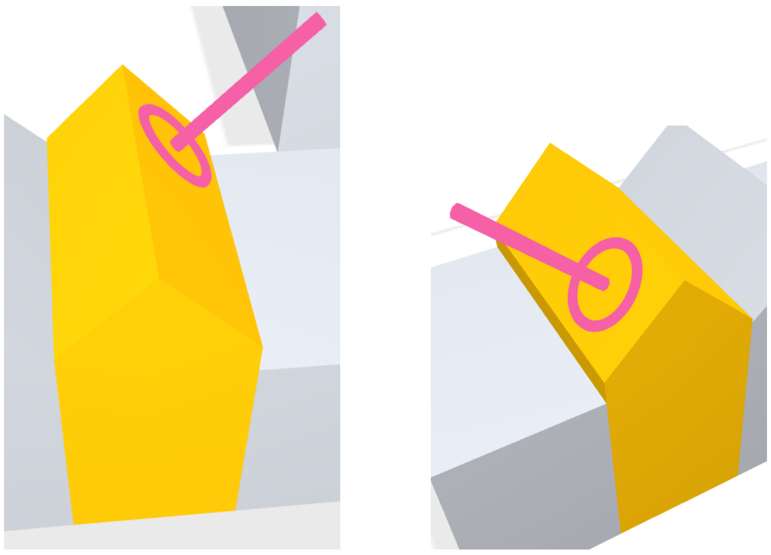

Figure 6: Chosen building to test Python libraries (taken from the 3D BAG viewer)

It yields reliable values.

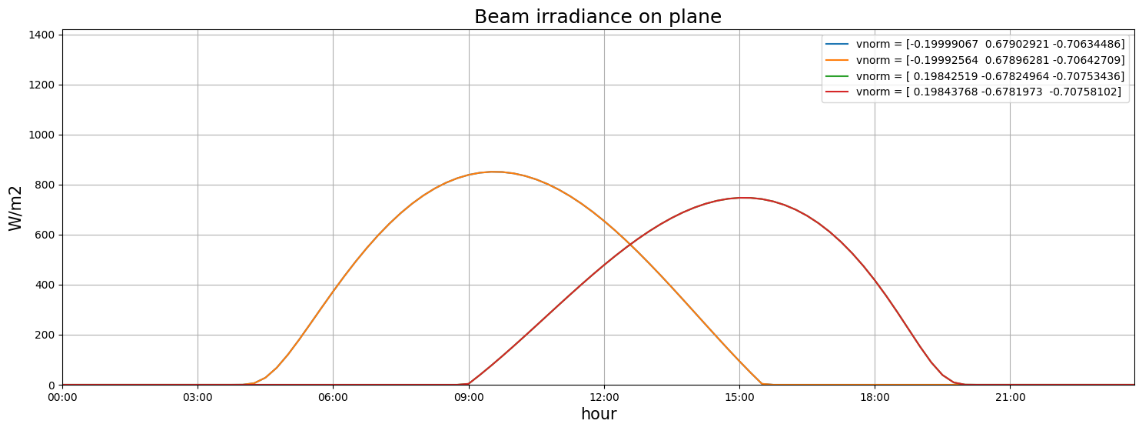

Figure 7: Beam irradiance graph for roof surfaces of chosen building

Furthermore, the resulting roof surfaces have been exported to an .obj file and visualised in MeshLab. Next to that, the ray tracing library pyRT (Christen, 2020) has been used to ray trace and render the building roofs from a fixed camera position to get a feeling for ray tracing.

7 Tools and data

7.1 Tools

To implement the proposed methodology Python will serve as the main programming language. I will use Python because I feel most confident in programming in this language and because it is used a lot for data science. Furthermore, because of its simplicity, scalability and availability of many libraries. The following open source Python libraries will be used:

- NumPy for mathematical computations (Oliphant, 2021);
- cjio for handling the input/output of CityJSON files (Ledoux, 2021b);

- solarpy for computing solar irradiation values (aqreed, 2019);
- aabbtree for creating a BVH tree as acceleration structure (Hart, 2021).

For graphical computations like ray tracing on the GPU, NVIDIA's CUDA Toolkit for Python (NVIDIA) will be used to speed up computation times. Ninja (Nin) will be used to easily visualise 3D city models in CityJSON format. Other visualisation tools like Meshlab and QGIS will also be used.

7.2 Data

The used data will play are large role in the implementation of the methodology. The open data set 3D BAG is considered as a use case and is described in more detail in section 4.

8 Schedule

This section outlines the schedule to be followed during this thesis. The Gantt chart in figure 8 shows the schedule schematically including the tasks and milestones from previous semester and for the remaining months until the final P5 presentation.

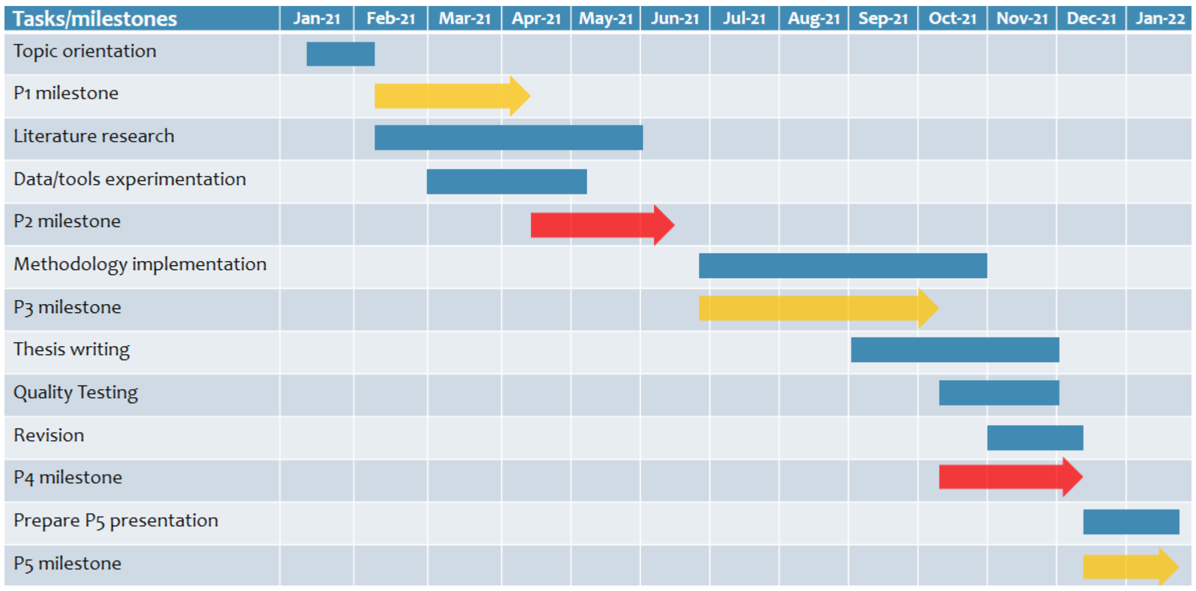

Figure 8: Gantt chart of the thesis' planning

During the whole thesis weekly meetings with my two supervisors will be held to update on the progress and to ask for assistance.

References

ninja. URL `https://github.com/cityjson/ninja`.

3D Geoinformation Research Group, TU Delft. 3D BAG, 2021. URL `https://3dbag.nl/`.

aqreed. solarpy, 2019. URL `https://pypi.org/project/solarpy/`.

ArcGIS. An overview of the Solar Radiation tools, 2016. URL https://desktop.arcgis.com/en/arcmap/10.3/tools/spatial-analyst-toolbox/an-overview-of-the-solar-radiation-tools.htm.

F. Biljecki, G. B. M. Heuvelink, H. Ledoux, and J. 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, 2015. ISSN 1365-8816 1362-3087. doi: 10.1080/13658816.2015.1073292.

M. C. Brito, R. Amaro e Silva, and S. Freitas. Characteristic declination—a useful concept for accelerating 3D solar potential calculations. *Energy Technology*, 9(3), 2021. ISSN 2194-4288 2194-4296. doi: 10.1002/ente.202000943.

A. Calcabrini, H. Ziar, O. Isabella, and M. Zeman. A simplified skyline-based method for estimating the annual solar energy potential in urban environments. *Nature Energy*, 4(3): 206–215, 2019. ISSN 2058-7546. doi: 10.1038/s41560-018-0318-6.

N. Charles. solpy, 2015. URL https://pypi.org/project/solpy/.

M. Christen. pyRT, 2020. URL https://github.com/martinchristen/pyRT.

R. Erdélyi, Y. Wang, W. Guo, E. Hanna, and G. Colantuono. Three-dimensional solar radiation model (SORAM) and its application to 3-D urban planning. *Solar Energy*, 101:63–73, 2014. ISSN 0038092X. doi: 10.1016/j.solener.2013.12.023.

EU Science Hub. NSRDB Solar Radiation Data, 2005-2015. URL https://ec.europa.eu/jrc/en/PVGIS/downloads/NSRDB.

S. Freitas, C. Catita, P. Redweik, and M. C. Brito. Modelling solar potential in the urban environment: State-of-the-art review. *Renewable and Sustainable Energy Reviews*, 41:915–931, 2015. ISSN 13640321. doi: 10.1016/j.rser.2014.08.060.

K. Hart. AABBTree, 2021. URL https://pypi.org/project/aabbtree/.

W. Hetrick, P. Rich, F. Barnes, and S. Weiss. GIS-based solar radiation flux models. *American Society for Photogrammetry and Remote Sensing Technical papers*, 3:132–143, 1993.

J. Hofierka and M. Ri. The solar radiation model for open source GIS: Implementation and applications. 2002.

J. Hofierka and M. Suri. *r.sun - Solar irradiance and irradiation model*, 2007. URL https://grass.osgeo.org/grass78/manuals/r.sun.html.

J. Hofierka and M. Zlocha. A new 3-D solar radiation model for 3-D city models. *Transactions in GIS*, 16(5):681–690, 2012. ISSN 13611682. doi: 10.1111/j.1467-9671.2012.01337.x.

F. Jaugsch and M. O. Löwner. Estimation of solar energy on vertical 3D building walls on city quarter scale. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W2:135–143, 2016. ISSN 2194-9034. doi: 10.5194/isprs-archives-XLII-2-W2-135-2016.

E. Kabir, P. Kumar, S. Kumar, A. A. Adelodun, and K.-H. Kim. Solar energy: Potential and future prospects. *Renewable and Sustainable Energy Reviews*, 82:894–900, 2018. ISSN 13640321. doi: 10.1016/j.rser.2017.09.094.

T. Karras. Thinking Parallel, Part II: Tree Traversal on the GPU, 2012. URL https://developer.nvidia.com/blog/thinking-parallel-part-ii-tree-traversal-gpu/.

J. B. Kodysh, O. A. Omitaomu, B. L. Bhaduri, and B. S. Neish. Methodology for estimating solar potential on multiple building rooftops for photovoltaic systems. *Sustainable Cities and Society*, 8:31–41, 2013. ISSN 22106707. doi: 10.1016/j.scs.2013.01.002.

H. Ledoux. *7.3. Case 1: Adding new complex attributes to existing City Objects*, 2021a. URL `https://www.cityjson.org/specs/1.0.2/#case-1-adding-new-complex-attributes-to-existing-city-objects`.

H. Ledoux. cjio, 2021b. URL `https://github.com/cityjson/cjio`.

H. Ledoux, K. Arroyo Ohori, K. Kumar, B. Dukai, A. Labetski, and S. Vitalis. CityJSON: a compact and easy-to-use encoding of the CityGML data model. *Open Geospatial Data, Software and Standards*, 4(1), 2019. ISSN 2363-7501. doi: 10.1186/s40965-019-0064-0.

J. Liang and J. Gong. A sparse voxel octree-based framework for computing solar radiation using 3D city models. *ISPRS International Journal of Geo-Information*, 6(4), 2017. ISSN 2220-9964. doi: 10.3390/ijgi6040106.

J. Liang, J. Gong, W. Li, and A. N. Ibrahim. A visualization-oriented 3D method for efficient computation of urban solar radiation based on 3D–2D surface mapping. *International Journal of Geographical Information Science*, 28(4):780–798, 2014. ISSN 1365-8816 1362-3087. doi: 10.1080/13658816.2014.880168.

J. Liang, J. Gong, J. Zhou, A. N. Ibrahim, and M. Li. An open-source 3D solar radiation model integrated with a 3d geographic information system. *Environmental Modelling Software*, 64:94–101, 2015. ISSN 13648152. doi: 10.1016/j.envsoft.2014.11.019.

J. Liang, J. Gong, X. Xie, and J. Sun. Solar3D: An open-source tool for estimating solar radiation in urban environments. *ISPRS International Journal of Geo-Information*, 9(9), 2020. ISSN 2220-9964. doi: 10.3390/ijgi9090524.

NVIDIA. GPU-Accelerated Computing with Python. URL `https://developer.nvidia.com/how-to-cuda-python`.

T. E. Oliphant. NumPy, 2021. URL `https://pypi.org/project/numpy/`.

M. Pharr, W. Jakob, and G. Humphreys. *Physically Based Rendering: Primitives and Intersection Acceleration*. 2004 - 2021.

F. Pinde and P. Rich. A geometric solar radiation model with applications in agriculture and forestry. 2002.

P. M. Redweik, C. Catita, and M. C. Brito. 3D local scale solar radiation model based on urban lidar data. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XXXVIII-4/W19:265–269, 2012. ISSN 2194-9034. doi: 10.5194/isprsarchives-xxxviii-4-w19-265-2011.

X. Song, Y. Huang, C. Zhao, Y. Liu, Y. Lu, Y. Chang, and J. Yang. An approach for estimating solar photovoltaic potential based on rooftop retrieval from remote sensing images. *Energies*, 11(11), 2018. ISSN 1996-1073. doi: 10.3390/en11113172.

N. Stendardo, G. Desthieux, N. Abdennadher, and P. Gallinelli. GPU-enabled shadow casting for solar potential estimation in large urban areas. application to the solar cadaster of greater Geneva. *Applied Sciences*, 10(15), 2020. ISSN 2076-3417. doi: 10.3390/app10155361.

J. D. Viana-Fons, J. Gonzálvez-Maciá, and J. Payá. Development and validation in a 2D-GIS environment of a 3D shadow cast vector-based model on arbitrarily orientated and tilted surfaces. *Energy and Buildings*, 224, 2020. ISSN 03787788. doi: 10.1016/j.enbuild.2020.110258.

I. Wald, S. Boulos, and P. Shirley. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Transactions on Graphics*, 26(1):6, 2007. ISSN 0730-0301. doi: 10.1145/1189762.1206075.

M. Wieland, A. Nichersu, S. M. Murshed, and J. Wendel. Computing Solar Radiation on CityGML Building Data. 2015.

A. Williams, S. Barrus, R. K. Morley, and P. Shirley. *An efficient and robust ray-box intersection algorithm*. 2005. doi: 10.1145/1198555.1198748.