SoftwareX 34 (2026) 102659

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original Software Publication

SWAN: An open-source 3D shadow analysis tool for cities

Ziya Usta

3D Geoinformation, Urban Data Science, Delft University of Technology, 2628 Delft, Netherlands

ARTICLE INFO

Keywords:
Shadow analysis
3D city models
3D Analysis
Urban planning
GIS

ABSTRACT

SWAN (**S**hado**W** **AN**alysis for Cities) is an open-source, platform-independent Python tool designed for high-fidelity 3D shadow analysis using CityJSON-format 3D city models. Unlike proprietary or limited tools, SWAN performs tessellation of building surfaces and applies ray-tracing via Open3D to compute shadow durations with fine spatial and temporal granularity. The software integrates results into PostGIS, enabling advanced spatial queries for urban planning, solar energy optimization, and microclimate studies. With modular architecture, reproducibility, and MIT licensing, SWAN provides a reusable and extensible solution for researchers, planners, and architects, advancing urban sustainability and supporting data-driven decisions in smart city applications.

Metadata

Nr	Code metadata description	Metadata
C1	Current code version	v1.0
C2	Permanent link to code/repository used for this code version	https://github.com/ZiyaUsta/SWAN
C3	Permanent link to reproducible capsule	NA
C4	Legal code license	MIT License.
C5	Code versioning system used	git,
C6	Software code languages, tools and services used	Python
C7	Compilation requirements, operating environments and dependencies	Python 3.10, astral 3.2, Numpy 2.0.1, open3d 0.19.0, psycopg2 2.9.10, pyproj 3.7.1, shapely 2.1.1, tqdm 4.67.1
C8	If available, link to developer documentation/manual	https://github.com/ZiyaUsta/SWAN/blob/main/README.md
C9	Support email for questions	

1. Motivation and significance

The estimation of shadows cast by buildings is a common topic in geoinformation science, and it is important for a number of application domains, such as thermal comfort [1] and solar energy [2]. According to Herbert and Chen [3], understanding shadows is a critical aspect of urban planning, as it enables the assessment of how new developments affect surrounding buildings and Xu et al. [4] states that 3D City Models are provide a structured dataset crucial for calculating the shadowing effect. By anticipating the shadow effect caused by nearby buildings, planners, architects, and engineers can make data-driven decisions to enhance urban liveability and optimize solar energy systems. Shadow analysis serves multiple purposes in geoinformation science. While 3D city models are used to simulate shadows for comfort and energy studies, the reverse approach is also common in remote sensing. Several studies have utilized shadow lengths from satellite or aerial imagery to estimate the height of [5–7] and vegetation [8]. Furthermore, the application of shadow-based height estimation extends beyond buildings to diverse urban and rural features. For instance, high-precision measurements of height differences have been conducted using shadows in non-stereo imagery [9]. Similar methodologies have been applied to estimate the heights of vegetation, such as olive trees in GIS-based agricultural studies [10], and to measure infrastructure components like power pylons using high-resolution satellite data [11]. Unlike these reconstruction-focused methods, SWAN leverages existing semantic 3D city models (CityJSON) to perform high-precision forward shadow simulation for urban planning.

Shadow estimation in an urban environment represents a fundamental component of 3D GIS and is employed in diverse domains [12]; however, 3D shadow analysis tools are quite limited. While shadow analysis tools are not new, the proprietary, closed-source nature of existing solutions has constrained their accessibility for wider use by decision makers. Furthermore, some software components only work in specific environments. Yue et al. [13] and Xu et al. [14] use Rhino3D parametric modelling software along with extensions of the Grasshopper and Ladybug simulation tools for the evaluation of solar radiation on building surfaces. In this software chain, Rhino3D is commercial, and Grasshopper does not work on Mac OS. Machete et al. [15] use Ecotect

E-mail addresses: z.usta@tudelft.nl, ziyausta@artvin.edu.tr.

https://doi.org/10.1016/j.softx.2026.102659
Received 1 October 2025; Received in revised form 8 April 2026; Accepted 14 April 2026
Available online 19 April 2026

for 3D Solar Analysis, which is no longer available. A widely used alternative open-source solution is the 'r.sun' solar radiation analysis module of the GRASS, open-source GIS software [16]. However, the majority of GIS software employs 2.5D surface models and has limited capabilities. In such models, data is represented as continuous surface and loss semantic information hence, vertical surfaces cannot be identified, and therefore, building facades can not be analysed. To address this limitation, the v.sun module has been developed by Hofierka and Zlocha [17]. Although it is claimed to be open-source and a module of GRASS, the source code of the v.sun module has not been made publicly available, and the module is not included in GRASS releases. Therefore, it cannot be considered an open-source and reusable solution.

Xu et al. [18] developed a method for 3D shadowing calculation using 3D city models and the Ray-Tracing algorithm. They implemented their solution in C++ along with well-known algorithms for optimization, such as Bounding Volume Hierarchies [19] and Nightside Filtering [20]. Although the implementation is clearly described, it has not been published in any accessible form, thereby making the shadow calculation methodology non-reproducible and non-reusable. Shadow Analysis software by DeltaCodes [21] is another shadow analysis tool which is developed as a commercial plug-in for SketchUp modeling software. Shadowmap [22] and ShadeMap [23] are both web applications that visualize real-time shadows for a specific location. Additionally, Shadowmap has also analysis capabilities. The major drawback of these two tools is that they use Open Street Map (OSM) generated 3D models which are not correct if the height information is missing in the OSM. Waqas et al. [24] calculates solar radiation includes shadow calculation using ESRI ArcGIS Pro 3.2 commercial software. SolarQGIS, introduced by Ilba [25], is an open-source QGIS plugin that calculates solar radiation on arbitrary 3D geometry. The main limitation is that SolarQGIS does not tessellate the 3D polygon surfaces into smaller surfaces and performs the analysis across each surface as a whole. However, while one part of a surface may receive greater exposure to sunlight, another part may be more affected by shading. Consequently, solar radiation is not distributed uniformly across polygonal surfaces. In particular, considering that the average size of solar panels is approximately 2 m $\times$ 2 m, surfaces larger than these dimensions need to be subdivided into smaller units to allow for a fine-grained analysis.

In general, an examination of the existing software components for 3D shadow analysis reveals that many are not open source, operate only on specific platforms, do not support 3D city model formats such as CityGML or CityJSON, hindering the widely use of 3D shadow analysis in decision-making processes. Additionally most of the existing software generate a rendered image (URL-1) or visualize results. Thus, results can be seen but cannot be queried. To bridge this gap, SWAN, an open-source 3D shadow analysis tool, has been developed.

SWAN (Shadow Analysis for Cities) addresses these gaps by providing an open-source, platform-independent Python tool that supports CityJSON-format 3D city models. It performs high-resolution shadow analysis by tessellating building surfaces and computing shadow durations using ray-tracing, leveraging the Open3D library. SWAN's accessibility and reproducibility make it a valuable tool for urban planners, architects, and researchers seeking data-driven decisions without reliance on costly or platform-specific software. Furthermore, SWAN imports results into a PostGIS table along with the 3D city model with the surface type information. Therefore, complex queries can be performed to for urban analysis purposes after the shadow analysis such as how much of the roof surfaces remain in shadow or what percentage of the building facades are shaded.

2. Software description

SWAN is a general-purpose Python tool designed to compute shadow durations on 3D building surfaces within CityJSON-format 3D city models. It processes building geometries, tessellates surfaces into a grid of points (e.g., 2 m spacing), and calculates shadow durations over user-specified time periods using solar position data and ray-tracing. SWAN is platform-independent, running on Windows, Linux, and macOS, and is released under the MIT License to ensure accessibility and reusability.

2.1. Software architecture

SWAN's architecture is modular, with distinct components for data input, processing, and visualization. The software is structured into six modular classes, each handling a distinct aspect of the shadow analysis pipeline, designed with the Single Responsibility Principle in mind (Fig. 1). Six classes distinctly separate the logical components of the code. Each class is independently testable and reusable:

CityJSONLoader: Reads CityJSON files, extracting building geometries and semantic surfaces such as "RoofSurface" and "WallSurface". The CityJSONLoader class is responsible for loading CityJSON files into memory and extracting geographical metadata, ensuring robust input validation. It computes the midpoint coordinates from the "geographicalExtent" and retrieves the coordinate reference system (CRS), which are critical for accurate solar calculations. The extraction of the CRS is a critical step in the SWAN workflow to bridge the gap between local 3D city model geometries and global astronomical calculations. While georeferenced CityJSON files typically use local projected Cartesian coordinate systems for metric accuracy such as RD New for Rotterdam, solar position algorithms—such as those implemented in the astral library—require geographic coordinates Latitude/Longitude in WGS84 as input. SWAN uses the CRS metadata to perform two essential transformations: first, it converts the local center coordinates of the model into WGS84 to compute precise solar azimuth and altitude angles for the specific location; and second, it ensures that the resulting 3D solar vectors are correctly aligned with the orientation of the building geometries for the ray-tracing process. This synchronization prevents spatial discrepancies between the sun's calculated position and the static 3D model, ensuring the accuracy of the shadow duration results. Without this step, solar vectors would be inaccurate for models in non-geographic CRS.

GeometryProcessor: Converts 3D surfaces to 2D planes, tessellates them into a grid, and maps points back to 3D space. The GeometryProcessor class handles geometric computations, including plane equation derivation, 2D projection, and 3D surface point generation. It processes building surfaces from CityJSON data, preparing point clouds for shadow analysis.

SunDirectionCalculator: Uses the Astral library to compute hourly solar directions based on location and date range. The SunDirectionCalculator class computes hourly solar direction vectors for a given date range, using the astral library for precise astronomical calculations. It integrates coordinate transformations to ensure location-specific accuracy.

ShadowAnalyzer: Employs Open3D's ray-tracing to detect intersections between solar rays and building geometries, excluding self-intersections. The ShadowAnalyzer class performs ray-tracing using Open3D to detect shadow intersections on building surfaces. It computes the daily average shadow duration by aggregating intersection results over the specified date range.

Visualizer: Saves results as JSON files and visualizes buildings and shadow points using Open3D, with color002Dcoded shadow durations. The Visualizer class generates a JSON output of shadow analysis results and visualizes building surfaces and shadow points using Open3D. A color gradient (green to red) represents the daily average shadow duration, enhancing human interpretability.

PostGISExporter: The PostGISExporter class is a pivotal component of the software, enabling integration of CityJSON data and shadow analysis results into a PostGIS database for efficient storage, querying, and spatial analysis. Designed to handle geospatial data with high flexibility, it supports two primary tables: cityobjects and surface_points

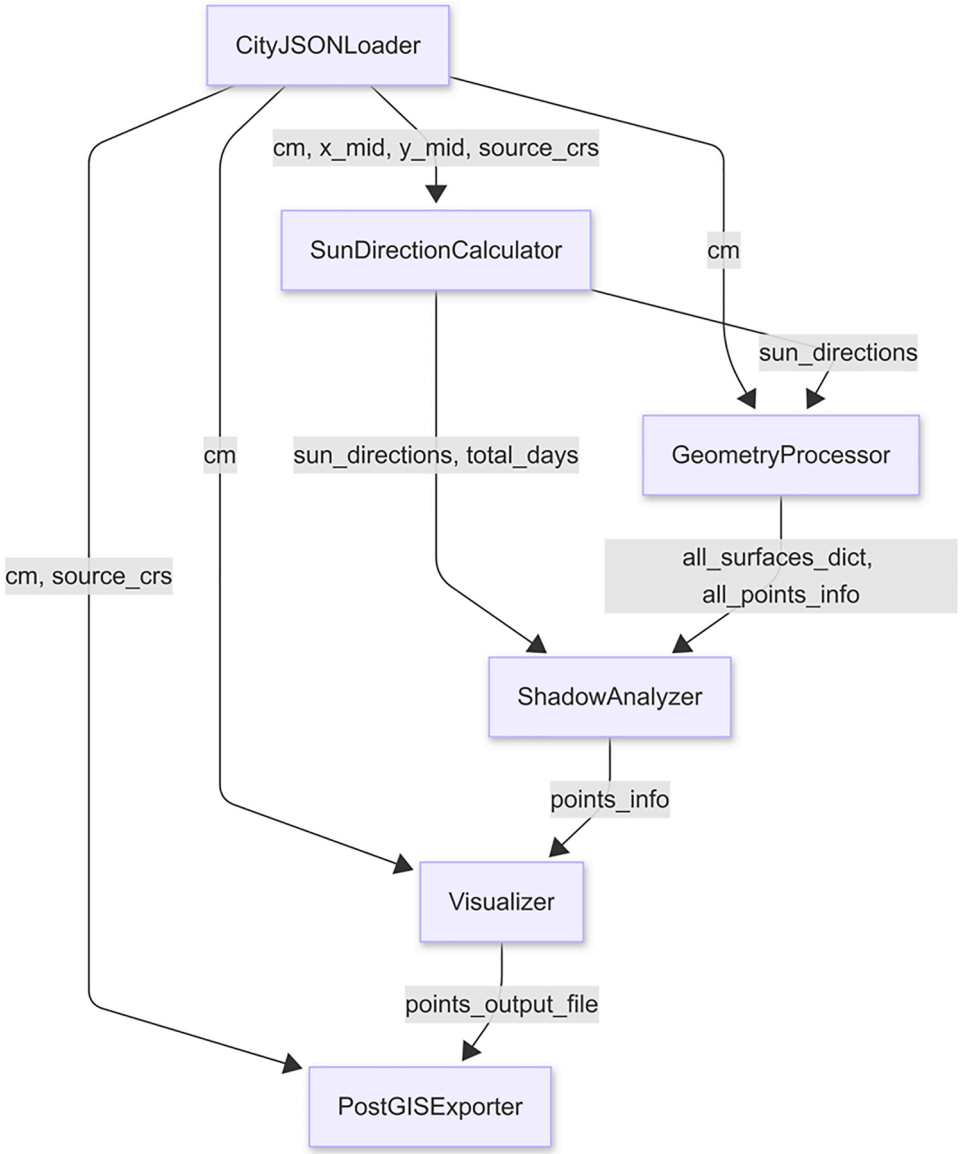

Fig. 1. Components of the SWAN.

2.2. *Software functionalities*

The software first parses the CityJSON file and loads it into memory. It reads the extent of the city model from a file and calculates the centre coordinates of the city model for the calculation of solar azimuth and altitude angles. Additionally, it extracts the coordinate reference system (CRS) of the city model for correct coordinate transformation. The user can define a grid in meters, and based on the user input in the main.py,

SWAN generates an equally spaced regular point grid on 3D surfaces (Fig. 2). While generating points on surfaces, ground surfaces are filtered and not taken into account in shadow analysis. This also reduces the number of points to be generated and, consequently, the number of intersection tests to be performed for the points.

Fig. 2 illustrates the fundamental mechanism of the shadow analysis process within the SWAN framework. It displays a segment of a 3D building facade where the surface has been discretized into a grid of sensor points, represented by red dots, through the tessellation process. From each sensor point, a solar vector, indicated by the blue lines, is projected toward the sun's calculated position for a specific time step. The ray-tracing engine then evaluates these vectors to determine if they intersect with any surrounding geometry, such as adjacent buildings or other parts of the same structure. A clear path for a vector indicates that the sensor point is in direct sunlight, while an intersection signifies that the point is in shadow for that particular moment. This visualization clarifies how the software transitions from static 3D geometry to dynamic, point-based shadow duration calculations.

Using centre coordinates and CRS, SWAN calculates solar azimuth and altitude angles for a given time. Using azimuth and altitude angle, it calculates the direction vector towards to sun and, using the direction vector, it implements ray-tracing using Open3D to determine whether the vector intersects with any other surfaces. Based on this intersection test, the shadow is determined if there is an intersection. Thus, for each time step between the sunrise and sunset hours intersection test is repeated and shadow values are recorded. Hours from sunset until sunrise of the next day are excluded because there is no sun in the sky at those times. This is required for the accuracy of the analysis. For example, the shadow analysis software (URL-1) leaves the selection of the time interval to the user, and if the user enters incorrect sunrise and sunset times, the results will also be incorrect. Additionally, excluding the times that the sun is not in the sky also reduces the number of intersection tests and improves performance. SWAN does this filtering automatically using the Astral Python library. This eliminates the need

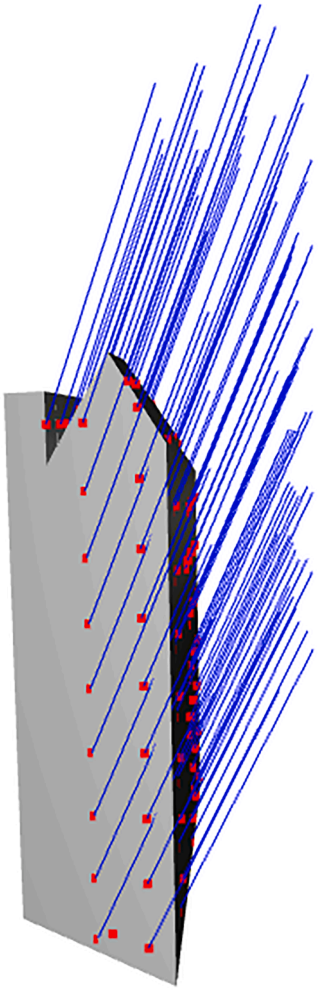

Fig. 2. Point grid on building surfaces and sun rays for a given time step. The red points represent the tessellated sensor grid on the building facade, while the blue lines indicate the solar vectors projected towards the sun's position for intersection testing.

for the user to know the sunrise and sunset times for the relevant days and ensures the correct time steps for the analysis. The general workflow of SWAN can be found in Fig. 3 and pseudo-code for SWAN in Annex 1.

Once the daily average shadow hours are determined, the results are visualized using the Open3D library by the SWAN (Fig. 4). Finally, for each point, SWAN saves the results into a JSON file and exports the results along with the 3d city model to a PostgreSQL database with PostGIS extension. Thus, users can integrate results with a database and perform advanced queries on the results. SWAN also stores the surface type, such as "WallSurface" or "RoofSurface", along with points, and the user can analyse this information as well. For the database connection user must change "db_params" accordingly in the main.py file.

3. Illustrative examples

To ensure a reproducible environment, SWAN manages its dependencies using Conda, an open-source package and environment management system that simplifies the installation of complex spatial libraries across different operating systems. By using Conda, a dedicated virtual environment can be created to isolate the specific versions of Python and libraries required for SWAN, preventing conflicts with other system-wide installations.

Using conda, a new environment can be created and dependencies can be installed easily using a requirements.txt file as following

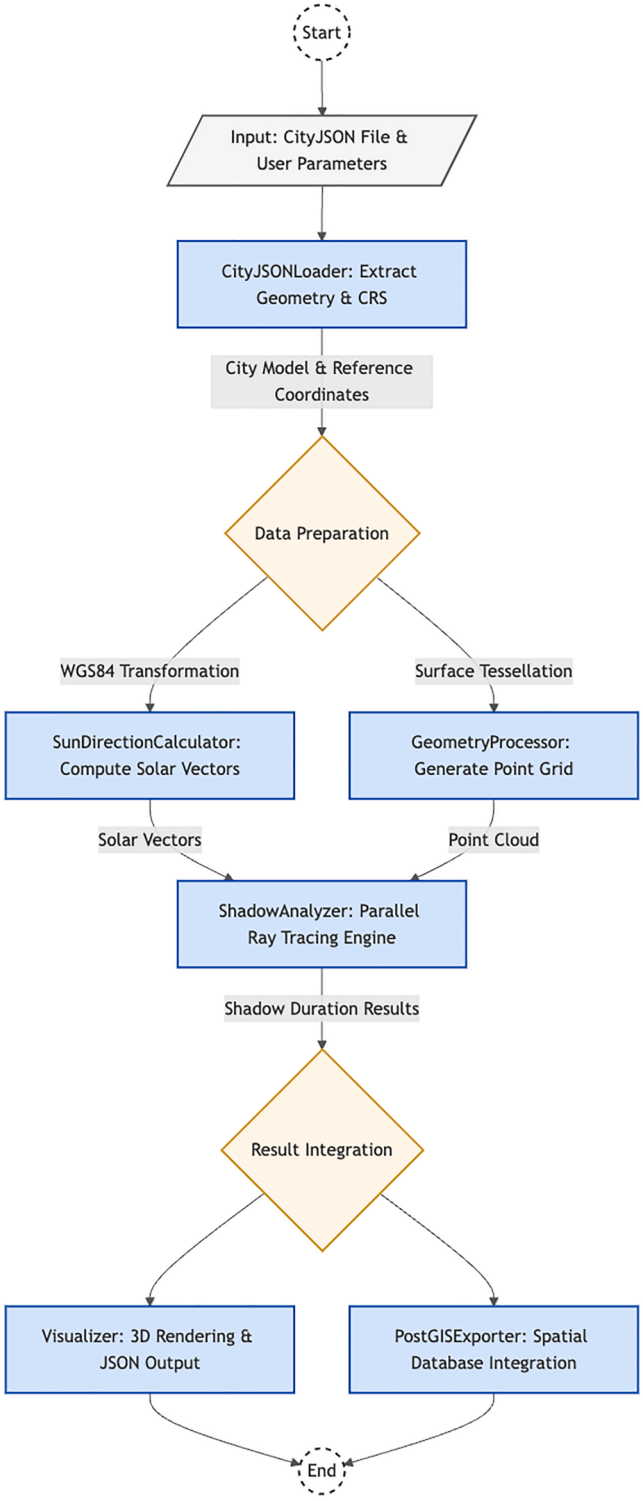

Fig. 3. Workflow diagram of the SWAN.

expression "conda install –file requirements.txt". After installing dependencies, in main.py file users can set the parameters and run the software (Listing 1). After importing required classes, users can set input 3D city model file, spacing for point grid generation, output file, start date, end date, hour step and database connection parameters. Note that database connection is optional and without it SWAN also work and visualize results and store shadow values in output file. If database connection provided, SWAN also generate two spatial tables in PostgreSQL database for further advanced queries (Fig. 5).

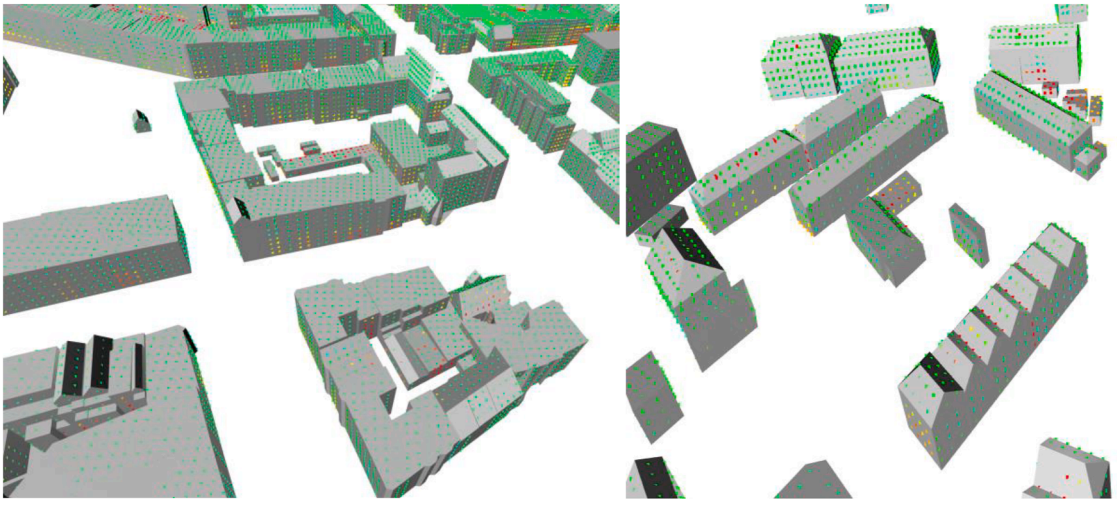

Fig. 4. Shadow analysis results for Rotteram model (left) and Den Haag model (right).

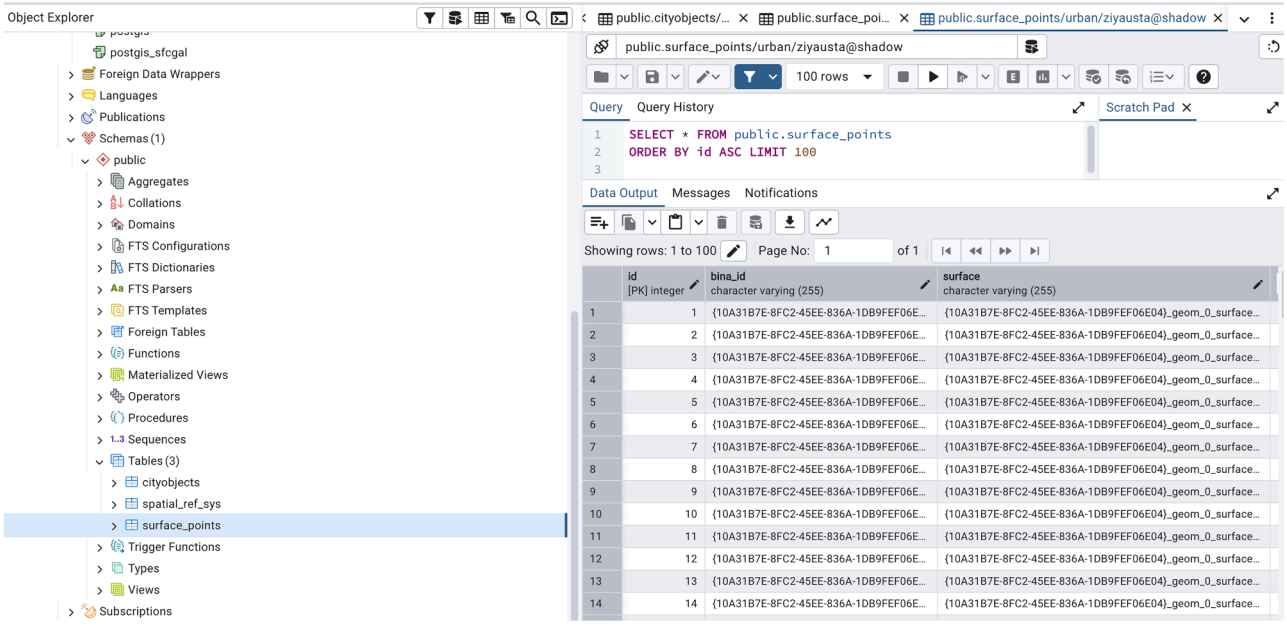

Fig. 5. SWAN generates “cityobjects” and “surface_points” spatial tables.

Listing 1. Usage of the SWAN software.

4. Validation and performance evaluation

To assess the reliability and accuracy of SWAN's shadow estimation algorithm, a cross-validation study was conducted using **Ladybug Tools** (for Rhino/Grasshopper), which is widely recognized as an industry-standard engine for environmental analysis in architecture and urban design.

For the validation setup, a representative urban blocks from the Rotterdam 3D city model and Den Haag 3D city model were selected to test complex shadowing scenarios, including self-shadowing and cast shadows from neighboring structures. To ensure a fair comparison, both software environments were configured with identical boundary conditions:

- **Geometry:** The same CityJSON building geometries were imported into Rhino as Mesh objects. Surface normals were unified to ensure correct ray-intersection directionality.
- **Geolocation:** Rotterdam with the UTC+1 time zone and Den Haag with the UTC+1 time zone
- **Temporal Resolution:** A simulation period was set between 15.01.2025–20.01.2025
- **Spatial Resolution:** A sensor grid spacing of 2.0 m was applied in both SWAN (using GeometryProcessor) and Ladybug (using LB Generate Point Grid).

Since, Ladybug calculates direct sun hours, The primary metric for comparison was "Direct Sun Hours", the inverse of shadow duration. To ensure clarity in the analysis results, it is essential to distinguish between the two primary metrics used in this study: shadow duration and direct sun hours.

Shadow Duration: Defined as the total cumulative time a specific sensor point remains in shadow due to obstructions from neighboring geometries during the analysis period.

Direct Sun Hours: Represents the total duration for which a sensor point is directly exposed to solar radiation.

Mathematically, these two metrics are inverse components of the total potential sunshine hours, the period between sunrise and sunset. SWAN explicitly calculates shadow duration by detecting ray-geometry intersections, and the resulting 'direct sun hours' are derived by subtracting the total shadowed intervals from the available daylight period to compare with Ladybug. This distinction is critical for applications such as solar panel optimization, where direct sun hours are the primary interest, versus thermal comfort studies, where shadow duration is the key variable. The analysis results from both tools were compared point-by-point.

5. Results

The comparison revealed a high degree of correlation between SWAN and Ladybug. As shown in Fig. 6. The shadow patterns generated by SWAN are visually indistinguishable from those produced by Ladybug. The minor discrepancies observed are attributed to the slight differences in the solar vector calculation algorithms used by the underlying libraries (Astral for Python vs. NOAA vectors in Ladybug) and floating-point precision in ray-intersection tests.

This validation confirms that SWAN's open-source, CityJSON-based workflow achieves accuracy levels comparable to established proprietary solutions, making it a reliable tool for scientific urban studies.

To assess the scalability and resource efficiency of SWAN, performance tests were conducted using varying dataset sizes from the Rotterdam 3D city model (100, 500, and 853 buildings) and Den Haag 3D city model (100, 500, 844 builddings). The tests were performed on a laptop with 32GB Ram and M3 Pro Max CPU, simulating a 5-day period (January 15–20) with hourly intervals and a grid spacing of 2.0 m As shown in Tables 1, and 2 the software demonstrates efficient memory management and rapid processing times. For the Rotterdam dataset of 853 buildings, the total analysis time was recorded at **27.0 s**, with a peak memory usage of only **96.2 MB**. And, for the Den Haag dataset of 844 buildings, the total analysis time was recorded at **7.52 s**, with a peak memory usage of only **43.19 MB**. The baseline memory consumption remains low, reflecting the minimal overhead of the Python libraries. These results indicate that SWAN is highly optimized for desktop environments, enabling researchers to perform district-scale shadow analyses without the need for high-performance computing infrastructure.

In addition to result consistency, the scalability of SWAN was evaluated to determine its suitability for large-scale urban applications. The performance results presented in Table 1 indicate a sub-linear increase in runtime relative to the number of buildings, suggesting that the ray-tracing engine integrated with Open3D efficiently handles increasing geometric complexity. While the peak memory usage for 873 buildings remained below 100 MB, the software's modular architecture and spatial indexing enhance scalability ensuring that SWAN remains a robust tool for diverse urban digital twins.

To provide a more comprehensive evaluation of SWAN relative to

Table 1
Performance evaluation of SWAN with Rotterdam dataset.

Number of Buildings	Peak Memory Usage (MB)	Runtime (sec)
100	58.5	3.8
500	74.1	14.2
853	96.2	27.0

Table 2
Performance evaluation of SWAN with Den Haag dataset.

Number of Buildings	Peak Memory Usage (MB)	Runtime (sec)
100	26.26	1.05
500	33.26	3.95
844	43.19	7.52

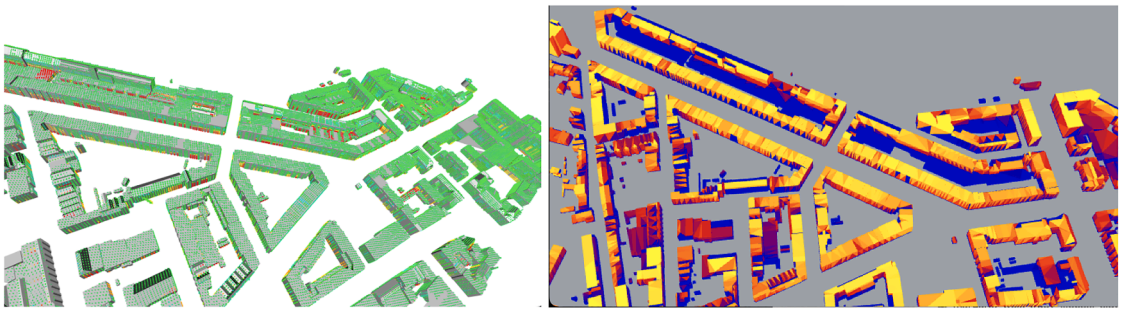

Fig. 6. Comparison of results SWAN (left) and Ladybug (right).

existing tools, we compare its computational efficiency, scalability, and suitability for large-scale urban scenarios with several prominent alternatives, including Ladybug Tools (used in the validation), the r.sun module in GRASS GIS, SolarQGIS, and selected commercial/proprietary solutions.

SWAN demonstrates strong performance on desktop hardware, completing shadow analysis for 873 buildings over a 5-day period (hourly intervals, 2 m grid spacing) in only 27 s with peak memory usage of 96.2 MB (Table 1). This efficiency stems from Open3D's optimized ray-tracing engine, which benefits from spatial indexing and efficient intersection tests, combined with automatic filtering of nighttime hours via the Astral library to reduce unnecessary computations. In contrast, Ladybug Tools — while achieving comparable accuracy in direct sun hours (as shown in Fig. 6) — typically requires conversion of CityJSON geometries to Rhino/Grasshopper meshes and operates within a parametric environment that can introduce overhead for very large models. SWAN's native Python implementation and platform independence thus offer advantages in workflow simplicity and scalability for district- to neighborhood-level analyses without requiring commercial CAD software.

Compared to 2.5D raster-based tools such as r.sun in GRASS GIS, SWAN provides full 3D semantic support (including vertical facades) at the cost of higher per-point computation due to ray-tracing. However, r.sun's raster approach is generally faster for large horizontal surfaces but loses accuracy on complex urban morphologies with overhangs, self-shadowing on walls, or tilted roofs — scenarios where SWAN excels. Extensions like Solar3D or v.sun attempt to bridge this gap but remain limited in availability or integration. Similarly, SolarQGIS performs well on smaller scales but lacks surface tessellation, leading to uniform treatment of large polygonal faces and reduced precision for applications like solar panel placement. SWAN scales sub-linearly and remains practical for urban district analyses without high-performance computing resources.

In summary, SWAN balances accuracy, accessibility, and efficiency for semantic 3D city model workflows. It outperforms many open-source alternatives in handling full 3D geometry and enabling post-analysis spatial querying via PostGIS, while maintaining lower resource demands compared to heavy parametric tools like Ladybug.

The current architecture is optimized for high-performance CPU-based ray-tracing. During development, explicit application-level multi-threading was evaluated; however, it was found that manual threading in Python introduced significant overhead due to the Global Interpreter Lock (GIL) and context switching, often resulting in longer execution times than single-process execution. To overcome this, SWAN is designed to process rays in large bulk tensors, allowing it to leverage Open3D's underlying C++ kernels powered by Intel Embree. This design allows the software to execute ray-tracing as a native, thread-safe, and highly optimized process at the C++ level. By feeding rays in large bulk tensors, SWAN achieves high-efficiency parallel execution across all available CPU cores, regardless of the operating system or the presence of a CUDA-enabled GPU.

This architectural choice ensures that SWAN remains a lightweight and robust tool for the majority of urban planning scenarios. For instance, the analysis of 873 buildings was completed in just 27.0 s on a standard laptop. The current CPU-optimized implementation was prioritized to guarantee that high-fidelity 3D shadow analysis remains accessible to all researchers without hardware-imposed barriers. Future extensions could incorporate GPU support for even ultra-large datasets, but the current CPU-optimized approach ensures broad accessibility, especially for machines or operating systems that do not have CUDA support.

6. Impact

SWAN addresses critical gaps in 3D urban shadow analysis by providing an open-source, platform-independent framework for high-fidelity simulation and data management. The software's impact is demonstrated through its core functional capabilities:

Unlike traditional 2.5D tools such as r.sun [16], which cannot represent vertical surfaces, SWAN preserves semantic information from CityJSON models such as "WallSurface" and "RoofSurface". This enables precise facade-level shadow assessments, which are critical for solar potential evaluations [2] and microclimate studies where vertical shading is a dominant factor [4]. The use of fine-grained tessellation (2 m grids) ensures that shadow variations across large surfaces are accurately mapped, avoiding the errors inherent in treating entire polygons as uniform units.

The PostGIS integration allows persistent storage of CityJSON geometries (as MULTIPOLYGON Z) and analysis points (as POINT Z), facilitating complex spatial queries—e.g., identifying surfaces with high shadow exposure during specific hours or periods, calculating the percentage of shaded roof area across districts, or evaluating solar potential variations due to building geometry and orientation. Such queries directly support evidence-based decisions in urban planning, such as optimizing building placements for improved solar access or informing energy-efficient retrofits and microclimate studies.

The software's performance—processing 873 buildings in 27 s with under 100 MB of memory—demonstrates that district-scale 3D analysis is achievable on standard desktop hardware. By removing reliance on proprietary, platform-restricted software such as Rhino3D/Grasshopper, SWAN provides a reproducible and extensible tool for municipalities and researchers, supporting data-driven decisions in urban sustainability and digital twin development.

7. Conclusions

SWAN introduces a fully open-source, reproducible, extensible, and platform-independent framework for 3D shadow analysis using CityJSON-based 3D city models. By leveraging tessellation, ray-tracing, and PostGIS integration, the software addresses long-standing limitations of proprietary and platform-restricted tools. Its modular architecture ensures flexibility, while database connectivity enables advanced spatial queries that support urban planning, solar energy assessment, and sustainability research. SWAN serves as a foundation for Urban Analytics studies and can be extended in that way in future studies.

Funding

This study was supported by the Scientific Research Projects Office of Artvin Çoruh University (AÇÜBAP) (Scientific Research Project No 2023. F40.02.01) and the Scientific and Technological Research Council of Türkiye (TÜBİTAK) with application numbers 1059B192402172.

CRediT authorship contribution statement

Ziya Usta: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

```python
from cityjson_loader import CityJSONLoader
from geometry_processor import GeometryProcessor
from sun_direction_calculator import SunDirectionCalculator
from shadow_analyzer import ShadowAnalyzer
from visualizer import Visualizer
from postgis_exporter import PostGISExporter
from astral import LocationInfo
import time

def main():
    input_file = "Rotterdam.city.json"
    spacing = 2.0
    points_output_file = "all_surface_points_with_shadow.json"
    start_date = "2025-01-15"
    end_date = "2025-01-20"
    hour_step = 1
    db_params = {
        "dbname": "your_db_name",
        "user": "your username",
        "password": "your password",
        "host": "localhost",
        "port": "5432"
    }

    start_time = time.time()
    try:
        # Load CityJSON
        cm, x_mid, y_mid, source_crs = CityJSONLoader.load_cityjson(input_file)
        location_info = LocationInfo("Rotterdam", "Netherlands", "Europe/Amsterdam", 0, 0)

        # Calculate solar directions
        sun_directions, total_days = SunDirectionCalculator.get_hourly_sun_directions(
            start_date, end_date, location_info, hour_step=hour_step, x_mid=x_mid, y_mid=y_mid
        )

        # Process building surfaces
        all_surfaces_dict, all_points_info = GeometryProcessor.process_all_buildings_surfaces(
            cm, sun_directions, spacing=spacing
        )

        if all_points_info:
            # Perform shadow analysis
            points_info = ShadowAnalyzer.check_all_intersections(cm, all_points_info, sun_dire
            print(f"Total runtime: {time.time() - start_time:.2f} seconds.")
            # Save and visualize results
            Visualizer.save_points_info_with_shadow(points_info, points_output_file)
            Visualizer.visualize_all_buildings(cm, points_info)

            # Export to PostGIS Please ensure that PostGIS is properly set up and the database
            exporter = PostGISExporter(db_params)
            try:
                success_count = exporter.export_cityobjects(cm, source_crs)
                print(f"{success_count} obje impoted into cityobjects table")
                exporter.export_surface_points(points_output_file, source_crs=source_crs)
                print(f"Surface_points table created")
            finally:
                exporter.close_connection()

    except Exception as e:
        print(f"Error: {str(e)}")

if __name__ == "__main__":
    main()
```

Appendix-1 Pseudo-code of SWAN

```
Algorithm 1: SWAN Computational Workflow
Input:
C_json: 3D City Model (CityJSON file)
T_start: Start date for analysis
T_end: End date for analysis
d_space: Grid spacing (e.g., 2.0 m)
DB_conf: PostGIS connection parameters (optional)
Output:
P_shadow: 3D Points with shadow duration attributes
DB_tbl: PostGIS tables ('cityobjects', 'surface_points')
Procedure:
function MAIN(C_json, T_start, T_end, d_space)
model, crs, center ← CityJSONLoader.load(C_json)

// Step 1: Calculate Sun Vectors
V_sun ← SunDirectionCalculator.get_vectors(T_start, T_end, center)
FILTER V_sun where solar_altitude > 0

// Step 2: Geometry Processing & Tessellation
S_surfaces ← EXTRACT surfaces from model
P_grid ← ∅
for each surface in S_surfaces do
points ← GeometryProcessor.tessellate(surface, d_space)
ADD points to P_grid
end for

// Step 3: Ray Tracing Analysis (Core)
for each vector in V_sun do
rays ← CONSTRUCT rays from P_grid along vector
hits ← ShadowAnalyzer.ray_trace(rays, model.geometries)
UPDATE shadow_counters for P_grid based on hits
end for

// Step 4: Export Results
Visualizer.save_and_render(P_grid)
if DB_conf is valid then
PostGISExporter.export(model, P_grid, crs)
end if
end function
```

References

[1] Hwang R-L, Lin T-P, Matzarakis A. Seasonal effects of urban street shading on long-term outdoor thermal comfort. Build Environ 2011;46(4):863–70.

[2] Strzalka A, Bogdahn J, Coors V, Eicker U. 3D City modeling for urban scale heating energy demand forecasting. HVAC&R Res 2011;17(4):526–39.

[3] Herbert G, Chen X. A comparison of usefulness of 2D and 3D representations of urban planning. Cartogr Geogr Inf Sci 2015;42(1):22–32.

[4] Xu L, León-Sánchez C, Agugiaro G, Stoter J. High resolution solar potential computation in large scale urban areas by means of semantic 3D city models. The International Archives of the Photogrammetry. Remote Sens Spat Inf Sci 2024;48:167–74.

[5] Liasis G, Stavrou S. Satellite images analysis for shadow detection and building height estimation. ISPRS J Photogramm Remote Sens 2016;119:437–50.

[6] Qi F, Zhai J, Dang G. Building height estimation using Google Earth. Energy Build 2016;118:123–32.

[7] Cao Y, Huang X. A deep learning method for building height estimation using high-resolution multi-view imagery over urban areas: a case study of 42 Chinese cities. Remote Sens Environ 2021;264:112590.

[8] Abdollahnejad A, Panagiotidis D. Tree species classification and height estimation using low-cost UAV technology. Remote Sens 2020;12(12):1964.

[9] Rada Giacaman CA. High-precision measurement of height differences from shadows in non-stereo imagery: new methodology and accuracy assessment. Remote Sens 2022;14(7):1702.

[10] Brigante R, Baiocchi V, Calisti R, Marconi L, Proietti P, Radicioni F, et al. GIS-based approach for estimating olive tree heights using high-resolution satellite imagery and shadow analysis. Appl Sci 2025;15(6):3066.

[11] Mi X, Yu T, Yang J, Lai J, Zhang Z, Zhang Y, Zhan Y. Estimating pylon height using differences in shadows between GF-2 images. J Indian Soc Remote Sens 2019;47(2):279–88.

[12] Biljecki F, Ledoux H, Stoter J. Does a finer level of detail of a 3D city model bring an improvement for estimating shadows? In: Abdul-Rahman A, editor. **Advances in 3D Geoinformation**. Cham: Springer; 2016. p. 31–47. https://doi.org/10.1007/978-3-319-25691-7_2.

[13] Yue Y, Yan Z, Ni P, Lei F, Qin G. Promoting solar energy utilization: prediction, analysis and evaluation of solar radiation on building surfaces at city scale. Energy Build 2024;319:114561.

[14] Xu J, Qi M, Jing H, Hancock C, Qiao P, Shen N. A real scene 3D model-driven sunlight analysis method for complex building roofs. Energy Build 2024;325:115051.

[15] Machete R, Falcão AP, Gomes MG, Rodrigues AM. The use of 3D GIS to analyse the influence of urban context on buildings' solar energy potential. Energy Build 2018;177:290–302.

[16] Anselmo S, Safaeianpour A, Moghadam ST, Ferrara M. GIS-based solar radiation modelling for photovoltaic potential in cities: a sensitivity analysis for the evaluation of output variability range. Energy Rep 2024;12:4656–69.

[17] Hofierka J, Zlocha M. A new 3-D solar radiation model for 3-D city models. Trans GIS 2012;16(5):681–90.

[18] Xu L, León-Sánchez C, Agugiaro G, Stoter J. Shadowing calculation on urban areas from semantic 3D city models. In: Proc. Int 3D Geoinfo Conf; 2023. p. 31–47. Springer.

[19] Meister D, Ogaki S, Benthin C, Doyle MJ, Guthe M, Bittner J. A survey on bounding volume hierarchies for ray tracing. Comput Graph Forum 2021;40:683–712.

[20] Wang X, Zhang X, Zhu S, Ren J, Causone F, Ye Y, et al. A novel and efficient method for calculating beam shadows on exterior surfaces of buildings in dense urban contexts. Build Environ 2023;229:109937.

[21] DeltaCodes. Shadow Analysis. 2025. Retrieved from https://deltacodes.net/ (Accessed: August 17, 2025).

[22] Shadowmap. The sun for everyone – Sunlight & shadow analysis in 3D. 2025. Retrieved from https://shadowmap.org/ (Accessed: August 17, 2025).

[23] ShadeMap. Simulate sun shadows for any time and place on Earth. 2025. Retrieved from https://shademap.app/ (Accessed: August 17, 2025).

[24] Waqas H, Jiang Y, Shang J, Munir I, Khan FU. An integrated approach for 3D solar potential assessment at the city scale. Remote Sens 2023;15(23):5616.

[25] Ilba M. SolarQGIS: a QGIS application for calculating solar radiation on 3D vector GIS data. SoftwareX 2025;31:102230.