

# <sup>1</sup> openpv/simshady: A Javascript Package for Photovoltaic Yield Estimation Based on 3D Meshes

<sup>3</sup> **Florian Kotthoff**  <sup>1,2</sup>¶, **Konrad Heidler**  <sup>1</sup>, **Martin Großhauser**  <sup>1</sup>, **Mino Estrella**  <sup>3</sup>, and **Korbinian Pöppel**  <sup>1</sup>

<sup>5</sup> 1 OpenPV GbR, Munich, Germany 2 OFFIS Institute for Information Technology, Escherweg 2, 26121 Oldenburg, Germany 3 Chair of Renewable and Sustainable Energy Systems, Technical University of Munich, Germany ¶ Corresponding author

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

## Software

- [Review](#) ↗
- [Repository](#) ↗
- [Archive](#) ↗

Editor: [Open Journals](#) ↗

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

## License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)). To meet global climate targets, solar photovoltaic (PV) capacity must expand significantly. Tripling renewable energy capacity by 2030 is essential to limit global warming to 1.5°C ([International Energy Agency, 2023](#)). The expansion of PV plays a crucial role, and PV systems offer an additional benefit: small-scale house-mounted PV systems enable public participation and legitimize the energy transition.

## <sup>8</sup> Summary

<sup>9</sup> openpv/simshady is a JavaScript package for simulating photovoltaic (PV) energy yields. It <sup>10</sup> integrates local climate data and 3D objects into its shading simulation, utilizing Three.js <sup>11</sup> meshes for geometric modeling. The package performs shading analysis using a WebGL-parallelized implementation of the Möller-Trumbore intersection algorithm ([Möller & Trumbore, 1997](#)), producing color-coded Three.js meshes that represent the expected PV yield. Simshady <sup>12</sup> provides both a package for web development, as well as a Command-line interface (CLI) tool <sup>13</sup> for simulations on your machine.

## <sup>16</sup> Statement of need

<sup>22</sup> For calculating the yield of PV systems, various factors are important, including the location of the planned installation, local climate, surrounding objects such as houses or trees, and terrain.

<sup>24</sup> To provide accurate estimates of expected yields, simulation tools are essential in both research and practical PV system planning. For these reasons, a variety of software tools for simulating photovoltaic systems already exist ([Holmgren, Hansen, Stein, et al., 2018; Jakica, 2018](#)). One widely used software is the Python package pvlib ([Anderson et al., 2023; Holmgren, Hansen, & Mikofski, 2018](#)), which offers a range of functionalities. However, the rather niche topic of shading simulation with 3D objects is not included in this package. Another Python-based software that enables irradiance modeling in two dimensions is pvfactors ([Anoma et al., 2017; Pvfactors, 2022](#)). Web-based tools for solar panel simulations, such as PVGIS, PVWatts, and RETScreen, provide an accessible means for non-technical individuals to estimate energy yields based on geographic location and building geometry ([Psomopoulos et al., 2015](#)). However, these tools lack the capability to perform shading simulations using 3D geometries.

<sup>35</sup> openpv/simshady (or short: simshady) now fills two existing gaps: First, it is implemented in typescript and hence simultaneously enables simulations in the browser and on a local machine. And Second, it tackles the calculation intensive task of raytracing for shading simulation with a performant WebGL implementation. Since WebGL is not well known in the science community, simshady provides access to this simulation code via better known javascript methods.

## 40 Software design

41 simshady simulates the yield of photovoltaic (PV) systems by considering weather/climate  
42 data and shading from local 3D geometry. It is build around three core principles: (1) To run  
43 the core simulation code efficiently on the GPU, (2) to expose two different interfaces for both  
44 browser-based applications and server-based simulation, and (3) to integrate standardized data  
45 models, namely from Three.js and from the Wavefront OBJ file format.

46 In simshady, a 3D scene is built that represents the environment, comprising primary objects  
47 for simulation (e.g., PV panels or target buildings) and surrounding objects that may cast  
48 shadows (e.g., neighboring buildings, trees). Weather and climate data are integrated using  
49 Global Horizontal Irradiance (GHI) and Direct Normal Irradiance (DNI) datasets, which are  
50 reconstructed to include directional irradiance information using the HEALPix framework  
[\(Górski et al., 2005; Zonca et al., 2019\)](#).

51 The simulation utilizes the Möller-Trumbore intersection algorithm ([Möller & Trumbore, 1997](#))  
52 to determine if any shading objects obstruct the view between a sky pixel and the main  
53 simulation geometry. For each triangle in the simulation geometry, a shading mask is generated,  
54 indicating whether an object blocks the line of sight from the sky pixel to the triangle. The  
55 shading mask values range from 0 to 1, where 0 indicates that an object shades the triangle,  
56 1 signifies that there is no obstruction and the line of sight is perpendicular to the triangle,  
57 and values between 0 and 1 represent cases where there is no obstruction but the angle of  
58 incidence is not perpendicular. The aggregated radiance values from all sky dome pixels  
59 are then multiplied by the corresponding shading mask values and summed to calculate the  
60 total energy received by each triangle. This computation is fully parallelizable and has been  
61 implemented using WebGL, allowing for GPU acceleration.

## 63 Simshady in the browser

64 Simshady reuses the data model and objects of Three.js, a common package for 3D web  
65 applications. The package processes Three.js meshes and adds the simulated PV yield as a  
66 color to the mesh, as shown in [Figure 1](#). Additionally, each triangle of the simulated buildings  
67 has its solar yield assigned as an attribute for further processing.



**Figure 1:** A simulated building with its solar yield, where dark purple represents low yields and light yellow represents high yields. The simulated shading from neighboring buildings is visualized through darker colors. Screenshot taken from ([Openpv.de](#))

### 68    **Simshady in the terminal**

69    The simshady CLI is a wrapper around the core WebGL-based simulation engine that enables  
70    batch processing of photovoltaic-yield analyses on a server. It first parses a small set of required  
71    arguments (the simulation geometry and the irradiance data). The supplied geometry files,  
72    either JSON objects or Wavefront OBJ files, are handed to a headless Chromium instance  
73    launched via Puppeteer ([Google Chrome Team, 2025](#)).

74    Inside the browser context the full simshady package is injected, the scene is reconstructed, and  
75    the GPU-accelerated Möller-Trumbore ray-tracing routine is executed. When the calculation  
76    finishes, the CLI extracts the mesh data from the browser, writes binary artefacts (positions.bin,  
77    colors.bin, intensities.bin), a colour-coded OBJ file, a top-down snapshot, and the simulation  
78    results into the user-specified output directory.

### 79    **Research impact statement**

- 80    ▪ First open source, free software of its kind  
81    ▪ Multi usage approach: Server and Client

### 82    **Conclusion**

83    The openpv/simshady package serves two primary purposes: it provides a solution for scientific  
84    calculations of PV yield, while also facilitating science communication through interactive and  
85    user-friendly simulations. This eliminates the need for specialized software or programming  
86    knowledge, making it accessible to a broader range of users. Furthermore, by implementing the  
87    main algorithm in WebGL, the package achieves higher performance than a pure Javascript  
88    implementation, and it offers a JavaScript wrapper around PV simulation in WebGL. This  
89    is particularly beneficial because WebGL is a language that is not widely known among

90 scientists, and thus can be challenging for them to implement their own code, making the  
91 openpv/simshady package a valuable tool for simplifying this process.

## 92 Credit Authorship Statement

93 FK: Conceptualization, Software, Funding acquisition, Writing – original draft  
94 MG: Conceptualization, Software, Funding acquisition, Writing – review & editing  
95 KH: Conceptualization, Software, Funding acquisition, Writing – review & editing  
96 ME: Software, Writing – review & editing  
97 KP: Conceptualization, Software, Funding acquisition, Writing – review & editing

## 98 AI usage disclosure

99 After designing the software architecture and writing core functionalities, both open-weight and  
100 properitarian LLM-based chatbots were used to implement or debug some of the methods in  
101 simshady. Generative agentic AI tools that can write and edit code autonomously were not  
102 used. The open-weight models gpt-oss:120b and deepseek-r1:70b were used for reviewing the  
103 written text of this paper.

## 104 Acknowledgements

105 The development of this software was funded by the German Federal Ministry of Research,  
106 Technology and Space within the “Software Sprint - Support for Open Source Developers”  
107 program by Prototype Fund.

## 108 References

- 109 Anderson, K. S., Hansen, C. W., Holmgren, W. F., Jensen, A. R., Mikofski, M. A., & Driesse,  
110 A. (2023). Pvlib python: 2023 project update. *Journal of Open Source Software*, 8(92),  
111 5994. <https://doi.org/10.21105/joss.05994>
- 112 Anoma, M. A., Jacob, D., Bourne, B. C., Scholl, J. A., Riley, D. M., & Hansen, C. W.  
113 (2017). View factor model and validation for bifacial PV and diffuse shade on single-  
114 axis trackers. *2017 IEEE 44th Photovoltaic Specialist Conference (PVSC)*, 1549–1554.  
115 <https://doi.org/10.1109/PVSC.2017.8366704>
- 116 Google Chrome Team. (2025). *Puppeteer: Headless chrome node.js API* (Version v24.31.0).  
117 Google. <https://github.com/puppeteer/puppeteer>
- 118 Górska, K. M., Hivon, E., Banday, A. J., Wandelt, B. D., Hansen, F. K., Reinecke, M., &  
119 Bartelmann, M. (2005). HEALPix: A framework for high-resolution discretization and  
120 fast analysis of data distributed on the sphere. *The Astrophysical Journal*, 622(2), 759.  
121 <https://doi.org/10.1086/427976>
- 122 Holmgren, W. F., Hansen, C. W., & Mikofski, M. A. (2018). Pvlib python: A python package  
123 for modeling solar energy systems. *Journal of Open Source Software*, 3(29), 884.
- 124 Holmgren, W. F., Hansen, C. W., Stein, J. S., & Mikofski, M. A. (2018). Review of open  
125 source tools for PV modeling. *2018 IEEE 7th World Conference on Photovoltaic Energy  
126 Conversion (WCPEC)(a Joint Conference of 45th IEEE PVSC, 28th PVSEC & 34th EU  
127 PVSEC)*, 2557–2560. <https://doi.org/10.1109/PVSC.2018.8548231>

- <sup>128</sup> International Energy Agency. (2023). *Tripling renewable power capacity by 2030 is vital to keep the 1.5°C goal within reach.* <https://www.iea.org/commentaries/tripling-renewable-power-capacity-by-2030-is-vital-to-keep-the-150c-goal-within-reach>.
- <sup>131</sup> Jakica, N. (2018). State-of-the-art review of solar design tools and methods for assessing daylighting and solar potential for building-integrated photovoltaics. *Renewable and Sustainable Energy Reviews*, 81, 1296–1328. <https://doi.org/10.1016/j.rser.2017.05.080>
- <sup>134</sup> Möller, T., & Trumbore, B. (1997). Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools*, 2(1), 21–28. <https://doi.org/10.1080/10867651.1997.10487468>
- <sup>136</sup> *Openpv.de.* [www.openpv.de](http://www.openpv.de).
- <sup>137</sup> Psomopoulos, C. S., Ioannidis, G. C., Kaminaris, S. D., Mardikis, K. D., & Katsikas, N. G. (2015). A comparative evaluation of photovoltaic electricity production assessment software (PVGIS, PVWatts and RETScreen). *Environmental Processes*, 2, 175–189. <https://doi.org/10.1007/s40710-015-0092-4>
- <sup>141</sup> *Pvfactors: Open-source view-factor model for diffuse shading and bifacial PV modeling* (Version 1.5.2). (2022). <https://github.com/SunPower/pvfactors>
- <sup>143</sup> Zonca, A., Singer, L., Lenz, D., Reinecke, M., Rosset, C., Hivon, E., & Gorski, K. (2019). Healpy: Equal area pixelization and spherical harmonics transforms for data on the sphere in python. *Journal of Open Source Software*, 4(35), 1298. <https://doi.org/10.21105/joss.01298>