The Journal of Open Source Software

Optiland: Open-Source Optical Design Software in Python

Kramer Harrison [1], **Manuel Fragata Mendes** [2], **Corné Haasjes** [3,4,5], **and Grégoire Hein** [1]

1 Independent Researcher **2** Friedrich Schiller University Jena, Germany **3** Department of Ophthalmology, Leiden University Medical Center, Leiden, The Netherlands **4** Department of Radiology, Leiden University Medical Center, Leiden, The Netherlands **5** Department of Radiation Oncology, Leiden University Medical Center, Leiden, The Netherlands

DOI: 10.21105/joss.11170

Software

- Review
- Repository
- Archive

Editor: Matthew Feickert

Reviewers:

- @manunicholasjacob
- @Jashcraf

Submitted: 09 September 2025
Published: 04 November 2026

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

Optiland (Harrison, 2026b) is an open-source optical design package written in Python. It offers a comprehensive platform for the design, analysis, and optimization of complex optical systems, catering to a wide audience from professional engineers and researchers to students and educators. Optiland handles a broad range of optical systems, from classical refractive and reflective designs to advanced freeform and computational optics.

Core features include sequential ray tracing, a rich library of surface types (spherical, aspheric, freeform), optimization and tolerancing support, and a suite of analysis tools for evaluating optical performance (e.g., spot diagrams, wavefront analysis, modulation transfer function). A key feature of Optiland is its dual-backend architecture, which allows users to switch between a NumPy (Harris et al., 2020) backend for fast CPU computations and a PyTorch (Paszke et al., 2019) backend for GPU acceleration and automatic differentiation. This enables the integration of Optiland with machine learning workflows and gradient-based optimization, as all calculations are differentiable. Optiland also includes a graphical user interface (GUI) for interactive design and analysis.

Further documentation is available online (Harrison, 2026a).

Statement of Need

Optiland addresses the need for a comprehensive open-source optical design package that unifies traditional lens design with modern, differentiable workflows. It enables a wide range of optical design, analysis, and optimization tasks that previously required costly commercial software, serving a target audience of students, educators, and researchers. By offering a fully differentiable PyTorch backend, Optiland enables computational optics research where optical systems can be embedded directly into neural networks. This capability is essential for emerging fields such as end-to-end optimization of optical systems and learned inverse design. The PyTorch backend also enables GPU acceleration of ray tracing (see Performance and benchmarking).

State of the Field

The field of optical design has long been dominated by commercial software tools such as OpticStudio (*Ansys Zemax OpticStudio*, 2025) and CODE V (*CODE V*, 2025). While powerful, these proprietary tools often carry high license costs that create significant barriers for students,

educators, and researchers. Furthermore, integrating these closed-source tools into modern deep-learning pipelines is often cumbersome, typically relying on slow external APIs that break the gradient chain required for backpropagation.

While other Python optical libraries exist, they target different scopes or architectures. Prysm (Dube, 2019) focuses largely on physical optics and diffraction, a domain distinct from the geometrical lens design and optimization core of Optiland. RayOptics (Hayford, 2025) offers geometrical ray tracing but relies on standard CPU-based execution.

Build vs. Contribute Justification

We determined that contributing to RayOptics was not feasible, as its architecture differs fundamentally from the vector-native approach required for high-performance differentiable programming. Retrofitting that codebase to support GPU acceleration and gradient tracking would have effectively required a total rewrite. Consequently, Optiland was built from the ground up to be tensor-compatible by default, ensuring it can serve both as a professional engineering tool and a differentiable layer in deep learning pipelines.

Software Design

Optiland's architectural goals are two-fold: to provide a robust, professional-grade environment for standard optical engineering while simultaneously enabling seamless integration with neural networks. To achieve this, the software is built on a modular, object-oriented framework. Optical components, fields, and wavelengths are strictly decoupled, ensuring that the codebase is intuitive for engineers to use and easy for developers to extend.

To support its hybrid application in classical design and AI research, Optiland utilizes a backend abstraction layer. This allows the same high-level API to dispatch calculations to either a NumPy backend (for broad compatibility and standard CPU workflows) or a PyTorch backend (for GPU acceleration and automatic differentiation). At the time of development, PyTorch was chosen over alternatives such as JAX or TensorFlow due to its mature dynamic autograd engine, robust hardware acceleration support, and ubiquitous adoption across the broader research and computer vision communities. Nonetheless, the backend abstraction layer does not preclude additional backends from later being added under the same API.

Functionalities

Optiland supports a wide range of design, analysis, and optimization tasks, making it suitable for both classical optical engineering and modern computational applications. Its main capabilities include:

- **Design Tools**: Sequential ray tracing, lens system modeling (spherical, conic, aspheric, freeform surfaces), and flexible aperture/field/wavelength configurations.
- **Analysis Tools**: Spot diagrams, wavefront analysis, OPD maps, polarization ray tracing, PSF/MTF evaluation, and scattering models.
- **Optimization and Tolerancing**: Gradient-based and global optimization (Virtanen et al., 2020), Monte Carlo tolerancing, parametric sweeps, and specialized glass selection tools.
- **Differentiable Ray Tracing**: A fully differentiable backend via PyTorch enables gradient-based optimization and integration with machine learning frameworks.
- **Material Database**: Built-in refractive index library based on the refractiveindex.info database (Polyanskiy, 2024). Support for user-defined materials.
- **Visualization**: 2D layout plots, 3D ray-trace visualization, and an interactive GUI.
- **Interoperability**: Import of Zemax OpticStudio files, JSON-based Optiland file format for import/export, and a full Python API.
- **Performance**: GPU acceleration with PyTorch and CPU acceleration with Numba (Lam et al., 2015).

Performance and benchmarking

To illustrate the performance gain from GPU acceleration, we benchmarked ray tracing through a Cooke triplet lens and measured throughput in terms of ray–surface interactions per second. Benchmarks were run on a system with an Intel Core i7-12700H CPU and an NVIDIA RTX 3070 GPU, with results shown in Table 1. Each configuration traces a fixed random distribution of 10 million rays (20 million for the float32 GPU configuration) through the lens once; reported figures are a single `trace()` measurement, not an average. We observed run-to-run spread of up to several tens of percent on the CPU configurations, so absolute throughput should be treated as illustrative rather than precise. The benchmark script is provided at `benchmarks/sequential/cooke_triplet.py` in the main Optiland repository so readers can reproduce or repeat these measurements on their own hardware.

Table 1: Benchmark of ray-tracing throughput for a Cooke triplet system. Reported accuracy is the root mean square error (RMSE) of ray intersection points on the image plane relative to NumPy float64.

Backend Configuration	RMSE vs NumPy float64 (m)	Throughput (ray-surfaces/s)	Relative Speedup
NumPy (CPU, float64)	0 (reference)	2.3×10^6	$1.0\times$
PyTorch (CPU, float64)	1.6×10^{-18}	7.1×10^6	$3.1\times$
PyTorch (GPU, float64)	1.2×10^{-18}	5.7×10^7	$24.8\times$
PyTorch (GPU, float32)	3.4×10^{-9}	2.3×10^8	$100.0\times$

As shown in Table 1, the GPU-accelerated 32-bit backend achieves two orders of magnitude higher throughput than the NumPy float64 baseline. This acceleration enables experiments that would otherwise be impractical, including large-scale Monte Carlo tolerancing, high-resolution simulations, and gradient-based inverse design.

As different numerical precisions can in principle affect ray accuracy, we also report the RMSE of intersection points relative to the NumPy float64 reference. For the float32 GPU backend, the observed errors are on the order of nanometers, well below typical optical tolerances except in the most demanding applications.

Research Impact Statement

Optiland bridges the gap between optical engineering and machine learning, enabling research that was previously difficult or computationally prohibitive in open-source environments.

- **Realized Impact:** Optiland is currently used as the optical computation engine for Visisipy (Haasjes et al., 2026), a Python library for simulating visual optics developed by the MReye group at Leiden University Medical Center. This integration demonstrates Optiland's reliability in handling real-world biomedical optical simulations.
- **Community Readiness:** Optiland is released with a comprehensive suite of analysis tools, a verified material database (Polyanskiy, 2024), extensive documentation, and unit tests, ensuring it is ready for immediate adoption by the research community.

Figures

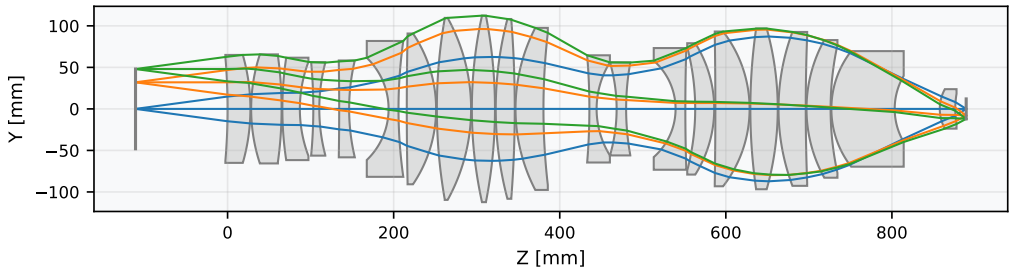


Figure 1: An example of a complex, multi-element UV photolithography lens modeled and rendered in Optiland. Based on U.S. Patent #5,831,776. Differently colored rays represent different fields.

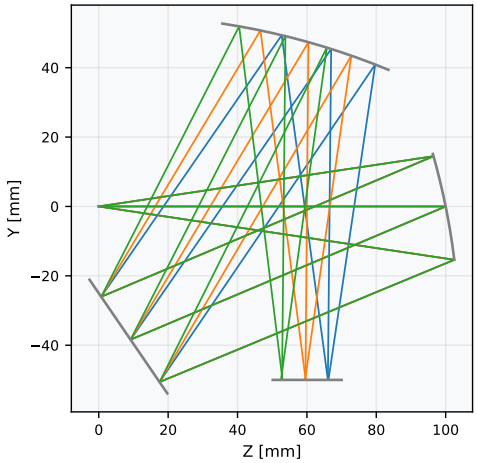


Figure 2: A model of a folded Czerny–Turner spectrometer. Differently colored rays represent different wavelengths.

AI usage disclosure

Project Origins and Confirmation of Review: The core architecture, design philosophy, and fundamental algorithms of Optiland were established beginning in 2017, predating the widespread availability of modern generative AI. The authors maintain sole responsibility for all architectural decisions, scientific validity, and the logical structure of the codebase. All AI-generated outputs were treated as raw suggestions and were manually reviewed, tested, and validated by the authors prior to inclusion.

Tool Usage: During the recent preparation of this submission and the refinement of the codebase (2023–2025), the authors utilized GitHub Copilot, ChatGPT (versions up to 5), Google Gemini (versions up to 2.5), and Google Jules (via Gemini 2.5) to assist with auxiliary tasks.

Nature and Scope of Assistance:

- **Code:** AI tools were employed as a force multiplier for refactoring, generating type hints, writing tests, and implementing feature enhancements based on existing architectural patterns.
- **Documentation:** Tools assisted in generating docstrings and refining documentation.

- **Manuscript Preparation:** LLMs were used for copy editing, grammar checks, and ensuring structural cohesiveness of the manuscript.

Acknowledgements

The authors thank Jan-Willem Beenakker for initiating the collaboration that led to the integration of Optiland into Visisipy. We also thank contributors and community members for feedback and code contributions, in particular Seçkin Berkay Öztürk, Hemkumar Srinivas, Matteo Taccola, Corentin Nannini, Kacper Rutkowski, and David Fariña.

The contributions of Corné Haasjes are part of the OPENOPTICS project with project number SD23.2.004 of the Open Science Fund, which is financed by the Dutch Research Council (NWO).

References

Ansys Zemax OpticStudio. (2025). Commercial optical system design software. https://www.ansys.com/products/optics/ansys-zemax-opticstudio

CODE V. (2025). Commercial optical system design software. https://www.keysight.com/us/en/products/software/optical-solutions-software/optical-design-solutions/codev.html

Dube, B. (2019). Prysm: A Python optics module. *Journal of Open Source Software*, *4*(36), 1352. https://doi.org/10.21105/joss.01352

Haasjes, C., Harrison, K., & Beenakker, J.-W. (2026). Visisipy: Open and accessible vision simulations in Python. *Optica Open*. https://doi.org/10.1364/opticaopen.34056642

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Harrison, K. (2026a). *Optiland documentation*. https://optiland.readthedocs.io/.

Harrison, K. (2026b). *Optiland* (Version 0.6.3). Zenodo. https://doi.org/10.5281/zenodo.23322710

Hayford, M. J. (2025). *Ray-Optics*. GitHub. https://github.com/mjhoptics/ray-optics

Lam, S. K., Pitrou, A., & Seibert, S. (2015). Numba: A LLVM-based Python JIT compiler. *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, 1–6. https://doi.org/10.1145/2833157.2833162

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Polyanskiy, M. N. (2024). Refractiveindex.info database of optical constants. *Scientific Data*, *11*, 94. https://doi.org/10.1038/s41597-023-02898-2

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in

Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2