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.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor:

Submitted: 09 September 2025
Published: unpublished

Summary

Optiland (Harrison, 2025) 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, 2024).

Statement of Need

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), which are powerful but expensive and proprietary. Licenses often cost tens of thousands of dollars, creating a significant barrier to entry for engineers, educators, and researchers.

Optiland addresses this need by providing 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. Optiland offers a differentiable PyTorch backend, which is particularly relevant for computational optics and machine learning-driven design, where novel optimization and inverse-design approaches are increasingly important. For example, optical systems modeled in Optiland can be embedded into deep learning pipelines and trained end-to-end using backpropagation, enabling tasks such as lens design via learned generative models. The PyTorch backend also provides substantial performance improvements through GPU acceleration. By leveraging modern hardware, workloads that would otherwise take minutes or hours on CPUs can be reduced to seconds on GPUs.

Several open-source optical design packages exist, such as Prysm (Dube, 2019), which provides advanced physical optics propagation and diffraction modeling, and RayOptics (Hayford, 2025), which offers ray tracing and lens analysis. Optiland complements these efforts by combining ray tracing, optimization, tolerancing, and differentiable machine-learning integration into a single, comprehensive platform. Optiland is not intended to replace mature commercial tools in every respect (e.g., non-sequential ray tracing, coating optimization, CAD integration), but instead provides an open, extensible framework for research and education in lens/system design.

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 using 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.

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.

Examples

The following examples demonstrate how to use Optiland, starting with a simple system definition, then optimization, and finally machine-learning integration with PyTorch.

1. Defining a simple optical system - The Cooke Triplet

```python
import numpy as np
from optiland import optic

# Create empty Optic instance
lens = optic.Optic()

# Define lens surfaces
lens.add_surface(index=0, radius=np.inf, thickness=np.inf)  # Object plane
lens.add_surface(index=1, radius=+22.01359, thickness=3.25896, material="SK16")
lens.add_surface(index=2, radius=-435.7604, thickness=6.00755)
lens.add_surface(index=3, radius=-22.21328, thickness=0.99997, material=("F2", "schott"))
lens.add_surface(index=4, radius=+20.29192, thickness=4.75041, is_stop=True)
lens.add_surface(index=5, radius=+79.68360, thickness=2.95208, material="SK16")
lens.add_surface(index=6, radius=-18.39533, thickness=42.2077)
lens.add_surface(index=7)

# Add an aperture
lens.set_aperture(aperture_type="EPD", value=10)

# Add fields
lens.set_field_type(field_type="angle")
lens.add_field(y=0)
lens.add_field(y=14)
lens.add_field(y=20)

# Add wavelengths
lens.add_wavelength(value=0.48)
lens.add_wavelength(value=0.55, is_primary=True)
lens.add_wavelength(value=0.65)

lens.update_paraxial()

# Visualize in 3D
lens.draw3D()
```

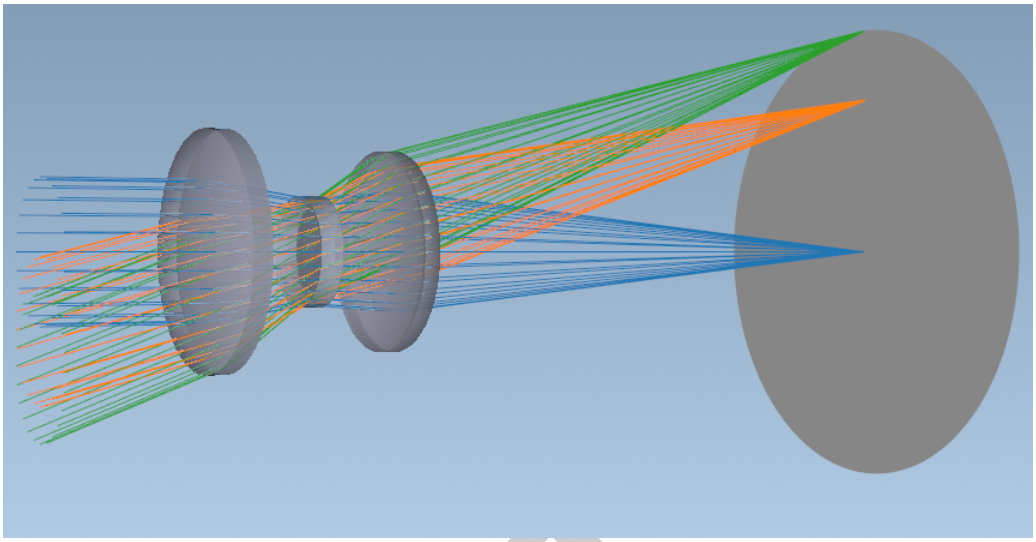

Figure 1: A 3D rendering of a Cooke triplet lens system modeled in Optiland.

2. Optimizing the lens

```python
from optiland import optimization

# Define the optimization problem
problem = optimization.OptimizationProblem()

# Set all radii as variables
for k in range(1, lens.surface_group.num_surfaces - 1):
    problem.add_variable(lens, "radius", surface_number=k)

# Set all thicknesses as variables
problem.add_variable(lens, "thickness", surface_number=1, min_val=2.5, max_val=5.0)
problem.add_variable(lens, "thickness", surface_number=2, min_val=4.0, max_val=8.0)
problem.add_variable(lens, "thickness", surface_number=3, min_val=2.5, max_val=5.0)
problem.add_variable(lens, "thickness", surface_number=4, min_val=4.0, max_val=8.0)
problem.add_variable(lens, "thickness", surface_number=5, min_val=2.5, max_val=5.0)
problem.add_variable(lens, "thickness", surface_number=6, min_val=40, max_val=50)

# RMS spot size operand - to be minimized
for (Hx, Hy) in lens.fields.get_field_coords():
    input_data = {
        "optic": lens,
        "surface_number": 7,
        "Hx": Hx,
        "Hy": Hy,
        "num_rays": 11,
        "wavelength": 0.587,  # μm
        "distribution": "uniform",
    }
    problem.add_operand(
        operand_type="rms_spot_size",
        target=0.0,
        weight=10,
        input_data=input_data,
```

```
    )

# Total track operand
input_data = {'optic': lens}
problem.add_operand("total_track", max_val=60, weight=1, input_data=input_data)

# Optimize
optimizer = optimization.OptimizerGeneric(problem)
optimizer.optimize()
lens.draw()
```

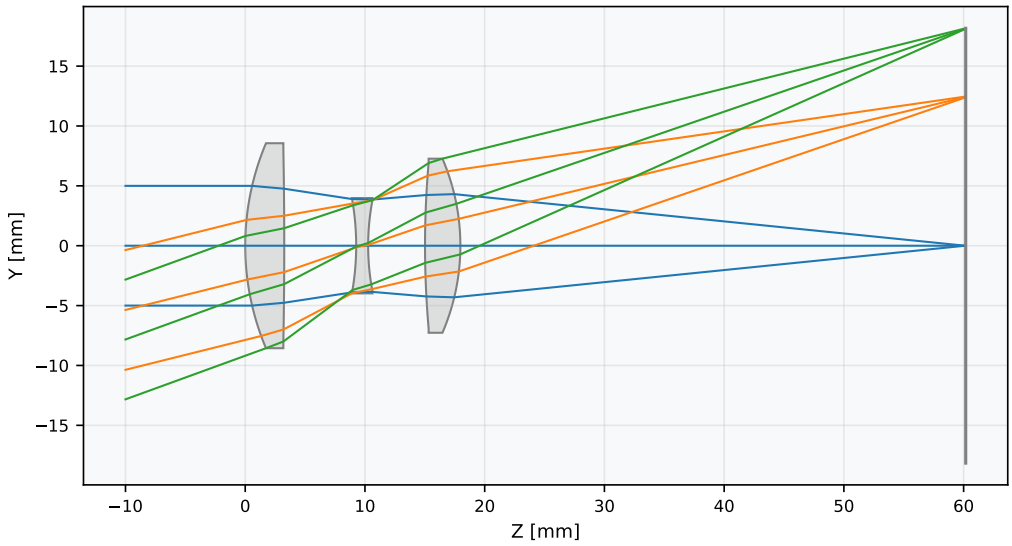

Figure 2: The 2D layout of the Cooke triplet lens.

The result can be refined further using Optiland's GlassExpert functionality to automatically choose high-quality glass candidates.

3. Switching to the PyTorch backend

Optiland's API is identical across backends. Switching to PyTorch enables gradient tracking and GPU acceleration.

```
import optiland.backend as be
be.set_backend("torch")       # Use the PyTorch backend
be.set_precision("float32")   # Set precision of calculations
be.grad_mode.enable()         # Enable gradient tracking
be.set_device("cuda")         # Use CUDA (GPU)
```

4. End-to-end optimization with PyTorch

The PyTorch backend allows integration with neural networks and gradient-based optimizers. Note that to enable this functionality, the lens should be built when the PyTorch backend is active.

```
import torch
from optiland.ml import OpticalSystemModule

# Wrap the lens system and optimization problem in a PyTorch module
```

```
model = OpticalSystemModule(lens, problem)

# Define PyTorch optimizer
optimizer = torch.optim.Adam(model.parameters(), lr=0.1)

losses = []
for step in range(250):
    optimizer.zero_grad()
    loss = model()      # lens merit function
    loss.backward()
    optimizer.step()
    model.apply_bounds()
    losses.append(loss.item())
```

This workflow enables research scenarios such as inverse design, generative modeling, and end-to-end optical design.

Research Enabled by Optiland

Optiland is actively used by researchers in the **MReye group** at the Leiden University Medical Center. It serves as a configurable backend for all optical computations within the Visisipy (Haasjes & Beenakker, 2025) project, a Python library for simulating visual optics.

Figures

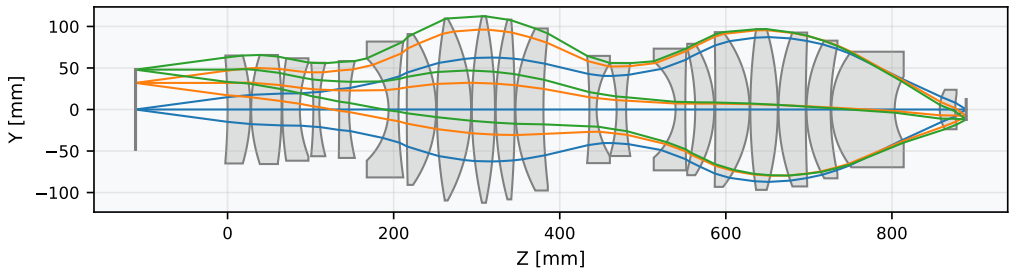

Figure 3: An example of a complex, multi-element UV photolithography lens modeled and rendered in Optiland. Based on U.S. Patent #5,831,776. Different colored rays represent different fields.

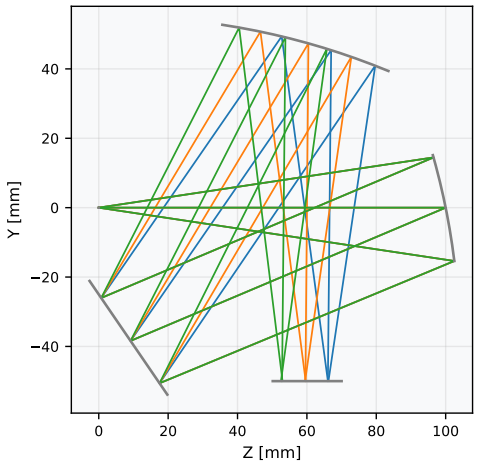

Figure 4: A model of a folded Czerny-Turner spectrometer. Different colored rays represent different wavelengths.

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.synopsys.com/optical-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., & Beenakker, J.-W. (2025). *Visisipy*. Zenodo. https://doi.org/10.5281/zenodo.15423159

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. (2024). *Optiland documentation*. https://optiland.readthedocs.io/.

Harrison, K. (2025). *Optiland* (Version 0.5.5). Zenodo. https://doi.org/10.5281/zenodo.14588961

Hayford, M. J. (2025). Ray-optics. In *GitHub repository*. 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.

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