Review of Scientific Instruments ARTICLE pubs.aip.org/aip/rsi

# RAYX—An optics simulation software for synchrotron applications

Cite as: Rev. Sci. Instrum. 96, 061302 (2025); doi: 10.1063/5.0253857
Submitted: 19 December 2024 • Accepted: 7 May 2025 •
Published Online: 4 June 2025



Sven Erdem,[1,a)] Peter Feuer-Forson,[1,2] Jannis Maier,[1,2] Felix Möller,[1] Enrico Philip Ahlers,[1,2] Valentin Stöcker,[1,3] Fanny Zotter,[1] Peter Baumgärtel,[1] and Jens Viefhaus[1]

## AFFILIATIONS

[1] Helmholtz-Zentrum Berlin für Materialien und Energie GmbH, Albert-Einstein-Straße 15, 12489 Berlin, Germany
[2] Humboldt Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany
[3] Technische Universität Berlin, Straße des 17. Juni 135, 10623 Berlin, Germany

**Note:** This paper is part of the Special Topic on The 8th International Workshop on X-Ray Optics and Metrology.
[a)] **Author to whom correspondence should be addressed:** sven.erdem@helmholtz-berlin.de

## ABSTRACT

We present RAYX, an advanced optics simulation software for synchrotron applications and the successor to RAY/RAY-UI [Schäfers, in *Modern Developments in X-Ray and Neutron Optics*, edited by A. Erko, M. Idir, T. Krist and A. G. Michette (Springer, Berlin, Heidelberg, 2008)], pp. 9–41. RAYX offers a modern, versatile platform designed to address and accelerate the process in the beamline design, its optimization, and data analysis, including machine learning approaches. The aim is to assist synchrotron facilities, including the upcoming fourth generation of synchrotrons, by providing accurate and efficient simulations across a wide range of the electromagnetic spectrum and optical elements. This paper presents the current capabilities of RAYX, including its advanced ray tracing algorithm, hardware optimization, Python bindings for seamless integration into scientific workflows, and a graphical user interface for real-time design and visualization for beamlines.





## I. INTRODUCTION

Synchrotron radiation facilities have become indispensable for a wide variety of scientific research, ranging from the study of fundamental material properties to biological and structural analysis. These facilities provide a unique, highly tunable source of electromagnetic radiation, spanning a broad range of wavelengths. Researchers utilize this radiation to conduct experiments that require extreme precision and control over the photon beam properties. The beamline, which transports the synchrotron radiation from the source to the experiment, consists of various optical components, such as mirrors, apertures, gratings, and slits, but also other specialized optics. The interaction between synchrotron radiation and these optical elements must be considered carefully to provide the experiment with the desired wavelength, intensity, and beam profile.

Accurate modeling of these optical components is essential for the effective design and optimization of beamlines. Given the complexity of these systems, ray tracing simulations offer a robust method for predicting properties of the photon beam throughout the beamline. This approach not only reduces the time and cost associated with beamline development but also enhances the precision and performance of experiments conducted at synchrotron facilities by providing accurate data to compare against experimental findings. Advances in synchrotron research increasingly rely on computational tools, such as machine learning, which depend on large simulation datasets, particularly from optical beamline systems. Such tools can then be applied to tasks such as beamline optimization, quality assurance, and the creation of digital twins of the beamlines.

In light of these trends, there is a growing demand for high-performance, accurate, and flexible ray tracing software tailored to synchrotron beamlines. With this paper, we introduce RAYX, a beamline simulation tool that addresses this demand. We focus on four key features: An advanced ray tracing algorithm that showcases our state-of-the-art ray tracer and its underlying model; hardware utilization & scalability, achieved through the Alpaka[1] framework, ensuring compatibility with a wide range of computer hardware; Python bindings for high-level interaction with the central ray

tracing engine, enabling seamless integration into scientific workflows; and finally, a graphical user interface (GUI) designed for efficient visualization, design manipulation, and optimized simulation workflows.

## II. RELATED WORK

Several software tools have been developed over the years to simulate synchrotron radiation and optical systems used in beamlines. The first versions of RAY[2] and SHADOW[3] emerged as early as the 1980s, both of which simulate the beamline environment by ray tracing using geometric optics. It should be noted that based on the development history of both programs, RAY better covers the requirements of soft x-ray beamlines compared to SHADOW, which focuses more on hard x-ray beamlines. By the late 1990s, new applications such as PHASE[4] and SRW[5] were introduced, offering complex optical wave propagation, including wavefront and coherence mapping along the beamline. Over time, these programs have evolved in line with the user requirements. Improved user interfaces, such as RAY-UI,[6,7] and increased interface compatibility with scripting options in Python (e.g., RayPyNG[8]) have made these tools more accessible. Additionally, new platforms such as xrt,[9] developed directly in Python, have increased the availability of scripting-based simulation tools.

Simulation platforms, such as OASYS[10] and SIREPO,[11] integrate a range of well-known programs and visualization tools, offering a centralized collection of simulation capabilities. SIREPO offers the SRW and SHADOW applications as a service with fee-based support options. The strength of OASYS lies in the fact that it has developed an open standard for integrating applications into the platform, making data exchange possible between the applications. The platform is widely used in the community due to its large number of applications and utilities.

## III. ADVANCED RAY TRACING ALGORITHM

Taking the needs of the synchrotron community into account, we are committed to providing our raytracing algorithm with state-of-the-art features and capabilities. Here are some of the first improvements that have been made:

Our internal model consists of sources and surfaces that are distributed and aligned in space. These objects can also be grouped arbitrarily and are placed relative to the group coordinate system. Grouped objects can thus be transformed together by simply changing the group coordinates accordingly. A key advantage of this approach is that objects within a group can be rotated relative to the group coordinate system, enabling precise transformations around a specific point. This is particularly relevant for components such as SX-700 type soft x-ray monochromators,[12] where optical elements, such as premirrors and gratings, are physically mounted to rotate about well-defined pivot points. This feature remains relevant in recent work.[13] Additionally, our approach serves as preparation for the implementation of volumetric optical elements, such as lenses.

The internal model is independent of the algorithm, which allows for the exchange of models between different simulation programs. This is beneficial for FAIR-Data[14] projects, such as NeXus,[15] and a collaboration to develop a standard for this exchangeability is being undertaken.[16] Furthermore, properties of these grouped objects can be added modularly. In the case of sources, they determine the ray generation, and in the case of surfaces, the behavior of the ray after a collision. These aspects are detailed in the next paragraphs.

Ray generation forms the foundation of the ray tracing sequence and can be customized according to user requirements. Each light source is characterized by its spatial and angular source distribution, energy spectrum, and polarization. For fast simulations, we offer synthetic sources, such as matrix source, point source, pixel source, and circular source. For more detailed but time-intensive simulations, we provide a simplified undulator source and a realistic dipole source.[17]

After the generation of rays, the next critical step is the detection of ray-element collisions. The elements of a beamline are represented by a surface. These surfaces, such as mirrors, gratings, and Fand slits, transfer different behaviors to the beam. Other surfaces, for example, an image plane, which is a virtual screen representing a detector, do not influence the beam. In addition, there is a geometry-based collision detection, in which the algorithm determines whether a ray intersects a beamline element, and if so, where. The geometry includes the dimension and local curvature of the surface, and we have already implemented a large number of surface curvatures for this purpose.[17]

In RAY/RAY-UI, the beam was traced through the beamline from object to object in a fixed order. This algorithm is called sequential ray tracing. Considering the limited hardware resources available when RAY was developed, this method is efficient and, in many cases, would be considered sufficient. However, in RAYX, we have now adopted a more general approach: Rays are treated individually, and all elements are taken into consideration throughout the entire trace of a ray. We call this approach non-sequential ray tracing, which offers several advantages compared to sequential ray tracing. For example, in a case where an element has more than one optically relevant surface, such as Montel optics,[18] the algorithm itself decides which surface to hit first. Additionally, non-sequential ray tracing can also be used to simulate shadowing effects, multiple reflections on a single surface, as well as back and forth motion of a ray. Setups with two or more sources are also far easier to simulate. An example is shown in Fig. 1.

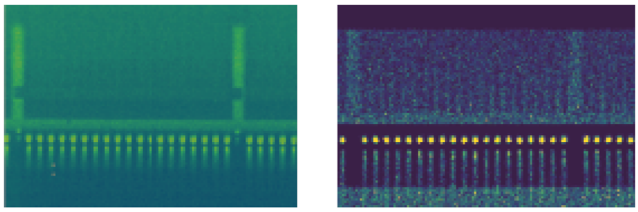

**FIG. 1.** Comparison of experimental and simulated results for a spectrometer utilizing a reflection zone plate (RZP) array as the diffractive element. The beamline features multiple narrow RZPs arranged in a fan configuration, simultaneously illuminated by the incident beam. Using RAYX, individual RZPs were defined and integrated as a group into the model and successfully simulated using non-sequential ray tracing. The figure represents a histogram over the rays that intersect with a detection apparatus (camera). Left: experiment, right: simulation.

In sequential ray tracing, as in RAY/RAY-UI, all rays of a beam are expected to diverge marginally, and thus, a paraxial approximation is utilized in the case of polarization calculus. With the rise of non-sequential ray tracing, paraxial approximation is no longer correct because every ray is self-contained in the global coordinate system, establishing the ability to correctly trace ray polarization independent of divergence.

While in RAY/RAY-UI, the polarization is described in the Stokes vector formalism, in RAYX, we adjusted the polarization calculations to align with the new paradigm of global coordinates. To achieve this, we track a three-dimensional complex electric field vector with each ray, following an approach of Chipman.[19] A conversion between the complex electric field and Stokes vector is provided.

## IV. HARDWARE UTILIZATION AND SCALABILITY

The software is designed with modern hardware and scalable performance in mind, utilizing a variety of platforms.

A core feature of its architecture is the abstraction of platform-specific details achieved through the use of the Alpaka[1] library. Alpaka provides a unified abstract software interface to different hardware backends, meaning that developers do not need to directly interact with multiple native software interfaces, each tailored to specific hardware. This abstraction minimizes code duplication by allowing the same ray tracing code to be seamlessly executed on both CPUs (central processing unit) and GPUs (graphics processing unit), while still allowing for platform-specific optimizations when necessary. This approach supports flexibility and ensures that the software remains future-proof as it can be easily adapted to new architectures without major modifications to the codebase. Additionally, Alpaka achieves performance levels comparable to native software interfaces, as demonstrated in evaluations[20] of the code. From Alpaka's available backends, we currently enable the OpenMP[21] backend for multithreaded execution on x86 CPUs, and we enable the CUDA[22] backend for execution on NVIDIA GPUs.

## V. PYTHON BINDINGS

As an alternative to using RAYX via command-line interface (CLI) or graphical user interface (GUI), we also provide a Python package, called `rayx`, hosted on PyPI.[23] It allows for easy installation via all common package managers without depending on an existing RAYX installation. The Python bindings for RAYX[24] provide a powerful scripting interface, allowing users to integrate beamline design, analysis, and optimization seamlessly into their workflows. By enabling direct interaction with the core functionality of RAYX from Python, these bindings facilitate a fast and flexible approach to beamline development and analysis.

### A. Motivation

Beamline design is an iterative process involving repeated cycles of tracing, analyzing results, and adjusting parameters. The Python bindings streamline this workflow by enabling users to perform each step directly within a Python environment, eliminating the need to switch between different tools or interfaces. This integration significantly speeds up the design process and enhances productivity.

Digital twins are virtual representations of real-world beamlines. By simulating the behavior of physical beamlines with high fidelity, users can predict and analyze performance under various conditions. Future advancements aim to synchronize digital twins with real-world beamline operations, enabling real-time monitoring and adjustments for improved accuracy and efficiency. Their development is leveraged by the Python bindings, facilitating the integration with, for example, Bluesky.[25]

Machine learning applications are also enabled by the Python bindings, expanding the capabilities of RAYX. This allows for a diverse set of underlying design tasks, including optimization of beamline design toward user-set criteria and finding surrogate models for faster simulations. These tasks can be addressed using various machine learning paradigms with the use of libraries such as PyTorch[26] or TensorFlow,[27] allowing for a wide range of approaches independent of methodology.

### B. Supported functionality

Standalone tracing enables users to access RAYX's functionality directly through Python, bypassing the CLI or GUI for more flexible workflows. The Python bindings interface with the same C++ library as the rest of RAYX, ensuring feature parity. Tracing results are generated in memory and returned as a Pandas DataFrame,[28] allowing seamless integration into data analysis workflows without requiring intermediate storage.

Import and export of beamlines in a file format common to both RAYX and RAY-UI is supported. This ensures compatibility with existing workflows and provides a robust exchange format for beamline designs compatible with the GUI and scripting environments.

Editing parameters may be done directly within Python, without modifying the underlying file, enabling quick iterations during the design process. While adding and deleting elements is currently limited, these features are planned for future releases.

## VI. GRAPHICAL USER INTERFACE

Modern beamline design requires tools that not only provide accurate simulations but also enable users to explore and refine configurations with minimal technical barriers.

RAYX-UI addresses this need by offering a GUI designed for real-time interaction with beamline simulations. The GUI combines an intuitive design environment with advanced visualization and analysis tools to streamline the development of beamline setups.

RAYX-UI integrates seamlessly with the computational back end, providing real-time visualization of beamline configurations. As illustrated in Fig. 2, the interface supports precise design adjustments and offers an orthographic camera mode for improved alignment of optical elements. Another implemented feature is the mapping of ray-element intersection footprints onto optical elements after tracing, which provides users with valuable insights into beamline behavior.

In addition to its visualization capabilities, RAYX-UI facilitates collaboration by allowing 3D designs to be shared easily alongside parameter files. This feature enables research teams to exchange and refine beamline configurations efficiently, fostering more effective teamwork.

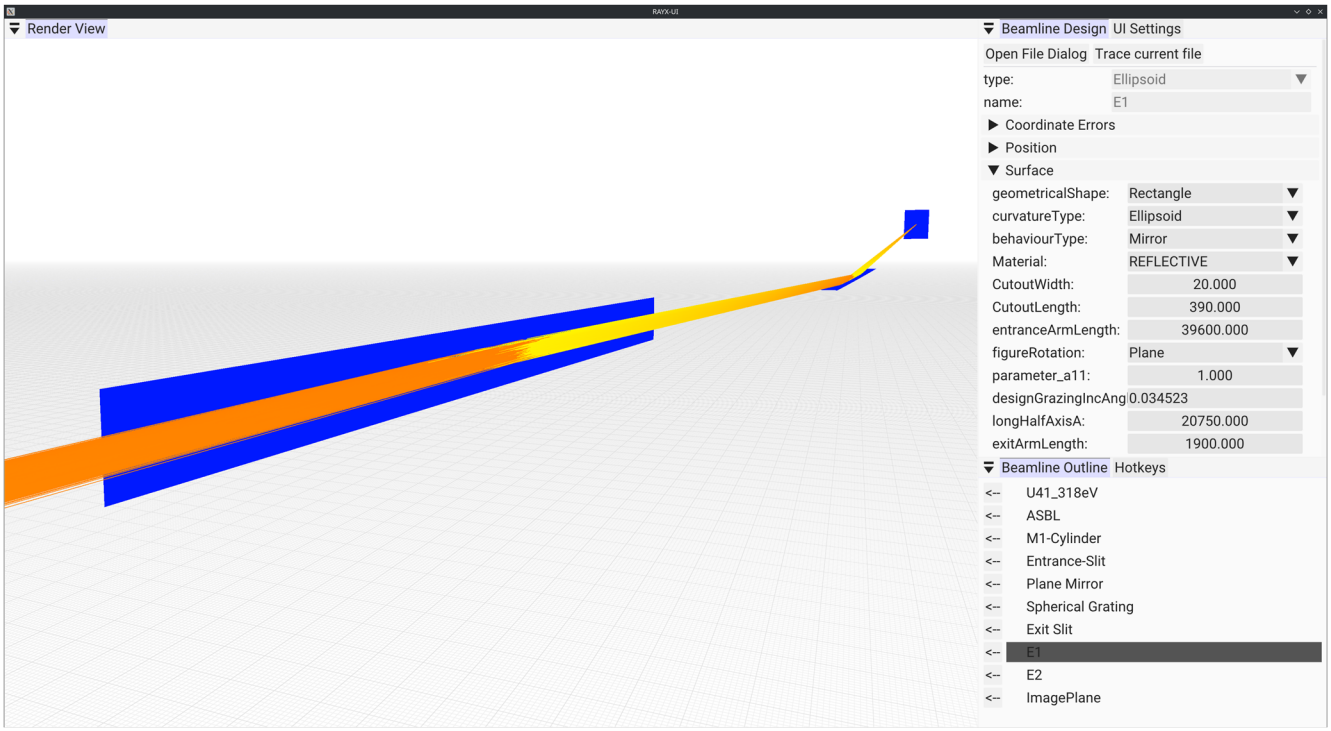


**FIG. 2.** Screenshot of RAYX-UI showing a section of the METRIX beamline at BESSY II. Optical elements are visually represented in blue shades. Ray paths are depicted with a gradient from yellow to orange, differentiating between incoming and outgoing trajectories. The 3D rendering includes two ellipsoid mirrors that focus the rays for experimental use, while an image plane captures the beam's footprint at the focal point. This live visualization exemplifies the interface's capacity to clarify complex spatial relationships within the beamline.



## VII. FUTURE WORK

We are aware that RAY/RAY-UI, as our standard simulation program, cannot yet be fully replaced by RAYX at the current stage of development. However, the following brief outlook shows that we are rapidly extending.

Our next goals are to implement an undulator source[29] and optical element "crystal" as it is already available in RAY/RAY-UI.

An important direction for future development is the implementation of Gaussian beamlet-based complex ray tracing, which is desirable for accurately modeling diffraction effects.[30]

RAY/RAY-UI includes an efficiency calculation for gratings based on Nevière's method,[31] but this functionality remains closed-source. To improve accessibility, we aim to provide an open-source alternative, potentially leveraging the work of Mark Boots.[32]

We aim to further extend the software's compatibility by enabling additional back ends through Alpaka.[1] Potential expansions include support for AMD GPUs via the HIP programming model and Intel GPUs through SYCL or HIP.

Looking ahead, the software's design emphasizes scalability to fully exploit modern hardware stacks comprising multiple CPUs and GPUs, enabling efficient parallel processing for larger and more complex simulations.

Future updates to the Python bindings will include advanced features, such as beamline creation and integration with visualization libraries like Matplotlib. These enhancements will further streamline the beamline design process, allowing users to create, trace, and analyze beamlines entirely within a Python-based workflow.

While RAYX-UI already offers robust features, future developments aim to enhance functionality and usability further. Planned additions include live tracing for direct visual feedback when manipulating elements, enabling more precise iterative design. Intuitive editing tools inspired by 3D design software, such as Blender,[33] will simplify workflows and make the interface accessible to a broader audience. A visual mode to reduce long distances between elements is also planned, improving usability in complex setups. Enhanced analysis tools, such as energy density plots, 2D schematic beamline views, and footprint visualizations (intensity plots), will provide deeper insights into beamline performance. Additionally, integrating a digital twin, as discussed in Sec. V, could enable a live 3D view for real-time parameter adjustments. These developments will solidify RAYX-UI as a cutting-edge platform for beamline simulation and collaborative design.

An important area for future development is the implementation of grating efficiency calculations, a capability currently available in RAY/RAY-UI but closed-source. Providing this feature as a

fully open-source solution would be highly valuable to the research community.

We also aim to develop benchmarks to evaluate the runtime performance of RAYX in comparison with RAY/RAY-UI and potentially other similar software. This will provide a comprehensive perspective on the performance landscape across various applications where runtime efficiency is a relevant factor.

## VIII. CONCLUSIONS

All the topics covered in this publication are strongly oriented toward the needs of the synchrotron community and were incorporated into RAYX as part of an HZB project. In this project, computer scientists and physicists have worked closely together to create a modern, stable, and future-proof open-source software. The first successful applications of RAYX in the field of machine learning demonstrated its proximity to the user community.[34,35]

Last but not least, we would like to emphasize that we are open to collaborations and invite developers, optical engineers, and physicists who would like to implement their own ideas to work with us.

## AUTHOR DECLARATIONS

### Conflict of Interest

The authors have no conflicts to disclose.

### Author Contributions

**Sven Erdem**: Conceptualization (equal); Investigation (equal); Methodology (equal); Resources (equal); Software (equal); Writing – original draft (lead); Writing – review & editing (equal). **Peter Feuer-Forson**: Writing – review & editing (equal). **Jannis Maier**: Conceptualization (equal); Investigation (equal); Methodology (equal); Resources (equal); Software (equal); Visualization (equal); Writing – original draft (equal); Writing – review & editing (equal). **Felix Möller**: Conceptualization (equal); Methodology (equal); Resources (equal); Writing – review & editing (equal). **Enrico Philip Ahlers**: Resources (equal); Software (equal); Visualization (equal); Writing – review & editing (equal). **Valentin Stöcker**: Conceptualization (equal); Investigation (equal); Methodology (equal); Resources (equal); Software (equal); Writing – original draft (equal); Writing – review & editing (equal). **Fanny Zotter**: Conceptualization (equal); Investigation (equal); Methodology (equal); Resources (equal); Software (equal); Writing – original draft (equal); Writing – review & editing (equal). **Peter Baumgärtel**: Conceptualization (equal); Data curation (equal); Formal analysis (equal); Investigation (equal); Methodology (equal); Project administration (equal); Resources (equal); Software (equal); Validation (equal); Writing – original draft (equal); Writing – review & editing (equal). **Jens Viefhaus**: Supervision (equal).

## DATA AVAILABILITY

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

## REFERENCES


[1] A. Matthes, R. Widera, E. Zenker, B. Worpitz, A. Huebl, and M. Bussmann, arXiv:1706.10086 (2017).
[2] F. Schäfers, "The BESSY Raytrace program RAY," in *Modern Developments in X-Ray and Neutron Optics*, edited by A. Erko, M. Idir, T. Krist and A. G. Michette (Springer, Berlin, Heidelberg, 2008), pp. 9–41.
[3] M. Sanchez del Rio, N. Canestrari, F. Jiang, and F. Cerrina, J. Synchrotron Radiat. **18**, 708 (2011).
[4] J. Bahrdt, U. Flechsig, S. Gerhardt, and I. Schneider, in *Advances in Computational Methods for X-Ray Optics II*, edited by M. S. del Rio and O. Chubar (SPIE International Society for Optics and Photonics, 2011), Vol. 8141, p. 81410E.
[5] O. Chubar, L. Berman, A. Fluerasu, Y. Gao, H. Goel, A. He, M. Rakitin, L. Wiegart, and G. Williams, Synchrotron Radiat. News **36**, 15 (2023).
[6] P. Baumgärtel, M. Witt, J. Baensch, M. Fabarius, A. Erko, F. Schäfers, and H. Schirmacher, AIP Conf. Proc. **1741**, 040016 (2016).
[7] P. Baumgärtel, P. Grundmann, T. Zeschke, A. Erko, J. Viefhaus, F. Schäfers, and H. Schirmacher, AIP Conf. Proc. **2054**, 060034 (2019).
[8] S. Vadilonga, G. Günther, S. Kazarski, R. Ovsyannikov, S. Sachse, and W. Smith, in JACoW ICALEPCS2023, THMBCMO18, 2023.
[9] K. Klementiev and R. Chernikov, in *Advances in Computational Methods for X-Ray Optics III*, edited by M. S. del Rio and O. Chubar (SPIE International Society for Optics and Photonics, 2014), Vol. 9209, p. 92090A.
[10] L. Rebuffi and M. S. del Rio, in *Advances in Computational Methods for X-Ray Optics IV*, edited by O. Chubar and K. Sawhney (SPIE International Society for Optics and Photonics, 2017), Vol. 10388, p. 103880S.
[11] M. S. Rakitin, P. Moeller, R. Nagler, B. Nash, D. L. Bruhwiler, D. Smalyuk, M. Zhernenkov, and O. Chubar, J. Synchrotron Radiat. **25**, 1877 (2018).
[12] F. Riemer and R. Torge, Nucl. Instrum. Methods Phys. Res. **208**, 313 (1983).
[13] P. Y. Wang, M. Bazan da Silva, M. Hand, H. Wang, P. Chang, V. Beilsten-Edmands, T. K. Kim, T. L. Lee, K. Sawhney, and A. C. Walters, J. Synchrotron Radiat. **32**, 261 (2025).
[14] P. Groth, H. Cousijn, T. Clark, and C. Goble, Data Intell. **2**, 78 (2020).
[15] M. Könnecke, F. A. Akeroyd, H. J. Bernstein, A. S. Brewster, S. I. Campbell, B. Clausen, S. Cottrell, J. U. Hoffmann, P. R. Jemian, D. Männicke, R. Osborn, P. F. Peterson, T. Richter, J. Suzuki, B. Watts, E. Wintersbergern, and J. Wuttkeo, J. Appl. Crystallogr. **48**, 301 (2014).
[16] G. Günther, O. Mannix, R. Ovsyannikov, and S. Vadilonga, in JACoW ICALEPCS2023, THMBCMO17, 2023.
[17] Helmholtz-Zentrum Berlin, "RAYX Wiki," GitHub, 2024 (accessed 19 November 2024).
[18] M. Montel, *X-Ray Microscopy and Microradiography* (Academic Press Elsevier, 1957), p. 177.
[19] R. Chipman, W. Lam, and G. Young, *Polarized Light and Optical Systems*, 1st ed. (CRC Press, 2018) see page 298 and following.
[20] N. Andriotis *et al.*, EPJ Web Conf. **295**, 11008 (2024).
[21] L. Dagum and R. Menon, IEEE Comput. Sci. Eng. **5**, 46 (1998).
[22] P. V. Nvidia and F. H. Fitzek, Cuda, release: 10.2.89, 2020.
[23] Helmholtz-Zentrum Berlin, RAYX Python bindings on PyPI, 2024.
[24] Helmholtz-Zentrum Berlin, "Rayx python," GitHub, 2024 (accessed 19 November 2024).
[25] D. Allan, T. Caswell, S. Campbell, and M. Rakitin, Synchrotron Radiat. News **32**, 19 (2019).
[26] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *Advances in Neural Information Processing Systems* (Curran Associates, Inc., 2019), Vol. 32, pp. 8024–8035.
[27] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow:

Large-scale machine learning on heterogeneous systems," (2015) software available from tensorflow.org.
[28]T. Pandas Development Team, "pandas-dev/pandas: Pandas," https://zenodo.org/records/13819579, 2020.
[29]M. Scheer, in JACoW IPAC2023, MOPM105, 2023.
[30]J. E. Harvey, R. G. Irvin, and R. N. Pfisterer, Opt. Eng. **54**, 035105 (2015).
[31]M. Nevière and E. Popov, *Light Propagation in Periodic Media: Differential Theory and Design* (CRC Press, Boca Raton, 1999).
[32]M. Boots, "Designing and optimizing gratings for soft X-ray diffraction efficiency," Doctoral dissertation (University of Saskatchewan, 2012).
[33]B. O. Community, Blender—A 3D Modelling and Rendering Package, Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018.
[34]P. Feuer-Forson, G. Hartmann, R. Mitzner, P. Baumgärtel, C. Weniger, M. Agåker, D. Meier, P. Wernet, and J. Viefhaus, J. Synchrotron Radiat. **31**, 698 (2024).
[35]E. Ahlers, P. Feuer-Forson, G. Hartmann, R. Mitzner, P. Baumgärtel, and J. Viefhaus, "Inverse surrogate model of a soft X-ray spectrometer using domain adaptation," arXiv:2502.17505 [physics.ins-det] (2025).