Physics based raytracing of X-ray telescopes

Master Thesis in Computer Science

presented by

Neo Reinmann

born on 01.07.2000 in Erlangen

submitted to

Department of Computer Science
Professorship for Visual Computing
Friedrich-Alexander-Universität Erlangen-Nürnberg

Supervisors: Prof. Dr. Jörn Wilms, Prof. Dr.-Ing. Tobias Günther

Begin date: 15th of March 2025

End date: 15th of September 2025

vce visual computing erlangen

Contents

Chapter 1

Introduction

The introduction needs to answer the following questions:

- Which area of visual computing are we in? (scientific visualization, appearance modeling, Monte Carlo rendering, etc.)
- Which problem are we going to solve?
- Why is this a challenging problem? What makes it difficult?
- Who will be the user of this method? Who benefits from it?
- How are you going to solve the problem?
- How will you evaluate the method?
- What are your most important findings?

Chapter 2

Related Work

There are two main approaches for simulating X-ray telescopes: one that is specific to a particular telescope, and another that generalizes the simulation to accommodate various types of telescopes. In this section 2, I will present a selection of projects that utilize these approaches. Additionally, there are other relevant ray-tracing packages [14], [12], [46].

With the deployment of the X-ray telescope XMM-Newton, SciSim was developed as a comprehensive simulator for the telescope and its observational targets [42]. Its primary purpose is the calibration and understanding of potential optical defects, representing a novel approach in astronomical simulations at the time. SciSim's architecture features a modular design, where each module handles a specific part of the simulation process. Among its functionalities, SciSim simulates the mirror module of XMM-Newton, including the spider support structure and the Wolter I mirror optics. This module provides a detailed model of the optics, incorporating real-world factors such as non-uniform mirror thickness and surface roughness.

Another telescope-specific simulation tool is NuSim [17], developed for the NuSTAR telescope. Like SciSim, NuSim can simulate X-ray sources and their spectra to a certain extent. Due to NuSTAR's dynamic environment, NuSim includes an advanced module to simulate effects such as thermal perturbations. Its optics engine features a full ray-tracing implementation for incoming photons, allowing it to simulate phenomena like mirror imperfections and ghost rays.

Recently, the release of *DarsakX*, an open-source Python package, has enabled the simulation of general Wolter I optics [33]. This flexibility allows the package to simulate various X-ray telescope configurations, making it adaptable to different architectures. The primary architecture consists of a ray-tracing component and a post-processing component. Unlike SciSim and NuSim, *DarsakX* does not account for mirror imperfections, surface roughness, or ghost rays.

While *DarsakX* allows for the simulation of different telescope configurations, its capabilities are limited to providing an overview of the system's performance, rather than detailed studies of scientific instrument behavior. This is where SIXTE (SImulation of X-ray TElescopes) excels, offering a modular simulation package focused on detailed modeling of complex astrophysical sources across several instruments [7]. The modular design enables users to replace individual components with more advanced models when necessary, and new instruments can be added without requiring modifications to the core software. However, unlike the other packages discussed, SIXTE does not currently feature a ray-tracing module for optical predictions and relies on calibration data.

Chapter 3

Fundamentals

In this chapter, we provide an overview of the fundamental concepts that are essential for understanding the simulation process of X-ray telescopes. We begin with a summary of the physical principles of X-rays, outlining their properties and interactions that are relevant in an astronomical context. Building on this foundation, we then describe the methods by which astronomical X-ray photons can be observed, highlighting the specific challenges that arise compared to observations in other wavelength regimes. After this, the simulation framework developed at the institute, known as *SIXTE* (Simulation of X-ray Telescopes), is introduced. We explain its purpose, the underlying structure, and the steps required to perform simulations as well as to analyze the generated results. The discussion then shifts towards the computer science aspects that are necessary to achieve the goals of this thesis. In this part, we outline the fundamentals of ray tracing, survey the range of techniques that exist for simulating light propagation, and explain how these are relevant for X-ray optics. Finally, we conclude the chapter with an introduction to the microfacet theory, which provides one possible approach to modeling surface roughness in a physically meaningful and accurate manner.

3.1 X-ray Physics

X-rays are a part of the electromagnetic spectrum and were first discovered by Wilhelm Conrad Röntgen in 1895. While Röntgen produced X-ray photons using a Crookes tube, X-ray radiation also occurs naturally, particularly in outer space. Various celestial objects and astrophysical events emit electromagnetic radiation across a wide range of wavelengths, extending far beyond the visible spectrum. However, due to the strong absorption of X-rays in Earth's atmosphere, they cannot penetrate to the surface and thus cannot be observed directly from the ground.

In contrast to visible light, X-rays are far more energetic and have much shorter wavelengths. Figure 3.1 illustrates the classification of X-rays within the high-energy portion of the spectrum. Typically, X-rays are defined as having wavelengths between 10 nm and 10 pm, corresponding to photon energies of approximately 100 eV to 100 keV.

Because Earth's atmosphere efficiently absorbs and scatters X-rays, observations of the X-ray sky are not possible from the surface. In space, however, where the density of matter is significantly lower, X-ray sources can be detected without substantial interference. Figure 3.2 shows the fraction of X-ray photons that are not absorbed by Earth's atmosphere. The dashed-dotted line indicates the altitude above

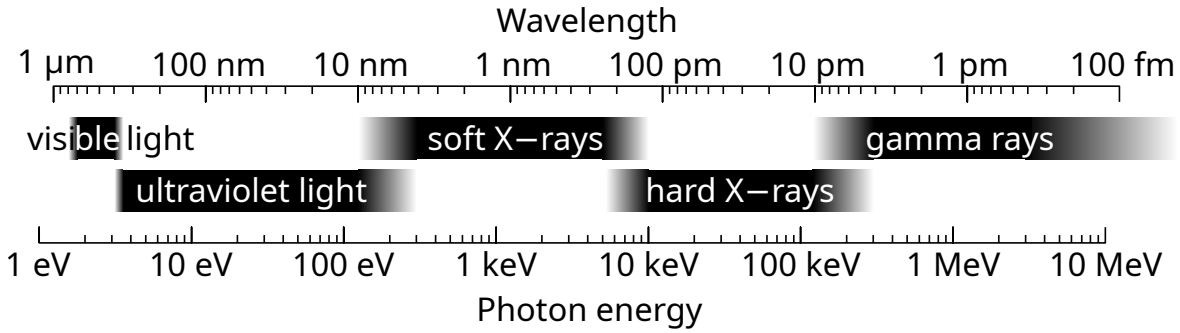

Figure 3.1: The high-energy electromagnetic spectrum [35].

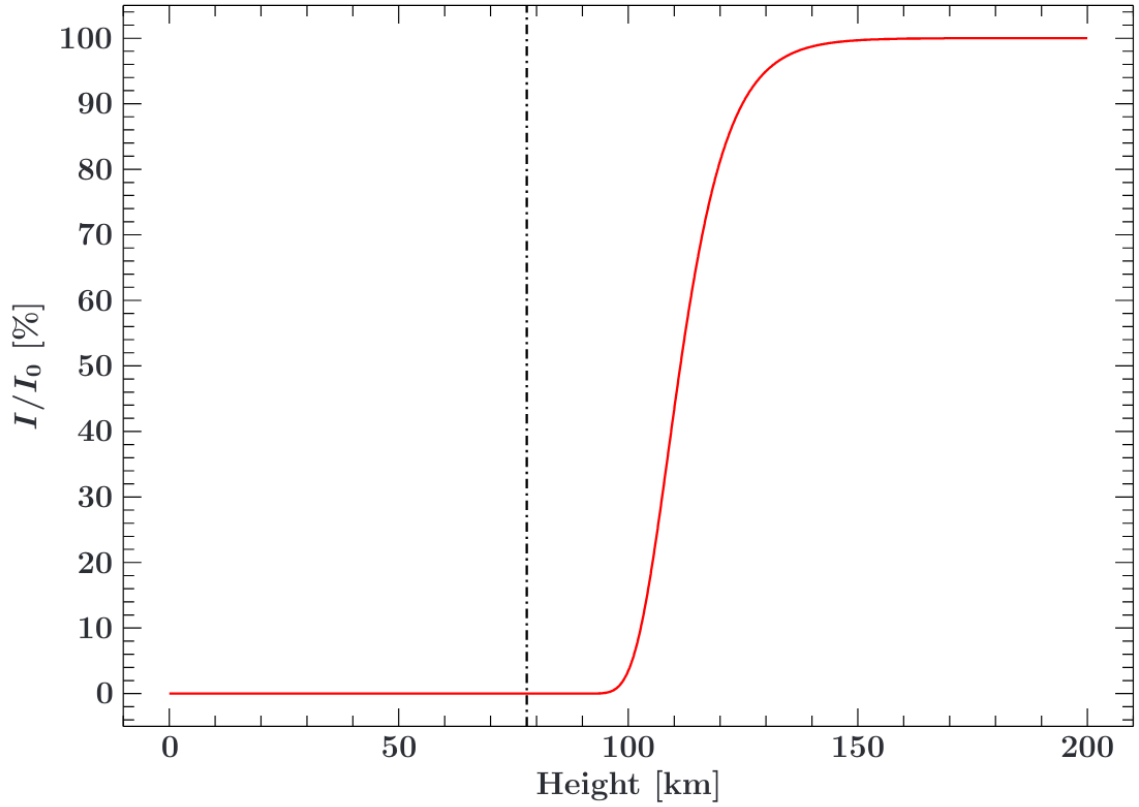

Figure 3.2: Fraction of radiation not absorbed by an atmosphere of 79% nitrogen and 21% oxygen, modeled for Earth at 1 keV and 250 K, as a function of altitude [39].

ground at which X-ray photons with an energy of 1 keV encounter an effectively opaque atmosphere. This illustrates the necessity of elevating instruments above the atmosphere, either on satellites or high-altitude missions, to enable scientific observations of X-ray sources.

Astrophysical phenomena such as supernovae, stellar collisions, and compact objects like neutron stars and black holes emit substantial amounts of X-ray radiation, making this part of the spectrum particularly important for astrophysics. Hot plasma composed primarily of hydrogen and helium ions,

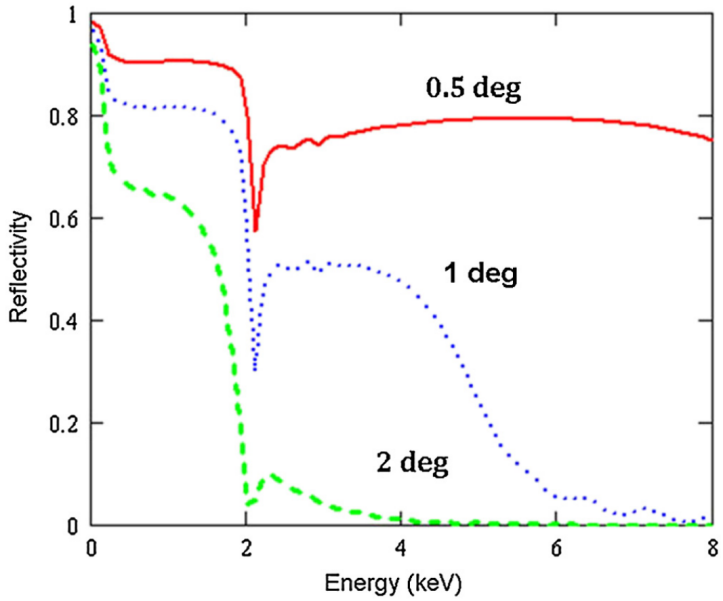

Figure 3.3: Decreasing reflectivity for fixed grazing incidence angles with increasing photon energy (Image © 2012 SPIE [11]).

with smaller fractions of heavier elements such as carbon, magnesium, or iron, is one prominent source of X-rays. Another significant mechanism is synchrotron radiation, which arises when high-energy electrons traverse magnetic fields and emit X-rays in the process. Additional mechanisms, such as Compton scattering and bremsstrahlung, also contribute to X-ray emission [11]. Since these processes are not the central focus of this thesis, we will not discuss them in further detail here. Nevertheless, they play crucial roles in shaping the dynamics of astrophysical events and provide observational evidence that supports theoretical physical models.

Detecting high-energy photons can in principle be achieved with simple instruments such as Geiger counters, but forming detailed astronomical images in the X-ray band is considerably more challenging than in the visible, infrared, or ultraviolet regimes. This difficulty arises because X-ray photons penetrate matter at high incidence angles rather than reflecting off conventional mirrors. Meaningful reflection is only possible at very shallow, grazing incidence angles of typically about 1°–2°. The precise reflection properties depend on both the photon energy and the material composition of the mirror [18].

Figure 3.3 demonstrates the rapid decline of reflectivity with increasing photon energy. The reflectivity also varies depending on the choice of mirror coating, with iridium shown here as an example. In the X-ray band, the refractive index of metals is slightly less than unity, resulting in very low critical angles for near-total reflection [11]. An analogous effect can be observed in visible light when a light ray traveling through air strikes a water surface: at higher incidence angles, the light refracts and enters the water, whereas at sufficiently shallow glancing angles, reflection occurs instead. A familiar manifestation of this phenomenon is the strong reflection of the Sun on water surfaces at low incidence angles, which can be dazzling at sunset.

With this foundation on the properties of X-rays and their astrophysical origins, we now turn to the instruments and techniques required to focus and observe them effectively.

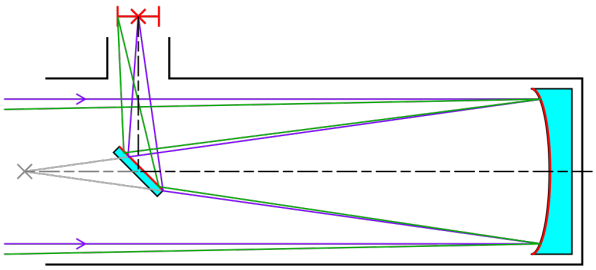

Figure 3.4: Schematic of a Newtonian telescope for observing photons in the visible band [15]. Because the incoming rays strike the mirror nearly perpendicularly, this configuration would completely absorb X-ray photons.

3.2 X-ray Telescopes

This section provides a detailed overview of X-ray telescopes, which form the foundation for the analytical modeling of mirrors in the later ray-tracing simulations. As outlined in the previous section, X-ray photons are difficult to detect and cannot be observed directly from Earth. In contrast to telescopes designed for the lower-energy ranges of the electromagnetic spectrum, X-rays cannot be focused by a primary parabolic mirror onto a secondary mirror, which then projects the light onto a detector. Figure 3.4 illustrates the schematic layout of a traditional optical telescope for visible light. The defining characteristic of the primary mirror is its parabolic shape: every on-axis ray, such as the one indicated by the purple line, converges at the same focal point regardless of where it strikes the mirror surface. The secondary mirror then redirects the converging rays out of the telescope aperture so that detectors such as a camera module or an eyepiece can be positioned without obstructing the incoming light.

For X-rays, however, different mirror geometries must be employed in order to achieve reflection at shallow grazing incidence angles and to focus the photons onto a defined focal point. In the following, we introduce the most widely used optical system for X-ray telescopes, namely Wolter optics. As a second example, we describe the Lobster-Eye optics, which is optimized for all-sky monitoring due to its much larger field of view. Before presenting these specific systems, we begin with a general overview of telescope characteristics.

3.2.1 Fundamentals of Telescope Characteristics

In general, there are two fundamental approaches for focusing photons onto a focal plane. The first is *refraction*, where light is bent as it passes between two media with different refractive indices. Due to their high energy and short wavelength, X-rays are not effectively refracted and thus refraction-based designs are not used in X-ray astronomy. The second approach is *reflection*, in which curved mirrors are shaped to direct photons to a common focus. Figure 3.4 already illustrates this principle for visible light.

All telescopes, regardless of the wavelength band they operate in, are characterized by several common parameters. The *aperture* A of a telescope describes the diameter of the optical system. A larger aperture allows more photons to reach the detector, thereby reducing the exposure time needed to resolve faint astronomical objects compared to telescopes with smaller apertures. A second key parameter is the *focal length* FL, defined as the distance between the focal point and the mirror. Together, aperture and focal length determine the *focal ratio* N, as expressed in Equation (3.1).

$$N = \frac{FL}{A} \tag{3.1}$$

As an example, the Hubble Space Telescope (HST) has a focal length of 57.6 m and a primary mirror aperture of 2.4 m, which yields a focal ratio of $f/24$. This configuration enables deep-space imaging with a relatively small field of view (FOV), depending on the type of detector instrument used [16].

The field of view of a telescope can be calculated with Equation (3.2). This represents the maximum angular extent that can be imaged on the detector. For Wolter telescopes, the field of view is further constrained by the obstruction caused by nested mirror shells. In such cases, the effective field of view is defined as the angular range within which the effective area is reduced to half of its maximum value [31].

$$FOV = 2 \times \arctan\left(\frac{s}{2 \cdot FL}\right), \quad s = \text{sensor size} \tag{3.2}$$

Once the field of view is determined, the *angular resolution per pixel* R_p can be obtained as in Equation (3.3), where R is the sensor resolution in one dimension:

$$R_p = \frac{FOV}{R} \tag{3.3}$$

These relations highlight the inherent trade-offs in telescope design: an optical system optimized for deep-sky imaging with a long focal length and narrow field of view cannot simultaneously capture a wide field of view. Consequently, different use cases require different optical parameters. These core concepts apply across all regions of the electromagnetic spectrum. With these basics established, we now examine the first mirror geometry capable of focusing X-ray photons: Wolter optics.

3.2.2 Wolter Optics

A Brief History of Wolter Optics

The name *Wolter optics* derives from the German physicist Hans Wolter, who in the early 1950s sought to design a novel type of high-resolution microscope that could be used to study living organisms under normal atmospheric conditions. The use of X-rays appeared promising, as they offered superior angular resolution compared to ultraviolet light, which is still partially absorbed by the atmosphere. In 1952, Wolter published his proposal of mirror systems based on grazing-incidence reflection for X-rays [44].

Figure 3.5 illustrates Wolter's initial concept. To exploit the total external reflection of X-rays at grazing incidence angles, he realized that incoming rays must strike the mirror at very small angles, as discussed earlier in Figure 3.3. Wolter introduced the secondary mirror as a means of reducing coma aberration, which would otherwise arise when focusing photons with a paraboloid mirror alone.

In a follow-up publication [45], Wolter refined his design by incorporating Schwarzschild's solution for an aplanatic mirror system. This modification introduced a convex correction to the paraboloid and a concave correction to the hyperboloid, which together slightly reduced coma aberrations. Figure 3.6 shows an exaggerated comparison between Wolter's original design and the Wolter–Schwarzschild configuration.

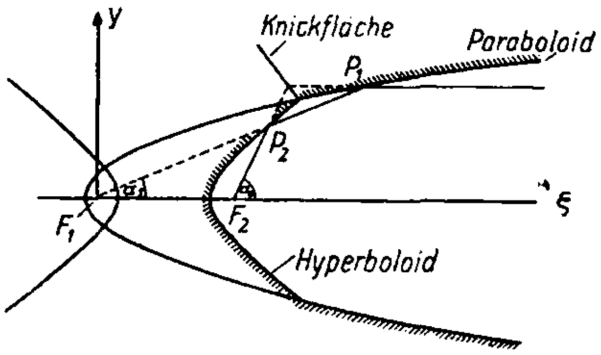

Figure 3.5: Wolter's initial concept of an X-ray microscope, consisting of a two-mirror system [44].

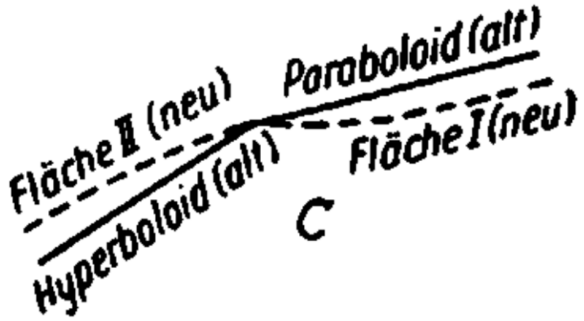

Figure 3.6: Exaggerated comparison of the Wolter–Schwarzschild mirror system, showing the modified paraboloid (Fläche I (neu)) and hyperboloid (Fläche II (neu)) in relation to Wolter's original design [45].

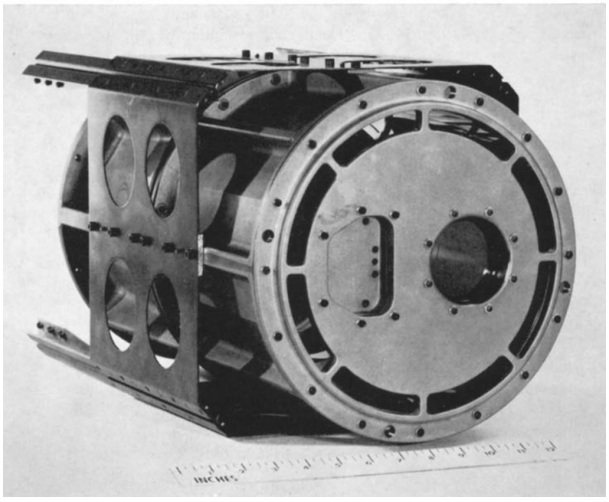

Figure 3.7: The 23-cm Kanigen telescope, a Wolter Type I configuration with a collecting area of 34 cm^2 and a focal length of 132 cm [10].

In his two seminal papers [44, 45], Wolter also proposed several variations of mirror systems. Each design used a paraboloid as the first mirror, but the secondary mirror differed: in one configuration, rays were reflected from the outer surface of a hyperboloid, yielding a relatively long focal length; in another, rays were reflected from the outer surface of a paraboloid onto an ellipsoidal secondary, resulting in a shorter focal length. Schwarzschild solutions also existed for these designs.

Although Wolter originally envisioned these systems for microscopes, the same optical principles apply to telescopes. Because Wolter was the first to propose this novel method of reflecting X-rays, these designs are now classified as Wolter Types I, II, and III. However, the manufacturing precision required to realize such systems was far beyond the technological capabilities of the 1950s. It was not until thirteen years later that a promising Wolter Type I telescope was successfully demonstrated by Giacconi et al. [10]. Their prototype, the 23-cm diameter Kanigen telescope shown in Figure 3.7, marked the first empirical step toward building functional X-ray telescopes. They observed that improvements in mirror surface roughness and slope accuracy were crucial for increasing angular resolution toward its theoretical limit.

Shortly thereafter, VanSpeybroeck and Chase [36] conducted a detailed analytical study of X-ray mirror geometries using a Monte Carlo ray-tracing method. Focusing on Wolter Type I designs, which were easier to manufacture, they derived practical formulae for confocal and coaxial paraboloid–hyperboloid pairs. They also analyzed the trade-off between angular resolution and effective area: longer mirrors increase effective area but degrade resolution. To address this, they proposed *nesting* multiple mirror shells of fixed length within one another, thereby increasing effective area without significantly sacrificing resolution. Their work further demonstrated that incorporating Schwarzschild corrections into Wolter–Schwarzschild systems, while theoretically reducing optical errors, was impractical for manufacturing and ultimately unnecessary, since other distortions dominated in practice.

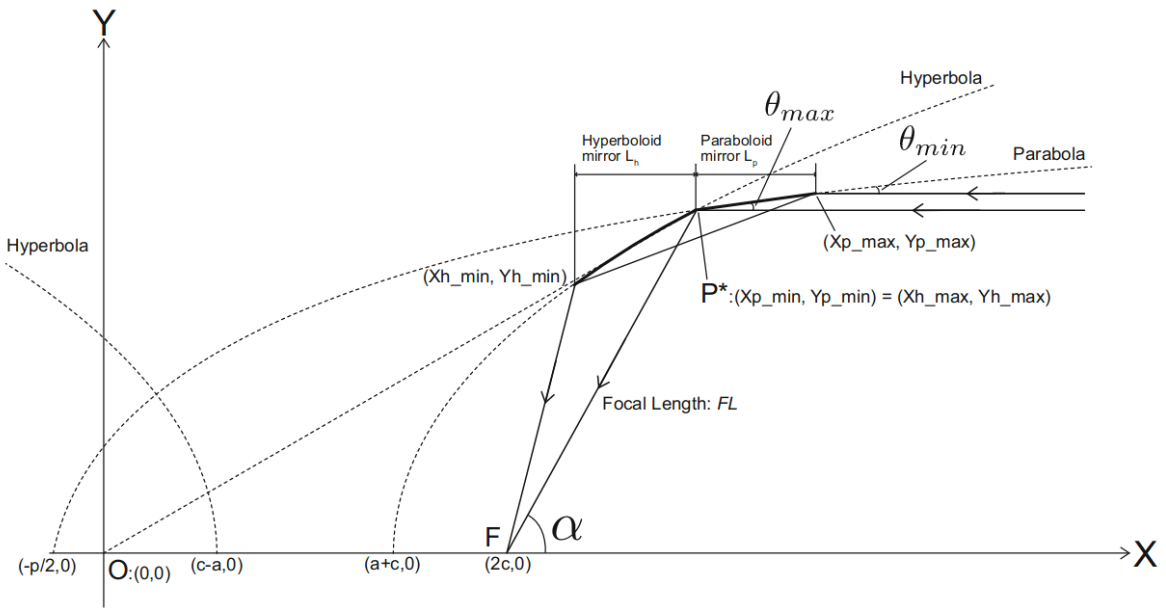

Figure 3.8: Geometric parameters of a Wolter I focusing mirror [25].

Mathematical Background

Building on these earlier developments, Pivovaroff and Okajima [25] recently reformulated the equations for Wolter I geometries. The present thesis relies on their formalism to generate mirror surfaces in simulation. Figure 3.8 shows the relevant parameters.

The paraboloid surface can be described in two dimensions as

$$y^2 = p(2x + p), \tag{3.4}$$

where the focus lies at the origin of the coordinate system. The x-coordinate of the focal point can be derived as

$$p_x = \frac{1}{4p} + t. \tag{3.5}$$

By restructuring Equation (3.4) into the standard parabola form,

$$x^2 = \frac{y^2}{2p} - \frac{p}{2}, \tag{3.6}$$

and substituting into Equation (3.5), one obtains

$$p_x = \frac{2p}{4} - \frac{p}{2} = 0, \tag{3.7}$$

confirming that the focal point remains fixed at the origin.

The secondary hyperboloid surface can be expressed as

$$\frac{(x-c)^2}{a^2} - \frac{y^2}{b^2} = 1. \tag{3.8}$$

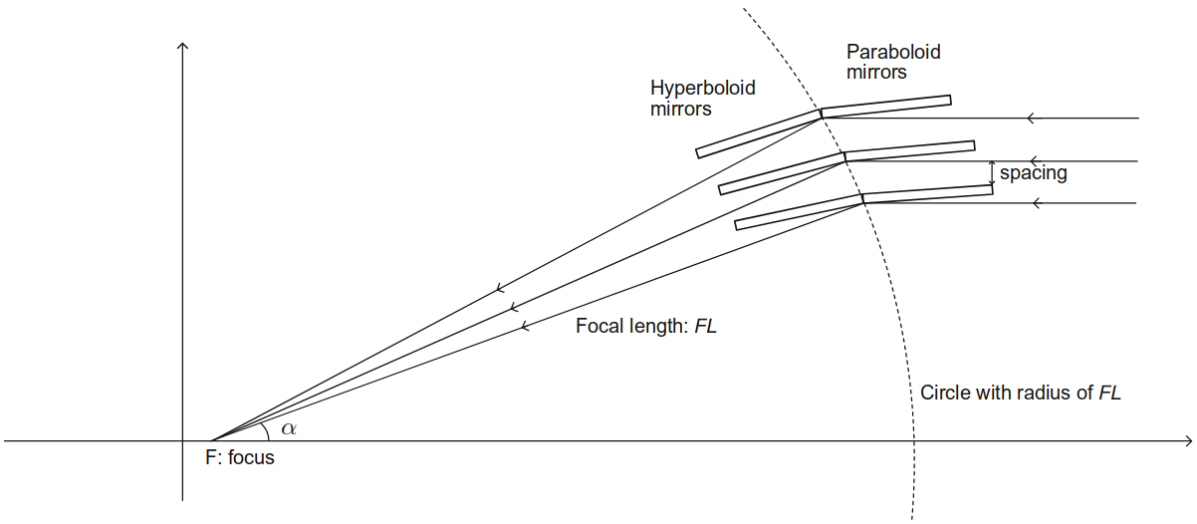

Figure 3.9: Three nested shells, each preserving its true focal length [25].

Together, Equation (3.4) and Equation (3.8) define a coaxial, confocal paraboloid–hyperboloid pair suitable for Wolter I optics. Appendix A lists the detailed formulae for calculating the parameters p, a, b, and c. The input values required to specify a mirror shell are its minimum radius Yp_{min}, focal length FL, and the lengths of the two mirror segments. In Chapter 4, these equations are extended into three dimensions and adapted for intersection tests within the ray-tracing simulation.

Wolter [45], Giacconi et al. [10], and VanSpeybroeck and Chase [36] also recognized that nesting multiple shells increased the effective collecting area of the system, enabling more photons to be gathered in the same exposure time. Figure 3.9 illustrates an ideal case in which three nested shells each preserve their true focal lengths, thereby satisfying the Abbe sine condition [25]. However, this ideal geometry is costly to manufacture, so VanSpeybroeck and Chase [36] demonstrated that less precise configurations, such as that shown in Figure 3.10, are acceptable in practice. Later in this thesis, we will evaluate simulation outputs for different nesting strategies.

Implementation of Wolter I Optics

The *Chandra X-ray Observatory* (CXO), launched in 1999, represents the most advanced implementation of Wolter I optics to date. With its four nested mirror shells and a focal length of 10 m, Chandra achieves an angular resolution of 0.5 arcsec, unmatched by any subsequent X-ray mission [41]. Its mirrors are constructed from Zerodur blanks coated with iridium to enhance reflectivity, with an outer diameter of 1.2 m. Although the mission was originally designed for a three-year lifetime, Chandra remains operational today, providing the highest-resolution data available in X-ray astronomy. Figure 3.11 illustrates its imaging capability with an observation of the supernova remnant Cassiopeia A. Compared to earlier missions such as ROSAT, Chandra's resolution represented an order-of-magnitude improvement [40].

However, telescopes are always designed for specific use cases, and no single instrument can perform optimally across all observational needs. While Chandra excels at high-resolution imaging and spectroscopy, it lacks the wide field of view required for all-sky surveys. To address this, the *eROSITA*

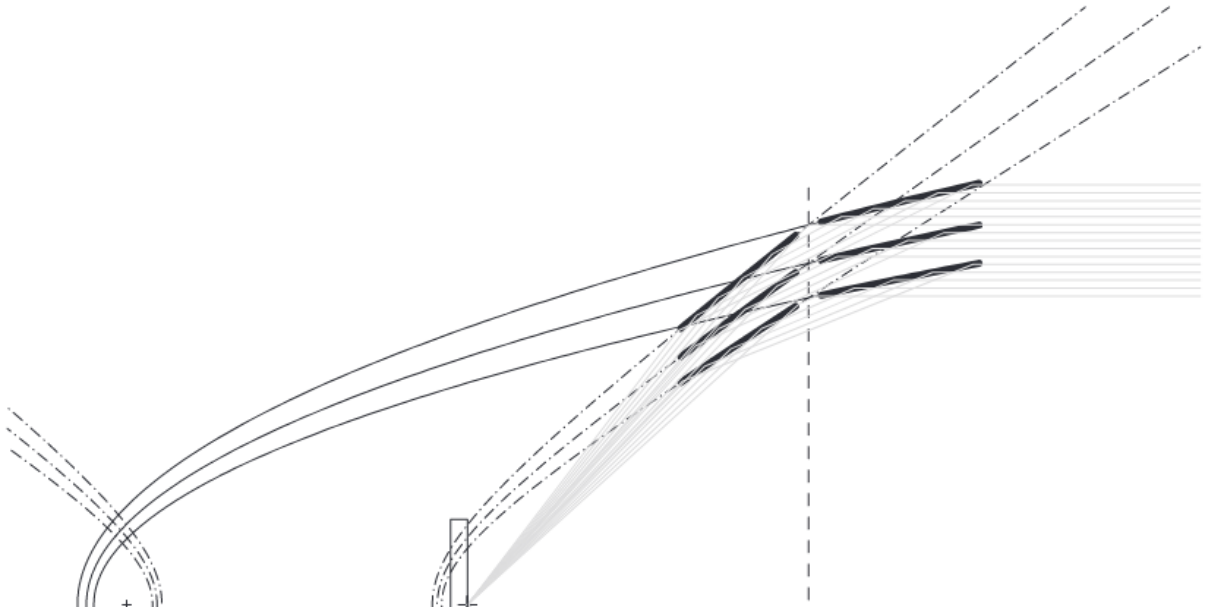

Figure 3.10: Nested Wolter shells in a relaxed configuration with mirrors aligned at the same axial position [43].

Figure 3.11: Supernova remnant Cassiopeia A observed in X-rays by Chandra (blue) and in infrared by Webb (orange, white) [22].

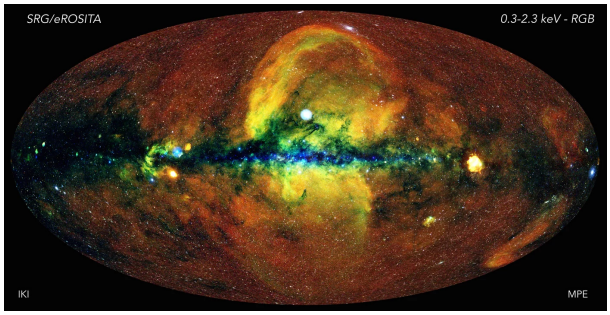

Figure 3.12: First all-sky survey of eROSITA, released in 2020 [28].

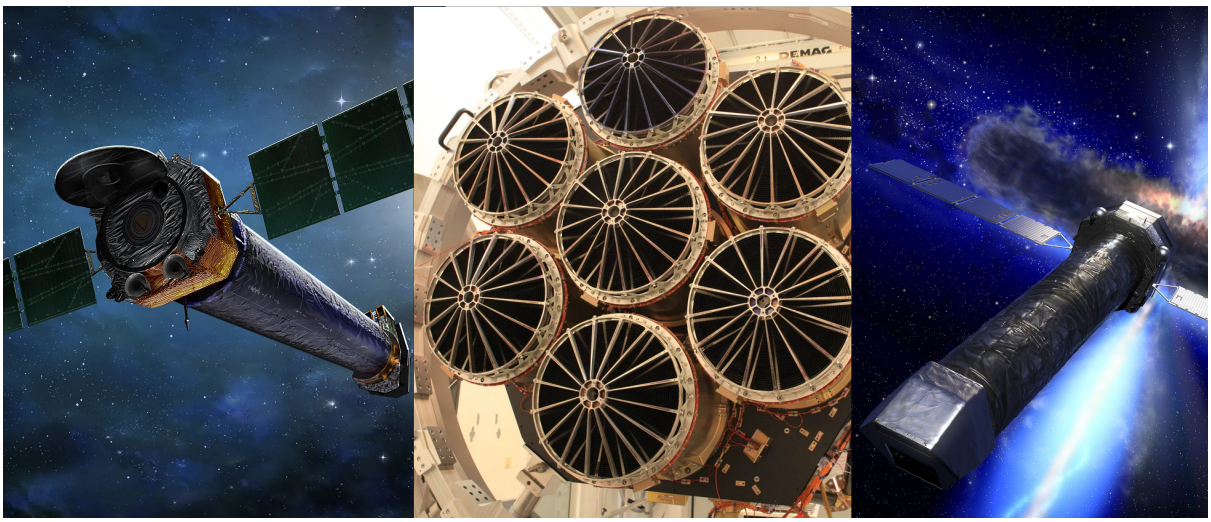

Figure 3.13: *Left:* Artistic rendering of the Chandra X-ray Observatory [21]. *Middle:* The seven mirror modules of eROSITA [20]. *Right:* Concept drawing of NewAthena [8].

mission was launched in 2019, designed specifically for large-scale surveys. Its spacecraft supports three observing modes, the most important being continuous scanning, in which the instrument rotates 90 degrees per hour to trace great circles across the sky. In its first four years, this mode enabled the creation of the most detailed all-sky X-ray survey to date.

Each of eROSITA's seven mirror modules contains 54 nested shells, with diameters of approximately 358 mm and focal lengths of 1600 mm. The mirrors are coated with gold, providing an on-axis angular resolution of 15 arcsec and about 28 arcsec in scanning mode. The field of view is about 1 degree, and the energy band covers 0.5–10 keV [26]. The first all-sky survey, released in 2020 and shown in Figure 3.12, doubled the number of known X-ray sources [28].

Looking ahead, ESA's planned *NewAthena* mission proposes a novel optical design that leverages advances in silicon wafer technology. The new *Silicon Pore Optics* (SPO) manufacturing technique enables highly accurate stacking of thin silicon plates, allowing the realization of Wolter–Schwarzschild geometries at scale. This design combines a larger effective area with improved angular resolution compared to traditional telescopes [2]. Newathena, scheduled for launch in 2037, will employ 600 mirror modules with a diameter of 2.5 m and a focal length of 12 m, achieving an angular resolution of about 5 arcsec.

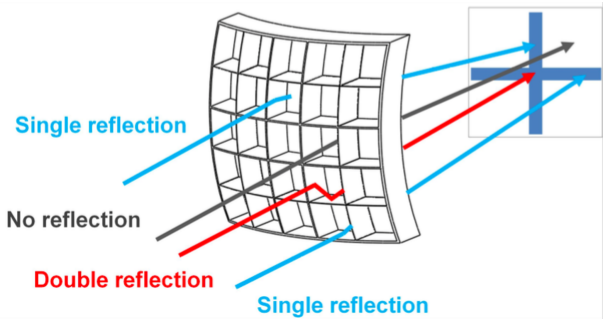

Figure 3.14: Possible reflection paths of X-rays within a Lobster-Eye optic [5].

3.2.3 Lobster-Eye Optics

In addition to survey studies and pointed observations, continuous monitoring of the entire sky is essential for the timely detection of transient X-ray events. Such monitoring does not require high angular resolution, but rather approximate localization of events, so that follow-up instruments operating at different wavelengths can quickly reorient and observe them in greater detail.

For this purpose, a large field of view is required. Conventional wide-angle lenses are not suitable for X-ray astronomy, and Wolter geometries cannot provide sufficiently wide coverage. Instead, the so-called *Lobster-Eye optics*, first proposed by Angel [1], have been developed for this application.

The point spread function (PSF) of a Lobster-Eye telescope has a distinctive cross-shaped structure, as shown schematically in Figure 3.14. This pattern arises from the rectangular micro-channels that constitute the optic: single reflections form the arms of the cross, while double reflections converge to form a central focal spot for on-axis photons.

The fabrication process is analogous to that of Silicon Pore Optics, but is instead termed *Micro Pore Optics* (MPO). This technique allows the construction of narrow channels, typically about 20 μm in size, which maximize effective area while minimizing unwanted single reflections. One notable implementation is the Soft X-ray Imager (SXI) planned for ESA's *THESEUS* mission, which will achieve a field of view of 31°, an order of magnitude larger than that of eROSITA, making it ideally suited for all-sky monitoring [4].

Digest of ow to onstruct hem in he con- ext of MPOs: naybe ring the nath, if ou un- erstand he math. Otherwise ust refer o the pa- er. Add efinetly he math escrip- ion of hannels vith their

3.3 Simulation of X-ray Telescopes (SIXTE)

In Chapter 2, we discussed several ray-tracing and simulation tools that are capable of modeling either specific instruments or selected components such as optics. In contrast, *SIXTE* (Simulation of X-ray Telescopes) integrates the entire simulation chain, thereby enabling the high-fidelity modeling of instrumental effects. This section outlines the overall workflow of SIXTE and concludes by indicating where the contribution of this thesis fits into the framework.

Figure 3.15 shows the three principal functional blocks of SIXTE. The first is *photon generation*, which produces a photon list. The second is *photon imaging*, where the telescope optics are simulated and an impact list is generated. Together, these two stages constitute the *Telescope and Mirror Model*. Finally, the *event detection* block simulates the detector and produces an event file.

In the photon generation step, the input is a *source catalog*, which lists X-ray sources together with

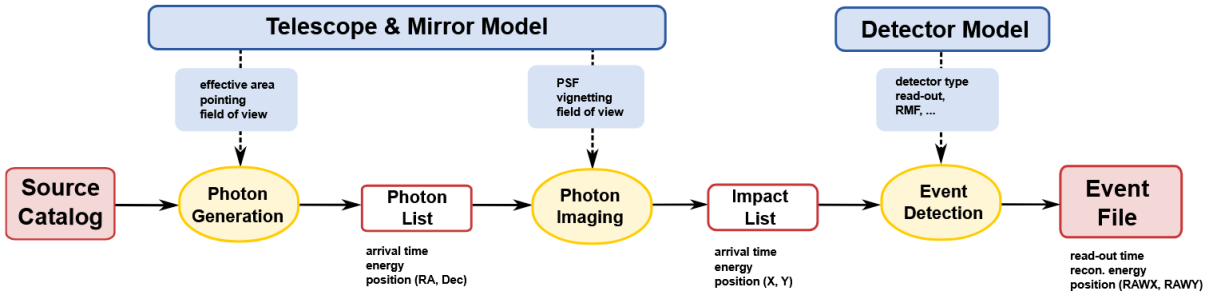

Figure 3.15: Flowchart of the three major functional blocks of SIXTE [7].

their physical properties. The file format used is SIMPUT (Simulation Input), a FITS-derivative format specifically designed for simulations. SIMPUT files can include a wide range of physically relevant attributes, such as spectra, temporal variability, sky position, and spatial extent. Accordingly, a SIMPUT file can represent a single point source, a large sky region, or even an entire all-sky distribution of X-ray sources [7].

For computational efficiency, only photons located within the field of view (FoV) of the telescope are generated. This requires providing the telescope with orientation parameters such as position and roll angle [7]. The output of this step is the photon list, which then serves as the input for the photon imaging block.

During photon imaging, photons with celestial coordinates are propagated through the telescope optics and mapped onto the detector coordinate system. Prior to this thesis, the imaging was performed exclusively using precomputed point spread functions (PSFs). In this approach, the impact position of a photon on the sensor is determined statistically based on the PSF rather than by explicitly tracing its trajectory through the optics. A PSF describes the image of a point source as it would appear on the detector, accounting for optical effects such as aperture vignetting, coma, and surface roughness. The impact list generated in this step, which records the sensor positions of photons, is then passed to the final stage.

In the event detection step, detector physics are simulated to produce an event file suitable for subsequent imaging and spectral analysis. The detector model incorporates properties such as pixel size, spatial resolution, and detector geometry, as well as characteristic noise patterns. Furthermore, different detection technologies can be modeled. For example, in simulating Athena's instruments, one may choose between the CCD-based Wide Field Imager (WFI) or the calorimeter-based X-IFU [7].

A key strength of SIXTE lies in its modular architecture, which allows individual components of the simulation chain to be exchanged or modified. For example, the Telescope and Mirror Model can be replaced independently of the detector model, enabling side-by-side comparisons of alternative designs. Figure 3.16 illustrates this capability: the same telescope model is simulated with two different detectors, demonstrating the scientific impact of instrument choice. Such simulations are valuable for evaluating the trade-offs between advanced but costly detectors and simpler alternatives.

Simulations are also essential when making the case for investing in new instruments or novel optical designs, particularly those not yet realized experimentally. This is where the contribution of the present thesis becomes relevant within the broader SIXTE framework. Currently, photon imaging in SIXTE relies on precomputed or measured PSFs. While measured PSFs are highly accurate, they are

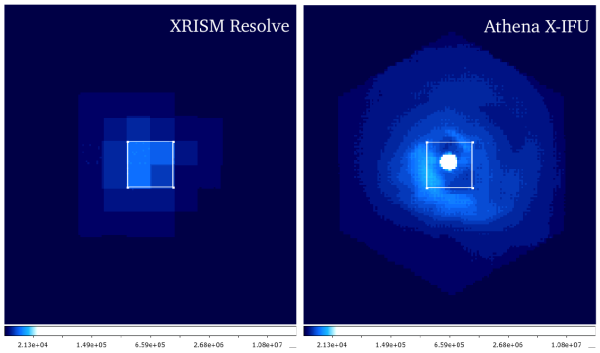

Figure 3.16: Comparison of XRSIM Resolve and the Athena X-IFU calorimeter detector for the Perseus Cluster.

only available for existing telescopes. The aim of this thesis is to extend the photon imaging engine with a ray-tracing module. In this approach, telescope optics are modeled directly in software, and photons are propagated through the mirrors until they reach the detector, at which point they are recorded in the impact list. This extension would allow simulations of instruments and optical systems that do not yet exist, eliminating dependence on external PSF data.

The details of the ray-tracing implementation within SIXTE will be presented in Chapter 4. Before that, the next section provides a general overview of ray-tracing principles.

3.4 Ray Tracing

Ray tracing is a fundamental technique in computer graphics used to achieve realistic rendering. Its primary goal is to generate two-dimensional images that represent three-dimensional scenes with high physical fidelity. While rasterization is another widely used rendering technique—particularly effective for real-time applications such as computer games—it does not accurately reproduce the physics of light. The advantage of rasterization lies in its comparatively low computational cost: objects in a 3-D scene are projected onto a 2-D image plane (the virtual camera), and additional techniques such as shading, light maps, and depth testing are applied to enhance realism. However, for truly photorealistic images, ray tracing provides a more physically accurate approach.

In ray tracing, light rays are followed as they traverse a scene. These rays, which represent photons, travel through space, interact with objects, and undergo reflection or refraction. For this thesis, we focus on *physics-based ray tracing*, in which the aim is to simulate optical effects as faithfully as possible according to physical models. Naturally, this approach is computationally far more expensive than rasterization, as it involves tracing millions of rays per image. Incorporating physical laws further increases the computational demand.

The use of ray tracing must therefore be justified by the application. For video games, maximizing frame rate is more important than physical accuracy, and thus rasterization is preferred. By contrast, in applications such as photorealistic rendering for movies, or in scientific simulations—such as modeling X-ray telescopes—real-time requirements can be relaxed, making physically accurate ray tracing highly advantageous.

This method allows us to describe how light is transported through a scene. Figure 3.17 shows an

Figure 3.17: Photorealistic rendered scene created with *pbrt* [9].

example rendered with the *pbrt* engine, which demonstrates several effects modeled by ray tracing: cast shadows, specular reflections of metallic surfaces, and the transparency of glass, all governed by physical models.

At its core, the ray-tracing algorithm is conceptually straightforward. A scene must contain at least two objects: a camera and a light source. As illustrated in Figure 3.18, there are two principal approaches to ray tracing. In *forward ray tracing*, rays are cast from the camera into the scene, reflecting and refracting until they reach a light source. In *backward ray tracing*, also known as *path tracing*, rays are emitted from the light source and traced until they arrive at the sensor.

Both approaches follow a recursive pattern: each time a ray intersects a surface, an interaction occurs, and new rays may be spawned depending on the optical properties of the surface. For example, in the case of partial transmission, part of the incoming light is reflected while another part is refracted, producing multiple secondary rays [23].

A ray can be formally described by its origin $\vec{o}$ and direction $\vec{d}$ as in Equation (3.9). An intersection with an object occurs when the parameter t positions the ray on a surface. If multiple intersections are possible, the closest one along the ray is selected.

$$r(t) = \vec{o} + t\vec{d} \tag{3.9}$$

Once an intersection is found, a scattering model is used to determine how the ray interacts with the surface. This interaction is commonly expressed in terms of the *bidirectional scattering distribution function* (BSDF), which combines two components: the *bidirectional reflectance distribution function* (BRDF), describing reflection, and the *bidirectional transmittance distribution function* (BTDF), describing transmission [24]. The BRDF in particular has been central to the development of physically based rendering, with Cook and Torrance [6] pioneering the use of microfacet reflection theory as a physically motivated model of surface roughness. The next section explores this theory in more detail.

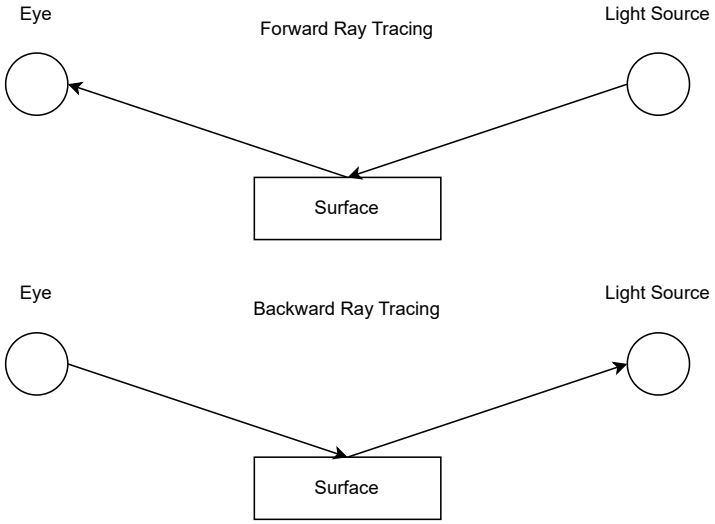

Figure 3.18: Comparison of forward ray tracing (from the camera) and backward ray tracing (from the light source).

A full treatment of BSDF theory is beyond the scope of this thesis. Unlike visual rendering applications, our focus is on simulating the behavior of X-ray photons interacting with telescope mirrors. In Chapter 4, we will describe in detail how X-ray photon–surface interactions are modeled within SIXTE. The next section, however, introduces the microfacet theory, which provides the theoretical foundation for modeling mirror surface roughness in the context of this work.

3.5 Microfacet Theory

As introduced in the previous section, the microfacet theory provides a framework for approximating surface roughness. Originally proposed by Torrance and Sparrow [34], the theory assumes that a surface is composed of many small facets, each behaving as a perfectly specular reflector. Realistic scattering behavior from rough surfaces can then be modeled statistically by approximating the distribution of these microfacet normals.

Figure 3.19 illustrates this concept. To reflect an incident vector d, the surface normal n—perpendicular to the macrosurface at a given point—is required. Equation (3.10) shows the reflection formula. Roughness can be simulated by perturbing the macrosurface normal n, resulting in a reflected direction vector d' that deviates from the ideal case. Rather than importing highly detailed meshes to represent surface irregularities (which would be prohibitively expensive in memory and computation), statistical distributions of microfacet normals n_f are used to model roughness efficiently.

Along the macrosurface normal n, assumed here to be $(0, 0, 1)$ in Cartesian coordinates, a distribution of microfacet normals n_f is defined. The degree of roughness determines the spread of these normals: highly polished, specular surfaces produce only small deviations, while rough surfaces exhibit much larger angular deviations.

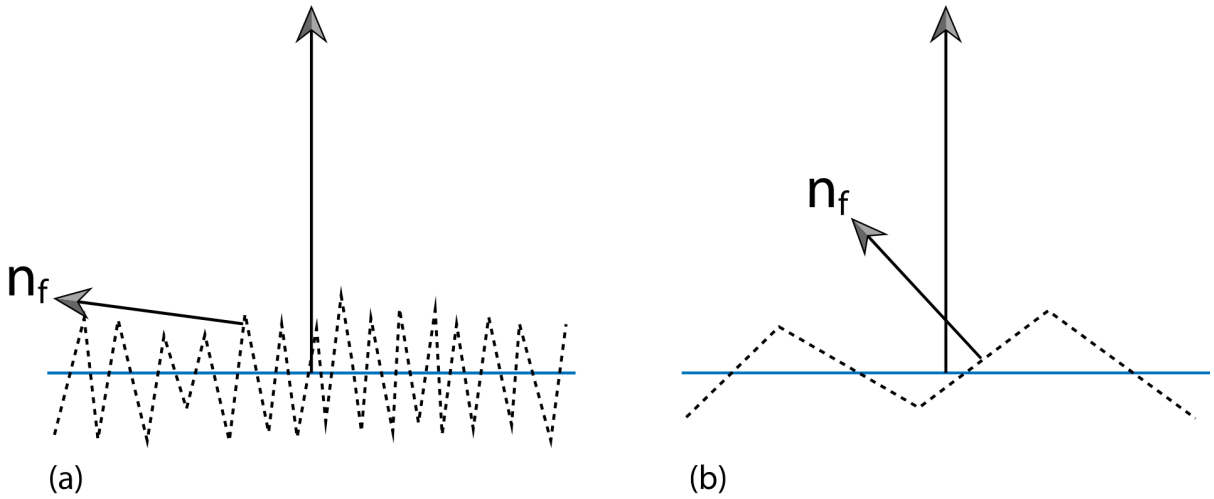

Figure 3.19: Microfacet distributions [19]. (a) Larger deviations of normals simulate a rougher surface. (b) Smaller deviations simulate a smoother, more specular surface.

$$d' = d - 2(d \cdot n)n \tag{3.10}$$

The distribution of microfacet normals $D(n_f)$ forms one part of the theory. The other part concerns the phenomena of masking, shadowing, and interreflection, as illustrated in Figure 3.20. These effects occur because microfacets may occlude one another. Masking reduces the number of outgoing rays, while shadowing blocks incoming rays. Interreflections, although theoretically present, are often neglected in practice. The shadowing–masking function G was first formalized by Smith [30] and has since been refined in subsequent work.

For this thesis, we concentrate on two commonly used distributions: the *Beckmann distribution* [3] and the *GGX distribution* [38]. In both cases, the roughness is parameterized by $\alpha > 0$, with smaller values corresponding to smoother surfaces (e.g., $\alpha < 0.01$ for mirror-like finishes).

The Beckmann distribution is given in Equation (3.11) [38], where θ_m is the angle between the macrosurface normal n and a microfacet normal n_f, and $\chi^+(a)$ is the positive characteristic function that equals one for $a > 0$ and zero otherwise.

$$D(n_f) = \frac{\chi^+(n_f \cdot n)}{\pi\alpha^2 \cos^4 \theta_n} \exp\left(-\frac{\tan^2 \theta_n}{\alpha^2}\right) \tag{3.11}$$

The corresponding shadowing function is given in Equation (3.12), where θ_v is the angle between the macrosurface normal and the incoming direction v. This version uses the approximation by Schlick [29], which Walter et al. [38] showed to have a relative error below 0.35%.

$$G_1(v, n_f) \approx \chi^+\left(\frac{v \cdot n_f}{v \cdot n}\right) \times \begin{cases} \dfrac{3.535a + 2.181a^2}{1 + 2.276a + 2.577a^2}, & \text{if } a < 1.6, \\ 1, & \text{otherwise,} \end{cases} \quad \text{with } a = (\alpha \tan \theta_v)^{-1} \tag{3.12}$$

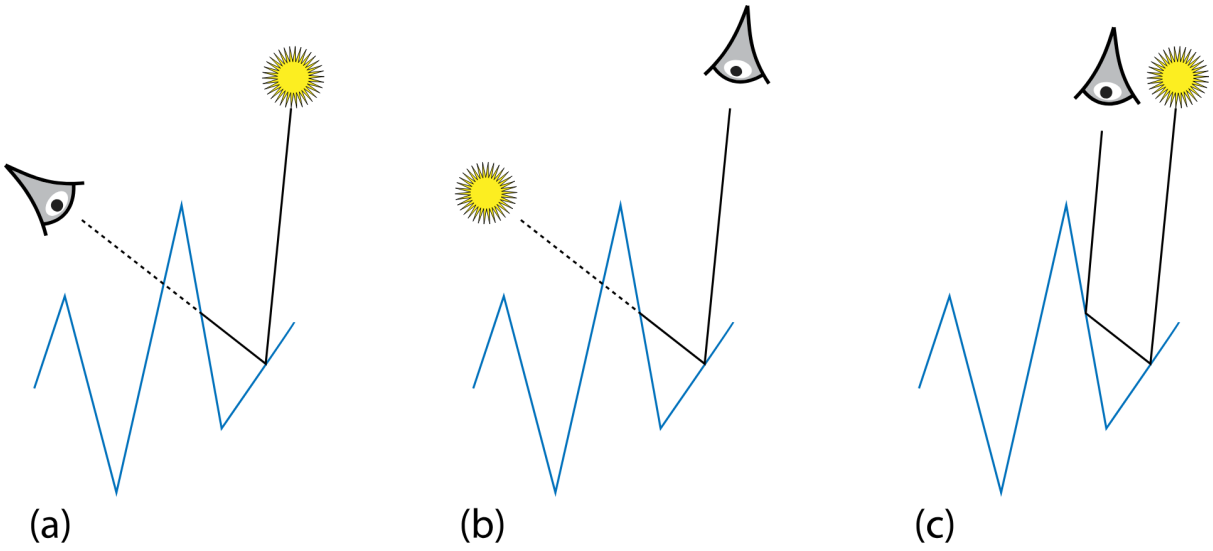

Figure 3.20: Illustration of masking (a), shadowing (b), and interreflection (c) [19].

For the GGX distribution, the normal distribution and shadowing–masking functions are given by Equation (3.13) and Equation (3.14), respectively:

$$D(n_f) = \frac{\alpha^2 \chi^+(n_f \cdot n)}{\pi \cos^4 \theta_n \left(\alpha^2 + \tan^2 \theta_n\right)^2} \tag{3.13}$$

$$G_1(v, n_f) = \chi^+\left(\frac{v \cdot n_f}{v \cdot n}\right) \frac{2}{1 + \sqrt{1 + \alpha^2 \tan^2 \theta_v}} \tag{3.14}$$

In summary, microfacet theory provides a compact yet powerful statistical model of surface roughness. By using distributions such as Beckmann or GGX, we can approximate the deviation of surface normals and account for shadowing–masking effects without explicitly modeling rough surfaces at the geometric level. This makes the theory particularly well suited for simulating the reflective properties of X-ray telescope mirrors, where microscopic surface irregularities significantly influence optical performance.

In the next chapter, we integrate these theoretical foundations into the SIXTE framework, extending its photon imaging capabilities with a ray-tracing module that incorporates microfacet-based surface modeling.

Chapter 4

Method

In this chapter provides a methodical description of the work conducted in this thesis. We begin by defining the functional and non-functional requirements of the software module, followed by an overview of the architectural design. Next, we describe the pipeline for creating optical components, including the mirror geometry and the optical blocking elements. The subsequent section details the simulation pipeline algorithm, where the relevant physical models—such as reflectivity and surface roughness—are introduced and the ray-tracing routine is explained in full detail. Finally, we discuss the simulation input and output formats. Together, these sections establish the overall framework of the ray-tracing extension for SIXTE and provide a foundation for the implementation details presented in the following chapter.

For clarification purposes, this chapter focuses on the substitution of the *Photon Imaging* step illustrated in Figure 3.15. As described in Chapter 3, the current implementation in SIXTE relies on precomputed or measured point spread functions (PSFs) to determine photon impact positions on the detector. While this method achieves scientifically useful accuracy in many cases, it has notable limitations: special features in the output data cannot always be explained, and PSFs may not be available for optical systems that have not yet been built or for which external sources do not provide measured data. To address these limitations, this thesis introduces a ray-tracing approach that replaces the PSF-based imaging engine. The following sections describe this approach in detail.

4.1 Requirements

Before any software product can be developed, it is essential that the requirements are clearly defined by all stakeholders. This ensures that development is aligned with the intended goals from the outset. In Section 4.1.1, we specify the functional requirements, while Section 4.1.2 outlines the non-functional (quality) requirements. The extent to which these requirements have been satisfied is evaluated in Chapter 6.

4.1.1 Functional Requirements

Functional requirements define what the product must be able to accomplish. For the ray-tracing module developed in this thesis, four fundamental requirements were identified:

1. **Ray tracing through telescope optics:** The system must simulate the optical components of an X-ray telescope such that rays originating from arbitrary X-ray sources can be traced through the telescope until they reach the focal plane, where the detector model continues the simulation.
2. **Support for different optical designs:** The system must provide functionality to create different telescope optics. At a minimum, the Wolter I optics and the Lobster-Eye optics must be supported.
3. **Configurable optical geometries:** The system must allow modification of geometric parameters, such as focal length and detector position, as well as mirror properties. This flexibility is required to represent different telescope designs and configurations.
4. **Inclusion of non-mirror components:** Since X-ray telescopes also include optical blocking structures, the system must support loading arbitrary mesh objects (e.g., baffles or support structures) in the form of STL files into the scene.
5. **Realistic surface roughness model:** Since the mirrors are not perfect reflectors, the ray tracer needs to incorporate surface roughness to simulate the scattering of photons accurately.

4.1.2 Non-Functional Requirements

For defining non-functional requirements, we adopt the arc42 Quality Model by Starke et al. [32], which provides a simplified and more practical alternative to the ISO 25010 standard [13]. This model is particularly suited for describing quality requirements in a structured and usable way.

Reliability

The ray-tracing module must be reliable in its simulation results. Specifically, the simulation output must always be consistent for identical inputs under identical simulation setups.

Furthermore, variations in the input should lead to the expected changes in the results. For example, the resulting PSF structures should vary appropriately when the input sources differ, even if the same off-axis angle is used.

Flexibility

The module must be flexible both for end users and for future development. For users, modifying the properties of telescope mirrors (see Section 4.2) should be straightforward, and switching between different mirror geometries should be supported with minimal effort.

For developers, the codebase should be designed in such a way that new mirror geometries and additional functionalities can be incorporated without extensive refactoring of existing components.

Efficiency

The introduction of ray tracing must not render simulations prohibitively slow. Certain efficiency requirements are therefore defined:

- The loading time for creating a telescope, including mirror surface generation and the import of external meshes, should not exceed five seconds.

- While ray tracing is inherently more computationally expensive than the existing PSF-based approach, the total simulation time (including telescope loading and photon propagation) should remain within a reasonable factor of the PSF method. Specifically, the runtime should not exceed twice the runtime of the conventional approach.

Usability

The ray-tracing option must be configurable within the existing SIXTE simulation pipeline. In addition, any external libraries required for the implementation should be straightforward to obtain, install, and integrate into the overall framework.

Operability

Since the module will be integrated into the SIXTE framework, it must provide clear and well-documented interfaces. This ensures that the module can be operated easily within SIXTE and that integration does not impose unnecessary complexity on users or developers.

4.2 Architecture

In this section, we present the overall software architecture of the ray-tracing module. We begin with a high-level view and progressively describe individual components, highlighting the design principles and rationale behind the chosen architecture.

SIXTE itself is designed as a modular software system. Modularity offers several advantages over monolithic design. A key benefit is compliance with the *Open–Closed Principle*, which states that software entities should be open for extension but closed for modification. This allows new functionality to be added without altering existing code. The *Single Responsibility Principle* ensures that each abstraction has a well-defined role, so that changes can be made in isolation. The *Interface Segregation Principle* further minimizes coupling by enforcing strict separation of interfaces. Together, these principles form part of the SOLID framework, which underpins robust software architecture.

These principles are already visible in SIXTE at the highest level of abstraction. Figure 4.1 illustrates the *Photon Imaging* step, implemented using the strategy pattern. Because the high-level class *PhotonImaging* adheres to the Open–Closed Principle, extending it with the ray-tracing module requires only the addition of a new strategy to the *ImagingStrategy* interface. The strict separation of modules prevents unnecessary coupling between components.

Proceeding one level deeper, we consider the *Raytracing* module itself. From the outset, the design was guided by the requirement for adaptability and extensibility. The first abstraction is the *MirrorModule* geometry, shown in Figure 4.2. The *MirrorModule* interface enables different optical systems to be defined, each inheriting from a common abstract class. In this thesis, two such systems are implemented: Wolter optics and Lobster-Eye optics. Their specific creation processes are described in Section 4.3.

At the next level, we examine the internal composition of telescope optics. In general, an optical system consists of three major parts:

1. *Optical reflecting parts* (e.g., mirrors),

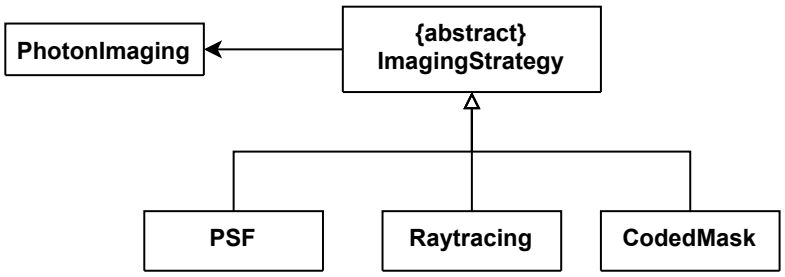

Figure 4.1: Integration of the ray-tracing module into the modular architecture of SIXTE.

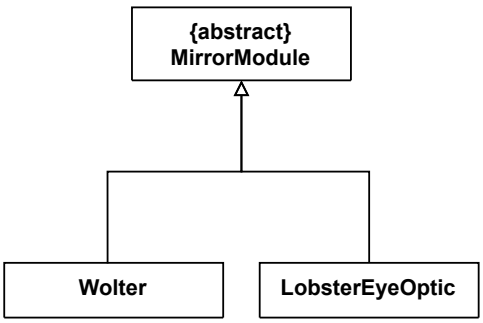

Figure 4.2: High-level abstraction of a mirror module.

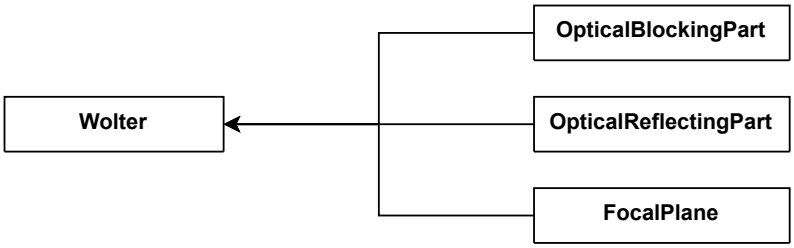

Figure 4.3: Design of the optical system of a Wolter telescope.

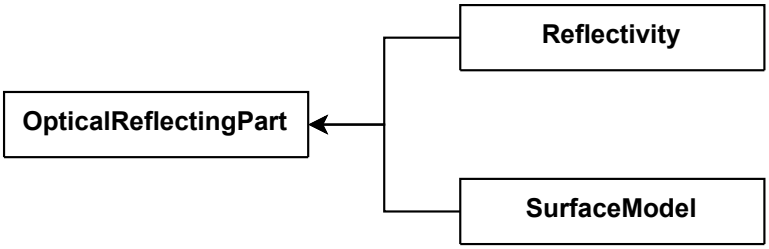

Figure 4.4: Characteristics of the *OpticalReflectingPart*.

2. *Optical blocking parts* (e.g., baffles, support structures), and

3. The *focal plane*.

There are multiple design options for representing these components in software. One approach would be to generalize all components under a base class *Shape* and distinguish between reflective, blocking, and focal-plane behavior at runtime. However, this design would violate the Single Responsibility Principle, as *Shape* would have multiple responsibilities. To remain aligned with SOLID principles, the architecture instead defines three separate classes, each corresponding to one of the above categories. This separation allows the software representation to mirror the physical composition of a telescope.

Figure 4.3 illustrates the design for Wolter telescopes. In physical terms, the *OpticalBlockingPart* corresponds to elements such as spiders (supporting structures for the mirrors) or inner baffles that prevent stray light from crossing between shells. The *OpticalReflectingPart* contains the mirrors themselves, which form paraboloid and hyperboloid shells depending on their placement in the coordinate system. The *FocalPlane* terminates the ray-tracing algorithm once a photon intersects it.

Lobster-Eye optics follow the same general structure, but with different implementations for each of the three components.

Each *OpticalReflectingPart* has three defining characteristics. First, it specifies its geometric surface, such as a paraboloid or hyperboloid. Second, it incorporates a *Reflectivity* module, which determines whether a photon is reflected or absorbed based on material properties and grazing incidence angle. Third, it integrates a *SurfaceModel*, which perturbs the surface normal to model roughness effects. This design allows additional surface models to be added without altering the existing structure. Figure 4.4 summarizes these relationships.

Finally, the *Ray* class itself is defined, as shown in Figure 4.5. A ray contains a position vector

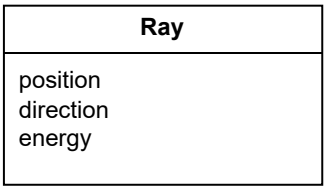

Figure 4.5: Class diagram of a ray.

and a direction vector, which together describe its trajectory in space. In addition, each ray carries an energy value. In the context of visible light, this energy would correspond to color; in X-ray simulations, however, it directly determines reflectivity, as discussed in Chapter 3.

In summary, this architecture enables the accurate software representation of telescope mirror modules while remaining highly adaptable. New reflective geometries can be incorporated with minimal effort, whether as analytical models (e.g., defined with Chebyshev polynomials) or as imported meshes.

With the architectural foundations established, the next section describes the algorithms used for constructing the optical components.

4.3 Creation Pipeline for the Optics

As described in the previous section, a telescope mirror module consists of three essential components: the optical blocking part, the reflecting part, and the focal plane. These elements must be instantiated when the mirror module is initialized. This section provides an overview of the process of creating a mirror module. We first describe the data format used to store all required information, followed by the creation algorithm that generates the module. In doing so, we extend the interfaces of the classes introduced in the previous section.

For storing the information in a coherent way, we use the structure illustrated in Table 4.1. Each keyword corresponds to a specific aspect of the mirror module. For example, *Mirror Module Type* specifies the geometric architecture of the telescope, as well as general properties such as focal length and mirror height. The *Mirror Configuration* keyword defines the positions of the mirrors, with specific values depending on the chosen optical geometry. The *Surface* keyword describes the roughness model and the mirror coating material for computing reflectivity. Finally, the keywords for the *Optical Blocking Part* and the *Focal Plane* specify the corresponding structures in the scene. Chapter 5 will provide concrete examples of the implementation for each keyword.

The creation algorithm begins by establishing a global Cartesian coordinate system, which is defined as right-handed with the z-axis pointing upward from the focal plane to the source. In this orientation, the y-axis points to the right and the x-axis points forward, as shown in Figure 4.6.

The first step of the algorithm is to determine which mirror module is to be created, based on the *Mirror Module Type* keyword from Table 4.1. Next, the *Mirror Configuration* defines the optical reflecting parts, and the individual mirrors are instantiated and inserted into the coordinate system. Each mirror is assigned its surface parameters, including reflectivity and surface roughness. The optical blocking part

Keyword
Mirror Module Type
Mirror Configuration
Surface
Optical Blocking Part
Focal Plane

Table 4.1: Format for storing mirror module information.

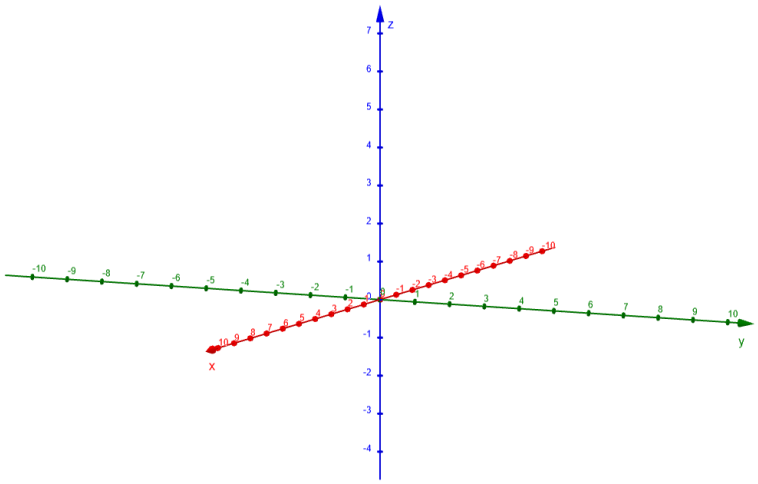

Figure 4.6: Right-handed coordinate system with the z-axis pointing upwards.

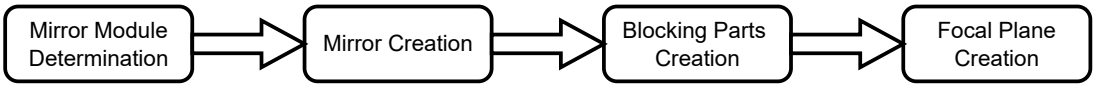

Figure 4.7: Creation algorithm for mirror modules.

is then added, followed by the focal plane, where the photons will ultimately be focused. The overall process is illustrated in Figure 4.7.

For Wolter and Lobster-Eye optics, different mirror types are created during the mirror-creation step. In the case of Wolter-I optics, paraboloids and hyperboloids are instantiated. As shown in Figures 3.9 and 3.10, the inner shells must be lowered so that they align with the height of the outermost shell.

There are several ways to reposition objects in a coordinate system. One option would be to alter the analytic equations of the Wolter mirrors directly. For instance, paraboloid and hyperboloid functions could be shifted by adding factors to their equations. However, this approach has significant drawbacks: shifting in the $x-y$ plane would require further modifications to the functions, and incorporating rotations would make the equations increasingly complex, making analytic intersection calculations error-prone.

A more general and robust method is to use coordinate transformations. Here, objects remain defined in their local coordinate systems, while translations and rotations are applied relative to the global system. During simulation, rays are transformed into the local coordinate system of each mirror before intersection calculations are performed. This approach preserves the stability of the analytic functions, simplifies intersection handling, and allows additional complexity (such as rotations or scaling) to be incorporated without altering the underlying equations.

For a translation, a vector v is transformed by subtracting an offset o, yielding the translated vector v', as shown in Equation 4.1.

$$v' = v - o \tag{4.1}$$

In the case of Lobster-Eye optics, it is inefficient to instantiate all individual channels. Instead, a single prototype channel is created in its local frame. During the simulation, rays are transformed into this local coordinate system, significantly reducing complexity. This forms the foundation for the simulation process described in the next section.

4.4 Simulation Pipeline for the Optics

This section describes in detail the simulation process. It is structured into several parts. First, we present the general algorithm for simulating a mirror module, explaining how ray tracing is integrated into the process and extended towards physics-based ray tracing. We then discuss the specific adaptations required for different mirror geometries.

4.4.1 Ray Tracing the Optics

Ray tracing is ideally suited for simulating telescope optics, as it enables the accurate propagation of photons and the study of optical effects. As discussed in Chapter 3, two general strategies exist for ray

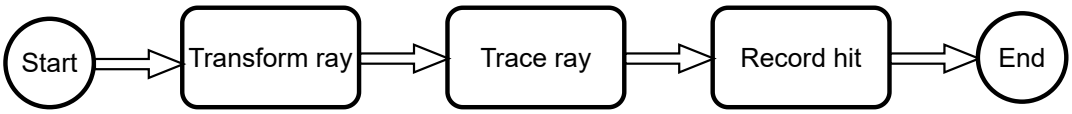

Figure 4.8: High-level view of the ray-tracing algorithm in SIXTE.

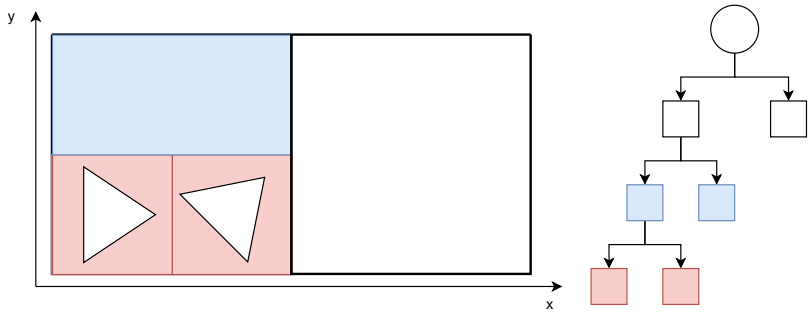

Figure 4.9: Bounding-box hierarchy for efficient intersection finding.

tracing. Given the requirements of SIXTE, we adopt forward ray tracing, also known as path tracing.

We begin with a high-level view of the algorithm. In accordance with the black-box principle, each step is separated conceptually to ensure modularity. As shown in Figure 4.8, the algorithm replaces the PSF imaging method in the SIXTE pipeline. First, a photon originating from a given sky position (RA, DEC) is transformed into the global coordinate system. Next, the starting position of the ray is set (details are provided in Chapter 5). The ray then undergoes the ray-tracing routine, and if it intersects the detector on the focal plane, its impact is recorded and passed on to the detector simulation within SIXTE.

At a lower level, the ray-tracing routine proceeds as follows:

- Find the closest intersection between the ray and scene objects.
- Apply the corresponding surface interaction.
- If the surface is reflective, compute the reflected ray and repeat the routine.
- Terminate if the ray exceeds the maximum number of reflections or intersects a termination object such as the detector.

Although conceptually simple, the efficiency of intersection testing is critical. A naïve approach would test the ray against every object in the scene and select the closest intersection. However, with scenes containing thousands of triangles and simulations involving millions of rays, this would result in excessive and redundant calculations.

To address this, bounding volumes are employed. Instead of testing against every primitive, objects are grouped within bounding boxes defined by their minimum and maximum coordinates in Cartesian space. Figure 4.9 illustrates a bounding box hierarchy that significantly reduces the number of intersection tests required.

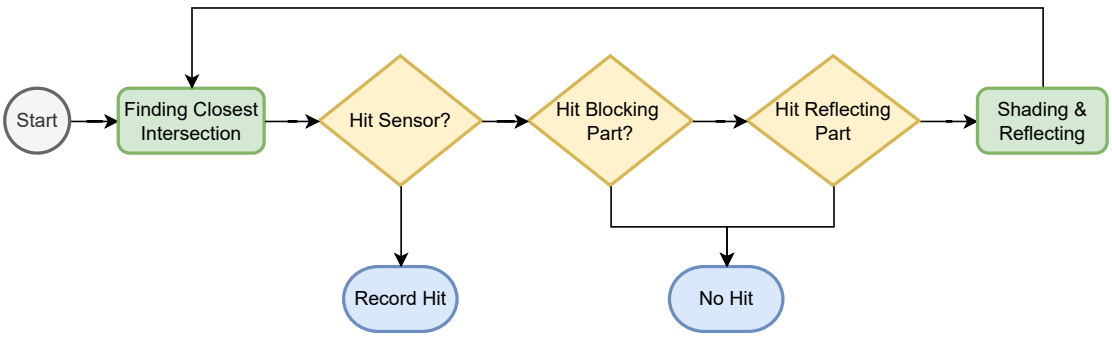

Figure 4.10: Ray-tracing routine.

After identifying the closest intersection, postprocessing determines the physical outcome. In traditional ray tracers, this would correspond to shading; here, in the X-ray regime, the outcome depends on the photon's energy and grazing angle. Three cases are possible:

- The ray intersects an *optical blocking part*: the ray is terminated, as high-energy photons will not reflect from these structures.
- The ray intersects the *focal plane*: the impact position is recorded for later analysis.
- The ray intersects an *optical reflecting part*: the reflectivity is computed stochastically, based on incidence angle and photon energy. If the photon is reflected, the surface roughness model perturbs the normal vector, and the ray is reflected accordingly.

This procedure is summarized in Figure 4.10.

Returning to the intersection step, the result is the scalar parameter t in Equation (3.9), which locates the ray on the surface. The surface normal n at the intersection point is then used in the reflection formula (Equation 3.10).

For meshes composed of triangles, the intersection is calculated in two stages: first, by finding the intersection with the plane defined by the triangle, and second, by testing whether the intersection point lies inside the triangle.

The general equation of a plane is given in Equation Equation (4.2).

$$L : \quad ax + by + cz + d = 0 \tag{4.2}$$

A ray r is expressed as in Equation Equation (4.3).

$$r = p + td = \begin{pmatrix} p_x + td_x \\ p_y + td_y \\ p_z + td_z \end{pmatrix} \tag{4.3}$$

Substituting Equation (4.3) into Equation (4.2) yields Equation (4.4), which determines whether a valid solution exists. The possible outcomes are summarized in Equation (4.5).

$$
\begin{aligned}
&a(p_x + t d_x) + b(p_y + t d_y) + c(p_z + t d_z) + d = 0 \\
&\underbrace{a p_x + b p_y + c p_z + d}_{A} + t \cdot \underbrace{(a d_x + b d_y + c d_z)}_{B} = 0 \\
&t = -\frac{A}{B}
\end{aligned} \tag{4.4}
$$

$$
t = \begin{cases} 0 & \text{if } p \text{ lies on the plane} \\ \mathbb{R} & \text{if one intersection exists} \\ \text{undefined} & \text{if no intersection exists} \end{cases} \tag{4.5}
$$

The surface normal of a plane is constant everywhere and can be obtained directly as shown in Equation (4.6). For numerical stability, it is normalized according to Equation (4.7).

$$
n_x = \frac{\partial L}{\partial x} = a, \qquad n_y = \frac{\partial L}{\partial y} = b, \qquad n_z = \frac{\partial L}{\partial z} = c \tag{4.6}
$$

$$
\hat{n} = \frac{n}{|n|} \tag{4.7}
$$

In the next subsection, we extend this intersection framework to analytic surfaces such as paraboloids and hyperboloids, which define the Wolter-I optics.

4.4.2 Wolter I Mirror Intersections

While triangle meshes could in principle be used to approximate all surfaces, they are not sufficiently accurate for this application and introduce discrete discontinuities at triangle boundaries. For high-precision modeling, the mirror surfaces must therefore be described analytically using paraboloid and hyperboloid equations, as introduced in Chapter 3.

Unlike the linear solution obtained for plane intersections, intersections with paraboloid or hyperboloid surfaces lead to quadratic equations. This is expected, since rays may intersect both sides of these surfaces. Consequently, we obtain up to two valid solutions for the intersection parameter t, compared to a single solution in Equation Equation (4.5). In the following, we first examine the paraboloid case, then the hyperboloid, and finally discuss how to interpret multiple solutions and compute surface normals.

Paraboloid

Equation (3.4) defines a paraboloid in two dimensions. Extending this to three dimensions yields Equation (4.8):

$$
P : \; z = \frac{x^2 + y^2}{2p} - \frac{p}{2} \tag{4.8}
$$

Substituting the ray equation into x, y, and z results in a quadratic equation in t, which can be solved using the quadratic formula (Equation (4.9)).

$$t_{1,2} = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A} \tag{4.9}$$

The explicit substitution yields Equation (4.10):

$$p_z + td_z = \frac{(p_x + td_x)^2 + (p_y + td_y)^2}{2p} - \frac{p}{2} \tag{4.10}$$

A full derivation is given in Appendix B. The resulting coefficients are summarized in Equation Equation (4.11):

$$\begin{aligned} A &= d_x^2 + d_y^2 \\ B &= 2(p_x d_x + p_y d_y - p d_z) \\ C &= p_x^2 + p_y^2 - p^2 - 2pp_z \end{aligned} \tag{4.11}$$

If the ray direction is parallel to the z-axis ($d_x = d_y = 0$), then $A = 0$, leading to undefined behavior in the quadratic formula. In this case, the simplified solution given in Equation (4.12) can be used:

$$t = \frac{-p(p + 2p_z) + p_x^2 + p_y^2}{2pd_z} \tag{4.12}$$

When two valid solutions t_1 and t_2 exist, the smaller positive value corresponds to the first intersection encountered along the ray. Negative solutions are disregarded, as they would correspond to intersections behind the ray origin and are therefore physically irrelevant.

To compute the surface normal at an intersection point, we take the partial derivatives of Equation (4.8), as shown in Equation (4.13), and normalize the result.

$$n_x = \frac{\partial P}{\partial x} = \frac{x}{p}, \quad n_y = \frac{\partial P}{\partial y} = \frac{y}{p}, \quad n_z = \frac{\partial P}{\partial z} = -1 \tag{4.13}$$

For paraboloids, the normal vector points outward by convention. To maintain consistency across all surface types, we flip the orientation if necessary so that the normal always points toward the reflective side of the surface.

With these two ingredients—the quadratic solution for t and the surface normal—the intersection of rays with paraboloid surfaces is fully defined. The procedure for hyperboloid surfaces follows an analogous approach.

Hyperboloid

Equation (4.14) defines a hyperboloid in three dimensions. As in the paraboloid case, substituting the ray equation r results in a quadratic equation, which is solved using the quadratic formula.

$$H : \frac{x^2}{b^2} + \frac{y^2}{b^2} - \frac{(z-c)^2}{a^2} = -1 \tag{4.14}$$

The detailed derivation of the coefficients is given in Appendix B. The final expressions are summarized in Equation (4.15):

$$\begin{aligned} A &= \frac{d_x^2 + d_y^2}{b^2} - \frac{d_z^2}{a^2} \\ B &= 2\left(\frac{p_x d_x + p_y d_y}{b^2} - \frac{(p_z - c)d_z}{a^2}\right) \\ C &= \frac{p_x^2 + p_y^2}{b^2} - \frac{(p_z - c)^2}{a^2} + 1 \end{aligned} \tag{4.15}$$

Because the hyperboloid consists of two separate shells, not all rays will produce valid intersections. Some rays will miss the surface entirely, while others may only touch the vertex of one of the shells. If both A and B are zero, no intersection exists. If $A = 0$ but $B \neq 0$, a single solution can be obtained from Equation (4.16):

$$t = -\frac{C}{B} \tag{4.16}$$

As with the paraboloid, the surface normal at an intersection point is computed from the gradient of the implicit surface equation. For the hyperboloid, this yields:

$$n_x = \frac{\partial H}{\partial x} = \frac{x}{b^2}, \quad n_y = \frac{\partial H}{\partial y} = \frac{y}{b^2}, \quad n_z = \frac{\partial H}{\partial z} = -\frac{z-c}{a^2} \tag{4.17}$$

With these results, intersections and surface normals can be computed for both paraboloid and hyperboloid surfaces, enabling the ray-tracing of Wolter I telescopes. In the next section, we turn to the description and simulation of Lobster-Eye optics.

4.4.3 Lobster-Eye Channel Intersections

The Lobster-Eye optics produce focal-plane images that are less intuitive than those of Wolter optics, where the geometry essentially acts as a magnifying system. Instead of mirror shells, Lobster-Eye optics consist of billions of microscopic pores. While it is feasible to model Wolter optics with fewer than 100 shells, explicitly constructing and simulating every pore of a Lobster-Eye optics is computationally intractable. Therefore, we develop a methodological approach: first describing the principle of simulating individual pores, and then deriving the mathematical framework for modeling a single representative pore while transforming incoming rays accordingly.

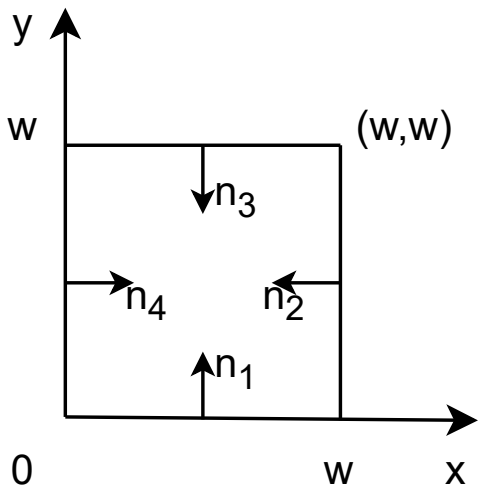

Figure 4.11: Top view of a single pore with wall definitions and corresponding inward-facing normals.

Channel Description

A single pore channel can be approximated by four planar walls forming a rectangular tube of length L and width w. Unlike paraboloid or hyperboloid surfaces, no curved geometry is required. Each wall is described as a plane according to Equation (4.2), and can be expressed in the form given in Equation (4.18):

$$W1 : y = 0, \quad W2 : x = w, \quad W3 : y = w, \quad W4 : x = 0 \tag{4.18}$$

Since planes extend infinitely, the valid reflecting regions must be restricted to the finite pore geometry. Each wall is therefore bounded by the pore's width w and length L, defining a rectangular region. The surface normals are oriented inward, following the convention that the reflective side always faces the interior of the pore.

Figure 4.11 illustrates the pore geometry with surface normals, while Table 4.2 summarizes the valid regions and normals for each wall.

This description provides the local geometry of a single pore. However, in its current form the channel is not yet correctly positioned within the global coordinate system, nor does it account for the vast number of pores required in real Lobster-Eye optics. The following section addresses how to simulate the global optic by transforming rays into the local frame of this representative pore.

Wall No.	Valid Area (min, max)	Surface Normal
1	(0,0,0) – (w,0,L)	(0,1,0)
2	(w,0,0) – (w,w,L)	(-1,0,0)
3	(0,w,0) – (w,w,L)	(0,-1,0)
4	(0,0,0) – (0,w,L)	(1,0,0)

Table 4.2: Bounding coordinates and surface normals for the four pore walls.

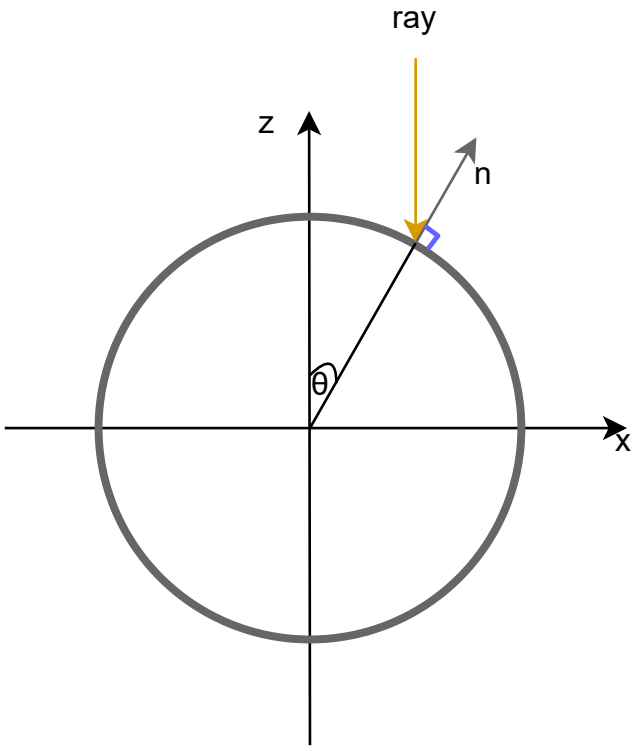

Figure 4.12: Rotation angle θ required to align a ray with the orientation of the pore at a given position.

Ray Transformation to the Local Channel Frame

To simulate the Lobster-Eye geometry efficiently, each incoming ray is transformed into the local coordinate frame of a representative pore. Once the ray propagates through the pore and reaches its exit plane, it is transformed back into the global frame. This procedure combines translation and rotation operations. A similar approach was already used for Wolter optics, where rays were translated into the local frames of individual shells. For the Lobster-Eye case, however, rotation is also required.

Figure 4.12 shows the rotation angle in the x-direction for aligning the ray with the pore axis. An

Figure 4.13: Illustration of different rotation strategies. Middle: base pore without rotation. Left: unstable configuration with z-axis rotation. Right: stable configuration with rotation only about the x- and y-axes.

analogous rotation must be performed in the y-direction. Conceptually, the rotation corresponds to projecting the incoming ray onto the surface of a sphere, since Lobster-Eye optics are spherical by design.

However, correctly rotating the ray is non-trivial. Unlike Wolter mirrors, the pores form a quadratic grid structure. Arbitrary rotations about the z-axis would distort this structure, as illustrated in Figure 4.13. To preserve the grid, the rotation must therefore be restricted to the x- and y-directions only. This constraint makes Rodrigues' rotation formula unsuitable, since it performs rotations around arbitrary axes. Instead, Euler angles are employed, allowing independent rotations about each coordinate axis.

The required Euler angles are given in Equation (4.19), where n denotes the surface normal at the impact position.

$$\alpha = \arctan 2(-n_y, \sqrt{n_x^2 + n_z^2}), \qquad \beta = \arctan 2(n_x, n_z) \tag{4.19}$$

The corresponding rotation matrices about the x- and y-axes are:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}, \qquad R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}. \tag{4.20}$$

By convention, the rotation about y is applied first, followed by the rotation about x. The combined rotation matrix R is then used to transform the ray direction vector d into the local channel frame (Equation (4.21)):

$$R = R_y R_x, \qquad d_{\text{local}} = R^T d \tag{4.21}$$

The position of the ray is initialized within the local frame by selecting a random x and y within the pore aperture and setting $z = L$, the pore length. The ray can then propagate through the pore, undergoing reflections until it reaches the exit plane. At this point, the ray is transformed back into the

global coordinate system. The new position is obtained by shifting the impact point along the surface normal by the pore length L (Equation (4.22)):

$$p_{\text{new}} = p_{\text{old}} - nL \tag{4.22}$$

Finally, the direction vector is rotated back to the global frame by applying R (Equation (4.23)):

$$d_{\text{new}} = Rd \tag{4.23}$$

This procedure enables efficient tracing of rays through the Lobster-Eye optic without explicitly simulating every pore. The wall thickness has been neglected here, since the pores are sufficiently small to approximate perfect open channels. Nevertheless, to account for photon absorption at pore walls, an additional statistical factor can be introduced to eliminate a fraction of rays, thereby reproducing the effective transmission properties of the microchannel plate.

4.5 Microfacet Surface

To approximate realistic surface roughness, we employ the microfacet theory introduced in Chapter 3. The idea is to perturb the surface normal n at the intersection point by sampling from the microfacet normal distribution. For convenience, the distribution is assumed to be centered at $(0, 0, 1)$, denoted as n_m. The task then becomes to rotate the ray's direction vector d into the coordinate system of n_m such that the relative angle between d and n is preserved.

We achieve this rotation using Rodrigues' rotation formula, which rotates a vector around a given axis a by an angle θ (4.24):

$$d_{\text{rot}} = d\cos\theta + (a \times d)\sin\theta + a(a \cdot d)(1 - \cos\theta) \tag{4.24}$$

The rotation axis is obtained from the cross product of the surface normal n and the reference normal n_m, normalized as shown in Equation (4.25):

$$a = \frac{n \times n_m}{|n \times n_m|} \tag{4.25}$$

The corresponding angle θ is computed from Equation (4.26), where the arctan 2 formulation resolves ambiguities:

$$\theta = \arctan 2(|a|, n \cdot n_m) \tag{4.26}$$

With this procedure, the direction vector d can be rotated into the coordinate system defined by n_m. After sampling a perturbed microfacet normal n'_m, representing the effect of surface roughness, the inverse rotation (using $-\theta$) is applied to transform n'_m back into the original surface coordinate system.

For sampling a microfacet normal, we will look at both the Beckmann and GGX microfacet distribution. This is illustrated in Equation (4.27) to (4.30). ξ is a random number in the range $(0, 1)$ and those formulae describe polar coordinates which we then have to convert to cartesian coordinates. α is the value determining the roughness of the surface. The higher it is, the rougher the surface is.

$$\theta_b = \arctan\sqrt{-\alpha_b^2 \log(1-\xi_1)} \tag{4.27}$$

$$\phi_b = 2\pi\xi_2 \tag{4.28}$$

$$\theta_{ggx} = \arctan\left(\frac{\alpha_{ggx}\sqrt{\xi_3}}{\sqrt{1-\xi_3}}\right) \tag{4.29}$$

$$\phi_{ggx} = 2\pi\xi_4 \tag{4.30}$$

The conversion is defined in Equation (4.31). This is basically the opposite of the same mechanism needed for the Euler rotations.

$$x = \sin\theta\cos\phi \qquad y = \sin\theta\sin\phi \qquad z = \cos\theta \tag{4.31}$$

In Chapter 6, we will try to determine a realistic factor for α. We will compare this with a real PSF measurement and try to mimic the scattering distribution. For that, we define the metric here, which we will use then.

A radial distribution histogram is ideal to compare point sources on axis which result into a PSF. For that, we define a certain binning, where each bin is a radial interval with a radius r. Because outer rings have more area with the interval radius than inner rings, we normalize against that. Additionally, we normalize the PSFs themselves, so that the amount of counts on the focal plane is relative to each other and not absolute.

In the end, we compare the normalized radial distribution histograms by division so that in the ideal case each bin approximates 1. This would then lead to a good scattering simulation for the surface roughness with the microfacet distribution function. More on that in Chapter 6.

This approach allows the incorporation of surface roughness into the ray-tracing routine for different reflective geometries. In this chapter, we have developed the methodological framework for simulating both Wolter I and Lobster-Eye optics. Furthermore, we have described the ray-tracing algorithm in detail and outlined how the microfacet theory is applied to approximate the scattering behavior of real mirror surfaces. The next chapter turns to the implementation details of these methods.

Chapter 5

Implementation

This chapter presents the implementation of the ray-tracing routine described in the previous chapter. We begin by outlining the background of SIXTE and the integration of the ray-tracing module into its existing framework. Subsequently, we discuss the incorporation of Intel's Embree library as the underlying acceleration structure for efficient intersection handling. Finally, we provide a step-by-step description of the complete routine for simulating photon propagation through ray tracing.

5.1 Overview of SIXTE's Environment

SIXTE originated as a C-based project and has since been almost entirely ported to C++17. To ensure long-term maintainability and compatibility, the number of external dependencies is deliberately kept to a minimum. At present, SIXTE relies only on the Boost library and, for certain algebraic calculations, selected functions from CGAL.

A key design principle of SIXTE is that external libraries are not embedded directly into the core code. Instead, they are encapsulated within wrapper files, which enables the substitution of libraries with minimal modifications to the main code base.

Instrument definitions are provided as external configuration files that SIXTE reads at the start of a simulation. This approach reinforces modularity and offers both developers and users the flexibility to extend or replace instruments without altering the core framework.

The adoption of CMake has further streamlined the build and installation process. External libraries that are not located in standard system paths can easily be specified during the CMake configuration step, simplifying the integration of additional dependencies.

SIXTE also provides a set of user-facing tools for conducting simulations and analyzing results. One key tool is `sixtesim`, which executes the complete simulation pipeline, from reading the SIMPUT input file to producing the final event file. Another example is `imgev`, which generates images from the event file output. In Chapter 6, these and other tools will be employed to evaluate the ray-tracing implementation.

5.2 Integration of the Ray-Tracing Module

Within SIXTE, the photon imaging step is encapsulated in the `PhotonImaging` class. At present, three different imaging approaches are implemented: the traditional PSF imaging, the coded-mask technique (not relevant for this thesis), and the newly introduced ray-tracing method. The imaging strategy is configured through a human-readable XML file, enabling the user to select the desired approach.

This setup provides an ideal case for applying the *strategy pattern*. As shown in Listing 5.1, the function `readImagingType()` returns the imaging type specified in the XML configuration. Depending on this choice, the corresponding imaging strategy is instantiated and subsequently executed as a functor. The creation of the mirror module, which underpins the ray-tracing strategy, will be discussed in detail later in this chapter.

```
auto imaging_type = readImagingType(xml_data);

switch (imaging_type) {
  case ImagingType::PSF:
    return PSFImagingStrategy(xml_data);

  case ImagingType::CODED_MASK:
    return CodedMaskImagingStrategy(xml_data);

  case ImagingType::RAYTRACING:
    return RaytracingImagingStrategy(xml_data);

  default:
    throw SixteException("Unknown imaging type");
}
```

Listing 5.1: Imaging strategy implementation.

Once the imaging strategy has been created, the simulation process can proceed: photons generated in the photon-creation step are passed to the imaging component.

Inside the functor of the `RaytracingImagingStrategy`, the relevant photon properties—position, direction, and energy—are retrieved, as illustrated in Listing 5.2. For internal consistency, the photon energy is converted from keV to eV. The mirror module exposes the method `ray_trace()`, which accepts a ray object as input and returns a `std::optional<Ray>`. The use of `std::optional` represents modern C++ best practice, eliminating the need for null-pointer checks and making the interface safer and more expressive.

If the photon successfully reaches the sensor, the impact position is returned and converted into the appropriate unit system. The result is then wrapped in a `SixtePhoton` object, which continues through the SIXTE simulation pipeline.

```
Ray ray = {position, photon_direction, photon.energy()*1000};
std::optional<Ray> result = mirror_module_->ray_trace(ray);
if (!result.has_value())
    return std::nullopt;
result.value().set_position(result.value().position() / 1000); //
    Conversion to [m]
```

```
return SixtePhoton(photon.time(), photon.energy(),
                   SixtePoint(result.value().position().x,
                              result.value().position().y, 0),
                   photon.photon_metainfo());
```

Listing 5.2: Starting the ray-tracing routine in `PhotonImaging`.

5.3 Embree

5.3.1 Overview

Embree is a high-performance ray-tracing library developed by Intel, supporting x86, ARM, and Intel Arc GPUs [37]. Since intersection finding is at the core of any ray-tracing algorithm, Embree provides optimized routines to perform this task with high efficiency. Although other libraries exist, Embree was chosen for SIXTE because it is open source under the Apache 2.0 license and allows the integration of only the minimal functionality required for the application. This ensures that the external dependency remains compact and can be bundled directly into the SIXTE package, avoiding additional installation steps for the user.

When coding with Embree, its API design must be followed. The central abstraction is the `RTCDevice`, to which one or more `RTCScene` objects are attached, as shown in Listing 5.3.

```
// Initialization for RTCDevice and RTCScene
RTCDevice device = rtcNewDevice(NULL);
RTCScene scene = rtcNewScene(device);
```

Listing 5.3: Initialization of an Embree scene.

Rays are represented in Embree by the `RTCRayHit` structure, which combines `RTCRay` and `RTCHit`. The former describes the ray itself, while the latter contains information about the intersection point.

Listing 5.4 shows the definition of `RTCRay`. Instead of vector structures, all components are represented as `float` values. The origin (`org`) and direction (`dir`) fields are self-explanatory. The attributes `tnear` and `tfar`, however, play a central role in intersection finding. While `tnear` excludes intersections that are closer than this threshold, `tfar` defines the maximum distance at which intersections are considered. Whenever a new closest intersection is found, `tfar` is updated accordingly. Setting `tnear` slightly above zero is good practice to avoid numerical instabilities, e.g., when a reflected ray might otherwise be considered to start behind the surface. The integer fields and the `time` attribute are not relevant for the purposes of this thesis.

```
struct RTC_ALIGN(16) RTCRay
{
  float org_x, org_y, org_z; // ray origin
  float tnear;               // start of ray segment
  float dir_x, dir_y, dir_z; // ray direction
  float time;                // ray time (e.g., motion blur)
  float tfar;                // end of ray segment (set to hit
   distance)
  unsigned int mask;         // ray mask
```

```
  unsigned int id;            // ray ID
  unsigned int flags;         // ray flags
};
```

Listing 5.4: RTCRay structure [37].

To complement this, RTCHit (Listing 5.5) describes the intersection. It includes the surface normal at the intersection point, the geometry and primitive IDs, and barycentric coordinates (u, v), which are particularly useful for interpolating positions on triangle meshes.

```
struct RTCHit
{
  float Ng_x, Ng_y, Ng_z;  // geometry normal
  float u, v;              // barycentric coordinates
  unsigned int primID;     // primitive ID
  unsigned int geomID;     // geometry ID
  unsigned int instID[RTC_MAX_INSTANCE_LEVEL_COUNT]; // instance ID
};
```

Listing 5.5: RTCHit structure [37].

Embree also provides object-oriented mechanisms for adding new geometries to a scene. The process is illustrated in Listing 5.6. A geometry object of type RTCGeometry is first created and associated with the device. Embree supports a variety of RTCGeometryType values, such as triangles or quads for meshes. Of particular relevance for this thesis is the RTC_GEOMETRY_TYPE_USER, which allows user-defined geometries to be integrated—essential for describing analytical surfaces such as paraboloids and hyperboloids.

After defining the geometry (e.g., by specifying vertices or providing user-defined callbacks), the geometry must be committed via rtcCommitGeometry(). It is then attached to a scene with rtcAttachGeometry(), which returns a unique geometry ID. Finally, the reference is released using rtcReleaseGeometry() to ensure proper memory management.

```
RTCGeometry geometry = rtcNewGeometry(device, RTCGeometryType);
/**
 * Geometry definition (vertices, indices, or callbacks).
 */
rtcCommitGeometry(geometry);
int geomID = rtcAttachGeometry(scene, geometry);
rtcReleaseGeometry(geometry);
```

Listing 5.6: Committing a geometry to a scene.

With these foundations in place, we now turn to the two primary mechanisms for adding geometries to a scene: first, conventional mesh-based geometries, and second, user geometries, which enable the integration of more complex analytical shapes.

5.3.2 Mesh Integration with STL

As previously discussed, the simplest class of geometries is given by triangular meshes, where each triangle is defined by three vertices forming its corners. The *STL* format natively supports this represen-

tation, and the ray-tracing module therefore provides functionality to import external models stored in STL files.

While triangular meshes are fast and convenient to use, they present a trade-off: increasing accuracy requires a higher number of triangles, which in turn increases file size. For non-reflecting parts of the telescope, however, such accuracy is unnecessary, since these components do not contribute to photon reflection and merely serve as blocking structures.

For reading STL files, we rely on the STL-reader header by Reiter [27], which supports both ASCII and binary STL formats. The returned data structure provides access to every triangle, which can then be added to the Embree geometry. To store vertices and their corresponding indices, Embree provides the `rtcSetNewGeometryBuffer` function. Here, the type of buffer must be defined: for vertices, we use `RTC_BUFFER_TYPE_VERTEX` with `float` as the data type, and for indices we use `RTC_BUFFER_TYPE_INDEX`. Since each triangle consists of three vertices, the size is set to `3*sizeof(float)` for the vertex buffer. The initial setup is shown in Listing 5.7.

```
stl_reader::StlMesh <float, unsigned int> mesh (path);
RTCGeometry rtcMesh = rtcNewGeometry (device,
    RTC_GEOMETRY_TYPE_TRIANGLE);
unsigned* indices = (unsigned*) rtcSetNewGeometryBuffer(rtcMesh,
    RTC_BUFFER_TYPE_INDEX, 0, RTC_FORMAT_UINT3,
     3*sizeof(unsigned), mesh.num_tris());
float* vertices = (float*) rtcSetNewGeometryBuffer(rtcMesh,
    RTC_BUFFER_TYPE_VERTEX, 0, RTC_FORMAT_FLOAT3,
     3*sizeof(float), mesh.num_tris()*3);
```

Listing 5.7: Initial setup for writing meshes into the geometry `rtcMesh`.

Adding triangles to the buffer is illustrated in Listing 5.8. The outer loop iterates over all triangles of the mesh, while the inner loop accesses each vertex using `tri_corner_coords`. If translation of the mesh is required, a translation vector can be applied directly to each vertex before insertion. After adding the vertices, the corresponding index values are updated. Once all triangles are processed, the procedure described in Listing 5.6 is used to commit the geometry to the scene.

```
int counter_indices = 0;
int counter_corner = 0;

for(size_t itri = 0; itri < mesh.num_tris(); ++itri) {
    for(size_t icorner = 0; icorner < 3; ++icorner) {
        const float* c = mesh.tri_corner_coords (itri, icorner);
        if (vertices && indices)
        {
            vertices[counter_corner] = c[0] + position.x;
            counter_corner++;
            vertices[counter_corner] = c[1] + position.y;
            counter_corner++;
            vertices[counter_corner] = c[2] + position.z;
            counter_corner++;
        }
    }
```

```
17      if (indices) {
18          indices[counter_indices] = counter_indices;
19          counter_indices++;
20          indices[counter_indices] = counter_indices;
21          counter_indices++;
22          indices[counter_indices] = counter_indices;
23          counter_indices++;
24      }
25  }
```

Listing 5.8: Algorithm for inserting all triangles into `rtcMesh`

This implementation allows the integration of various structural components of the telescope. Non-reflective parts such as the spider, which mechanically supports the mirror shells, are well suited for representation as meshes, since their accuracy in reflecting X-rays is irrelevant. Similarly, the sensor can be integrated as a mesh. Although in the simplest case of a flat plane this may be redundant, for more complex detector assemblies with multiple or tilted sensors, the mesh-based approach becomes advantageous.

By contrast, for the optical elements of Wolter telescopes, meshes are unsuitable. The parabolic and hyperbolic mirror surfaces must be described analytically to ensure accurate reflection of X-rays. As will be shown in the next section, Lobster Eye optics employ a hybrid approach, combining analytical descriptions with mesh representations.

5.3.3 User Geometry

In order to implement a user-defined geometry representing an arbitrary surface, several Embree-specific extensions are required. Listing 5.9 shows the setup procedure. As before, a geometry object of type `RTCGeometry` is created, but in this case the geometry type is set to `RTC_GEOMETRY_TYPE_USER`. In the given example, we construct a hyperboloid, which is represented by a dedicated parameter structure containing all relevant surface descriptors, as well as auxiliary information such as bounding box parameters. This parameter structure is attached to the geometry by calling `rtcSetGeometryUserData()`.

In addition, three function pointers must be specified: the bounding-box function, the intersection function, and the occlusion function. For our application the occlusion test is not relevant, and thus only an empty function head is required. The bounding-box and intersection functions, however, are central and will be discussed in more detail below.

Finally, the definition of the user geometry proceeds as with any other geometry type: the geometry is committed, attached to the scene, and its geometric ID is stored before the object is released.

```
1  RTCGeometry geometry = rtcNewGeometry(device, RTC_GEOMETRY_TYPE_USER);
2  auto* para = &hyperboloid.hyperboloid_parameters;
3
4  rtcSetGeometryUserPrimitiveCount(geometry, 1);
5  rtcSetGeometryUserData(geometry, para);
6  para->geometry = geometry;
7
8  rtcSetGeometryBoundsFunction(geometry,
9      Hyperboloid::hyperboloidBoundsFunc, nullptr);
```

```
rtcSetGeometryIntersectFunction(geometry,
    Hyperboloid::hyperboloidIntersectFunc);
rtcSetGeometryOccludedFunction(geometry,
    Hyperboloid::hyperboloidOccludedFunc);

// Commit the geometry and attach it to the scene.
rtcCommitGeometry(geometry);
para->geomID = rtcAttachGeometry(scene, geometry);
hyperboloid.geomID = para->geomID;
rtcReleaseGeometry(geometry);
```

Listing 5.9: Steps to insert a user-defined geometry.

The first step when defining a user geometry is the bounding-box specification. Embree uses this to build its internal bounding volume hierarchy (BVH), ensuring that intersection tests are only carried out in the smallest possible spatial region. An example implementation for the hyperboloid is given in Listing 5.10. After retrieving the parameter structure from the geometry via typecasting, the bounding box is constructed by specifying the lower and upper coordinates in the global 3D frame. To account for floating-point inaccuracies near the boundaries, a small symmetric padding is applied, thereby ensuring that the entire mirror is covered.

```
static void Hyperboloid::hyperboloidBoundsFunc(
    const RTCBoundsFunctionArguments *args) {

    const auto* para =
        (const Hyperboloid_parameters*) args->geometryUserPtr;
    RTCBounds* b = args->bounds_o;

    const float pad = 0.5f; // mm safety margin

    // axial (Z)
    b->lower_z = (float)para->Xh_min - pad;
    b->upper_z = (float)para->Xh_max + pad;

    // radial circle (X,Y), symmetric padding
    float rmin = (float)para->Yh_min;
    float rmax = (float)para->Yh_max;

    b->lower_x = -rmin - pad;
    b->upper_x =  rmax + pad;
    b->lower_y = -rmin - pad;
    b->upper_y =  rmax + pad;
}
```

Listing 5.10: Definition of a bounding box for a user geometry.

This example illustrates how Embree's user geometry interface allows analytical surfaces such as paraboloids and hyperboloids to be integrated into the ray-tracing framework. Unlike mesh approximations, these analytical definitions provide smooth surfaces without discretization artifacts, ensuring

accuracy in X-ray reflection.

In summary, Embree proves to be a highly suitable library for efficient intersection handling. With the capability to define both mesh-based and analytical user geometries, it offers the flexibility required for simulating complex optical systems. In the next section, we turn to the ray-tracing routine itself, where Embree is again leveraged for multiple core steps.

5.4 Implementation of the Ray Tracing Routine

This section details the ray-tracing routine, which was conceptually introduced in Chapter 4. We proceed in four stages. First, we describe how a telescope's mirror module is specified in a human-readable XML format. Second, we outline the allocation process, in which the XML parameters are parsed and the corresponding mirror module is instantiated. Third, we discuss the execution of the ray-tracing routine itself. Finally, we briefly examine the output interface when SIXTE completes this step in its simulation pipeline.

5.4.1 Scene Specification

Before a mirror module can be instantiated, its parameters must be defined. For this purpose, SIXTE uses a human-readable XML format derived from Table 4.1. This approach enables users and developers to conveniently modify telescope configurations without recompilation.

Listing 5.11 shows an example for a Wolter telescope. The attribute `type="raytracing"` specifies that the ray tracer is to be used for imaging. Within this block, the user defines the mirror module type, key geometric parameters, and—optionally—an exact placement for each mirror shell. If `exact="false"`, a uniform spacing of mirror shells is assumed, whereas `exact="true"` allows explicit specification of shell positions. The subsequent `surface` block describes the surface roughness model (e.g., `microfacet` or `gaus`) and its parameters, while the `blocking_part` block can optionally include mesh files for non-reflective components. Finally, the `sensor` block specifies the position and dimensions of the focal-plane detector.

```
<imaging type="raytracing">
  <type type="wolter" focal_length="1600" outer_diameter="358"
        inner_diameter="76" mirror_shells="54" mirror_height="150"/>
  <mirror exact="true">174.2,169.19,...,38.47,37.24</mirror>
  <surface model="microfacet" type="ggx" shadowing="ggx"
   roughness="0.0012"
           shadowing_alpha="0.0012" material="AU"
           material_path="/path/to/AtomicScatteringFactors_new.fits"/>
  <blocking_part flag="true" path="/path/to/baffle_spider.stl"
                 position_x="0" position_y="0" position_z="0"/>
  <sensor offset="-0.4" sensor_x="280" sensor_y="280" sensor_z="0"/>
</imaging>
```

Listing 5.11: XML format for specifying a Wolter mirror module.

For Lobster Eye optics, the specification differs. Since these systems use micro-channel plates (MPOs) instead of shells, the MPOs are imported directly as mesh geometries (Listing 5.12).

```
<optical path="/path/to/MPO.stl" position_x="0" position_y="0"
   position_z="0"/>
```

Listing 5.12: Specification of a Lobster Eye optic.

Furthermore, the sensor description may involve multiple plates aligned on a spherical surface, which can also be represented as meshes (Listing 5.13).

```
<sensor mesh="true" path="/path/to/sensor.stl" offset="0"
        sensor_x ="0" sensor_y="0" sensor_z="0"/>
```

Listing 5.13: Sensor configuration for a Lobster Eye optic.

In these examples, we assume that meshes are already aligned correctly in the STL files. If this is not the case, translation vectors can be defined directly in the XML to reposition the geometry appropriately.

5.4.2 Scene Allocation

The next step is the allocation of the mirror module. After parsing the XML file, the code initializes the corresponding objects. The procedure differs slightly for Wolter and Lobster Eye optics.

For Wolter telescopes, the allocation step first calculates the reference position for the outermost shell and stores it as the alignment height. Inner shells are then shifted accordingly, as shown in Listing 5.14.

```
if (first_shell) {
    p_pars.origin = Vec3fa(0,0, 0);
    first_shell_height = p_pars.Xp_max;
    first_shell=false;
} else {
    auto z_offset = (float) (first_shell_height - p_pars.Xp_max);
    p_pars.origin = Vec3fa(0, 0, z_offset);
    h_pars.origin = Vec3fa(0, 0, z_offset);
}
```

Listing 5.14: Translation vector initialization for all inner shells.

Surface roughness models are also allocated at this stage. While this may seem redundant during development, it provides a placeholder for future extensions where shell-specific roughness parameters may be available. The allocation leverages the strategy pattern, thereby facilitating the integration of additional surface models (Listing 5.15).

```
p_pars.surface = std::make_unique<SurfaceModel>(
                    std::make_unique<Model>(/* parameter list*/));
```

Listing 5.15: Allocation of the surface roughness model.

Finally, all paraboloid and hyperboloid mirrors are added to the Embree scene (as described in Section 4.2), resulting in a fully allocated mirror module. This module exposes the interface `ray_trace(Ray &ray)`, which is now ready to trace incoming photons.

5.4.3 Execution

The execution phase implements the ray-tracing routine itself. We illustrate this first for Wolter telescopes, then highlight differences for Lobster Eye optics.

Listing 5.16 shows the intersection-finding loop. For each ray, the routine determines the closest intersection using `rtcIntersect1()`. If no valid intersection is found, the ray is terminated.

```
bool EmbreeScene::embree_ray_trace(Ray &ray, int depth) {
  while (depth > 0) {
    // Intersect
    rtcIntersect1(scene, &ray.rayhit);

    if (ray.rayhit.hit.geomID == RTC_INVALID_GEOMETRY_ID)
        return false;
  // A and B
  }
  return false;
}
```

Listing 5.16: Intersection finding in the scene.

Next, the algorithm checks whether the intersection occurred on the sensor or the spider (Listing 5.17). If the ray hits the sensor, its impact position is logged. If it hits the spider, the photon is discarded.

```
// A:
    // Check if sensor was hit
    if (sensor.isOnSensor(ray.rayhit)) {
        ray.set_position(ray.position()
            + ray.rayhit.ray.tfar * ray.direction());
        return true;
    }
    // Check if spider was hit
    if (ray.rayhit.hit.geomID == spider.geomID) {
        return false;
    }
```

Listing 5.17: Testing for sensor or spider hits.

If the ray intersects a reflective surface, the reflection routine is invoked (Listing 5.18). The order of operations is critical: (i) apply surface roughness models, (ii) determine reflectivity stochastically based on energy and incidence angle, and (iii) compute the reflected ray. If the ray survives all stages, the loop continues.

```
// B:
    // Add roughness if there is any
    surfaceModel = find_surface_model(ray.rayhit.hit.geomID);
    if (surfaceModel != nullptr)
        if(!surfaceModel->simulate_surface(ray))
            return false;
```

```
7
8       // Reflectivity
9       if (!mirror_coating.doesReflect(ray.energy,
10          get_angle(-1*ray.direction(), ray.normal())))
11          return false;
12
13      // Reflect ray
14      if(!reflect_ray(ray))
15          return false;
16
17      // Decrease depth
18      depth--;
19  }
```

Listing 5.18: Reflecting part of the ray tracing routine.

For Lobster Eye optics, the routine differs. Instead of storing all pores explicitly, only the MPO mesh is included in the scene. When the MPO is hit, the ray is transformed into the local coordinate system of a single pore, where internal reflections are simulated (Listing 5.19).

```
1  if(ray.rayhit.hit.geomID == opticalMesh.geomID) {
2      ray.set_position(ray.position()
3          + ray.rayhit.ray.tfar * ray.direction());
4      if(!pore.ray_trace(ray, depth)){
5          return false;
6      }
```

Listing 5.19: Reflection handling for the Lobster Eye optic.

Surface roughness effects are implemented using the microfacet theory. Listing 5.20 shows a simplified version, where normals are perturbed according to the chosen distribution (Beckmann or GGX). The probability terms for masking and shadowing are computed, and the ray's survival is decided stochastically.

```
1      Vec3fa m;
2      Vec3fa outcoming;
3      double prob_shadowing, prob_masking;
4      if (ggx) {
5          m = get_gxx_m();
6          prob_masking = ggx_shadowing_term(incoming, m);
7          outcoming = reflect(incoming, m);
8      } else {
9          m = get_beckmann_m();
10         prob_masking = beckmann_shadowing_term(incoming, m);
11         outcoming = reflect(transformed_incoming, m);
12     }
13     if (ggx_shadowing) {
14         prob_shadowing = ggx_shadowing_term(outcoming, m);
15     } else {
16         prob_shadowing = beckmann_shadowing_term(outcoming, m);
```

```
17      }
18      if (sixte::getUniformRandomNumber()
19          > prob_shadowing * prob_masking) {
20          return false;
21      }
22      ray.set_normal(m);
23      return true;
```

Listing 5.20: Microfacet theory in the ray tracing routine.

It is worth noting that the routine requires frequent random number draws: for initial photon positions, for reflectivity tests, and for microfacet sampling. While this ensures physical realism, it is computationally expensive. Potential optimizations could involve precomputing large randomized arrays for microfacet normals, though uniform sampling for photon positions and angle-dependent reflectivity still requires on-the-fly generation.

In conclusion, this section has demonstrated how the ray-tracing routine for X-ray telescopes is realized within the SIXTE framework. With Embree as a high-performance intersection engine, ray tracing is now efficiently integrated into the simulation pipeline. Nevertheless, the current implementation is limited to single-core CPU execution. Future developments could significantly accelerate the routine by porting computationally intensive parts to GPUs. In the next chapter, we evaluate the performance and scientific validity of this implementation.

Chapter 6

Results

This chapter presents the results of the ray tracing library. We will start by investigating to which extent the defined software requirements are fulfilled and then go over to the simulation results. First, we start with the Wolter 1 geometry and present the PSF from on axis and off axis point sources. There we will compare them with real data to analyze and verify our test results. Then we go over to to the surface roughness evaluation and present the performance of the microfacet model. After that, we present the integration into SIXTE's simulation process and evaluate it to the original PSF imaging. Finally, we conduct performance measurements which we can compare to older measurements of the ray tracing library.

Lobster Eye optics follows a similar pattern with the exception that we don't have old performance data. But still, we will conduct comparisons between Wolter and the Lobster Eye optics performance.

6.1 Software Requirement Evaluation

Before we evaluate the simulation output, we review the software requirement list. We will start with the functional and then the non-functional requirement.

6.1.1 Functional Requirement Review

1. **Ray tracing through telescope optics:** The current implementation is designed with a ray tracing algorithm which is agnostic towards its input and rays travel through the scene until termination through intersecting with the sensor or expiration.

2. **Support for different optical designs:** The system provides functionality to create different telescope optics. The minimum, Wolter I optics and Lobster-Eye optics, is achieved with the modularity to implement every kind of new optic in the future.

3. **Configurable optical geometries:** Regardless of Wolter I or Lobster Eye optics, a free configuration of those optics is possible. Telescope properties like the amount of shells, position of the sensor plane, the sensor itself, the focal length and so on are freely adjustable.

4. **Inclusion of non-mirror components:** With the import of meshes into the scene, this requirement is achieved.

5. **Realistic surface roughness model:** With the implementation of the microfacet surface distribution theory, a potent and recognized surface roughness model is implemented into the simulation system.

6.1.2 Non-Functional Requirements

1. **Reliability** Test cases ensure the reliable propagation of rays in a scene when excluding the non-deterministic factors like the random number drawing. Those non-deterministic factors like reflectivity were tested against other data and verified. Also when choosing the same seed for two simulation runs, the output will be the same.

   Furthermore, the variation change will lead to the expected change of the output. This will be discussed later in this chapter.

2. **Flexibility** This system proves to be very flexible which can be determined by the several factors. For the user, building a Wolter I or a Lobster Eye telescope happens in the same process as it was with the proven PSF imaging. Only keywords in the XML file need to be changed and the properties of the mirror module can be changed arbitrary. Adding meshes is done by just altering the path to the right mesh, one want to import it to.

   For developers, the code base is highly modular. Through the introduction of design patterns, it facilitates the later modification. Adding different surface roughness model just needs small modification due to the introduction of the strategy pattern.

   Through object oriented programming, design a novel mirror module geometry is no problem by just creating a new class of that new design and it won't affect the current implementation. Every shape has its interface for ray tracing, adding a new shape can be easily done and furthermore, adding meshes as optical or non optical parts is supported.

3. **Efficiency** We go into more detail for this later in this chapter. First simulation performance measurements suggest that even very large mesh files will keep the allocation time of the mirror module below five seconds.

   Also is the simulation time of the whole SIXTE pipeline in the same magnitude as it is with the PSF imaging setup.

4. **Usability** The only external library used for the ray tracing module is Embree. Including Embree into the installation process of CMake can be easily done and is in the same effort to be done as it is with adding CGAL into the installation process if it is not on the standard path for CMake.

5. **Operability** Documentation of this ray tracing module is provided with this thesis but also the interfaces are documented and from the photon creation step to the photon imaging step, the ray tracing modules integrates well into SIXTE and its execution is seamless for the user.

6.2 Wolter I

For the Wolter I evaluation, we take the properties of the eRosita mirror module [26]. Table 6.1 lists the elements of the mirror specification. We will first look on the the PSF on axis and off axis as well to when

Focal Length	1600 *mm*
Outer Diameter	358 *mm*
Inner Diameter	76 *mm*
Amount of Shells	54
Mirror Height	150 *mm*
Sensor Offset	0.4 *mm*
Mirror Coating Material	AU

Table 6.1: Properties of the eRosita mirror module.

introduce the surface roughness and compare it with measurements. Then we evaluate the integration into SIXTE and use SIXTE's tool to verify the simulation. In the end, a performance measurement will compare the current implementation with the rudiment prototype that we implemented before of the thesis.

6.2.1 PSF On Axis

First, we start to simulate on axis PSFs. On axis means that the light, that is reaching the mirror module is perpendicular to the focal plane. Figure 6.1 plots different setups for on-axis PSFs. The setups differentiate from the mirror position in the z-axis to the sensor position in z. The upper row aligns the mirrors on the same height of the most outer shell. This produces that accumulation of photon impacts towards the outer rings of the PSF. Each ring is there a shell and the focus point diverts increasingly with the inner shells producing the outer rings on the PSF. The distance of the inner mirror shells of the module is also smaller which explains the density of the outer rings.

In the upper row, the left plot shows the result if the sensor is lifted for 11 *mm*. This produces are de-focused image which let the spider arms seen better. Those are producing the black spikes in the PSFs because they are essentially casting a shadow on that position. The plot in the middle shows the PSF when only being lifted 0.4 *mm* up in z from the focal plane. This is the true sensor distance in eRosita chosen for better PSF quality for off axis.

In the lower row, we see the same setup but with fulfilling Abbe's sine condition so that all shells are sharing the same focal plane. This produces those PSFs where the rings are having more equidistance. If we would put the sensor on the actual focal plane, all rings would collapse to one single point.

If we now compare both rows we see especially in the middle and right column size differences. This opens up the discussion whether eRosita mirror shells are really at the same height because the off-focus of the upper row is significant. Unfortunately, there are no documentation to the real mirror position when eRosita was launched. Through conversations with scientists of the MPE, we discovered that they aligned the final position of the shells at Panter, a X-ray test facility. Their alignment efforts are not documented and those efforts aimed to get a focused point. This also means, that we don't know how much the mirrors are rotated and therefore assume to have no rotation. Later, for the surface roughness evaluation, we will position all mirrors not only at the same height, but will also investigate the behavior if the mirrors are just half of the actual offset lowered towards the height of the outer shell.

This simulation here also does not include any surface roughness at the moment because this would just make analysis of the optic more difficult.

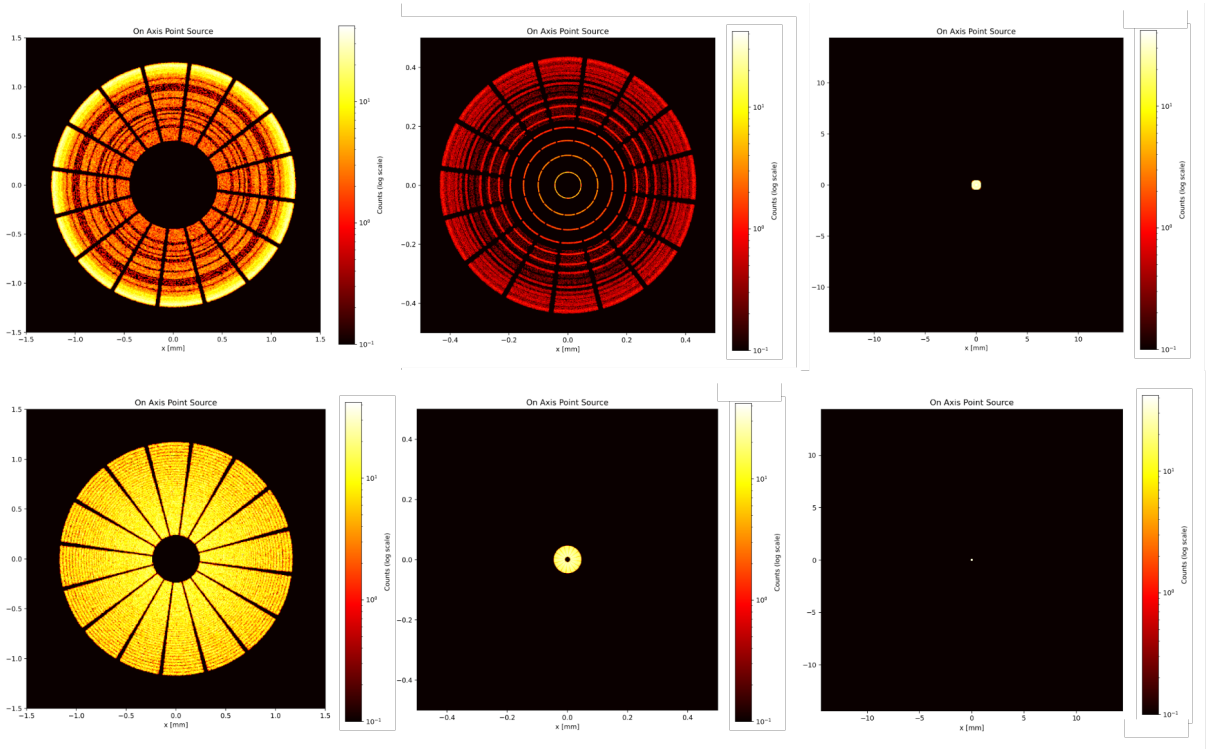

Figure 6.1: On Axis PSFs. Up: When the mirrors are at the same height aligned as they are on eRosita. Down: If Abbe sine condition would be fulfilled. Left is complete de-focus of 11 *mm*, middle is sensor position of eRosita and right is the real size on the chip.

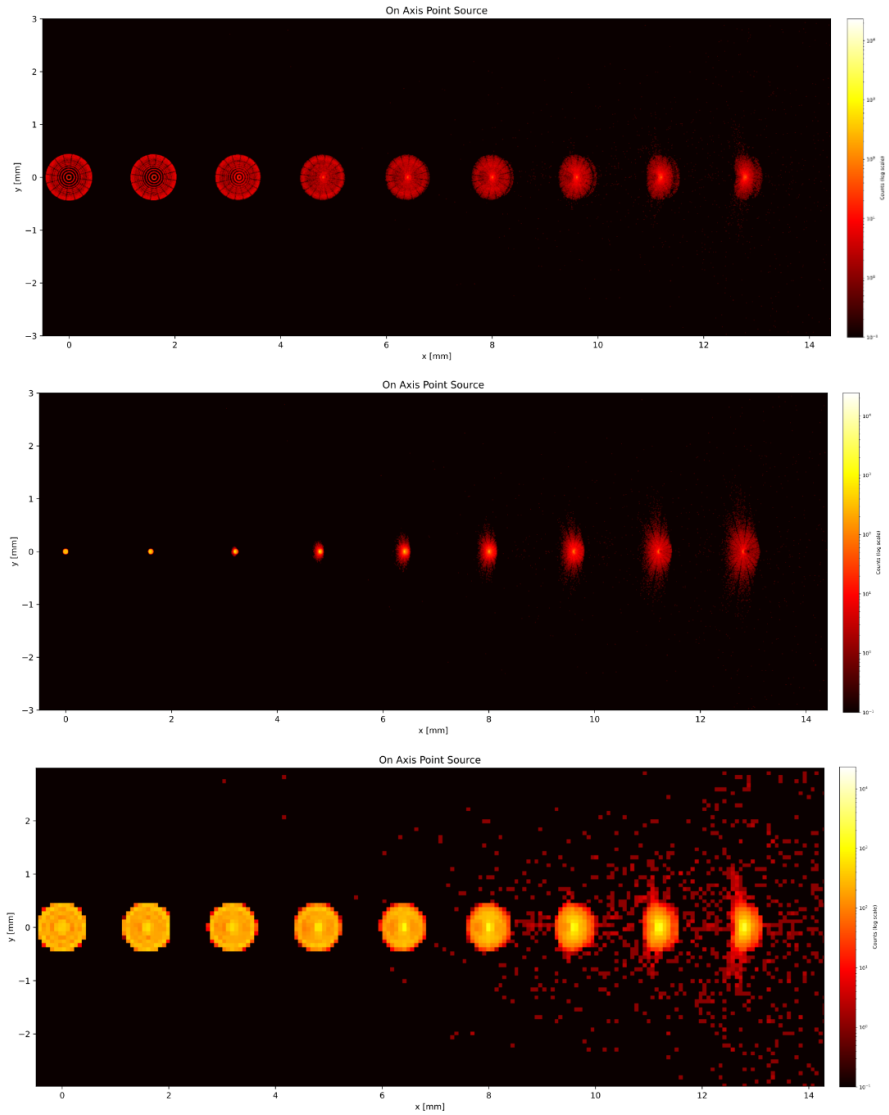

Figure 6.2: Enter Caption

6.2.2 PSF Off Axis - Butterfly analysis

While the on-axis analysis can hint the right path for verifying the simulation output, we have to look also at the off-axis PSFs. As Wolter already mentioned, while this geometry is able to focus X-ray photons, it suffers from heavy optical aberrations.

6.2.3 Roughness Parameter Evaluation

COnsideration whether the parameter evaluation and scoring is part of the methods part. COuld be part either or.

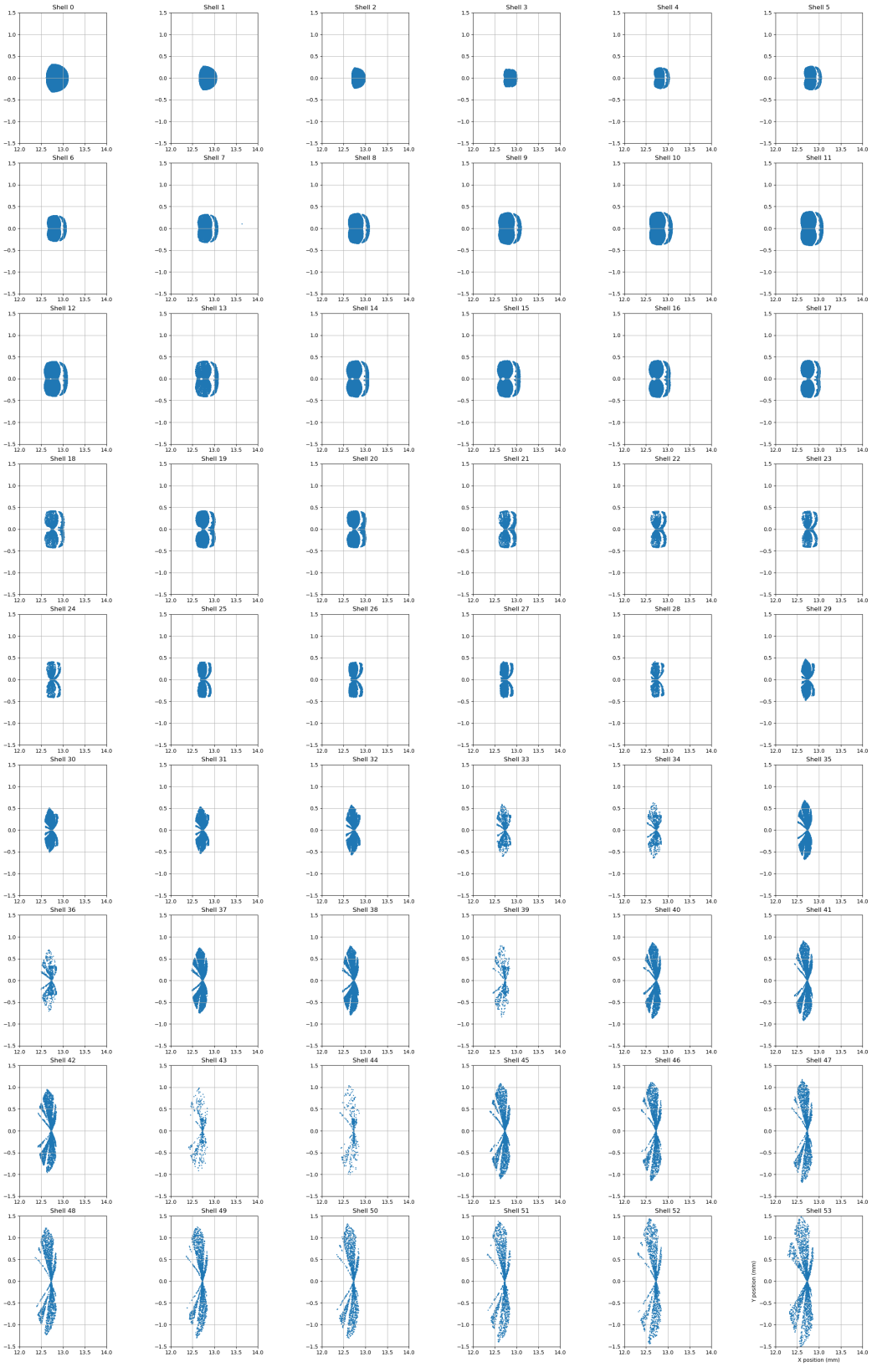

Figure 6.3: Enter Caption

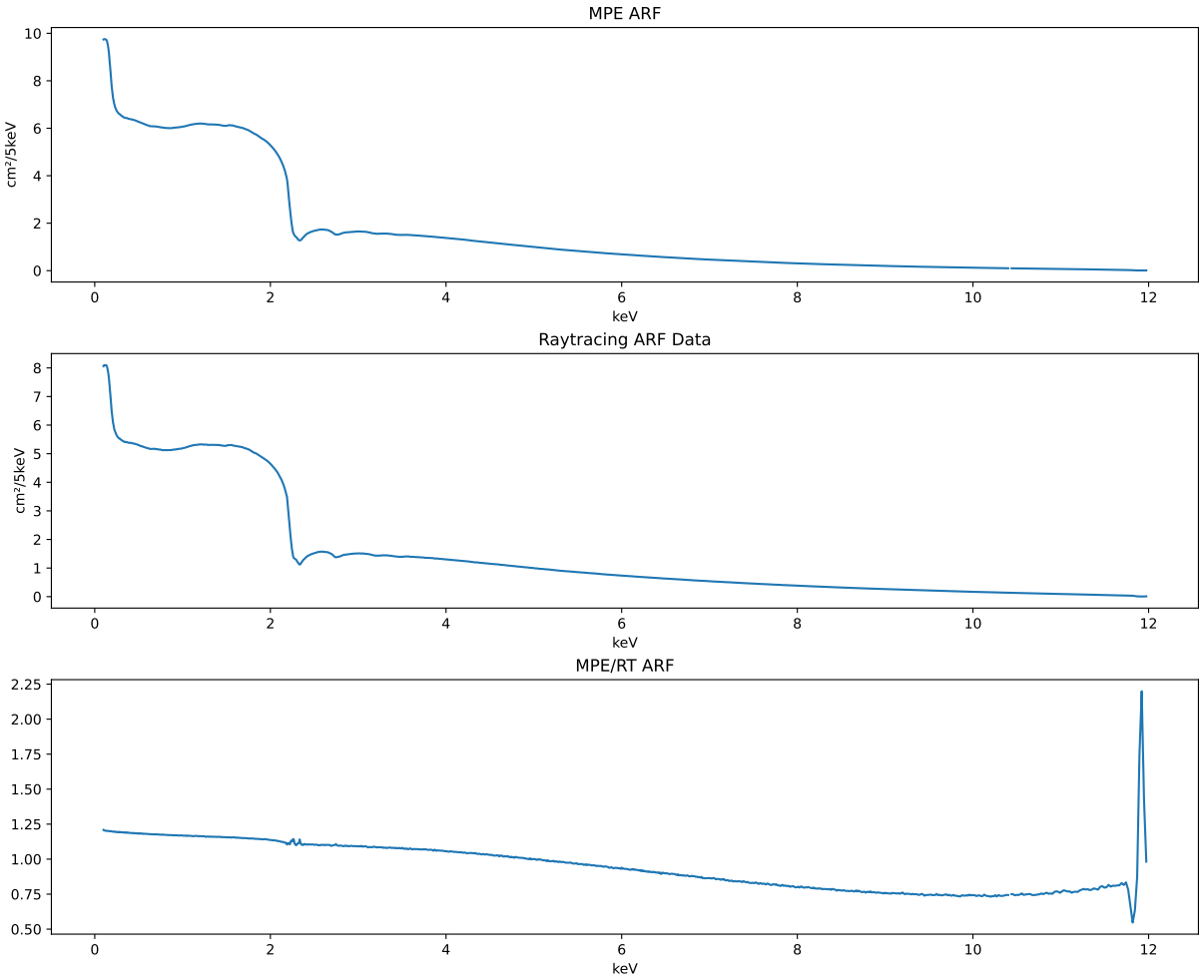

Figure 6.4: Enter Caption

6.2.4 SIXTE integration evaluation

6.2.5 Performance Measurements

6.3 Lobster Eye Optic

6.3.1 PSF Analysis On and Off Axis

6.3.2 Focal Plane Evaluation with tilted sensors

6.3.3 SIXTE Integration

6.3.4 Performance Measurements

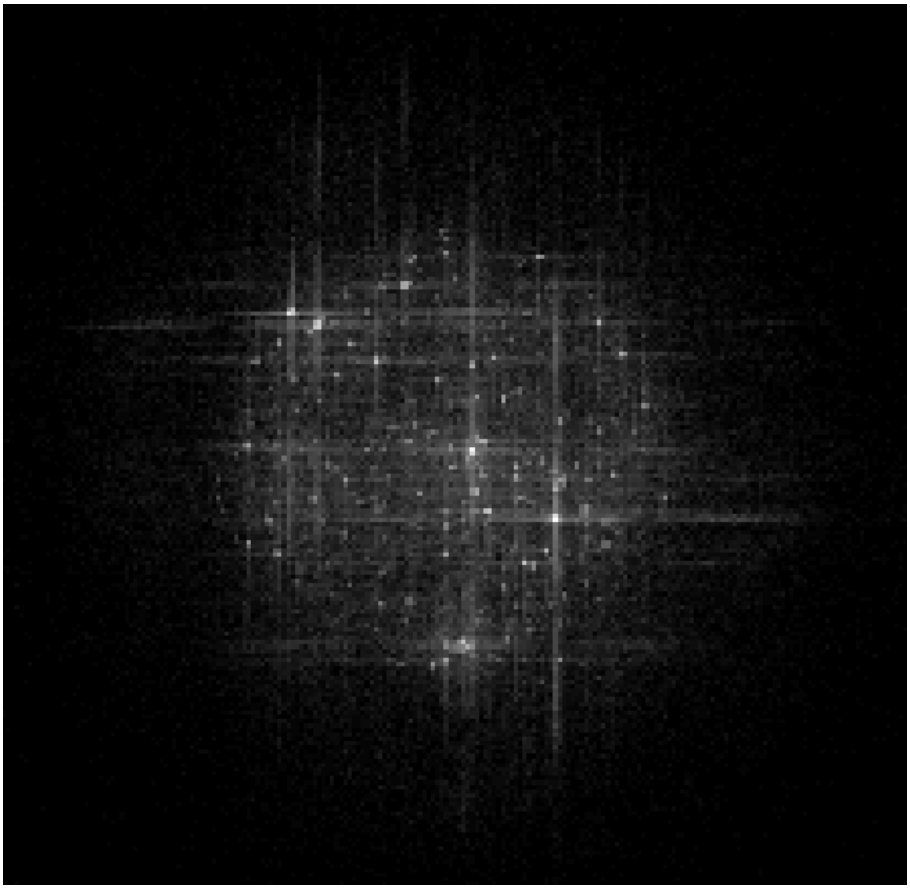

Figure 6.5: Enter Caption

Chapter 7

Conclusions

Give a brief summary of your work and your contributions. Afterwards, briefly outline potential opportunities for future work.

Appendix A

Wolter I detailed parameter description

write down all the formulae for getting the parameters. use [25]'s paper for that.

Appendix B

Quadratic solutions for Paraboloid and Hyperboloid

Step by step solution way is shown here.

List of Figures

List of Tables

List of Listings

Bibliography

[1] J. R. P. Angel. "Lobster Eyes As X-Ray Telescopes". In: *Space Optics Imaging X-Ray Optics Workshop*. Space Optics Imaging X-Ray Optics Workshop. Vol. 0184. SPIE, Aug. 9, 1979, pp. 84–85. DOI: 10.1117/12.957437. URL: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/0184/0000/Lobster-Eyes-As-X-Ray-Telescopes/10.1117/12.957437.full.

[2] Nicolas M. Barrière, Marcos Bavdaz, Maximilien J. Collon, Ivo Ferreira, David Girou, Boris Landgraf, and Giuseppe Vacanti. "Silicon Pore Optics". In: 2022, pp. 1–44. DOI: 10.1007/978-981-16-4544-0_5-1. arXiv: 2206.11291 [astro-ph]. URL: http://arxiv.org/abs/2206.11291.

[3] Petr Beckmann and Andre Spizzichino. "The scattering of electromagnetic waves from rough surfaces". In: *Norwood* (1987).

[4] Enrico Bozzo, Lorenzo Amati, Paul O'Brien, Diego Goetz, and Andrea Santangelo. "THESEUS: Transient High Energy Sky and Early Universe Surveyor". In: *Space Telescopes and Instrumentation 2024: Ultraviolet to Gamma Ray*. Space Telescopes and Instrumentation 2024: Ultraviolet to Gamma Ray. Ed. by Jan-Willem A. Den Herder, Kazuhiro Nakazawa, and Shouleh Nikzad. Yokohama, Japan: SPIE, Aug. 21, 2024, p. 86. DOI: 10.1117/12.3022804. URL: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/13093/3022804/THESEUS-Transient-High-Energy-Sky-and-Early-Universe-Surveyor/10.1117/12.3022804.full.

[5] Chinese Academy of Sciences. *Lobster-eye optics of the Einstein Probe: single, double, and no reflection paths*. ESA – Einstein Probe factsheet, Science & Exploration, Space Science. Image credit: ESA; URL: https://www.esa.int/Science_Exploration/Space_Science/Einstein_Probe_factsheet, accessed 2025-08-15. 2025.

[6] Robert L Cook and Kenneth E Torrance. "A Reflectance Model for Computer Graphics". In: *ACM Transactions on Graphics* 1.1 (1982).

[7] Thomas Dauser, Sebastian Falkner, Maximilian Lorenz, Christian Kirsch, Philippe Peille, Edoardo Cucchetti, Christian Schmid, Thorsten Brand, Mirjam Oertel, Randall Smith, et al. "SIXTE: a generic X-ray instrument simulation toolkit". In: *Astronomy & Astrophysics* 630 (2019), A66.

[8] European Space Agency (ESA). *Artist's impression of the NewAthena X-ray Observatory*. ESA - NewAthena factsheet, Science & Exploration, Space Science. Image credit: ESA; URL: https:

//www.esa.int/Science_Exploration/Space_Science/NewAthena_factsheet, accessed 2025-08-15. 2025.

[9] Angelo Ferretti. *Cover image for the fourth edition of *Physically Based Rendering: From Theory to Implementation**. Matt Pharr's blog post "Some News About the 4th Edition of Physically Based Rendering". Image licensed from Angelo Ferretti; URL: https://pharr.org/matt/blog/2022/12/22/pbr-4ed, accessed 2025-08-15. 2022.

[10] R. Giacconi, W. P. Reidy, G. S. Vaiana, L. P. Van Speybroeck, and T. F. Zehnpfennig. "Grazing-Incidence Telescopes for X-ray Astronomy". In: *Space Science Reviews* 9.1 (Feb. 1969), pp. 3–57. DOI: 10.1007/BF00187578. URL: http://link.springer.com/10.1007/BF00187578.

[11] Paul Gorenstein. "Grazing Incidence Telescopes for X-Ray Astronomy". In: *Optical Engineering* 51.1 (Feb. 2012), p. 011010. DOI: 10.1117/1.OE.51.1.011010. URL: https://www.spiedigitallibrary.org/journals/optical-engineering/volume-51/issue-1/011010/Grazing-incidence-telescopes-for-x-ray-astronomy/10.1117/1.OE.51.1.011010.full.

[12] Hans Moritz Günther, Jason Frost, and Adam Theriault-Shay. "MARXS: a modular software to ray-trace x-ray instrumentation". In: *The Astronomical Journal* 154.6 (2017), p. 243.

[13] *ISO/IEC 25010:2023 – Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. accessed 21 August 2025. International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), 2023. URL: https://www.iso.org/obp/ui#iso:std:iso-iec:25010:ed-2:v1:enl.

[14] Erik B. Knudsen, Desiree D. M. Ferreira, Niels J. Westergaard, Finn E. Christensen, Sonny Massahi, Brian Shortt, Ivo Ferreira, and Daniele Spiga. "The McXtrace AstroX toolbox: a general ray tracing software package for end to end simulation of x-ray optics for astronomical instrumentation". In: *Space Telescopes and Instrumentation 2018: Ultraviolet to Gamma Ray*. Ed. by Jan-Willem A. den Herder, Shouleh Nikzad, and Kazuhiro Nakazawa. Vol. 10699. International Society for Optics and Photonics. SPIE, 2018, 106993S. DOI: 10.1117/12.2312272. URL: https://doi.org/10.1117/12.2312272.

[15] Krishnavedala. *Image from Wikimedia Commons*. https://commons.wikimedia.org/w/index.php?curid=34890843. Own work, CC BY-SA 4.0. 2015.

[16] Lockheed Martin Missiles & Space. "Hubble Space Telescope Systems". In: *Hubble Space Telescope: Servicing Mission 3A — Media Reference Guide*. Report No. K9322-05M. Greenbelt, MD: NASA Goddard Space Flight Center, 1999, pp. 5-1–5-35. URL: https://asd.gsfc.nasa.gov/archive/sm3a/downloads/sm3a_media_guide/HST-systems.pdf.

[17] Kristin Madsen, D. Alexander, V. Bhalerao, S. Boggs, B. Greffenstette, D. I. Harp, F. Harrison, T. Kitagutchi, H. Miyasaka, A. Ptak, D. Stern, J. Tomsick, V. Rana, and A. Zoglauer. "Simulating Galactic and Extragalactic Surveys with the NuSTAR Simulator NuSIM." In: *AAS/High Energy Astrophysics Division #12*. Vol. 12. AAS/High Energy Astrophysics Division. Sept. 2011, 43.08, p. 43.08.

[18] Kristin K Madsen, David Broadway, and Desiree Della Monica Ferreira. "Single-layer and multilayer coatings for astronomical X-ray mirrors". In: *Handbook of X-ray and Gamma-ray Astrophysics*. Springer, 2022, pp. 1–39.

[19] Matt Pharr, Wenzel Jakob, and Greg Humphreys (eds.) *8.4 Microfacet Models*. Physically Based Rendering, 3rd ed. — "Microfacet Models" page. Image credit: Physically Based Rendering authors; URL: https://www.pbr-book.org/3ed-2018/Reflection_Models/Microfacet_Models, accessed 2025-08-19. 2018.

[20] Peter Friedrich / MPE. *eROSITA X-ray telescope mirror modules (clean-room integration)*. Max-Planck-Gesellschaft website. Image credit: Peter Friedrich/MPE; URL: https://www.mpg.de/13559111/erosita, accessed 2025-08-15. 2025.

[21] NASA/CXC & J. Vaughan. *Illustration of the Chandra X-ray Observatory*. https://www.nasa.gov/chandra-overview. Accessed: 2025-08-15. 2024.

[22] NASA/CXC/SAO. *Montage of 25 new cosmic X-ray images from Chandra's 25th anniversary*. NASA News Release. Image credit: NASA/CXC/SAO; URL: https://www.nasa.gov/image-detail/to-celebrate-the-25th-anniversary-of-its-launch-nasas-chandra-x-ray-observatory-is-releasing-25-never-before-seen-views-of-a-wide-range-of-cosmic-objects-3/, accessed 2025-08-15. 2024.

[23] Matt Pharr, Wenzel Jakob, and Greg Humphreys. "Physically Based Rendering: From Theory to Implementation". In: 4th ed. MIT Press, 2023. Chap. 1, pp. 1–51. ISBN: 978-0262048026.

[24] Matt Pharr, Wenzel Jakob, and Greg Humphreys. "Physically Based Rendering: From Theory to Implementation". In: 4th ed. MIT Press, 2023. Chap. 9, pp. 535–631. ISBN: 978-0262048026.

[25] Michael J Pivovaroff and Takashi Okajima. "Geometries for Grazing Incidence Mirrors". In: *Handbook of X-ray and Gamma-ray Astrophysics*. Springer, 2023, pp. 1–21.

[26] Peter Predehl. "eROSITA". In: *Space Telescopes and Instrumentation 2012: Ultraviolet to Gamma Ray*. Space Telescopes and Instrumentation 2012: Ultraviolet to Gamma Ray. Vol. 8443. SPIE, Sept. 17, 2012, pp. 498–507. DOI: 10.1117/12.925843. URL: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/8443/84431R/eROSITA/10.1117/12.925843.full.

[27] Sebastian Reiter. *stl_reader: One-header C++ library to load ASCII and binary STL geometry files*. https://github.com/sreiter/stl_reader/tree/abcdef1234567890abcdef1234567890abcdef12. BSD-2-Clause license, snapshot at commit abcdef1234567890abcdef1234567890abcdef12. 2025.

[28] Jeremy Sanders, Hermann Brunner, the eSASS team (MPE), Eugene Churazov, and Marat Gilfanov (on behalf of IKI). *All-sky X-ray survey image from eROSITA (first data release)*. Max-Planck Institute for Extraterrestrial Physics (MPE) press kit. Image credit: Jeremy Sanders, Hermann Brunner and the eSASS team (MPE); Eugene Churazov, Marat Gilfanov (IKI); URL: https://www.mpe.mpg.de/7463647/erass1-presskit, accessed 2025-08-15. 2020.

[29] Christophe Schlick. "An inexpensive BRDF model for physically-based rendering". In: *Computer graphics forum*. Vol. 13. 3. Wiley Online Library. 1994, pp. 233–246.

[30] B. Smith. "Geometrical Shadowing of a Random Rough Surface". In: *IEEE Transactions on Antennas and Propagation* 15.5 (Sept. 1967), pp. 668–671. DOI: 10.1109/TAP.1967.1138991. URL: https://ieeexplore.ieee.org/document/1138991/.

[31] Daniele Spiga. "X-Ray Mirror Module Analytical Design from Field of View Requirement and Stray Light Tolerances". In: *Space Telescopes and Instrumentation 2016: Ultraviolet to Gamma Ray*. Space Telescopes and Instrumentation 2016: Ultraviolet to Gamma Ray. Vol. 9905. SPIE, July 18, 2016, pp. 1951–1961. DOI: 10.1117/12.2222907. URL: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/9905/99056R/X-ray-mirror-module-analytical-design-from-field-of-view/10.1117/12.2222907.full.

[32] Gernot Starke et al. *The arc42 Quality Model (Q42)*. Accessed 2025-08-21. 2025. URL: https://quality.arc42.org/.

[33] Neeraj K. Tiwari, Santosh V. Vadawale, N. P. S. Mithun, C. S. Vaishnava, and Bharath Saiguhan. "DarsakX: A Python Package for Designing and Analyzing Imaging Performance of X-ray Telescopes". In: *Astronomy and Computing* 47 (Apr. 2024). arXiv:2405.06343 [astro-ph], p. 100829. ISSN: 22131337. DOI: 10.1016/j.ascom.2024.100829. URL: http://arxiv.org/abs/2405.06343 (visited on 06/12/2024).

[34] K. E. Torrance and E. M. Sparrow. "Theory for Off-Specular Reflection From Roughened Surfaces*". In: *J. Opt. Soc. Am.* 57.9 (1967), pp. 1105–1114. DOI: 10.1364/JOSA.57.001105. URL: https://opg.optica.org/abstract.cfm?URI=josa-57-9-1105.

[35] Ulflund. *Image on Wikimedia Commons*. https://commons.wikimedia.org/w/index.php?curid=22541840. Own work, CC0; Accessed: 2024-08-15.

[36] LP VanSpeybroeck and RC Chase. "Design parameters of paraboloid-hyperboloid telescopes for X-ray astronomy". In: *Applied Optics* 11.2 (1972), pp. 440–445.

[37] Ingo Wald, Sven Woop, Carsten Benthin, Will Usher, and Manfred Ernst. *Embree: A High-Performance Ray Tracing Library*. https://www.embree.org/. Version 4.4; developed at Intel, Apache 2.0 license. Intel, 2025.

[38] Bruce Walter, Stephen R Marschner, Hongsong Li, and Kenneth E Torrance. "Microfacet Models for Refraction through Rough Surfaces". In: *Rendering techniques* 2007 (2007), 18th.

[39] Philipp Weber and Joern Wilms. "eROSITA's View on 1H 0707-495 and the Near-Real-Time Analysis Pipeline". MA thesis. Friedrich-Alexander-Universität Erlangen-Nürnberg, 2021.

[40] Martin C. Weisskopf. "Chandra X-Ray Optics". In: *Optical Engineering* 51.1 (Feb. 2012), p. 011013. DOI: 10.1117/1.OE.51.1.011013. URL: https://www.spiedigitallibrary.org/journals/optical-engineering/volume-51/issue-1/011013/Chandra-x-ray-optics/10.1117/1.OE.51.1.011013.full.

[41] Martin C. Weisskopf, Harvey D. Tananbaum, Leon P. Van Speybroeck, and Stephen L. O'Dell. "Chandra X-ray Observatory (CXO): Overview". In: *X-Ray Optics, Instruments, and Missions III*. X-Ray Optics, Instruments, and Missions III. Vol. 4012. SPIE, July 18, 2000, pp. 2–16. DOI: 10.1117/12.391545. URL: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/4012/0000/Chandra-X-ray-Observatory-CXO-overview/10.1117/12.391545.full.

[42] MC Weisskopf, B Brinkman, C Canizares, G Garmire, S Murray, and LP Van Speybroeck. "An Overview of the Performance and Scientific Results from the Chandra X-Ray Observatory". In: *Publications of the Astronomical Society of the Pacific* 114.791 (2002), p. 1.

[43] Niels J. Westergaard, Desiree D. M. Ferreira, and Sonny Massahi. "On X-ray Telescopes in General and the Athena Optics in Particular". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*. Imaging 2016 873 (Nov. 21, 2017), pp. 5–11. DOI: 10.1016/j.nima.2017.01.058. URL: https://www.sciencedirect.com/science/article/pii/S016890021730147X.

[44] Hans Wolter. "Spiegelsysteme streifenden Einfalls als abbildende Optiken für Röntgenstrahlen". In: *Annalen der Physik* 445.1-2 (1952), pp. 94–114.

[45] Hans Wolter. "Verallgemeinerte Schwarzschildsche Spiegelsysteme streifender Reflexion als Optiken für Röntgenstrahlen". In: *Annalen der Physik* 445.4-5 (1952), pp. 286–295. DOI: 10.1002/andp.19524450410. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/andp.19524450410.

[46] Donghua Zhao, Chen Zhang, Weimin Yuan, Shuangnan Zhang, Richard Willingale, and Zhixing Ling. "Geant4 simulations of a wide-angle x-ray focusing telescope". In: *Experimental Astronomy* 43 (2017), pp. 267–283.

Declaration

I certify that I have written this thesis without outside help and without using sources other than those indicated, and that the thesis has not been submitted in the same or similar form to any other examination authority and has not been accepted by them as part of an examination. All statements that have been taken over verbatim or in spirit are marked as such.

During the preparation of this work, AI technologies were used to assist in the writing process. Specifically, <AI Tool 1 (Company, City, State, Country)> was used in order to check for grammar and style consistency, and <AI Tool 2 (Company, City, State, Country)> was used in order to assist with rephrasing and improving readability. After using these tools, the manuscript was carefully reviewed and the content was edited as needed. No tools or services were used for content generation. The content of this thesis was entirely created by myself, and I ensured that all material presented reflects my original work and research. I take full responsibility for the content of the thesis.

I agree that the thesis may be published and that it may be referred to in scientific publications.

The Friedrich-Alexander-Universität Erlangen-Nürnberg, represented by the Professorship for Visual Computing, is granted a (non-exclusive) right to use this work and the programs created in connection with it.

Erlangen, 15th of September 2025

(Neo Reinmann)