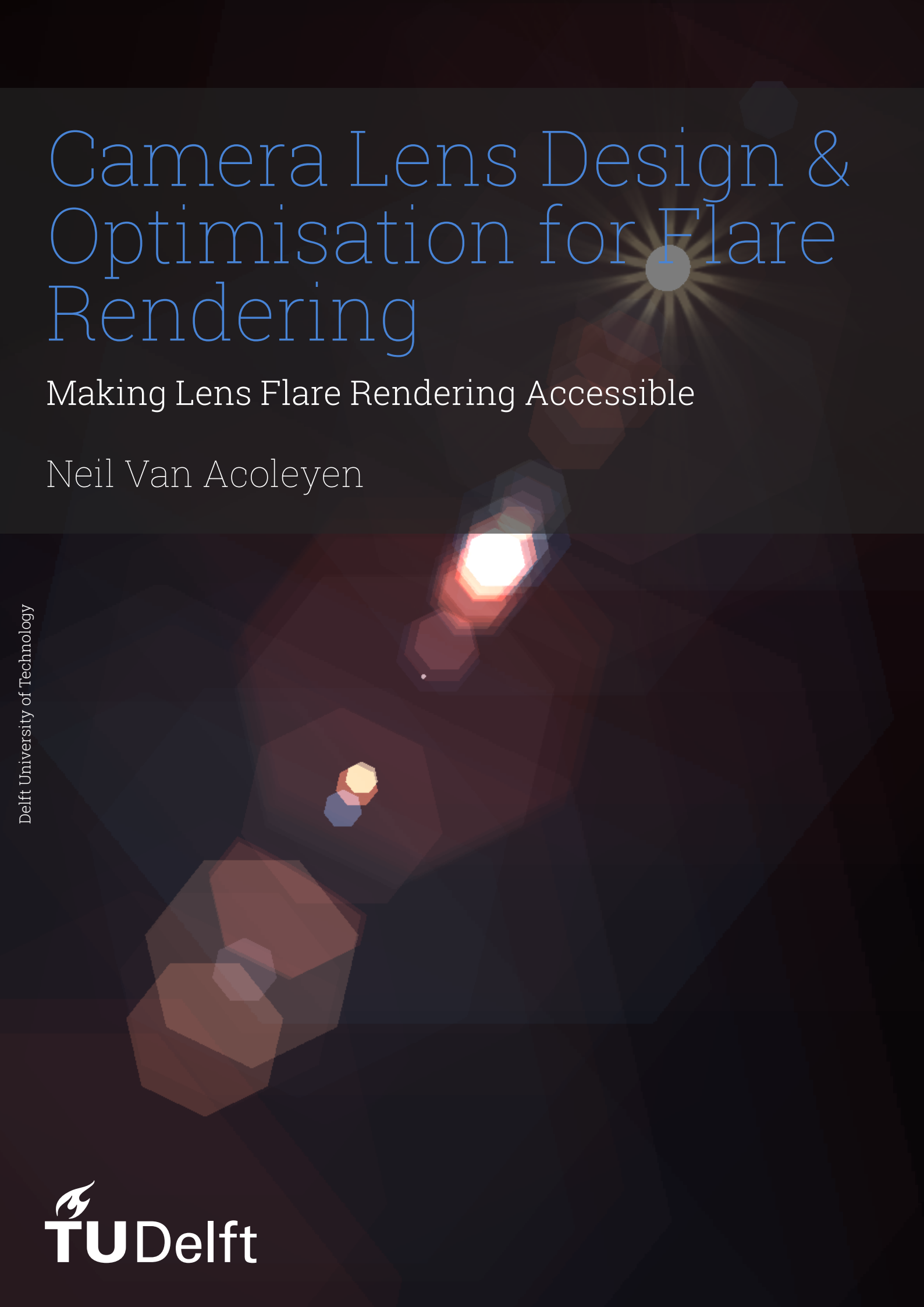
Camera Lens Design & Optimisation for Flare Rendering

Making Lens Flare Rendering Accessible

Neil Van Acoleyen



Delft University of Technology

TU Delft

Camera Lens Design & Optimisation for Flare Rendering

Making Lens Flare Rendering Accessible

by

Neil Van Acoleyen

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on July 10th, 2025, at 9:30 AM.

Student number:	4996097	
Project duration:	November 2024 – June 2025	
Thesis committee:	Dr. R. Marroquim,	TU Delft
	Prof. Dr. P.A.N. Bosman,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

The motivation behind this thesis stems from my passion for photography and computer graphics. I want to express my sincere gratitude towards my supervisor, Prof. Dr. Elmar Eisemann, for helping me find a project I could be passionate about. I am also thankful for his guidance, which contributed to the progress and quality of my work.

I am grateful to Dr. Ricardo Marroquim for stepping in as my supervisor during unforeseen circumstances and joining the committee on short notice. Together with Dr. Mark L. van de Ruit, he provided insightful and constructive feedback that significantly improved this thesis.

I am deeply grateful to my parents, Cuc and Koen, the wonderful people who have supported me unconditionally, not only throughout this thesis, but throughout my entire journey. It is thanks to them that I have come this far.

Neil Van Acoleyen
Delft, July 2025

Abstract

Lens flares occur when a bright light source induces light to travel through an optical lens system via unintended paths, reaching the sensor at an undesired location. Although arising from the imperfection of lens systems, flares are widely used in the visual entertainment industry for artistic purposes and to increase perceived brightness.

Research on physically accurate lens flare rendering has come far and produced convincing results by considering the inner construction of the lens and how light interacts with it. However, obtaining a specific flare signature through these algorithms requires lens design expertise that the typical artist does not possess.

This thesis demonstrates how to obtain optical lens systems that achieve a desired flare effect, without requiring prior knowledge of lens construction. We achieve this through simplified controls and evolutionary algorithms. With this method, existing lenses can be tweaked, or even new ones can be built from scratch. The process abstracts the lens and algorithmic parameters, making these rendering algorithms accessible to a wider range of users.

Keywords: Lens Flare Rendering, Camera Lens Design & Optimisation, Evolutionary Algorithms

Contents

List of Figures

1

Introduction

Figure 1.1: Real-time lens flare rendering in GTA VI. *Credit: Rockstar Games*.

Lens flares occur when a bright light source causes light to be reflected and diffracted within an optical lens system. This deviates the light from the intended path, and when reaching the sensor, leads to ghost and glare artefacts, also known as lens flares. While these effects can be unwanted, viewers associate their presence with a bright environment [31] and may find them visually appealing. Therefore, creators in the visual entertainment industry often incorporate flares into their work for artistic purposes or to surpass the contrast limit of the viewing medium.

Several methods have been developed for realistic lens flare rendering in recent years. Techniques such as raytracing and sprite-based approaches have been used to simulate how light interacts with lens systems. These methods are applied in games, films, and post-production to add flare effects to images and videos. The rendering pipelines use lens prescriptions to capture the inner workings of existing lenses. These are parametric descriptions of the lens structure and its multiple lens interfaces.

These camera lens prescriptions are not always publicly available, as the design is proprietary and often kept secret by manufacturers. This becomes a problem if an artist is after the flare effects of a specific lens. Another challenge is that designing these prescriptions requires expertise in optical systems. This makes it difficult for artists to adjust existing systems or create new ones to produce custom flares. Each optical element in the system is characterised by three parameters relevant to rendering. Since camera lenses typically consist of between 5 and 30 elements, the interaction among these parameters presents a highly complex problem.

In this thesis, I propose a method for addressing complex changes in lens flares using evolutionary

algorithms to explore the high-dimensional parameter space. User annotations made directly within the rendering viewport serve as targets for lens optimisation. Additionally, the system identifies the parameters for simpler modifications and allows direct manipulation. The proposed approach generates or adapts an optical lens system to achieve the desired flare effect.

The contributions made by my work include:

- An analysis of lens flare characteristics and their source in the lens prescription.
- A novel procedure for obtaining a camera lens design from desired flare properties.
- An analysis of evolutionary algorithms and their performance on optimising lens systems.

First, Chapter 2 introduces the foundational concepts necessary for understanding this thesis, followed by a summary of related work on lens flare rendering in Chapter 3. Then, Chapter 4 describes the flare renderer, including an analysis of the origins of each flare property. Next, Chapter 5 addresses the process of obtaining a lens design capable of producing the desired flares. Subsequently, Chapter 6 presents the performance results of the employed methods. Chapter 7 reflects on my work and discusses suggestions for future research directions. Finally, Chapter 8 concludes this thesis.

2

Background

This chapter introduces the concepts necessary to understand the methodologies and results discussed in this thesis.

2.1. Lens Flares

Lens Flares occur when light from a bright source is diffracted or reflected through a lens system. The light takes an alternative path through the multiple elements that make up the lens, resulting in artefacts on the image. Figure 2.1 shows an example of this path deviation.

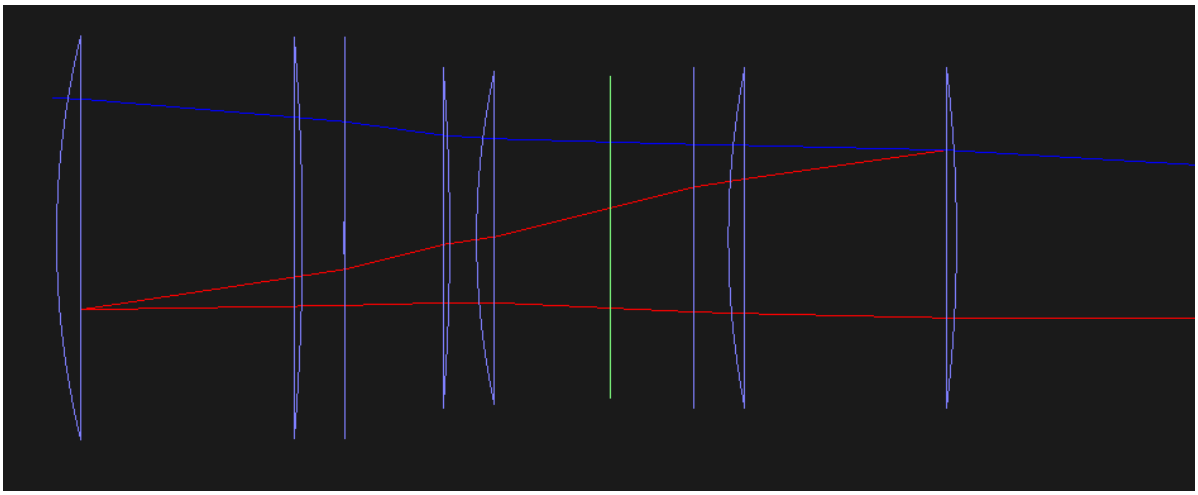


Figure 2.1: Schematic of light passing through the optical elements of a camera lens. The intended light path for a ray is shown in blue. The red line shows the path deviation from reflections at the first and last lens interface.

We identify two types of flares: ghosting artefacts and glare. Ghosting artefacts, or Ghosts, result from an even amount of reflections occurring in the lens system, causing stray light rays to reach the sensor at an unintended position [22]. Glare, often taking the shape of a starburst, is a result of diffraction through the lens interfaces [34]. In Figure 2.2, ghosts are highlighted in the red square, and glare can be seen as the starburst pattern on the sun, highlighted by the black square.

2.2. Lens Prescriptions

Lens systems are a composition of multiple interfaces made of different types of glass, other materials and air gaps. Figure 2.3 illustrates the inner construction of the Heliar Tronnier lens. These lens compositions are defined in their respective lens prescriptions, available in patents and other specialised sources, or remain unpublished by lens manufacturers. Table 2.1 details the prescription for the Heliar

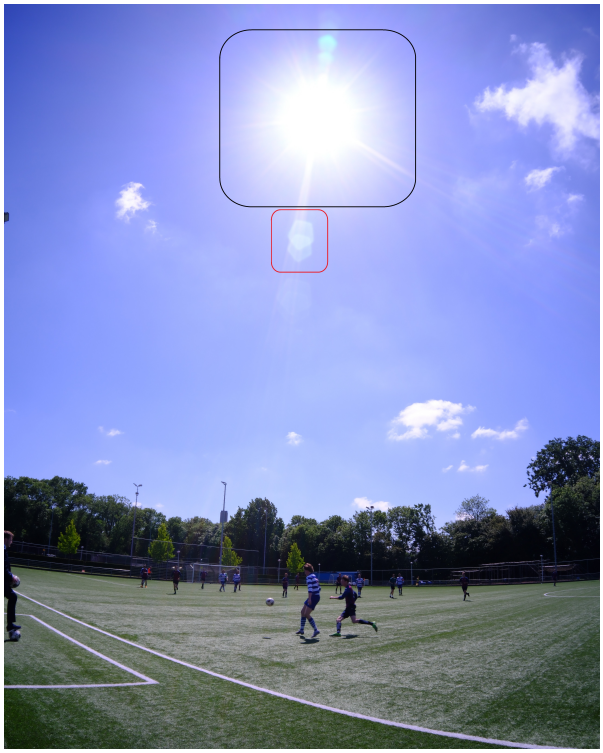


Figure 2.2: Real world flare artefacts example with the AstrHori 10mm f/8 II lens. The red square highlights ghost flares, and the black square highlights glare.

Tronnier lens. In this algebraic specification, we find values such as an interface's thickness (d), radius of curvature of the element (R), and refractive index (n). These prescriptions form the basis for recreating a specific lens's behaviour necessary for physically accurate lens simulations. Lee and Eisemann [26] define a lens system with k interfaces by the set of interfaces it is made of:

$$Ls \coloneqq \{I_1, I_2, \ldots, I_k\}$$

and each interface I_i is characterized as:

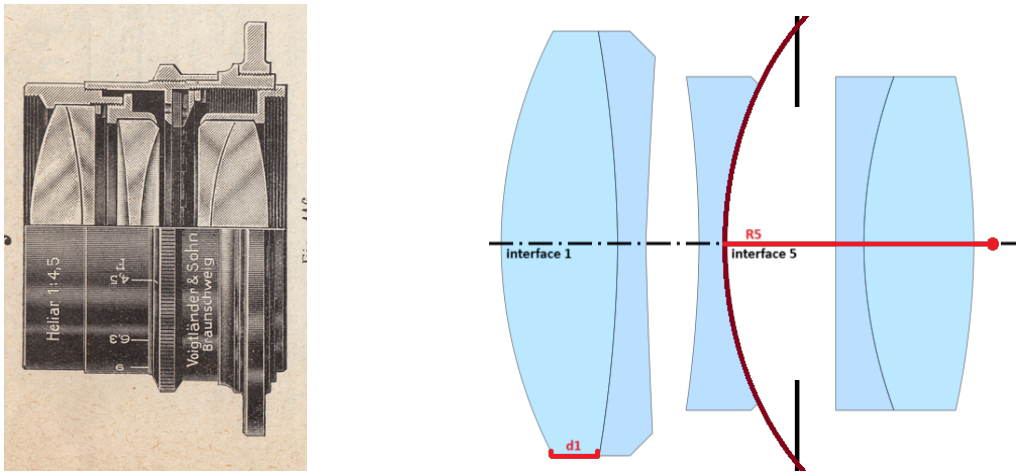
$$I_i \coloneqq \{d_i, R_i, n_i\}$$

where d_i denotes the thickness of interface i, R_i denotes the radius of interface i, defined as the distance from the point where the optical axis meets the interface surface to the centre of the sphere from which the curve of the interface is derived (See Figure 2.3b), and n_i denotes the refractive index of the material of interface i.

In a lens prescription, a positive radius indicates a convex interface. When the radius is not specified, the interface is flat. The refractive index is not stated for air gaps but can be safely approximated to 1.

Interface	Radius (R)	Thickness (d)	Material	Refractive Index (n)
1	30.810	7.700	LAKN7	1.652
2	-89.350	1.850	F5	1.603
3	580.380	3.520	air	
4	-80.630	1.850	BAF9	1.643
5	28.340	4.180	air	
6		3.000	air (iris aperture)	
7		1.850	LF5	1.581
8	32.190	7.270	LAK13	1.694
9	-52.990	81.857	air	

Table 2.1: Lens Prescription for the Heliar Tronnier (USP 2645156)



(a) Sideview. *Credit: Photo-Sport (Paris) Catalogue 1930.*

(b) Interface Schematic. *Credit: Mliu92/Wikimedia Commons, Licensed under CC BY-SA 4.0 (Annotated)*

Figure 2.3: Lens Construction of the Heliar Tronnier (USP 2645156). Left: Side view of the inner construction of the lens, where multiple curved interfaces can be observed. Right: Schematic of the lens interfaces, where the thickness of interface 1 ($d1$) and the radius of curvature of interface 5 ($R5$) are highlighted.

2.3. Anti-Reflection Coatings

Anti-reflection coatings in lenses use destructive interference between reflected light. When light interacts with an interface, part of the light is reflected and the rest is transmitted. When applying a thin-film coating, the reflections from the coating surface and the interface surface can be superimposed. If the reflected waves have equal amplitudes and opposite phases, they cancel each other out (Figure 2.4a). In practice, anti-reflection coatings are not perfect, so they minimise the reflectivity around a central wavelength [17] (Figure 2.4b).

A common type of coating is the quarter-wave coating. In this single-layer design, the coating's optical thickness is set to

$$t_c = \frac{\frac{\lambda_0}{4}}{n_c}$$

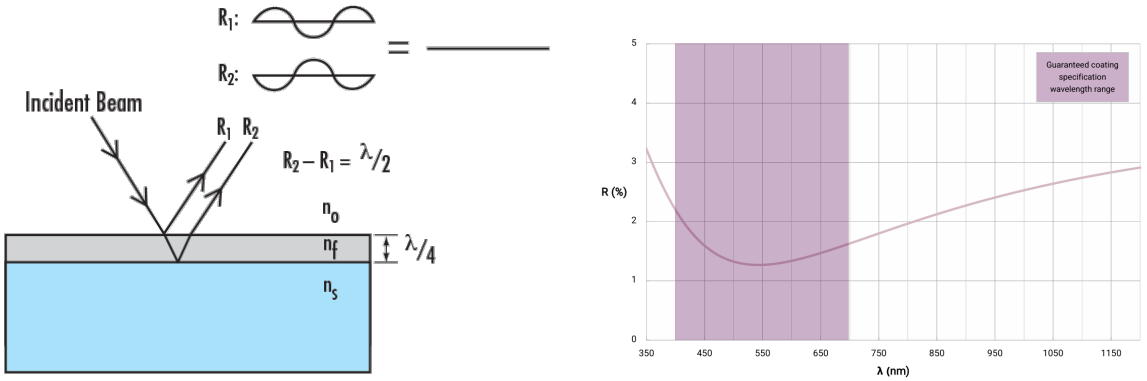
where λ_0 is the central wavelength, n_c is the refractive index of the coating, and t_C is the physical thickness of the layer. This configuration ensures that the reflected waves from the top and bottom surfaces of the coating are $\pi/2$ out of phase, minimising the reflection at wavelength λ_0 [17].

Hullin et al. [17] simulate the reflectivity of these quarter-wave coatings. Their computation, derived from the Fresnel equations, determines the refractive index and thickness of the coating layer. Given a ray with wavelength λ, an angle of incidence θ_0, the refractive indices n_1 and n_2 of the adjoining media, and a central wavelength λ_0, the computation scheme outputs the reflectivity R for λ. The transmission is then obtained as $T = 1 - R$.

2.4. Evolutionary Algorithms

Evolutionary algorithms (EAs) are metaheuristic optimisation techniques that draw inspiration from natural evolution. They encode potential solutions as individuals within a population and iteratively improve the population. Each individual is evaluated using a fitness function tailored to the specific optimisation objective [12]. Here, the fitness function evaluates how well a lens system can approximate a target flare render. EAs provide a robust framework for solving complex (black-box) optimisation problems, particularly when the parameter space is high-dimensional or non-linear. This is the case for our lens design optimisation.

Figure 2.5 depicts how the general EA scheme is applied to lens system optimisation. The algorithm begins by generating a diverse, random population of lenses. From this parent generation, a selection process chooses the lenses to create new candidate solutions (offspring). Crossover operators combine lens interface parameters from the selected parents, so offspring may inherit useful characteristics.



(a) Quarter-wave coating destructive interference. n_f = coating, n_s = lens element

(b) Reflectivity per wavelength for quarter-wave MgF_2 coating with $\lambda_0 = 550nm$.

Figure 2.4: Functioning of quarter-wave coatings. *Credit: Edmund Optics.*

Mutation then introduces small random changes, maintaining diversity and exploring new regions of the parameter space.

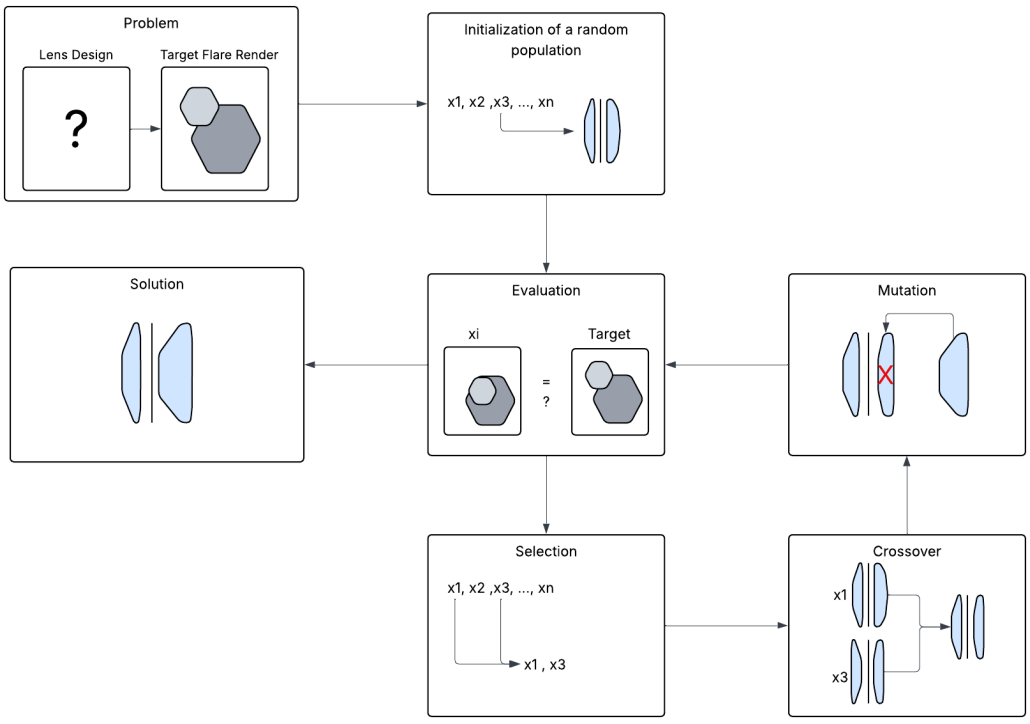


Figure 2.5: EA scheme in the context of lens design optimisation for a target flare render.

2.4.1. Archipelago Structure

The archipelago structure divides the population into several independent subpopulations, or islands, that evolve in parallel. Each island performs its instance of the EA. After several generations, individuals migrate between islands, sharing promising solutions and enhancing diversity. This decentralised approach helps prevent premature convergence and improves exploration. Island parallelisation allows multiple instances of the EA to run concurrently on multi-core or distributed computing systems.

2.4.2. Differential Evolution

This subsection explains how Differential Evolution (DE) [35], a type of EA, iteratively improves the population to find the best-performing solution. Each lens system is encoded as a vector where each value corresponds to a lens interface parameter. We denote the current generation as g and the individual i in the population at generation g as $\mathbf{x}_i^{(g)}$

Mutation

For each candidate $\mathbf{x}_i^{(g)}$, a mutant vector $\mathbf{v}_i^{(g)}$ is generated by selecting three distinct candidates $\mathbf{x}_{r_1}^{(g)}$, $\mathbf{x}_{r_2}^{(g)}$, and $\mathbf{x}_{r_3}^{(g)}$ such that $r_1, r_2, r_3 \neq i$. The mutant vector is defined as:

$$\mathbf{v}_i^{(g)} = \mathbf{x}_{r_1}^{(g)} + F\left(\mathbf{x}_{r_2}^{(g)} - \mathbf{x}_{r_3}^{(g)}\right),$$

where $F \in [0, 1]$ is the mutation scaling factor.

Crossover

A trial vector $\mathbf{u}_i^{(g)}$ is then created via a crossover operation. Given D lens interface parameters, for each dimension $j = 1, \ldots, D$, the j-th component of $\mathbf{u}_i^{(g)}$ is defined by

$$u_{ij}^{(g)} = \begin{cases} v_{ij}^{(g)}, & \text{if } \mathrm{rand}_j \leq CR \text{ or } j = j_{\text{rand}}, \\ x_{ij}^{(g)}, & \text{otherwise}, \end{cases}$$

where $CR \in [0, 1]$ is the crossover probability, rand_j is a uniformly distributed random number in $[0, 1]$, and j_{rand} is a randomly chosen dimension index to ensure that at least one component is taken from $\mathbf{v}_i^{(g)}$.

Selection

The next generation is formed by comparing the fitness values of $\mathbf{x}_i^{(g)}$ and $\mathbf{u}_i^{(g)}$. The candidate for the next generation is selected as:

$$\mathbf{x}_i^{(g+1)} = \begin{cases} \mathbf{u}_i^{(g)}, & \text{if } f(\mathbf{u}_i^{(g)}) \leq f(\mathbf{x}_i^{(g)}), \\ \mathbf{x}_i^{(g)}, & \text{otherwise}. \end{cases}$$

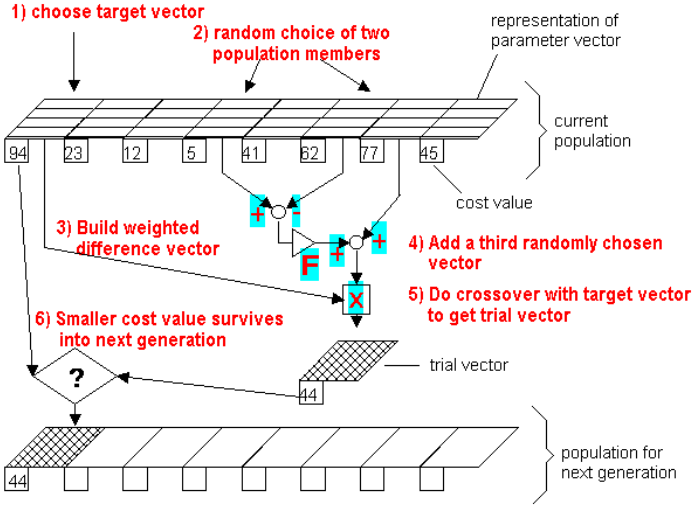


Figure 2.6: Differential Evolution Strategy. *Credit: Pagmo.*

2.4.3. Self-Adaptive Differential Evolution

Self-Adaptive Differential Evolution (SADE) adjusts the control parameters of classic DE (F and CR) during the optimisation process. In this thesis, I employ a variant known as jDE[8]. Each candidate solution $\mathbf{x}_i^{(g)}$ is associated with its own scaling factor $F_i^{(g)}$ and crossover probability $CR_i^{(g)}$. At each generation g, the parameters of individual i are updated stochastically before mutation and crossover. The update rules are as follows:

$$F_i^{(g+1)} = \begin{cases} F_l + r_1\, F_u, & \text{if } r_2 < \tau_1, \\ F_i^{(g)}, & \text{otherwise,} \end{cases}$$

$$CR_i^{(g+1)} = \begin{cases} r_3, & \text{if } r_4 < \tau_2, \\ CR_i^{(g)}, & \text{otherwise,} \end{cases}$$

where $r_1, r_2, r_3, r_4 \sim U(0,1)$ are random numbers drawn from a uniform distribution, F_l and F_u define the allowed range for F, and τ_1 and τ_2 are the update probabilities for F and CR, respectively.

Parameter values that improve fitness are favoured in subsequent generations, balancing exploration and exploitation without manual tuning. The mutation, crossover, and selection operators remain as described in the classic DE method.

2.4.4. Particle Swarm Optimisation

Particle Swarm Optimisation (PSO) [18] is based on the social behaviour of a swarm of particles. In this algorithm, candidate solutions are referred to as particles. Each particle i is represented by its position $\mathbf{x}_i^{(g)}$ and velocity $\mathbf{v}_i^{(g)}$ at generation g. The update rules are:

$$\mathbf{v}_i^{(g+1)} = \omega\, \mathbf{v}_i^{(g)} + c_1\, \mathbf{r_1} \cdot \left(\mathbf{y}_i^{(g)} - \mathbf{x}_i^{(g)}\right) + c_2\, \mathbf{r_2} \cdot \left(\mathbf{z}_i^{(g)} - \mathbf{x}_i^{(g)}\right),$$

$$\mathbf{x}_i^{(g+1)} = \mathbf{x}_i^{(g)} + \mathbf{v}_i^{(g+1)},$$

where:

- ω is the inertia weight.
- c_1 and c_2 are acceleration coefficients for the cognitive and social components, respectively.
- $\mathbf{r_1}$ and $\mathbf{r_2}$ are vectors of uniformly random values in $[0, 1]$.
- $\mathbf{y}_i^{(g)}$ is the best position found by particle i at generation g.
- $\mathbf{z}_i^{(g)}$ is the best position found by the particles within the neighbourhood of particle i at generation g.

PSO iteratively updates the particles' velocities and positions, guiding the swarm toward optimal solutions.

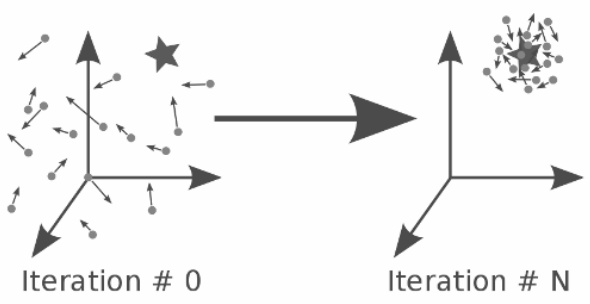


Figure 2.7: PSO particle convergence. *Credit: Pagmo.*

3

Related Works

Lens flare effects impact image quality but can also improve visual appeal. While Kuwabara [22] showed that flare reduces contrast, which can affect computer vision tasks where detail is critical, Ritschel et al. [31] found that adding glare can enhance the perceived contrast by the viewer, surpassing the limits of the viewing medium.

Early research on lens flare rendering relied on approximations and heuristics for imitating lens flares, placing flare sprites along a line [19]. Modern techniques use a camera model to simulate how light behaves within the optical system, giving rise to physically accurate lens flare rendering. Hullin et al. [17] developed a ray-tracing method that considers lens prescriptions, imperfections, lens coatings, and optical distortions. Their work remains a common reference for researchers and is used in industry [29]. Lee et al. [26] created a real-time method using sprites and matrices to make ray transfer calculations faster. Bodonyi et al. [6] focused on improving how pixel data is processed by organising ray-traced data into tiles and reducing the workload with a merging strategy. Later, Bodonyi et al. [5] replaced ray-tracing with a polynomial-based method, further speeding up rendering while keeping realistic results. Other approaches treat the lens system as a black box. Both Walch et al. [39] and Maurer et al. [28] used a data-driven model for flare prediction. Libraries like OpenLensFlare [9] and RealFlare [30] offer developers tools to render physically accurate lens flares through preset systems and direct lens design parameter editing.

There are multiple obstacles when adopting the available physically accurate flare rendering techniques. Several books, including Kingslake [20], Laikin [23], and Shannon [33], provide foundational knowledge of optical design. These books demonstrate the complexity of the lens design process and the necessary expertise to create a system for a specific purpose. Lens optimisation software, often relying on neural network processing [37, 41], is widely available to simplify this process. However, it does not focus on the resulting flares and requires multiple parameters to be fixed beforehand [40]. Additionally, they have the flaw of not providing the user with a starting point [36]. Lü et al. [27] and Zhang et al. [43] demonstrate how particle swarm optimisation, an evolutionary algorithm, is employed to optimise camera parameters for camera calibration, often used in computer vision tasks. Blackler et al. [4, 3] emphasised that tools for artists should have intuitive controls, aligning with workflows they are already familiar with. Following this principle, Sangmin Lee [25] proposed an interactive framework for editing glare images derived from the Fourier transform of aperture shapes.

This collection of research highlights the importance of lens flares, the progress in lens flare rendering, mainly focused on improving accuracy and speed, and the current methods for optimising camera lenses.

4

Lens Flare Rendering

This chapter discusses the chosen lens flare rendering technique. By analysing the renderer, we gain a deeper understanding of the origin of each flare property and how these properties can be manipulated.

4.1. The Rendering Pipeline

Among the many methods available, I chose to use the practical real-time approach by Lee and Eisemann [26]. This sprite-based method adopts a simpler rendering mechanism that approximates each interaction of a light ray with a lens interface through a 2x2 matrix. This allows the propagation of light through the optical system to be computed in constant time, a key feature for achieving high frame rates. This factor is important for ease of use [38]. However, the model cannot simulate non-linear ray propagation. Consequently, we sacrifice non-linear ghost deformations in favour of efficient real-time computation.

Although this thesis's methodology is built around this lens flare rendering technique, the same logic can be applied to other realistic rendering algorithms that also rely on a lens prescription.

4.2. Ray Transfer Model

A ray transfer model defines the behaviour of light propagation through the lens. This is necessary to project points from the entrance plane onto the sensor plane, for a given light direction. Following Lee et al. [26], a light ray **r** is represented as a two-dimensional vector:

$$\mathbf{r} = \begin{bmatrix} r \\ \theta \end{bmatrix}$$

where r is the signed offset of the ray's origin from the optical axis (axis along the centre of the lens), and θ is the angle between the ray's direction and the optical axis, measured positively for upward directions (Figure 4.1). Each interaction of the ray with an interface can be approximated through the matrices in Table 4.1.

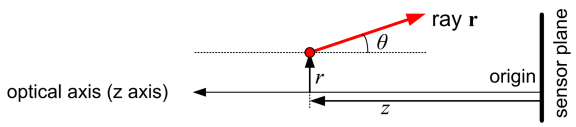


Figure 4.1: Light ray representation *Credit: Lee and Eisemann [26]*

Using this model, the ray propagation through the lens system is defined with straightforward matrix multiplications and can be computed in constant time.

Optical component	Ray transfer matrix
Translation (T_i)	$\begin{pmatrix} 1 & d_i \\ 0 & 1 \end{pmatrix}$
Refraction at spherical dielectric interface (R_i)	$\begin{pmatrix} 1 & 0 \\ \frac{n_1-n_2}{n_2 R_i} & \frac{n_1}{n_2} \end{pmatrix}$
Reflection from a spherical mirror (L_i)	$\begin{pmatrix} 1 & 0 \\ \frac{2}{R_i} & 1 \end{pmatrix}$

Table 4.1: Ray transfer matrices where d_i denotes the thickness of the optical element under consideration, n_1 and n_2 represent the refractive indices of the materials on the incident and transmitted sides of the interface, respectively, and R_i is the radius of curvature of that interface [26].

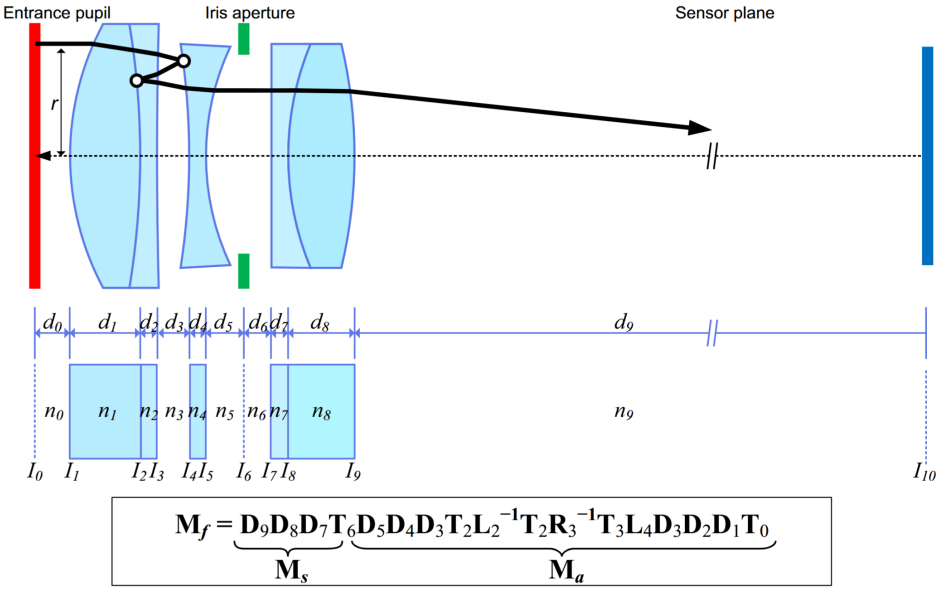


Figure 4.2: Flare matrix formulation for Heliar Tronnier. I, d, and n indicate optical interfaces, distances between optical interfaces, and refractive indices, respectively. *Credit: Lee and Eisemann [26]*

4.3. Rendering Ghosts

To obtain a specific ghost flare look, we must first understand how these are rendered and identify where the different properties come from.

4.3.1. Ghost Amount

Ghosts are the result of an even number of reflections, causing the light to hit the sensor at another location. To render all of them, we must first consider the totality of paths through the lens that lead to a ghost in the image. In our system, each possible light path through the lens system that includes two reflections on the same side of the iris aperture is enumerated. The iris aperture is an adjustable diaphragm within the lens that controls the amount of light entering the system. In the lens prescription, it appears as an air gap, and is one of the only interfaces (the other one being the entrance pupil) for which we consider the height. Only paths including two reflections are considered because additional reflections induce severe intensity losses, resulting in ghost images with negligible brightness. Moreover, when the reflections occur on opposite sides of the aperture, the resulting ghosts are likely to be blocked by the aperture. These considerations accelerate the rendering process, and the number of paths corresponds to the number of ghosts. Changing the number of optical interfaces and changing the aperture position will, therefore, affect the number of ghosts rendered. One additional consideration has to be made in this work because lens interfaces can be adjusted: no reflections between two air gaps. In real lens prescriptions, this case does not arise, but a check is added to remove the ghost if it were to arise from a reflection between two air gaps.

For each ghost path, we concatenate the propagation matrices into two. The first matrix M_a is built by multiplying all the propagation matrices for each interface from the entrance pupil to the iris aperture. The second matrix, M_s, is built by multiplying all the propagation matrices for each interface from the

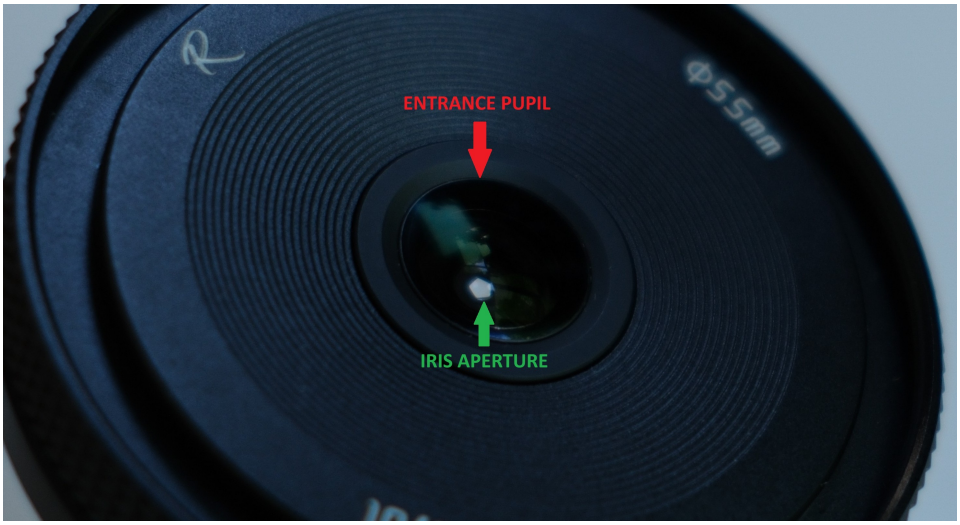


Figure 4.3: Entrance Pupil and Iris Aperture of the AstrHori 10mm f/8 II lens.

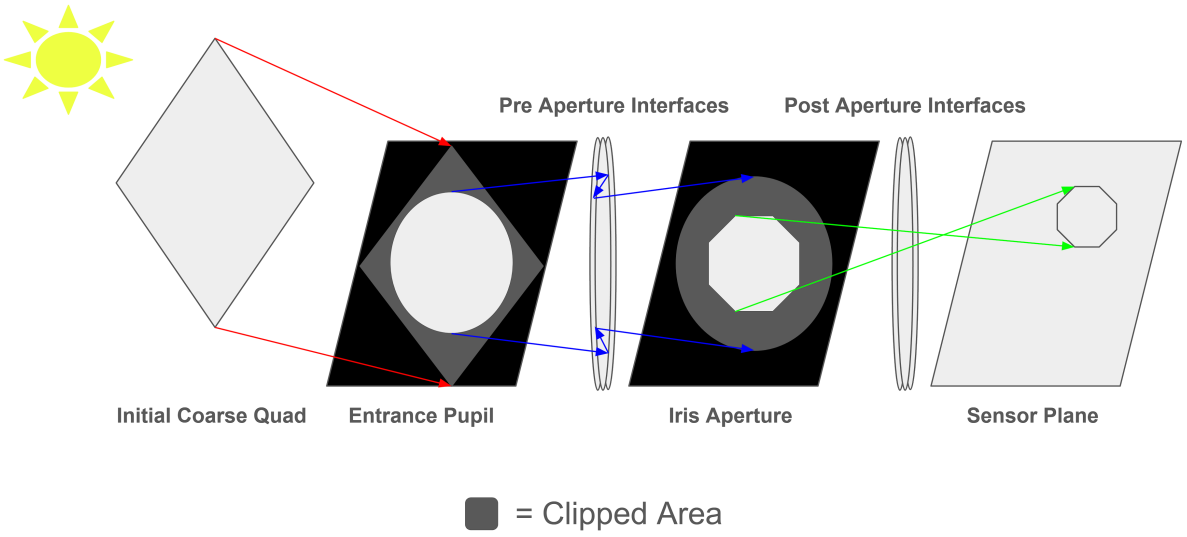
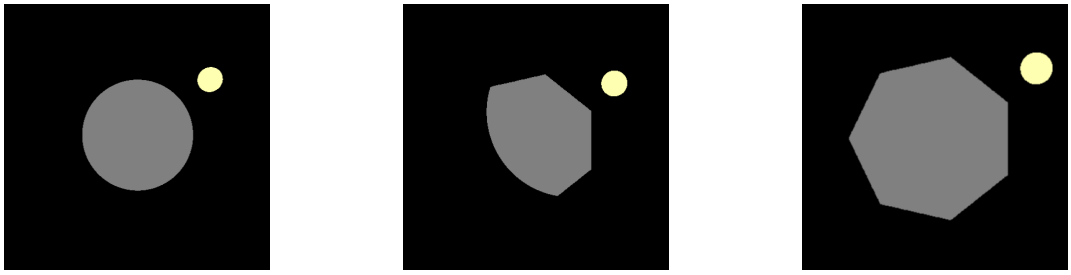


Figure 4.4: Schematic of quad clipping through the lens system.

(a) f/2.8 **(b)** f/5.6 **(c)** f/11

Figure 4.5: Aperture shape of the Pentacon Prakticar 50mm f/1.8 at different aperture settings.

Figure 4.6: Example texture of the iris aperture.



(a) Fully clipped by the entrance pupil. **(b)** Clipped by entrance pupil and aperture. **(c)** Fully clipped by the aperture.

Figure 4.7: Example results of ghost clipping. The yellow circle corresponds to the light source.

iris aperture to the sensor. This two-step propagation is needed to perform clipping at the iris aperture and other calculations explained in Section 5.5.

We now denote two groups of ghosts, those with reflections pre-aperture and those with reflections post-aperture. Ghosts in the first group will share a common M_s and the second group a common M_a. The other matrix will be unique to each ghost.

4.3.2. Ghost Location and Size

To render the ghosts, the propagation matrices project a quad that covers the entire entrance pupil onto the sensor plane for a given light direction. The matrices thus determine the location and size of the quad projection on the sensor. This directly translates to ghost location and relative ghost size.

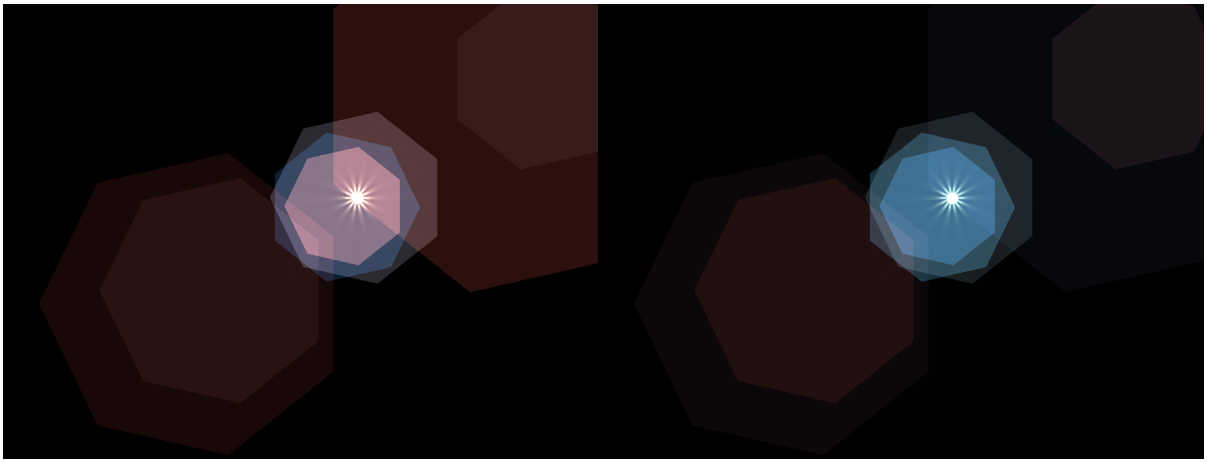
4.3.3. Ghost Shape

The quad projections will determine the size differences between different ghosts, but the ghost shapes come from another factor. The quads are clipped at two different stages in the propagation through the lens, leading to the final size and shape of the ghosts as shown by the schematic in Figure 4.4.

The first clipping stage happens at the entrance pupil, mimicking the blocking of light from the lens barrel, only allowing in light that passes through the entrance pupil. To simulate this, the quad is refined using a distance threshold relative to the optical axis, giving it a round shape. The second clipping stage takes place at the iris aperture. In the camera lens, the aperture stop is composed of multiple blades that rotate inward to let only a central opening of light through. The shape of this opening often takes the form of a regular polygon, as can be observed in Figure 4.5. Applying an aperture sprite (via a texture lookup) to the quad at the corresponding aperture position during ray propagation simulates the light blocking. Figure 4.6 illustrates an example sprite for an aperture with 7 blades. The combination of the different clipping stages results in various types of ghost shapes, as depicted in Figure 4.7.

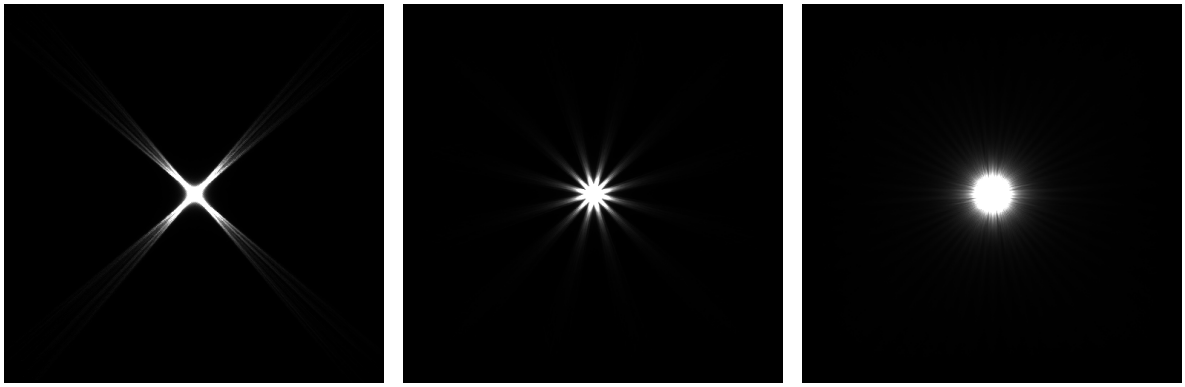
4.3.4. Ghost Colour

Each ghost's colour is determined by simulating the reflectivity of quarter-wave coatings on the lens interfaces. Each quarter-wave coating is optimised around a central wavelength λ_0. Although these coatings are engineered to minimise reflections via interference, they also produce the distinctive hue of each ghost. The detailed formulations of these coatings are proprietary and remain confidential to



(a) Coating Set 1. **(b)** Coating Set 2.

Figure 4.8: Different coating settings for the same lens (Heliar Tronnier).



(a) 4 blades. **(b)** 12 blades. **(c)** 25 blades.

Figure 4.9: Starburst texture for different aperture shapes.

manufacturers. I use the reflectivity simulation code provided in the supplementary material of [17]. For each ghost, calculating the colour by propagating a central ray through the system, while computing its transmission and reflectivity for each RGB channel at each interface, adequately captures its overall colour. The colour is then adjusted by an intensity factor computed as the ratio of the aperture height to the ghost height. This is to simulate the concentration of the amount of light passing through the aperture onto a smaller or larger surface. The final colour of each ghost is thus computed as:

$$RGB_{\text{ghost}} = RGB_{\text{coatings}} \cdot I_{\text{light}} \cdot \frac{h_{\text{aperture}}}{h_{\text{ghost}}}$$

Figure 4.8 demonstrates the effect of different coating settings on the ghost colours.

4.4. Rendering Glare

A starburst texture is generated by processing the aperture image using the Fraunhofer approximation to simulate how light diffracts through an optical aperture. This is the same technique from Hullin et al. [17]. Figure 4.9 highlights the impact of the aperture shape on the glare pattern.

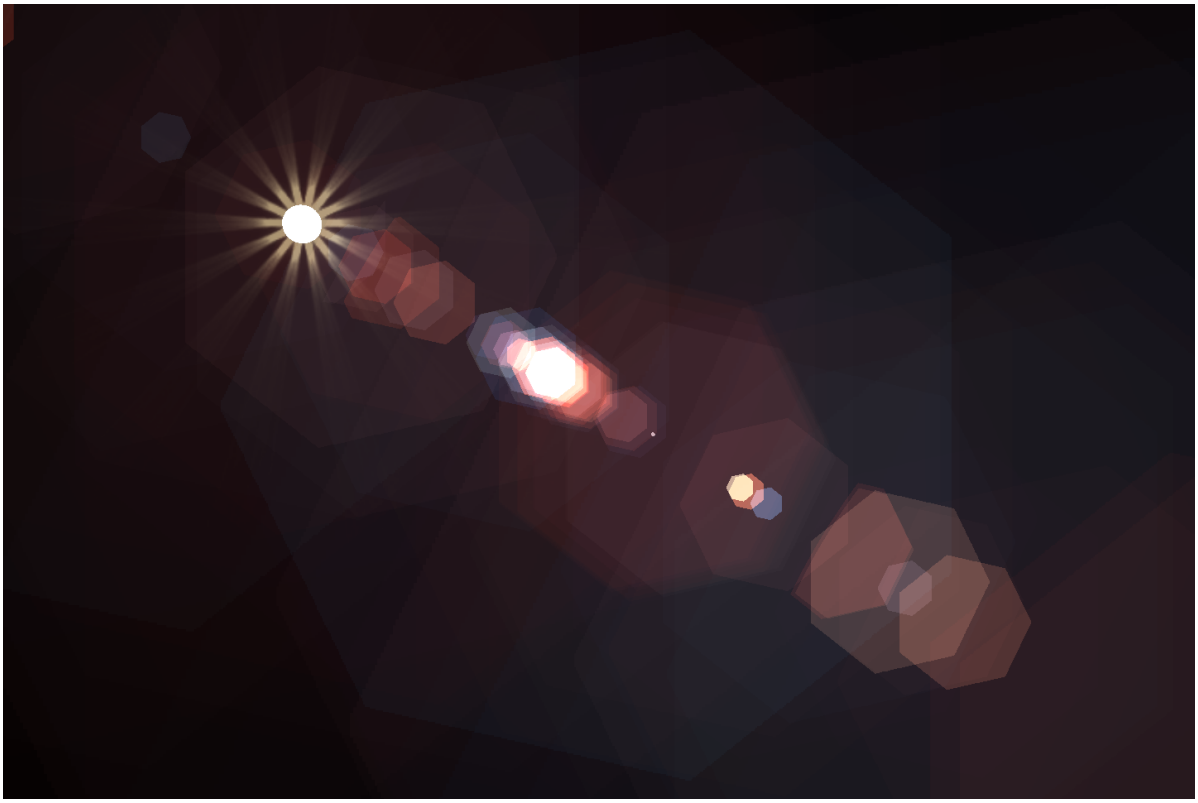


Figure 4.10: Flare Render of the Canon 28-80mm f/2.8 (US5576890) lens.

5 Methodology

In the following chapter, I develop my approach. I will first focus on how the artists can interact with the flare render, then delve into direct changes to the lens design for simple effects, and finally propose a solution for addressing complex changes to the render.

5.1. Controls

To minimise the adoption threshold, I develop controls similar to existing artist tooling, such as click-and-drag navigation to change the viewing direction, useful in our case to change the incoming light direction. If viewport editing is impractical for a given action, we provide the user with the relevant prescription values that need adjustment.

5.1.1. Selecting Ghosts

A key operation in ghost editing is selecting a target ghost. This is necessary to supply the user with the right controls. The transfer matrices determine where the initial quad projects onto the sensor. By assigning a unique Ghost ID to each quad and accounting for clipping, we can retrieve the list of ghosts rendered at the selected position. The intensity of each ghost at that point is also extracted, allowing the list to be sorted by decreasing intensity. This prioritises smaller, brighter ghosts, which are typically more noticeable. The user can scroll through the list, with the selected ghost highlighted for clarity, as shown in Figure 5.1.

5.1.2. Additional Controls

A feature useful during flare development is disabling the colours derived from the anti-reflectivity coating simulation. The RGB colours are replaced with a fixed intensity value. Note that the intensity adjustment based on ghost size remains active. The greyscale helps editing less visible ghosts due to coating computations. Another practical feature is the ability to disable entrance clipping. Removing entrance clipping is needed to edit ghosts hidden under the current light direction. The effects of both

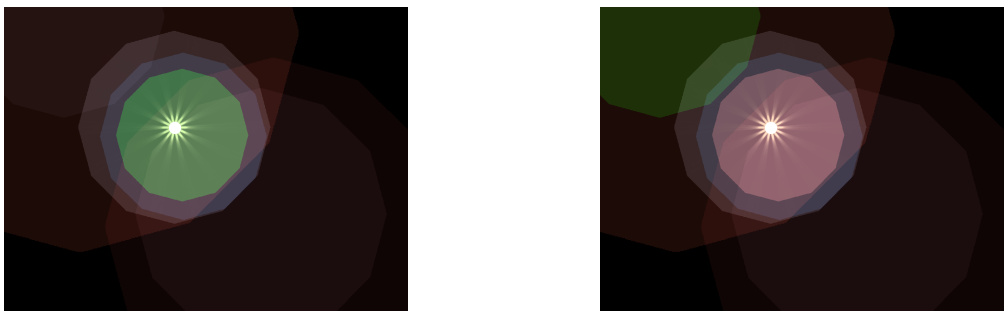
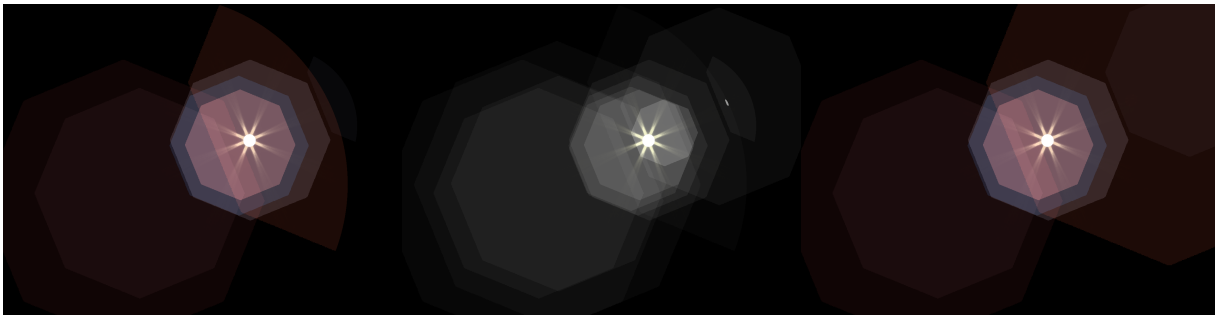


Figure 5.1: Selected ghost highlighted in green.



(a) Original. **(b)** Greyscale. **(c)** Entrance Clipping Disabled

Figure 5.2: Additional settings helpful during development.

functionalities can be observed in Figure 5.2.

5.2. Physical Bounds for Lens Interface Parameters

Before adjusting a lens prescription, physical bounds and constraints must be set to match ranges observed in real camera lenses. This restricts the search space for the EAs in section 5.6 and improves the accuracy of the physical modelling.

5.2.1. Upper and Lower Bounds

Table 5.1 shows the lower and upper bounds used to constrain parameters within the range of typical camera lens materials. Radii can extend to infinity to represent flat surfaces. However, increasing the radius has a negligible visual impact beyond a certain point. Empirically, a threshold of 1000 was found sufficient to approximate flatness while limiting the search space and reducing complexity. Additionally, the anti-reflective coatings' central wavelength λ_0 is restricted to the previously defined visible spectrum range (380–750 nm).

Parameter	Lower Bound	Upper Bound
Thickness di (mm)	0.1	100.0
Refractive Index n_i	1.0	2.0
Radius R_i (mm)	-1000.0	1000.0
Coating λ_0 (nm)	380	750

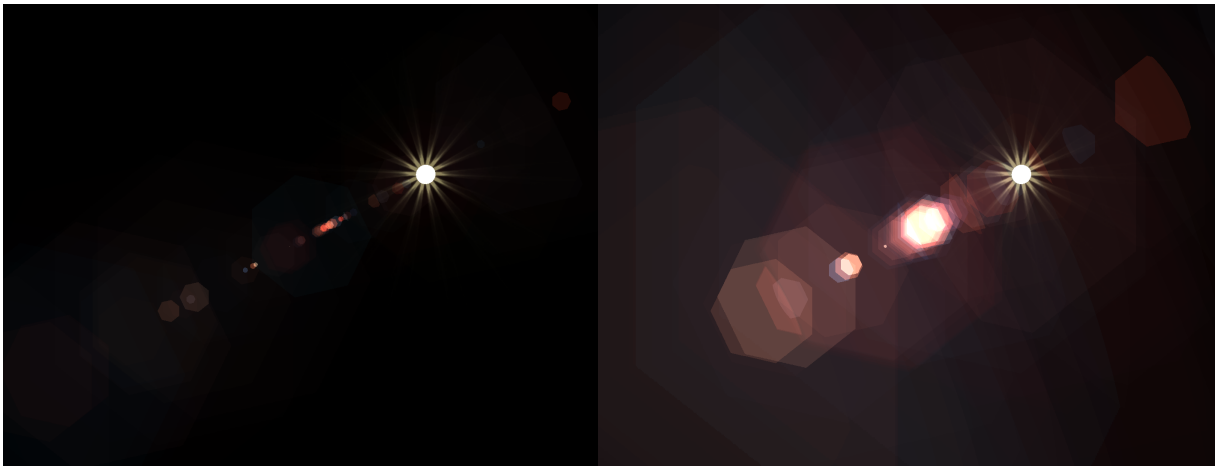
Table 5.1: Parameter Bounds

5.2.2. Constraints

In designing lens systems, several constraints must be considered. Although air gaps between optical components can be relatively large, the thickness of material interfaces is generally confined to the range of 1–15mm [15]. Since these interfaces are primarily intended to refract light, increasing their thickness beyond a certain limit offers little benefit and incurs additional costs [23]. Moreover, while air gaps have a refractive index of 1, material interfaces typically exhibit refractive indices in the range of 1.5–2.0 [1]. Although materials with refractive indices outside this range are available, often used in specialised equipment such as microscopes, they are rarely employed in standard lens design. Finally, the curvature of lens surfaces imposes further restrictions. As the radius of curvature approaches zero, surfaces become excessively curved, which is impractical due to physical limitations related to lens thickness and material characteristics. Therefore, the range near 0 is excluded (we choose [-5, 5]).

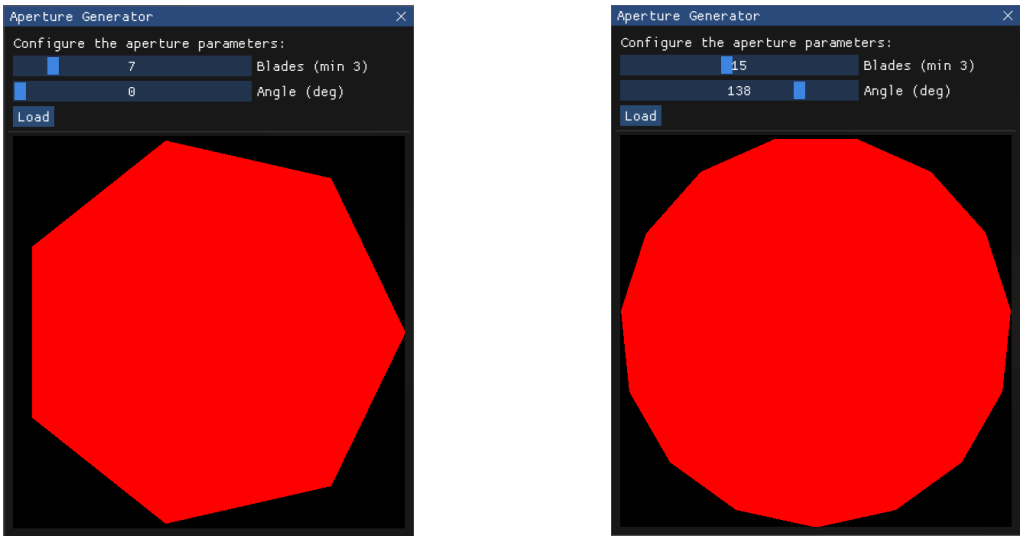
5.3. Direct Changes

The following section tackles changes to the lens flares that can be done by directly modifying the lens prescription and do not require complex solving methods.



(a) 3 mm **(b)** 15 mm

Figure 5.3: Effect of resizing the aperture height on the flare rendering.



(a) 7 blades, 0 degrees **(b)** 15 blades, 138 degrees

Figure 5.4: Aperture Generator

5.3.1. Uniform Ghost Resizing

Adjusting the entrance pupil height changes the extent of clipping on a ghost. In cases where a ghost is fully clipped by the entrance pupil, its size and shape will consequently change. A smaller entrance pupil produces smaller ghosts. Similarly, modifying the aperture height alters the size of all ghosts not clipped by the entrance pupil uniformly, as demonstrated by Figure 5.3. These parameters produce global effects on the rendered image, enabling a uniform resizing of all ghost images.

5.3.2. Ghost Shape

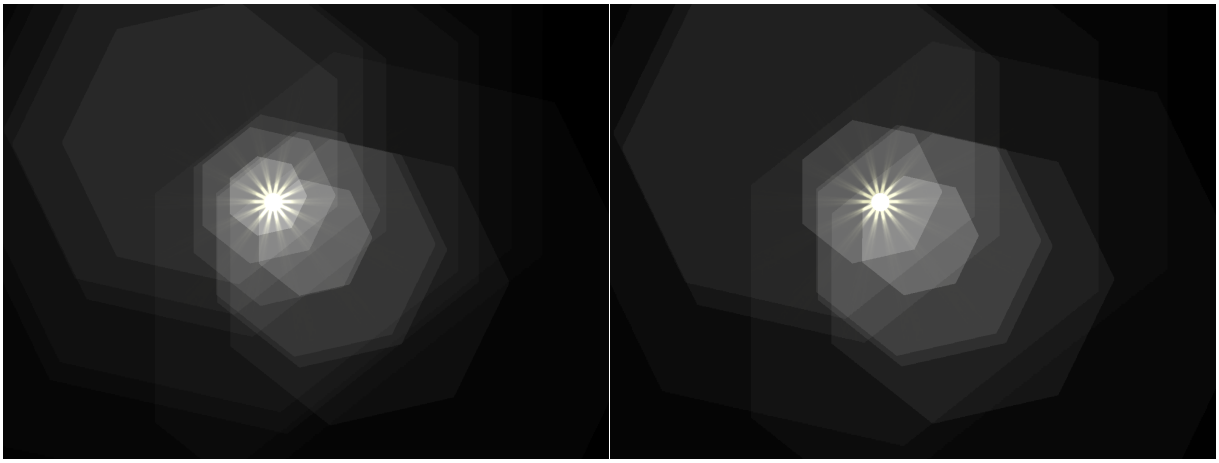
To adapt the ghost shapes, the aperture shape must be modified. Two approaches can achieve this:

Iris Aperture Opening Simulation: The shape of the iris aperture opening in commercial camera lenses can be simulated with a regular polygon, as depicted in Figure 5.4. Two user inputs are needed: the number of aperture blades and a rotation angle. We generate a regular polygon with an equal number of edges to the specified blade count. This polygon is rotated by the specified angle offset and scaled to fill the mask, ensuring that at least one vertex reaches the edge.

(a) Example of custom aperture mask.

(b) Flare render of the Heliar Tronnier with custom aperture.

Figure 5.5



(a) Second interface as aperture

(b) Third interface as aperture

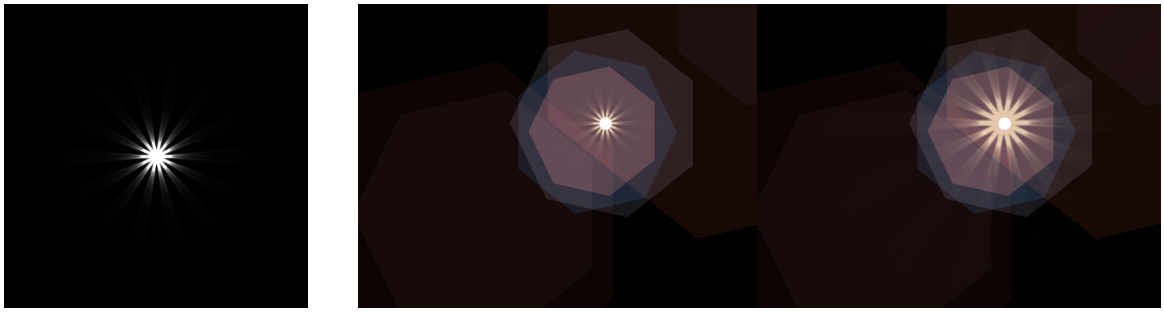
Figure 5.6: Greyscale flare render of the Heliar Tronnier with different aperture positions.

Custom Aperture Mask: The user uploads a custom square aperture mask. This option remains physically accurate, but no longer mimics commercial camera lenses. However, it enables more creative ghost rendering. Figure 5.5 shows an example render using a custom aperture mask.

This dual approach offers flexibility, allowing the user to choose between the common camera lens model (Iris Aperture Opening Simulation) and creative freedom (Custom Aperture Mask). After adjusting the aperture shape, the glare texture is recomputed to obtain the new diffraction pattern.

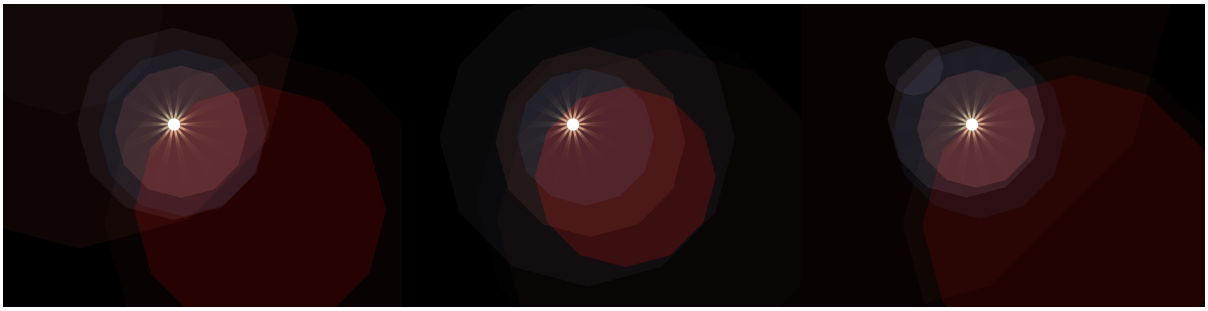
5.3.3. Adjusting the Aperture Position

In a physical camera lens, the aperture position is fixed. In contrast, our renderer allows us to adjust this parameter freely. To maintain physical accuracy when selecting a different lens interface as the iris aperture, its curvature and refractive index must be overwritten to simulate a flat air surface. This is done by setting the radius of curvature to infinity and the refractive index to 1. Following this adjustment, all potential reflection pairs, which depend on the aperture position, must be recomputed, and the corresponding transfer matrices updated to generate a new render. As a result, some ghost artefacts remain unchanged, some disappear, and new ones are produced (Figure 5.6).



(a) Starburst texture. **(b)** Smaller starburst. **(c)** Bigger starburst.

Figure 5.7: Glare scaling.



(a) Original. **(b)** Smaller. **(c)** Bigger.

Figure 5.8: Changing the size of the selected ghost (highlighted in red) through the radius of the first reflection matrix.

5.3.4. Starburst Adjustments

The starburst texture rendered at the light source location can be resized to make the glare bigger or smaller (Figure 5.7). The intensity of the starburst can also be adjusted easily through a multiplier.

5.3.5. Changing the selected ghost size

While resizing an individual ghost without affecting the whole flare signature is a complex problem involving multiple transfer matrix parameters, some heuristics can be made to approximate the desired effect. Adjustments to the thickness, refractive index, or radius of any given interface will affect the size of all ghosts. To modify a specific ghost, the most direct approach is to alter one of its reflection matrices, as each ghost is primarily defined by two such matrices. Although these matrices are not unique, since different ghost paths can share a reflection interface, changing one of the two reflection matrices is generally sufficient to produce the rescaling effect.

Within the reflection matrix, the radius value is the only parameter. Modifying this value offers a means to adjust the ghost's size while minimising unintended changes in other ghosts (Figure 5.8). It should be noted, however, that this approach is not flawless. Especially at low radius values, the optical system's overall flare signature is significantly altered.

To enable further exploration, the other parameters, thickness and refractive index, are available for modification. By restricting the options to only two interfaces, the search space is reduced, allowing the user to explore new flare setups without being overwhelmed.

5.3.6. Changing the selected ghost colour

Modifying the coatings at the reflection interfaces enables the tuning of reflectivity toward a target colour for the selected ghost. In our coating reflectivity model, two factors contribute: the wavelength for which the coating is optimised (λ_0) and the refractive indices of the materials on either side of the coating. During colour optimisation, the refractive indices are held constant, since modifying them would also affect the size and position of ghost images. This leaves the λ_0 parameter of the 2 reflection interfaces as the only variables to optimise for the desired colour.

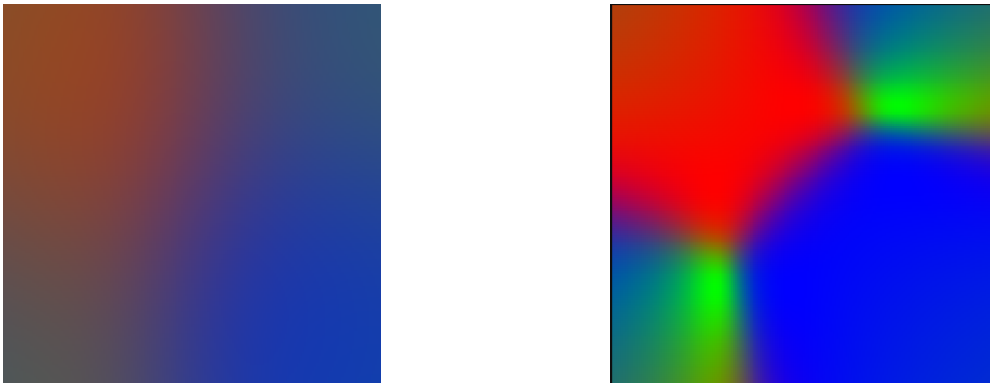


Figure 5.9: Example colour maps for the λ_0 combinations of a reflection interface pair.

Isolating λ_0 in reflectivity calculations leads to a non-linear periodic relationship. Each RGB channel requires minimising a distinct error equation per interface, forming a system of six non-linear equations. Instead of using gradient-based solvers prone to local optima or computationally expensive alternatives, we apply a grid search across the visible spectrum. The bounded range and 2 nm step size ensure efficient computation while capturing colour granularity.

This approach not only finds the optimal parameters for a target colour but also generates a colour map displaying the λ_0 combinations and their resulting colours (see Figure 5.9). This is useful for the user to build an understanding of the achievable colours with the current refractive indices. In this map, each cell shows the colour of the normalised combined reflectivity:

$$C^{\mathrm{RGB}}_{\lambda_{0\mathrm{refl}_1},\lambda_{0\mathrm{refl}_2}} = \frac{I^{\mathrm{RGB}}_{\lambda_{0\mathrm{refl}_1}} \odot I^{\mathrm{RGB}}_{\lambda_{0\mathrm{refl}_2}}}{\left\| I^{\mathrm{RGB}}_{\lambda_{0\mathrm{refl}_1}} \odot I^{\mathrm{RGB}}_{\lambda_{0\mathrm{refl}_2}} \right\|} \tag{5.1}$$

where $\lambda_{0\mathrm{refl}_1}$ and $\lambda_{0\mathrm{refl}_2}$ denote the optimised wavelengths for the first and second reflection interfaces, respectively.

Additionally, we produce a graph (Figure 5.10) that displays the per-channel reflectivity for a single interface, isolating the effect of its refractive index on the colour.

As a side effect, this adjustment technique also alters the colour of other ghosts that share the same reflection interface or, to a lesser extent, through transmission.

5.4. Unlocking Quarter-Wave Coatings

Until now, the rendering pipeline has been configured to simulate quarter-wave coatings. In this simulation, the coating's refractive index and thickness are determined from a central wavelength (λ_0) and the refractive indices of the surrounding media. However, by manually setting the coating's refractive index and thickness, we can explore new ghost colours. The thickness is kept within 25-750nm, and the refractive index between 1.38-1.9. These ranges capture the limits for common single-layer and multi-layer coating solutions [42, 24].

These custom coating solutions deviate from conventional lens design but enable more artistic freedom (Figure 5.11). We allow the user to select between quarter-wave coatings or custom coatings.

5.5. User Annotations

For changes to the flare render that cannot be solved by simple changes in the lens interface parameters, without changing the overall signature of the flares, the user can bypass the physically accurate render through annotations. These annotations are saved for each ghost and contain a target position, height and colour. These changes are then applied to the realistic render to create a ”fictional” target render. The annotations for position and height are performed in greyscale to enable adjustment of ghosts barely visible due to their current coating reflectivity computations. Once the annotations are

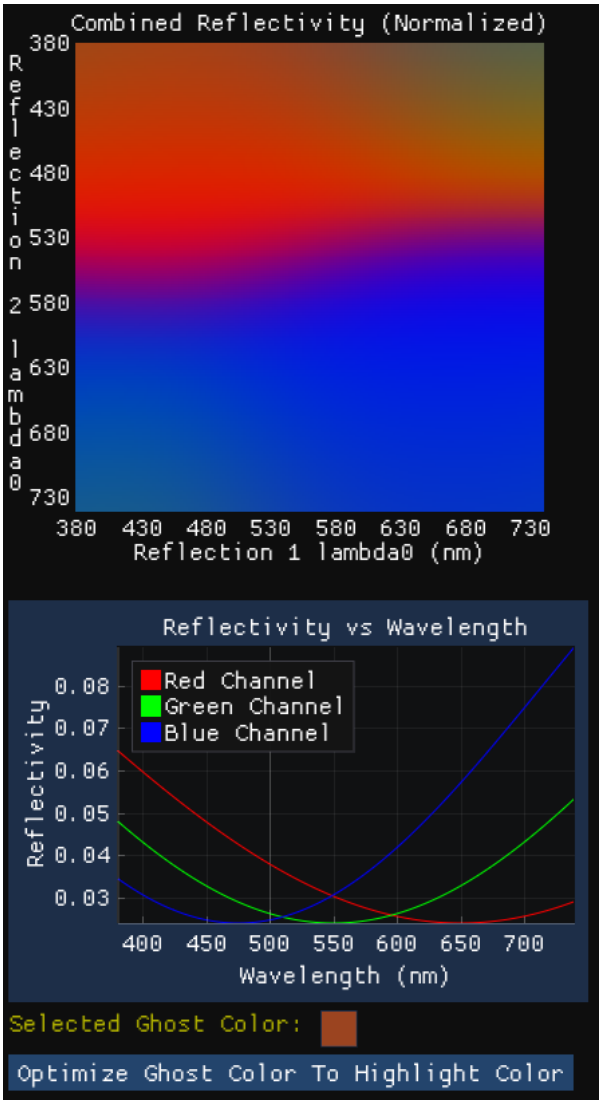


Figure 5.10: Visual guidance for adjusting coatings.

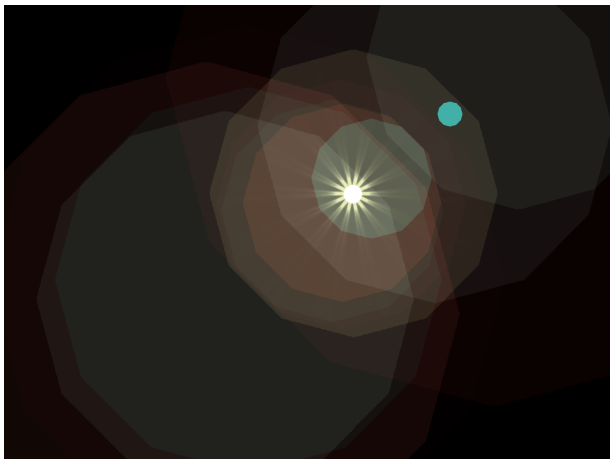


Figure 5.11: Example of ghost colours achievable with custom coatings but not with quarter-wave coatings

final, they are fed to the EA process to find a lens system that best approximates the newly created target render. This is explained in Section 5.6.

5.5.1. Ghost Position

When modifying a ghost's position, the user selects the target ghost and moves it to a new location. The new target position is restricted to the line connecting the optical centre and the light source. Allowing the ghost to deviate from this line would conflict with the underlying physics of ghost formation.

After establishing the new target position, the next step is to compute the difference vector between this target and the ghost's current position to deliver visual feedback. To obtain a ghost's position, we need the ghost's centre point.

Following the ray transfer model (Section 4.2), a ray is represented as a two-dimensional vector $\mathbf{r}$ and is sequentially transformed by two 2×2 matrices. The first, M_a, propagates the ray from the entrance pupil to the iris aperture, and the second, M_s, propagates the ray from the iris aperture to the sensor plane:

$$\mathbf{r}_{\text{sensor}} = M_s \, M_a \, \mathbf{r}_{\text{entrance}} \tag{5.2}$$

When a ghost is fully clipped by the entrance pupil, its centre corresponds to the projection of the centre of the entrance pupil onto the sensor for a given light direction. However, when a ghost is fully clipped by the aperture, its centre on the sensor corresponds to the projection of the centre of the iris aperture. Since the available information is only the light direction at the entrance pupil, characterised by the incident angles θ, we must determine the corresponding point on the entrance pupil that projects onto the iris aperture centre.

For a given light direction, the ray on the entrance pupil that will give the centre of the ghost clipped by the aperture is computed as:

$$\mathbf{r}_{\text{entrance}} = \begin{pmatrix} -\theta \, \dfrac{M_a(0,1)}{M_a(0,0)} \\ \theta \end{pmatrix} \tag{5.3}$$

where $M_a(i,j)$ denotes the element in the i-th row and j-th column of M_a.

This comes from solving the linear constraint imposed by the transformation matrix Ma that ensures the projected ray aligns with the centre of the aperture plane:

$$\begin{pmatrix} M_a(0,0) & M_a(0,1) \\ M_a(1,0) & M_a(1,1) \end{pmatrix} \begin{pmatrix} x \\ \theta \end{pmatrix} = \begin{pmatrix} 0 \\ \theta_a \end{pmatrix} \tag{5.4}$$

where x denotes the point at the entrance plane that will project onto the centre of the iris aperture.

Applying the propagation matrices M_a and then M_s to this ray yields the position of the ghost centre on the sensor.

To determine if the ghost shape on the sensor is clipped by the entrance pupil or the iris aperture, I compare the projected heights of the entrance pupil and iris aperture on the sensor. The smallest of the two will indicate which clipping is more prominent on the ghost shape.

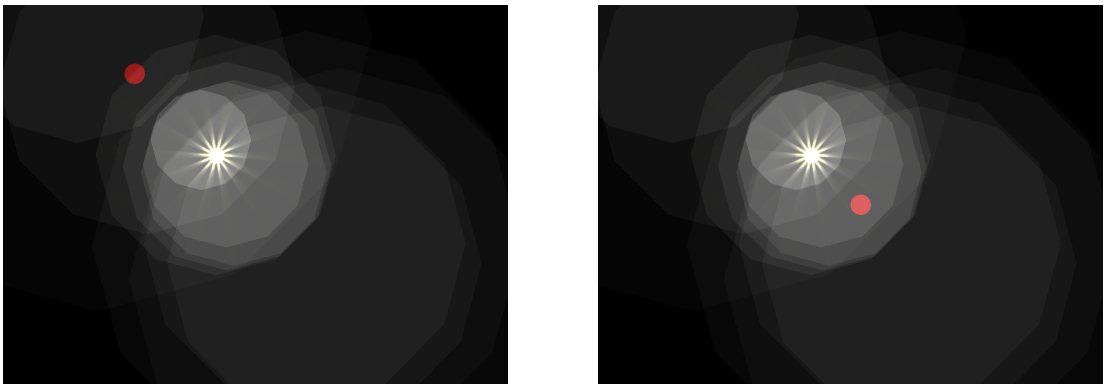
An example of annotating the position of ghosts is shown in Figure 5.12

5.5.2. Ghost Height

For annotating ghost heights, a multiplier value is kept that can be adjusted by dragging the mouse to minimise or expand. This multiplier is then applied to the projected quad point positions for visual feedback. An example of annotating the size of ghosts is shown in Figure 5.13.

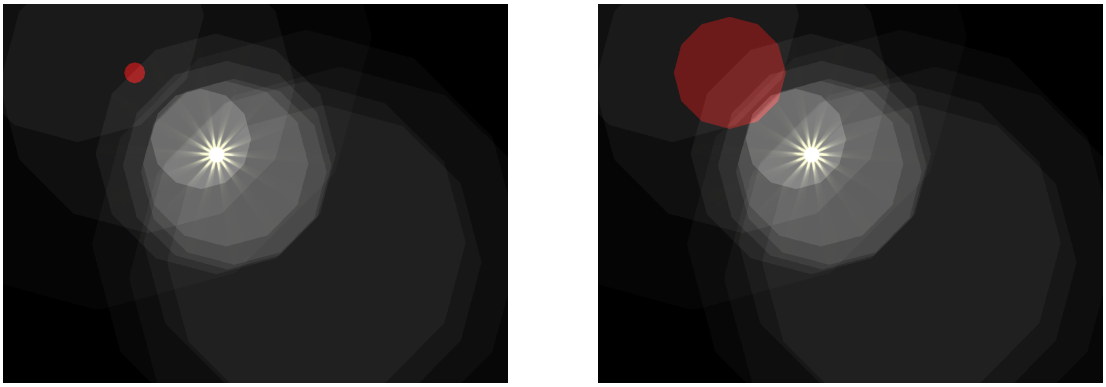
5.5.3. Ghost Colour

To annotate ghost colour, we override the original hue resulting from the anti-reflectivity coatings, as can be observed in Figure 5.14.



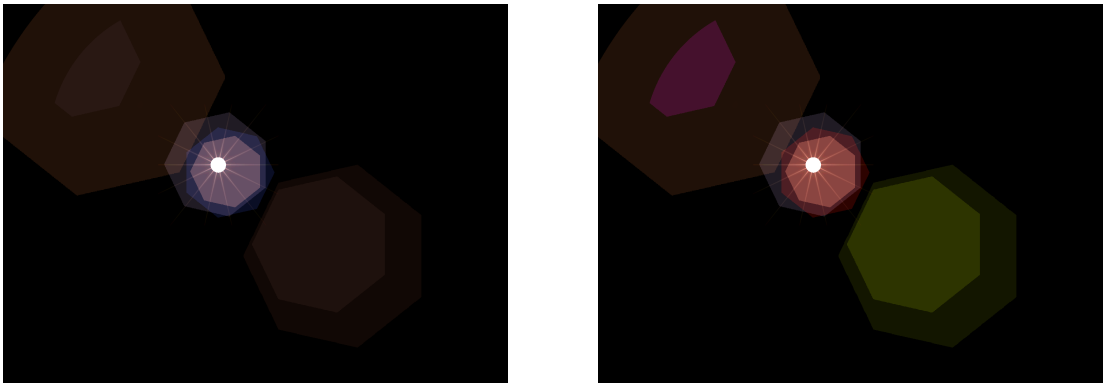
(a) Original **(b)** Annotated

Figure 5.12: Example of ghost position annotation. The annotated ghost is highlighted in red.



(a) Original **(b)** Annotated

Figure 5.13: Example of ghost height annotation. The annotated ghost is highlighted in red.



(a) Original **(b)** Annotated

Figure 5.14: Example of ghost colour annotation.

5.6. Solving the User Annotations with Evolutionary Algorithms

The following explains why and how the user annotations are solved using EAs.

5.6.1. Why use EAs?

We want to find the lens design that produces ghost flares closely matching the user's annotated targets. Each optical interface in the lens prescription contributes four parameters relevant to ghost rendering (thickness, refractive index, radius of curvature, and the coating's central wavelength). Additionally, the aperture's height and position introduce two more variables. Consequently, the optimisation problem involves $4N_{\text{interfaces}} + 2$ variables, making it a high-dimensional problem.

Each ghost is defined by a unique lens traversal path, resulting in a distinct propagation matrix. Therefore, the optimisation must account for a system of equations equal in number to the total number of ghosts. Additionally, the design parameters are interdependent: modifying a single variable, such as the thickness of an interface, influences the impact of other variables on the render. This strong coupling between variables adds further complexity to the search space.

Given these characteristics, evolutionary algorithms are a fitting solution. They are well-suited for black-box optimisation problems, can efficiently explore high-dimensional and non-linear search spaces, and are robust against local optima [11].

EAs iteratively evolve a population of candidate solutions to an optimisation problem. These candidate solutions, referred to as individuals, are defined by their decision vectors. In this case, an individual represents a candidate lens design, and its decision vector is an array that encodes all the lens prescription parameters needed to simulate a render and compute the error relative to user annotations.

5.6.2. Separation of Concern

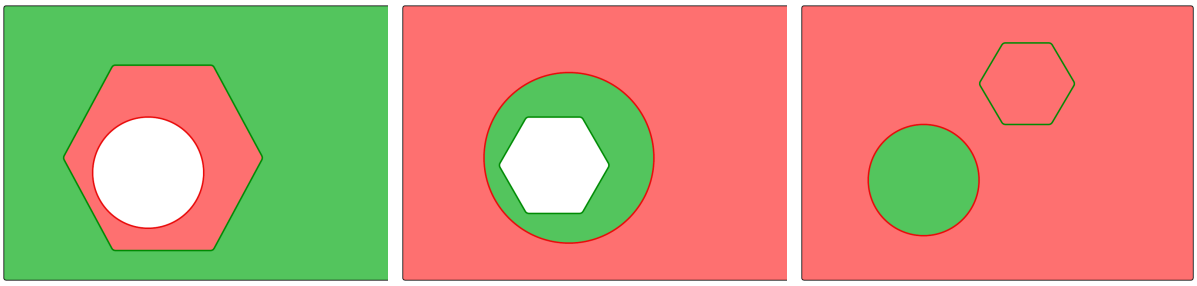
To manage the inherent complexity of the optimisation, the process is divided into two stages. The first stage focuses on achieving the correct positioning and sizing of the ghosts, while the second stage adjusts the coatings so that the colours match the annotations. This approach reduces problem dimensionality and thus complexity.

5.6.3. Location and Size Optimisation

Decision Vectors For the location and size optimisation, the decision vector of each individual comprises the aperture position, the aperture height, followed by the thickness, refractive index, and radius for each lens interface. These parameters provide the necessary information to construct the transfer matrices for every ghost and compute their position and heights. The entrance pupil height was omitted from the optimisation process as it can easily be adjusted afterwards by the user. Although it does contribute to ghost size via entrance plane clipping, removing it from the decision vector prevents the evolutionary algorithm from converging on solutions where ghost size is determined solely by entrance clipping, thereby neglecting the influence of the aperture's overall shape. This scenario is rarely encountered in practical lens designs, justifying its treatment as a post-optimisation parameter.

Initialization The EA process initialises a population of individuals with random decision vectors, each constrained within predefined physical limits as specified in Section 5.2. The population is organised as an archipelago consisting of 15 islands. This structure enables parallel execution of the EA on each island, while allowing periodic migration of individuals between islands. Each island is initialised with $15 * ||decision\_vector||$ individuals. This population size setup was found to be optimal to find the best results in the shortest amount of time (see Chapter 6). Given that the entrance pupil height is not optimised, it is set to a fixed value. I chose a big height (100mm) to minimise its clipping impact during optimisation. In each island, we also introduce the current lens prescription (unannotated). This will help the convergence process since the annotations might not significantly change the current flare render.

Fitness Function Before evaluating an individual, its decision vector is processed to ensure that it satisfies the physical constraints. Moreover, since we allow the aperture position to change, directly influencing the number of possible reflection pairs, individuals that lead to rendering an insufficient



(a) Ghost centre determined by the entrance pupil projection

(b) Ghost centre determined by the aperture projection

(c) Ghost hidden.

Figure 5.15: Three possible clipping scenarios, determining the ghosts' location and size. Colour code: white = visible part of the ghost, red = quad clipped by the entrance pupil, green = quad clipped by the aperture.

number of ghosts are discarded.

To assess an individual's fitness in approximating the target render, the corresponding transfer matrices are extracted from its decision vector. These matrices are then used to compute the new locations and sizes of each ghost.

Two cases are considered for determining a ghost's location. In the first case, if the quad edges of the ghost are entirely clipped by the entrance pupil, the ghost's centre is determined by projecting the centre of the entrance pupil onto the sensor, yielding a circular ghost shape. In the second case, if the quad edges are clipped by the aperture, the centre is defined by projecting the aperture's centre onto the sensor, and the ghost's shape is determined by the aperture sprite. To assign the correct case, the fitness function compares the projected sizes of the entrance height and the aperture height, which also provides the ghost height.

Furthermore, there is a scenario where a ghost may be completely hidden due to clipping by both the entrance pupil and the aperture. This occurs when the sensor projections of both heights are smaller than the distance between their centres. Figure 5.15 shows the three cases.

Once the location and size of each ghost are computed, the ghosts are sorted by increasing size and compared to the target render using the following error metric:

Algorithm 1 Location and Size Fitness Metric Computation

Input:
- `objective`: list of target ghost parameters
- `newRender`: list of computed ghost parameters

Output: Fitness value, f

$f \leftarrow 0.0$

Let $N \leftarrow \text{length}(\texttt{objective})$

for $i = 0$ to $N - 1$ **do**

  posError $\leftarrow ||\texttt{objective}[i].\text{centerPos} - \texttt{newRender}[i].\text{centerPos}||$

  sizeError $\leftarrow \texttt{objective}[i].\text{height} - \texttt{newRender}[i].\text{height}$

  $f \leftarrow f + (\text{posError})^2 + (\text{sizeError})^2$ ▷ Square errors to penalize larger deviations

end for

if $\text{length}(\texttt{newRender}) > N$ **then**

  for $i = N$ to $\text{length}(\texttt{newRender}) - 1$ **do**

    $f \leftarrow f + \dfrac{500}{\texttt{newRender}[i].\text{height}}$ ▷ Penalty for extra ghosts

  end for

end if

$f \leftarrow f/N$ ▷ Normalize fitness by the number of target ghosts

return f

The error terms are squared to account for the visual system's nonlinear sensitivity to deviations in both location and area. According to Knill and Pouget [21], the Bayesian brain framework suggests that spatial discrepancies become increasingly significant as their size grows, because the uncertainty in neural coding rises with larger errors. Similarly, Zhou et al. [44] demonstrate that the mechanisms linking perceived intensity to discrimination naturally lead to an error metric that increases quadratically. This means that minor deviations in position or area result in relatively small errors, while larger deviations are amplified in their impact, which is reflected in our fitness function.

Any ghost over the target count is penalised by a term inversely proportional to the extra ghost's height. As a ghost increases in size, its intensity decreases substantially, making its visual impact negligible.

The EA iteratively evolves the population to minimise this fitness function.

Choice of EA Several EAs suitable for real-valued, single-objective optimisation problems are considered in this study. We include Differential Evolution (DE [35]) and its self-adaptive variants (SADE 2006 [8], SADE 2011 [13], and DE1220 [14]) for their robustness and effective continuous search, with built-in parameter tuning. Particle Swarm Optimisation (PSO [18]) is selected for its rapid convergence, and Extended Ant Colony Optimisation (GACO [32]) is added for its technique to balance exploration and exploitation. These methods provide a comprehensive comparison of different approaches in continuous optimisation. Their performance is analysed in Chapter 6.

Evolutionary algorithms that rely on sampling distributions to generate new candidate solutions, such as Covariance Matrix Adaptation Evolution Strategy (CMA-ES), are excluded from this comparison. Due to the strict bounds and physical constraints of the problem domain, these algorithms exhibited noticeably degraded performance.

Stopping Criteria To reduce computation time, the EAs are run for a fixed number of generations, set at the start of population convergence. This trade-off prioritises faster execution over the marginal perceptual gains of extended optimisation. The optimal generation count is discussed in Chapter 6.

Retrieving the best solutions Upon completion of the evolutionary optimisation, rather than only extracting the single best solution, the tool presents the top five candidates. This approach is motivated by several factors. First, the best overall fitness solution may not adequately capture specific ghost artefacts that the user considers important. Second, although the solutions are evaluated under a particular light direction, their behaviour varies when the light conditions change. Finally, while the optimal solution might offer the closest match in terms of greyscale rendering, the application of coatings, a process dependent on the refractive indices, can modify the flare look, with some ghosts losing intensity through reflection losses. Providing a ranked set of candidates thus gives the user the flexibility to select the most visually suitable outcome for their specific requirements.

This method is possible in the archipelago, as the algorithms on each island may converge on different regions of the problem space.

5.6.4. Color Optimisation

Decision Vector The quarter-wave coating simulation calculates the reflectivity of the coating for a given wavelength using three parameters derived from the lens prescription: the refractive index of the current interface, the refractive index of the subsequent interface, and the wavelength λ_0 for which the coating is optimised under normal incidence. Since the refractive indices are fixed from the earlier optimisation for ghost location and size, only λ_0 is optimised in this phase to achieve the desired colours. Optimising the refractive indices would also change the location and size of ghosts. Consequently, the decision vector in this step is composed solely of the λ_0 values for every interface coating in the system. The search space is bound within the visible spectrum range we selected (380-750 nm).

When using custom coatings, the decision vector contains the refractive index and thickness of each coating. This leads to a decision vector that is twice the size of that for the quarter-wave case. As stated in Section 5.4, the thickness ranges from 25 nm to 750 nm and the refractive index from 1.38 to 1.9.

Initialization The colour optimisation process is initialised similarly to the position and size optimisation process. The population is initialised randomly and includes the base coating parameter set for the unannotated colours.

Fitness Function Each individual proposes a new set of coating parameters for which we will have to shoot a ray through each ghost path to recompute the colours. To evaluate the ghost colours from the new parameter set against the target render, the error metric is calculated by adding up the norms of the normalised RGB difference vectors for each ghost:

Algorithm 2 Colour Fitness Metric Computation

1: **Input:**
   - `targetRGB`: list of target ghost RGB vectors
   - `renderRGB`: list of computed ghost RGB vectors
2: **Output:** fitness value, f
3: $f \leftarrow 0.0$
4: Let $N \leftarrow$ length$(\texttt{targetRGB})$
5: **for** $i = 0$ to $N - 1$ **do**
6: $\quad \vec{d} \leftarrow \texttt{normalise(targetRGB}[i]) - \texttt{normalise(renderRGB}[i])$
7: $\quad f \leftarrow f + \|\vec{d}\|$ ▷ Euclidean distance
8: **end for**
9: **return** f

Choice of EA The same evolutionary algorithms as for location and size optimisation are considered. The number of variables and search space for coating optimisation is much smaller, allowing the EAs to converge much faster. See Chapter 6 for the performance results.

Stopping Criteria We use the same stopping criteria principle as for optimising location and size.

Champion Solution For coating optimisation, returning the top 5 would not present the same advantages as optimising location and size. Returning the best solution is sufficient.

5.7. Making a Lens from Scratch

Subsection 5.6 addresses the application of annotations on an existing lens system. However, when no starting system is available, only a few modifications to the optimisation process are necessary to establish an initial configuration. In this case, instead of rendering quads through transfer matrices, the quads are directly rendered in the view. A texture lookup on the aperture provides the shape for the quads. Their sizes are initialised to a default value, which users can later refine using annotations. The ghost locations are similarly set, with the additional constraint that they lie along the line passing through the light source and the optical centre. Since the units for size and position are consistent, this target render can be passed directly to the optimisation process.

One variable that requires careful determination to define the decision vector length is the number of interfaces in the system. While various configurations are possible, in practice, the iris aperture is typically located near the centre of the system. Therefore, the minimum number of interfaces necessary to generate the required number of ghosts G, with the aperture positioned centrally, is computed as follows:

$$n_{\min} = \min\left\{ n \in \mathbb{N} : n \geq 4 \quad \text{and} \quad \binom{\lfloor \frac{n-1}{2} \rfloor}{2} + \binom{\lceil \frac{n-1}{2} \rceil}{2} \geq G \right\} \tag{5.5}$$

Although this approach may sometimes produce additional ghosts, these extras have low light intensity and a negligible visual impact. The fitness function sorts the ghosts based on size before comparison, ensuring that the smaller ghosts adhere to the target render and the extra ghosts correspond to the largest ones. These disperse the same amount of light over a larger area, reducing their intensity.

(a) 0 ghosts. **(b)** 3 ghosts. **(c)** 6 ghosts.

Figure 5.16: Building a lens from scratch

6 Results

This chapter evaluates the performance of various evolutionary algorithms in solving the target annotations. The entire project was developed in C++ and uses OpenGL for rendering. All results were computed using an AMD Ryzen 7 7800X3D and an AMD Radeon RX 6700 XT.

6.1. Implementation

The UI was built using Dear ImGui [10], a library with many widgets for user input, such as sliders and colour pickers. I used Pagmo 2 [2] to implement the EAs. The Pagmo framework supports user-defined problems and facilitates running different EAs on them. Pagmo also supports parallelisation to accelerate the optimisation.

6.2. Metrics

To compare the resulting render from the optimisation process to the target render, we will employ 2 common metrics in image comparison [16]:

Peak Signal-to-Noise Ratio PSNR quantifies image quality by measuring the ratio between the maximum signal power and the noise that distorts the image, expressed in decibels (dB). Higher PSNR values indicate that the test image is closer to the reference image. For a reference image f and a test image g, the PSNR is defined as:

$$\text{PSNR}(f,g) = 10\log_{10}\left(\frac{\text{MAX}_I^2}{\text{MSE}(f,g)}\right) \tag{6.1}$$

where MAX_I is the maximum possible pixel value of the image, and the Mean Squared Error (MSE) is given by:

$$\text{MSE}(f,g) = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}(f_{ij}-g_{ij})^2 \tag{6.2}$$

Structural Similarity Index Measure SSIM assesses image quality by comparing structural information, luminance, and contrast between the two images. SSIM values range from -1 to 1, with values closer to 1 signifying a high degree of similarity. For a reference image f and a test image g, the SSIM is defined as:

$$\text{SSIM}(f,g) = \frac{(2\mu_f\mu_g + C_1)(2\sigma_{fg} + C_2)}{(\mu_f^2 + \mu_g^2 + C_1)(\sigma_f^2 + \sigma_g^2 + C_2)} \tag{6.3}$$

where:

- μ_x and μ_y denote the mean intensities of images x and y,
- σ_x^2 and σ_y^2 represent the variances,
- σ_{xy} is the covariance between x and y,
- C_1 and C_2 are small constants to avoid instability.

6.3. Benchmark Tests

A benchmark test is necessary to assess the performance of various EAs on the optimisation problem. One challenge in using a user-annotated render for evaluation is that errors may originate either from algorithmic inefficiencies or from the annotations themselves, potentially leading to a target render that no physically viable lens can reproduce. To resolve this issue, the base lens is excluded from the population, and the optimisation algorithm is instead provided with an unannotated render of an existing camera lens as the target. This approach ensures that an exact solution to the optimisation problem exists. Utilising this method, two benchmark tests were developed using the Heliar Tronnier and Canon lenses. The lens prescription for the Heliar Tronnier was presented in Table 2.1, and the prescription for the Canon can be found in Appendix A. In the ideal scenario, the algorithm can recover the prescriptions of the test lenses.

6.4. Ghost Location and Size Optimisation on a Simple Lens

Heliar Tronnier - Small Angle As an initial test, the suite of algorithms described in Subsection 5.6.3 was applied to the 8-interface Heliar Tronnier lens configuration, which generates 13 ghost images. Each algorithm was executed 10 times on the same set of random seeds and initial populations to ensure consistency and eliminate the influence of stochastic variability. The incident light angle was fixed at $5.45°$ for both the pitch and yaw. As illustrated in Figure 6.1, both PSO and the 2006 variant of SADE outperform the other algorithms in terms of final average fitness values. The GACO algorithm ranked third in final average fitness, but suffers a substantial computational overhead due to the expensive estimation of its internal parameters. Consequently, the variant of SADE, DE1220, is considered more effective, as it achieves comparable fitness scores in a shorter amount of time. In contrast, DE exhibited the weakest performance, primarily due to its static parameter configuration, which lacks the adaptive capabilities present in SADE. Further analysis, presented in Appendix C, explores the performance of DE under various combinations of difference vector weights, crossover probabilities, and mutation strategies. These results confirm that even with parameter tuning, DE remains less effective than PSO and SADE for this problem domain.

Heliar Tronnier - Large Angle Next, the top 3 algorithms of the previous test were pitted against each other on the same lens test, but this time with a bigger light direction angle of $13.40°$. A bigger light direction emphasises the lens's characteristic spread of the ghosts more, increasing the specificity of the ghost location factor, making the optimisation problem harder. On a smaller angle, more alternative lens constructions can approximate the render well by having the same ghost sizes. On this test, the three algorithms perform similarly in terms of final average fitness, but DE1220 falls behind in terms of convergence speed, as can be observed in Figure 6.2.

It is also noticeable that the average champion fitness has increased substantially, due to the nature of the harder problem. With these default settings, the algorithms have converged on a local optimum, no longer exploring new regions but rather exploiting the current ones. Nevertheless, these sub-optimal solutions might be good enough for the goals of the user. Therefore we asses the similarity of the resulting renders with the target render using the PSNR and SSIM metrics. Figures 6.3 & 6.4 show the original and the optimisation result side-by-side for comparison. The error metrics demonstrate that the overall flare structure and signature are well conserved, proving the optimisation results are good enough for our application. Although the differences are noticeable when looking closely, the bigger picture is similar.

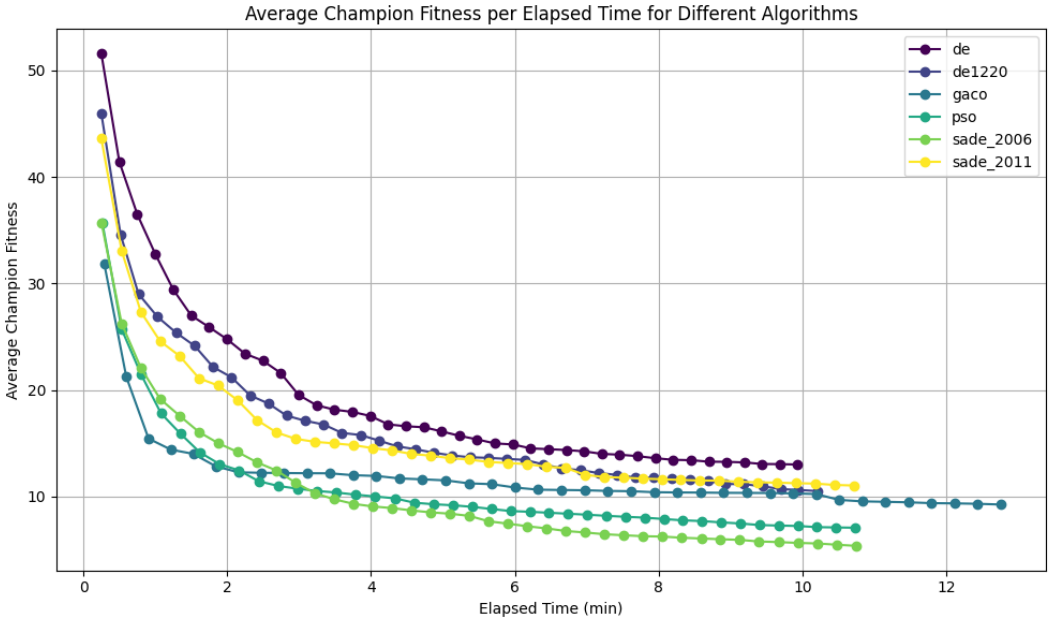


Figure 6.1: Algorithm performance on the Heliar Tronnier test with a $5.45°$ light angle for both yaw and pitch. Each algorithm is run 10 times for 4000 generations, with 15 islands and $15 * ||decisionVector||$ individuals per island.

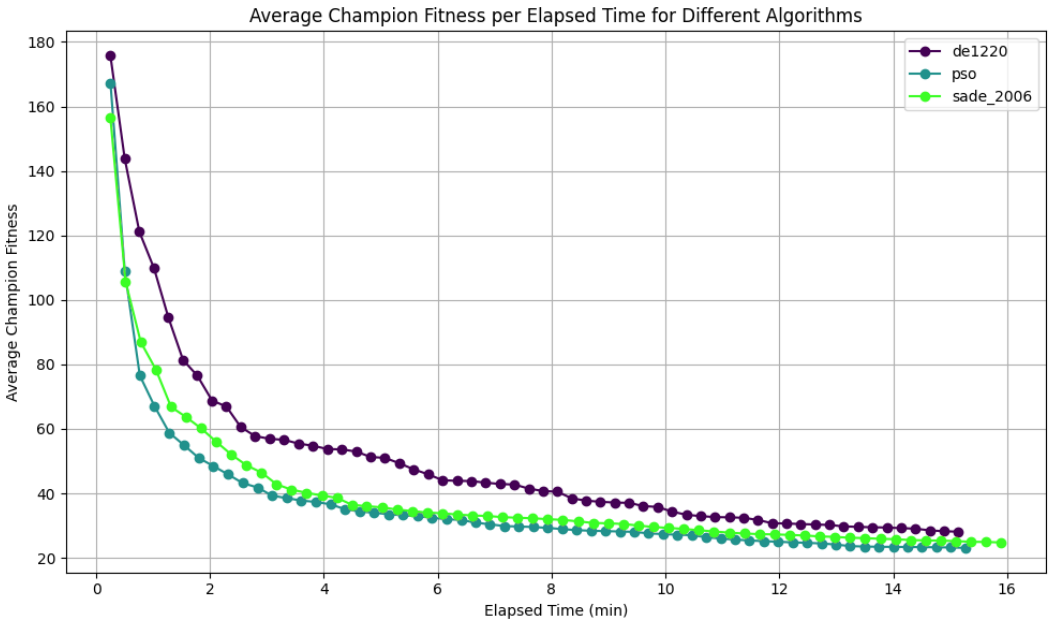
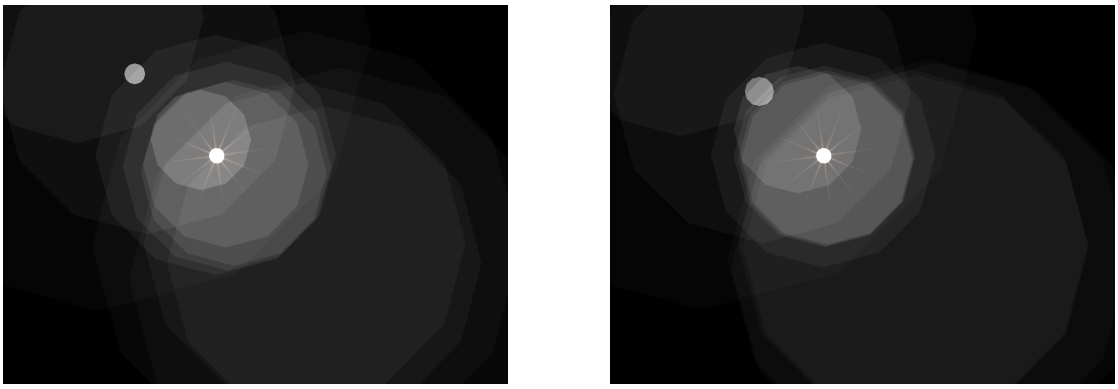
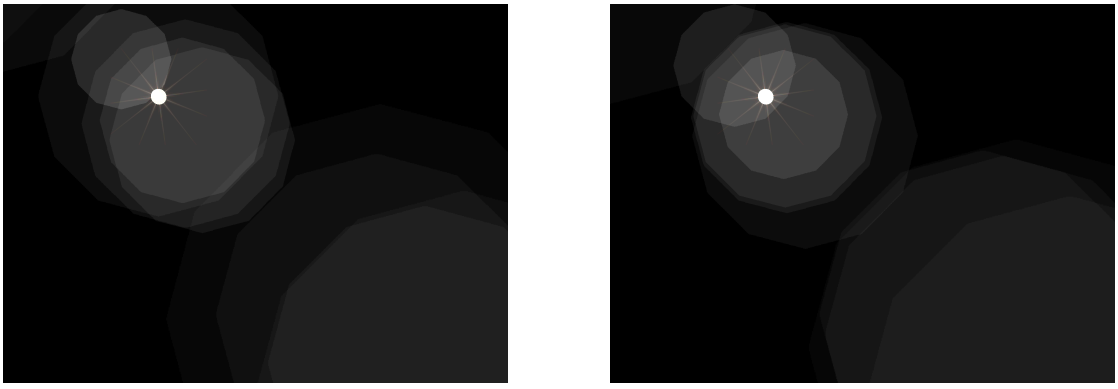


Figure 6.2: Algorithm performance on the Heliar Tronnier test with a $13.40°$ light angle for both yaw and pitch. Each algorithm is run 10 times for 6000 generations, with 15 islands and $15 * ||decisionVector||$ individuals per island.



(a) Target **(b)** Result

Figure 6.3: Result of PSO on the Heliar Tronnier test with a $5.45°$ light angle for both yaw and pitch. PSNR=25.89dB, SSIM=0.9146. (10min)



(a) Target **(b)** Result

Figure 6.4: Result of PSO on the Heliar Tronnier test with a $13.40°$ light angle for both yaw and pitch. PSNR=30.38dB, SSIM=0.8512. (10min)

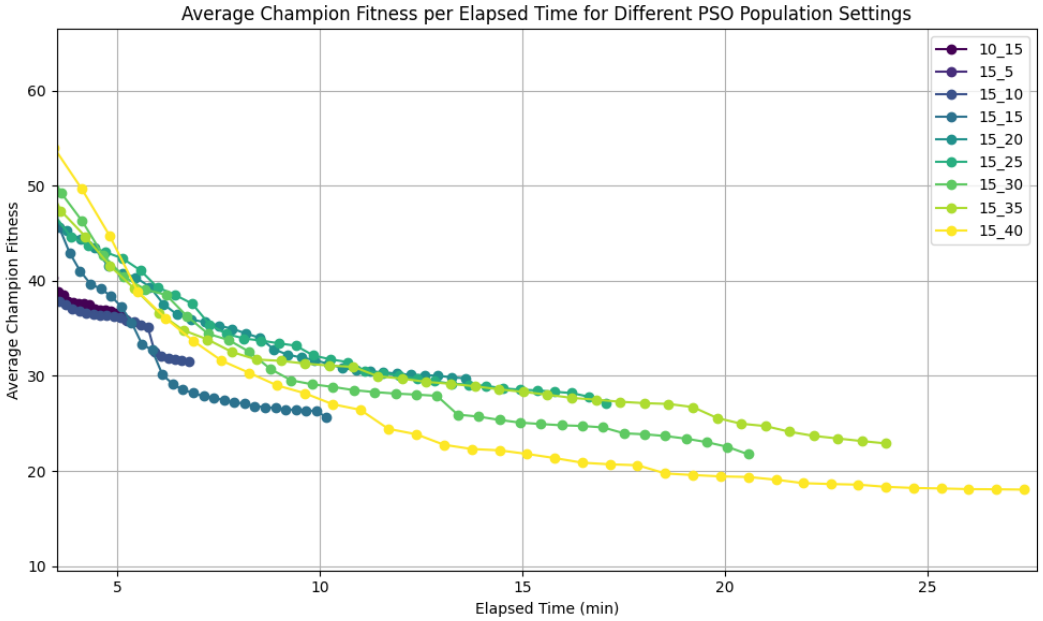


Figure 6.5: Effect of population settings on PSO performance. Helair Tronnier test with $13.40°$ angle for both yaw and pitch. Each variant is run 5 times for 4000 generations. (X_Y signifies a population of X islands with $Y * ||decisionVector||$ individuals)

6.5. Population Size

An important factor influencing algorithmic performance is the population size, which directly affects both the initial diversity and the algorithm's exploratory capacity. In this thesis, population size is defined by two parameters: the number of islands in the archipelago and the number of individuals per island. Figure 6.5 illustrates the impact of varying these parameters on the performance of the best-performing algorithm, PSO. The number of islands was fixed at 15, as this configuration fully utilises the available CPU capacity without incurring parallelisation overhead. Increasing the number of islands beyond this threshold would lead to slower convergence due to scheduling overhead, while fewer islands would underutilise computational resources. The results indicate that increasing the number of individuals per island improves the average champion fitness, at the cost of increased computational time. This trade-off is expected, as larger populations require more evaluations per generation, thereby increasing the computational burden. Considering our application, artists designing lens flares, a maximum optimisation time of 10 minutes for simple lenses is acceptable. Within this constraint, a population size of $15 * ||decisionVector||$ individuals per island offers the best balance between solution quality and computational efficiency.

6.6. Ghost Location and Size Optimisation on a Complex Lens

The second benchmark test evaluates algorithm performance on a significantly more complex optical system: the Canon lens, which comprises 27 interfaces and generates 171 ghost images. This test serves to assess the scalability and robustness of the two best-performing algorithms, SADE (2006) and PSO, under high-dimensional and computationally intensive conditions. As shown in Figure 6.6, PSO demonstrates a clear advantage in convergence speed. In contrast, SADE's adaptive parameter tuning (specifically for crossover probability and difference vector weight) introduces additional computational overhead, which becomes more pronounced in this complex scenario.

The increased problem complexity also leads to a noticeable decline in average champion fitness, as reflected in the quantitative metrics and the rendered output shown in Figure 6.7. While optimisation successfully captures the dense cluster of central ghosts and several larger, subtler artefacts, some ghost elements critical to the lens flare's visual signature are lost. Consequently, although the similarity metrics suggest a reasonable match, the perceptual quality is degraded compared to the simpler Heliar

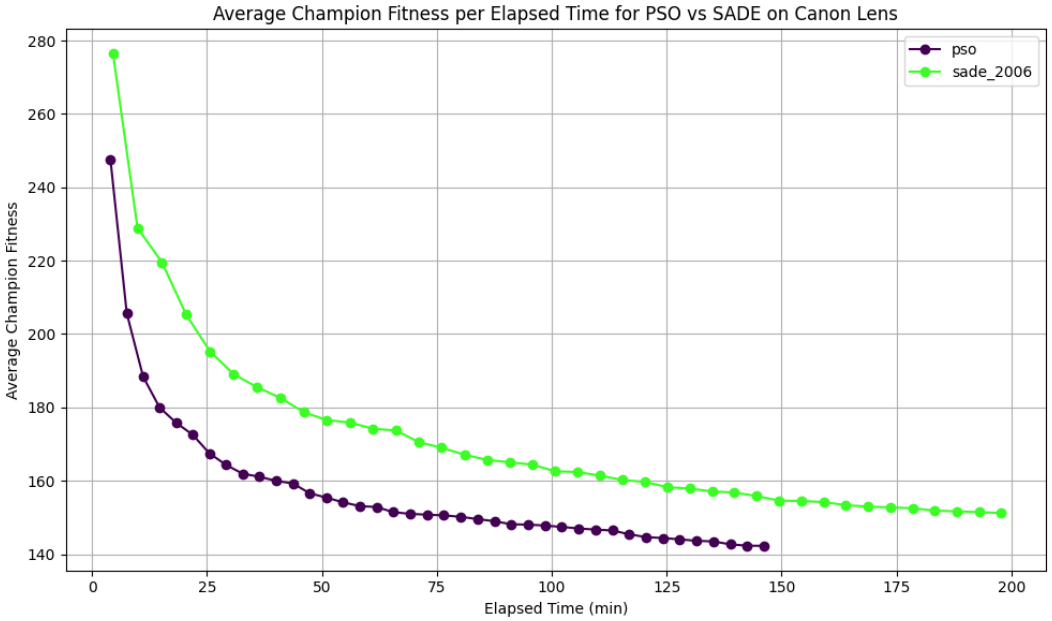


Figure 6.6: PSO vs SADE performance on the Canon test with $5.45°$ light direction angle for both yaw and pitch. Each algorithm is run 10 times for 4000 generations, with 15 islands and $15 * ||decisionVector||$.

Tronnier case. The acceptability of the final result is therefore subjective and may vary depending on artistic intent and application context.

6.7. Color Optimisation

Similar to the evaluation approach used for ghost location and size optimisation, the performance of EAs in coating optimisation is assessed by tasking them with recovering a known coating configuration. This is achieved by removing the current lens coating settings from the population and providing colour annotations generated from a specific, predefined coating setup, which the algorithms must then attempt to replicate. This method ensures that the evaluation is based on a physically accurate reference, thereby avoiding potential errors introduced by the user's annotations.

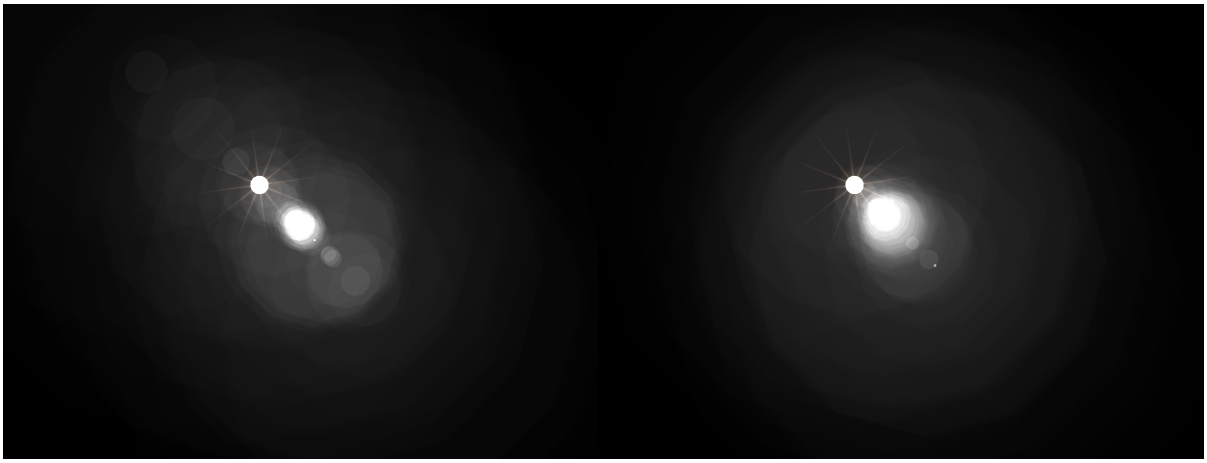
6.7.1. Quarter-Wave Coatings

For quarter-wave coating optimisation, the performance of SADE (2006) and PSO on the Heliar Tronnier lens is presented in Figure 6.8. Compared to the location and size optimisation task, both algorithms converge significantly faster, primarily because the fitness function no longer requires rebuilding the ray transfer matrices. However, the evaluation remains computationally intensive, as it involves tracing a ray through the system for each ghost to determine its colour contribution.

In the relatively simple Heliar Tronnier case, both algorithms successfully reverse-engineered the exact coating parameter set, with only negligible deviations. When applied to the more complex Canon lens system, a clearer distinction in algorithm performance emerges. Contrary to the results observed in the location and size optimisation, SADE outperforms PSO in this context, largely due to its ability to avoid premature convergence to local optima. This performance difference is visualised in Figure 6.9. The visual similarity between the target and optimised renders for both lenses is demonstrated in Figures 6.10 and 6.11, confirming the effectiveness of the optimisation process in reproducing the desired spectral characteristics.

6.7.2. Custom Coatings

Thanks to the higher colour expressiveness of custom coatings, the physically inaccurate user annotations can be approximated better than with quarter-wave coatings. The performance of the EAs in recovering the coating parameters with custom thickness and refractive indices is similar to the case



(a) Target **(b)** Result

Figure 6.7: Result of PSO on the Canon test. PSNR=27.20dB , SSIM=0.8898

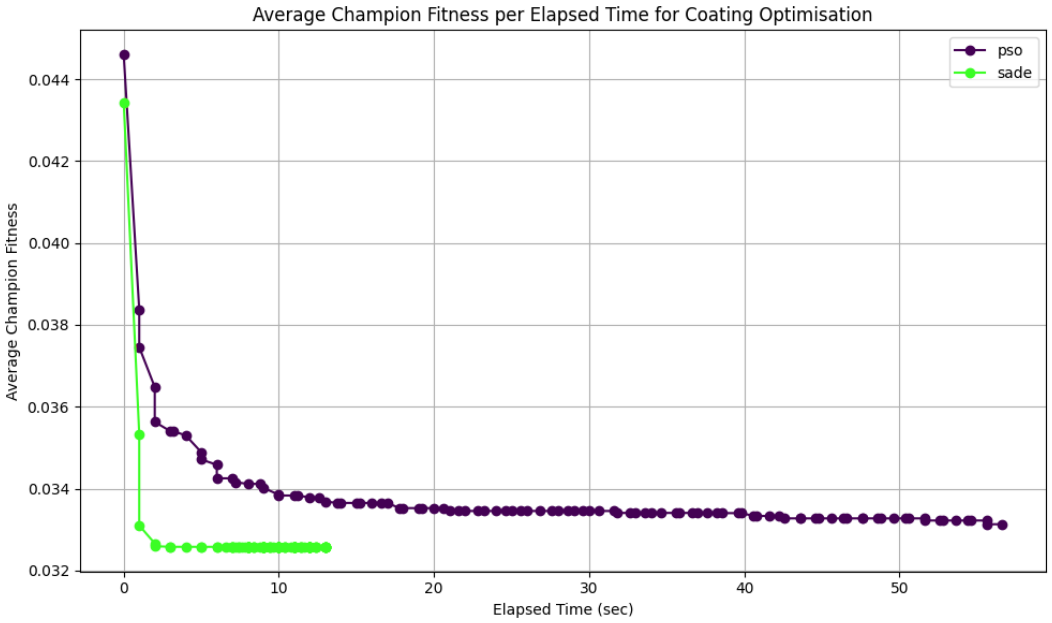


Figure 6.8: PSO vs SADE performance on the Heliar Tronner coating test. Each algorithm is run 5 times for 1500 generations, with 15 islands and $15 * ||decisionVector||$.

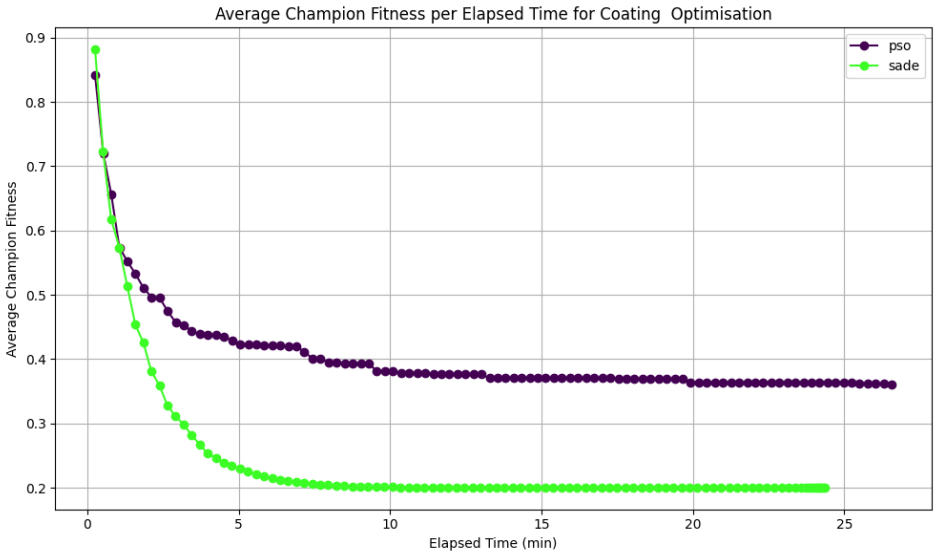
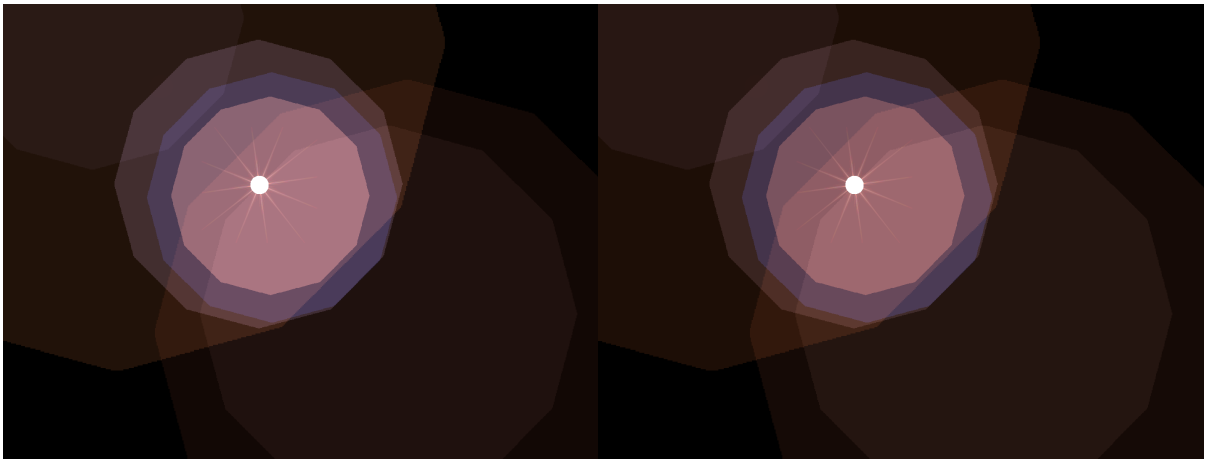
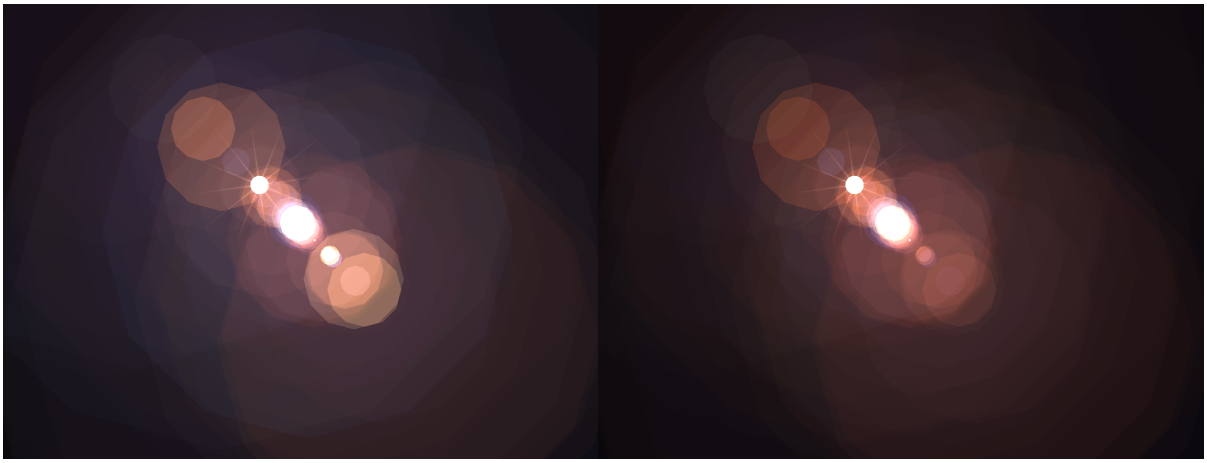


Figure 6.9: PSO vs SADE performance on the Canon coating test. Each algorithm is run for 500 generations, with 15 islands and $15 * ||decisionVector||$.



(a) Target **(b)** Result

Figure 6.10: Coating result of SADE on the Heliar Tronner test. PSNR=31.82dB.



(a) Target **(b)** Result

Figure 6.11: Coating result of SADE on the Canon test. PSNR=24.44dB.

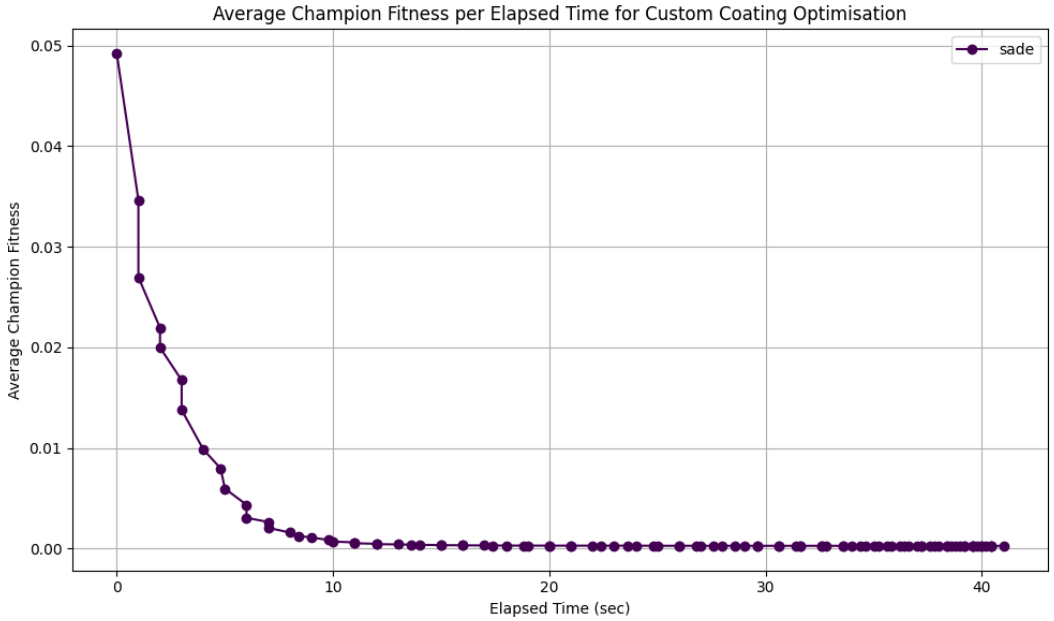


Figure 6.12: SADE performance on the Heliar Tronner custom coating test. The algorithm is run 5 times for 1500 generations, with 15 islands and $15 * ||decisionVector||$.

for quarter-wave coatings, as can be seen in Figure 6.12 & 6.13. However, the process takes more time as the decision vector is twice as large.

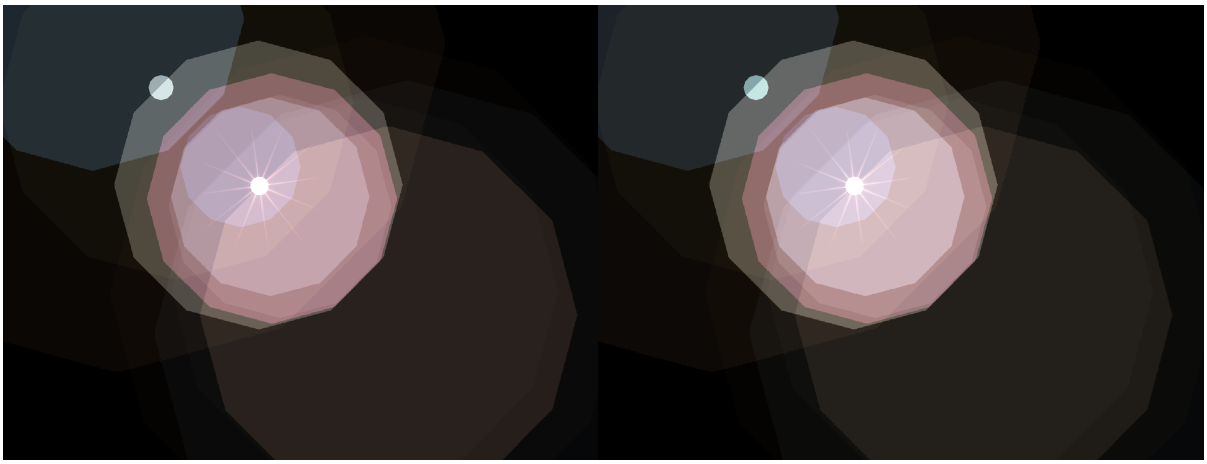
6.8. Making a Lens from Scratch

As a supplementary evaluation, the capability of PSO to synthesise a lens system from scratch, based solely on user-provided annotations, was assessed. Three test cases were considered, targeting configurations with 5, 10, and 15 ghost images, as shown in Figures 6.14, 6.15, and 6.16, respectively. The annotations were generated arbitrarily to simulate a user aiming to replicate a specific lens flare signature. While similarity metrics such as PSNR and SSIM tend to decline with an increasing number of ghosts, the overall visual resemblance remains consistent. As previously discussed, the lower similarity scores are attributable to the physically unrealistic nature of the annotations, rather than limitations in the algorithm itself. In practice, further refinement through the user interface can be employed to enhance specific features that were not fully captured during the optimisation process.

6.9. GPU Parallelisation of the Fitness Evaluation

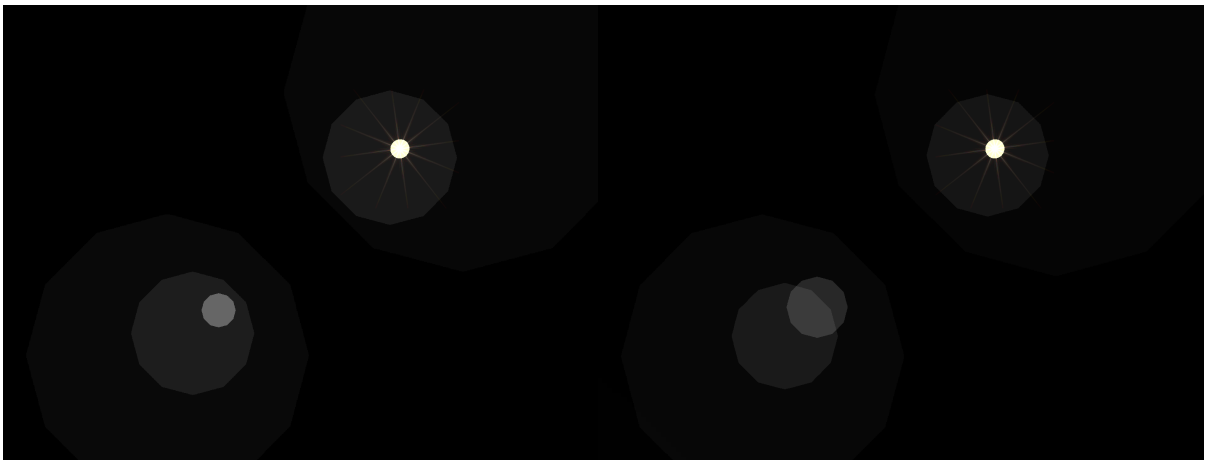
Fitness evaluation is a computationally intensive component of the EA process. Leveraging the GPU can significantly accelerate this step by exploiting its massive parallel processing capabilities. For PSO, the algorithm first updates the velocities and positions of all particles. Once the new positions are determined, the fitness of each particle can be evaluated independently. This independence makes the fitness evaluation suitable for parallel execution on a GPU.

With this acceleration, we replace the archipelago structure with a single population pool, enhancing diversity and knowledge within the PSO instance. This would have incurred heavy computation costs at the fitness evaluation step on the CPU. Figures 6.17 & 6.18 compare PSO performance with CPU vs GPU fitness evaluation on the Heliar Tronnier and Canon lenses, respectively. The GPU variant uses a population of $500 * ||decisionVector||$, slightly more than twice that of the CPU variant. The GPU version converges noticeably faster to somewhat better champion fitness.



(a) Target **(b)** Result

Figure 6.13: Custom coating result of SADE on the Heliar Tronner test. PSNR=31.64dB.

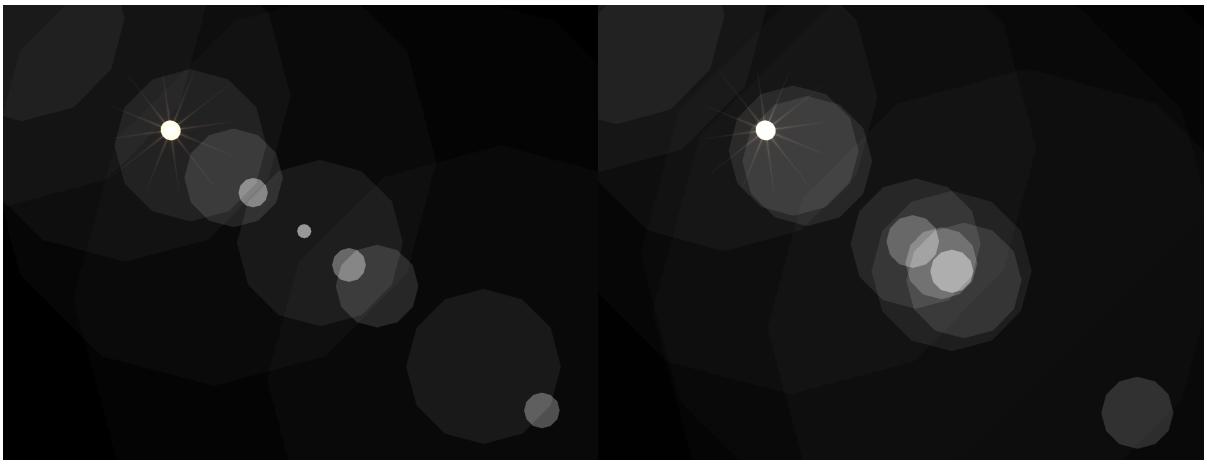


(a) Target **(b)** Result

Figure 6.14: Example result of building a lens from scratch with 5 ghosts. PSNR=34.85dB, SSIM=0.9582.

(a) Target **(b)** Result

Figure 6.15: Example result of building a lens from scratch with 10 ghosts. PSNR=26.08dB, SSIM=0.8798.



(a) Target **(b)** Result

Figure 6.16: Example result of building a lens from scratch with 15 ghosts. PSNR=23.77dB, SSIM=0.8185.

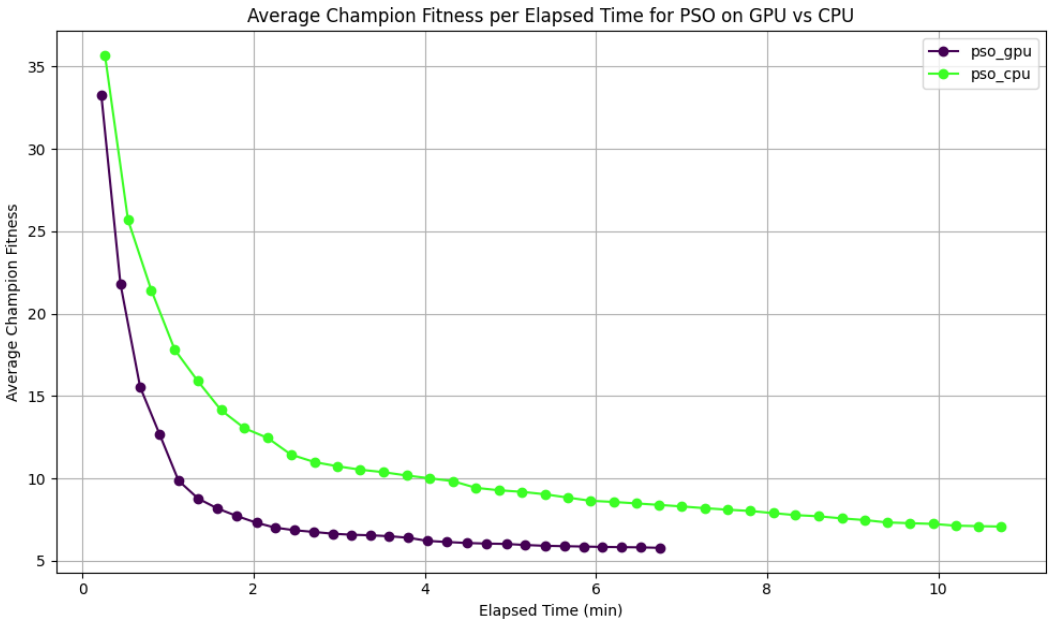


Figure 6.17: Performance comparison of fitness evaluation on CPU vs GPU with PSO on the Heliar Tronnier test. The algorithm is run 10 times for 4000 generations, with 15 islands and $15 * ||decisionVector||$ on the CPU and 10 times for 6000 generations, with a population of $500 * ||decisionVector||$ on the GPU.

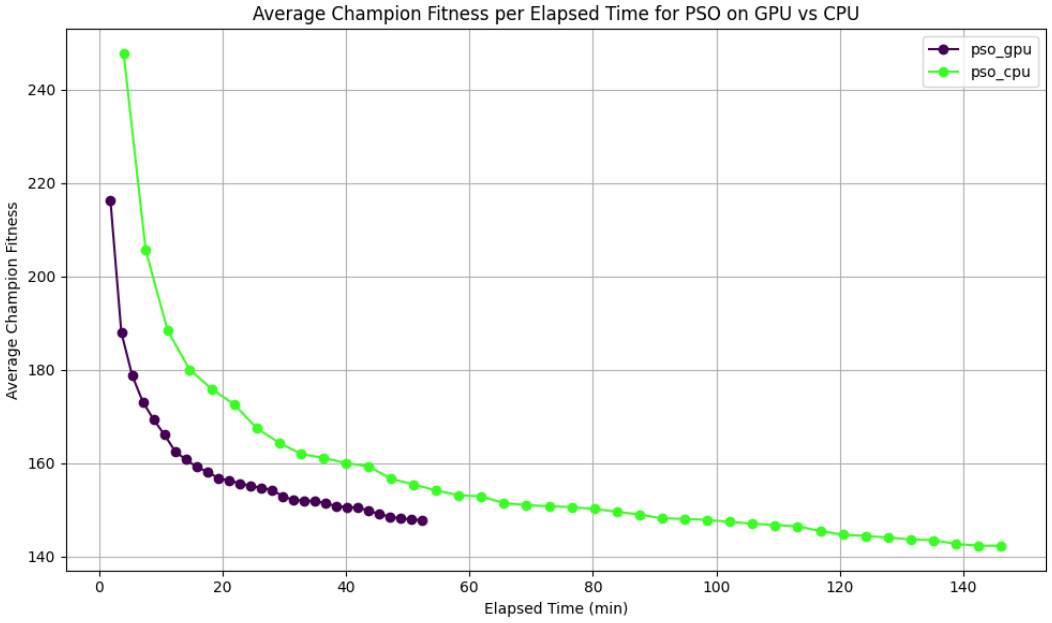


Figure 6.18: Performance comparison of fitness evaluation on CPU vs GPU with PSO on the Canon test. The algorithm is run 10 times for 4000 generations, with 15 islands and $15 * ||decisionVector||$ on the CPU and 10 times for 4000 generations, with a population of $500 * ||decisionVector||$ on the GPU.

7

Discussion & Future Works

This chapter highlights the proposed method's strengths and limitations. It also outlines potential extensions and improvements for future work.

7.1. Discussion

In ghost size and location optimisation, the evolutionary algorithm approach proves effective in approximating user-defined annotations on simpler lens systems (around eight interfaces). The exact lens prescription used to generate the annotations could not be fully recovered, primarily due to the complexity of the solution space and the high density of local optima. Yet, the overall flare signature is successfully captured. As the lens system complexity increases, the reliability and visual fidelity of the results diminish. In such cases, the additional tools designed, such as ghost editing through direct parameter manipulation, enable further refinement of the EA lens configuration. This allows users to emphasise important features that align with their creative intent.

A practical trade-off between solution quality and computational efficiency was achieved. This results in optimisation times ranging from 5 to 10 minutes for simpler lens systems, while more complex configurations require approximately one hour to converge. Through GPU parallelisation, these convergence times are significantly reduced. The extended computation times observed during optimisation can be attributed to two primary factors. First, the fitness function is computationally expensive, as it requires reconstructing the ray propagation matrices from the decision vector and simulating a render to extract ghost characteristics such as size and centre location. Second, the evolutionary algorithm's efficiency contributes to the overall runtime, as it performs numerous generations to converge on a satisfactory solution. In some cases, additional overhead is introduced by algorithms that dynamically adjust internal parameters during the optimisation process.

In the context of ghost colour optimisation, evolutionary algorithms demonstrated higher effectiveness. For simpler lens configurations, the algorithms recovered the original coating parameters used to generate the render targets. In more complex scenarios, while the recovered parameters were not exact, the resulting renders remained visually similar to the reference, indicating a strong approximation capability.

7.2. Future Works

Exploiting parameter linkage Rather than treating lens optimisation as a black-box problem, it can be reframed as a grey-box problem by leveraging structural knowledge of the system. Specifically, we know which parameters in the decision vector correspond to the same lens interface. By constructing linkage sets based on this information, the evolutionary algorithm can better preserve beneficial parameter groupings. This approach reduces the likelihood of disrupting well-performing interface configurations during variation. RV-GOMEA is a prominent example of an evolutionary algorithm that utilises linkage sets. It also supports partial evaluations, allowing only the affected components of the objective function to be recomputed when a subset of parameters changes, significantly improving computational

efficiency [7].

Using Interface Building Blocks In the current approach, each lens interface is optimized by freely adjusting its refractive index, radius, and thickness within predefined physical bounds and constraints. If the user were to specify a set of materials to use in the lens design, the search space can be significantly reduced. This constraint-driven reduction not only improves computational efficiency but also aligns the optimisation process more closely with practical manufacturing considerations. This approach is particularly relevant for industrial applications, where manufacturers may wish to explore lens configurations that produce specific flares using a known set of materials. Similarly, if the spatial layout is predetermined but the materials are unknown, the optimisation can focus solely on finding the necessary material properties.

Reverse-Engineering the Lens Construction from Real Flare Images In the ideal case where the optimisation process is able to recover the exact lens prescription given input flares from a real lens, the methodology can be extended to reverse-engineer the unknown internal structure of existing lenses. Flare features such as ghost locations, sizes, and colours can be extracted from images captured through the lens in a bright environment. These extracted features, along with the incoming light direction, can then serve as input for the optimisation algorithm, enabling the reconstruction of the underlying lens structure.

Different Physically-Accurate Lens Flare Rendering Techniques A natural extension of this work is to apply the methodologies developed in this thesis to other physically accurate lens flare rendering techniques that also rely on a lens prescription. By adapting the optimisation framework to alternative rendering models, the approach could be generalized to support a broader range of flare simulations.

8

Conclusion

This thesis presented a novel approach to lens design optimisation for physically accurate flare rendering. The research focused on simple and complex flare editing without relying on lens design expertise.

The work began by analysing the origins of flare characteristics in a physically accurate real-time lens flare renderer. This analysis linked specific flare properties to parameters in the lens prescription, enabling methods for simple flare editing through direct parameter manipulation.

The thesis introduced a two-stage optimisation pipeline using evolutionary algorithms for more complex modifications and creating entirely new lens systems. These algorithms were shown to be effective in solving the high-dimensional, non-linear lens design optimisation problem for a set of desired flare properties. Benchmark tests demonstrated that the system could approximate the renders of real lens prescriptions and produce visually convincing results, even when synthesising lenses from scratch.

The results confirmed that Particle Swarm Optimisation (PSO) and Self-Adaptive Differential Evolution (SADE) are particularly well-suited for this task. PSO excelled in optimising ghost size and location, and SADE offered near-exact results for coating optimisation.

In conclusion, this thesis contributes to the accessibility of physically accurate lens flare rendering by introducing lens design and optimisation techniques that achieve desired flare effects without requiring expertise in optics.

References

[1] Schott AG. *Interactive Abbe Diagram*. https://www.schott.com/en-gb/special-selection-tools/interactive-abbe-diagram. Accessed: 09 May 2025.

[2] Francesco Biscani and Dario Izzo. “A parallel global multiobjective framework for optimization: pagmo”. In: *Journal of Open Source Software* 5.53 (2020), p. 2338. DOI: 10.21105/joss.02338. URL: https://doi.org/10.21105/joss.02338.

[3] Alethea Blackler and Vesna Popovic. *Towards intuitive interaction theory*. 2015.

[4] Alethea Blackler, Vesna Popovic, and Doug Mahar. “Investigating users’ intuitive interaction with complex artefacts”. In: *Applied ergonomics* 41.1 (2010), pp. 72–92.

[5] Andrea Bodonyi, István Csoba, and Roland Kunkli. “Real-time ray transfer for lens flare rendering using sparse polynomials”. In: *The Visual Computer* (2024), pp. 1–18.

[6] Andrea Bodonyi and Roland Kunkli. “Efficient tile-based rendering of lens flare ghosts”. In: *Computers & Graphics* 115 (2023), pp. 472–483.

[7] Anton Bouter et al. “Exploiting linkage information in real-valued optimization with the real-valued gene-pool optimal mixing evolutionary algorithm”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017, pp. 705–712.

[8] Janez Brest et al. “Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems”. In: *IEEE transactions on evolutionary computation* 10.6 (2006), pp. 646–657.

[9] István Coba. “OpenLensFlare: an open-source, lens flare designing and rendering framework”. In: (2017).

[10] O. Cornut. *Dear ImGui (Version 1.86)*. https://github.com/ocornut/imgui. 2021.

[11] Judith Echevarrieta, Etor Arza, and Aritz Pérez. “Speeding-up Evolutionary Algorithms to Solve Black-Box Optimization Problems”. In: *arXiv preprint arXiv:2309.13349* (2023). URL: https://arxiv.org/abs/2309.13349.

[12] Agoston E Eiben et al. “What is an evolutionary algorithm?” In: *Introduction to evolutionary computing* (2015), pp. 25–48.

[13] Saber M Elsayed, Ruhul A Sarker, and Daryl L Essam. “Differential evolution with multiple strategies for solving CEC2011 real-world numerical optimization problems”. In: *2011 IEEE Congress of Evolutionary Computation (CEC)*. IEEE. 2011, pp. 1041–1048.

[14] ESA. *DE1220 Algorithm*. https://esa.github.io/pagmo2/docs/cpp/algorithms/de1220.html. Accessed: June 11, 2025. n.d.

[15] Nathan Hagen. *LensLibrary*. https://github.com/nzhagen/LensLibrary. Accessed: 2025-06-09. 2021.

[16] Alain Horé and Djemel Ziou. “Image Quality Metrics: PSNR vs. SSIM”. In: *2010 20th International Conference on Pattern Recognition (ICPR)*. IEEE. 2010, pp. 2366–2369.

[17] Matthias Hullin et al. “Physically-based real-time lens flare rendering”. In: *ACM SIGGRAPH 2011 papers*. 2011, pp. 1–10.

[18] James Kennedy and Russell Eberhart. “Particle swarm optimization”. In: *Proceedings of ICNN’95-international conference on neural networks*. Vol. 4. ieee. 1995, pp. 1942–1948.

[19] MJ Kilgard. “Fast OpenGL-rendering of lens flares”. In: *http://www. opengl. org/developers/code/mjk-tips/lensflare/* (2000).

[20] Rudolf Kingslake and R Barry Johnson. *Lens design fundamentals*. academic press, 2009.

[21] David C Knill and Alexandre Pouget. “The Bayesian brain: the role of uncertainty in neural coding and computation”. In: *TRENDS in Neurosciences* 27.12 (2004), pp. 712–719.

[22] Goro Kuwabara. “On the flare of lenses”. In: *Journal of the Optical Society of America* 43.1 (1953), pp. 53–57.

[23] Milton Laikin. *Lens design*. Crc Press, 2018.

[24] Laser Focus World. *Thin-Film Coatings: Understanding Key Design Principles of Anti-Reflection Coatings*. https://www.laserfocusworld.com/optics/article/16547029/thin-film-coatings-understanding-key-design-principles-of-antireflection-coatings. Accessed: June 11, 2025. n.d.

[25] Sangmin Lee. “Interactive Expressive Editing of Lens Flare Effect”. In: ().

[26] Sungkil Lee and Elmar Eisemann. “Practical real-time lens-flare rendering”. In: *Computer Graphics Forum*. Vol. 32. 4. Wiley Online Library. 2013, pp. 1–6.

[27] Xueqin Lü et al. “Comprehensive improvement of camera calibration based on mutation particle swarm optimization”. In: *Measurement* 187 (2022), p. 110303.

[28] Vincent Maurer. “Capturing Light with Robots: A Novel Workflow for Reproducing Realistic Lens Flares”. In: *SIGGRAPH Asia 2024 Technical Communications*. 2024, pp. 1–4.

[29] Erik Pekkarinen and Michael Balzer. “Physically based lens flare rendering in” The Lego Movie 2””. In: *Proceedings of the 2019 Digital Production Symposium*. 2019, pp. 1–3.

[30] Beat Reichenbach. *RealFlare*. https://github.com/beatreichenbach/realflare. Accessed: 8 May 2025. 2022.

[31] Tobias Ritschel et al. “Temporal glare: real-time dynamic simulation of the scattering in the human eye”. In: *Computer graphics forum*. Vol. 28. 2. Wiley Online Library. 2009, pp. 183–192.

[32] Martin Schlüter, Jose A Egea, and Julio R Banga. “Extended ant colony optimization for non-convex mixed integer nonlinear programming”. In: *Computers & Operations Research* 36.7 (2009), pp. 2217–2229.

[33] Robert R Shannon. *The art and science of optical design*. Cambridge University Press, 1997.

[34] Greg Spencer et al. “Physically-based glare effects for digital images”. In: *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. 1995, pp. 325–334.

[35] Rainer Storn and Kenneth Price. “Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces”. In: *Journal of global optimization* 11 (1997), pp. 341–359.

[36] Wen-Shing Sun et al. “Optical lens optimization design and tolerance analysis for a depth camera”. In: *Optik* 302 (2024), p. 171711.

[37] Ethan Tseng et al. “Differentiable compound optics and processing pipeline optimization for end-to-end camera design”. In: *ACM Transactions on Graphics (TOG)* 40.2 (2021), pp. 1–19.

[38] Kevin Van Nerum. “Real-time physically based rendering from a game development perspective”. In: (2016).

[39] Andreas Walch et al. “Lens flare prediction based on measurements with real-time visualization”. In: *The Visual Computer* 34.9 (2018), pp. 1155–1164.

[40] Walter E Woeltche. “Structure and image forming properties of asymmetrical wide angle lenses for 35-mm photography”. In: *Applied Optics* 7.2 (1968), pp. 343–351.

[41] Wenqi Xian et al. “Neural lens modeling”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 8435–8445.

[42] Yudi Optics. *Anti-Reflective Coating*. https://www.yudi-optics.com/technical-article/anti-reflective-coating/. Accessed: June 11, 2025. n.d.

[43] Kunlun Zhang et al. “Optimization of tilt-shift lens camera calibration parameters based on genetic algorithm and artificial neural network”. In: *Fourth International Conference on Image Processing and Intelligent Control (IPIC 2024)*. Vol. 13250. SPIE. 2024, pp. 436–443.

[44] Jingyang Zhou, Lyndon R Duong, and Eero P Simoncelli. “A unified framework for perceived magnitude and discriminability of sensory stimuli”. In: *Proceedings of the National Academy of Sciences* 121.25 (2024), e2312293121.

A

Canon lens prescription

Interface	Radius	Thickness	Refractive Index
1	684.66	2.62	1.805
2	-1055.76	0.20	
3	149.76	2.10	1.713
4	53.30	18.02	
5	-488.25	2.00	1.773
6	44.81	0.53	
7	43.27	3.50	1.847
8	78.34	40.13	
9	84.43	1.20	1.847
10	30.98	7.20	1.560
11	-1529.08	0.15	
12	50.67	6.00	1.652
13	-110.42	0.15	
14	40.57	3.30	1.652
15	71.98	6.91	
16		1.50	
17	-145.10	3.00	1.847
18	-34.13	1.20	1.603
19	112.83	2.00	
20	-42.83	1.40	1.603
21	66.44	13.17	
22	347.07	5.00	1.560
23	-26.27	1.50	1.805
24	-35.22	0.15	
25	104.39	5.00	1.713
26	-51.25	5.44	
27	-30.94	1.35	1.847
28	-84.63	80.00	

Table A.1: Lens Prescription for the Canon 28–80mm f/2.8 (US5576890). Interface 16 corresponds to the iris aperture of the lens.

B

UI

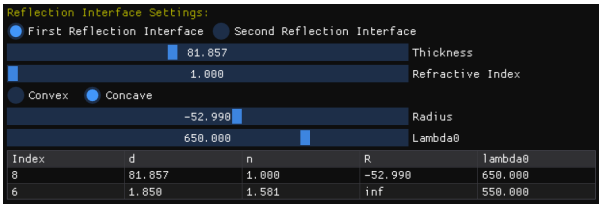


Figure B.1: Selected ghost's reflection interfaces adjustments.

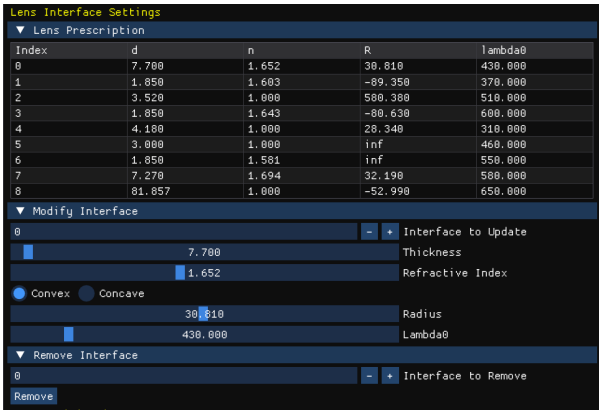


Figure B.2: Lens prescription table and interface adjustments.

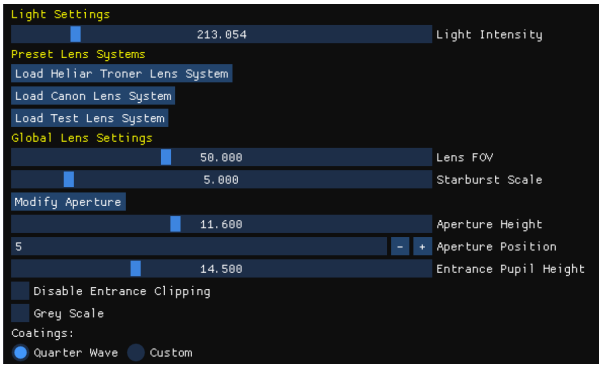


Figure B.3: Global scene and lens adjustments.

C

Differential Evolution Optimisation

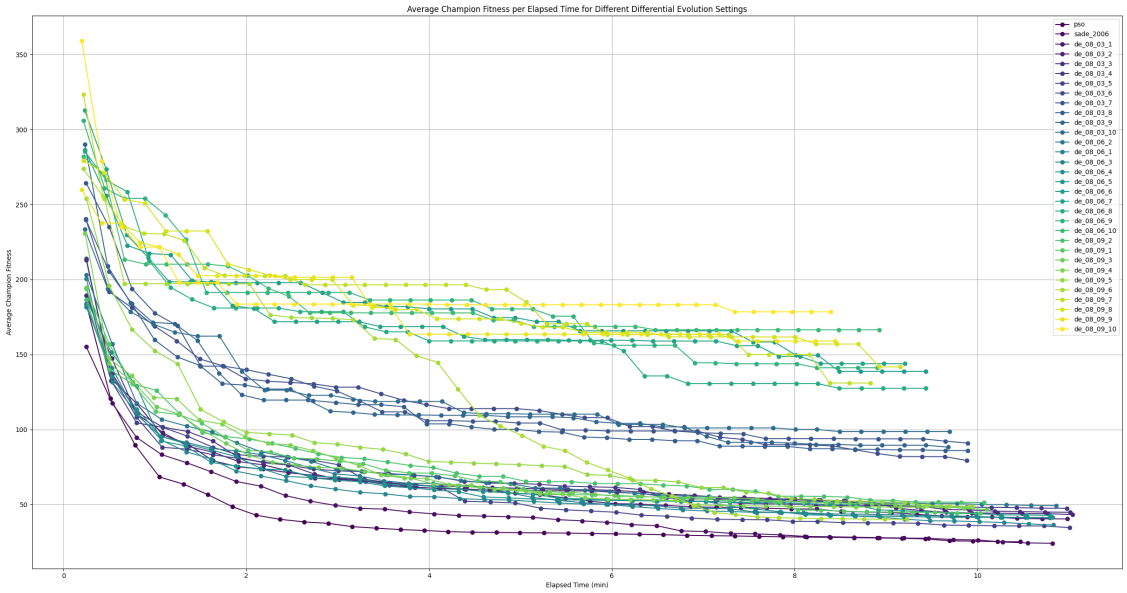


Figure C.1: Performance results of different DE configurations. Even with parameter tuning, SADE and PSO still outperform DE. The tests are run with the Heliar Tronnier, $5.45°$ light angle and 5 runs per variant.