dO: A differentiable engine for Deep Lens design of computational imaging systems

Congli Wang, Ni Chen, and Wolfgang Heidrich, Fellow, IEEE

Abstract—Computational imaging systems algorithmically post-process acquisition images either to reveal physical quantities of interest or to increase image quality, e.g., deblurring. Designing a computational imaging system requires co-design of optics and algorithms, and recently Deep Lens systems have been proposed in which both components are end-to-end designed using data-driven end-to-end training. However, progress on this exciting concept has so far been hampered by the lack of differentiable forward simulations for complex optical design spaces.

Here, we introduce dO (DiffOptics) to provide derivative insights into the design pipeline to chain variable parameters and their gradients to an error metric through differential ray tracing. However, straightforward back-propagation of many millions of rays requires unaffordable device memory, and is not resolved by prior works. dO alleviates this issue using two customized memory-efficient techniques: differentiable ray-surface intersection and adjoint back-propagation. Broad application examples demonstrate the versatility and flexibility of dO, including classical lens designs in asphere, double-Gauss, and freeform, reverse engineering for metrology, and joint designs of optics-network in computational imaging applications. We believe dO enables a radically new approach to computational imaging system designs and relevant research domains.

Index Terms—end-to-end lens design, image reconstruction, memory-efficient backpropagation, physics-based learning

I. INTRODUCTION

Cameras are designed with a non-trivial tradeoff between image quality (optical aberrations) and practical considerations (constraints, cost, form-factor, fabrication availability). In traditional optical design complex lens assemblies are simulated using ray-tracing and then semi-manually tuned to satisfy the performance requirements. Commercial software for automated lens design is nowadays the norm, such as ZEMAX [1] and Code V [2].

Computational imaging systems extend the capability of conventional imaging pipelines by having an additional degree of freedom in software. The spirit is to customize cameras or imaging modalities, and acquisition images are regarded as optically encoded information about the physical world. Computational methods, e.g., model-based numerical optimization or data-driven machine learning, are applied to raw images and decode the information. Examples are wavefront coding [3] and coded aperture variants [4], [5], [6] to extend depth-of-field or to perceive depth, cameras to capture individual rays (i.e., light fields) [7], [8], and cameras capable of high dynamic ranges [9].

The design of computational imaging systems is particularly challenging in that the design tradeoff happens in both hardware and software. As such the final design typically occupies a very high dimensional design space, that complicates easy and intuitive solutions. One way to explore the design space and evolve the co-design is to do stochastic optimization by gradient-descent, as shown in prior end-to-end trained Deep Lens works [10], [11], [12], [13], [14]. Such an approach requires the derivative of the final image concerning individual optical design parameters. This derivative-aware modeling is non-trivial for complex lens groups, and hence fully differentiable optical models have so far only been available for simplistic design spaces such as a diffractive optical element or a lensless mask, as demonstrated by various application-oriented deep optics designs, for image classification [15], depth estimation [16], [17], [18], lensless imaging [19], HDR acquisitions [20], [21], extended depth-of-field [22], computational microscopy [23], [24], and for versatile purposes [25], [26]. See also [27], [28] for perspectives. These are a long way from the expressive power of ray-tracing based systems like ZEMAX [1] and Code V [2].

To partially tackle these challenges, derivative-aware lens design engines [29], [30], [31] were inspired by automatic differentiation (AD) [32], a fundamental technique in deep learning. The main distinction of a derivative-aware engine compared to conventional ones is differentiability: The availability of derivative information relating design parameters and the error metric. Via differential ray tracing [30], design parameters and their gradients are chained to the error metric through a so-called computational graph, on which back-propagation results in how each design parameter should quantitatively change to reduce the error metric. Together with gradient-based optimization, the obtained derivatives provide a searching direction in the hyper-parameter space to locally guide evolution of current design, improving performance in terms of the error metric. Combined with deep neural networks, such a differentiable engine could be employed for generating lens designs [33], [34] or image restoration [35], [36]. Recent works [13], [14] rely on differentiable ray tracing for end-to-end designs in computational imaging. Similar trends also appear in other research domains for solving inverse problems utilizing this additional amount of...
information on derivatives, in computer graphics [37], 3D reconstruction [38], psychography [39], [40], head-mounted displays calibration [41], lens metrology [42], and phase microscopy [43].

However, one main challenge of direct applying backpropagation to gradient computations, is memory-hunger, in that the underlying computational graph could grow large when many millions of Monte Carlo rays are sampled, as required by rendering a megapixel image for a design algorithm to process with. As such, intermediate variables and computations could easily fill up device memory, limiting the scope of scaling-up, hence reducing the overall performance of a derivative-aware pipeline. Compared to model-based (or, physics-based) learning scenarios [44], our applications require the image formation to be computed in a Monte Carlo fashion and cannot be explicitly stated, and thus memory-efficient techniques rely on known image formation models like [44] are not directly applicable. Similar memory-hunger issue exists also in differentiable rendering for end-to-end learning [37], [45], and solutions have been proposed [46]. However, an optical design engine differs from a general-purpose graphics render, in that optical surface geometry representations are well-parameterized surfaces (e.g., aspheric or freeform splines) rather than discrete meshes. For non-spherical representations, the parameterization is so complicated that there are no analytical solutions for ray-surface intersection. As a result, ray-surface intersections are dominant computations, and become the memory-hunger bottleneck in rendering sensor image and its associated derivatives to design parameters. Specific problems like this in a differentiable optical design engine, are problems that this paper seeks crafted solutions for.

This memory issue has not been fully addressed in previous derivative-aware ray tracing works [29], [30], [33], [13] due to different application-oriented goals. In [13], the implementation of a differentiable ray tracing scheme enables image rendering, and thus rays are traced backwardly starting from the sensor plane, through the lenses, landing at the scene. Gradients are naively accumulated by unrolling iterations of a ray-surface intersection root finder, i.e. straightforward backpropagation. This most frequent yet time-critical operation consumes a large amount of memory and computing resources that limit the number of rays permissible for each render batch. Further, the gradient aggregation at the sensor plane suffers additionally to the large amount of rays sampled per pixel, and [13] sidesteps this issue by splitting the sensor image into blocks and rendering them independently. However this issue can be addressed in a more principled fashion where gradients are computed and back-propagated in a way that is efficient in terms of both compute time and memory. This is made possible by computing the gradients instead of directly back-propagating the computational graph, as explained below.

In this work we propose dO (DiffOptics), a derivative-aware lens design optimization pipeline using differentiable ray-tracing. To back-propagate from the error metric to design variable parameters in a memory-efficient way, we analyze the gradients in situations of ray-surface intersection and adjoint back-propagation, to enable validity and efficiency for gradient computation and inference. Based on the obtained derivatives, advanced algorithms could be developed for specific design applications.

Various applications are demonstrated using the proposed derivative-aware pipeline, ranging from classical usage such as spherical and aspherical lens group design optimization, local sensitivity analysis, freeform surface optimization, setup misalignment estimation, to advanced computational imaging applications. We believe that these examples highlight the potential and possible application domains shown by dO. This diverse application range differentiates dO from previous works as a general-purpose derivative-aware pipeline based on geometric ray tracing.

In contrast to earlier works that rely on external ray tracing engines such as [13], dO is fully implemented in PyTorch, which makes it easily portable and simplifies integration with existing machine learning image reconstruction methods, while at the same time utilizing GPU compute resources in an efficient fashion. Source code will be available at [47].

II. dO: Method and Approach

A. Overview

We use geometric ray tracing to model the light transport in a sequential-mode lens design. A lens system is parameterized by a vector variable $\theta \in \mathbb{R}^n$, a collection of $n$-number design parameters. Starting from one end of the lens system, sampling rays are sequentially traced through a set of parameterized optical surfaces, intersecting only once for each surface, while traveling towards the other end of the lens system. Rays can be traced starting the object plane towards the image plane, as the preferable way in lens design for aberration analysis. Alternatively, rays are traced reversely starting from the image plane towards the object plane preferably in graphics for image rendering. For the $i^{th}$ ray, intersection on the image (object) plane is determined by ray tracing, as a “black-box” function of $\theta$, denoted as $p_i(\theta) \in \mathbb{R}^2$. Given $m$ number of sampling rays, the collection of $p_1, \ldots, p_m$ is known as the spot diagram in lens design, denoted as $p \in \mathbb{R}^{2m}$, a concatenation of vectors.

To optimize a design, a merit function $\epsilon(p) : \mathbb{R}^{2m} \rightarrow \mathbb{R}$ is applied to $p$, producing a scalar error $\epsilon(p(\theta)) \in \mathbb{R}$, as an indicator for design performance. Design optimization aims to solve for an optimal $\theta^*$ by minimizing the error:

$$\theta^* = \arg \min_\theta \epsilon(p(\theta)). \quad (1)$$

For example, the usual merit function is the spot RMS error, by comparing the current spot diagram against a target one $p_{target}$, in a least square sense:

$$\epsilon(p) = \|p - p_{target}\|^2. \quad (2)$$

In dO, the merit function can also be at the irradiance level, when optimizing in the image space rather in the intersection space, e.g. in Refs. [43], [13]. This requires integrating a sensor image $I(p)$ from intersections $p$, given a pre-defined pixel size. The error merit can be defined such that the produced image is close to a target image $I_{target}$ in a least square sense using pixel-by-pixel comparison:

$$\epsilon(I(p)) = \|I(p) - I_{target}\|^2. \quad (3)$$
Crucially, the advantage of such an irradiance-based merit function allows for the consideration of software reconstruction modules such as deep neural networks in the design process. This allows for end-to-end designs in which the optics and the image processing software are jointly optimized, and design solutions are found in which the optical design maximizes the ability of the software module to restore quality images.

Given a merit function $\epsilon(\cdot)$, our goal is to evolve variable parameters $\theta$ iteratively towards an optimal $\theta^*$, using gradient-based optimization. This requires computing $\partial \epsilon / \partial \theta \in \mathbb{R}^n$, the partial derivatives that indicate how design parameters affect the error metric locally. Assuming $\epsilon(p(\theta))$ is a continuous function of $\theta$, by the chain rule, the partial derivatives in Eq. (1) expand as, without or with $I$:

\[
\frac{\partial \epsilon}{\partial \theta} = \frac{\partial p}{\partial \theta} \frac{\partial \epsilon}{\partial p},
\]

\[
\frac{\partial \epsilon}{\partial \theta} = \frac{\partial p}{\partial \theta} \frac{\partial I}{\partial \epsilon},
\]

In the following, we rely on using AD for computing Eq. (4). In AD, sole Jacobian matrices are rarely evaluated, instead the vector-Jacobian products are computed for a given perturbation, in our case denoted as $\Delta \epsilon$ and $\Delta \theta$, with $J^T(\cdot)$ denoting the Jacobian transposes, Eq. (4) rewrites as:

\[
\Delta \theta = J_p^T J^T \Delta \epsilon,
\]

\[
\Delta \theta = J_p^T J^T J_{\epsilon}^T \Delta \epsilon.
\]

Given a desired decrease $\Delta \epsilon$ in the error metric, our goal in this paper is to compute the corresponding variable parameter changes $\Delta \theta$ (i.e., the derivatives), by evaluating Eq. (5), and preferably being memory-efficient for scaling up to large design problems.

**B. Ray tracing with back-propagation**

A straightforward way to compute the derivatives in Eq. (4) is by using AD to perform all the computations in ray tracing. This requires using a derivative-aware numerical library, e.g., PyTorch [49], to track and compute both the primal value and its derivatives for every elementary arithmetic operation. For our multiple-input and single-output case, i.e., many variable parameters and one scalar error metric, for numerical efficiency it is preferable to employ the reverse-mode AD, or back-propagation in the context of machine learning. Briefly speaking, to perform back-propagation on Eq. (1) to calculate the derivatives, two stages are required, the forward pass and the backward pass.

Figure 1 illustrates this process by applying the reverse-mode AD to ray tracing. In the forward pass, values of differentiable parameters ($\theta$ in our case) are inputs from the starting plane to the ending plane, by ray tracing, to obtain intersections $p(\theta)$. With a user-defined merit function, an error metric $\epsilon(p(\theta))$ is calculated. The forward pass defines a computational graph in progressive that relates all differentiable variables, and non-differentiable variables are not needed for the forward pass with AD (i.e., forward with no AD in Figure 1). In the backward pass, this computational graph is back-propagated to compute gradients following the chain rule, starting from the previously obtained error metric $\epsilon(p(\theta))$, to compute variable derivatives $\Delta \theta$ as in Eq. (5). Figure 2 illustrates a derivative-aware example where for illustrative purposes the partial derivatives are shown for interpretation. For example, the derivatives of the spot diagram are the “flow” diagram $\partial p / \partial \theta$ revealing the motion of the spot diagram when changing $\theta$, where the arrows indicate moving directions and lengths indicate magnitudes. In this case, increasing $\theta$ will focus the rays, affecting more in the peripheral spots than the central ones. Similarly, the derivative of the rendered image is a “derivative” image $\partial I / \partial \theta$ that tells how would the pixel value changes when changing $\theta$. The “derivative image” indicates that the peripheral pixels are more sensitive than the central ones, as expected.

To ensure differentiability, all modules must be differentiable, i.e., modular computations are derivative-aware so that both the primals and their derivatives are computed and tracked.
in a computational graph. This requires both the optical system and the merit function to be fully differentiable, so that \( \theta, p, \epsilon(p(\theta)) \), and the desired derivatives \( \Delta \theta \) given \( \Delta \epsilon \), can all be related through ray tracing.

C. Differentiable ray-surface intersection

To perform ray tracing from surface to surface, the crucial part in a differentiable optical system is to calculate ray-surface intersection, where an intersection point \( x \) in \( \mathbb{R}^3 \) is computed. The intersection lies on an optical surface described by an implicit function \( f(x; \theta) = 0 \), and lies along the ray direction. Given a ray \( \{o, d\} \) of origin \( o \in \mathbb{R}^3 \) and direction \( d \in \mathbb{R}^3 \) of unit length, \( x = o + td \) for a positive marching distance \( t \in \mathbb{R}_+ \). Figure 3 illustrates this problem, which finds \( t > 0 \) such that:

\[
f(x; \theta) = f(o + td; \theta) = 0.
\]

It can be solved using iterative root finders (e.g., Newton’s method) implemented in AD, unrolling the iterations for gradient evaluations, so that the solution \( t \) and its gradients can be related to the lens parameters \( \theta \), as in Ref. [13]. However, this straightforward approach is memory consuming because of storing the intermediate iteration variables. This issue can be avoided by taking advantage of the fact that solution to Eq. (6) is independent on initialization of \( t \), and hence the optimal solution \( t^* \) can be first solved with no AD, no need for storing intermediate states, and calculate the AD-version \( t \) by one Newton iteration:

\[
t \leftarrow t^* - \frac{f(x; \theta)}{\nabla f(x; \theta)}, \quad \text{with} \quad x = o + t^*d,
\]

where \( \cdot \) denotes inner product, and \( \nabla f(x; \theta) \) denotes the spatial derivatives of the implicit function with respect to \( x \), parameterized by \( \theta \). Refer to Supplemental Document for specific forms of \( f(.) \) and \( \nabla f(.) \). Example surfaces are aspheres, \( XY \) polynomials, and B-splines. Notice our approach does not depend on specific iteration solvers chosen for solving Eq. (6), yet keeping the solution differentiable. Our approach is simple yet effective, and to our best knowledge is the first to address this memory issue in differentiable optics research.

We employ Newton’s method for obtaining \( t^* \), initialized by a non-singular estimate \( t^{(0)} = (zf - oz)/dz \), as in Figure 3 and the iteration stops when residual is smaller than the tolerance. Convergence of \( t^{(k)} \) and \( f(o + t^{(k)}d; \theta) \) is within a few iterations of \( k \). For spherical lenses, Newton’s method converges in theory within exactly one iteration. Figure 4 shows the superiority of the proposed approach, in comparison against the unrolled approach [13]. The unrolled approach needs to store intermediate states for AD, and consumes a large memory that cannot be affordable on a single GPU, and hence limiting the total number of freeform coefficients in the optimization. In contrast, our proposed differentiable solver drastically alleviates this issue, and imagine how often ray-surface intersection happen in ray tracing the design. This improvement allows \( dO \) to perform memory-efficient back-propagation when evaluating \( J_p^T \) in Eq. (5), no matter what merit function is chosen.

Our proposed method improves gradient estimates for Deep Lens designs, by allowing more rays to be sampled per pixel. In Figure 5 a doublet aspheric lens (see Table III in Supplemental Document for prescription) is under investigation, with its first surface’s 4th aspheric coefficient \( \theta = 0 \) being differentiated. Given a planar texture image object, \( dO \) renders the corresponding sensor image \( I \) and its gradient \( \partial I/\partial \theta \). Figure 5 shows the same gradient image computed using method in [13] and ours, as well as a finite difference version for reference, which is inaccurate due to limited numerical precision. Method in [13] is limited to fewer rays per pixel because of high memory consumption in computing ray-surface intersection, producing a noisy gradient image. In contrast, our method allows for more rays to be sampled per pixel, yielding a cleaner gradient image. This example demonstrates the superiority of our approach over [13], and the benefit of applying our approach for robust gradient estimate, and hence to Deep Lens training.

D. Adjoint back-propagation

Evaluating Eq. (5) depends also on the specific merit function chosen. For simple metrics such as the spot RMS in conventional lens designs, evaluating Eq. (5a) does not involve many rays and the memory issue is not urgent. However, when the merit function is in the image space, e.g. Eq. (3), and involves rendering images, for example a megapixel resolution image \( I(p) \) may take an intensive sampling of many millions of rays. In this scenario, direct back-propagating on Eq. (5b) could be problematic and may require impractical amounts of
In end-to-end learning, the goal is to optimize both the optical system parameters $\theta$ and network weights $\theta_{\text{net}}$ such that the hardware-software combination produces the optimal image quality for the neural network to sharpen raw sensor images. See Figure 15 and 16 for examples.

Figure 7 shows a memory consumption comparison between the two back-propagation methods, applying to Figure 2 assuming a dummy merit function of Eq. (3). The adjoint back-propagation takes more time, but the memory consumption maintains at a low level, allowing for aggregations and sampling scale-ups. This memory-efficiency enables a large number of sampling rays for faithful image rendering for practical usage in end-to-end computational imaging designs.

III. IMPLEMENTATION

A. Optimization

Given the derivatives $\Delta \theta$ from back-propagating Eq. (5), $dO$ performs optimization and iteratively changes the variable parameters $\theta$. When there are constraints in the design, e.g., positive air-spacing, minimum glass thickness or back focal length, maximum system overall size, Eq. (1) can be modified by adding a vector constraint function.

The specific optimization method depends on the number of variables. When the number of variables is small (for example $\theta \in \mathbb{R}^n$, $n < 20$), Classical damped least squares [50] are employed to efficiently optimize Eq. (1), that the required Jacobians can be constructed from the derivative vectors. In this case it does not take full advantage of the derivative-aware property of $dO$. This is in contrast to the case when $n$ is large (e.g. for free-form surface optimization) and the Jacobians are too large to be explicitly constructed, and popular gradient descent methods such as Adam [51] can be employed. If desirable, additional regularization terms are possible, for example when solving Eq. (15). This optimization flexibility feature differentiates $dO$ from existing optical design software.

B. Lens system

We follow the standard lens design pipeline [52], [53] to model a lens system. We focus on the sequential mode, where starting from one end of the lens system, rays are sequentially traced through a sequence of parameterized optical surfaces (including the image plane, i.e., the sensor plane), intersecting only once for each surface, while traveling towards the other end of the lens system. In the sequential mode, the exact visibility ordering of the surfaces is known a priori, and thus no need for finding the closest surface intersection when performing ray tracing.

Ray propagation through a lens system involves two major steps, finding the ray-surface intersection point (finding intersection points by solving Eq. (6)), and refraction of the ray at material interfaces with chromatic effects. Only valid rays are traced in continuity, whereas invalid rays happen when the intersections are outside of the lens geometry or when total internal reflection takes place.
At material interfaces, transmitted direction $d_t$ is determined from surface normal direction $n = \nabla f / \| \nabla f \|$ and incident direction $d_i$, by Snell’s law \[54\]:

$$d_t = n \sqrt{1 - (1 - \cos^2 \psi_i) \eta^2} + \eta (d_i - n \cos \psi_i), \quad (9)$$

where $\cos \psi_i = d_i \cdot n$ and $\eta = n_1 / n_2$ is the ratio of refraction indices of the two materials. Refractive index follows Cauchy’s equation $n(\lambda) = A + B / \lambda^2$, with $A$ and $B$ determined from central refractive index and Abbe number.

Though in this work we focus on the sequential mode where optical surfaces are fixed in a known order, non-sequential mode should also be possible with proper extensions and modifications on the current ray tracing engine.

### C. Two tracing modes: forward and backward

Depending on the needs, rays can be traced through a lens system in two different modes, forward mode or backward mode. In the forward mode, rays are traced starting from the object plane towards the image plane. This is the preferable way in lens design for aberration analysis, e.g., generation for spot diagrams. In the backward mode, rays are traced reversely, starting from the image plane towards the object plane. This is a sampling efficient way for sensor image rendering, and thus is the preferable way in computer graphics to render realistic images. We will be using these two modes interchangeably depending on specific needs.

#### D. Differentiable image rendering from intersections

Once tracing is finished, a synthetic image $I(p)$ can be generated given the intersection points $p$, provided with a proper image integrator. This process (termed rendering) of pixelization from a continuous light signals to discretized pixel values, is handled differently in the two tracing modes.

**Reconstruction filter in the forward mode:** In the forward tracing mode, performance analysis is conduct to understand the optical property of the current design, where rays are purposely generated according to analysis-specific criteria. This process may involve gradient computations, and differentiability is desired in that the generated image pixel values are differentiable to intersection point movements. To ensure differentiability, the derivatives of this mapping from intersection to pixel values need to be continuous. $dO$ employs a linear interpolation scheme to round the fractional intersection point $p$ into the nearest four neighboring pixels, with the corresponding four pixel values linearly weighted.

**Monte Carlo integrator in the backward mode:** In the backward tracing mode, rays are initialized from the pixelated sensor plane, and are traced outwards, and the major goal is to render a physically correct image for the current design given a specific scene. This is achieved by integration of the rendering equation \[55\], and is evaluated by renderers using a Monte Carlo integrator for discrete sampling the integral. This backward tracing mode is utilized in end-to-end system designs, see Section \[V\].

#### E. Entrance/Exit pupil computation for all fields

To perform ray tracing for a lens system, the entrance pupil has to be determined first, which is the area over the very front lens element where rays from a given viewing angle will finally reach the sensor plane. This is easily determined for paraxial angles, but not for larger angles. Figure \[8\] demonstrates this vignette effect of a double-Gauss design \[56\], with the calculated entrance pupils shown at different views. Our
engine determines the entrance pupil by ray tracing a dense grid (1025 × 1025) at specific viewing angles. Entrance pupils are determined if the sampled rays propagate through all the optical elements successfully. Exit pupils can be determined in a similar manner as described in [57].

![Double-Gauss design | Entrance pupils at 0°, 7°, 14°, and 21°](image)

**Fig. 8:** Entrance pupil calculation. By tracing a dense grid at the very first optical surface, whose aperture is the dark circle, overlaid on which we can obtain the entrance pupil area (bright region) for subsequent ray spatial sampling, at different viewing angles.

**F. Derivative-aware implementation**

dO is implemented from scratch in PyTorch [49]. In comparison to [13], which is based on Mitsuba2 [37], dO is therefore more portable and very easy to combine with existing deep neural network code for image reconstruction. This makes it easier to integrate dO in various end-to-end design problems while at the same time offering more degrees of freedom in terms of flexibility for additional applications. Due to the PyTorch backend, dO effectively uses GPU compute resources and due to the algorithmic improvements outlined in the previous section, it is significantly more memory efficient than previous end-to-end differentiable ray-tracing systems. dO produces highly identical results to modern lens design software. To verify this, a double-Gauss design [56] is under scrutiny check with Zemax [1], using single wavelength λ = 587.56 nm at four field of views, as in Figure 8. The spot diagrams and the RMS errors are almost identical to the Zemax results despite a slight variation due to different aperture sampling strategies.

**A. Design optimization**

**Spherical aberration minimization:** The first example is to optimize the aspheric coefficients of an aspheric lens to minimize the axial spherical aberration. In Figure 10 parameters of a revised asphere lens design (Thorlabs, ACL5040U) are optimized in the hope of reducing the axial RMS spot. Compared to the initial design, our differentiable engine ends up with a nearly six times smaller RMS spot.

![Aspheric lens | Original (initial) | Optimized](image)

**Fig. 10:** Spherical aberration minimization. Differentiable parameters θ are surface curvature, conic and 4th asphere coefficient. Our solver converges in 1 s.

**Photographic camera design optimization:** The engine can also optimize complicated lens group for design optimization. Figure 11 shows the second example to re-parameterize curvatures and aspheric coefficients of a Nikon patent design to minimize the total RMS spot error at different field of views (0°, 10°, 20°, 32.45°), at three wavelengths (656.27 nm, 587.56 nm, 486.13 nm). The design is initialized by removing all aspheric coefficients, showing large aberrations. After optimization, our optimized design shows a comparable mean RMS error of the original design. This example demonstrates the capability of dO to perform multi-lens and aspheres optimization.

![Initial | Optimized (Ours) | Original design](image)

**Fig. 11:** Nikon lens group total aberration minimization. Starting from an all-spherical design, with the differentiable parameters θ being all surface curvatures and aspheric coefficients of two surfaces, our optimized version achieves similar performance as the original design.

**B. Tolerance analysis**

In tolerancing a lens system, or known as sensitivity analysis, a presuming small, linear parameter perturbation ∆θ is enforced, and the total effect of the perturbation, ∆c, is calculated through the nominal system to determine the potential effects. To avoid confusion between perturbation and the gradients considered before, we use the () notation. This is mathematically paraphrased by relating ∆c to ∆θ through

**IV. CLASSICAL APPLICATIONS**

dO can manage classical design problems, as will be demonstrated in this section. Spot RMS error at different viewing angles is chosen as the merit function c() for design optimization. See Supplemental Document for lens prescriptions and full details.
the derivatives, known as the Jacobian-vector product in AD terminology:

$$\Delta \epsilon = J_p \Delta \theta.$$  \hspace{1cm} (10)

Notice there is a reciprocal relationship between Eq. (10) and Eq. (5). In the forward analysis, $\Delta \theta$ is given to compute $\Delta \epsilon$, whereas in the inverse analysis, $\Delta \epsilon$ is given to compute $\Delta \theta$, assuming proper prior probabilities or regularization on $\Delta \theta$. Despite alternative methods such as finite difference, analytical gradients, or the wavefront differential method, $\Delta \theta$ can evaluate Eq. (10) intrinsically, without additional implementation efforts.

In Figure 12 a Cooke triplet is under sensitivity analysis, with all the optical element positional misalignment parameters $\theta$ being tolerated. $\Delta \theta$ can obtain the Jacobian matrix, known as the sensitivity matrix for further analysis. See Supplemемtal Document for misalignment parameter definitions.

In Figure 12 a Cooke triplet is under sensitivity analysis, with all the optical element positional misalignment parameters $\theta$ being tolerated. $\Delta \theta$ can obtain the Jacobian matrix, known as the sensitivity matrix for further analysis. See Supplemental Document for misalignment parameter definitions.

render caustic images. To enable a satisfactory reconstruction, the desired freeform surface is assumed to be smooth and hence is represented by B-splines with a large degree of freedom approximately equal to the pixel number of the target image. The error metric was the standard mean-square-error (MSE), and we optimize the B-spline coefficients $\theta$ that characterize the freeform surface:

$$\min_{\theta} \| I(\theta) - I_{\text{target}} \|^2.$$  \hspace{1cm} (11)

Eq. (11) is optimized using Adam [51] because construction of Jacobian is computationally prohibitive due to the large number of variables in Eq. (11). Figure 14 shows the results along with intermediate optimization states, where the optimized freeform surface is shown in height maps. The reconstruction image is contrast-preserving, and the optimized freeform surface is smooth. Although in this particular example, the contrast in our result is not high compared to alternative specific solvers, e.g., using optimal transportation [60] or iterative warping [61], our result shows freeform designs would be more approachable when a plug-and-play differentiable engine such as $\Delta \theta$ is available.

V. ADVANCED APPLICATIONS

Thanks to differentiability and the versatile optimizer, $\Delta \theta$ can be combined with advanced post-processing computational algorithms for complex designs or setup reverse-engineering, and beyond the usual application range of lens design software. Such domain-specific applications are deemed not easily configurable with existing design software, as three examples shown in Figure 13.

A. Caustic engineering

Caustic engineering aims to produce a target image by pixelwisely changing the directions of a directional light source, by optimizing a freeform optical surface [60], [61]. Here, we demonstrate caustic engineering as one of the applications of $\Delta \theta$. We parameterize the desired surface in spline freeform, and thus this representation ensures smoothness and higher-order continuities of the surface, in contrast to the mesh-based representations in alternative differentiable solvers [32]. The freeform knot positions are fixed, spline coefficients are optimized and consequently changing the surface geometry. This flexibility of geometry representation allows $\Delta \theta$ to perform freeform surface optimization. We employ the forward mode to
the polynomial coefficients third-order XY polynomials, and end-to-end jointly optimize by ray tracing. We parameterize the phase plate using only limitation, in that the optical system is faithfully reproduced PSFs. Here, \( dO \) approximation \[10\], ignoring the spatially variant nature ofing algorithm. Prior end-to-end approach relies on paraxial algorithmic problem in that the final image is the output from PSFs for depth retrieval. This process is a joint optical-aperture \[4\] and lattice-focal \[63\] that produce depth-sensitive depth-invariant PSFs for extended-depth-of-field, and coded (PSF) engineering, e.g., cubic phase plate \[3\] that produces VGG16 \[65\]: end computational imaging designs, using goal in the following is to minimize Eq. (12) for two end-to-loss \( L \)Specifically, the total error metric in Eq. (12) consists an MSE where \( \epsilon \) is a general loss metric function to quantify a total error difference between the post-processed image \( N(I(\theta); \theta_{\text{net}}) \) and the ground truth target image \( I_{\text{target}} \). The goal in the following is to minimize Eq. (12) for two end-to-end computational imaging designs, using \( dO \).

**End-to-end design for extended-depth-of-field imaging:** In wavefront coding, a pupil plane phase modulator is introduced to deliberately distort input lights for point-spread-function (PSF) engineering, e.g., cubic phase plate \[3\] that produces depth-invariant PSFs for extended-depth-of-field, and coded aperture \[4\] and lattice-focal \[63\] that produce depth-sensitive PSFs for depth retrieval. This process is a joint optical-algorithmic problem in that the final image is the output from a sequential appliance of the encoding optics and the decoding algorithm. Prior end-to-end approach relies on paraxial approximation \[10\], ignoring the spatially variant nature of PSFs. Here, \( dO \) provides an initial solution to break this limitation, in that the optical system is faithfully reproduced by ray tracing. We parameterize the phase plate using only third-order XY polynomials, and end-to-end jointly optimize the polynomial coefficients \( \theta_{XY} \) and U-Net \[64\] parameters \( \theta_{\text{net}} \) for extended depth of field applications, as in Figure 15. Specifically, the total error metric in Eq. (12) consists an MSE loss \( L_{\text{mse}} \) and a channel feature loss \( L_{\text{vgg16}} \) on pre-trained VGG16 \[65\]:

\[
\min_{\theta_{XY}, \theta_{\text{net}}} \epsilon(\theta_{XY}, \theta_{\text{net}}) = L_{\text{mse}} + L_{\text{vgg16}}. \tag{13}
\]

See Supplemental Document for full details on network architecture, definitions of loss terms \( \mathcal{L} \), and training set generation. Initialized from a null zero phase, the optimized phase in Figure 15(a) exhibits a similar structure as in \[3\], verified by the central PSFs in Figure 15(b). Given the blurry raw input images, the post-processed images in Figure 15(c) reveal sharp features.

**End-to-end design for large field-of-view (FOV) imaging:** Large field-of-view (FOV) imaging is challenging for cemented doublets due to the severe aberrations that cannot be fully compensated because of the lack of design degree of freedoms. We argue this large aberrations can be corrected not through additional optical components, but through a computational post-processing algorithm that applies on the perceived aberration image. Using an end-to-end learning, this optical aberration limitations can be addressed by using a post-processing neural network, here we use \[66\]. The training follows Eq. (12), optimizing:

\[
\min_{\theta_{\text{asphere}}, \theta_{\text{net}}} \epsilon(\theta_{\text{asphere}}, \theta_{\text{net}}) = L_{\text{mse}} + L_{\text{vgg16}}, \tag{14}
\]

A cemented doubleter parameterized by its aspheric coefficients \( \theta_{\text{asphere}} \) and network parameters \( \theta_{\text{net}} \) are jointly optimized following a similar metric as in Eq. (13). Figure 16 shows how the original blurry images can be post-processed and be deblurred, after the end-to-end learning, justifying the applicability of \( dO \).

**C. Misalignment estimation**

Finally, we show real experimental results to leverage potentials of the proposed differentiable pipeline, \( dO \), by using it in reverse to estimate misalignment of an experimental setup. This task of misalignment estimation is not possible with previous differentiable renderers, e.g., Mitsuba2 \[37\] or Ref. \[13\], because of geometric representation incompatibility and missing implementations for optics element perturbations, and most importantly, the memory efficiency of \( dO \) allows for large number of rays to be sampled per pixel for gradient estimates. In Figure 17(a), a pinpoint light source, consists of an LED (central wavelength 622 nm) and an iris of radius 0.4 mm, is placed far in front of a misaligned plano-convex lens. The lens was able to rotate freely, but the exact angular values were unknown, and the imperfect mounting leads to slight yet noticeable tilting. The setup is imaged by a monochromatic CMOS sensor (FLIR, GS3-U3-51S5M-C, pixel size of 3.45 µm). Without knowing the exact position and angle parameters \( \theta \), e.g., light source position, sensor to lens distancing, lens yaw/pitch angles, it is very challenging to reproduce experimental measurements by manual parameter tuning a simulation setup. In Figure 17 we show the success of \( dO \) to estimate such misalignment parameters, by minimizing the MSE error between the simulation image \( I(\theta) \) and the target real captured image \( I_{\text{real}} \). We employ the forward mode to render \( I(\theta) \). To escape local stationary points and to regularize gradient computation, we enforce centroid alignment between the two images, by denoting \( C(\cdot) \) as an operator for calculating image centroid:

\[
\min_{\theta} \| I(\theta) - I_{\text{real}} \|^2 + \mu \| C(I(\theta)) - C(I_{\text{real}}) \|^2, \tag{15}
\]

where \( \mu \) is a tradeoff parameter to balance between MSE and alignment errors. Empirically, Eq. (15) is optimized by Adam and damped least squares in alternation. The optimization usually takes \(< 0.5 \text{ min} \) to finish for a megapixel image resolution on a GPU (Nvidia, GeForce RTX 2080 Ti).
**Fig. 15:** dO can perform end-to-end wavefront coding, i.e., jointly optimize phase optics profile and deconvolution algorithm for extended depth of field applications, here by simulation. (a) Setup geometry and optimized phase optics. (b) Central PSFs at different distances. (c) Raw and post-processed images by the neural network.

**Fig. 16:** dO can perform end-to-end large FOV imaging by jointly optimizing aspheric coefficients and a neural network for large FOV applications, here by simulation. Raw and post-processed images by the neural network.

Two lenses of different focal lengths (Thorlabs, LA1131, focal length of 50 mm; LA1986, focal length of 125 mm) were under investigation, as shown in Figure 17(b) and Figure 17(c). In Figure 17(b), the lens was focused but tilted, and the goal is to re-parameterize the simulation to fit real measurements. In Figure 17(c), the lens was slightly defocused, introducing a blurry bright disk on the sensor plane. With an increase of angular misalignment, the image smears and elongates in the horizontal direction. Initialized from a coarse setup estimation, the final optimized images match real images visually well. Refer to Supplemental Document for full results. This example proves the validity of the differentiability of dO, and demonstrates the possibility of using dO.
for setup calibration. The simplicity of using the proposed engine for misalignment estimation allows integrating self-calibration into existing numerical modeling pipelines. From a broader perspective, we demonstrate the possibility of using a differentiable ray tracer such as dO, as a general inverse solver for metrology problems.

VI. DISCUSSION AND CONCLUSION

In principle, ray optics limits the application range of dO in that the wave nature of light is ignored. Resolution demanding imaging applications towards diffraction-limited performance such as telescope or microscope designs are hence not possible at this point. In methodology, we rely on gradient descent and damped least squares as the optimization techniques, and thus a number of iteration steps are required for convergence. Due to its local optimization nature, the solver suffers from the local minima problem and the initialization sensitivity issue. In software, dO relies mostly on reverse-mode AD, which is known to be memory-consuming especially for large amount of Monte Carlo samples, though partially alleviated, can still limit the number of rays used when optimizing the design. More memory-efficient approaches could be explored. We also expect faster speed performance after careful code improvements.

The current ray tracing engine could be further enhanced. From design perspective, more surface representations could be implemented, e.g., Laguerre, Hermite, and Zernike polynomials. New tracing methods are possible, for example paraxial tracing [67] and Gaussian beam tracing [68]. Also, new features could be implemented, e.g., diffraction and gratings, Fresnel equation and polarization ray tracing, coatings, stray light and ghost analysis, and non-sequential tracing. From application perspective, further applications could be explored, e.g., lens metrology [69], realistic lens rendering [70], and wavefront sensor designs [71, 72]. Hardware-software end-to-end optimization for domain-specific applications [62], [10], [11], [12], or semi-supervised training for automatic lens design [62], wavefront sensor designs [71, 72]. Hardware-software end-to-end optimization for domain-specific applications [62], [10], [11], [12], or semi-supervised training for automatic lens design [33, 34] are also target topics. We believe dO serves as an initial starting point towards these applications.

To conclude, we have proposed dO, a differentiable ray tracing engine for lens design. Board applications are demonstrated, ranging from classical design optimization and sensitivity analysis, to computationally intensive freeform design, end-to-end learning for computational imaging applications, and to challenging experimental misalignment estimation. We envision the potential of dO, as it opens up an exciting aspect to bring first-order gradient insights into lens design and relevant problems.

REFERENCES


Congli Wang received the B.Eng. degree in electrical engineering from Tianjin University in 2015 and the M.Sc. and the Ph.D. in electrical engineering from King Abdullah University of Science and Technology in 2016 and 2021. He is currently a postdoctoral researcher at the University of California, Berkeley. His research interests include computational imaging, wavefront sensing, and adaptive optics.

Ni Chen received the B.S. degree in software engineering from Harbin Institute of Technology, China, in 2008, the M.S. and Ph.D. degrees in electrical engineering from Chungbuk National University and Seoul National University, Korea, in 2010 and 2014, respectively. She is currently a Researcher at the King Abdullah University of Science and Technology. From 2014 to 2016, she was a Research Scientist with the University of Hong Kong, and from 2016 to 2017, and she was an Associate Professor with the Shanghai Institute of Optics and Fine Mechanics, Chinese Academy of Sciences. From 2018 to 2019, she was a Research Assistant Professor in the Department of Electrical and Computer Engineering at Seoul National University, Seoul, Korea. Her research interests include three-dimensional optical imaging and display.

Wolfgang Heidrich is a Professor of Computer Science at King Abdullah University of Science and Technology (KAUST). Dr. Heidrich joined KAUST in 2014, after 13 years as a faculty member at the University of British Columbia. He received his Ph.D. from the University of Erlangen in 1999, and then worked as a Research Associate in the Computer Graphics Group of the Max-Planck-Institute for Computer Science in Saarbrucken, Germany, before joining UBC in 2000. Dr. Heidrich’s research interests lie at the intersection of imaging, optics, computer vision, computer graphics, and inverse problems. His more recent interest is in computational imaging, focusing on hardware-software co-design of the next generation of imaging systems, with applications such as High-Dynamic Range imaging, compact computational cameras, hyperspectral cameras, to name just a few. Dr. Heidrich’s work on High-Dynamic Range Displays served as the basis for the technology behind Brightside Technologies, which was acquired by Dolby in 2007. Dr. Heidrich has chaired the papers program for both Siggraph Asia and the International Conference of Computational Photography (ICCP) among others. He is the recipient of a 2014 Humboldt Research Award.