DOI: 10.1111/cgf.13087

COMPUTER GRAPHICS *forum*
*Volume 36 (2017), number 8 pp. 390–401*

# NeuroLens: Data-Driven Camera Lens Simulation Using Neural Networks

Quan Zheng[1,2] and Changwen Zheng[1]

[1]Science and Technology on Integrated Information System Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing, China
quan.zheng@outlook.com, cwzheng@ieee.org
[2]University of Chinese Academy of Sciences, Beijing, China

**Abstract**

*Rendering with full lens model can offer images with photorealistic lens effects, but it leads to high computational costs. This paper proposes a novel camera lens model, NeuroLens, to emulate the imaging of real camera lenses through a data-driven approach. The mapping of image formation in a camera lens is formulated as imaging regression functions (IRFs), which map input rays to output rays. IRFs are approximated with neural networks, which compactly represent the imaging properties and support parallel evaluation on a graphics processing unit (GPU). To effectively represent spatially varying imaging properties of a camera lens, the input space spanned by incident rays is subdivided into multiple subspaces and each subspace is fitted with a separate IRF. To further raise the evaluation accuracy, a set of neural networks is trained for each IRF and the output is calculated as the average output of the set. The effectiveness of the NeuroLens is demonstrated by fitting a wide range of real camera lenses. Experimental results show that it provides higher imaging accuracy in comparison to state-of-the-art camera lens models, while maintaining the high efficiency for processing camera rays.*

**Keywords:** camera lens simulation, neural networks, regression, lens effects

**ACM CCS:** I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism Raytracing

## 1. Introduction

Real-world lens effects, such as depth of field, bokeh and distortion, are strongly influenced by the interaction of light with a camera lens. To obtain realistic lens effects, we usually have to resort to simulating the image formation of a real camera lens, which contains an array of lens elements. However, accurately simulating light transport through all lens elements of a camera lens is computationally expensive.

In past decades, much research work in computer graphics has been conducted to simulate camera lenses and their image formation. Several camera lens models, such as pinhole model, thin lens model and thick lens model, have been designed according to basic geometrical optics principles. Full lens model [KMH95] provides a brute-force solution to simulate the image formation. Recently, Taylor polynomial model [HHH12] is proposed to approximate the image formation with a polynomial system. The interaction between a ray and a lens element is expressed as a polynomial basis, and the polynomial system of a camera lens is constructed by concatenating all polynomial bases. Truncation operation is utilized to control the number of terms, but it leads to accuracy loss. In addition, it can be hard to represent camera lenses with unknown interior lens elements, or non-analytic lens surfaces.

In this paper, we propose the NeuroLens for simulating image formation of real camera lenses. Firstly, we formulate the mapping from incident rays to emergent rays as an imaging regression function (IRF), which describes the imaging properties of a camera lens. We then model the IRF with an acyclic multi-layer feed-forward neural network. Neural networks learn the mapping from accurate ray samples, which are produced by spectral ray tracing in an off-line renderer. In addition, the parallel structure of neural networks also enable the efficient evaluation on GPU.

Secondly, to exploit the local coherence of imaging properties of a camera lens, we subdivide the space spanned by all incident rays into multiple subspaces. We fit a separate IRF for each subspace. Instead of using a single neural network, we construct a neural network ensemble to model an IRF. The output of an IRF is approximated by

the average output of neural networks in the ensemble. Therefore, the error caused by a single neural network which falls in a local optimum can be alleviated.

The NeuroLens provides a new data-driven model for simulating real camera lenses. It is able to produce high-quality images similar to those generated by accurate ray tracing in a full lens model, and it can process camera rays more efficiently with parallel evaluation.

The rest of the paper is organized as follows: Section 2 summarizes previous work. Then, the definition of IRF is presented in Section 3. Details of the NeuroLens are given in Section 4, followed by more specific implementation details. In Section 6, we validate robustness of the NeuroLens and compare it with state-of-the-art camera lens models. Before the conclusion, several issues with respect to the design of a NeuroLens are discussed in Section 7.

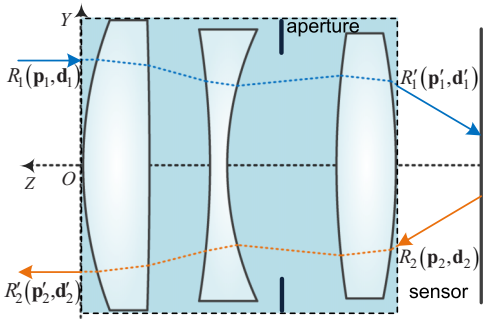


**Figure 1:** *Schematic of a Schneider F/4.5 lens. By convention, object space is on the left and coordinates of z-axis increase from right to left. The camera lens (shaded area) can be abstracted as a 'black box' which transforms an incident ray to an emergent ray.*

## 2. Previous Work

### 2.1. Camera lens models

**Geometrical camera lens models.** The most basic camera lens model in computer graphics is the pinhole model with an ideal point-sized aperture. It originates from the basic optical imaging model in optics. Subsequently, thin lens model [PC81] with infinitesimal thickness and finite aperture size is proposed in the research of rendering depth of field. Because of its simplicity, thin lens model is widely applied in real-time rendering. However, advanced lens effects, including monochromatic aberrations and chromatic aberrations, cannot be produced by thin lens model. As an extension, thick lens model [KMH95] with finite thickness and finite aperture is designed to approximate camera lenses. Then, Kolb *et al.* [KMH95] propose a full lens model, which comprises a sensor, several lens elements and stops. They employ distributed ray tracing [CPC84] in the camera lens to produce nearly accurate lens effects. Unfortunately, it requires a large amount of computation of ray-lens intersection within a camera lens and costs considerable time. Steinert *et al.* [SDHL11] and Wu *et al.* [WZHX13] further extend the full lens model to render chromatic aberrations caused by dispersion of lens elements.

**Polynomial models.** Approaching from another perspective, Hullin *et al.* [HHH12] propose an approximate model based on Taylor polynomials. The interactions between a ray and a lens element, including propagation, refraction and reflection, are formulated as a set of polynomial bases. Along the route, Lee and Eisemann [LE13] apply a first-order polynomial system to render lens flare at real-time frame rate. While the Taylor polynomial model can be applied for rendering lens flare, it is not accurate enough for general imaging. Hanika and Dachsbacher [HD14] propose a calibration step to optimize the coefficients of polynomials, effectively improving the imaging precision. However, some complex camera lenses, such as wide-angle and 'fisheye' lenses, can hardly be well calibrated. The above methods construct the polynomial system for a camera lens by concatenating the polynomial basis of each lens element. To limit the increase of polynomial terms and its orders, truncation operation is performed to restrict the polynomial to a fixed order. Unfortunately, loss of imaging accuracy is introduced. To obtain high accuracy with limited polynomial terms, Schrade *et al.* [SHD16] propose to fit a sparse polynomial model from high-order polynomials, which is able to simulate difficult wide-angle lenses.

By contrast, the NeuroLens learns the mapping between incident rays and outgoing rays via supervised training on accurate path-traced ray samples. There is no additional accuracy loss in the construction process. Furthermore, because of the subdivision of input space and the use of neural network ensemble, a trained NeuroLens can achieve high imaging accuracy for a wide range of camera lenses, including the challenging wide-angle lenses.

### 2.2. Neural network applications in computer graphics

Neural networks have been adopted in several topics. Grzeszczuk *et al.* [GTH98] use neural networks to synthesize animation sequences and fit animation controllers. Then, Nowrouzezahrai *et al.* [NKF09] give a method for producing self-shadowing based on neural networks. In 2011, Dachsbacher [Dac11] utilizes neural networks for classifying different visibility configurations. Ren *et al.* [RWG*13] propose to predict local radiance in the scene with neural networks. Recent year has witnessed more applications of neural networks, such as relighting of photos [RDL*15], vector texture compression [SWWW15] and removal of Monte Carlo noise [KBS15]. By contrast, this paper leverages neural networks to approximate the mapping of image formation, which can be formulated as an IRF.

## 3. Imaging Regression Function

Many applications in computer graphics try to reproduce images as realistic as ones produced by real cameras. Commodity optical cameras are designed with sophisticated combination of multiple lens elements, stops and a sensor. Figure 1 shows the schematic of a Schneider F/4.5 camera lens [S*05].

In the imaging process of a camera lens, incident rays from the scene traverse all lens elements and reach the sensor to form an image. A camera lens can be viewed as a mapping between incident rays and outgoing rays, which transforms the light field in the scene to the light field at the sensor. The calculation of outgoing

rays via the mapping boils down to a regression problem. Thin lens model and thick lens model approximate the mapping with perspective transformation matrix, whereas the full lens model evaluates the mapping with brute-force distributed ray tracing. Polynomial models approximate the mapping with polynomials.

We define an IRF $\Phi$ to express the above mapping. The IRF is designed in terms of the vector representation of an incident ray $R(\mathbf{p}, \mathbf{d})$, where $\mathbf{p}$ is the origin and $\mathbf{d}$ represents the unit direction vector along the ray. If dispersion of lens elements is considered, wavelength $\lambda$ can be included to obtain $\Phi(\mathbf{p}, \mathbf{d}, \lambda)$.

A camera lens is treated as a 'black box' module, which transforms an incident ray $R_i(\mathbf{p}, \mathbf{d})$ to an emergent ray $R_e(\mathbf{p}', \mathbf{d}')$ with: $R_e(\mathbf{p}', \mathbf{d}') = \Phi(R_i(\mathbf{p}, \mathbf{d}))$. Rays may enter a camera lens from either the sensor side (forward ray tracing) or the scene side (backward ray tracing), therefore a pair of IRFs can be defined for the two cases.

Note that Fresnel reflection is not included in the NeuroLens. Rays involving an odd number of Fresnel reflections, will revert its direction and then exit from the entrance or hit the lens barrel, leading to discontinuities in $\Phi$. Similarly, rays involving an even number of Fresnel reflection do not contribute to normal imaging, but produce lens flare. In addition, stochastic scattering and diffraction are not included, because the IRF is defined on deterministic imaging properties.

## 4. NeuroLens Model for Imaging

In this section, we firstly introduce the neural network IRF to represent the mapping of image formation. Then, we describe the detailed design of a NeuroLens. After that, we illustrate the supervised training procedure, focusing support and cascaded design. Lastly, we depict the Jacobian computation of a NeuroLens.

### 4.1. Neural network representation for IRF

According to Gaussian optics, the mapping of image formation has a linear form in the paraxial region of an ideal optical system. However, the paraxial region is theoretically defined as an infinitesimal area. The mapping in most off-axis regions is non-linear because of the inherent aberrations of an optical system.

We utilize neural networks to represent non-linear IRFs. Neural networks with one or two hidden layers can approximate any continuous and square-integrable function with desired accuracy provided sufficient network size and training samples [HSW89]. Besides, neural networks have a compact form and are efficient to evaluate in parallel. To obtain a representation of $\Phi$, we need to decide the proper structure of neural networks.

**Neural network structure.** We adopt the acyclic multi-layer perceptron model. It can be abstracted as a directed and weighted graph (shown in Figure 2), whose nodes are organized into several layers. Nodes within two adjacent layers are fully connected by weighted edges. Input layer comprises nodes representing each element of an input vector $(\mathbf{p}, \mathbf{d}, \lambda)$. Output layer consists of nodes corresponding to components of an output vector $(\mathbf{p}', \mathbf{d}')$. Intermediate layers are called hidden layers which contain adaptable number of nodes.

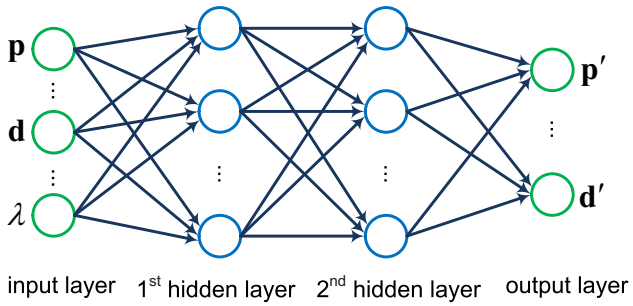


**Figure 2:** *Two-dimensional directed graph for a neural network with two hidden layers. Nodes in adjacent layers are connected with directed edges.*

We represent an incident ray with $R_i = (x, y, u, v, w, \lambda)$. Here, $x$ and $y$ correspond to Cartesian coordinates of the origin. Note that the $z$ component of the origin is omitted, because all incident rays starting from the same plane with equal $z$. $(u, v, w)$ is the unit direction vector of $R_i$, and $\lambda$ stands for a wavelength within the visible spectrum. If the spectral effect is not needed, $\lambda$ can be removed from $R_i$. Similarly, we represent an outgoing ray with $R_e = (x', y', u', v', w')$, where $(x', y')$ and $(u', v', w')$ represent the origin and direction vector, respectively. We move origins of all outgoing rays to the plane with $z = 0$ (Figure 1), and therefore $z'$ is omitted. Fluorescence and phosphorescence rarely occur in a camera, thus the wavelength of an outgoing ray remains the same as the incident ray and $\lambda$ is not included in $R_e$. It is a rule-of-thumb to reduce the dimension of output vectors, because it helps to reduce the inference work conducted by neural networks.

As to the parametrization of a ray's direction, we test the angular form $(\alpha, \theta)$, the light field form $(m, n)$, the normalized form $(u, v, w)$ and its reduced form $(u, v)$ . For the same training data, the $(u, v, w)$ empirically gives lower relative prediction error than others, thus $(u, v, w)$ is adopted in the paper.

Layers are indexed from 1 to $N$, starting from the input layer. We denote the $i$-th node in the $k$-th layer as $n_i^k$. The node count of the $k$-th layer is $M^k$. $n_i^k$ $(k > 1)$ receives inputs from all nodes in the preceding layer. The output $Y_i^k$ of the node is calculated by imposing a transfer function $\sigma_k$ to the weighted sum of all the inputs.

$$Y_i^k = \sigma_k(S_i^k); \qquad S_i^k = \sum_{j=1}^{M^{k-1}} w_{ij}^k \cdot Y_j^{k-1} + w_{i0}^k. \tag{1}$$

Here, $S_i^k$ stands for the weighted sum. $w_{ij}^k$ is the weight of connection from $n_j^{k-1}$ to $n_i^k$. $Y_j^{k-1}$ is the output of node $n_j^{k-1}$ in the $(k-1)$-th layer. $w_{i0}^k$ is a bias term. $\sigma_k$ is a non-linear transfer function, to which the non-linear property of neural networks is mainly attributed. We use a hyperbolic tangent function $\sigma_k = 2/(1 + e^{-2t}) - 1$. It is of similar shape as the logistic function $\sigma_k = 2/(1 + e^{-t})$, but it is symmetric about the origin. For the output layer, the output $Y_i^k$ of a node $n_i^k$ is equal to the weighted sum of all its inputs, that is, $\sigma_k = 1$.

In this paper, we use neural networks with two hidden layers. The node count $M$ of a hidden layer is controlled by a threshold $\Theta$.

For simplicity, each hidden layer has equal number of nodes. We empirically decide $M$ by starting from $M = 4$ and increase it if the relative prediction and testing error of neural networks exceeds $\varepsilon$ (0.2–0.8%). Neural networks with more than two hidden layers are seldom used for fitting functions, for it is practically difficult to train them. While multi-layer feed-forward network with one hidden layer can approximate many functions with any degree of accuracy, a large number of nodes may be needed to represent complex functions. Unfortunately, neural networks with too many nodes are prone to over-fitting [HYPI*12]. Generally, neural networks with two hidden layers can approximate a function with fewer nodes than using one hidden layer.

**NeuroLens training** The neural network representation of an IRF can be expressed as $\Phi(R, \mathbf{W})$, where $R$ represents an input ray and $\mathbf{W}$ is the weight matrix of a neural network. Given a training data set of $\Omega$ training samples $\{(R^i, T^i) \mid i = 1 \cdots \Omega\}$, where $R^i$ is the input vector and $T^i$ is the corresponding output vector, we determine $\Phi(R, \mathbf{W})$ by regulating $\mathbf{W}$ to minimize the following loss function:

$$E = \sum_{i=1}^{\Omega} \parallel T^i - \Phi(R^i, \mathbf{W}) \parallel^2 . \tag{2}$$

The loss function measures the sum of squared errors between outputs of $\Phi$ and target values.

Training samples are generated by performing distributed ray tracing in a spectral full lens model. We use Latin hypercube sampling [MBC00] to generate stratified ray samples. It is beneficial to use stratified training samples across the sampling space to capture the imaging properties as complete as possible. Besides, Neural network training converges more quickly on training samples with low correlations. We train neural networks on the training data set with Levenberg–Marquardt (LM) algorithm [HM94], in which weight matrices are updated after all training samples have been presented for once.

In rendering, a trained NeuroLens can be directly integrated into a renderer as an independent module. This module simply receives incident rays and outputs outgoing rays which enter the scene subsequently.

### 4.2. Focusing support

Focusing is a basic function of real cameras and it is usually achieved by adjusting the position of several lens elements. There are no lens elements in a NeuroLens, so we achieve focusing by moving the sensor. Shifting the sensor leads to the change of focal distance, and therefore the change of depth of field. In Figure 3, if the sensor is placed at $\Pi_3$, rays from $A$ form a blurry pattern (i.e. bokeh) on sensor. After moving the sensor to $\Pi_0$, these rays converge to a focused point on $\Pi_0$.

As to typical imaging configuration, the sensor is located at a distance more than the focal length from the rear pupil. For the NeuroLens, we use a fixed sensor plane located at the focal point of image space. The NeuroLens is trained on rays traversing two fixed planes $\Pi_1$ and $\Pi_2$ (Figure 3). After shifting the sensor, ray propagation in free space is calculated between the sensor plane and $\Pi_1$.

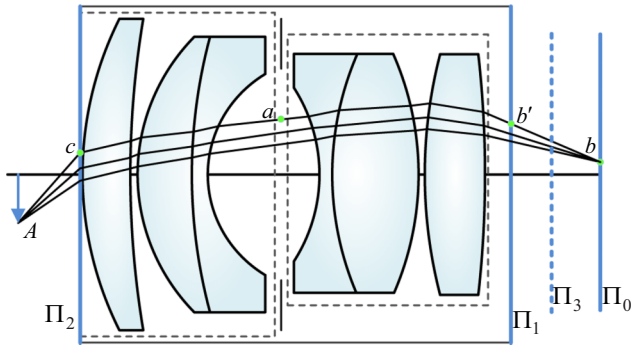


**Figure 3:** *Focusing is supported by adjusting the position of sensor plane, without changing the NeuroLens. A plane/plane light transport is implemented from b to b′.*

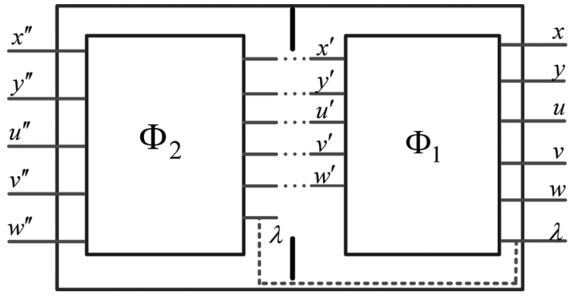


**Figure 4:** *To handle the discontinuities caused by aperture stop, a two-stage cascaded structure is designed. An output ray of $\Phi_1$ is sent to $\Phi_2$ as the input, if it passes the aperture stop.*

### 4.3. Cascaded NeuroLens

Given an incident ray of a camera lens, it can be blocked by the aperture stop. Blocked incident rays give no outgoing rays, thus introducing discontinuities in the IRF. To fit the discontinuities caused by the aperture stop, neural networks with more complex structure and far more training samples have to be used. Instead, we propose to use two-stage cascaded NeuroLenses (Figure 4) and handle the aperture stop independently.

In particular, we separately construct a NeuroLens for lens elements on each side of the aperture stop. The output of $\Phi_1$ is used as the input of $\Phi_2$. The discontinuities caused by the aperture stop are accurately handled using simple ray–plane intersection test. As the aperture stop is separately considered, various shapes of the aperture stop can be applied. Similarly, camera lenses with more than one aperture stop can be handled with multi-stage cascaded NeuroLenses as above.

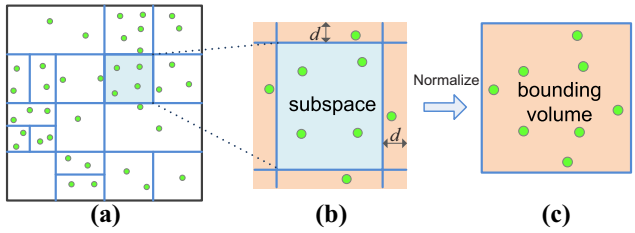


**Figure 5:** *(a) A 2D example of kd-tree subdivision scheme. The bounding volume for collecting training samples extends each dimension of the subspace (b) for d, and it is further normalized to a unit hypercube (c) before training.*

### 4.4. Input space subdivision and local fitting

Because of the inherent optical properties of lens elements, the images of most objects in the scene do not coincide with their ideal (Gaussian) images. The deviation from the ideal images is dubbed aberrations, including five Seidel aberrations (spherical aberration, comet, astigmatism, field of curvature and distortion) and chromatic aberrations.

The distribution of aberrations is a function of field of view and aperture size. If a camera lens is represented with a single neural network, a high number of nodes will be needed to fit the heterogeneous imaging properties. Neural networks with many nodes, however, spend longer time in both training and evaluation. Besides, over-fitting is likely to happen if training samples are not sufficient.

To cope with the above issue, we partition the six-dimensional input space constituted by all input vectors into multiple non-overlapping subspaces. All subspaces are organized by a kd-tree. Non-leaf nodes store splitting axes and the position of splitting plane, and leaf nodes represent local subspaces. Each subspace is locally fitted with a separate IRF.

Initially, the entire input space $\Psi$ is set as the root node of kd-tree. We firstly determine the node count $M$ for hidden layers using the method in Section 4.1. If $M$ exceeds $\Theta$, the root node is subdivided. Then the kd-tree is constructed in a top-down approach. A 2D subdivision scheme is shown in Figure 5. A node space is recursively subdivided into two subspaces, if the relative training error of its IRF with $M = \Theta$ is larger than a threshold (0.2–0.8%). Before a subdivision operation, we try each splitting axis in turn and select the one which leads to the lowest relative training error and testing error in child nodes. The splitting plane is chosen at the midpoint of a splitting axis.

The IRF for a leaf node is trained on local training samples. Eighty percent samples are used for training, and the rest is used for testing. To alleviate the boundary discontinuity caused by non-overlapping subdivision, local training samples are collected in a slightly larger bounding volume which extends each dimension of a subspace by 10% (Figure 5c).

### 4.5. Neural network ensemble

NeuroLenses are trained with LM algorithm. While it converges fast, it is not bound to find the globally optimum weight matrix **W** of the neural network. There are many local minima on the surface of the loss function (Equation 2). Training process may trap at local minima of the surface. Neural networks which fall into local optima provide suboptimal results.

To alleviate the influence of a single neural network which falls into a local optimum, we leverage a neural network ensemble [HS90] to model an IRF. A neural network ensemble consists of a set of neural networks. A neural network in the ensemble is independently trained on a different subset of the training data. The output of an IRF is set as the average result of a neural network ensemble. With the averaging operation, the deviated output of a single neural network which converges to a local optimum can be effectively corrected.

For each subspace, we fit a neural network ensemble to model the local IRF. We empirically find that an ensemble size between three and five works well for all camera lenses tested in this paper.

**Table 1:** *Fitting statistics of four camera lenses: single bi-convex, DGauss F/1.35, wide-angle F/3.4 and fisheye F/7.7 [S*05].*

| Camera lenses | Lens count | Ensemble size | $\Theta$ | Storage (KB) | Training time (h) |
|---|---|---|---|---|---|
| Bi-convex | 2 | 3 | 6 | 2.78 | 1 |
| DGauss | 11 | 3 | 10 | 46.26 | 3 |
| Wide-angle | 14 | 3 | 12 | 115.38 | 6 |
| Fisheye | 8 | 4 | 16 | 310.92 | 7.5 |

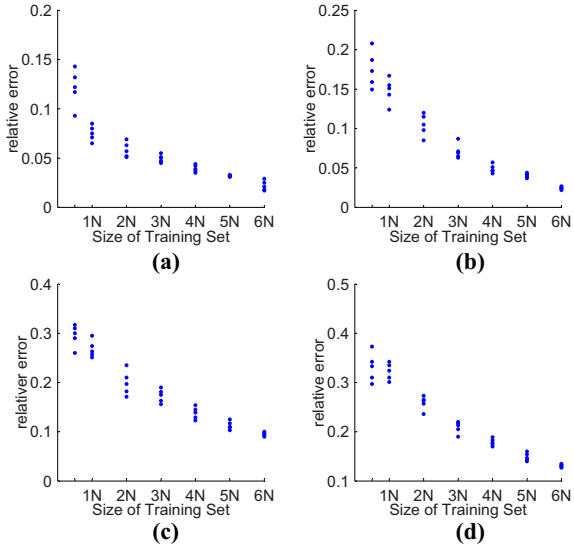


**Figure 6:** *Relative testing error for training four cameras lenses: (a) single bi-convex lens, (b) DGauss F/1.35 lens, (c) wide-angle F/3.4 lens and (d) fisheye F/7.7 lens. $N = 512^2$ is the total pixel count.*

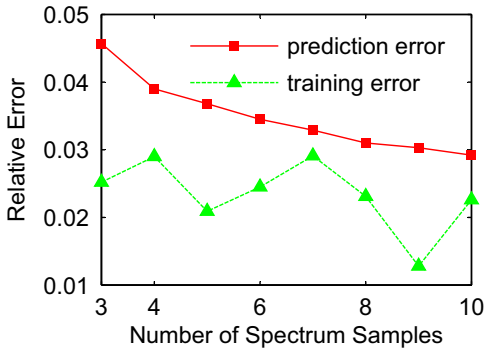


**Figure 7:** *Analysis of how the relative training error and testing error are affected by the spectral sampling rates. The prediction error goes down as the rate increases.*

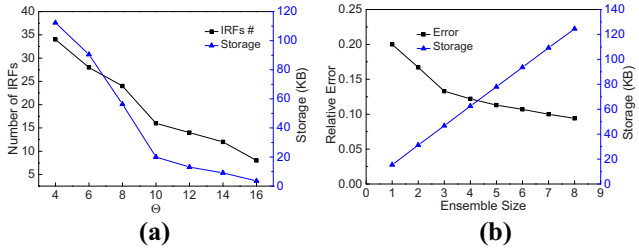


**Figure 8:** *(a) Analysis of kd-tree subdivision with respect to Θ and (b) analysis of the relative error and storage as to ensemble size.*

**Table 2:** *Accuracy and speed comparison. The testing data set contains 1.17M valid rays. For the Taylor polynomial model, plain for-loop evaluation is implemented, whereas compiled native code is used for the fitted polynomial model. The ensemble size of the NeuroLens is 3. As to the implementation of NeuroLenses on GPU, a thread block handles 1024 rays. The timing is given in seconds.*

| Camera | Taylor polynomial | Fitted polynomial | NeuroLens | NeuroLens (GPU) |
|---|---|---|---|---|
| Bi-convex | 0.228/21.1 | 0.213/17.1 | 0.195/17.9 | 0.191/1.6 |
| DGauss | 0.325/27.4 | 0.255/17.9 | 0.156/49.2 | 0.162/4.1 |
| Wide-angle | 0.478/32.7 | 0.327/18.6 | 0.193/43.2 | 0.189/5.6 |
| Fisheye | 0.768/26.8 | 0.573/18.1 | 0.283/57.4 | 0.303/6.9 |

### 4.6. Jacobian calculation

Monte Carlo ray tracing methods compute a pixel measurement by integrating a measurement contribution function over the path space. If the first vertex of a path is set on the sensor (e.g. $b$ in Figure 3), the measurement contribution function can be defined as

$$f = W_e(b)G(b\leftrightarrow c)L(c, \text{-}\omega_i). \tag{3}$$

Here, $L$ represents the incident radiance at $c$ along a direction $\omega_i$, and $W_e$ is the importance. $G(b\leftrightarrow c)$ is a generalized geometric factor between $b$ and $c$, and it can be computed from the transport Jacobian, which consists of partial derivatives. We derive the calculation of partial derivatives and Jacobian in the Appendix.

In a camera lens, several lens elements are located between the sensor and the aperture. To implement aperture-based importance sampling, Newton iteration method [HD14] can be applied to iteratively adjust the direction of the initial ray such that it goes through the aperture. Partial derivatives of the IRF to the direction components are computed with $\mathrm{d}\Phi/\mathrm{d}\{u, v\}$ (the Appendix). In the NeuroLens, partial derivatives of an IRF and Jacobian can be simultaneously computed in the feed-forward process.

## 5. Implementation Details

**Training details.** As suggested in [TF95], the proper size of a training data set is usually more than 10 times the number of weights. We assign the size as the product of samples per pixel (*spp*) and pixel count. For an image with fixed resolution, we adjust the *spp* to obtain enough samples. Prior to training, all training samples are firstly standardized with zero mean and unit variance, and they are further normalized to a range $[-1, 1]$. The pre-processing step is adopted because it helps to accelerate the convergence of training. All weights and bias values are initialized with random values in $(-1, 1)$.

We train the neural network with LM algorithm in a batch-mode. Weights are updated after all training samples have been presented for once [HM94]. As with a neural network ensemble, each network is trained on a separate subset of samples which are randomly selected from the training data. In this paper, we use 80% samples for training and the remaining 20% for testing.

**IRF evaluation.** The NeuroLens is designed as an independent module which can be easily integrated in a renderer. We test a trained NeuroLens in a hybrid mode of both CPU and GPU. The generation of initial camera rays and subsequent radiance estimation in the scene are conducted with eight threads on CPU (one thread per ray). The evaluation of a NeuroLens is implemented with CUDA on GPU. The input vectors of eight rays constitute a matrix $\Gamma$. For an initial ray, the traversing kernel on CUDA firstly searches the kd-tree down to a leaf node. A GPU thread processes one column (a ray) of $\Gamma$ and returns a set of neural network identifiers. For a neural network evaluation kernel, each GPU thread feeds forward one element of the input vector and evaluates the output.

**Rays clipping.** The NeuroLens fit a camera lens based on valid rays which can pass through all lens elements. The restriction of aperture stop is separately handled by accurate ray tracing. To further reduce the invalid rays which are blocked by stops of other lens elements, lens barrel and lens hood, we clip rays at both the rear pupil and the front pupil.

## 6. Results

The neural network training is implemented on CPU and rendering is implemented in a hybrid mode with both CPU multi-threads and GPU. All experiments are conducted on a workstation with a dual quad-core 2.4 GHz Intel Xeon CPU and Nvidia Quadro 6000 graphics card.

We simulate four typical camera lenses using the proposed NeuroLens. The settings and statistics for fitting are listed in Table 1. The number of IRFs mainly depends on both the imaging properties and the complexity of each neural network.

### 6.1. Method validation

In this section, we firstly validate the robustness of the training method. As for spectral rendering, we analyse the training error and testing error as to the spectral sampling rate. Subsequently, we analyse how the subdivision is affected by $\Theta$. In the above analysis, the ensemble size of each leaf node is 1. Following that, we validate the use of neural network ensembles. Finally, we compare the evaluation accuracy and performance of the NeuroLens with two polynomial models.

**Training method.** The training of neural networks involves the selection of training samples, initial conditions and the size of training data set. To examine how the training method behaves with different training data sets, initial weights and selection of training samples, we fit each camera lens in Table 1 for five times. Each time 80% samples are randomly selected from the training data, and neural works are randomly initialized with weights from $(-1, 1)$. Figure 6 plots the relative testing error with respect to training sets of different sizes. The relative testing error is calculated by

$$\epsilon = \sqrt{\frac{\sum_{i=1}^{\Omega} \parallel \Phi(R^i, \mathbf{w}) - T^i \parallel^2}{\sum_{i=1}^{\Omega} \parallel T^i \parallel^2}}. \tag{4}$$

The deviation of points on a column in Figure 6 characterizes the variance. For the same training data set, the training method is not sensitive to the selection of training samples and initial weights. As the size of training data set increases, both the relative testing error and variance decrease.

**Spectral sampling rate.** The accuracy of spectral ray tracing in a camera lens is mainly affected by the number of discrete spectrum samples. In this paper, we use equidistant wavelength samples within the visible spectrum range. Figure 7 plots the testing and training error of IRFs as to the spectral sampling rates. It indicates that the training error is not sensitive to the spectral sampling rate, because training may stop early once the preset error threshold is achieved. The prediction error drops as the spectral sampling rate increases, but the line gets stable after the rate exceeds 7. As higher spectral sampling rate leads to more training samples and thus longer training time, we set it to 6 in this paper.

**Subdivision granularity.** With an error-driven partition mode, the subdivision granularity is directly affected by the complexity of the neural network used for each subspace. The complexity of a neural network is restricted by $\Theta$. Figure 8(a) illustrates how the number and storage of neural networks change with $\Theta$. The camera lens used here is the DGauss F/1.35 and the ensemble size is 1. With a small $\Theta$, the kd-tree is partitioned into many leaf nodes, therefore requiring larger storage space. By contrast, a larger $\Theta$ results in fewer leaf nodes. We find that $\Theta$ between 6 and 16 generally works well for all camera lenses in the experiments.

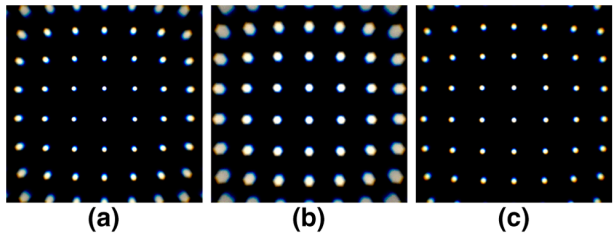


**Figure 9:** *(a) Focus is set at the plane of dots and central dots get focused. (b) Polygonal bokeh is caused by the clipping with a hexagonal aperture stop. (c) After reducing the aperture size by half, blur and aberrations can be greatly reduced.*

**Neural network ensemble.** Neural network ensemble is helpful to mitigate the influence of a single neural network which traps in a local optimum. Figure 8(b) plots the relative testing error and model storage with respect to different ensemble sizes. The reduction of relative testing error justifies the use of neural network ensemble. Yet, the improvement of evaluation accuracy gradually fades after the ensemble size exceeds 5. Meanwhile, the storage size rises almost linearly as the ensemble size increases. Taking into account the evaluation efficiency, a relatively small ensemble size is preferred. We set the ensemble size as 3–5 in the experiments.

**Accuracy and efficiency comparison.** We fit the above four camera lenses and compare the NeuroLens with the Taylor polynomial model [HHH12] and the fitted polynomial model [HD14] under the same conditions. All models are implemented with one thread on CPU, and are evaluated with the same testing data. The statistics are tabulated in the central three columns of Table 2. In each column, the left side is the relative testing error, and the right is the timing for generating rays from these models. For the bi-convex lens, all the three models are able to robustly fit it with similar low error. For the complex 'fisheye' lens, the NeuroLens yields lower testing error. The NeuroLens costs more time than others because of the lookup in kd-tree and evaluation of neural network ensembles, but it pays back with low testing error. Implementing neural networks evaluation and kd-tree lookup on GPU provides a speed-up (see columns 4 and 5 in Table 2).

### 6.2. Rendering results

In this part, we test the NeuroLens by rendering various aberrations, simulating focusing, and comparing with two state-of-the-art polynomial models [HHH12, HD14]. For polynomial models, we utilize the suggested settings from the original papers. All scenes are rendered with LuxRender [lux13] at a resolution of $512 \times 512$ pixels. As to the aberration scene, path tracing [Kaj86] is applied. The rest scenes are rendered by the stochastic progressive photon mapping (SPPM) [HJ09] for its robustness to handle complex global illumination.

In Figure 9, we fit the single bi-convex lens with an aperture at f/4, whose aberrations are not corrected. The NeuroLens faithfully produces typical Seidel aberrations. When the focus is set at the plane of dots in Figure 9(a), the periphery dots and central dots cannot be focused at the same time. Colour shift around the dots

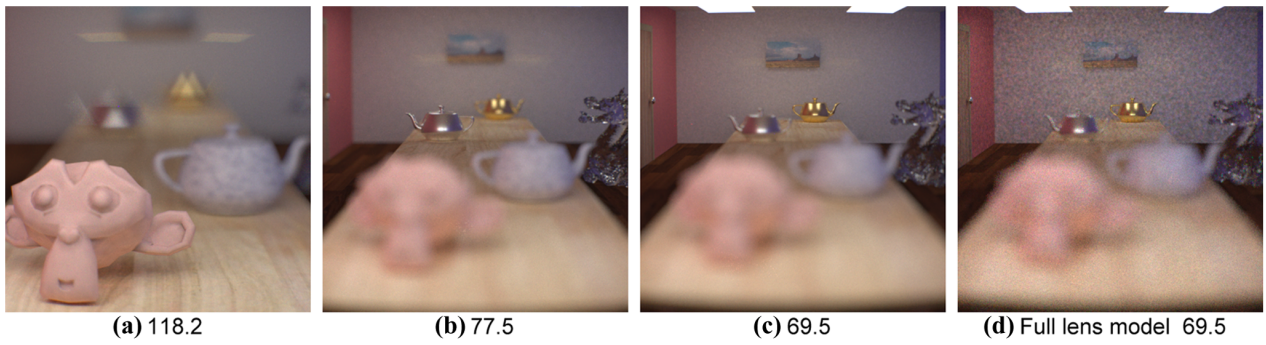


**Figure 10:** *(a–c) show the focusing effect via moving the sensor. The distance from sensor to the inner pupil is given below each image. All distances are in millimetres. (d) Equal time rendering result of a full lens model.*

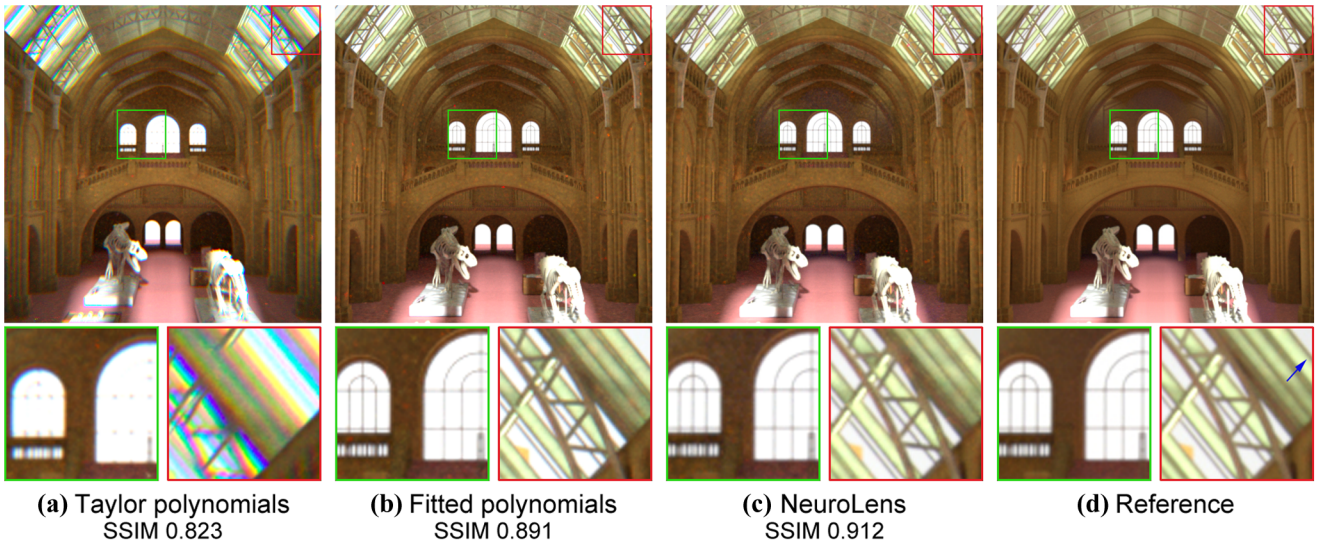


**Figure 11:** *Equal time comparisons (1.5 h). The reference image is rendered with a full lens model using 92 h. See the loss of details and false chromatic contouring in (a). Image shift can also be observed in (a) and (b) (see area indicated by blue arrow).*

is caused by dispersion of lens elements, which is called chromatic aberration. The polygonal shape of dots in Figure 9(b) is mainly affected by the clipping at the aperture stop. Figure 9(c) shows that stopping down the aperture generally reduces the overall blur and off-axis aberrations.

Figure 10 shows the room scene imaged with the DGauss F/1.35 lens. Via moving the sensor, the change of focal distance can be achieved (Figures 10 a–c). Note the triangular bokeh in the defocus region of Figure 10(a) is caused by a triangular aperture stop. Figure 10(d) gives the equal time result rendered with a full lens model, whose focusing configuration is the same as that in Figure 10(c). Because of the compact structure of the NeuroLens and parallel evaluation for camera rays, the NeuroLens behaves more efficient than the full lens model and produces a result with less noise.

The natural history museum scene in Figure 11 is photographed by the wide-angle F/3.4 lens. Here, comparisons focus more on the integrity of image content. The Taylor polynomial model loses details at the far-end windows, where the window frames almost disappear. Besides, there are noticeable chromatic stripes at the dormer (red inset of Figure 11a). Both polynomial models lead to overall shift of the image (see the red close-up view). In contrast, the NeuroLens robustly preserves important details and produces a result that is close to the reference. Here, we evaluate the image quality with structural similarity (SSIM)[WBSS04] instead of MSE, because image content of the results is not exactly the same.

Figure 12 shows the shots of the museum scene produced by Merte Muller F/7.7 lens [S*05], which is a challenging 'fisheye' lens with field of view of 160°. Note that, in no means can the thin lens model generate the "fisheye" effect. False chromatic contouring and blurriness can be observed in Figure 12(a) (see blue and green insets). Because of the extremely wide field of view, both polynomial models cannot reliably capture the complete imaging properties, causing decrease of field of view and evident loss of image content (see the red close-up view). Besides, overall radially

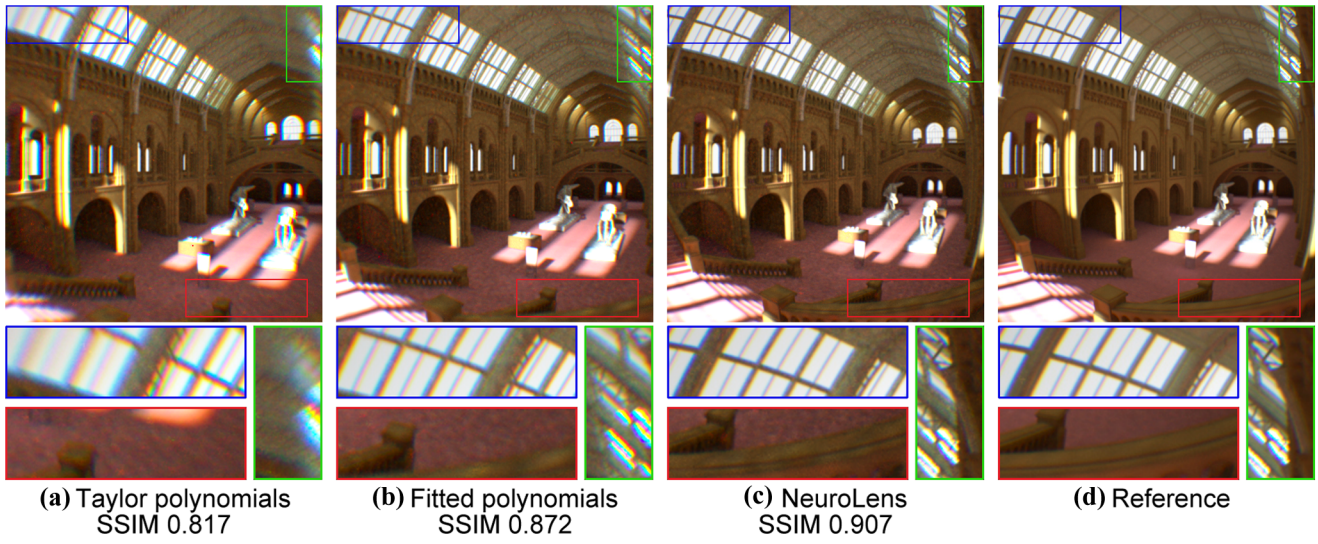


**Figure 12:** *Equal time comparisons (2 h). The reference image is rendered with a full lens model for 96 h. The overall radially expansion and loss of image content are shown in blue and green insets.*

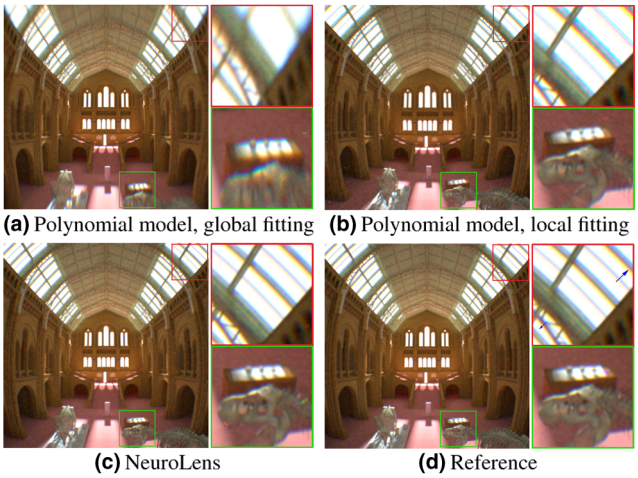


**Figure 13:** *Equal time comparisons (3 h) except for the reference image which is rendered for 40 h. (a) The polynomial system is globally fitted [HD14] with 30K rays. (b) A polynomial system is locally fitted for each subspace. The same input space subdivision and training data set containing 2.33M rays are used for both (b) and (c). Notice the 'fisheye' distortion (see the area indicated by blue arrow) is better reproduced by the NeuroLens.*

outward expansion can be observed in Figures 12(a) and (b). In contrast, the NeuroLens faithfully fit the camera lens and reproduce a result which is closer to the reference image.

Figure 13 compares the fitting of the Merte Muller fisheye F/7.7 lens. The model of Figure 13(a) is fitted with the method of [HD14], which uses several thousands of ray samples to globally optimize polynomial coefficients. We make an attempt of applying the input space subdivision and local fitting to polynomial model (Figure 13b). Each subspace is fitted with a 5×4 polynomial system. The same kd-tree subdivision scheme and training data are used for Figures 13(b) and (c). Coefficients of polynomials are optimized using Levmar [Lou15] with its default parameters. Compared with the reference image, the globally fitted polynomials cannot well fit the lens, exhibiting different distortions at the image boundary. The locally fitted polynomial model (Figure 13b) brings improvements and gives a better result than the globally fitted one, whereas the NeuroLens faithfully preserves the field of view and reproduces the distortion that is closer to the reference.

**Table 3:** *Comparison of different hidden layer count. The 2nd column records the minimum required neuron count for each hidden layer. The number of successful training is in the 4th column.*

| Hidden layers# | Min neurons# | Weights# | Success training# | Average iters# |
|---|---|---|---|---|
| 1 | 19 | 233 | 29 | 69.3 |
| 2 | 8 | 173 | 30 | 23.1 |
| 3 | 6 | 119 | 26 | 36.1 |

**Table 4:** *Efficiency comparison of Jacobian calculation. The ensemble size of the NeuroLens is 1. The testing data set contains 1.17M valid rays and the timing is given in seconds.*

| Camera | Hanika method | NeuroLens | NeuroLens (GPU) |
|---|---|---|---|
| Bi-convex | 16.9 | 19.7 | 2.6 |
| Fisheye | 17.8 | 29.6 | 4.7 |

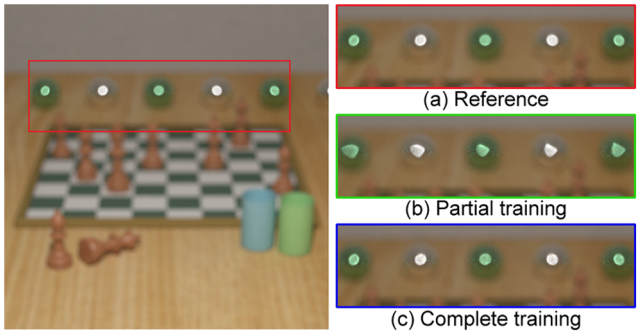


**Figure 14:** *Equal time comparison (2 h) between a partially trained NeuroLens (b) and a completely trained one (c). The reference image (left) is rendered for 30 h.*

## 7. Discussion

**The number of hidden layers.** We compare neural networks with 1 to 3 hidden layer(s) by fitting the DGauss F/1.35 lens with the same training data. Training is repeated 30 times. The training error threshold is 0.2% and the same kd-tree subdivision with four leaf nodes is used. The maximum allowed number of iterations is set to 200. Table 3 presents the statistics. As can be noticed, neural networks with two hidden layers provide higher successful training rate and cost fewer iterations. Therefore, we use two hidden layers in the experiments.

**Performance of Jacobian calculation.** Transport Jacobian consists of the derivatives of IRF to each element of an input vector. The calculation can be performed simultaneously in the feed-forward pass of an input vector (Appendix). We compare the performance of Jacobian evaluation between the NeuroLens and the fitted polynomial model using 1 thread on CPU. Polynomials are evaluated with compiled native code. As shown in the central two columns of Table 4, the performance of the fitted polynomial model for different camera lenses are similar. The NeuroLens of the 'fisheye' camera lens consumes more computation time, because it is represented with complex neural networks. Implementation of the NeuroLens on GPU reduces its time cost.

**Extension for Monte Carlo ray tracing.** In addition to SPPM, the NeuroLens can support general Monte Carlo ray tracing methods like bidirectional path tracing (BPT) and Metropolis light transport (MLT).

In BPT [LW93], if the first vertex of a camera subpath is on the front pupil, it involves no geometric term within the camera. If the first vertex is on the sensor, geometric term within the camera is required in the Monte Carlo estimator. The forward light tracing along light subpath is implemented in an adjoint NeuroLens fitted from the front pupil to the rear one. To reduce the discrepancy between a forward NeuroLens and a backward NeuroLens, we train the backward NeuroLens with an 'adjoint' training data set, which is built by reversing the directions of rays in the training data set used for training the forward one. This is a reasonable choice because light paths are generally reversible in an optical system.

MLT [VG97] tends to sample new paths in the proximity of paths with large contribution. Taking into account the dependence between a new path and the histories, it is better to assign a thread to each Metropolis sampler which independently runs a Markov chain.

**Limitations and future work.** NeuroLenses depend on the local coherence within each dimension of the input space. The subdivision of input space and localized training ensure the local coherence in most regions. But, the imaging properties which are not encoded in the training data can hardly be deduced by NeuroLenses. Training NeuroLenses from incomplete training data results in poor fit to the IRFs.

Figure 14 shows a chess scene imaged with two NeuroLenses. The first one is trained on paraxial rays (the angle between a ray and the optical axis is less than $10^{\circ}$), and the other one is trained on the complete training data set. It can be seen that the first model (Figure 14b) cannot recover the missing imaging properties and yields irregular bokeh patterns, whereas the second one (Figure 14c) learns more imaging properties and its result is closer to Figure 14(a). Notice that this case is also used to illustrate that dispersion can be switched off in a NeuroLens. The wavelength component is accordingly removed from the input layer.

The efficiency of NeuroLens evaluation on GPU depends on the pace of an incident ray bundle. When implementing BPT with a NeuroLens in a hybrid mode, bidirectional subpath connections may degenerate the throughput of GPU. In addition, MLT randomly accepts a candidate path, which may disrupt the pace of the ray bundle and reduce the GPU throughput. Designing robust GPU implementation of BPT and MLT compatible with the NeuroLens would be an interesting avenue for future work.

The evaluation of a NeuroLens on GPU is currently limited by the number of initial rays generated with CPU threads. While more CPU threads are expected to be used, the cost for managing and synchronizing CPU threads also rises with the thread count. We find it is a good trade-off to use eight threads in our experiments. To raise the number of initial rays, ray tracer implemented on GPU can be employed to accelerate the generation of initial rays.

The NeuroLens is defined in the geometrical optics sense, thus it cannot account for phenomenons of the physical optics, such as diffraction and interference. Also, the Fresnel reflection is not included, thus it cannot be directly applied to generate lens flare. A remedy method is to enumerate all possible lens flare light paths [HESL11] and fit a separate IRF for each path.

## 8. Conclusion

We have presented a new camera lens model, NeuroLens, using a data-driven approach. Our method exploits neural networks to approximate the locally coherent mapping between incident rays and emergent rays, which is formulate as an IRF. The NeuroLens learns the IRF from accurate ray samples. To ensure that the NeuroLens achieves high accuracy across the input space, we subdivide the input space into local subspaces and organize subspaces with kd-tree for fast lookup. Besides, we further improve the evaluation accuracy of the NeuroLens via neural network ensembles. Compared with other camera lens models, the NeuroLens is able to provide higher

imaging accuracy for a wide range of real camera lenses and yields images which are closer to the reference ones.

**Acknowledgements**

We acknowledge the anonymous reviewers for their constructive suggestions and comments. We thank Alvaro Luna Bautista and Joel Andersdon for the natural history scene.

**Appendix: Partial Derivatives and Jacobians Calculation for the Neural Networks**

We set the bias input to 1, and $w_{i0}^k$ is the weight of the bias input. Equation (1) can be rewritten as

$$Y_i^k = \sigma_k\left(S_i^k\right); \qquad S_i^k = \sum_{j=0}^{M^{k-1}} w_{ij}^k \cdot Y_i^{k-1}. \tag{A1}$$

Here, we take a neural network with four layers as an example (Figure 2). The $i$-th output in the output layer, $Y_i^4$, is calculated with

$$Y_i^4 = \sigma_4\left(\sum_{j=0}^{M^3} w_{ij} \cdot \sigma_3\left(\sum_{r=0}^{M^2} w_{jr} \cdot \sigma_2\left(\sum_{t=0}^{M^1} w_{rt} \cdot Y_t^1\right)\right)\right). \tag{A2}$$

Then, the partial derivative of $Y_i^4$ to input $Y_t^1$ can be computed with

$$\begin{aligned}\frac{\partial Y_i^4}{\partial Y_t^1} &= \sigma_4'\left(S_i^4\right) \cdot \frac{\partial S_i^4}{\partial Y_t^1} \\ &= \sigma_4'\left(S_i^4\right) \cdot \left(\sum_{j=0}^{M^3} w_{ij} \cdot \sigma_3'\left(S_j^3\right) \cdot \frac{\partial S_j^3}{\partial Y_t^1}\right) \\ &= \sigma_4'\left(S_i^4\right) \cdot \left(\sum_{j=0}^{M^3} w_{ij} \cdot \sigma_3'\left(S_j^3\right) \cdot \sum_{r=0}^{M^2}\left(w_{jr} \cdot \sigma_2'\left(S_r^2\right) \cdot w_{tr}\right)\right).\end{aligned}$$

The Jacobian $\mathbf{J}$ can be constructed by taking derivatives of output to every component of the input vector as $dY_{\{1,\cdots,M^4\}}^4 / d\{Y_1^1, \cdots, Y_{M^1}^1\}$. To support the implementation of the NeuroLens on GPU, it is useful to express $\mathbf{J}$ in the form of matrix multiplication:

$$\mathbf{J} = \mathbf{D}_{\sigma_4}\mathbf{W}_{34}\mathbf{D}_{\sigma_3}\mathbf{W}_{23}\mathbf{D}_{\sigma_2}\mathbf{W}_{12}. \tag{A3}$$

Here, $\mathbf{W}_{ij}$ stands for the matrix of weights from layer $i$ to layer $j$ and they are known after the training step. $\mathbf{D}_{\sigma_k}$ $(k = 2, 3, 4)$ is a diagonal matrix:

$$\mathbf{D}_{\sigma_k} = \begin{pmatrix} \sigma_k'(S_1^k) & & \\ & \ddots & \\ & & \sigma_k'(S_{M^k}^k) \end{pmatrix}, \tag{A4}$$

where $\sigma_2$ and $\sigma_3$ are hyperbolic tangent functions and their derivatives are $\sigma_k'(S_i^k) = 1 - (\sigma_k(S_i^k))^2$. $\sigma_4$ is a linear function and its derivative is 1. Note that $\mathbf{W}_{ij}$ is known and $\sigma_k(S_i^k)$ can be calculated in the feed-forward process. As a result, the calculation of $\mathbf{J}$ can be carried out along with the feed-forward of an input vector.

**References**

[CPC84] Cook R. L., Porter T., Carpenter L.: Distributed ray tracing. In *ACM SIGGRAPH Computer Graphics* (1984), vol. 18, ACM, pp. 137–145.

[Dac11] Dachsbacher C.: Analyzing visibility configurations. *IEEE Transactions on Visualization and Computer Graphics 17*, 4 (2011), IEEE, pp. 475–486.

[GRB*13] Grimaldi J. -P., Romang J. -F., Bech T., Bienkowski P., et al.: 2013. http://scr.luxrender.net.

[GTH98] Grzeszczuk R., Terzopoulos D., Hinton G.: Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of SIGGRAPH '98* (New York, NY, USA, 1998), ACM, pp. 9–20.

[HD14] Hanika J., Dachsbacher C.: Efficient Monte Carlo rendering with realistic lenses. In *Computer Graphics Forum (Proceedings of EG)* (2014), vol. 33, pp. 323–332.

[HESL11] Hullin M., Eisemann E., Seidel H.-P., Lee S.: Physically-based real-time lens flare rendering. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2011)* (2011), vol. 30, ACM, 108:1–108:9.

[HHH12] Hullin M. B., Hanika J., Heidrich W.: Polynomial optics: A construction kit for efficient ray-tracing of lens systems. In *Computer Graphics Forum (Proceedings of EGSR)* (2012), Blackwell Publishing Ltd, vol. 31, pp. 1375–1383.

[HJ09] Hachisuka T., Jensen H. W.: Stochastic progressive photon mapping. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH ASIA 2009)* (New York, NY, USA, 2009), vol. 28, ACM, 141:1–141:8.

[HM94] Hagan M. T., Menhaj M. B.: Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks 5*, 6 (1994), 989–993.

[HS90] Hansen L. K., Salamon P.: Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence 12* (1990), 993–1001.

[HSW89] Hornik K., Stinchcombe M., White H.: Multilayer feedforward networks are universal approximators. *Neural Networks 2*, 5 (1989), 359–366.

[HYPI*12] Hunter D., Yu H., Pukish III M. S., Kolbusz J., Wilamowski B. M.: Selection of proper neural network sizes and architectures: A comparative study. *IEEE Transactions on Industrial Informatics 8*, 2 (2012), 228–240.

[Kaj86] KAJIYA J. T.: The rendering equation. In *ACM Siggraph Computer Graphics* (New York, NY, USA, 1986), vol. 20, ACM, pp. 143–150.

[KBS15] KALANTARI N. K., BAKO S., SEN P.: A machine learning approach for filtering monte carlo noise. *ACM Transactions on Graphics (Proc. SIGGRAPH 2015) 34*, 4 (New York, NY, USA, 2015), ACM, pp. 122:1–122:12.

[KMH95] KOLB C., MITCHELL D., HANRAHAN P.: A realistic camera model for computer graphics. In *Proceedings of SIGGRAPH '95* (Los Angeles, CA, USA) (1995), ACM, pp. 317–324.

[LE13] LEE S., EISEMANN E.: Practical real-time lens-flare rendering. In *Computer Graphics Forum (Proceedings of EGSR)* (2013), vol. 32, pp. 1–6.

[Lou15] LOURAKIS M.: levmar: Levenberg-Marquardt nonlinear least squares algorithms in C/C++. http://www.ics.forth.gr/~lourakis/levmar/, Aug. 2015.

[LW93] LAFORTUNE E. P., WILLEMS Y. D.: Bi-directional path tracing. In *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics' 93)* (1993), Harold P Santo (Ed.), pp. 145–153.

[MBC00] MCKAY M. D., BECKMAN R. J., CONOVER W. J.: A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics 42*, 1 (2000), 55–61.

[NKF09] NOWROUZEZAHRAI D., KALOGERAKIS E., FIUME E.: Shadowing dynamic scenes with arbitrary BRDFs. In *Computer Graphics Forum* (2009), vol. 28, pp. 249–258.

[PC81] POTMESIL M., CHAKRAVARTY I.: A lens and aperture camera model for synthetic image generation. *ACM SIGGRAPH Computer Graphics 15*, 3 (New York, NY, USA, 1981), ACM, 297–305.

[RDL*15] REN P., DONG Y., LIN S., TONG X., GUO B.: Image based relighting using neural networks. *ACM Transactions on Graphics (Proc. SIGGRAPH 2015) 34*, 4 (New York, NY, USA, 2015), ACM, pp. 111:1–111:12.

[RWG*13] REN P., WANG J., GONG M., LIN S., TONG X., GUO B.: Global illumination with radiance regression functions. *ACM Transactions on Graphics (Proc. SIGGRAPH 2013) 32*, 4 (New York, NY, USA, 2013), ACM, pp. 130:1–130:12.

[Smi05] SMITH W. J.: *Modern Lens Design*, 2 ed. McGraw-Hill, New York, 2005.

[SDHL11] STEINERT B., DAMMERTZ H., HANIKA J., LENSCH H. P.: General spectral camera lens simulation. In *Computer Graphics Forum* (2011), vol. 30, pp. 1643–1654.

[SHD16] SCHRADE E., HANIKA J., DACHSBACHER C.: Sparse high-degree polynomials for wide-angle lenses. In *Computer Graphics Forum (Proceedings of EGSR)* (2016), vol. 35, pp. 89–97.

[SWWW15] SONG Y., WANG J., WEI L.-Y., WANG W.: Vector regression functions for texture compression. *ACM Transactions on Graphics (TOG) 35*, 1 (New York, NY, USA, 2015), ACM, pp. 5:1–5:10.

[TF95] TURMON M. J., FINE T. L.: Sample size requirements for feedforward neural networks. *Advances in Neural Information Processing Systems* (1995), 327–334.

[VG97] VEACH E., GUIBAS L. J.: Metropolis light transport. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1997), ACM Press/Addison-Wesley Publishing Co., pp. 65–76.

[WBSS04] WANG Z., BOVIK A. C., SHEIKH H. R., SIMONCELLI E. P.: Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (2004), 600–612.

[WZHX13] WU J., ZHENG C., HU X., XU F.: Rendering realistic spectral bokeh due to lens stops and aberrations. *The Visual Computer 29*, 1 (2013), Springer, pp. 41–52.