Neural Cameras: Learning Camera Characteristics for Coherent Mixed Reality Rendering

David Mandl*
Graz University of Technology

Peter Mohr
VRVis Research Center

Tobias Langlotz
University of Otago

Christoph Ebner
Graz University of Technology

Shohei Mori
Graz University of Technology

Stefanie Zollmann
University of Otago

Peter M. Roth
Technical University of Munich

Denis Kalkofen
Graz University of Technology

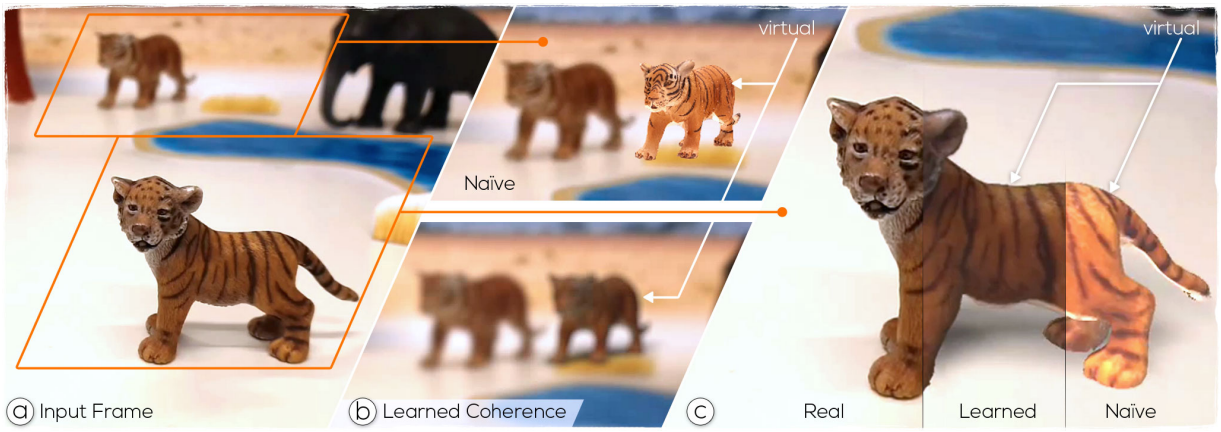

Figure 1: Learned visual coherence for simulating the characteristics of physical cameras. (a) Input frame showing a scene capture with a physical camera. (b, top) Inserting a virtual object (the tiger on the right side) using a naïve rendering introduces a break-in visual coherence. (b, bottom) We address this gap with a Neural Camera which is able to reproduce characteristics such as blur and color filtering. (c) A direct comparison between a naïve rendering of a virtual object (right-Naïve), our approach (middle-Learned), and to the output of a real camera (left-Real).

Abstract

Coherent rendering is important for generating plausible Mixed Reality presentations of virtual objects within a user's real-world environment. Besides photo-realistic rendering and correct lighting, visual coherence requires simulating the imaging system that is used to capture the real environment. While existing approaches either focus on a specific camera or a specific component of the imaging system, we introduce Neural Cameras, the first approach that jointly simulates all major components of an arbitrary modern camera using neural networks. Our system allows for adding new cameras to the framework by learning the visual properties from a database of images that has been captured using the physical camera. We present qualitative and quantitative results and discuss future direction for research that emerge from using Neural Cameras.

1 Introduction

Mixed Reality (MR) blends the real-world environment with 3D computer-generated graphics. Azuma [1] highlights the need for real-time rendering and precise spatial registration. However, many MR applications additionally require a high level of visual coherence that makes it difficult to differentiate real from virtual scene elements [17]. Naïvely rendering virtual content often reveals its artificial nature and thus, easily breaks the illusion of a single MR environment [31].

A great deal of MR research has focused on photo-realistic rendering [17] and on estimating the real-world lighting to align the lighting of the virtual scene accordingly [24]. However, the challenges for achieving visual coherence also depend on the MR device that is used to blend real and virtual scene content [3]. For example, applications that focus on video-based MR [15] need to account for the characteristics of the camera in order to visually align the rendering with a capturing of the real environment. This includes the lens, the sensor, and pixel conversions of the image signal processor (ISP). Although existing approaches can simulate several components of the camera pipeline [19, 40, 45], they require detailed camera specifications, often not obtainable, or special calibration targets to be visible in the scene [21, 28]. This makes them difficult to apply to new cameras when specifics are unknown or calibration targets are not available at runtime.

We introduce *Neural Cameras*, a generic and practical solution for increasing the visual coherence in video-based MR applications. Neural Cameras learn the characteristics of their physical counterpart from an image database that has been captured with the camera of interest. At runtime, Neural Cameras apply the learned characteristics to the renderings of virtual MR scene elements. Contrary to existing approaches, adding support for a new camera is as simple as capturing a new image database. For example, the results in this paper have been generated with databases that took approximately three hours in total for training one Neural Camera. No further knowledge about the physical camera was required because Neural Cameras learn to mimic the camera characteristics which are common to the images in the database.

Neural Cameras also present the first approach for visual coherence in video-based MR that jointly supports lens, sensor and ISP effects. This is demonstrated in Figure 1(b and c). Figure 1(b-top) shows a real toy tiger next to a rendering of its 3D scan, using a traditional virtual camera. As a result, the virtual toy tiger suffers from mismatching colors and lens blur. In contrast, Figure 1(b-bottom) shows the same MR scene but rendered with a Neural Camera which

*e-mail: mandl@icg.tugraz.at

has been trained to mimic the visual characteristics of the physical camera. The Neural Camera aligns colors and lens blur of the rendering to those produced by the physical camera that is used to capture the real toy tiger.

A more direct comparison of a traditional rendering and Neural Cameras is provided in Figure 1(c), where the 3D scan has been 3D registered to the real object and partially rendered with a traditional camera (right-Naïve) and with the learned Neural Camera (middle-Learned). Notice the visual similarity between the image captured with the physical camera and the rendering generated with the Neural Camera.

Neural Cameras significantly contribute to the state-of-the-art of visual coherence in video-based MR. While existing approaches are often limited by missing camera specifications [19] and missing calibration targets in the scene at runtime [21, 28], Neural Cameras overcome those issues as they learn the camera characteristics offline. These are then applied to renderings in new and unprepared environments at runtime. Therefore, no knowledge about camera specifications is necessary and no calibration targets have to be included in the MR scene at runtime. In summary, this paper presents the following primary contributions:

- We introduce Neural Cameras, a novel approach for coherent rendering in video-based MR by learning the camera characteristics.
- We introduce individual approaches for learning the characteristics of the lens, the sensor and the ISP from an image database of the specific camera.
- We provide a thorough analysis of Neural Cameras.

Prototype. For our prototype implementation, we focused on the most prominent effects that appear in video-based handheld MR applications with modern cameras, but left others that are less prominent, such as chromatic aberration and vignetting, for future work. Also, we left the simulation of motion blur for future work.

2 Related Work

Visual coherence in MR has been demonstrated using photorealistic and non-photorealistic rendering techniques and approaches to camera simulation.

2.1 Photorealistic rendering

Photo-realistic rendering has been proposed using 3D stock models, textured with photographs [18]. While this technique achieves impressive results it is tailored to stills and, thus, not feasible for interactive MR applications which are driven by real-time video input.Research has also been conducted to support rendering global illumination effects in MR [23]. Existing approaches enable realistic rendering of reflections, refractions, caustics [17, 22], radiosity [20, 43], material estimation [25], coherent lighting [24, 34], and shadows, which are cast between virtual and real scene elements [4, 33]. Although we emphasize the importance of illumination effects, it is not the focus of this work, as we entirely focus on the effects introduced by the camera. Thus, our current prototype only supports coherent lighting and shadow rendering to demonstrate the impact of Neural Cameras. However, since our approach is designed to extend existing MR frameworks, it can be easily be extended to include further approaches to photo-realistic rendering in MR.

2.2 Non-photorealistic rendering

Meeting the visual quality of the physical world is challenging, in particular, when considering the real-time requirements of most MR applications. Thus, several research groups have also approached the issue of visual coherence by reducing the visual fidelity of the real world [9, 31] to create a low-fidelity but coherent MR [12, 41]. This has been done by applying illustrative rendering techniques to all scene elements (real and virtual), so that they appear similar to each other [10]. However, since all of these approaches only deliver a low visual quality, they are unsuitable when details of the environment need to be preserved (e.g., text, small objects, etc.).

2.3 Camera simulation

One of the key elements to visual coherence in video-based MR is to mimic the characteristics of the physical camera of the MR device in the rendering. An early work by Klein and Murray [19] demonstrates the impact of a camera simulation on visual coherence. While they simulate several components of a real camera their work assumes detailed knowledge about the internals of the camera. As such, the approach and its parameters are tuned to a very specific camera only. In addition, the approach does not provide a tool for reproducing the colors of the camera. In contrast, our work focuses on the three main components of the camera pipeline: the lens, the sensor, and the ISP. We use advanced learning-based methods for each of the components to make calibration procedures feasible and more applicable, while simultaneously not requiring detailed internal knowledge. Therefore, we review related work on simulating each of these components.

Lens simulation Simulation of radial lens distortion is included in many MR renderings because it can be corrected at high frame rates [32, 45] and calibrated with simple and well-known techniques based on a few images of a black and white checkerboard [47]. Other effects, such as depth of field (DoF), chromatic aberration [2], and motion blur [19, 29] are more complex and, thus, less often considered in MR renderings. For example, applying accurate DoF-effects to virtual content requires densely sampling the camera aperture using a number of pinhole cameras [14, 27]. Numerous methods have emerged for accelerating DoF rendering using post-processing techniques [7, 35, 38]. However, these techniques usually lead to artifacts around occlusion boundaries [5]. Recently, the DeepFocus deep neural network (DNN) demonstrated high-quality DoF-effects from RGB-D images at real-time update rates [46]. While this is a very promising direction of research, DeepFocus requires a large amount of training data which consists of several thousand focal stack images. Such a large database can be generated in a virtual environment but is difficult to produce with a physical camera in a real-world setting. Since DeepFocus trains from renderings, it learns a virtual camera model and produces the DoF-effects accordingly. Therefore, we cannot apply DeepFocus to match the DoF-effect of real cameras. Instead, we aim at a network that requires less training data so that we can feed it with images taken by the physical camera. Therefore, we trade the high-quality blur as produced by DeepFocus with one that considers the lens model of the physical camera.

Sensor and ISP simulation There are only a few works that focus on approximating the color response of the sensor and the color characteristics of the ISP in real-time applications. Knecht et al. [21] consider both but treat the sensor and the ISP as one black box. They simulate the behavior by creating an online color mapping, requiring knowledge about colors in the physical scene and its lighting condition. Furthermore, the physical scene needs to exhibit the full-color range in at least one color channel. In cases where image areas that contribute to the color range are occluded, the mapping will be incorrect. The approach by Rohmer et al. [36] builds upon the work of Knecht et al. but computes a mapping between reference colors in a point cloud reconstruction and the camera color space. While demonstrating impressive results, the output strongly depends on the current view and a stable scene lighting at runtime. Also, the unknown reference colors prevent from correctly mapping virtual replicas of real objects. In contrast, our approach learns the characteristic in an offline step making it robust to new environments and arbitrary views.

In a physical camera, the RAW image data is commonly also filtered by an ISP, which applies several operations such as demosaic-

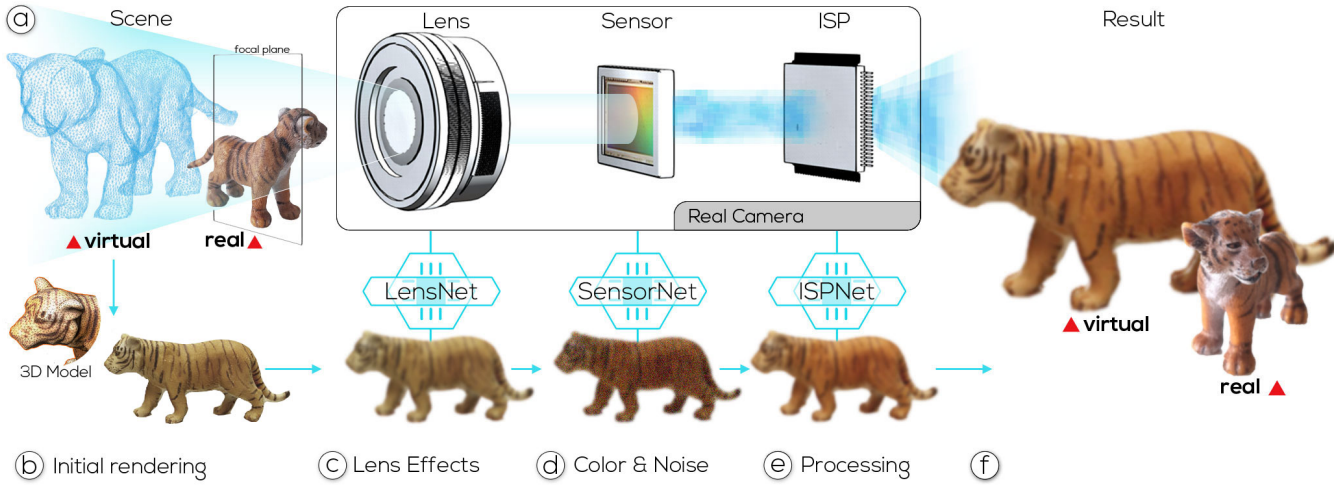

Figure 2: Overview. (a) MR scenes consist of real and virtual elements. In video-based MR, a physical camera captures the real scene, exhibiting the characteristics of its lens, sensor, and ISP. (b) Our approach renders the virtual scene with traditional approaches and additionally applies learned (c) lens effects, (d) sensor characteristics such as color and noise, and (e) characteristics of the ISP, to (f) produce the display image.

ing, denoising, white balancing, and compression into a displayable output format. Recent work based on DNNs proposes simulating the ISP independent of the sensor. For example, Schwartz et al. [40] propose DeepISP, a two-stage DNN for denoising and demosaicing only. A more complete simulation of the ISP was proposed by Gao et al. [11], who design a dual network based on the approach of Nam and Kim [26]. The approach first converts a JPEG camera frame to RAW image data, to which virtual content is composited, before the composited RAW image is converted to JPEG and extracted to the displayable sRGB output. We adapt this network for adding the characteristics of the ISP to the output of SensorNet and to use the YUV image output of the Android camera directly, which we map to sRGB output only before we display the image.

In summary, contrary to approaches focusing on non-photorealistic and photorealistic rendering, this work contributes by simulating the characteristics of the camera pipeline. While current approaches are limited in their general applicability, Neural Cameras overcome these limitations by introducing a learning-based approach that considers arbitrary cameras and arbitrary scenes.

3 Overview

We designed Neural Cameras to easily integrate with traditional rendering frameworks. Therefore, we have built Neural Cameras to operate on an initial rendering to a G-buffer [37], which consists of color and depth channels. We start by rendering the 3D model using a traditional MR camera (Figure 2(b)), before applying a series of post-rendering steps in image-space to add lens, sensor and ISP effects (Figure 2(c–e)). Since the resulting image includes camera effects it coherently blends in with the captured frame (Figure 2(f)).

3.1 Initial rendering

When initially rendering virtual structures, we also consider the illumination in the real-world environment. We use an active light probe [39], a common state-of-the-art technique for estimating the lighting and restrict the current application to diffuse objects. We align the diffuse shading of virtual objects by using image-based lighting based on the information in the light probe. In addition, we render shadows similar to the approach of Rhee et al. [33] and we account for lens distortion, as it has been efficiently implemented in traditional rendering approaches [32]. Since we focus on simulating camera effects, we did not implement other effects, such as global illumination or refraction in our current prototype. However, since we designed our work to integrate with existing frameworks we can easily extend it to other effects.

3.2 Mapping the characteristics of the physical camera

The camera-specific effects will be added by applying a learned Neural Camera to the G-buffer output of the initial rendering phase. To learn the camera characteristics, we designed three neural networks, each for one of the major components of the camera pipeline. More specifically, we designed a network *LensNet* to mimic the behavior of the lens system, a network *SensorNet* to mimic the image sensor, and a network *ISPNet* to include effects that are otherwise added by the ISP. Note that the separation into three different networks allows skipping LensNet for objects within the depth of field of the camera, and for training SensorNet and ISPNet with different databases. We show that both networks require different scales in size and complexity. In the following, we provide a high-level overview of all three networks before subsequent sections present details and results for each.

LensNet. We train LensNet to blur the initial rendering according to object depth and the focal length of the real camera. We use a Gaussian mixture model to render high-quality blur and we use a neural network to find the parameters for the renderer that match the characteristics of the physical camera lens. To train the network, we capture images of a planar object, which is placed at several distances and captured with varying focus distances.

SensorNet. We mimic the image sensor by adding noise and mapping rendered colors to the color space produced by the physical camera using SensorNet. We train the network with pairs of captured and known colors. To produce the training database we take pictures of a reference object, i.e., a color chart showing known colors, while varying the lighting, to cause the ISO, and exposure settings of the real camera to change. We store the RAW images with the corresponding ISO and exposure values in the database. To reduce the computational effort at runtime, we do not add artifacts of a bayer filter to the rendering. Such effects are well corrected by modern consumer-grade cameras and commonly do not appear in the output image. Therefore, we automatically apply a de-bayering to all captured RAW images using the AMaZE algorithm[1].

ISPNet. The color chart used for training SensorNet provides enough color variation for training the color and noise mapping as they appear in a RAW image. However, a single object does not provide enough variation to learn the entire filter pipeline which is introduced by the ISP, when mapping RAW to output images. Therefore, we additionally produce a larger database of pairs of

[1] http://www.rawtherapee.com

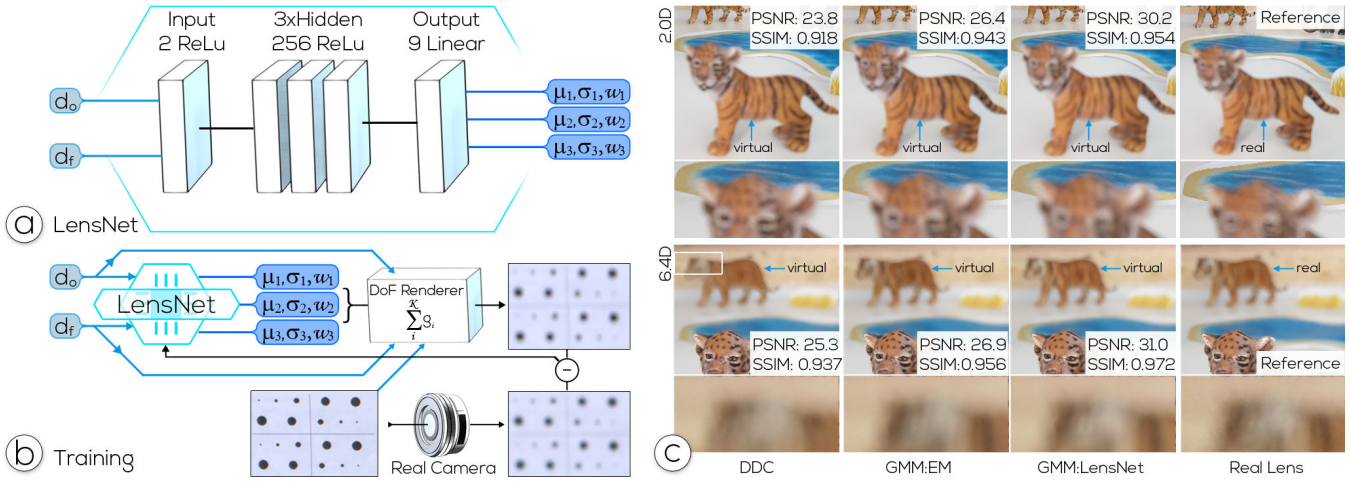

Figure 3: LensNet. (a) We designed LensNet as a multilayer perceptron, composed of a two-dimensional input layer (for object depth and focus distance), three hidden layers with 256 units each, and a nine-dimensional output layer, where the neurons correspond to three sets of Gaussian kernel parameters for a mixture model (GMM) with $K = 3$ components. (b) For training, we capture blurred and in-focus images of a calibration target. The DoF renderer artificially blurs the in-focus image with the GMM. The parameters are updated to minimize the difference between captured and rendered image. (c) We compare the results of our approach to depth dependent convolution (DDC), to results using a GMM which has been estimated using expectation maximization (EM), and to a reference image captured by the physical camera (Real Lens).

RAW and output images. Including the color chart to map captured to known colors adds a large overhead to the capture process. Thus, to reduce the effort required to generate the database for training, we design an additional network (ISPNet) that only maps from already known colors in RAW image space, as produced by SensorNet, to the camera's output space, which includes the filtering introduced by the ISP.

4 Learning the Lens System

Lenses appear to have many imperfections that introduce optical effects such as distortion and chromatic aberration. However, modern cameras often effectively counter them by using corrective elements in the lens system or through software as part of the ISP. One effect that remains prominently visible is the blur that results from objects outside the focus distance of the camera. Such focus blur provides important depth cues to the human visual system, which is why renderings of mismatched DoF effects may appear at a wrong distance. Therefore, besides undistorting the image during the initial G-buffer rendering phase, we focus the simulation of the lens system on reproducing DoF effects.

DoF effects originate from light rays which are bent by the individual lens elements before they hit the sensor. Thereby, the lens focuses light rays onto the camera sensor to get a sharp image of the scene. However, depending on the properties of the lens system, i.e., the aperture and focal length, only objects in a certain depth range, the DoF, appear sharp. Light rays from objects outside the DoF do not fully converge to a single pixel on the sensor, and thus, spread within a circle of confusion (CoC), occupying several pixels on the sensor, which causes the corresponding objects to appear blurred.

The CoC diameter $c(d_o)$ on the camera sensor of a point at distance d_o to the camera can be computed as

$$c(d_o) = A \frac{f}{d_f - f} \frac{|d_o - d_f|}{d_o}, \tag{1}$$

where A denotes the diameter of the aperture, d_f the focal distance, and f the focal length of the lens system. Thus, given these camera parameters, we can calculate the CoC diameter, which can be used to blur renderings in a post-processing step using a simple depth-dependent convolution [5] (see Figure 3(c-DDC)).

However, since real lenses do not match the thin lens model, the intensity of light rays within the CoC degrades towards the edge of the CoC, hence using a uniform blur kernel will not model the lens correctly. In order to reproduce blur more accurately, we introduce a Gaussian Mixture Model (GMM), comprised of weighted symmetrical 2D Gaussians to weight the convolution kernels accordingly. We set up the GMM according to

$$GMM = \sum_i^K g_i(r, c(d_o)), \tag{2}$$

where r is the radius from the pixel center, and K is the number of individual Gaussian components. Each Gaussian of the GMM is defined as

$$g_i(r, c(d_o)) = \begin{cases} w_i \cdot e^{\frac{(r-\mu_i)^2}{2\sigma_i^2}} & r \leq c(d_o)/2 \\ 0 & r > c(d_o)/2\,, \end{cases} \tag{3}$$

where σ_i, μ_i and w_i denote the standard deviation, the center position, and the weighting of the i^{th} Gaussian respectively.

Figure 3(c-GMM:EM) demonstrates the advantage of a normalized GMM to model lens blur over a uniformly weighted blur kernel Figure 3(c-DDC), rendered using the depth-dependent convolution [5]. While the GMM allows for an arbitrary number of Gaussian components, K, the examples in Figure 3(c) have been generated using $K = 3$, which was sufficient to demonstrate the impact. We find the parameters σ_i, μ_i and w_i by fitting the GMM with the expectation-maximization (EM) algorithm in Matlab [6] on an image that presents the calibration target (shown in Figure 3(b)) out of focus to provide enough out of focus blur for the EM to work.

Although our GMM-based lens model enables for better approximating blur generated by a physical camera lens, fitting the model with the EM algorithm requires manual intervention to extract the blur profile from a calibration target. Therefore, we train a neural network to estimate the parameters σ_i, μ_i, and w_i based on a given pixel depth and focus distance. Since the neural network is trained automatically, the user effort is constant over an arbitrarily large number of training samples. Since more training samples allow obtaining a better model the lens blur produced from those GMMs will closer match the results of physical lenses. Figure 3(c-GMM:LensNet) demonstrates the improvement over parameters that have been carried out by the EM algorithm in Figure 3(c-GMM:EM).

Network design. To learn the parameters σ_i, μ_i, and w_i of the GMM, we introduce LensNet, a multilayer perceptron (MLP) network, consisting of three hidden fully connected layers with 256

units each. LensNet maps the two-dimensional input, i.e., the object depth d_o and the focus distance d_f, to a $3 \times K$ output layer, representing σ_i, μ_i, and w_i for the K mixture components, respectively. Note that the object depth of real objects can be derived from the DepthAPI provided by ARCore and the depth of virtual objects from its rendering. We apply Rectified Linear Unit (ReLU) as activation function in the first two layers and a linear function in the output layer. The network architecture is illustrated in Figure 3(a). Note that our current prototype does not change the aperture or the focal length, since both cannot be changed in many modern smartphones. However, in the future, we will extend our network to also consider changing aperture and focal length settings.

Network training. To learn rendering lens blur, which is similar to those in the training database, we train the network with the DoF-renderer in the loop (see Figure 3(b) for an illustration). We render lens blur using a depth-dependent blur, which takes the rendered image, the depth map, and the camera parameters from Eq. (1) as input. LensNet provides a GMM for each rendered pixel which is used to compute the weights of the blur kernel accordingly.

The network learns the parameters of the GMM by comparing the corresponding output of the DoF-renderer for a given depth d_o and a focus distance d_f to an image in the training database with the same d_o and d_f parameters. Therefore, we train the network using the following loss function:

$$\underset{\sigma_i,\mu_i,w_i}{\arg\min} ||I_f * \sum_{i}^{K} g_i(r, c(d_o)) - I_b||^2, \tag{4}$$

where I_f denotes the image with the image target in focus, I_b the image with the object out of focus, and $*$ the image convolution that is applied in the DoF renderer.

We acquire the training data by taking nine out-of-focus and a single in-focused image of a textured planar object. The objects is placed at ten different distances w.r.t. the camera. For each distance, we vary the focus of the camera ten times. In total, we acquire a data set of 100 images. We calculate the per pixel depth d_o by estimating the six degrees of freedom pose of the planar object (using ARCore[2] in our prototype implementation). Since camera tracking is commonly affected by image blur, we obtain the depth from the in-focus image and turn off tracking while capturing the out-of focus images.

Evaluation. We measured the qualitative and quantitative performance of approximating lens blur using LensNet and compared the results to depth-dependent convolution [5] (referred to as DDC) and to an implementation of expectation-maximization (EM) [6] with user interaction for identifying the blur profile.

Figure 3(c) shows visual results and corresponding Peak Signal-to-Noise Ratio (PSNR) [13] and Structured Similarity Image Metric (SSIM) [44] values using a near and a far focus distance. To measure the impact of the approach, independent of its relative size in the image plane, we compute both metrics only from the pixel values inside the bounding rectangle of the footprint of the virtual object. In addition to the 3D MR environments shown in Figure 3(c), we test our approach on the entire image plane using a lens test chart[3]. The lens test chart has been optimized for identifying lens effects in a photograph and thus, provides a well suited scene for measuring the performance of a lens simulation.

We compare the simulation results to an image captured with the physical camera using the simulated focal distance. In an initial evaluation, we noticed only a small difference between the three approaches. Since image noise and color differences diminish the difference, we extended the evaluation setup so that we get measurements that are less affected by sensor and ISP artifacts. Therefore,

[2] https://developers.google.com/ar

[3] https://www.benny-rebel.de/objektiv-und-kameratests/

Table 1: Lens blur approximation. Bold fonts indicate best results. The captured object is placed at 1.29D in front of the camera.

	d_f	1.03D	4.05D	6.1D	8.06D	10D
LensNet	PSNR	**30.06**	**32.68**	**31.40**	**30.60**	**29.97**
	SSIM	**0.848**	**0.912**	**0.922**	**0.932**	**0.942**
EM	PSNR	27.67	29.90	27.85	27.60	26.88
	SSIM	0.845	0.910	0.921	0.931	0.941
DDC	PSNR	23.51	23.02	23.13	23.16	22.73
	SSIM	0.843	0.898	0.912	0.924	0.934

we use projective texture mapping [8] to generate the color information of the 3D registered virtual object. As projective texture input, we use an image where the virtual object is in-focus. To furthermore reduce camera noise in our test images, we use the average pixel value from a series of 30 images, captured from the same position, orientation, and with the same camera settings.

Quantitative results at several focal distances using the lens test chart are shown in Table 1. The average PSNR values are 31.07dB for LensNet, 27.91dB for EM, and 23.29dB for DDC. The average SSIM results are 0.9071 for LensNet, 0.9069 for EM, and 0.8991 for DDC. The GMM-based approaches with $K = 3$ outperform the simple DDC technique, whereas LensNet is able to generate parameters that better approximate the lens blur compared to the EM approach. We believe this is the result of being able to use many more samples for training compared to fitting the model with EM.

The runtime for inferring the parameters with LensNet was measured on average with 1ms for a resolution of 640×480 and 3ms for a resolution of 1280×720. The averaged runtimes for rendering blur across all focus distances were 1.46ms and 2.47ms for scene resolutions of 640×480 and 1280×720, respectively. All measures were carried out on a PC with an AMD Ryzen 9 3900X CPU and an NVIDIA GTX 2080 Ti GPU.

5 Learning the Sensor

The second network in our pipeline is SensorNet. It receives the output from the previous stage and adds the characteristics of the camera's imaging sensor. Therefore, it adds noise and it maps the colors from rendered color space to the color space of the real camera sensor.

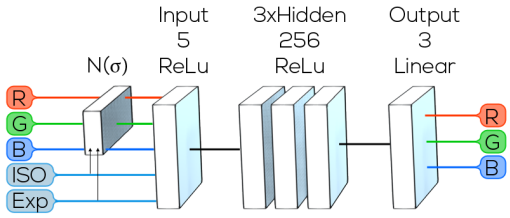

Figure 4: Architecture of SensorNet. SensorNet is a multilayer perceptron composed of an input layer for the ISO value, exposure, and three color channels, three hidden layers with 256 neurons each, and a linear output layer with three outputs values, each for every color channels. We add per-channel noise by choosing $N(\sigma_j)$ according to the current ISO and exposure settings.

Network design. To learn the color mapping, we introduce SensorNet, an MLP with three fully-connected hidden layers with 256 neurons each (see Figure 4). We map a five-dimensional input layer that represents a pixel's red, green, and blue color values, in addition to the ISO value and exposure time, to a three-dimensional output layer, which represent final red, green, and blue color values. Noise is added by altering input colors based on the noise model $N(\sigma_j)$, which is derived from the ISO and exposure values. In our prototype implementation we retrieve ISO and exposure values from the Android camera API. For the output layer, a linear activation function was used, because we are interested in regressing continuous output values. For all other layers, we use ReLU activation.

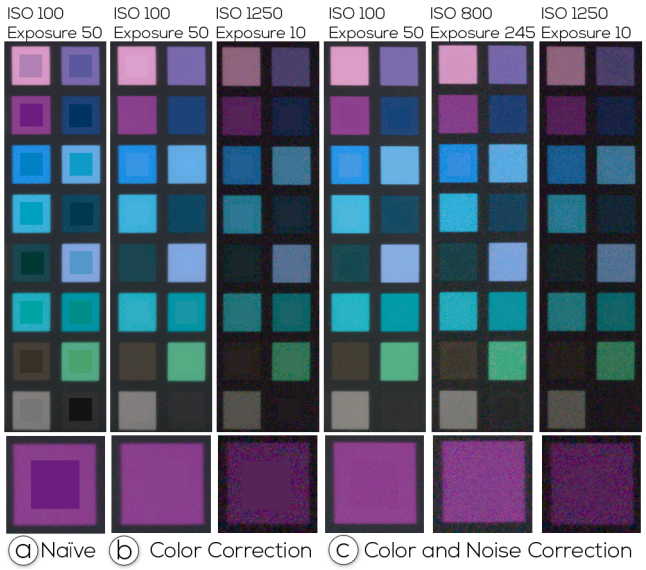

Figure 5: Results of SensorNet. (a) Central patches show the initial rendering without correction. (b) The initial renderings have been transformed by SensorNet without considering noise. (c) The output of SensorNet when adding noise. Different camera settings have been caused by changing the lighting while using the camera's auto-setting.

Network training. We train the mapping with pairs of colors, which we retrieve by capturing images of a calibration target with known radiance. We estimate the training loss by computing the mean square error between the output color values and the target color values as they appear in the target image color of the captured color patch. We used the 'Colorchecker' calibration chart target from X-Rite, which provides measurements for 24 different color samples. To add variation, we capture images while varying the scene lighting, which causes varying ISO and exposure settings. Therefore, lighting is implicitly included in the database. We generate twelve different ISO and exposure settings. We train the model using the Adam optimizer and a learning rate of $1e^{-4}$.

We add sensor noise to input colors by assuming normally distributed noise $n(\sigma_j)$ on each color channel, $j \in \{R,G,B\}$ [19]. Since our calibration target consists of patches of the same color, we can deduct the noise from analyzing the color distribution with each patch. Therefore, in the captured RAW images of the color chart, we calculate the standard deviation σ_j within each color patch, and we add the resulting noise model $n(\sigma_j)$ to the input of the network. We use the noise model to add noise to input colors, which the network maps to noisy output colors. We use the training database to compute a mapping from ISO and exposure values to σ_j, which we use to calculate $n(\sigma_j)$ and to noise accordingly to the input colors at runtime. Note, we add noise to input colors, instead of output colors, to learn a mapping of noisy input colors to noisy output colors. Otherwise, a single color would map to several output colors.

Evaluation. Figure 5 shows the rendering on top of the captured RAW image of our test color chart. Note that we train the network with colors from the 'Colorchecker' chart from XRite and evaluate on images of the Digital SG chart, also from X-Rite. In each of the test images, a synthetic rectangle with a known color is overlaid on top of the corresponding area in the color chart. This allows us to see rendered and captured colors next to each other. Figure 5(a-c) compares results without SensorNet, to results from SensorNet with and without additional noise. This demonstrates that our approach is able to estimate color and noise similar to the image sensor of the physical camera. The variation in ISO and exposure settings furthermore demonstrates that the approach is robust to lighting conditions as they have been included in the training database.

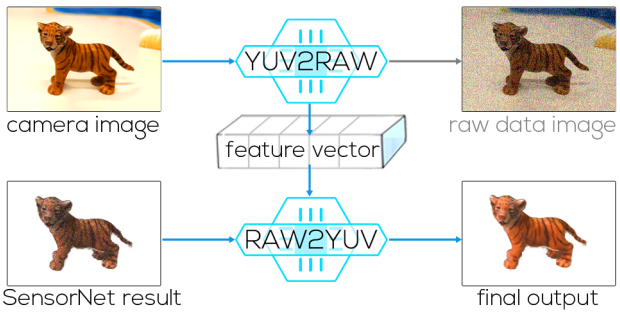

Figure 6: ISPNet consists of two sub-networks. (top row) The first sub-network maps the incoming camera frame from its YUV representation to a RAW representation. It also generates a feature vector (bottom row) which is used to guide the mapping from the output of SensorNet to the YUV output image.

We also measured the runtimes for processing an image with SensorNet. For images with 640×480 pixels we measure a average processing time of $1ms$ and $2ms$ for images with 1280×720 pixels. All measurements have been performed on an AMD Ryzen 9 3900X CPU with an NVIDIA GTX 2080 Ti GPU.

6 Learning the Image Signal Processor

SensorNet maps the initial rendering to RAW images. We designed ISPNet to emulate the image signal processor to transform the RAW color image into a displayable output. The ISP commonly improves the image quality by applying various image filter operations, such as white balancing and noise reduction. However, the details of the ISP are very specific to each camera model and most often unknown to the public. Therefore, we train a neural network to mimic the behavior of the processor instead. Thus, adjusting the system to a new camera only requires training the system with a new database that has been captured with the camera of interest.

Network design. We built ISPNet upon the dual network proposed by Gao et al. [11], which consists of two networks to first transform a JPEG-compressed image to a RAW image before the RAW image is augmented and then transformed back to JPEG-format. During the JPEG-to-RAW mapping, the network extracts a feature vector which is used to narrow the solution space for the RAW-to-JPEG transformation (Figure 6). Due to space constraints, we refer the reader to the work of Gao et al. [11] for details about the architecture.

We adopt this approach to handle input video frames from state-of-the-art AR frameworks, such as ARCore[4], which often use YUV color space. Therefore, ISPNet receives a simulated RAW-image of virtual scene elements from SensorNet and transforms it into a YUV color image, which includes the image characteristics of the learned ISP. We derive the feature vector, which is necessary for converting RAW-to-YUV, by transforming the input YUV image to a RAW image first (Figure 6). To display the output of ISPNet we converted them to sRGB color space using a gamma value of 2.2.

Network training. Depending on the characteristics of the RAW image, the ISP often performs several operations using parameters that it estimates at runtime. Therefore, we need a large variability in the training data and consequently a large amount of RAW-YUV image pairs. We trained our prototype with approximately 400 image pairs that have been captured in many different environments and at different times during the day. Note that the requirement for a large variability caused us to introduce an additional network, next to SensorNet, for learning the ISP. Learning the sensor requires pairs of known and captured colors, which is very difficult to extract from a large number of arbitrary scenes.

[4] https://developers.google.com/ar

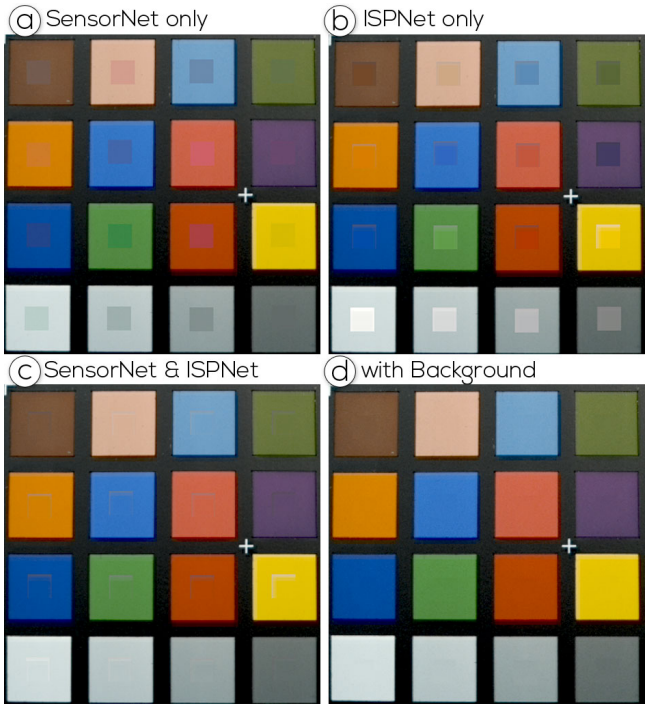

Figure 7: ISPNet. (a) The output of SensorNet overlaid on the captured image. (b) Processing the initial rendering with ISPNet leads to wrong colors since ISPNet has been trained with RAW images. (c) ISPNet can produce matching colors by processing the output of SensorNet. To increase rendering performance we process renderings only which causes artifacts at the border. (d) We remove artifacts by processing the rendering and its background around the border.

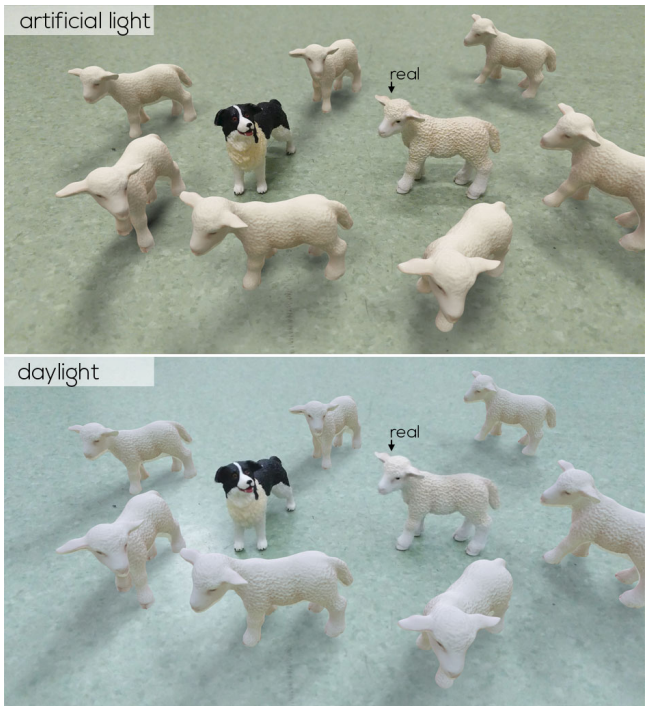

Figure 8: Varying environment light. (top) Artificial office light, and (bottom) natural daylight is provided to the MR environment. The Neural Camera is able to adapt the rendering accordingly.

Evaluation Figure 7 shows the output of SensorNet augmented on the YUV image that we receive from ARCore (Figure 7(a)). While the overlay clearly stands out from the background after processing with ISPNet the results become coherent and difficult to identify (Figure 7(c)). Note that we need to input simulated RAW images because ISPNet has been trained to map RAW-to-YUV. Figure 7(b) demonstrates the error when the initial rendering is used as input.

Since ISPNet follows the approach of Gao et al. [11], we send the augmented RAW image through the network. Although this results in high-quality visual coherence, it requires two passes over the entire image, which demands a large amount of computational resources. We achieve a runtime performance of approximately 119*ms* and 335*ms* on images with 640×480 pixels and 1280×720 pixels respectively, using a GTX 2080 Ti. Since the performance scales with the size of the image, we can improve it by sending only the synthetic overlay through ISPNet. Note, as this approach only alters the synthetic elements in the scene, it reveals the error that is introduced by ISPNet (Figure 7(c)) while it increases the performance relative to the ratio of virtual to real content. In our test environment, we achieve a runtime performance of approximately 11*ms* for an augmentation of 338×39 pixels.

Note that the error is most noticeable at the edges of the overlay. As this is introduced by network convolutions that include the uniform black background in our variation, we can remove the artifacts by adding the generated RAW image in the background around the border of the initial rendering, before sending the combination through the network. We use an image-based approach similar to image-based silhouette rendering [30]. However, to not alter the size of the overlay, we display only the pixels inside the original footprint.

7 Discussion

A lack of documentation for the technical details of the camera pipeline is common and often prevents an accurate simulation of the camera pipeline in practice. Instead, our approach has been designed to easily include new cameras, as long as we can obtain the images required to train our networks. The focus on generalisability is what sets apart our work from existing approaches for simulating the camera characteristics in MR. Most modern camera vendors do not release the details required for a camera simulation using existing approaches, like the one proposed by Klein et al. [19]. This prevents a comparison with our approach for many modern cameras. Assuming that a simulation-based on vendor knowledge yields an exact solution in the best case, we test our approach against the exact solution by comparing it with ground truth images that are captured by the physical camera itself. Our evaluation focuses on providing feedback on the efficacy of our approach while also demonstrating the performance and the effort needed to train the networks. For example, the results in this paper have been generated using the back-facing camera on a Google Pixel 2XL mobile phone and a Samsung Galaxy S9 mobile phone. Both represent typical devices used in handheld MR applications for which the vendors do not provide detailed information to replicate the camera pipeline.

Generating the image database for one Neural Camera required capturing in various scenes approximately 400 pairs of RAW and YUV images for training ISPNet, 100 images for training LensNet, and 12 images of a color calibration chart while varying the lighting for training SensorNet. Collecting the data used for generating the results in this paper took an untrained person approximately two hours for training ISPNet and 45-60 minutes for all remaining training data. Training the networks took approximately 30 minutes for SensorNet, two hours for LensNet, and ten hours for ISPNet on an AMD Ryzen 9 3900X CPU and an NVIDIA GTX 2080 Ti GPU.

Figure 8 and Figure 9 provide further results to demonstrate the applicability of Neural Cameras across differently colored scene objects and different environmental conditions. Notice how the Neural Camera adapts the rendering to the color of environment light in

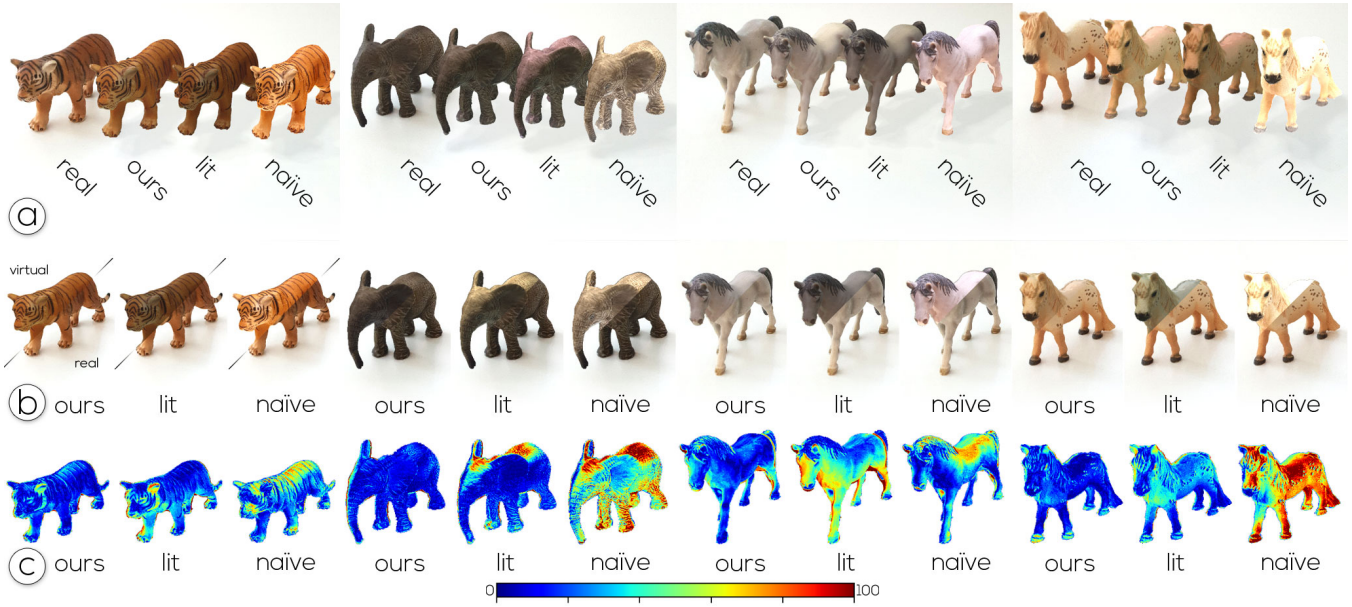

Figure 9: Results. (a) Side-by-side comparison of camera capturings (real) to renderings using the our approach (ours), correctly lit renderings without camera effects (lit), and renderings using the unlit textured object only (naïve). (b) The same renderings but in split view and 3D registered to the real object. The top triangle shows the rendering, bottom shows captured image. The dashed line in the tiger indicates the split. (c) Visualization of the RGB-color difference between the rendering and the captured image per pixel. The color map encodes the L2 distance, where blue represents small distances and red large distances.

Table 2: Quantitative results. Bold fonts indicate best results.

	Ours		Shading		Naïve	
	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
Tiger	**22.0**	**0.78**	20.0	0.75	19.1	0.76
Elephant	**22.9**	**0.70**	20.2	0.68	14.8	0.56
Horse	**22.4**	**0.83**	17.6	0.79	19.1	0.79
Pony	**23.4**	**0.77**	19.9	0.76	13.7	0.68

Figure 8. Figure 9 furthermore shows the impact of Neural Cameras on the rendering of four different 3D scans. The four scanned 3D objects have been selected to provide color and brightness variations across the scans. To calibrate the colors in the texture map of the 3D models, we include the small color chart that is used to train the networks in the scanning process. We use the approach of Knecht et al. [21] to map texture values to the trained color space of the color chart. We selected diffuse objects and we estimate the environment lighting by capturing a 360 high dynamic range light-probe.

Figure 9(a) provides a side-by-side comparison of a camera frame showing the real object, next to the results of our approach, and the initial rendering (called 'naïve'). Since our approach considers the estimated real-world lighting, we also include results of the initial rendering under the estimated environment lighting (called 'lit'). In Figure 9(b), we provide a more direct comparison by showing 3D registered renderings in a split-view visualization, which is indicated by a black line in the image showing the tiger. In each image, the upper half presents the rendering and the lower half shows the captured frame. Finally, in Figure 9(c), we complement the direct comparison with an error visualization of the difference between rendered and captured pixels. Furthermore, we compute PSNR and SSIM scores of capturing and rendering (Table 2).

The results clearly demonstrate that Neural Cameras can decrease the visual difference between rendered and captured scene elements. The naïve rendering suffers from incorrect lighting and missing camera effects, and therefore, shows the highest error and lowest PSNR and SSIM results. By respecting the environment lighting the error decreases. By additionally including camera effects, the results further improve for all objects in our test database. Note that all measures include registration and shading errors, which result from an imperfect light and pose estimation. Since all renderings suffer from the same systematic errors, the comparison provides a better understanding of the performance than the absolute measures.

8 Conclusion

Previous approaches for camera simulation are either limited to the availability of technical details [19] or require a calibration target visible in the camera image at runtime [28]. Our approach is less invasive and, thus, more applicable to unknown cameras. We learn the characteristics of the major components of the entire camera pipeline offline and apply them to the initial rendering at runtime. While existing approaches already showed the importance of simulating the camera characteristics for visual coherence, our work is paving the way to a practical solution.

By designing Neural Cameras based on traditional rendering techniques and a series of post-rendering processes, it easily integrates with existing rendering tools, which allows us to use well-known and often well-optimized rendering methods. Therefore, our approach can directly improve most MR applications. Also, applications to mediated reality, such as ClayVison [42] and AR exploded views [16], which enable animating real-world objects, can highly benefit from Neural Cameras support coherently rendering 3D MR environments from novel points of view.

Our results demonstrate the potential of Neural Cameras. However, we can still improve the results by improving the initial rendering and the virtual scene representation with fine detail, which is often present in real objects. In addition, we see potential in extending our approach to include camera characteristics currently not considered. Furthermore, potential in improving the current process of capturing the input images by adding user guidance for capturing databases with high enough variability.

Acknowledgments

This work was enabled by the Austrian Science Fund FWF (grant no. P30694), MBIE Endeavour Smart Ideas, the German BMBF in the framework of the international future AI lab "AI4EO" (grant no. 01DD20001), and the Competence Center VRVis. VRVis is funded by BMK, BMDW, Styria, SFG, Tyrol and Vienna Business Agency in the scope of COMET - Competence Centers for Excellent Technologies (879730) which is managed by FFG.

REFERENCES

[1] R. T. Azuma. A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, 6(4):355–385, 1997.

[2] S. A. Cholewiak, G. S. Love, P. P. Srinivasan, R. Ng, and M. S. Banks. Chromablur: Rendering chromatic eye aberration improves accommodation and realism. *ACM Transactions on Graphics*, 36, 2017.

[3] J. Collins, H. Regenbrecht, and T. Langlotz. Visual coherence in mixed reality: A systematic enquiry. *Presence: Teleoperators and Virtual Environments*, 26(1):16–41, 2017.

[4] P. Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proc. Int. Conf. on Comput. Graph. and Interactive Techniques*, pp. 189–198, 1998.

[5] J. Demers. Depth of field: A survey of techniques. In *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*, chap. 23, pp. 375–390. Addison-Wesley Professional, 2004.

[6] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[7] R. Du, E. Turner, M. Dzitsiuk, L. Prasso, I. Duarte, J. Dourgarian, J. Afonso, J. Pascoal, J. Gladstone, N. Cruces, S. Izadi, A. Kowdle, K. Tsotsos, and D. Kim. DepthLab: Real-time 3D Interaction with Depth Maps for Mobile Augmented Reality. In *Proc. ACM Symp. on User Interface Software and Technology*, 2020.

[8] C. Everitt. Projective texture mapping. 2001.

[9] J. Fischer, D. Bartz, and W. Straßer. Reality tooning: Fast non-photorealism for augmented video streams. In *Proc. Int. Symp. on Mixed and Augmented Reality*, pp. 186–187, 2005.

[10] J. Fischer, D. Bartz, and W. Straßer. Stylized augmented reality for improved immersion. In *Proc. IEEE Virtual Reality*, pp. 195–202, 2005.

[11] J. Gao, X. Li, L. Wang, S. Fidler, and S. Lin. Mimicking the in-camera color pipeline for camera-aware object compositing. *arXiv:1903.11248*, 2019.

[12] L. Gruber, D. Kalkofen, and D. Schmalstieg. Color harmonization for augmented reality. In *Proc. Int. Symp. on Mixed and Augmented Reality*, pp. 227–228, 2010.

[13] A. Hore and D. Ziou. Image quality metrics: PSNR vs. SSIM. In *Int. Conf. on Pattern Recognition*, pp. 2366–2369, 2010.

[14] A. Isaksen, L. McMillan, and S. J. Gortler. Dynamically reparameterized light fields. In *Proc. Int. Conf. on Comput. Graph. and Interactive Techniques*, pp. 297–306, 2000. doi: 10.1145/344779.344929

[15] Y. Itoh, T. Langlotz, J. Sutton, and A. Plopski. Towards indistinguishable augmented reality: A survey on optical see-through head-mounted displays. *ACM Comput. Surv.*, 54(6), July 2021. doi: 10.1145/3453157

[16] D. Kalkofen, M. Tatzgern, and D. Schmalstieg. Explosion diagrams in augmented reality. In *Proc. IEEE Virtual Reality*, pp. 71–78, 2009.

[17] P. Kán and H. Kaufmann. High-quality reflections, refractions, and caustics in augmented reality and their contribution to visual coherence. In *Proc. Int. Symp. on Mixed and Augmented Reality*, pp. 99–108, 2012.

[18] N. Kholgade, T. Simon, A. Efros, and Y. Sheikh. 3D object manipulation in a single photograph using stock 3D models. *ACM Transactions on Graphics*, 33(4), July 2014.

[19] G. Klein and D. W. Murray. Simulating low-cost cameras for augmented reality compositing. *IEEE Transactions on Visualization and Computer Graphics*, 16(3):369–380, 2010.

[20] M. Knecht, C. Traxler, O. Mattausch, W. Purgathofer, and M. Wimmer. Differential instant radiosity for mixed reality. In *Proc. Int. Symp. on Mixed and Augmented Reality*, pp. 99–107, 2010.

[21] M. Knecht, C. Traxler, W. Purgathofer, and M. Wimmer. Adaptive camera-based color mapping for mixed-reality applications. In *Proc. Int. Symp. on Mixed and Augmented Reality*, pp. 165–168, 2011.

[22] M. Knecht, C. Traxler, C. Winklhofer, and M. Wimmer. Reflective and refractive objects for mixed reality. *IEEE Transactions on Visualization and Computer Graphics*, 19(4):576–582, 2013.

[23] J. Kronander, F. Banterle, A. Gardner, E. Miandji, and J. Unger. Photorealistic rendering of mixed reality scenes. *Computer Graphics Forum*, 34(2):643–665, 2015.

[24] D. Mandl, K. M. Yi, P. Mohr, P. Roth, P. Fua, V. Lepetit, D. Schmalstieg, and D. Kalkofen. Learning lightprobes for mixed reality illumination. In *Proc. Int. Symp. on Mixed and Augmented Reality*, pp. 82–89, 2017.

[25] A. Meka, M. Zollhöfer, C. Richardt, and C. Theobalt. Live intrinsic video. *ACM Transactions on Graphics*, 35(4), 2016.

[26] S. Nam and S. Joo Kim. Modelling the scene dependent imaging in cameras with a deep neural network. In *Proc. Int. Conf. on Comput. Vision*, pp. 1717–1725, 2017.

[27] R. Ng. Fourier slice photography. *ACM Transactions on Graphics*, 24(3):735–744, July 2005.

[28] B. Okumura, M. Kanbara, and N. Yokoya. Augmented reality based on estimation of defocusing and motion blurring from captured images. In *Proc. Int. Symp. on Mixed and Augmented Reality*, pp. 219–225, 2006.

[29] Y. Park, V. Lepetit, and W. Woo. Handling motion-blur in 3d tracking and rendering for augmented reality. *IEEE Transactions on Visualization and Computer Graphics*, 18(9):1449–1459, 2011.

[30] R. Raskar and M. Cohen. Image precision silhouette edges. In *Proc. Symp. on Interactive 3D Graphics*, pp. 135–140, 1999.

[31] H. Regenbrecht, K. Meng, A. Reepen, S. Beck, and T. Langlotz. Mixed voxel reality: Presence and embodiment in low fidelity, visually coherent, mixed reality environments. In *Proc. Int. Symp. on Mixed and Augmented Reality*, pp. 90–99, 2017.

[32] G. Reitmayr and T. W. Drummond. Going out: Robust model-based tracking for outdoor augmented reality. In *Proc. Int. Symp. on Mixed and Augmented Reality*, pp. 109–118, 2006.

[33] T. Rhee, L. Petikam, B. Allen, and A. Chalmers. MR360: Mixed reality rendering for 360° panoramic videos. *IEEE Transactions on Visualization and Computer Graphics*, 23(4):1379–1388, 2017.

[34] T. Richter-Trummer, D. Kalkofen, J. Park, and D. Schmalstieg. Instant mixed reality lighting from casual scanning. In *Proc. Int. Symp. on Mixed and Augmented Reality*, pp. 27–36, 2016.

[35] G. Riguer, N. Tatarchuk, and J. Isidoro. Real-time depth of field simulation. In *ShaderX2: Shader Programming Tips and Tricks with DirectX*, vol. 9, pp. 529–556. Wordware Publishing, Inc., 2004.

[36] K. Rohmer, J. Jendersie, and T. Grosch. Natural environment illumination: Coherent interactive augmented reality for mobile and non-mobile devices. *IEEE Transactions on Visualization and Computer Graphics*, 23(11):2474–2484, 2017.

[37] T. Saito and T. Takahashi. Comprehensible rendering of 3-d shapes. *Computer Graphics, Proceedings of ACM Siggraph 1990*, 24(4):197–206, 1990.

[38] T. Scheuermann and N. Tatarchuk. Improved depth of field rendering. In *ShaderX3: Advanced Rendering with DirectX and OpenGL (Shaderx Series)*. Charles River Media, 2004.

[39] D. Schmalstieg and T. Höllerer. *Augmented reality: Principles and practice*. Addison-Wesley Professional, 2016.

[40] E. Schwartz, R. Giryes, and A. M. Bronstein. DeepISP: toward learning an end-to-end image processing pipeline. *IEEE Transactions on Image Processing*, 28(2):912–923, 2019.

[41] W. Steptoe, S. Julier, and A. Steed. Presence and discernability in conventional and non-photorealistic immersive augmented reality. In *Proc. Int. Symp. on Mixed and Augmented Reality*, pp. 213–218, 2014.

[42] Y. Takeuchi and K. Perlin. ClayVision: The (elastic) image of the city. In *Proc. ACM Conf. on Human Factors in Computing Systems*, pp. 2411–2420, 2012.

[43] L. Wang, R. Li, X. Shi, L.-Q. Yan, and Z. Li. Foveated instant radiosity. In *Proc. Int. Symp. on Mixed and Augmented Reality*, 2020.

[44] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[45] B. A. Watson and L. F. Hodges. Using texture maps to correct for optical distortion in head-mounted displays. In *Proc. Virtual Reality Annual International Symposium*, pp. 172–178, 1995.

[46] L. Xiao, A. Kaplanyan, A. Fix, M. Chapman, and D. Lanman. DeepFocus: Learned image synthesis for computational displays. *ACM Transactions on Graphics*, 37(6), 2018.

[47] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.