BokehMe: When Neural Rendering Meets Classical Rendering

Juewen Peng[1], Zhiguo Cao[1], Xianrui Luo[1], Hao Lu[1], Ke Xian[1]*, and Jianming Zhang[2]
[1]Key Laboratory of Image Processing and Intelligent Control, Ministry of Education,
School of Artificial Intelligence and Automation, Huazhong University of Science and Technology
[2]Adobe Research
{juewenpeng, zgcao, xianruiluo, hlu, kexian}@hust.edu.cn, jianmzha@adobe.com
https://github.com/JuewenPeng/BokehMe

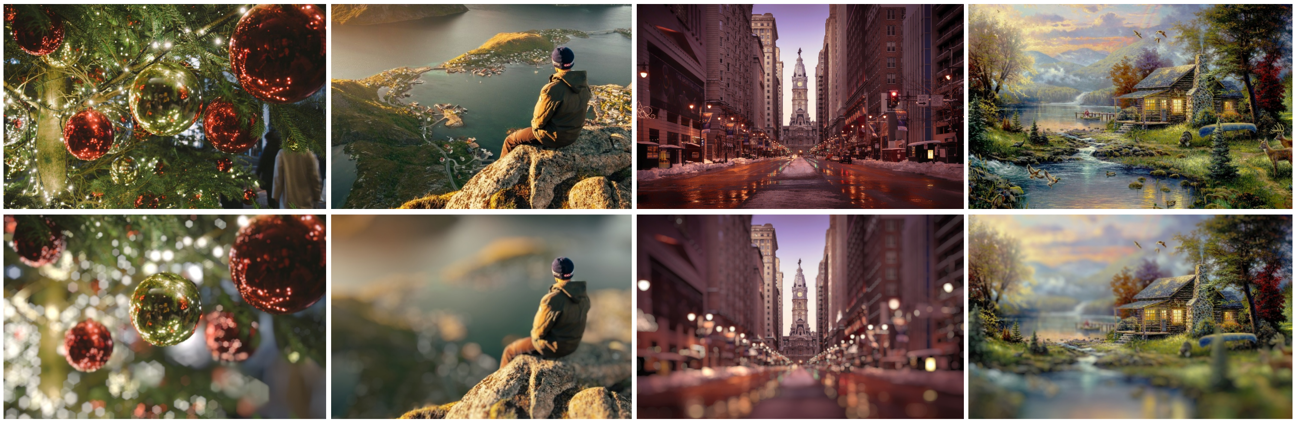


Figure 1. BokehMe creates photo-realistic and highly controllable bokeh effects from high-resolution images and imperfect disparity maps predicted by DPT [26]. The first column shows the result with a hexagon aperture shape, and the rest of them use a circular shape.

Abstract

We propose BokehMe, a hybrid bokeh rendering framework that marries a neural renderer with a classical physically motivated renderer. Given a single image and a potentially imperfect disparity map, BokehMe generates high-resolution photo-realistic bokeh effects with adjustable blur size, focal plane, and aperture shape. To this end, we analyze the errors from the classical scattering-based method and derive a formulation to calculate an error map. Based on this formulation, we implement the classical renderer by a scattering-based method and propose a two-stage neural renderer to fix the erroneous areas from the classical renderer. The neural renderer employs a dynamic multi-scale scheme to efficiently handle arbitrary blur sizes, and it is trained to handle imperfect disparity input. Experiments show that our method compares favorably against previous methods on both synthetic image data and real image data with predicted disparity. A user study is further conducted to validate the advantage of our method.

1. Introduction

Bokeh effect refers to the way the lens renders the out-of-focus blur in a photograph (Fig. 1). With different lens designs and configurations, various bokeh styles can be created. For example, the shape of the bokeh ball can be controlled by the aperture. Classical rendering methods [6, 20, 31, 40] can change bokeh styles easily by controlling the shape and size of the blur kernel. However, they often suffer from artifacts at depth discontinuities. Neural rendering methods [11,25,32] can address this problem well by learning from image statistics, but they have difficulty simulating real bokeh balls and can only produce the bokeh style from the training data. In addition, previous neural rendering methods lack a mechanism to produce large blur size on high-resolution images, because of the fixed receptive field of the neural network and the blur size limit of the training data.

To produce artifact-free and highly controllable bokeh effects, we propose a novel hybrid framework, termed BokehMe, which makes the best of the two worlds by fusing the results from a classical renderer and a neural renderer (Fig. 2). We use the scattering-based method [31]

*Corresponding author.

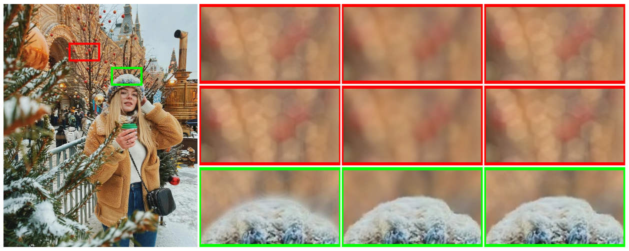


Figure 2. BokehMe combines a classical renderer (CR) and a neural renderer (NR) to create bokeh effects with stunning bokeh balls and adjustable aperture shapes (row 1: circle; row 2: hexagon).

as our classical renderer. To determine where this method may render noticeable boundary artifacts, we model the lens system and conduct a comprehensive analysis of the error between scattering-based rendering and real rendering. A soft but tight error map is derived to identify regions with boundary artifacts. Using the error map to replace the artifact region with the neural rendering result, we are able to preserve the bokeh style from the classical renderer without apparent visual artifacts. For the neural renderer, to break the blur size limit, we decompose it to two sub-networks: adaptive rendering network (ARNet) and iterative upsampling network (IUNet). In ARNet, we resize the input images adaptively and generate a bokeh image in low resolution. Then, IUNet is used to upsample the low-resolution bokeh image iteratively guided by the initial high-resolution input images. As a result, our neural renderer can handle arbitrarily large blur sizes.

Our main contributions are summarized as follows.

- We propose a novel framework, which combines a classical renderer and a neural renderer for photo-realistic and highly controllable bokeh rendering.
- We analyze the lens system and propose an error map formulation to effectively fuse the classical rendering and the neural rendering.
- We propose a two-stage neural renderer which uses adaptive resizing and iterative upsampling to handle arbitrary blur sizes for high-resolution images, and it is robust to potentially imperfect disparity input.

In addition, due to the lack of test data in the field of controllable bokeh rendering, we contribute a new benchmark: BLB, synthesized by Blender 2.93 [5], together with EBB400, processed from EBB! [11]. Since the evaluation of bokeh effects is subjective, we also conduct a user study on images captured by iPhone 12. Extensive results show that BokehMe can render images that appear physically sound and maintain the diversity of the bokeh style.

2. Related Work

Classical Rendering. Classical rendering can be classified into two categories: object space methods and image space ones. Object space methods [1, 16, 33, 39], based on ray tracing, render exact results. However, most are time-consuming and require complete 3D scene information, resulting in poor practicality. Compared with object space methods, image space ones [3, 4, 10, 29, 38] only require a single image and its corresponding depth map, which are easier to implement. In recent years, more and more methods [6, 20, 23, 27, 28, 31, 34, 40] combine different modules, such as depth estimation, semantic segmentation, and classical rendering, to construct an automatic rendering system. To prevent the color of background from bleeding into foreground, most methods decompose the image to multiple layers conditioned on the estimated depth map, and execute rendering from back to front.

Despite the fact that classical rendering is flexible, this paradigm suffers from artifacts at depth discontinuities, especially when the focal plane targets background.

Neural Rendering. To improve efficiency and avoid boundary artifacts, many recent works use neural networks to simulate the rendering process. For example, Nalbach *et al.* [21] and Xiao *et al.* [35] train networks to produce a bokeh effect from an all-in-focus image and its corresponding perfect depth map. By training on the synthetic data created by OpenGL shaders and Unity Engines [30], boundary artifacts can be effectively alleviated. However, perfect depth maps are not always easy to obtain in the real world. Wang *et al.* [32] thus propose an automatic rendering system comprised of depth prediction, lens blur, and guided upsampling to generate high-resolution depth-of-field (DoF) images from a single image. Besides, encoder-decoder networks [9, 11–13, 25], that map all-in-focus images into shallow DoF images in an end-to-end manner, have also been studied recently. Unlike aforementioned methods, Xu *et al.* [37] focus on fully automatic portrait rendering. They use recurrent filters [18] to approximate the conditional random field-based rendering method and achieve a significant speed improvement.

However, the main problem of neural rendering is lack of controllability. For a trained neural network, the bokeh style cannot be changed and the blur range is limited. In addition, bokeh balls produced by the network are not real as the network tends to learn a simple fuzzy effect.

3. BokehMe: A Hybrid Rendering Framework

As shown in Fig. 3, our framework generates a bokeh image B from an all-in-focus image I, a disparity map D, and controlling parameters via two renderers: a classical renderer and a neural renderer. Their rendered results are fused based on an error map E that identifies the potentially

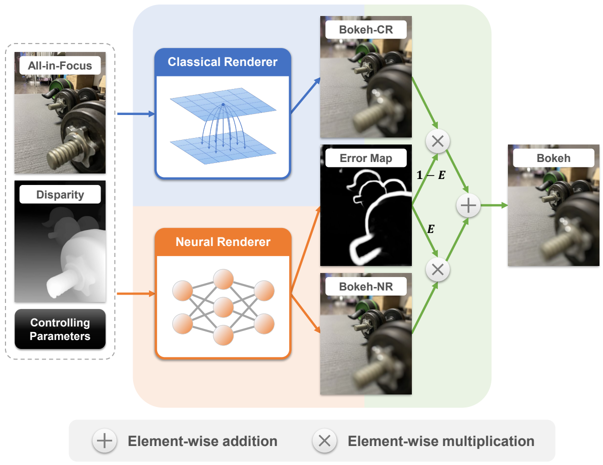


Figure 3. Framework of BokehMe. The bokeh image is obtained by fusing the outputs of a classical renderer and a neural renderer.

erroneous areas from the classical renderer. The controlling parameters include blur parameter K, refocused disparity d_f, gamma value γ, and some parameters about the bokeh style, *e.g.*, aperture shape. Specifically, K reflects the blur amount of the whole image. d_f determines the disparity (inverse depth) of the focal plane. γ, used in gamma correction, controls the brightness and salience of bokeh balls.

3.1. Classical Renderer and Error Analysis

Classical Renderer. We expect the classical renderer to focus on rendering realistic bokeh effects in depth-continuous areas. After comparing different methods, we find pixel-wise rendering methods based on scattering [23, 31] have relatively small error in these areas despite causing serious color-bleeding artifacts at depth discontinuities. The core idea of this method is scattering each pixel to its neighbor areas where the distance between them is less than the blur radius of the pixel. As discussed in [31, 38], given the disparity d of a pixel, its blur radius can be calculated by

$$r = K\left|d - d_f\right|. \tag{1}$$

We implement the algorithm with CuPy package to achieve a significant parallel speedup (refer to the supplementary material). Since the transformation from scene irradiance to image intensity is nonlinear [38], an additional gamma correction [17] is applied before and after the rendering.

Lens System. To understand why scattering-based methods cause error at depth discontinuities, we model a virtual lens system. For a simple scenario (Fig. 4) where two objects exist in space, we derive 8 rendering cases at depth discontinuities (2 cases are shown here while the others are shown in the supplementary material). Taking the center pixel (the black dot) as an example, only neighbor pixels on red gradient foreground plane and those on blue gradient background plane can pass to the center pixel. Apparently, the scattering-based rendering is different from the real one.

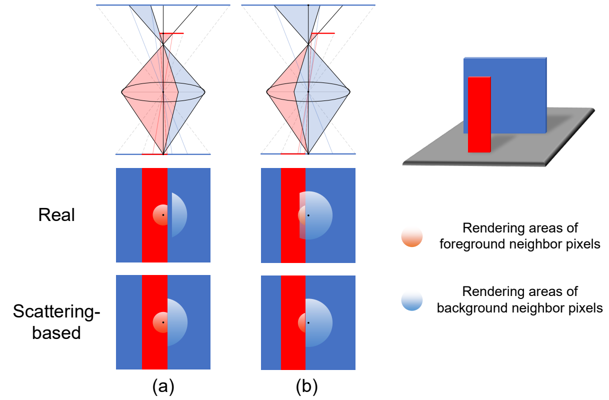

Real
Scattering-based
(a)
(b)
Rendering areas of foreground neighbor pixels
Rendering areas of background neighbor pixels

Figure 4. Comparison of real rendering and scattering-based rendering at depth discontinuities. Rendered result of the center pixel (the black dot) is the integration of its neighbor pixels on red gradient foreground plane and those on blue gradient background plane.

Initial Error Map. We aim to obtain an error map to identify areas rendered incorrectly by the classical renderer. Later, we will train a neural network to predict the error map formulated in this section. Let E^* denote the target error map. Since only regions within the scattering radius from the depth boundary may have significant difference from the real rendering, E^* can be conservatively formulated as the spatially variant dilation of the depth boundary, and the dilation size depends on the maximum blur radius of the pixels located on both sides of the depth boundary. Take the scenario in Fig. 4 as an example, the i-th element of E^* can be defined by

$$E_i^* = \mathbb{1}\big(\alpha_i < 1\big), \quad \alpha_i = \frac{l_{ii'}}{\max\left(r_i, r_{i'}\right)}, \tag{2}$$

where α_i can be treated as a variable of E_i^*. i' is the index to the nearest pixel of the i-th pixel in the other depth plane. $l_{ii'}$ is the distance between the two pixels. r_i and $r_{i'}$ are the blur radii of the corresponding pixels.

Improved Error Map. Considering the fact that the classical renderer generates high-quality results in depth-continuous regions with controllable bokeh style, we would like to appropriately narrow and soften the initial error map to preserve more of the bokeh result from the classical renderer without obvious artifacts in fusion boundary.

Through the theoretic and numerical analysis shown in the supplementary material, we derive that for each pixel, the color difference between the scattering-based rendering result and the real rendering result is

$$H_i = k_i\left|c_i - c_{i'}\right|, \quad k_i = f(\alpha_i, \beta_i), \tag{3}$$

where c_i and $c_{i'}$ are the colors of the i-th pixel and the i'-th pixel before rendering. k_i is a function of two variables α_i

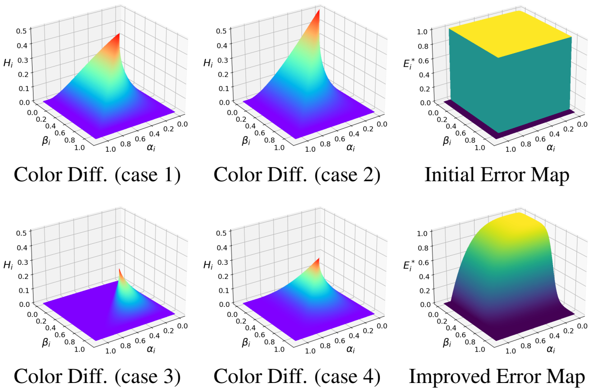


Figure 5. Column 1, 2: graphs of the color difference H_i between real rendering and scattering-based rendering in 4 cases. Column 3: graphs of the error map E_i^* w.r.t. the erroneous areas from the classical renderer. The improved error map is softer and tighter than the initial one, and covers the color difference on the whole.

and β_i. α_i has been defined in Eq. 2 while β_i takes the form

$$\beta_i = \frac{\min(r_i, r_{i'})}{\max(r_i, r_{i'})}\,, \tag{4}$$

which represents the ratio of the smaller and the larger blur radius of the two pixels. k_i varies with the refocused disparity and the shortest distance between the processing pixel and the depth boundary. For clarity, we assume $|c_i - c_{i'}| = 1$ and draw the graphs of H_i in the first two columns of Fig. 5. Based on the observation that H_i is reduced with the increase of α_i and β_i, we heuristically rewrite Eq. 2 to

$$E_i^* = \max\left(0, 1 - {\alpha_i}^{\delta_1}\right) \cdot \mathbb{1}\left(\beta_i < \delta_2\right), \tag{5}$$

where δ_1 and δ_2 are two hyperparameters. This formula will be equivalent to Eq. 2 if setting $\delta_1 = \infty$ and $\delta_2 = 1$. Note that in our implementation, we replace the second indicator function term with a smooth one, *i.e.*, $0.5 + 0.5\tanh(10(\delta_2 - \beta_i))$. After comparing the model trained with different hyperparameters (in the supplementary material), we empirically set $\delta_1 = 4$ and $\delta_2 = \frac{2}{3}$. We also show the graphs of the initial E_i^* (Eq. 2) and the improved E_i^* (Eq. 5) in the last column of Fig. 5. Note that as $0 \leq \beta_i \leq 1$, we define $E_i^* = 0$ if $\beta_i > 1$. One can observe that the improved E_i^* is softer and tighter than the initial one, and still covers the area with large color difference. An additional practical example is shown in Fig. 6.

3.2. Neural Renderer and Model Training

To handle the rendering at depth discontinuities and overcome the limitations of the blur range, we propose a neural renderer consisting of two sub-networks: ARNet and IUNet (Fig. 7). To simplify the input of the neural renderer, we define a signed defocus map S based on Eq. 1:

$$S = K(D - d_f)\,, \tag{6}$$

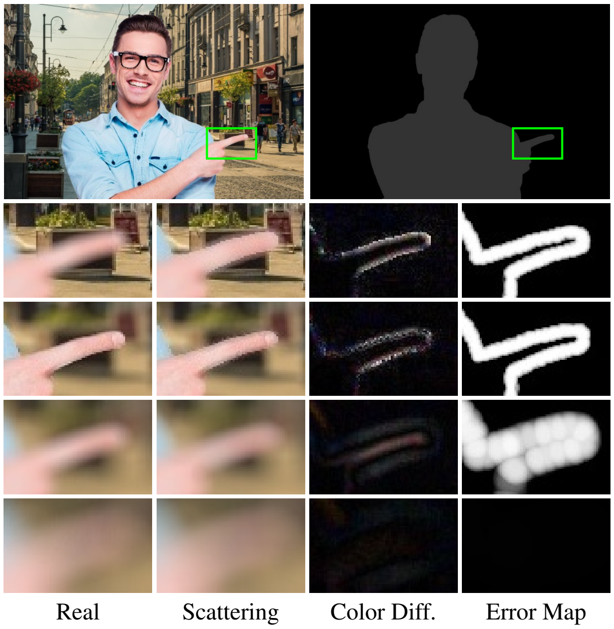


Figure 6. In this example, the disparities of background and foreground are fixed to 0 and 0.2, respectively. The refocused disparity is set to 0, 0.2, 0.5 and 1 from row 2 to row 5, and the variable β_i for each case can be calculated by Eq. 4, *i.e.*, 0, 0, 0.6 and 0.8. One can see that the color difference between real rendering and scattering-based rendering "fades out" with the increase of β_i, and our improved error map can cover the color difference on the whole, which is consistent with the observation in Fig 5.

which encodes the information about the depth relationship and the spatially variant blur radius. To match the gamma correction in the classical renderer, we use a map filled with the normalized gamma value as an additional input.

ARNet resizes the input images adaptively, and outputs an error map and a bokeh image B_{nr}^{lr} in low resolution (Fig. 8). The adaptive resizing layer consists of two steps. The first step is to calculate the downscale factor

$$w^{(0)} = \min\left(1, \frac{\max(|S|)}{\hat{R}}\right), \tag{7}$$

where $\max(|S|)$ corresponds to the maximum blur radius of the whole image. $\hat{R}$ is the maximum blur radius we set for the neural network. The second step is to downsample all images and reduce the numerical range of the signed defocus map by the ratio of $w^{(0)}$. The middle part of the network is lightweight and replaceable. We use the same architecture as DeepFocus (fast version) [35] in this work.

IUNet iteratively upsamples the low-resolution bokeh image B_{nr}^{lr} by a factor of 2 until reaching the original resolution (Fig. 9). To avoid the fuzziness around in-focus areas caused by direct bilinear upsampling, we use the original high-resolution input as a guidance map. In each iteration, it is resized to twice the resolution of the input bokeh im-

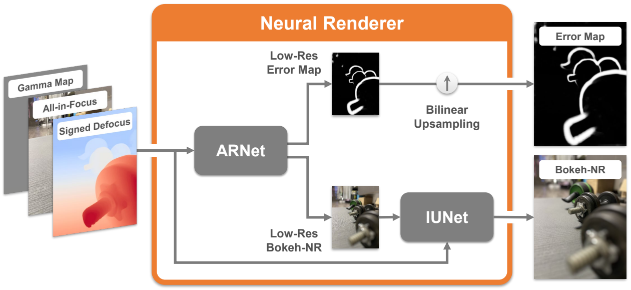


Figure 7. Architecture of the neural renderer. ARNet first estimates a low-resolution bokeh image and an low-resolution error map. Then, the error map is restored to original resolution by bilinear upsampling, while the bokeh image is upsampled by IUNet.

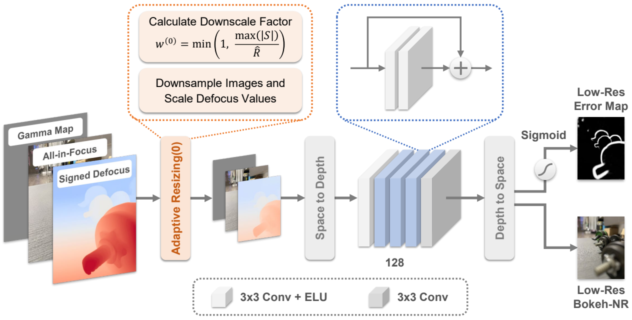


Figure 8. Architecture of ARNet. The adaptive resizing layer downsamples the input images and reduces the numerical range of the signed defocus map to ensure that the defocus values are in the acceptable range of the neural network without decreasing the blur amount of the whole image.

age. To match the increasing blur size during the iteration, we also need to dynamically adjust the values of the defocus map. Specifically, we once again use the adaptive resizing layer, and the downscale factor of each iteration t is set as

$$w^{(t)} = \frac{1}{2}\, w^{(t-1)}, \quad t = 1, ..., T\,. \tag{8}$$

However, with the progress of iteration, the scaled defocus values may exceed the acceptable range $[-\hat{R}, \hat{R}]$ of the neural network. Fortunately, the fuzziness caused by direct bilinear upsampling is unnoticeable for the areas with large amount of bokeh blur. Thus, we can just refine the areas whose defocus values are in the range. To this end, we first clip the out-of-range defocus values to ensure that the subsequent network can work without collapse. Then, we threshold the dilated defocus map S^d to produce a mask, which indicates the effectively rendered areas without defocus clipping. In these areas, we use the output of the network, while for the rest of the areas, we use the input bokeh image after bilinear upsampling. Here, we use S^d instead of S because the negative effects caused by defocus clipping will spread during the rendering. The detailed calculation of S^d is in the supplementary material. Overall, with the increase of iteration, the resolution of the bokeh image will be higher, but the effective area refined by the network will be smaller. In other words, areas nearby the focal plane will be refined more times.

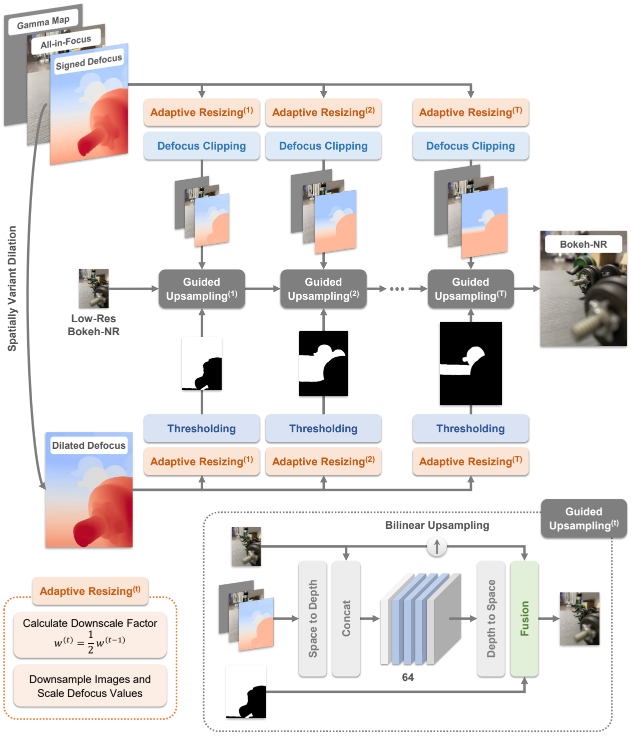


Figure 9. Architecture of IUNet. The low-resolution bokeh image will be upsampled iteratively to generate a high-quality high-resolution bokeh image. In each iteration, the defocus clipping layer aims to prevent the scaled defocus values from exceeding the acceptable range of the subsequent network, and the thresholding layer produces a mask to replace the rendered result within clipping areas with the bilinear upsampled input bokeh image.

Finally, following alpha blending [19, 36], we use the predicted error map E to fuse the bokeh results of the classical renderer B_{cr} and that of the neural renderer B_{nr}:

$$B = (1 - E) \cdot B_{cr} + E \cdot B_{nr}\,. \tag{9}$$

Loss Functions. We train ARNet and IUNet separately. When training ARNet, the adaptive resizing layer is unused. B is fused by B_{cr} and B_{nr}^{lr}. The loss is defined by

$$\begin{aligned} \mathcal{L}_{AR} &= \mathcal{L}_{\ell_1}(B, B^*) + \mathcal{L}_{\ell_1}(\nabla B, \nabla B^*) \\ &+ \mathcal{L}_{\ell_1}(B_{nr}^{lr}, B^*) + \mathcal{L}_{\ell_1}(\nabla B_{nr}^{lr}, \nabla B^*) \\ &+ \lambda_{bce}\, \mathcal{L}_{bce}(E, E^*)\,, \end{aligned} \tag{10}$$

where ground-truth maps are marked with a superscript $*$. ∇ denotes the image gradient. λ_{bce} is empirically set to 0.1. When training IUNet, we freeze ARNet and use the following loss:

$$\begin{aligned} \mathcal{L}_{IU} &= \mathcal{L}_{\ell_1}(B, B^*) + \mathcal{L}_{\ell_1}(\nabla B, \nabla B^*) \\ &+ \mathcal{L}_{\ell_1}(B_{nr}, B^*) + \mathcal{L}_{\ell_1}(\nabla B_{nr}, \nabla B^*)\,. \end{aligned} \tag{11}$$

Table 1. Quantitative results on the BLB dataset. Different levels correspond to different blur parameters of bokeh images, *e.g.*, “Level 1” denotes that the blur parameter is 10, and “Level 5” denotes that the blur parameter is 50. The best performance is in **boldface**.

Methods	Level 1			Level 2			Level 3			Level 4			Level 5		
	PSNR	SSIM	Time(s)	PSNR	SSIM	Time(s)	PSNR	SSIM	Time(s)	PSNR	SSIM	Time(s)	PSNR	SSIM	Time(s)
VDSLR [38]	41.13	0.9891	0.06	39.15	0.9848	0.23	37.64	0.9812	0.53	36.48	0.9783	0.97	35.57	0.9760	1.55
SteReFo [6]	37.21	0.9831	0.13	35.28	0.9818	0.60	33.99	0.9813	1.69	32.94	0.9809	3.74	32.12	0.9805	6.87
RVR [40]	32.35	0.9648	0.10	32.00	0.9321	0.43	28.36	0.9011	1.11	25.80	0.8775	2.30	23.94	0.8596	4.12
RVR† [40]	37.15	0.9836	0.13	38.55	0.9880	0.62	35.56	0.9854	1.82	33.03	0.9815	3.97	31.15	0.9774	7.21
DeepLens [32]	33.68	0.9679	0.14	31.43	0.9603	0.14	30.16	0.9564	**0.14**	29.30	0.9539	0.14	28.68	0.9521	0.14
DeepFocus [35]	38.92	0.9900	0.71	36.13	0.9857	0.71	31.47	0.9623	0.71	25.55	0.9089	0.71	21.04	0.8227	0.71
DeepFocus† [35]	38.92	0.9900	0.71	35.74	0.9861	0.49	34.21	0.9833	0.22	33.21	0.9809	**0.13**	32.44	0.9788	**0.09**
Ours (CR)	41.32	0.9900	**0.03**	39.51	0.9877	**0.10**	38.35	0.9868	0.20	37.53	0.9864	0.34	36.86	0.9862	0.52
Ours (NR)	40.41	0.9905	0.13	40.16	0.9904	0.13	39.21	0.9896	**0.14**	38.01	0.9884	0.16	37.20	0.9875	0.16
Ours	**43.30**	**0.9932**	0.16	**42.21**	**0.9924**	0.23	**41.02**	**0.9915**	0.34	**39.78**	**0.9906**	0.50	**38.80**	**0.9898**	0.68

Note that, for fast convergence, we supervise the training of both ARNet and IUNet with the intermediate result B_{nr}^{lr} or B_{nr}, aside from the final result B.

Implementations. Our implementation is based on PyTorch [22]. To train the neural renderer, we synthesize a bokeh dataset using a simplified ray tracing method. This dataset contains 150 scenes. For each scene, it consists of an all-in-focus image, a disparity map ranging from 0 to 1, and a stack of bokeh images with 2 blur parameters (12, 24), 20 refocused disparities (0.05, 0.1, ..., 1), and 5 gamma values (1, 2, ..., 5). We use the data with the blur parameter of 12 for ARNet training and 24 for IUNet training. We follow the same data pre-processing configurations as in [35]. To improve the generalization, we additionally augment the input disparity map with random gaussian blur, dilation and erosion. The acceptable defocus ranges of ARNet and IUNet are both set to $[-12, 12]$ for training and $[-10, 10]$ for inference. Both networks are trained for 50 epochs with a batch size of 16. The learning rate is set to 10^{-4}. Adam optimizer [14] is used for optimization. All experiments are conducted on a NVIDIA GeForce GTX 1080 Ti GPU.

4. Experiments

4.1. Test Data

For all test data, without loss of generality, we assume that the aperture shape is circular and the gamma value is 2.2 to create a level playing field for different methods. The disparity maps of all datasets are normalized to $[0, 1]$ range.

BLB contains 500 test samples synthesized by Blender 2.93 [5]. Specifically, we download 10 3D scene models of Blender splash screens from different versions [2]. For each scene model, we use Cycles Engine [5] to render an all-in-focus image, a disparity map, and a stack of bokeh images with 5 blur parameters and 10 refocused disparities. The image resolution is set to 1920 × 1080.

EBB400 contains 400 wide and shallow DoF image pairs which are randomly selected from EBB! [11]. For each sample, we predict a disparity map by DPT [26], and manually label a bounding box referred to the in-focus areas, so that we can obtain the refocused disparity by taking the median value of the disparity map within the bounding box [23]. The image resolution is about 1536×1024.

IPB contains 40 images captured by iPhone 12 Portrait mode. For each scene, we first export an all-in-focus image and a bokeh image post-processed by the Portrait mode from iPhone 12. Then, using the online photo editor Photopea [24], we can further extract a disparity map and an irradiance map from the bokeh image. All images are shot vertically with the resolution of 3024×4032.

4.2. Compared Methods

We compare BokehMe with two types of methods: classical rendering methods and neural rendering methods. For simplicity, we represent them as “C” and “N” in the following. For a fair comparison, we provide the same disparity map for all methods, and we only preserve their bokeh rendering modules, while the others are discarded.

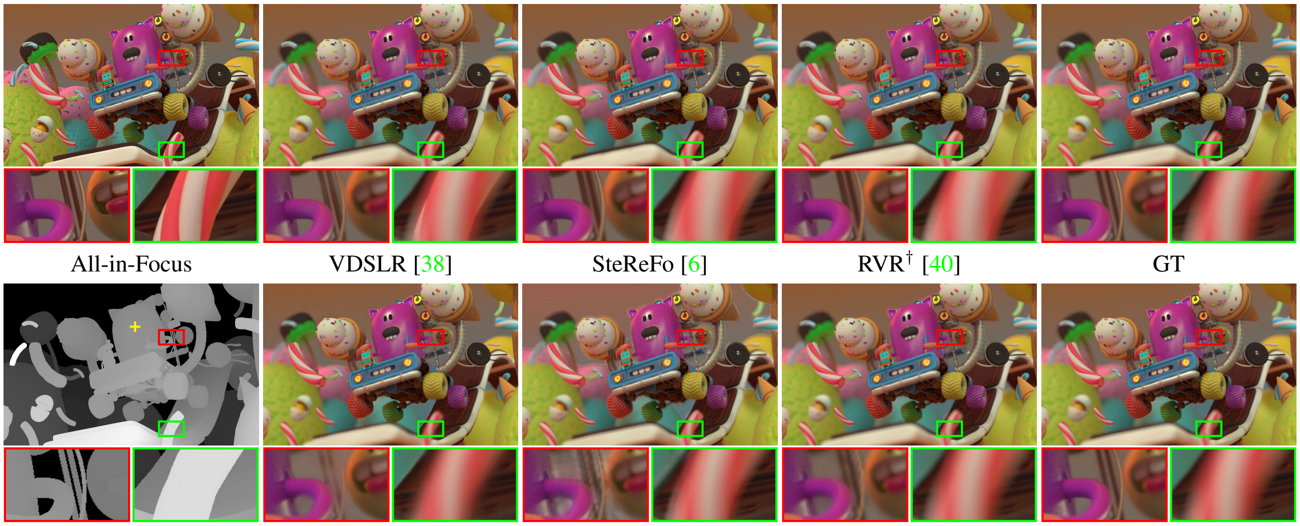
VDSLR [38] (C) is a pixel-wise pseudo ray tracing method accelerated by randomized intersection searching.

SteReFo [6] (C) decomposes the image into layers according to the depth and renders the image from back to front.

RVR [40] (C) is similar to SteReFo. However, as discussed in [34], original RVR lacks weight normalization, resulting in serious artifacts among different depth layers, so we add extra weight normalization as in SteReFo, and mark this modified method with superscript †.

DeepLens [32] (N) is trained on a homemade synthetic dataset and can generate high-resolution outputs.

DeepFocus [35] (N) is trained on Unity data [30]. As DeepFocus cannot handle large blur sizes, we apply the adaptive resizing layer proposed in our paper to the head of its model, and upsample its result to original resolution directly. Similarly, this modified method is marked with superscript †.


All-in-Focus VDSLR [38] SteReFo [6] RVR† [40] GT

Disparity DeepLens [32] DeepFocus [35] DeepFocus† [35] Ours

Figure 10. Qualitative results on the BLB dataset. The rough refocused plane is labelled with a yellow cross on the disparity map.

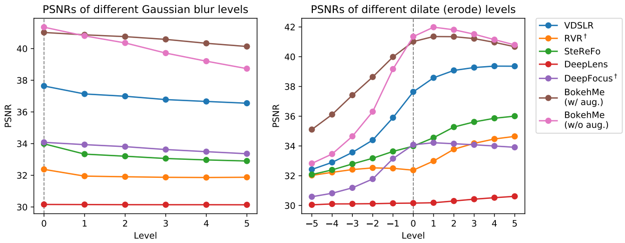


Figure 11. Evaluation on the BLB dataset with corrupting disparity maps. In the right chart, positive levels correspond to dilation levels while negative levels correspond to erosion levels.

4.3. Zero-shot Cross-dataset Evaluation

Following [6, 35, 37], we use PSNR and SSIM as metrics. We test BokehMe on the BLB dataset. As shown in Table 1, BokehMe achieves the best PSNR and SSIM scores compared with other state-of-the-art methods among all levels of blur, and our final model incorporating the classical renderer and the neural renderer outperforms either separate one, demonstrating strong complementarity between the two renderers. In addition, as the level of blur increases, the classical rendering methods become more time-consuming, while the neural rendering methods maintain high efficiency. We also show some visual results in Fig. 10. One can observe: (i) The performance of classical methods degrades at depth discontinuities when the background is refocused on; (ii) DeepLens renders smooth results at depth discontinuities, but they seem not in line with the actual rendering; (iii) Compared with DeepFocus, DeepFocus† avoids corruption in processing large blur sizes but generates blurry results around in-focus areas. (iv) Our approach renders most realistic bokeh effects for both in-focus and out-of-focus areas.

Since it is hard to acquire a disparity map in the real world, a common practice is to estimate one. Nevertheless, the predicted disparity map may be blurry and not align with the RGB image at boundary. Therefore, we redo the "Level 3" experiment (Table. 1) by corrupting the disparity map with 5 levels of gaussian blur, dilation and erosion, respectively. We also retrain BokehMe without disparity augmentation for extra comparison. As shown in Fig. 11, BokehMe trained with augmentation better adapts to imperfect disparity maps. Another interesting observation is that the moderate dilation improves the performance of most methods, especially for the classical ones. The reason may be that the dilated pixels that extend beyond the boundaries of the foreground object act as the occluded background pixels, leading to a significant improvement of metrics in case of background refocusing. However, as shown in Fig. 12, it causes more boundary artifacts at the same time.

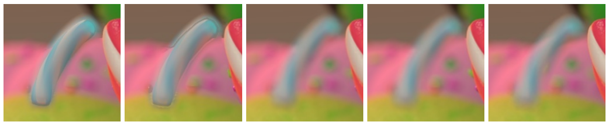

VDSLR VDSLR^{d_3} Ours Ours^{d_3} GT

Figure 12. Rendered results after dilating the disparity map on the BLB dataset. "d_3" means the level of dilation is 3 (the kernel size is 7×7). The image originates from Fig. 10.

Table 2. Quantitative results on the EBB400 dataset.

Methods	VDSLR	SteReFo	RVR†	DeepLens	DeepFocus†	Ours
PSNR	23.78	23.56	23.56	23.46	23.81	**23.85**
SSIM	0.8738	0.8674	0.8690	0.8707	0.8754	**0.8770**

To further evaluate the generalization of the model given an imperfect disparity map as input, we compare different methods on the EBB400 dataset where disparity maps are predicted by DPT [26]. As the blur parameter of each sample is unknown, we pick out the optimal value from 1 to 100

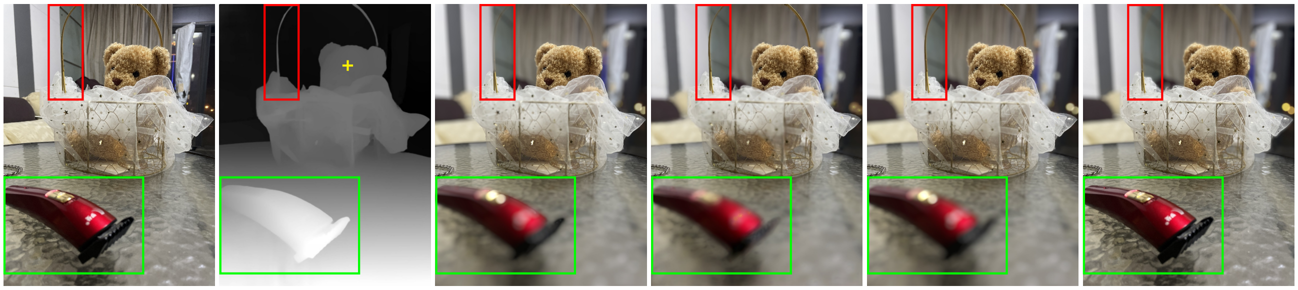


Figure 13. An example of user study on the IPB dataset. The rough refocused plane is labelled with a yellow cross on the disparity map.

Table 3. Ablation study of IUNet. "B-Up": bilinear upsampling; "I-Up": iterative upsampling by IUNet; "Clip": clipping of the signed defocus map; "S-Fuse": fusion with the mask thresholded by the signed defocus map; "D-Fuse": fusion with the mask thresholded by the dilated defocus map.

No.	B-Up	I-Up	Clip	S-Fuse	D-Fuse	PSNR	SSIM
B1	✓					37.30	0.9830
B2		✓				23.23	0.8719
B3		✓	✓			38.55	0.9883
B4		✓	✓	✓		39.19	0.9894
B5		✓	✓		✓	**39.21**	**0.9896**

for each method. Despite the fact that color inconsistency and scene misalignment exist between the wide and shallow DoF image pairs, BokehMe still ranks the first in both metrics as shown in Table 2. Refer to the supplementary material for qualitative results.

4.4. Ablation Study

IUNet supports arbitrary-scale upsampling without losing quality. To better understand how this outstanding characteristic is obtained, we conduct an ablation study on the BLB dataset (Level 3). Note that we only evaluate the neural renderer. Table 3 shows: (i) Upsampling by IUNet without "clipping" will destroy the results because of out-of-range defocus values (B1 vs. B2 and B2 vs. B3); (ii) Using the low-resolution input bokeh image to compensate the clipping areas improves PSNR by 0.64 dB (B3 vs. B4); (iii): Replacing the signed defocus map with dilated defocus map further improves metrics slightly (B4 vs. B5). Besides, we show in the supplementary material that this operation will provide a more natural boundary transition when the focal plane targets background.

4.5. User Study

Since PSNR and SSIM cannot fully reflect the actual quality of the rendered bokeh images, we conduct a user study on the IPB dataset. For all methods, the blur parameter and the refocused disparity are manually adjusted to match iPhone 12 Portrait mode. The study involves 53 participants. From Table 4 and Fig. 13, one can see that our approach is most favored with a clear boundary of in-focus objects and natural bokeh effects for foreground blur. Note that iPhone 12 Portrait mode can only produce bokeh effects for objects behind the focal point.

Table 4. User study results. Given a scene, participants are required to select one option from "Good", "Normal", and "Bad" for each anonymous method.

Methods	iPhone 12	VDSLR	DeepLens	Ours
Good (%)	19.3	26.6	26.3	**55.0**
Normal (%)	29.3	47.7	45.0	38.5
Bad (%)	51.4	25.7	28.7	**6.5**

5. Discussion and Conclusion

Classical rendering methods are flexible but suffer from artifacts at depth discontinuities. Neural rendering methods are capable of handling boundary artifacts but lack controllability and have difficulty in generating stunning bokeh balls in out-of-focus areas. To exploit the advantages of two paradigms, we propose BokehMe, a general framework that combines a classical renderer and a neural renderer. Extensive experiments illustrate that BokehMe can produce photo-realistic and highly controllable bokeh effects from an all-in-focus image and a potentially imperfect disparity map, demonstrating strong complementarity of classical rendering and neural rendering.

For BokehMe, the bokeh style can be controlled by changing the kernel shape of the classical renderer. It works for most scenes, however, if highlights happen to lie at the boundary of the error map, the bokeh style inconsistency may be noticeable. In addition, given a 8-bit digital image where bright lights exist in the scene, the gamma correction is insufficient to create prominent bokeh balls in out-of-focus areas. An ideal way is to transform the LDR image to an HDR image [8], by inverse tone mapping [7, 15], which is beyond the scope of this paper. Although similar effects can be achieved by forcibly enhancing the RGB values of input images, there is still room for improvement. We leave this in our future work.

Acknowledgements. This work was funded by Adobe.

References

[1] Guillaume Abadie, Steve McAuley, Evegenii Golubev, Stephen Hill, and Sebastien Lagarde. Advances in real-time rendering in games. In *ACM SIGGRAPH 2018 Courses*, pages 1–1. 2018. 2

[2] Art gallery. https://cloud.blender.org/p/gallery. 6

[3] Jonathan T Barron, Andrew Adams, YiChang Shih, and Carlos Hernández. Fast bilateral-space stereo for synthetic defocus. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4466–4474, 2015. 2

[4] Marcelo Bertalmio, Pere Fort, and Daniel Sanchez-Crespo. Real-time, accurate depth of field using anisotropic diffusion and programmable graphics cards. In *Proc. International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT)*, pages 767–773, 2004. 2

[5] Blender. https://www.blender.org. 2, 6

[6] Benjamin Busam, Matthieu Hog, Steven McDonagh, and Gregory Slabaugh. Sterefo: Efficient image refocusing with stereo vision. In *Proc. IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 0–0, 2019. 1, 2, 6, 7

[7] Gaofeng Cao, Fei Zhou, Kanglin Liu, and Liu Bozhi. A brightness-adaptive kernel prediction network for inverse tone mapping. *Neurocomputing*, 464:1–14, 2021. 8

[8] Paul E Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *ACM SIGGRAPH 2008 classes*, pages 1–10. 2008. 8

[9] Saikat Dutta, Sourya Dipta Das, Nisarg A Shah, and Anil Kumar Tiwari. Stacked deep multi-scale hierarchical network for fast bokeh effect rendering from a single image. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2398–2407, 2021. 2

[10] Thomas Hach, Johannes Steurer, Arvind Amruth, and Artur Pappenheim. Cinematic bokeh rendering for real scenes. In *Proc. European Conference on Visual Media Production (CVMP)*, pages 1–10, 2015. 2

[11] Andrey Ignatov, Jagruti Patel, and Radu Timofte. Rendering natural camera bokeh effect with deep learning. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 418–419, 2020. 1, 2, 6

[12] Andrey Ignatov, Jagruti Patel, Radu Timofte, Bolun Zheng, Xin Ye, Li Huang, Xiang Tian, Saikat Dutta, Kuldeep Purohit, Praveen Kandula, et al. Aim 2019 challenge on bokeh effect synthesis: Methods and results. In *Proc. IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 3591–3598. IEEE, 2019. 2

[13] Andrey Ignatov, Radu Timofte, Ming Qian, Congyu Qiao, Jiamin Lin, Zhenyu Guo, Chenghua Li, Cong Leng, Jian Cheng, Juewen Peng, et al. Aim 2020 challenge on rendering realistic bokeh. In *Proc. European Conference on Computer Vision Workshops (ECCVW)*, pages 213–228. Springer, 2020. 2

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. International Conference on Learning Representations (ICLR)*, 2014. 6

[15] Yuma Kinoshita and Hitoshi Kiya. Itm-net: Deep inverse tone mapping using novel loss function considering tone mapping operator. *IEEE Access*, 7:73555–73563, 2019. 8

[16] Sungkil Lee, Elmar Eisemann, and Hans-Peter Seidel. Real-time lens blur effects and focus control. *ACM Transactions on Graphics (TOG)*, 29(4):1–7, 2010. 2

[17] Haiting Lin, Seon Joo Kim, Sabine Süsstrunk, and Michael S Brown. Revisiting radiometric calibration for color computer vision. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, pages 129–136. IEEE, 2011. 3

[18] Sifei Liu, Jinshan Pan, and Ming-Hsuan Yang. Learning recursive filters for low-level vision via a hybrid neural network. In *Proc. European Conference on Computer Vision (ECCV)*, pages 560–576. Springer, 2016. 2

[19] Hao Lu, Yutong Dai, Chunhua Shen, and Songcen Xu. Indices matter: Learning to index for deep image matting. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, pages 3266–3275, 2019. 5

[20] Xianrui Luo, Juewen Peng, Ke Xian, Zijin Wu, and Zhiguo Cao. Bokeh rendering from defocus estimation. In *Proc. European Conference on Computer Vision Workshops (ECCVW)*, pages 245–261. Springer, 2020. 1, 2

[21] Oliver Nalbach, Elena Arabadzhiyska, Dushyant Mehta, H-P Seidel, and Tobias Ritschel. Deep shading: Convolutional neural networks for screen space shading. *Computer Graphics Forum*, 36(4):65–78, 2017. 2

[22] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *Advances in Neural Information Processing Systems Workshops (NIPSW)*, 2017. 6

[23] Juewen Peng, Xianrui Luo, Ke Xian, and Zhiguo Cao. Interactive portrait bokeh rendering system. In *Proc. IEEE International Conference on Image Processing (ICIP)*, pages 2923–2927. IEEE, 2021. 2, 3, 6

[24] Photopea. https://www.photopea.com. 6

[25] Ming Qian, Congyu Qiao, Jiamin Lin, Zhenyu Guo, Chenghua Li, Cong Leng, and Jian Cheng. Bggan: Bokeh-glass generative adversarial network for rendering realistic bokeh. In *Proc. European Conference on Computer Vision (ECCV)*, pages 229–244. Springer, 2020. 1, 2

[26] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, pages 12179–12188, 2021. 1, 6, 7

[27] Xiaoyong Shen, Aaron Hertzmann, Jiaya Jia, Sylvain Paris, Brian Price, Eli Shechtman, and Ian Sachs. Automatic portrait segmentation for image stylization. *Computer Graphics Forum*, 35(2):93–102, 2016. 2

[28] Xiaoyong Shen, Xin Tao, Hongyun Gao, Chao Zhou, and Jiaya Jia. Deep automatic portrait matting. In *Proc. European Conference on Computer Vision (ECCV)*, pages 92–107. Springer, 2016. 2

[29] Cyril Soler, Kartic Subr, Frédo Durand, Nicolas Holzschuch, and François Sillion. Fourier depth of field. *ACM Transactions on Graphics (TOG)*, 28(2):1–12, 2009. 2

[30] Unity engine. http://unity3d.com. 2, 6

[31] Neal Wadhwa, Rahul Garg, David E Jacobs, Bryan E Feldman, Nori Kanazawa, Robert Carroll, Yair Movshovitz-Attias, Jonathan T Barron, Yael Pritch, and Marc Levoy. Synthetic depth-of-field with a single-camera mobile phone. *ACM Transactions on Graphics (TOG)*, 37(4):1–13, 2018. 1, 2, 3

[32] Lijun Wang, Xiaohui Shen, Jianming Zhang, Oliver Wang, Zhe Lin, Chih-Yao Hsieh, Sarah Kong, and Huchuan Lu. Deeplens: Shallow depth of field from a single image. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018. 1, 2, 6, 7, 8

[33] Jiaze Wu, Changwen Zheng, Xiaohui Hu, and Fanjiang Xu. Rendering realistic spectral bokeh due to lens stops and aberrations. *The Visual Computer*, 29(1):41–52, 2013. 2

[34] Ke Xian, Juewen Peng, Chao Zhang, Hao Lu, and Zhiguo Cao. Ranking-based salient object detection and depth prediction for shallow depth-of-field. *Sensors*, 21(5):1815, 2021. 2, 6

[35] Lei Xiao, Anton Kaplanyan, Alexander Fix, Matthew Chapman, and Douglas Lanman. Deepfocus: Learned image synthesis for computational displays. *ACM Transactions on Graphics (TOG)*, 37(6):1–13, 2018. 2, 4, 6, 7

[36] Ning Xu, Brian Price, Scott Cohen, and Thomas Huang. Deep image matting. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2970–2979, 2017. 5

[37] Xiangyu Xu, Deqing Sun, Sifei Liu, Wenqi Ren, Yu-Jin Zhang, Ming-Hsuan Yang, and Jian Sun. Rendering portraitures from monocular camera and beyond. In *Proc. European Conference on Computer Vision (ECCV)*, pages 35–50, 2018. 2, 7

[38] Yang Yang, Haiting Lin, Zhan Yu, Sylvain Paris, and Jingyi Yu. Virtual dslr: High quality dynamic depth-of-field synthesis on mobile platforms. *Electronic Imaging*, 2016(18):1–9, 2016. 2, 3, 6, 7, 8

[39] Xuan Yu, Rui Wang, and Jingyi Yu. Real-time depth of field rendering via dynamic light field generation and filtering. *Computer Graphics Forum*, 29(7):2099–2107, 2010. 2

[40] Xuaner Zhang, Kevin Matzen, Vivien Nguyen, Dillon Yao, You Zhang, and Ren Ng. Synthetic defocus and look-ahead autofocus for casual videography. *ACM Transactions on Graphics (TOG)*, 38:1 – 16, 2019. 1, 2, 6, 7