Are Realistic Training Data Necessary for Depth-from-Defocus Networks?

Zhuofeng Wu
Tokyo Institute of Technology
Tokyo, Japan
zwu@ok.sc.e.titech.ac.jp

Yusuke Monno
Tokyo Institute of Technology
Tokyo, Japan
ymonno@ok.sc.e.titech.ac.jp

Masatoshi Okutomi
Tokyo Institute of Technology
Tokyo, Japan
mxo@ctrl.titech.ac.jp

IECON 2022 – 48th Annual Conference of the IEEE Industrial Electronics Society | 978-1-6654-8025-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/IECON49645.2022.9968692

***Abstract*—Image-based depth estimation is one of the important tasks in computer vision. Depth-from-defocus (DfD) methods estimate the scene depth from a single or multiple defocused images by exploiting depth-dependent defocus blur cues. Because of the difficulty in obtaining a real-world dataset with ground-truth scene depth, most deep-learning-based DfD methods rely on a synthetic training dataset, where more realistic scene rendering is considered desirable for more accurate depth estimation. In this paper, we consider if realistic 3D objects are really necessary for training DfD networks. To investigate this, we design a very simple and fast synthetic training data generation method for DfD using only two front-parallel texture planes in one scene and compare it with a widely-applied path-tracing method using a common 3D object dataset. Through real-world experiments, we show that the 2-plane method provides comparable and even slightly better performance than the path-tracing method and can be considered as an alternative method for simple and practical DfD network training.**

***Index Terms*—Depth estimation, depth from defocus, depth from focal stack, deep learning**

I. Introduction

Research on image-based depth estimation is gaining more attention as it applies to a lot of computer vision applications. Depth from defocus (DfD) is one of the depth sensing approaches, which estimates the depth map by using one or multiple defocused images taken by the same camera based on the physical property that the amount of defocus blur at each pixel is related to the scene depth. Compared with other active or passive depth sensing approaches such as time-of-flight, structured-light, and stereo, DfD has the advantage that it only requires a single camera and is not affected by occlusions.

Classic DfD methods compute the depth as an optimization problem [1], [2], [3]. Recently, deep-learning-based DfD methods have achieved better performance by training a network model to learn the relationship between the scene depth and the amount of blur from a large amount of training data [4], [5], [6], [7]. For the acquisition of a training dataset, some studies obtained real-world data [7], [8]. However, obtaining real-world data takes a lot of time and labor. It is also hard to obtain accurate ground-truth scene depth perfectly aligned to input defocused images. Thus, many deep-learning-based DfD methods are based on synthetic training data generation.

There are several previous works on synthetic defocused image generation for DfD network training. The pixel-wise blurring method in [5] converts each pixel's ground-truth depth value to the standard deviation of Gaussian point spread function (PSF) and applies the corresponding Gaussian PSF to an all-in-focus image in a per-pixel manner. However, this method cannot synthesize accurate defocus blur around depth boundaries because blurring the all-in-focus image does not consider the textures of occluded areas, which actually affect the appearance of real defocus blur. This method also takes a lot of time to apply blurring processes on each pixel. To reduce the computational cost, the studies [9], [10] adopt a layer-driven blur model, which decomposes an all-in-focus image into discrete depth layers according to per-pixel ground-truth depth values and applies corresponding Gaussian PSFs to each layer. The blurred layers are finally composed to form the defocused image. Although the computational speed is accelerated by per-layer blurring, this model still does not consider the occluded areas and thus generates inaccurate defocus blur at depth boundaries. The study [4] applies a render engine, Cycles, in Blende [11], where a full 3D scene is rendered to obtain more realistic defocused images by using a path-tracing method, at the cost of longer computational time. Generally, for DfD, a path-tracing method using realistic 3D object models is expected to bridge the gaps between synthetic and real-world data.

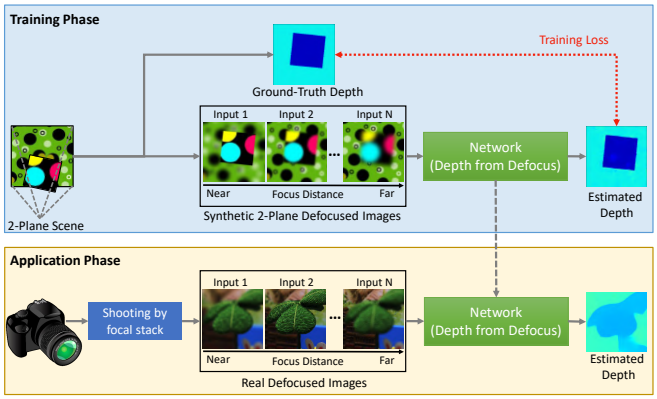


Fig. 1. The overview of the 2-plane method. In the training phase, a DfD network is trained using synthetic defocused images generated assuming 2-plane scenes. In the application phase, the trained network is applied to real defocused images captured by using a focal stack function of a camera.

In this study, we consider if we really need realistic 3D objects for training DfD networks. To investigate this, we first present a very simple and fast training data generation method based on only two front-parallel texture planes, as illustrated in the top of Fig. 1. Using the input defocused

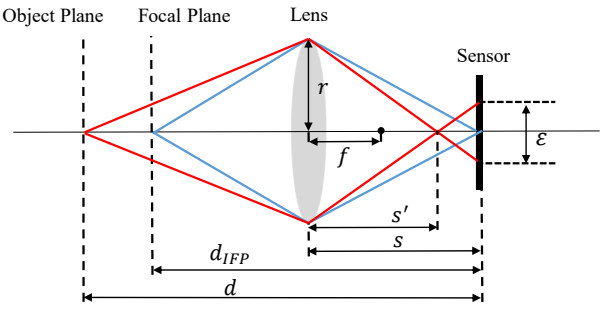


Fig. 2. Thin lens model.

image sets generated from 2-plane training scenes, we train a DfD network, which is applied to real defocused images at the application phase. We experimentally compare this 2-plane method with a path-tracing method using a common 3D object dataset, as in [4], and show that the 2-plane method can provide competitive performance for real-world scenes, which suggests that the 2-plane method can be used as a simple and practical method for training DfD networks.

II. 2-Plane Method

A. Thin Lens Model

We apply a thin lens model [12] to calculate the relationship between the depth and the blur amount. According to the illustration in Fig. 2, we describe the lens radius as r, the focal length as f, the distance from the lens to the image as s', the distance from the image sensor to the lens as s, the focus distance from the sensor as d_{IFP}, and the scene or object depth from the sensor as d. In the past literature, the distances d_{IFP} and d are often defined as the distances from the optical center (the lens center). Instead, in our study, the distances are measured from the image sensor because an actual camera lens is made by combining multiple lenses and it is impossible to accurately measure the position of the optical center, while the distances from the image sensor can be measured using a reference mark on the camera identifying the position of the image sensor.

According to the above descriptions and Fig. 2, the thin lens model derives the following two equations.

$$\frac{1}{s}+\frac{1}{d_{IFP}-s}=\frac{1}{f}. \tag{1}$$

$$\frac{1}{s'}+\frac{1}{d-s}=\frac{1}{f}. \tag{2}$$

Furthermore, following [13] and the calculation of similar triangles, the blur circle diameter ε follows the relationship of

$$\frac{\varepsilon}{2r}=\frac{s-s'}{s'}. \tag{3}$$

This can be rewritten using the aperture size $F_{\#}=f/2r$ as

$$\varepsilon=\frac{f}{F_{\#}}\frac{s-s'}{s'}, \tag{4}$$

where the focal length f and the aperture size $F_{\#}$ can be obtained according to camera settings or camera calibration. Although the distances s and s' are not measurable in practice by the reason mentioned before, we apply Eq. (1) and (2) to eliminate them as

$$\begin{aligned}\varepsilon(d)=&\frac{d_{IFP}-\sqrt{{d_{IFP}}^2-4fd_{IFP}}}{2F_{\#}}\\&-\frac{2fd}{F_{\#}(2d-d_{IFP}+\sqrt{{d_{IFP}}^2-4fd_{IFP}})},\end{aligned} \tag{5}$$

where the blur circle ε is expressed as a function of the depth d under the pre-calibrated known focus distance d_{IFP}. We describe the calibration process for d_{IFP} in Sec. III.

B. 2-Plane Defocused Image Generation

Figure 3 illustrates the overall flow of defocused image generation by the 2-plane method. Firstly, we prepare an all-in-focus synthetic texture dataset and randomly select two textures for the back-plane texture I^B and the front-plane texture I^F. To generate a two-plane scene and the corresponding depth map, we create a random front mask M to identify the front-plane region. The size and the angle of the front plane, which is a squared shape, are randomly determined. Then, according to the random front mask M, the depth map can be obtained by assigning two random depth values (d_B, d_F), where $d_B > d_F$, for the back plane and the front plane, respectively.

To generate a defocused image under a known d_{IFP}, we apply multi-plane image representation, which has been adopted for a view synthesis problem [14], [15], using only two planes. The blurring of each texture is performed by applying a Gaussian PSF kernel [13], which is expressed as

$$G(x,y,\sigma)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}, \tag{6}$$

where (x, y) represents the coordinate in the PSF kernel and σ is the standard deviation of the Gaussian PSF. According to [13], the standard deviation σ is proportionally related to the blur diameter ε of Eq. (5) and expressed by a function of the scene depth d as

$$\sigma(d)=k\frac{1}{p}\varepsilon(d), \tag{7}$$

where p is the pixel size which converts the diameter of the blur circle ε to the pixel unit and k is a constant parameter to determine the proportionality characteristic of the used camera, for which we perform calibration explained in Sec. III.

Given the ground-truth two-plane depth values (d_B, d_F) and the calibrated focus distance d_{IFP} for n-th input image, the Gaussian kernel for the back plane G_n^B and that for the front plane G_n^F can be calculated using Eq. (5), (6), and (7). Using these kernels, the n-th defocused image D_n is generated by an alpha blending as

$$\begin{aligned}D_n&=\alpha\odot(G_n^F*I^F)+(1-\alpha)\odot(G_n^B*I^B),\\&=(G_n^F*M)\odot(G_n^F*I^F)+(1-G_n^F*M)\odot(G_n^B*I^B),\end{aligned} \tag{8}$$

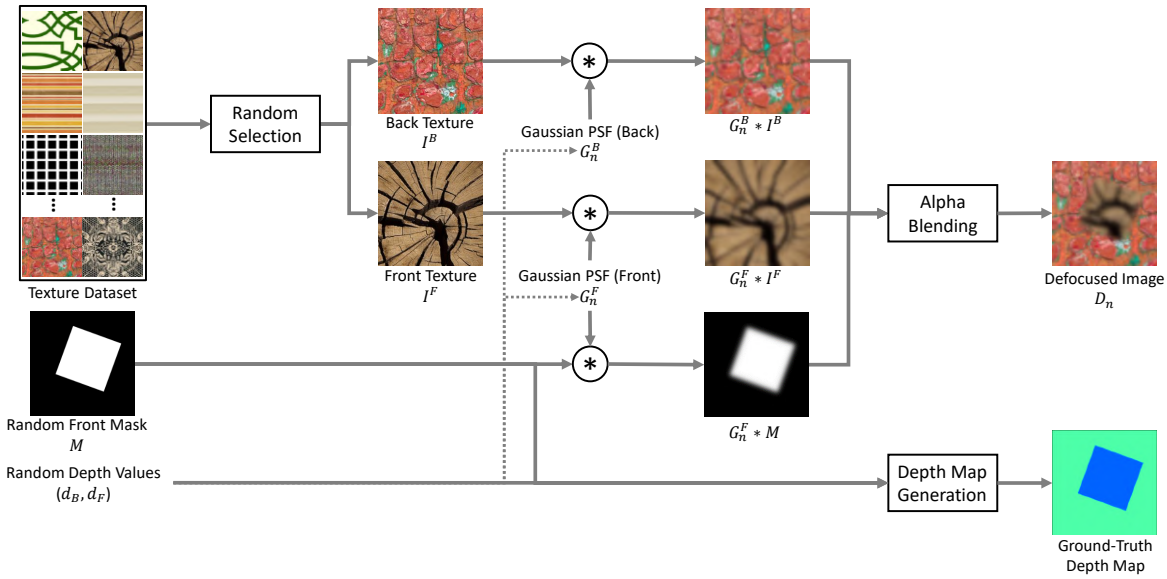


Fig. 3. The overall flow of the 2-plane defocused image generation. The back-plane texture I^B and the front-plane texture I^F are randomly selected from a texutre dataset. A random front mask M is generated and the corresponding ground-truth 2-plane depth map is created by assigning two random depth values (d_B, d_F). After applying the back and the front Gaussian PSFs, which are calculated under a known d_{IFP} for n-th input image, to I^B, I^F, and M, the defocused image is generated by alpha blending. This process is performed for each of N input images.

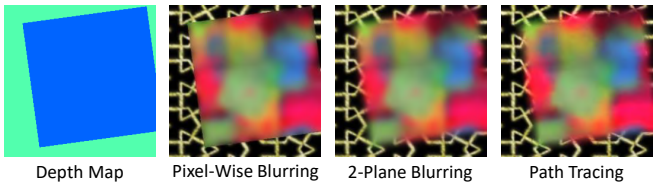
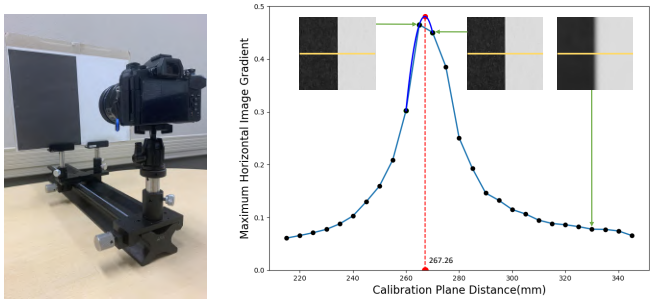


Fig. 4. The examples of 2-plane defocused images generated with different blurring methods. Compared with the pixel-wise all-in-focus image blurring of [5], the 2-plane blurring can produce more realistic depth boundary blurs, which are close to the blurs of more complex path-tracing of Blender.



(a) The calibration setup (b) The calibration of focus distance

Fig. 5. The process of focus distance calibration. (a) is the setup for the calibration, where we put a calibration plane with one black-white edge in front of a fixed camera and captured the images of the plane at different distances from the image sensor. As in (b), we calculated the maximum horizontal gradient value of each distance image and obtained the focus distance by the vertex of a parabola fitted around the largest gradient value.

where α is determined using the blurred front mask as $\alpha = G_n^F * M$, $\odot$ represents the pixel-wise product, and $*$ represents the convolution operation.

Figure 4 shows the examples of 2-plane defocused images generated with different blurring methods. Compared with the pixel-wise all-in-focus image blurring of [5], the 2-plane blurring can produce more realistic depth boundary blurs, which are close to the blurs of more complex path-tracing of Blender. This is because that the 2-plane method considers the occluded back-plane texture to generate the blurs, while the pixel-wise blurring ignores that texture.

III. Calibration Procedure

Although the 2-plane method is applicable to any digital camera that has a focus stack function, to apply it to a real camera, we need the calibration of the focus distance d_{IFP} for each of N input images and the constant parameter k, since those parameters are not directly obtainable and required to synthesize the input defocused image set as explained in the previous section.

In our experiments, we used Olympus OM-D E-M5 Mark III camera with the pixel size $p = 3.3\mu m$. We set the aperture size $F_\#$ to 3.2. The focal length f can be read from the camera lens setting or obtained by a standard camera calibration [16]. We used the focal length of $f = 12.22mm$ calibrated using the MATLAB camera calibration toolbox. To calibrate the focus distance of each input image, we used a calibration plane with one black-white edge, as shown in Fig. 5(a). We fixed the camera on a camera slider with a scale and captured the calibration plane by changing the distance from the plane to the image sensor reference mark on the camera by 5mm intervals from 210mm to 425mm. Then, as shown in Fig. 5(b), for each distance image, we calculated the maximum horizontal image gradient value of the black-white edge. Then, a parabola fitting is performed using the largest gradient value and its two neighboring values. The focus distance is then estimated by the distance corresponding to the vertex of the parabola, which is shown as the red point in Fig. 5(b).

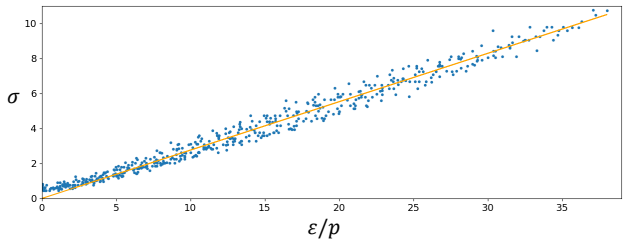
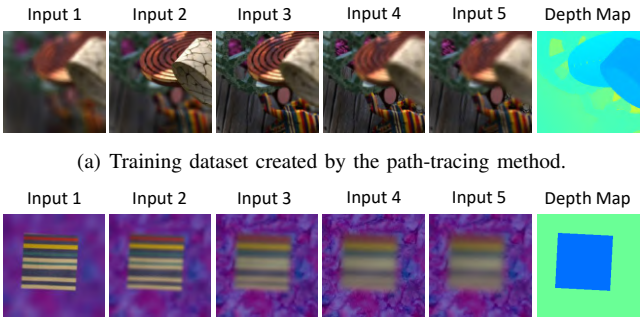


Fig. 6. The calibration of the constant parameter k. The horizontal axis presents the blur circle $\varepsilon(d)/p$ in pixel unit and the vertical axis stands for the real image's $\sigma(d)$. Each blue dot corresponds to the ε/p and σ pair of each real image taken at different distances d. The slope of the orange line represents the calibrated constant parameter k.



(a) Training dataset created by the path-tracing method.

(b) Training dataset created by the 2-plane method.

Fig. 7. The samples of synthetic training datasets.

As for the calibration of the constant parameter k in Eq. (7), we used the same calibration images. For each distance, we calculated the value of $\varepsilon(d)$ according to Eq. (5) using the camera parameters, the corresponding depth, and the calibrated focus distance. We then converted it to ε/p by using pixel size p. Since we cannot directly obtain $\sigma(d)$ from the real captured image, we first blurred a synthetic black-white edge image with discrete σ values and then searched the closest blurred image that has the most similar intensity change in the horizontal direction to the real captured image. Then, we regarded the corresponding σ value as the σ value of the real captured image. Finally, we estimated the parameter $k = 0.2765$ from all the σ and the ε/p pairs obtained from all the calibration images by applying the least squares method as shown in Fig. 6.

IV. Experimental Results

A. Training Datasets

We compared the 2-plane method with a more realistic path-tracing method. The training datasets are generated by each method assuming a real camera setup with Olympus OM-D E-M5 Mark III, where we used the focal length of 12.22mm, the aperture size of 3.2, and five input images with the calibrated focus distances of [213.75mm, 267.26mm, 321.75mm, 379.57mm, 422.45mm]. The target depth range was set as [215mm, 420mm].

1) Path-Tracing Dataset: To generate 3D scenes with more realistic and complex 3D objects, we used the same generation code as the DefocusNet dataset [4]. The code generates a 3D scene by randomly placing several 3D objects of the Thingi10K object dataset [17]. As object textures, we used 300 texture images from the DTD texture dataset [18] and randomly assigned one texture to one object. The input defocused image set was then rendered by utilizing Cycles, which is Blender's path-tracing-based rendering engine [19]. The samples of generated data are shown in Fig. 7(a), where each input image set contains five defocused images of 256×256. We generated 1,600 scenes for training. For the generation time, it takes around 18 seconds to generate the five defocused images of one scene with Nvidia GeForce GTX 1080 Ti GPU.

2) 2-Plane Dataset: We generated the 2-plane dataset by the method explained in Section II-B. We used the same 300 texture images as the path-tracing dataset and generated 1,600 scenes for training. The sample of generated data can be seen in Fig. 7(b). For the time required for the data generation, it takes only about 0.2 seconds for one scene using Intel(R) Core(TM) i7-7820X CPU, which is 90 times faster than the path-tracing method.

B. DfD Networks

We adopted two existing DfD networks: AiFDepthNet [6] and DefocusNet [4]. The network models were trained from the scratch using the Adam optimizer [20] with the parameters of ($\beta 1 = 0.9, \beta 2 = 0.999$), the learning rate of 10^{-4}, and the training patch size of 256×256.

For DefocusNet, we trained the network with the batch size of 4 and the epoch of 2,000, which are the original settings in the provided training code. For AiFDepthNet, we empirically used the batch size of 16 and the epoch of 1,000 because only the network architecture is available and the training code is not provided.

C. Real-World Testing Data

We evaluated the trained network models using three real-world testing scenes. The real-scene defocused images were captured by an Olympus OM-D E-M5 Mark III camera with the camera parameter settings as mentioned in Section IV-A. The image resolution is 2358×1760 and enlarged parts of the captured images are shown in Fig. 8.

To obtain the ground-truth depth, we used an Intel RealSense LiDAR L515 sensor with 1024×768 depth map resolution, which was placed next to the camera. The intrinsic and extrinsic parameters between the camera and the LiDAR were obtained by stereo camera calibration [16].

D. Evaluation Metric

Figure 9 shows our evaluation procedure. We compared the ground-truth LiDAR depth and the estimated DfD depth by converting them to the point clouds at the same 3D coordinates using pre-calibrated extrinsic and intrinsic parameters. This is because the LiDAR depth map and the estimated DfD depth map are obtained for different viewpoints and resolutions, and thus warping the LiDAR depth to the camera viewpoint causes an occlusion problem. Specifically, since the 3D points

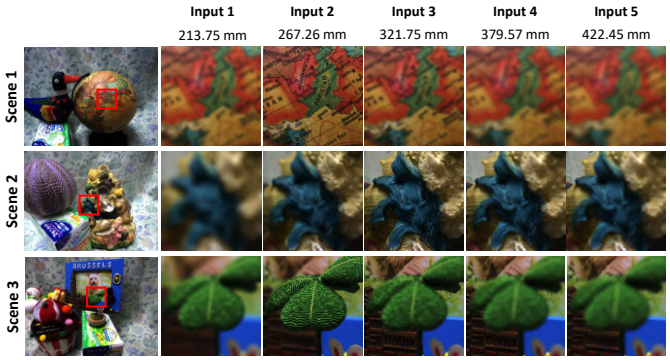


Fig. 8. The real-world data for evaluation. We obtained the real-world data by a real SLR camera. The corresponding calibrated focus distance for each image is shown on the top of each column.

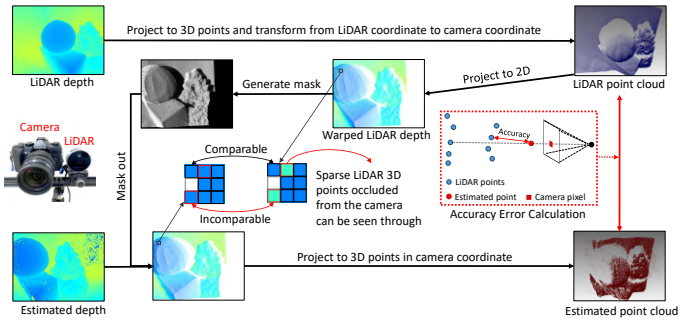


Fig. 9. Our evaluation procedure. Because the warped LiDAR depth is sparse and some LiDAR 3D points may be occluded from the camera, the LiDAR depth and the estimated depth are not directly comparable in the depth map domain. Thus, we compare them as the 3D point clouds projected to the same 3D coordinate and use Accuracy error, which is defined as the distance from each estimated 3D point to its closest ground-truth LiDAR 3D point.

converted from the LiDAR depth are sparse, the occluded 3D points from the camera viewpoint can be projected to the camera image plane, leading to inappropriate evaluation of depth maps. To address the sparseness of the LiDAR 3D points, we reproject the LiDAR 3D points to the camera image to just mask out the camera pixels that warped LiDAR 3D depth does not exist. Then, to address the occlusion problem, we evaluate the depth results as the 3D point clouds by using the Accuracy error [21], where the Accuracy is defined as the distance from each estimated 3D point to its closest ground-truth LiDAR 3D point.

E. Comparison on Each Network

We trained AiFDepthNet and DefocusNet using 1,600 training scenes generated by the 2-plane and the path-tracing method, respectively. Then, the depth maps for testing real-world scenes were estimated by each trained model. The visualization results are shown in Fig. 10. The quantitative results in Table I indicates that the 2-plane method can provide similar and even slightly better performance compared to the path-tracing method. It also shows that the performance of DefocusNet is slightly better on average than AifDepthNet for both the path-tracing and the 2-plane methods.

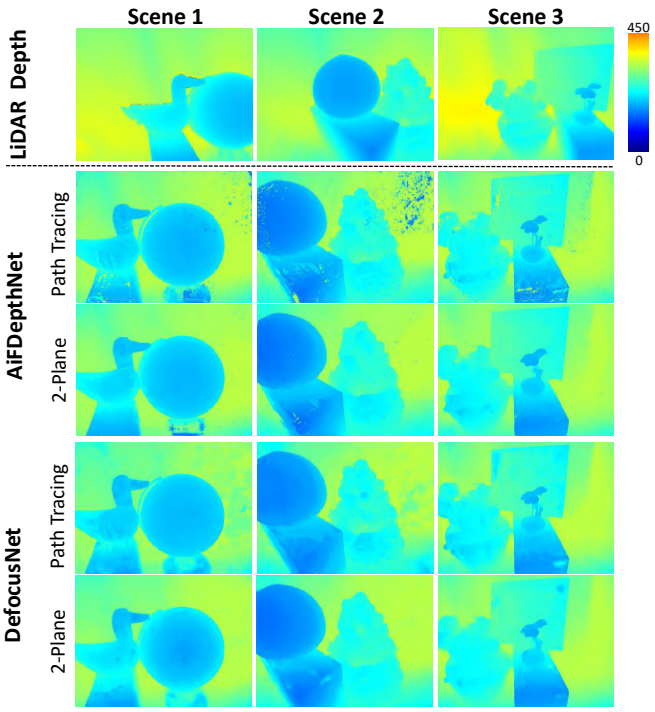


Fig. 10. The visualization results of the 2-plane method and the path-tracing method with AiFDepthNet and DefocusNet.

TABLE I
THE ACCURACY ERROR (THE LOWER IS THE BETTER) OF THE 2-PLANE AND THE PATH-TRACING METHODS WITH AIFDEPTHNET AND DEFOCUSNET. ALL NUMERICAL VALUES ARE IN MILLIMETERS.

Network	Method	Scene 1	Scene 2	Scene 3	Mean
AiFDepthNet	Path Tracing	6.65	6.65	5.80	6.37
	2-Plane	**6.56**	**5.63**	**5.39**	**5.86**
DefocusNet	Path Tracing	6.31	6.21	5.89	6.03
	2-Plane	**5.88**	**5.61**	**4.95**	**5.48**

F. Effect of Number of Training Data

To investigate the effect of the number of training data, we evaluated the 2-plane method and the path-tracing method trained with AiFDepthNet or DefocusNet using 100, 200, 400, 800, and 1,600 training scenes, respectively. From the results shown in Fig. 11, we can see that the 2-plane method provides competitive performance compared with the path-tracing method for both networks. With regard to the performance of the two networks, AiFDepthNet converges faster using the 2-plane method than the path-tracing method. As for DefocusNet, both the path-tracing and the 2-plane methods show relatively good performance with only 100 training data.

G. Comparison of 2-Plane and 1-Plane Dataset

To verify that at least two texture planes are needed for training DfD networks, as the 2-plane scene contains the scenarios of depth discontinuities, we compared the 2-plane method with the 1-plane method that only uses one texture plane in one scene. For each method, DefocusNet was trained with 1,600 scenes of training data.

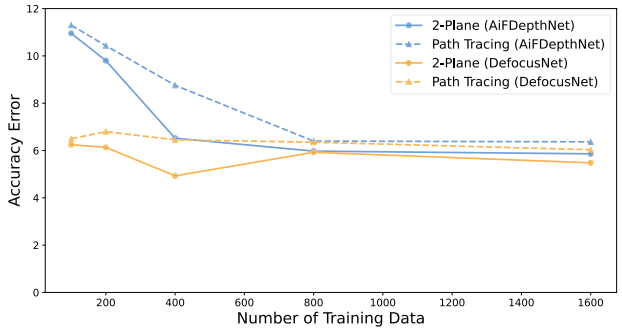


Fig. 11. The accuracy error (the lower is the better.) of the 2-plane method and the path-tracing method with different numbers of training data. Each method was trained using 100, 200, 400, 800, and 1,600 scenes of training data and the same real-world data were used for testing.

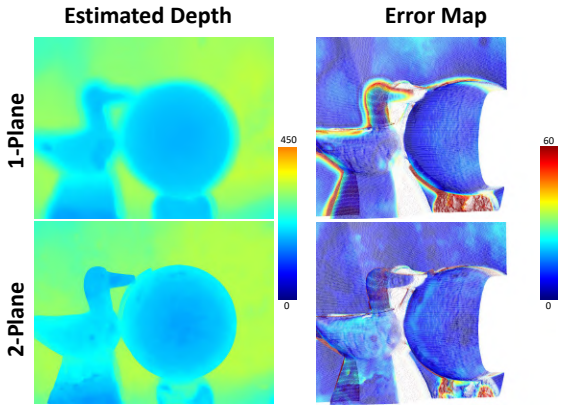


Fig. 12. The visual comparison of the 1-plane and the 2-plane methods with DefocusNet. The 1-plane method has difficulty in accurately estimating the depth of object boundaries.

TABLE II
THE ACCURACY ERROR OF 1-PLANE AND 2-PLANE METHODS WITH DEFOCUSNET. THE LOWER IS THE BETTER.

Network	Method	Scene 1	Scene 2	Scene 3	Mean
DefocusNet	1-Plane	7.15	5.78	5.68	6.20
	2-Plane	**5.88**	**5.61**	**4.95**	**5.48**

The visual results in Fig. 12 demonstrate that the 2-plane method can obtain the depth map with more sharp and accurate object boundaries compared with the 1-plane method. The numerical evaluation in Table II also verifies that the 2-plane method outperforms the 1-plane method in all the scenes.

V. CONCLUSION

In this paper, we have presented a very simple and fast 2-plane training data generation method for DfD and have investigated whether realistic 3D object models are necessary for training DfD networks by comparing the performance of the 2-plane method and a path-tracing method using a common 3D object dataset. Experimental results using real-world data have demonstrated that the 2-plane method can yield similar and even slightly outperformed results compared with the path-tracing method. This suggests that the 2-plane method can be applied as a simple and practical method for DfD network training. Our future work includes the evaluation of the 2-plane method on more large-scale real-world scenes.

REFERENCES

[1] Shinsaku Hiura and Takashi Matsuyama, "Depth measurement by the multi-focus camera," in *IEEE Conference on Computer Vision and Pattern Recognition*, 1998, pp. 953–959.

[2] Supasorn Suwajanakorn, Carlos Hernandez, and Steven M Seitz, "Depth from focus with your mobile phone," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3497–3506.

[3] Murali Subbarao and Gopal Surya, "Depth from defocus: A spatial domain approach," *International Journal of Computer Vision*, vol. 13, no. 3, pp. 271–294, 1994.

[4] Maxim Maximov, Kevin Galim, and Laura Leal-Taixé, "Focus on defocus: Bridging the synthetic to real domain gap for depth estimation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1071–1080.

[5] Gwangmo Song and Kyoung Mu Lee, "Depth estimation network for dual defocused images with different depth-of-field," in *IEEE International Conference on Image Processing*, 2018, pp. 1563–1567.

[6] Ning-Hsu Wang, Ren Wang, Yu-Lun Liu, Yu-Hao Huang, Yu-Lin Chang, Chia-Ping Chen, and Kevin Jou, "Bridging unsupervised and supervised depth from focus via all-in-focus supervision," in *IEEE/CVF International Conference on Computer Vision*, 2021, pp. 12621–12631.

[7] Gwangmo Song, Yumee Kim, Kukjin Chun, and Kyoung Mu Lee, "Multi image depth from defocus network with boundary cue for dual aperture camera," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2020, pp. 2293–2297.

[8] Marcela Carvalho, Bertrand Le Saux, Pauline Trouvé-Peloux, Andrés Almansa, and Frédéric Champagnat, "Deep depth from defocus: How can defocus blur improve 3D estimation using dense neural networks?," in *European Conference on Computer Vision Workshops*, 2018.

[9] Yawen Lu, Garrett Milliron, John Slagter, and Guoyu Lu, "Self-supervised single-image depth estimation from focus and defocus clues," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 6281–6288, 2021.

[10] Junyong Lee, Sungkil Lee, Sunghyun Cho, and Seungyong Lee, "Deep defocus map estimation using domain adaptation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12222–12230.

[11] Blender Foundation, "Blender," https://www.blender.org/download/.

[12] Eugene Hecht, *Optics*, Pearson Education, 2016.

[13] Muralidhara Subbarao and Natarajan Gurumoorthy, "Depth recovery from blurred edges," in *IEEE Conference on Computer Vision and Pattern Recognition*, 1988, pp. 498–499.

[14] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely, "Stereo magnification: Learning view synthesis using multi-plane images," *ACM Transactions on Graphics*, vol. 37, no. 4, 2018.

[15] Richard Tucker and Noah Snavely, "Single-view view synthesis with multiplane images," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 551–560.

[16] Zhengyou Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[17] Qingnan Zhou and Alec Jacobson, "Thingi10k: A dataset of 10,000 3D-printing models," *arXiv preprint 1605.04797*, 2016.

[18] Mircea Cimpoi, Subhransu Maji, Iasonas Kokkinos, Sammy Mohamed, and Andrea Vedaldi, "Describing textures in the wild," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3606–3613.

[19] Ganga Rama Koteswara Rao, P Vidya Sgar, Thulasi Bikku, Chitturi Prasad, and Naresh Cherukuri, "Comparing 3D rendering engines in blender," in *International Conference on Smart Electronics and Communication*, 2021, pp. 489–495.

[20] Diederik P. Kingma and Jimmy Lei Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[21] Andreas Ley, Ronny Hänsch, and Olaf Hellwich, "Syb3r: A realistic synthetic benchmark for 3D reconstruction from images," in *European Conference on Computer Vision*, 2016, pp. 236–251.