Deblur-NeRF: Neural Radiance Fields from Blurry Images

Li Ma\(^1\)*  Xiaoyu Li\(^2\)  Jing Liao\(^3\)  Qi Zhang\(^2\)  Xuan Wang\(^2\)
Jue Wang\(^2\)  Pedro V. Sander\(^1\)

\(^1\)The Hong Kong University of Science and Technology
\(^2\)Tencent AI Lab  \(^3\)City University of Hong Kong

Abstract

Neural Radiance Field (NeRF) has gained considerable attention recently for 3D scene reconstruction and novel view synthesis due to its remarkable synthesis quality. However, image blurriness caused by defocus or motion, which often occurs when capturing scenes in the wild, significantly degrades its reconstruction quality. To address this problem, we propose Deblur-NeRF, the first method that can recover a sharp NeRF from blurry input. We adopt an analysis-by-synthesis approach that reconstructs blurry views by simulating the blurring process, thus making NeRF robust to blurry inputs. The core of this simulation is a novel Deformable Sparse Kernel (DSK) module that models spatially-varying blur kernels by deforming a canonical sparse kernel at each spatial location. The ray origin of each kernel point is jointly optimized, inspired by the physical blurring process. This module is parameterized as an MLP that has the ability to be generalized to various blur types. Jointly optimizing the NeRF and the DSK module allows us to restore a sharp NeRF. We demonstrate that our method can be used on both camera motion blur and defocus blur: the two most common types of blur in real scenes. Evaluation results on both synthetic and real-world data show that our method outperforms several baselines. The synthetic and real datasets along with the source code is publicly available at https://limacv.github.io/deblurnerf/.

1. Introduction

Tremendous progress has been witnessed in the past few years in novel view synthesis, where an intermediate 3D representation is reconstructed from sparse input views to interpolate or extrapolate arbitrary novel views. Recently, NeRF [22] emerged as an effective scene representation that achieves photorealistic rendering results. It models a static scene as a continuous volumetric function that maps 3D location and 2D view direction to color and density. This function is parameterized as a multilayer perceptron (MLP), and its output can be rendered by volume rendering techniques in a differentiable manner.

To reconstruct a NeRF, several images from different views are needed. While the original approach for train-
ing NeRF works well when these images are well captured and calibrated, it would produce obvious artifacts when blur occurs. For example, when using a long exposure setting to capture a low-light scene, the images are more sensitive to camera shake, resulting in camera motion blur. Furthermore, defocus blur is inevitable when capturing scenes with large depth variation using a large aperture. These blurs will significantly decrease the quality of the reconstructed NeRF, resulting in artifacts in the rendered novel views.

Many works have recently been proposed to tackle abnormal input while training NeRF. NeRF-W [20] focuses on images with illumination change and moving objects. Mip-NeRF [1] improves the NeRF when the input spans different scales. Distortion in input is considered and calibrated simultaneously in SCNeRF [11]. To the best of our knowledge, none has considered addressing the problem of training NeRF from blurry input images. One solution is to first deblur the input in image space and then train the NeRF with deblurred images, which we refer to as the image-space baseline. This baseline improves the novel view synthesis quality of NeRF to some extent by utilizing recent single image or video deblurring methods. However, the single-image deblur methods fail to aggregate information from neighbor views and can not guarantee the multi-view consistent result. Video-based methods manage to take multi-frame into consideration, usually relying on image space operations such as optical flows and feature correlation volumes. However, these methods fail to exploit the 3D geometry of the scene, leading to inaccurate correspondences across views, especially when they have a large baseline. On the contrary, our method deblurs by aggregating information from all observations with full awareness of the 3D scene.

In this paper, we propose Deblur-NeRF, an effective framework that explicitly models the blurring process in the network, and is capable of restoring a sharp NeRF from blurry input. We model the blurring process by convolving a clean image using a blur kernel similar to blind deconvolution methods [2]. A novel deformable sparse kernel (DSK) module is proposed to model the blur kernel inspired by the following observations. First, convolving with dense kernels is infeasible for scene representations such as NeRF due to the dramatic increase in computation and memory usage during rendering. To address this, DSK uses sparse rays to approximate the dense kernel. Second, we show that the actual blurring process involves combining rays from different origins, which motivates us to jointly optimize the ray origins. Finally, to model spatially-varying blur kernels, we deform a canonical sparse kernel at each 2D spatial location. The deformation is parameterized as an MLP that can be generalized to different types of blur. During training, we jointly optimize the DSK and a sharp NeRF with only blurry input as supervision, while in the inference stage, clear novel views can be rendered by removing the DSK.

We conduct extensive experiments on both synthetic and real datasets with two types of blur: camera motion blur and defocus blur. Results show that the proposed method outperforms the original NeRF and image-space baselines (i.e., combining NeRF with the state-of-the-art image or video deblurring methods), for these two blur types, as shown in Fig. 1 and the experiments section. Our contributions can be summarized as follows:

- We propose the first framework that can reconstruct a sharp NeRF from blurry input.
- We propose a deformable sparse kernel module that enables us to effectively model the blurring process and is generalizable for different types of blur.
- We analyze the physical blurring process and extend the 2D kernel to 3D space by considering the translation of the ray origin for each kernel point.

2. Related Work

Neural radiance field. Our work extends the NeRF [22], a coordinate-based implicit 3D scene representation, which has gained popularity over the past few years due to its state-of-the-art novel view synthesis results. The success of NeRF has inspired many follow-up works that extend the NeRF [7, 8, 18, 25, 30–32, 34, 38]. Several works have explored to train the NeRF with non-ideal input. For example, BRAF [19], NeRF—[47] and GNeRF [21] try to train the NeRF without camera poses. SCNeRF [11] focuses on jointly calibrating a more complex non-linear camera model. To address the NeRF training under uncontrolled, in-the-wild photographs, NeRF-W [20] introduces several extensions to NeRF that successfully model the inconsistent appearance variations and transient objects across views. PixelNeRF [51] reconstructs a neural volume with only one or few images. Moreover, Jonathan et al. proposes Mip-NeRF [1] which improves the NeRF under input with different scales, producing anti-aliased results. However, training NeRF with blurry images is still an unexplored area, as none of the aforementioned works seem to explicitly consider this kind of degradation.

Single image deblurring. Image deblurring aims at recovering a sharp image from blurry input. Usually, the blurry image is modeled as the sharp image convolved with a blur kernel, and the deblurring process is formulated as jointly solving the sharp image and the kernel given the blurry observation. This task is ill-posed since there are many sets of image-blur pairs that can synthesize the observed blurry image [46]. Classical blind deblur algorithms tackle the ill-posedness by introducing hand-crafted or learned image priors while optimizing the sharp image and kernel, such as total variation [4, 35], normalized gradient sparsity [13] and unnatural $l_0$ [49]. Since blur in
real world photographs is usually spatially-varying, many works try to reparameterize the blur kernels to a smaller solve space. Early work uses projective motion blur [42] which fits spatially-varying blur kernels using multiple homography, while region-based methods assume piece-wise constant [16] or piece-wise projective [28]. Moreover, a depth-based model is used to optimize the depth map and camera poses jointly [29, 33]. Another approach to model blur kernels is to use optical flow [9]. These methods either make a strong assumption on the blur pattern, or can only model one specific type of blur. In contrast, our method models the spatially-varying kernel using MLP, which can be generalized to different blur types. The recent trend of image deblurring is to introduce deep neural networks that directly map the blurry image to the latent sharp image [3, 14, 15, 23, 26, 39, 44, 48, 52, 53]. These approaches have outperformed traditional methods. However, this line of work highly depends on the training data and the methods often have difficulty generalizing to unseen blur types in the real world [45].

**Multi-image deblurring.** Deblurring with multi-image settings poses new challenges in aggregating information across frames and preserving temporal consistency. Optical flow is a useful tool for registering neighbor frames to the reference frame [9, 27]. However, estimating accurate optical flow is difficult and ill-posed, especially when the input is blurry. With the advancement of deep learning, one can design flow-free methods by concatenating multiple frames and directly restoring the clean frame using a CNN [41]. Another option is to use recurrent structure that propagates features across frames [10, 24, 40, 56]. Li et al. [17] extends the optical flow to feature correlation volume, which greatly improves the performance. Similarly, Son et al. [40] propose pixel volume that relaxes the requirement for accurate flow. However, these multi-image deblurring methods, which are built on image space operations, fail to exploit the 3D geometry of the scene and have difficulty addressing the multi-view input with a large baseline.

### 3. Preliminary

We first review the NeRF representation [22] for a static 3D scene. A NeRF defines the scene as a continuous volumetric function that maps a 3D position $x$ and a 2D view direction $d$ to color $c$ and volume density $σ$. Formally:

$$ (c, σ) = F_Θ (γ_L(x), γ_L(d)), $$  

where $F_Θ$ represents an MLP with parameters $Θ$, and $γ_L(·)$ is the positional encoding that maps each element of a vector into a higher dimensional frequency space:

$$ γ_L(x) = [\sin πx, \cos πx, ..., \sin 2^{L-1} πx, \cos 2^{L-1} πx]^T, $$  

where the hyper-parameter $L$ indicates the highest frequency used in the mapping and can be used to control the smoothness of the scene function [43]. To render a pixel centered at image coordinate $p$, we first emit a ray $r_p(t) = o + td_p$ from camera projection center $o$ along the viewing direction $d_p$. Then a sampling strategy is used to determine $D$ sorted distances $t^{(i)}_D$ between predefined near and far planes $t^{(0)}_D$ and $t^{(D+1)}_D$. We estimate the color $c^{(i)}$ and density $σ^{(i)}$ at each sample point $r_p(t^{(i)})$ using $F_Θ$. The final color of the pixel is computed as:

$$ \hat{c}_p = \hat{c}(r_p) = \sum_{i=1}^{D} T^{(i)} \left( 1 - \exp(-σ^{(i)}δ^{(i)}) \right)c^{(i)}, $$  

where $δ^{(i)} = t^{(i+1)} - t^{(i)}$ is the distance between adjacent samples, and $T^{(i)} = \exp(-\sum_{j=1}^{i-1} σ^{(j)}δ^{(j)})$. In this paper we use $c_p$ and $c(r_p)$ interchangeably. Note that this rendering process is trivially differentiable.

### 4. Method

Our task is to train the NeRF with blurry input. The training pipeline is visualized in Fig. 2. The core idea is to explicitly model the blurring process and seek to optimize a sharp NeRF and the blur parameters jointly so that the synthesized blurry images match the input. Specifically, to render a blurry pixel during training, we first generate multiple optimized rays using a newly proposed Deformable Sparse Kernel (DSK) module, which mimics the blurring process. We render these rays using the NeRF, and blend results to get the finally blurry color, which is then supervised by the blurry input. Note that in the inference stage, we can directly render the NeRF without the DSK to get sharp novel views. We describe the DSK as well as some other designs in the following subsections.

#### 4.1. Deformable Sparse Kernel

Akin to most image deblurring algorithms [2, 46], we model the blurring process by convolving the sharp image with a blur kernel $h$:

$$ b_p = c_p * h, $$  

where $c_p$ is the color of sharp pixel at $p$, which ideally is also the output of a sharp NeRF in our model. $b_p$ is the corresponding blurry color and $*$ stands for the convolution operator. The support of the blur kernel is usually defined in a $K × K$ window centered at $p$. To compute $b_p$, we take the sum of element-wise multiplication of $c_p$ and $h$ inside the window. This can be computed efficiently in the map-based image representation. However, a problem arises when $c_p$ is modeled as a NeRF because rendering becomes quite computation and memory consuming. For each pixel, there are $K × K$ rays that need to be rendered.
in the supporting window, thus making training infeasible. Therefore, we propose to approximate the dense blur kernel with a small number of sparse points:

\[ b_p = \sum_{q' \in \mathcal{N}'(p)} w_{q'} c_{q'}, \text{ w.r.t. } \sum_{q' \in \mathcal{N}'(p)} w_{q'} = 1, \]  

(5)

where \( \mathcal{N}(p) \) is the set of \( N \) locations sparsely distributed around \( p \) that compose the support of our sparse kernel. \( w_{q'} \) is the corresponding weight at each location. We set \( N \) to be a fixed number and ablate this hyper-parameter in our experiments (Sec. 5). Note that \( q' \) is a continuous value and we can jointly optimize the locations \( \mathcal{N}'(p) \), \( w_{q'} \) and the NeRF so that the best sparse kernel is regressed.

The blur kernel is usually spatially-varying in real world images. Inspired by the NeRF that uses an MLP as a continuous 5D function, we also choose to use an MLP to model the spatially-varying kernel. Specifically, for each input view, we assign “canonical kernel locations” \( \mathcal{N}'(p) = \{ q_i \}_{i=0}^{N-1} \), and use an MLP to deform the locations and while also predicting the weights:

\[ (\Delta q, w_q) = G_{\Phi}(p, q', I), \text{ where } q' \in \mathcal{N}'(p). \]  

(6)

Here \( G_{\Phi} \) indicates an MLP with parameter \( \Phi \) and \( I \) is a learned view embedding. This view embedding is necessary since the blur patterns usually differ across views. Optimizing a different view embedding for each view allows the DSK module to fit a different blur kernel for each view. In our experiments we set the view embedding to be a vector of length 32. We compute the final sparse kernel location in \( \mathcal{N}'(p) \) as \( q = q' + \Delta q \). Note that we need to forward the MLP \( G_{\Phi} \) for \( N \) times to get all the deformed locations. One option that may potentially boost the performance is introducing positional encoding to \( G_{\Phi} \) by replacing the input \( p \) in Eq. (6) with \( \gamma(p) \). However, we find this operation does not help to improve the quality. One possible reason is that the spatially-varying kernel changes gradually along spatial positions without high-frequency variation.

### 4.2. Convolution with Irradiance

As pointed out by Chen et al. [5], this blur convolution model should be applied to scene irradiance instead of image intensity. A more physically correct model should be

\[ b_p = g(c'_p \ast h), \]  

where \( c' \) indicates the scene irradiance and \( f(\cdot) \) is the camera response function (CRF) that maps the scene irradiance to image intensity. A nonlinear CRF will increase the complexity of the blur kernel and make the learning of DSK difficult if the linear model in Eq. (4) is used, especially in high contrast regions [46]. To compensate for the nonlinear CRF, we assume that our sharp NeRF predicts colors in linear space and adopt a simple gamma correction function in the final output:

\[ b_p = g(\sum_{q' \in \mathcal{N}'(p)} w_{q'} c'_{q'}), \]  

(7)

where \( g(c') = c'^{\frac{1}{\gamma}} \) is the gamma correction function. More complex CRFs could be used to model real world cameras, such as pre-calibrated CRFs, or jointly optimizing the CRFs during training. But we find this simple scheme is enough to compensate the nonlinearity in the imaging process and improve the quality. More discussions about modeling the CRF can be found in supplementary material.

### 4.3. Optimizing the Ray Origin

One observation of the convolutional model is that it is a 2D approximation of the actual blur model. In the convo-
lutional model, the blurry result is a combination of neighbor pixels, which are the rendering results of neighbor rays with the same camera center as the origin. However, the actual blurring process usually involves blending rays cast from different origins. Consider the two blurring processes shown in Fig. 3. When capturing camera motion blur, the camera center moves during one shot, leading to the change of the ray origins. And for defocus blur, the ray gets scattered to different directions which is equivalent to a mixture of rays from different origins. When the scene is mostly planar, the translation of the ray origin can be well approximated by the 2D translation of the pixel position. However, due to the parallax effect and occlusion, this is not the case when there is depth discontinuity. Since we have access to the 3D scene representation, we can develop kernels that consider the change of ray origins. Thus we jointly optimize the translation of the ray origin of each sparse kernel location. Specifically, we jointly predict the origin translation for each kernel location as follows Eq. (6):

$$\hat{b} \in \mathcal{N}(\mathbf{p}), \mathbf{q} \in \mathcal{N}(\mathbf{p}),$$

and then generate rays by:

$$\mathbf{r}_q = (\mathbf{o} + \Delta \mathbf{o}_q) + t \mathbf{d}_q, \mathbf{q} = \mathbf{q}' + \Delta \mathbf{q}.$$ (9)

These optimized rays are rendered and combined to get the final blurry pixels.

The training process is summarized as follows: we first predict tuples \(\{\Delta \mathbf{o}_q, \Delta \mathbf{q}, \mathbf{w}_q\}_{\mathbf{q} \in \mathcal{N}(\mathbf{p})}\) using Eq. (8), with which we generate multiple optimized rays \(\{\mathbf{r}_q\}_{\mathbf{q} \in \mathcal{N}(\mathbf{p})}\) by deforming the canonical sampling location and optimizing the ray origin as in Eq. (9). We render these rays to get \(\mathbf{c}_q'\) using Eq. (3) and blend to get a blurry pixel \(\mathbf{b}_p\) using Eq. (7). This synthetic blurry pixel is supervised by the corresponding ground truth pixel color \(\mathbf{b}_{gt}\):

$$\mathcal{L}_{\text{reconstruct}} = \sum_{\mathbf{p} \in \mathcal{R}} ||\mathbf{b}_p - \mathbf{b}_{gt}||^2,$$ (10)

where \(\mathcal{R}\) is the set of pixels in each batch. Note that our pipeline is only used during training. At test time, we can directly render the sharp results using the restored sharp NeRF with gamma correction.

4.4. Aligning the NeRF

As shown in the experiments, if we freely optimize all the learnable components, i.e., the NeRF and the deformable sparse kernel, the reconstructed NeRF may undergo some non-rigid distortion. This is in line with expectations because it is possible that the scene represented by the NeRF and the learned kernel deform together without affecting the reconstructed blurry result. However this is usually not desired. To constrain the NeRF model to align with the observations, we first initialize the deformable sparse kernel so that all the optimized rays \(\mathbf{r}_q\) are close to the input ray \(\mathbf{r}_p\). This is implemented by multiplying a small gain of \(\epsilon = 0.1\) to each element of the output tuples \(\{\Delta \mathbf{o}_q, \mathbf{q}, \mathbf{w}_q\}\). As a result, the ray origins \(\mathbf{o}_q\) and the kernel points \(\mathbf{q}\) are initialized to be close to the camera centers and the pixel locations, respectively. And all kernel points will have roughly the same weights at the beginning of training. We additionally introduce an alignment loss that forces one of the optimized rays \(\mathbf{r}_q\) to be similar to the input ray \(\mathbf{r}_p\):

$$\mathcal{L}_{\text{align}} = ||\mathbf{q}_0 - \mathbf{p}||^2 + \lambda_o ||\Delta \mathbf{o}_q||^2,$$ (11)

where \(\mathbf{q}_0\) is a fixed element in \(\mathcal{N}(\mathbf{q})\). By applying \(\mathcal{L}_{\text{align}}\), we supervise \(\mathbf{q}_0\) to be the center of the kernel. We set \(\lambda_o = 10\) in all of our experiments.

Our final loss is a combination of the NeRF reconstruction loss and the alignment loss:

$$\mathcal{L} = \mathcal{L}_{\text{reconstruct}} + \lambda_o \mathcal{L}_{\text{align}}.$$ (12)

In our experiments we set \(\lambda_o = 0.1\).

5. Experiments

5.1. Implementation Details

Training. We build our deformable sparse kernel on the Pytorch re-implementation of the NeRF [50]. We use a batch size of 1024 rays, each sample at 64 coordinates in the coarse volume and 64 additional coordinates in the fine volume. We set the number of sparse locations \(N = 5\). We use the Adam optimizer [12] with default parameters. We schedule the learning rate to start at \(5 \times 10^{-4}\) and decay exponentially to \(8 \times 10^{-5}\) over the coarse of the optimization. We train each scene for 200k iterations on a single NVIDIA V100 GPU. We adopt the same MLP structure of \(\mathcal{F}_\phi\) as the original NeRF [22], and for \(G_\phi\), we use MLP with 4 fully-connected hidden layers, each layer having 64 channels and ReLU activations. We also add a shortcut that connects the first layer to the last layer.

Datasets. In the experiments we focus on two blur types: camera motion blur and defocus blur. For each type of blur we synthesize 5 scenes using Blender [6]. We manually place multi-view cameras to mimic real data capture. To render images with camera motion blur, we randomly perturb the camera pose, and then linearly interpolate poses between the original and perturbed poses for each view. We render images from interpolated poses and blend them in linear RGB space to generate the final blurry images. For defocus blur, we use the built-in functionality to render depth-of-field images. We fix the aperture and randomly perturb the camera pose, and then linearly interpolate poses between the original and perturbed poses for each view. We render images from interpolated poses and blend them in linear RGB space to generate the final blurry images. For defocus blur, we use the built-in functionality to render depth-of-field images. We fix the aperture and randomly perturb the camera pose, and then linearly interpolate poses between the original and perturbed poses for each view. We render images from interpolated poses and blend them in linear RGB space to generate the final blurry images.
was a Canon EOS RP with manual exposure mode. We captured the camera motion blur images by manually shaking the camera during exposure, while the reference images are taken using a tripod. To capture defocus images, we choose a large aperture. We compute the camera poses of blurry and reference images in the real world scenes using the COLMAP [36, 37]. Although the estimated poses from COLMAP may be ambiguous due to the blur, we find our method is robust to inaccurate poses. One reason is that optimizing ray origins compensate for the registration errors.

### 5.2. Ablation Study

**Effectiveness of main components.** We first conduct ablations on several components in our framework: gamma correction (w/o gamma), ray origin optimization (w/o origin opt.), and the alignment loss (w/o align). We remove these components individually and train a separate NeRF in each of the synthetic camera motion blur and defocus blur datasets. We report the PSNR, SSIM, and LPIPS [55] metrics between the synthesized novel views and ground truth views. As shown in Tab. 1, overall, the best result is achieved when the full model is used. We visualize the results and the error maps of two examples in the Fig. 4. We note that other methods present larger errors, especially at the boundaries of objects. Without $L_{\text{align}}$, the NeRF produces sharp but misaligned novel views.

**Number of kernel points.** One important hyper-parameter in our method is the number of sparse locations $N = |\mathcal{N}|$. We experimented with different values of $N$ with our full model in various blur situations, considering both blur types and three degrees of blur. We plot the PSNR and SSIM curves in Fig. 5. We note that in all cases the quality of the results improves as $N$ increases. However, the improvement is less substantial beyond $N = 5$. In case
the input is extremely blurry, further increasing the number of kernel points can potentially help. However, increasing $N$ comes at a larger computation and memory cost during training. Therefore, we use $N = 5$ for all other experiments, providing a good balance between rendering quality and efficiency.

### 5.3. Comparisons

Since there are no existing works that try to reconstruct the NeRF from blurry input for novel view synthesis, we carefully select several possible baselines to compare with. The most straightforward one would be to directly train the NeRF using the blurry input (naive NeRF). Additionally, we also compare to the image-space baselines that first restore the input using the existing image or video based deblurring techniques and then train the NeRF with the deblurred images. For camera motion blur, we compare with the current state-of-the-art methods for deblurring on single image \(^{(MPR + NeRF)}\) and video \(^{(PVD + NeRF)}\). For defocus blur, we compare to the KPAC \(^{(KPAC + NeRF)}\) baselines. We show quantitative results on synthetic scenes in Tab. 2. For real scenes, due to the nature of capturing, the ground truth images are not available due to the misalignment for camera motion blur or exposure variations for defocus blur. We can see that while the image-space deblurring baseline improves compared to the naive NeRF baseline, our full pipeline outperforms these baselines to a large extent in the synthetic scenes among two blur types. The video based deblurring method PVD + NeRF sometimes performs worse than the single-image based method MPR + NeRF. A possible reason is that PVD + NeRF aggregates features from neighbor frames based on optical flow, which is really challenging for blurry input with large baselines. Fig. 6 and 7 showcase the qualitative comparison results on camera motion blur and defocus blur, respectively, in real scenes. Our method produces novel views with sharp edges and rich details, being

---

**Table 2.** Quantitative comparison on synthetic scenes of two blur types. We color code each row as best and second best.
the closest to ground truth. Previous methods demonstrate artifacts near the object boundary and blurry textures. The depth maps we predict are also sharper compared to other methods. Moreover, our method could produce more view consistent results than other baselines. And we provide additional results in conjunction with video output in the supplementary material.

6. Discussion and Conclusion

6.1. Why our framework works

Blindly recovering a sharp NeRF and the blur kernel simultaneously with only blurry images is an ill-posed problem, as the NeRF can also reconstruct a blurry scene that may “explain” the blurry images. Then how does our framework ensure that we get a sharp NeRF? As illustrated in the NeRF++ [54], the NeRF encodes priors for view consistent reconstruction. When the blurry input is view inconsistent, our framework compensates for the inconsistency using the DSK module, leading to the decomposition of the consistent sharp scene and the inconsistent blur pattern. Note that we define the blurry input as view consistent if they are equivalent to sharp observations of one blurry 3D scene. Blur in the real world is usually inconsistent. Each shot has a different blur pattern due to the randomness of camera movement or variability of focus distance. This can be further validated by the fact that when the naive NeRF reconstructs the real scene dataset, the results flicker severely when the viewpoint changes. Our framework addresses this issue, which is common in real world data.

6.2. Limitations

Our method can fail when the blur is view consistent, e.g., the camera coincidentally shaking in roughly the same direction across all views, or the camera having a fixed focal point (i.e., focuses on a single target). Deblurring a consistent blur can potentially be solved by introducing image priors, which we treat as future work. Our method may also fail when encounter input images that are severely blurred because the COLMAP may fail to reconstruct the camera poses. But in the experiments we find that this is only an issue in very blurry cases. For further discussion about such limitations, please refer to the supplementary material.

6.3. Conclusion

In this paper, we propose a simple but effective framework for training a sharp NeRF under blurry input. Experiments on both synthetic and real world scenes verify the effectiveness of our framework and demonstrate the significant improvement in quality over naive NeRF and image-space deblurring approaches. We hope that this work will further motivate research into NeRF-based approaches for deblurring applications.

Acknowledgements. Authors at HKUST and CityU were partly supported by the Hong Kong Research Grants Council (RGC), including the RGC Early Career Scheme under Grant 9048148 (CityU 21209119).
References


