# CoMoGaussian: Continuous Motion-Aware Gaussian Splatting from Motion-Blurred Images

Supplementary Material

## 1. Implementation Details

CoMoGaussian is trained for 40k iterations based on Mip-Splatting [12]. We set the number of poses $N$ that constitute the continuous camera trajectory to 9, which indicates that $M$ is 5. The embedding function of Sec. 4.2 is implemented by `nn.Embedding` of PyTorch, and the embedded features have the sizes of hidden state of 64. The sizes of hidden state of the single-layer encoders $\mathcal{E}_r$, $\mathcal{E}_c$, and the single-layer decoders $\mathcal{D}_r$, $\mathcal{D}_c$ are also 64. $\mathcal{D}_r$ consists of three head MLPs that extract the screw axis parameters $\omega$ and $v$, along with $\theta$, while $\mathcal{D}_c$ comprises two head MLPs that extract $\mathbf{A}_c$ and $\mathbf{t}_c$ as described in Sec. 4.3. The neural derivative function $f$ consists of two parallel single MLP layers, where one is designated for the rotation component and the other for the translation component, where $f$ and $g$ share the learnable parameters. To ensure nonlinearity in the camera motion within the latent space, we apply the `ReLU` activation function to each layer. The CNN $\mathcal{F}$ consists of two convolutional layers with 32 channels with kernel size of $5\times5$ for the first layer and $3\times3$ for the rest one. The pixel-wise weights are obtained by applying a pointwise convolutional layer to the output of $\mathcal{F}$, and the scalar mask $\mathcal{M}$ is obtained by applying another pointwise convolution to the output of $\mathcal{F}$. For first 1k iterations, Gaussian primitives are roughly trained without rigid body transformation and CMR transformation. After 1k iterations, those transformations start to be trained without the pixel-wise weight and the scalar mask to allow the initial camera motion path to be sufficiently optimized. After 3k iterations, the pixel-wise weight and the scalar mask start training. We set $\lambda_c$, $\lambda_o$, and $\lambda_{\mathcal{M}}$ to 0.3, $10^{-4}$, and $10^{-3}$ respectively for the objective function. All experiments are conducted on a single NVIDIA RTX 4090 GPU.

## 2. Additional Ablation Study

**Number of Poses on Camera Motion.** We conduct an ablation study on the number of camera poses used to model continuous camera motion, with the results presented in Tab. 1. To validate the effectiveness of the CMR transformation, we perform two separate experiments: one using only the rigid body transformation described in Sec. 4.2, and the other incorporating both the rigid body transformation and the CMR transformation discussed in Sec. 4.3. The trends in the three evaluation metrics with respect to different values of $N$ in Tab. 1 are visualized in Fig. 1.

When using only the rigid body transformation, performance improves almost linearly as $N$ increases across all metrics. Since a motion-blurred image is fundamentally the result of integrating sharp images over time, increasing $N$ allows for a finer discretization of the continuous camera trajectory, leading to better performance. However, numerical integration is inherently discrete, even when the camera motion itself is continuous, which explains why a larger $N$ results in improved accuracy.

Table 1. Experimental results based on the number of poses $N$ along the camera motion trajectory.

| # Cam. | Rigid Only (Sec. 4.2) | | | Rigid (Sec. 4.2) + CMR (Sec. 4.3) | | |
|---|---|---|---|---|---|---|
| | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS(↓) |
| 5 | 26.60 | 0.8101 | 0.1217 | 26.90 | 0.8245 | 0.1073 |
| 6 | 26.94 | 0.8187 | 0.1101 | 27.40 | 0.8345 | 0.0996 |
| 7 | 27.08 | 0.8218 | 0.1059 | 27.52 | 0.8377 | 0.0925 |
| 8 | 27.08 | 0.8241 | 0.1025 | 27.67 | 0.8399 | 0.0865 |
| 9 | 27.19 | 0.8249 | 0.1005 | 27.85 | 0.8435 | 0.0822 |
| 10 | 27.29 | 0.8267 | 0.0972 | 27.89 | 0.8442 | 0.0814 |
| 11 | 27.36 | 0.8295 | 0.0933 | 27.79 | 0.8423 | 0.0833 |
| 12 | 27.43 | 0.8312 | 0.0906 | 27.86 | 0.8399 | 0.0826 |
| 13 | 27.57 | 0.8341 | 0.0879 | 27.81 | 0.8406 | 0.0813 |

Nevertheless, increasing $N$ also leads to a significant rise in computational complexity during training, as each motion-blurred image requires rendering $N$ sharp images. This creates a trade-off: smaller values of $N$ suffer from discretization artifacts, while larger values of $N$ lead to increased computational costs due to multiple renderings. To address this, we introduce the CMR transformation, which adds a small degree of flexibility to the transformation matrix, effectively compensating for the limitations of discretized integration at smaller $N$.

As shown in Tab. 1 and Fig. 1, when CMR is applied alongside the rigid body transformation, performance saturates at $N = 9$ across all metrics, even surpassing the performance of the rigid body transformation alone at $N = 13$. This demonstrates that the CMR transformation mitigates the limitations of rigid body transformations at lower $N$ and justifies its inclusion in our framework.

**Orthogonal Regularization.** We conduct a qualitative ablation study on the regularization loss for the orthogonality condition introduced in Sec. 4.3 of the main paper. Although the impact of this loss may appear minor in Tab. 3 of the main paper, a qualitative comparison reveals a noticeable difference.

As shown in Fig. 2, including this regularization results in significantly improved visual quality compared to when

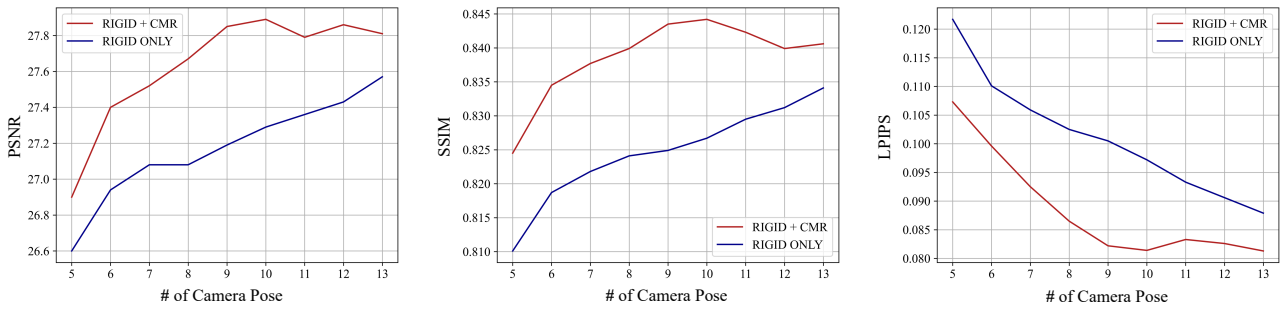


Figure 1. Performance variation based on the number of poses $N$ used to construct the camera motion trajectory.

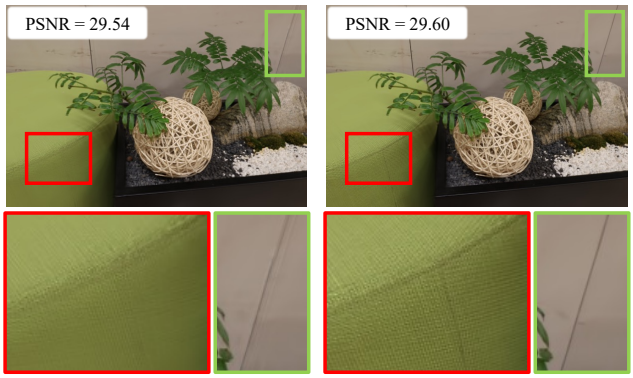


Figure 2. Qualitative comparison of results with and without orthogonal regularization. Despite similar PSNR values, the results with regularization capture more fine details.

it is omitted. Without regularization, the $3 \times 3$ linear transformation matrix is learned without constraints on shearing and scaling, which leads to unintended distortions. When such unconstrained affine transformations are included, the overall scene structure may still be well captured, but finer details tend to be lost.

By enforcing the orthogonality condition, we restrict the transformation matrix to allow only minimal deviations, preventing excessive distortions. As demonstrated in Fig. 2, while the quantitative performance remains similar, the model captures finer details more effectively, highlighting the importance of this regularization.

## 3. Difference from SMURF [6]

In this section, we compare our approach with SMURF, a methodology for handling the continuous dynamics of camera motion blur. SMURF utilizes neural ODEs to warp a given input ray into continuous rays that simulate camera motion. However, its continuous dynamics are applied only in the 2D pixel space, lacking the inclusion of higher-dimensional camera motion in 3D space. Additionally, as SMURF is implemented on Tensorial Radiance Fields (TensoRF) [1], a ray tracing-based method, it exhibits relatively slower training and rendering speeds.

In contrast, our model uses neural ODE to obtain the 3D camera poses which constitue the camera motion trajectory. Our approach incorporates higher-dimensional information compared to SMURF by operating directly in 3D space rather than the 2D pixel space. Furthermore, as our method is implemented on Mip-Splatting [12], a rasterization-based method, it ensures faster training and rendering speeds than SMURF.

## 4. Camera Pose Visualization

**Visualization for Other Estimators.** To provide a qualitative evaluation of the camera motion estimators in Tab. 4 of our main paper, we visualize the camera poses generated by the MLP-based and GRU-based estimators in Fig. 3.

Since the MLP estimator does not account for the sequential nature of camera motion, it inherently lacks continuity, resulting in a trajectory that appears discontinuous and inconsistent. While the GRU estimator produces trajectories that seem more continuous, it separately implements GRU cells for forward and backward propagation, causing discontinuities between camera poses along the trajectory.

In contrast, our neural ODE-based estimator ensures a fully continuous camera motion over time, achieving both visually smooth trajectories and superior quantitative performance compared to the MLP and GRU estimators.

**Visualization for Sharp Images.** We visualize the camera poses to examine how CoMoGaussian's camera trajectory modeling operates on the NeRF-LLFF dataset [9, 10], which consists of sharp images. As shown in Fig. 4, the camera motion trajectory for sharp images remains nearly stationary, demonstrating the generalization capability of our proposed method.

**Additional Visualization** We visualize the camera motion trajectories for input blurry images predicted by Co-

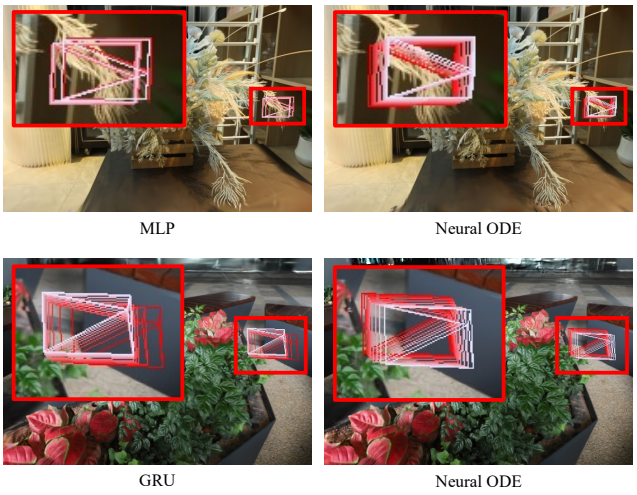


Figure 3. Comparison between the Neural ODE-based estimator and other estimators. Only the Neural ODE-based estimator exhibits a continuous camera trajectory.

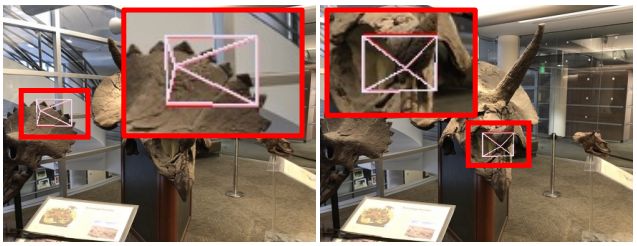

Figure 4. Visualization of camera motion trajectories for sharp images.

MoGaussian in Fig. 5 and Fig. 6. Fig. 5 illustrates camera motions for images with significant blur, where the predicted trajectories are continuous over time and align precisely with the input images. Fig. 6 depicts camera motions for images with relatively less blur, where the predicted trajectories show minimal movement, yet still match the input images accurately. These results demonstrate that our blurring kernel effectively models precise continuous camera motion.

Table 2. Comparison of training and rendering speeds across various 3DGS-based methods. * indicates that the speed is identical to that of the corresponding model.

| Methods | Training Time (hours) | Rendering Speed |
|---|---|---|
| BAD-Gaussians [13] | 0.37 | *3DGS [2] |
| Deblurring 3DGS [3] | 0.20 | *3DGS [2] |
| BAGS [11] | 0.83 | *Mip-Splatting [12] |
| **CoMoGaussian** | 1.33 | *Mip-Splatting [12] |

## 5. Training and Rendering Speed

In this section, we compare the training time and rendering speed of our method with recent 3DGS-based approaches. As shown in Tab. 2, CoMoGaussian requires a longer training time compared to other methods. However, given the quantitative results in Tabs. 3 to 5, as well as the qualitative comparisons in our *supplementary videos*, our contributions remain significant. We believe that addressing the limitations discussed in the main paper will enable faster training in the future.

Additionally, since sharp rendering is performed solely using 3DGS [2] or Mip-Splatting [12] without additional modules, our method achieves fast rendering speeds comparable to those of other approaches.

## 6. Derivation of Rigid Body Motion [7]

In this section, we explain the derivation process for Eq. (9) and Eq. (10) from the main paper. This derivation aims to expand and simplify the process described in Modern Robotics [7] for better clarity and accessibility.

The components of a given screw axis include the unit rotation axis $\hat{\omega} \in \mathbb{R}^3$ and the translation component $v \in \mathbb{R}^3$. The unit rotation axis consists of the angular velocity $\omega$ and the rotation angle $\theta$:

$$\hat{\omega} = \frac{\omega}{\theta}, \quad where \quad ||\hat{\omega}|| = 1. \tag{1}$$

We combine the rotation axis $\hat{\omega}$ and the rotation angle $\theta$ to represent an element of the Lie Algebra, $\mathfrak{so}(3)$, which serves as the linear approximation of the rotation matrix. Before proceeding, $\hat{\omega}$ is converted into a $3 \times 3$ skew-symmetric matrix $[\hat{\omega}]$ to compactly express the cross-product operation as a matrix multiplication:

$$[\hat{\omega}] = \begin{bmatrix} 0 & -\hat{\omega}_z & \hat{\omega}_y \\ \hat{\omega}_z & 0 & -\hat{\omega}_x \\ -\hat{\omega}_y & \hat{\omega}_x & 0 \end{bmatrix} \in \mathfrak{so}(3), \; where \; [\hat{\omega}]^3 = -[\hat{\omega}]. \tag{2}$$

Using the skew-symmetric matrix $[\hat{\omega}]$ and the translation component $v$, the screw axis $[S]$ is expressed. By multiplying this screw axis with $\theta$, we incorporate the magnitude of the rotation and translation along the screw axis:

$$[\mathcal{S}]\theta = \begin{bmatrix} [\hat{\omega}]\theta & v\theta \\ 0 & 0 \end{bmatrix} \in \mathfrak{se}(3), \tag{3}$$

where $\mathfrak{se}(3)$ represents the Lie Algebra, which corresponds to the infinitesimal changes of the Lie Group $SE(3)$. To map this infinitesimal change to the $SE(3)$ transformation matrix $\mathbf{T} = e^{[\mathcal{S}]\theta}$, we use the Taylor expansion, following these steps:

$$e^{[\mathcal{S}]\theta} = \sum_{n=0}^{\infty} [\mathcal{S}]^n \frac{\theta^n}{n!} \tag{4}$$

$$= I + [\mathcal{S}]\theta + [\mathcal{S}]^2 \frac{\theta^2}{2!} + \cdots \tag{5}$$

$$= \begin{bmatrix} I + [\hat{\omega}]\theta + [\hat{\omega}]^2 \frac{\theta^2}{2!} + \cdots & \left(I\theta + [\hat{\omega}] \frac{\theta^2}{2!} + \cdots\right) v \\ 0 & 1 \end{bmatrix} \tag{6}$$

$$= \begin{bmatrix} e^{[\hat{\omega}]\theta} & G(\theta)v \\ 0 & 1 \end{bmatrix} \in SE(3) \tag{7}$$

$$\because [\mathcal{S}]^n = \begin{bmatrix} [\hat{\omega}]^n & [\hat{\omega}]^{n-1} v \\ 0 & 0 \end{bmatrix} \tag{8}$$

For the rotation matrix $e^{[\hat{\omega}]\theta}$, we simplify it using the Taylor expansion and Eq. (2), resulting in:

$$e^{[\hat{\omega}]\theta} = I + [\hat{\omega}]\theta + [\hat{\omega}]^2 \frac{\theta^2}{2!} + [\hat{\omega}]^3 \frac{\theta^3}{3!} + + [\hat{\omega}]^4 \frac{\theta^4}{4!} \cdots \tag{9}$$

$$= I + \left(\theta - \frac{\theta^3}{3!} + \cdots\right) [\hat{\omega}] + \left(\frac{\theta^2}{2!} - \frac{\theta^4}{4!} + \cdots\right) [\hat{\omega}]^2 \tag{10}$$

$$= I + \sin\theta [\hat{\omega}] + (1 - \cos\theta)[\hat{\omega}]^2 \in SO(3) \tag{11}$$

The translational component $G(\theta)$ is also derived using the Taylor expansion and Eq. (2):

$$G(\theta) = I\theta + [\hat{\omega}] \frac{\theta^2}{2!} + [\hat{\omega}]^2 \frac{\theta^3}{3!} + [\hat{\omega}]^3 \frac{\theta^4}{4!} + \cdots \tag{12}$$

$$= I\theta + \left(\frac{\theta^2}{2!} - \frac{\theta^4}{4!} + \cdots\right) [\hat{\omega}] + \left(\frac{\theta^3}{3!} - \frac{\theta^5}{5!} + \cdots\right) [\hat{\omega}]^2 \tag{13}$$

$$= I\theta + (1 - \cos\theta)[\hat{\omega}] + (\theta - \sin\theta)[\hat{\omega}]^2 \tag{14}$$

The term $G(\theta)$ physically represents the total translational motion caused by the rotational motion as the rigid body rotates by $\theta$. In other words, $G(\theta)$ indicates how rotational motion contributes to translational motion, which can also be expressed as an integral of the rotation motion:

$$G(\theta) = \int_0^{\theta} e^{[\hat{\omega}]\theta} d\theta \tag{15}$$

$$= \int_0^{\theta} \left(I + \sin\theta [\hat{\omega}] + (1 - \cos\theta)[\hat{\omega}]^2\right) d\theta \tag{16}$$

$$= I\theta + (1 - \cos\theta)[\hat{\omega}] + (\theta - \sin\theta)[\hat{\omega}]^2 \tag{17}$$

Through the above process, we derive Eq. (9) and Eq. (10) in the main paper, improving readability of the paper and providing a clear foundation for understanding the mathematical framework.

## 7. Per-Scene Quantitative Results

We show the per-scene quantitative performance on Deblur-NeRF real-world, synthetic, and ExbluRF real-world dataset in Tab. 3, Tab. 4, and Tab. 5. CoMoGaussian demonstrates superior quantitative performance on most scenes in the real-world dataset.

## 8. Additional Qualitative Results

We provide additional visualization results in Fig. 7, which demonstrate that our CoMoGaussian outperforms not only in quantitative metrics but also in qualitative performance. For comparative videos, please refer to *the supplementary materials*.

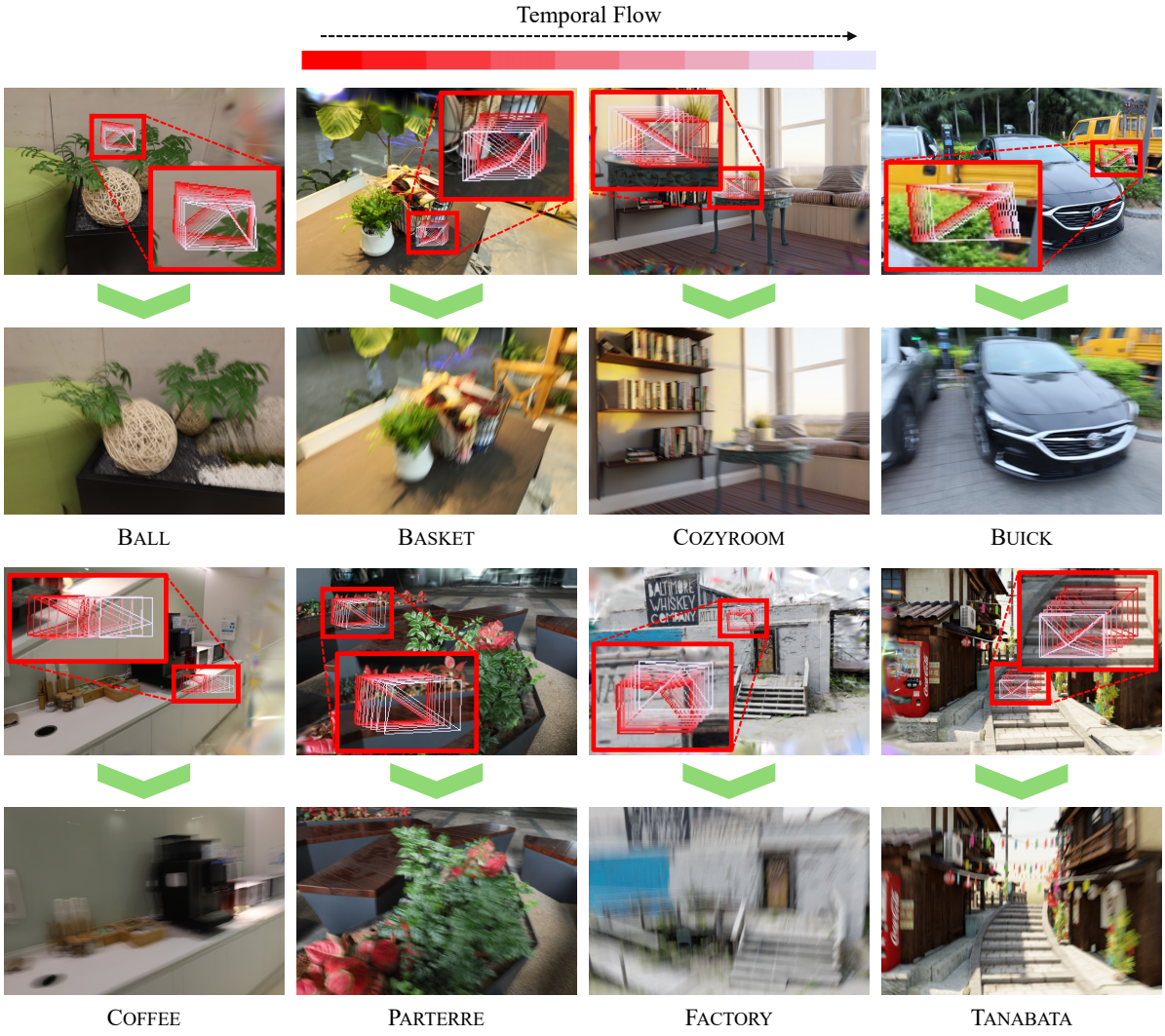


Figure 5. Camera motion trajectory predicted by CoMoGaussian for input images with significant blur.

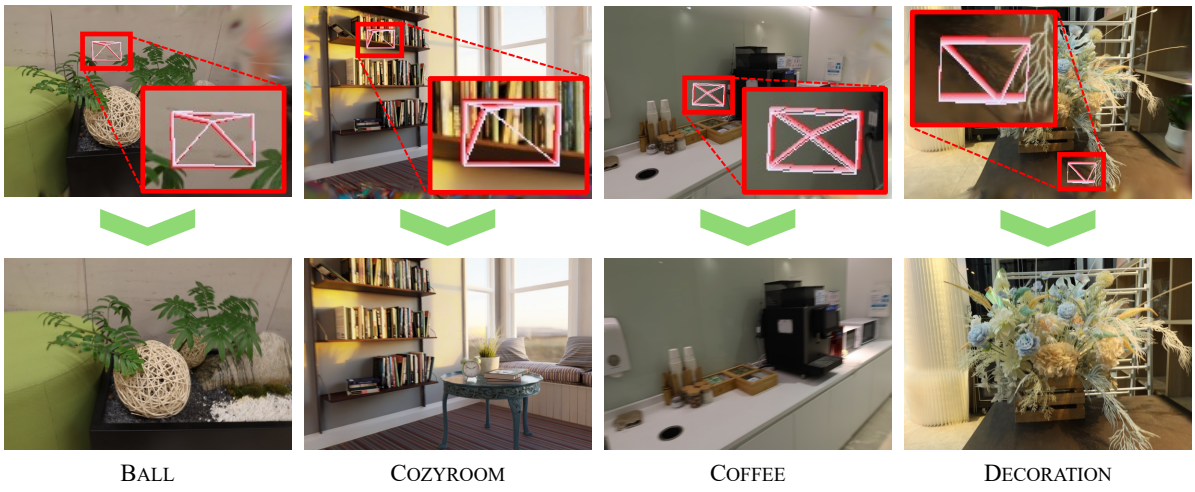


Figure 6. Camera motion trajectory predicted by CoMoGaussian for input images with moderate blur.

Table 3. Per-Scene Quantitative Performance on Deblur-NeRF Real-World Scenes.

| Real-World Scene | Ball | | | Basket | | | Buick | | | Coffee | | | Decoration | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| Naive NeRF [10] | 24.08 | 0.6237 | 0.3992 | 23.72 | 0.7086 | 0.3223 | 21.59 | 0.6325 | 0.3502 | 26.48 | 0.8064 | 0.2896 | 22.39 | 0.6609 | 0.3633 |
| Mip-Splatting [12] | 23.22 | 0.6190 | 0.3400 | 23.24 | 0.6880 | 0.2880 | 21.46 | 0.6590 | 0.2660 | 24.73 | 0.7490 | 0.2880 | 20.55 | 0.6410 | 0.2990 |
| Deblur-NeRF [8] | 27.36 | 0.7656 | 0.2230 | 27.67 | 0.8449 | 0.1481 | 24.77 | 0.7700 | 0.1752 | 30.93 | 0.8981 | 0.1244 | 24.19 | 0.7707 | 0.1862 |
| DP-NeRF [4] | 27.20 | 0.7652 | 0.2088 | 27.74 | 0.8455 | 0.1294 | 25.70 | 0.7922 | 0.1405 | 31.19 | 0.9049 | 0.1002 | 24.31 | 0.7811 | 0.1639 |
| BAD-Gaussians [13] | 22.28 | 0.6032 | 0.2054 | 22.02 | 0.7004 | 0.1197 | 19.95 | 0.6127 | 0.1103 | 25.58 | 0.7965 | 0.0932 | 21.11 | 0.6651 | 0.1185 |
| Deblurring 3DGS [3] | 28.27 | 0.8233 | 0.1413 | 28.42 | 0.8713 | 0.1155 | 25.95 | 0.8367 | 0.0954 | 32.84 | 0.9312 | 0.0676 | 25.87 | 0.8540 | 0.0933 |
| BAGS [11] | 27.68 | 0.7990 | 0.1500 | 29.54 | 0.9000 | 0.0680 | 26.18 | 0.8440 | 0.0880 | 31.59 | 0.9080 | 0.0960 | 26.09 | 0.8580 | 0.0830 |
| **CoMoGaussian** | 29.60 | 0.8422 | 0.1115 | 30.78 | 0.9041 | 0.0761 | 27.23 | 0.8502 | 0.0742 | 33.04 | 0.9247 | 0.0578 | 26.44 | 0.8601 | 0.0891 |

| Real-World Scene | Girl | | | Heron | | | Parterre | | | Puppet | | | Stair | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| Naive NeRF [10] | 20.07 | 0.7075 | 0.3196 | 20.50 | 0.5217 | 0.4129 | 23.14 | 0.6201 | 0.4046 | 22.09 | 0.6093 | 0.3389 | 22.87 | 0.4561 | 0.4868 |
| Mip-Splatting [12] | 19.87 | 0.7140 | 0.2780 | 19.43 | 0.5050 | 0.3320 | 22.28 | 0.5900 | 0.3210 | 22.05 | 0.6310 | 0.2670 | 21.91 | 0.4740 | 0.3870 |
| Deblur-NeRF [8] | 22.27 | 0.7976 | 0.1687 | 22.63 | 0.6874 | 0.2099 | 25.82 | 0.7597 | 0.2161 | 25.24 | 0.7510 | 0.1577 | 25.39 | 0.6296 | 0.2102 |
| DP-NeRF [4] | 23.33 | 0.8139 | 0.1498 | 22.88 | 0.6930 | 0.1914 | 25.86 | 0.7665 | 0.1900 | 25.25 | 0.7536 | 0.1505 | 25.59 | 0.6349 | 0.1772 |
| BAD-Gaussians [13] | 19.16 | 0.7037 | 0.1178 | 19.47 | 0.5264 | 0.1747 | 21.71 | 0.6154 | 0.1165 | 21.74 | 0.6506 | 0.1166 | 23.86 | 0.5969 | 0.0892 |
| Deblurring 3DGS [3] | 23.26 | 0.8390 | 0.1011 | 23.14 | 0.7438 | 0.1543 | 26.17 | 0.8144 | 0.1206 | 25.67 | 0.8051 | 0.0941 | 26.46 | 0.7050 | 0.1123 |
| BAGS [11] | 25.45 | 0.8690 | 0.0790 | 22.04 | 0.7150 | 0.1260 | 25.92 | 0.8190 | 0.0920 | 25.81 | 0.8040 | 0.0940 | 26.69 | 0.7210 | 0.0800 |
| **CoMoGaussian** | 27.08 | 0.8884 | 0.0733 | 23.18 | 0.7350 | 0.1326 | 26.11 | 0.8131 | 0.0872 | 27.09 | 0.8337 | 0.0658 | 27.92 | 0.7792 | 0.0541 |

Table 4. Per-Scene Quantitative Performance on Deblur-NeRF Synthetic Scenes.

| Synthetic Scene | Factory | | | CozyRoom | | | Pool | | | Tanabata | | | Trolley | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| Naive NeRF [10] | 19.32 | 0.4563 | 0.5304 | 25.66 | 0.7941 | 0.2288 | 30.45 | 0.8354 | 0.1932 | 22.22 | 0.6807 | 0.3653 | 21.25 | 0.6370 | 0.3633 |
| Mip-Splatting [12] | 18.21 | 0.4234 | 0.4769 | 25.25 | 0.7968 | 0.1646 | 30.57 | 0.8483 | 0.1456 | 21.54 | 0.6754 | 0.3075 | 20.82 | 0.6388 | 0.3165 |
| Deblur-NeRF [8] | 25.60 | 0.7750 | 0.2687 | 32.08 | 0.9261 | 0.0477 | 31.61 | 0.8682 | 0.1246 | 27.11 | 0.8640 | 0.1228 | 27.45 | 0.8632 | 0.1363 |
| DP-NeRF [4] | 25.91 | 0.7787 | 0.2494 | 32.65 | 0.9317 | 0.0355 | 31.96 | 0.8768 | 0.0908 | 27.61 | 0.8748 | 0.1033 | 28.03 | 0.8752 | 0.1129 |
| BAD-Gaussians [13] | 17.86 | 0.3892 | 0.1440 | 23.50 | 0.7396 | 0.0616 | 26.90 | 0.7296 | 0.1127 | 20.54 | 0.6379 | 0.0860 | 21.26 | 0.6921 | 0.0963 |
| Deblurring 3DGS [3] | 24.01 | 0.7333 | 0.2326 | 31.45 | 0.9222 | 0.0367 | 31.87 | 0.8829 | 0.0751 | 27.01 | 0.8807 | 0.0785 | 26.88 | 0.8710 | 0.1028 |
| BAGS [11] | 22.35 | 0.6639 | 0.2277 | 32.21 | 0.9359 | 0.0245 | 28.72 | 0.8404 | 0.0804 | 26.79 | 0.8735 | 0.1099 | 26.61 | 0.8627 | 0.1156 |
| **CoMoGaussian** | 29.32 | 0.8971 | 0.0563 | 33.34 | 0.9427 | 0.0239 | 32.45 | 0.8924 | 0.0705 | 29.53 | 0.9273 | 0.0408 | 30.45 | 0.9240 | 0.0546 |

Table 5. Per-Scene Quantitative Performance on the ExbluRF Real-World Scenes.

| ExbluRF | Bench | | | Camellia | | | Dragon | | | Jars | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| Mip-Splatting [12] | 24.58 | 0.5671 | 0.6190 | 23.28 | 0.5151 | 0.5886 | 28.65 | 0.5403 | 0.7002 | 24.08 | 0.5335 | 0.6094 |
| ExbluRF [5] | 24.75 | 0.5783 | 0.3003 | 23.14 | 0.4925 | 0.3630 | 24.26 | 0.4042 | 0.5641 | 22.21 | 0.4591 | 0.4213 |
| BAD-Gaussians [13] | 28.27 | 0.7125 | 0.2266 | 23.39 | 0.5102 | 0.3034 | 30.24 | 0.6383 | 0.4374 | 28.41 | 0.7041 | 0.3347 |
| Deblurring 3DGS [3] | 30.44 | 0.7708 | 0.2587 | 26.26 | 0.6401 | 0.3964 | 30.87 | 0.6643 | 0.5561 | 27.56 | 0.6559 | 0.4431 |
| BAGS [11] | 25.40 | 0.6142 | 0.4962 | 23.29 | 0.5177 | 0.5450 | 29.06 | 0.5577 | 0.6738 | 24.00 | 0.5402 | 0.5534 |
| **CoMoGaussian** | 31.82 | 0.8011 | 0.2170 | 28.53 | 0.7004 | 0.2846 | 31.95 | 0.7166 | 0.4476 | 29.63 | 0.7351 | 0.3474 |

| ExbluRF | Jars2 | | | Postbox | | | Stone Lantern | | | Sunflowers | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ | PSNR↑ | SSIM↑ | LPIPS↓ |
| Mip-Splatting [12] | 22.10 | 0.5682 | 0.5817 | 23.19 | 0.5277 | 0.5710 | 22.64 | 0.6004 | 0.6308 | 25.78 | 0.6775 | 0.5020 |
| ExbluRF [5] | 21.99 | 0.5736 | 0.3519 | 23.34 | 0.5287 | 0.2978 | 26.18 | 0.6832 | 0.4236 | 25.25 | 0.6765 | 0.3223 |
| BAD-Gaussians [13] | 26.27 | 0.6914 | 0.3326 | 25.01 | 0.6264 | 0.2760 | 25.19 | 0.6724 | 0.3794 | 27.82 | 0.7443 | 0.2865 |
| Deblurring 3DGS [3] | 26.76 | 0.7100 | 0.3942 | 23.89 | 0.5563 | 0.3492 | 23.32 | 0.6430 | 0.4687 | 29.75 | 0.7955 | 0.3248 |
| BAGS [11] | 22.20 | 0.5658 | 0.5268 | 24.76 | 0.5891 | 0.4205 | 22.72 | 0.5979 | 0.5556 | 26.14 | 0.6920 | 0.4508 |
| **CoMoGaussian** | 29.72 | 0.7699 | 0.3283 | 29.99 | 0.7631 | 0.2479 | 28.66 | 0.7549 | 0.3402 | 30.90 | 0.8062 | 0.2622 |

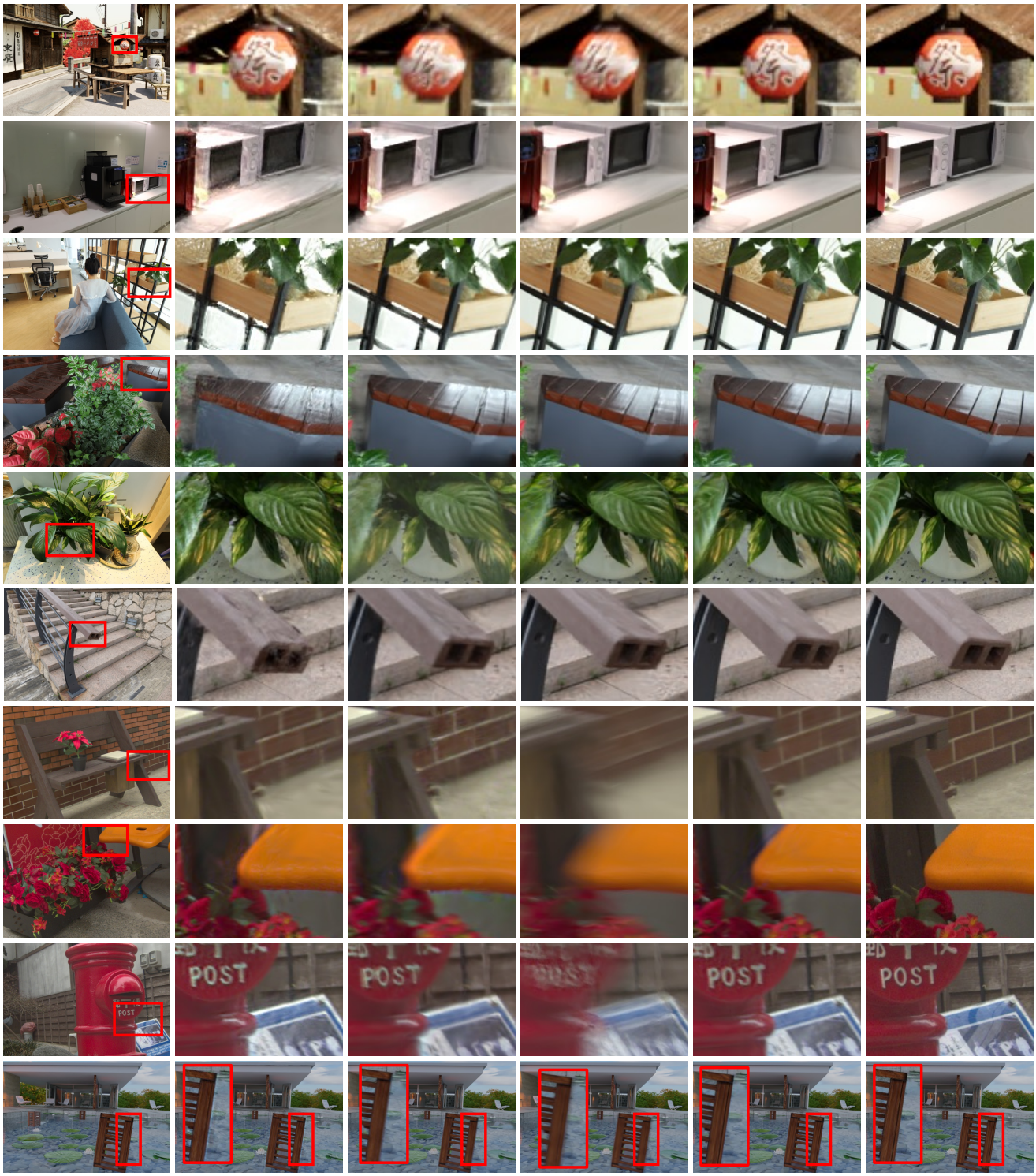


Figure 7. Additional Qualitative Comparison on the Synthetic and Real-World Scenes.

## References


[1] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision*, pages 333–350. Springer, 2022. 2

[2] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42 (4):1–14, 2023. 3

[3] Byeonghyeon Lee, Howoong Lee, Xiangyu Sun, Usman Ali, and Eunbyung Park. Deblurring 3d gaussian splatting. *arXiv preprint arXiv:2401.00834*, 2024. 3, 6

[4] Dogyoon Lee, Minhyeok Lee, Chajin Shin, and Sangyoun Lee. Dp-nerf: Deblurred neural radiance field with physical scene priors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12386–12396, 2023. 6

[5] Dongwoo Lee, Jeongtaek Oh, Jaesung Rim, Sunghyun Cho, and Kyoung Mu Lee. Exblurf: Efficient radiance fields for extreme motion blurred images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17639–17648, 2023. 6

[6] Jungho Lee, Dogyoon Lee, Minhyeok Lee, Donghyung Kim, and Sangyoun Lee. Smurf: Continuous dynamics for motion-deblurring radiance fields. *arXiv preprint arXiv:2403.07547*, 2024. 2

[7] Kevin M Lynch and Frank C Park. *Modern robotics*. Cambridge University Press, 2017. 3

[8] Li Ma, Xiaoyu Li, Jing Liao, Qi Zhang, Xuan Wang, Jue Wang, and Pedro V Sander. Deblur-nerf: Neural radiance fields from blurry images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12861–12870, 2022. 6

[9] Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (ToG)*, 38(4):1–14, 2019. 2

[10] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I*, pages 405–421, 2020. 2, 6

[11] Cheng Peng, Yutao Tang, Yifan Zhou, Nengyu Wang, Xijun Liu, Deming Li, and Rama Chellappa. Bags: Blur agnostic gaussian splatting through multi-scale kernel modeling. *arXiv preprint arXiv:2403.04926*, 2024. 3, 6

[12] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-splatting: Alias-free 3d gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19447–19456, 2024. 1, 2, 3, 6

[13] Lingzhe Zhao, Peng Wang, and Peidong Liu. Bad-gaussians: Bundle adjusted deblur gaussian splatting. *arXiv preprint arXiv:2403.11831*, 2024. 3, 6