

PanSplat: 4K Panorama Synthesis with Feed-Forward Gaussian Splatting

Supplementary Material

The supplementary material is organized as follows. In Sec. A, we provide additional details on the network architectures. In Sec. B, we provide additional details on the Gaussian parameter prediction and rendering. In Sec. C, we provide additional details on the experiment settings. In Sec. D, we provide quantitative comparisons on narrow baselines. In Sec. E, we provide more ablation studies. In Sec. F, we provide details on extending to real data. In Sec. G, we provide details on scaling up to 4K resolution. Finally, in Sec. H, we provide details on the demo video.

A. Network Architectures

In Sec. 3 of the main paper, we present our PanSplat architecture in two parts: the Hierarchical Spherical Cost Volume (Sec. 3.2) and the Gaussian Heads (Sec. 3.3). Here, we provide additional details on the network architectures.

Hierarchical Spherical Cost Volume. For feature pyramid extraction, we adopt a FPN architecture [46] enhanced with a Swin Transformer [48]. The Swin Transformer consists of 6 Transformer blocks, each with a self-attention layer and a cross-view attention layer. We use the xFormers [36] library for the transformer-based network for better efficiency. We apply Swin Transformer to the coarsest level of the feature map from the FPN encoder, then upsample the feature map to different levels with the FPN decoder. The result is a feature pyramid with 4 levels, with channel dimensions of 128, 96, 64, 32 from the coarsest to the finest level. For hierarchical spherical cost volume refinement, we adopt a 2DU-Net [16] with cross-view attention at the bottleneck layer for each level. We set depth candidates to 128, 64, 32 and channel dimensions of 2D U-Net to 128, 64, 32 for each level, respectively.

Gaussian Heads. We adopt a lightweight 3-layer CNN architecture for each Gaussian head, with a kernel size of 3×3 and a stride of 1, to extract feature map \tilde{F}_i^l for each view i at level l . We then sample a feature vector from the feature maps for each Gaussian, based on the pixel location defined on the Fibonacci lattice. Finally, a linear layer is applied to predict the Gaussian parameters $(\mu_i^l, \alpha_i^l, \Sigma_i^l, c_i^l)$. Specifically, to estimate Gaussian centers μ_i^l , we first estimate the correlation vectors c_i^l , then apply the same operations used for the cost volume to get a depth, which is then unprojected to 3D coordinates as mentioned in Sec. 3.1 of the main paper. The opacity α_i^l is predicted as a scalar value, followed by a sigmoid activation to normalize it to $[0, 1]$. The covariance Σ is composed of scaling vectors and quaternions, where the scaling is calculated as predicted normalized vectors $s_i^l \in [s_{\min}, s_{\max}]$ multiplied by the pixel size. This re-

stricts the Gaussian to a similar scale as the pixel, accounting for the change in pixel size across different levels. The color c_i^l is represented as spherical harmonic coefficients.

B. Gaussian Parameter Prediction and Rendering Details

In Sec. 3.3 of the main paper, we introduce Gaussian heads with local operations and a cubemap renderer. Based on these two components, we propose a two-step deferred backpropagation technique to enable training at 4K resolution. Here, we provide additional details on the deferred backpropagation technique, as shown in Fig. B.1, as well as the two components it relies on.

Tiled Operation for Gaussian Heads. We mentioned in Sec. 3.3 of the main paper that we exploit the local operations in the Gaussian heads to enable tiled operation for inference and deferred backpropagation. To be more specific, the inputs to the Gaussian heads on different levels are evenly split into $N \times N$ tiles, then fed into the Gaussian heads separately. However, this naive tiled operation impacts the boundary value of the output tiles, due to the zero padding of each convolutional layer, leading to discontinuity at the tile boundaries. Instead, we refine this design to output results identical to the non-tiled operation with a pre-padding operation. First the inputs are padded by 3 pixels, to accommodate the field of perception of the Gaussian heads. The padding involves copying the border pixels of left and right sides to the opposite side, which ensures loop continuity of the spherical geometry. The top and bottom sides are padded with zeros. Then, the tile regions are enlarged by 3 pixels to include the above padding, and introduce a 3-pixel overlap between adjacent tiles. The output tiles are finally cropped to the original size, stitched to a continuous, full resolution output.

Details of Cubemap Renderer. One key component of two-step deferred backpropagation is the cubemap renderer, which provides a differentiable rendering pipeline for the spherical 3D Gaussian pyramid. As shown in Fig. B.1, the cubemap renderer renders 6 faces (front, back, left, right, top, bottom) of the cubemap separately, then stitches them into an equirectangular panorama. This allows sequential face rendering for memory efficiency or batched face rendering for speedup. We build the cubemap renderer based on the CUDA 3DGS renderer [34] that implements with perspective camera projection. After rendering each face, we apply a bilinear grid sampling to stitch the faces into an equirectangular panorama. Specifically, the coordinates of pixels in the equirectangular panorama are first transformed

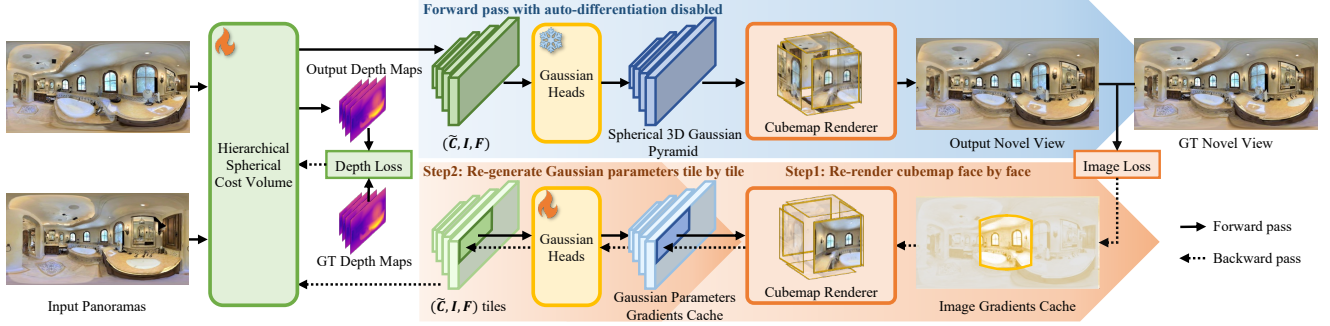


Figure B.1. **Two-step deferred backpropagation.** We propose a training strategy tailored for high-resolution panorama novel view synthesis. See Sec. B for details. *For simplicity, intermediate results of only a single view are shown.*

to the corresponding coordinates on the cubemap image. Then the pixel values are sampled from the cubemap image using bilinear interpolation. To achieve seamless stitching, we pad the edge pixels of the adjacent 4 faces to each face, ensuring the pixels interpolated on the edge have correct neighboring pixels from two nearby faces.

Details of Two-step Deferred Backpropagation. As shown in Fig. B.1, the two-step deferred backpropagation consists of a forward pass and two deferred backpropagation steps. Before the forward pass, we construct the hierarchical spherical cost volume with auto-differentiation on, and preserves the computational graph throughout the training step for efficiency. Then we disable auto-differentiation for a forward pass to render the full panorama. The full panorama is used for computing an image loss, with auto-differentiation on, to backpropagate and cache gradients to the image. Subsequently, we enable auto-differentiation and backpropagate gradients in two steps. In step one, the panorama is re-rendered face by face as cubemap to backpropagate and accumulate gradients to the Gaussian parameters. In step two, the Gaussian parameters are re-generated tile by tile, with gradients backpropagated and accumulated to the network parameters. Additionally, the gradients from the depth loss are accumulated to the network together with the gradients from the image loss. When training on real datasets without ground truth depth, the depth loss is replaced by auxiliary Gaussian heads and image loss as discussed in Sec. 3.4 of the main paper. In Sec. G, we provide more details on how the two-step deferred backpropagation saves memory consumption during training.

C. Experiment Details

High-resolution Synthetic Datasets. For synthetic data, we use the low-resolution (512×1024) synthetic datasets Matterport3D [11], Replica [56], and Residential [26] rendered by PanoGRF [17]. Additionally, we render two high-resolution datasets (1024×2048 / 2048×4096) using Matterport3D for fine-tuning. Specifically, we follow PanoGRF’s rendering protocol to render 6 perspective images at 512×512 / 1024×1024 resolution respectively on

the cubemap faces, then stitch them into an equirectangular panorama image. We render 2 views with a baseline of 1.0 meter as input, and 1 view in the middle as the target view. The two datasets contain 5,000 / 2,000 samples for training. We render the test set in consistent with PanoGRF, with 10 samples for each dataset, which are used for demonstration in the demo video.

High-resolution Real Datasets. We use two real-world datasets to demonstrate generalization to real-world scenarios. For fine-tuning to real images, we use the 360Loc [30] dataset as it provides accurate pose registration from dense point cloud reconstructions and lidar scans. In addition, it is the largest dataset with high-resolution panoramic image sequences as far as we know, with 18 sequences (12 daytime and 6 nighttime) across 4 scenes, totaling 9,334 frames. We select one scene with 5 sequences as the test set, and fine-tune on the other 3 scenes with 13 sequences. When fine-tuning, we randomly sample two views with varying baselines spaced 1 to 4 frames apart and select a target view in between. During evaluation, we select two views spaced 2 frames apart as input, and use all 4 views as the target to calculate the metrics. For analyzing image quality over different frame distances in Sec. F, we find that 360Loc is too sparse (average baseline of 0.47 meters) to provide a reasonable amount of frame distance samples. Therefore, we also capture a high-resolution Insta360 dataset with two sequences (one indoor and one outdoor) totaling 38K frames. Insta360 is recorded at 8K resolution and 24 FPS, later down-sampled to 4K for evaluation. We use OpenVSLAM [57] for camera pose estimation, disabling loop closure to avoid bad loop detection in repetitive environments. For evaluation purposes, we select two views spaced 15 frames apart as input, and evaluate all 17 frames. For evaluation on both datasets, we evenly sample 100 pairs of input views for each sequence, and average the results over all target views.

Implementation Details. We set the number of depth candidates D for the coarsest level to 128. Our model is implemented in PyTorch and trained on a single 80GB NVIDIA A100 GPU using the Adam optimizer with a learning rate of 2×10^{-4} . We use the pre-trained weights of UniMatch [71]

Baseline	0.2m				0.5m			
Method	PSNR \uparrow	WS-PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow	PSNR \uparrow	WS-PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
S-NeRF	20.79	19.52	0.697	0.376	17.95	16.81	0.628	0.486
OmniSyn	28.95	28.26	0.913	0.180	26.59	26.07	0.890	0.201
IBRNet	30.53	29.63	0.927	0.136	28.22	27.26	0.884	0.199
NeuRay	33.54	32.33	0.949	0.107	30.88	29.81	0.920	0.154
PanoGRF	34.29	33.27	0.952	0.098	31.41	30.46	0.924	0.132
MVSplat	32.93	32.04	0.955	0.063	31.55	30.58	0.943	0.075
PanSplat	33.92	32.88	0.959	0.066	32.46	31.42	0.950	0.072

Table D.1. **Quantitative comparison on narrow baselines.** We compare on Matterport3D under the baseline of 0.2 and 0.5 meters. Top results are highlighted in top1, top2, and top3.

to initialize the Swin Transformer of feature pyramid extractor. We also load the pre-trained weights of the monocular depth model [33] trained by PanoGRF [17] and freeze it during training. Initially, we train the model on Matterport3D with an image height of 256 and a batch size of 6 for 10 epochs, then fine-tune it with an image height of 512 and a batch size of 2 for 5 epochs. For 4K Matterport3D fine-tuning, we gradually increase the resolution from a height of 1024 to 2048 over 3 epochs at each stage. To fine-tune on 4K 360Loc, we incrementally raise the resolution from a height of 512 to 1024 and finally 2048, with 65K, 26K, and 13K iterations for each stage, respectively. At resolutions of 1024 and 2048, we enable two-step deferred backpropagation with 4 and 16 tiles, setting batch sizes to 3 and 1, respectively. When fine-tuning on 360Loc at 1024 and 2048, we freeze the hierarchical spherical cost volume and only fine-tune the Gaussian heads. During evaluation, we generalize the model directly from Matterport3D to the Replica and Residential, and from 360Loc to the Insta360 dataset, without additional fine-tuning.

D. Quantitative Comparisons on Narrow Baselines

We follow the evaluation protocol of PanoGRF [17] to further evaluate on generalization to narrow baselines on Matterport3D. As shown in Tab. D.1, while PanSplat achieves the best performance at the 0.5m baseline, it also demonstrates competitive results at the 0.2m baseline, indicating strong generalization across varying baseline distances.

E. More Ablation Studies

In Sec. 4.3 of the main paper, we conduct an ablation study to analyze the contributions of Fibonacci Gaussians and the 3D Gaussian pyramid. Here, we provide additional ablation studies in Tab. E.1 and Fig. E.1 to further analyze the impact of specific design choices in PanSplat.

Monocular Depth Features. We first ablate the use of monocular depth features in the hierarchical spherical cost

Setup	WS-PSNR \uparrow	SSIM \uparrow	LPIPS \downarrow
w/o Mono depth	28.84	0.929	0.092
w/o 3DGP residual	28.14	0.922	0.102
w/o Hierarchical CV	26.95	0.857	0.180
w/o First three GHs	28.05	0.919	0.105
Full	28.81	0.931	0.091

Table E.1. **Full ablation study.** We evaluate the impact of certain design choices on PanSplat’s performance. Mono depth refers to integrating monocular depth feature from PanoGRF [17] to the hierarchical spherical cost volume, which is not our contribution and is insignificant to performance, but we include it in the Full model for the best results. Other design choices are ablated from the Full model, and significantly affect the performance.

volume (w/o Mono depth) in Tab. E.1. We note that integrating monocular depth features is a common practice in multi-view stereo methods [17, 73]. Although in our case, the improvement is marginal, we include it in our final model for the best performance.

Residual Design of 3D Gaussian Pyramid. Second, we ablate the residual design of the Gaussian heads (w/o 3DGP residual), which leads to a significant drop in performance. To justify the performance gain from the residual design, we separately render the Gaussians from each level in Fig. E.1. It is shown that without the residual design, the coarsest two levels (Level #3 and #2) fail to output meaningful Gaussians, while the full model successfully distributes low frequency details to the coarser levels. This demonstrates the effectiveness of the residual design in guiding the Gaussian heads to capture multi-scale details.

Hierarchical Designs. Finally, we ablate the Hierarchical Cost Volume (w/o Hierarchical CV) and the First three Gaussian heads (w/o First 3 GH) respectively to analyze the joint impact of the two hierarchical designs. Similar to Sec. 4.3 of the main paper, for w/o Hierarchical CV, we replace the hierarchical cost volume with a single 1/4-resolution cost volume with 128 depth candidates to main-

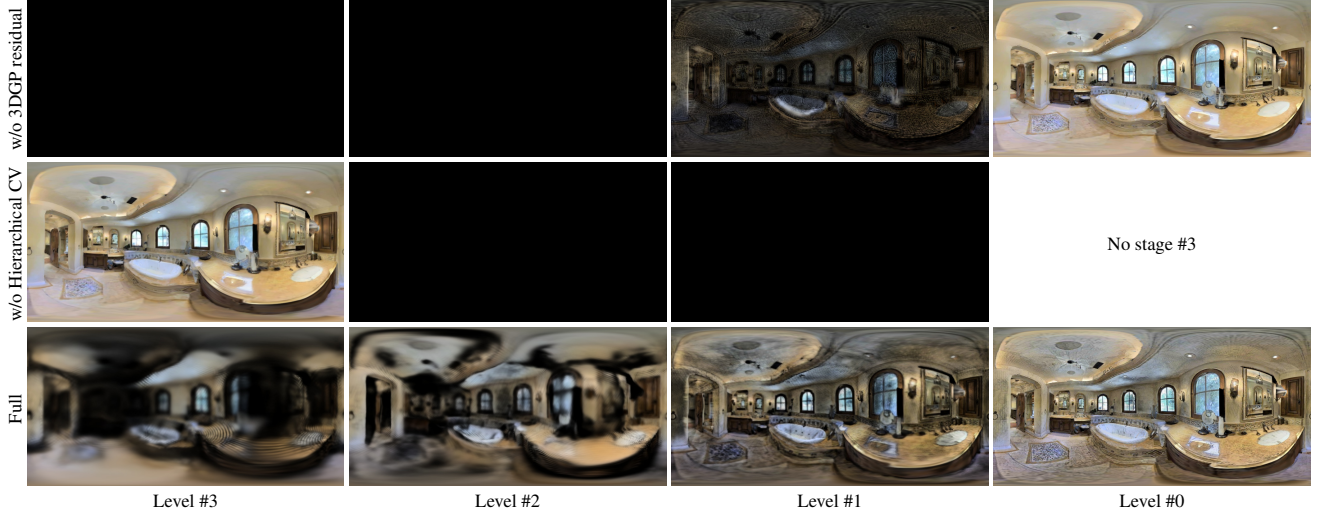


Figure E.1. **Visualization of 3D Gaussian Pyramid.** We visualize the rendering results of Gaussians from different levels of our 3D Gaussian Pyramid. Our full model (Full) successfully exploits the hierarchical structure of the 3D Gaussian Pyramid, where coarser levels mainly capture global structures and finer levels capture high-frequency details. In contrast, the ablated models (w/o 3DGP residual and w/o Hierarchical CV) fail to utilize all levels.

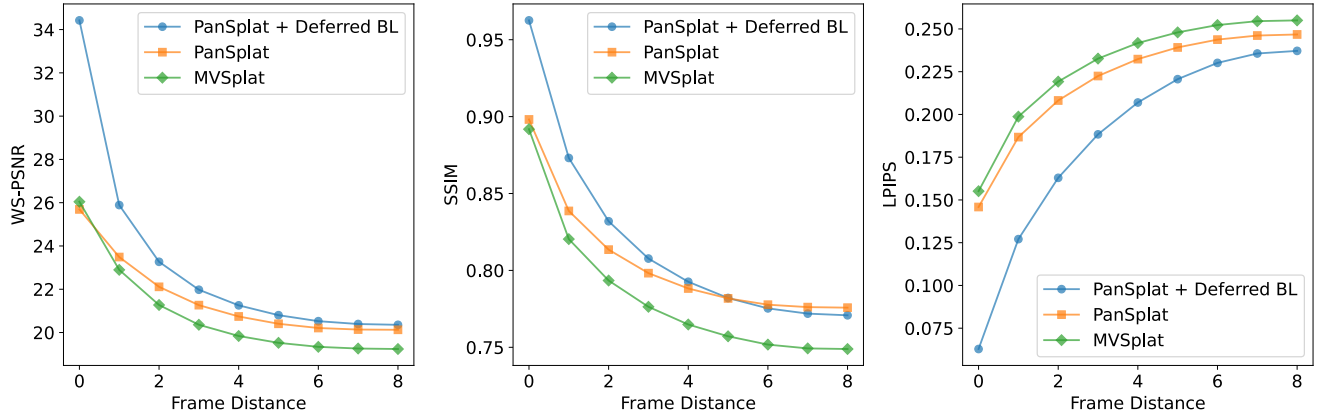


Figure F.1. **Quantitative comparisons on different frame distances.** We evaluate image quality metrics on Insta360 dataset with varying frame distances, comparing PanSplat with (PanSplat + Deferred BL) and without (PanSplat) deferred blending against MVSplat.

tain comparable computational cost and memory usage. The removal of each of the two components hurts the performance significantly, indicating that the two hierarchical designs complement each other to achieve the best performance. We find that w/o Hierarchical CV tends to fall into local minima where only the coarsest level is utilized, as shown in Fig. E.1.

F. Extending to Real Data

Deferred Blending. In Sec. 3.3 of the main paper, we introduce a deferred blending technique to mitigate artifacts from misaligned Gaussians due to moving objects and depth inconsistencies. Here we provide additional details. Specifically, on real datasets, instead of directly consolidating the Gaussians from two input views for rendering, we first sep-

arately render them from the same target view into two different images, which we denote as $\{\tilde{I}_i\}_{i=0}^1$. Then we blend them based on the distances d_i to the input views i by:

$$I = \frac{d_1 \tilde{I}_0 + d_0 \tilde{I}_1}{d_0 + d_1}. \quad (4)$$

The deferred blending aims to mitigate the influence of farther input view when rendering close to one of the input views, and relief the burden of matching moving objects.

Experiments. To evaluate the impact of deferred blending, we analyze the relationship between image quality (WS-PSNR, SSIM, and LPIPS) and frame distance (the number of frames between the target view and the nearest input view) on the Insta360 dataset. We compare PanSplat with (PanSplat + Deferred BL) and without (PanSplat) de-

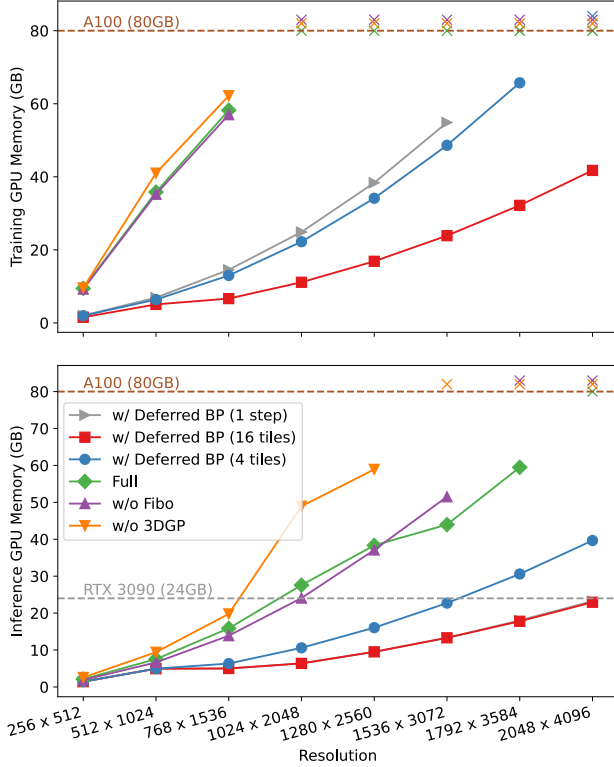


Figure G.1. **Full GPU memory consumption at different resolutions**, where \times indicates out-of-memory errors even on a 80GB A100. Note that w/ Deferred BP (1 step) is overlapped with w/ Deferred BP (16 tiles) for inference. Memory consumption is tested with a batch size of 1.

ferred blending, using MVSplat as a baseline. As shown in Fig. F.1, PanSplat consistently outperforms MVSplat across all metrics and frame distances. In addition, deferred blending provides notable performance gains, especially when the frame distance is small. We further show visual comparisons on the 360Loc dataset in Figs. F.2 to F.5 and on the Insta360 dataset in Figs. F.6 to F.9. These results demonstrate that deferred blending significantly reduces artifacts arising from misaligned Gaussians (e.g., the dot pattern on the ceiling in Fig. F.7) and moving objects (e.g., the camera operator at the bottom in Fig. F.3). It also provides nearly perfect results when rendering at the same location as one of the input views by isolating the influence of the farther input view. This is particularly important for smooth transitions in virtual tours applications as shown in the demo video.

G. Scaling Up to 4K Resolution

In Sec. 4.3 of the main paper, we evaluate how two-step deferred backpropagation saves memory consumption during training. Here, we provide additional details on the both

training and inference memory usage in Fig. G.1.

How do Fibo and 3DGP help save memory? Comparing PanSplat (Full) with ablated versions (w/o Fibo, w/o 3DGP), we find that although the removal of 3D Gaussian pyramid (w/o 3DGP) introduces less Gaussians, it still consumes more memory due to slightly larger memory footprint of single cost volume. On the other hand, during inference, the removal of Fibonacci Gaussians (w/o Fibo) causes out-of-memory error starting from 1792×3584 resolution, a resolution that PanSplat can still support.

How does deferred backpropagation help save memory?

We then add deferred backpropagation (w/ Deferred BP) with tile settings of 2×2 (4 tiles) and 4×4 (16 tiles). As shown, the memory consumption drops significantly, with 16 tiles further enabling 4K inference on a 24GB RTX 3090 GPU. We use 4 tiles for fine-tuning at 1024×2048 resolution and 16 tiles for fine-tuning at 2048×4096 resolution, with a batch size of 3 and 1, respectively.

How does two-step design based on cubemap renderer help save memory? We also include an ablated version with only step 2 of deferred backpropagation (1 step) with 16 tiles setting. The results show that the one-step version consumes significantly more memory than the two-step version when training, showing the effectiveness of cubemap renderer in reducing memory consumption. We note that the inference memory usage stays consistent as they share the same cubemap renderer with sequential face rendering.

H. Demo Video

By enabling 4K resolution support, PanSplat becomes a promising solution for immersive VR and virtual tours applications. We provide a demo video to demonstrate the superior image quality of PanSplat on diverse datasets, and to showcase its potential applications in real-world scenarios.

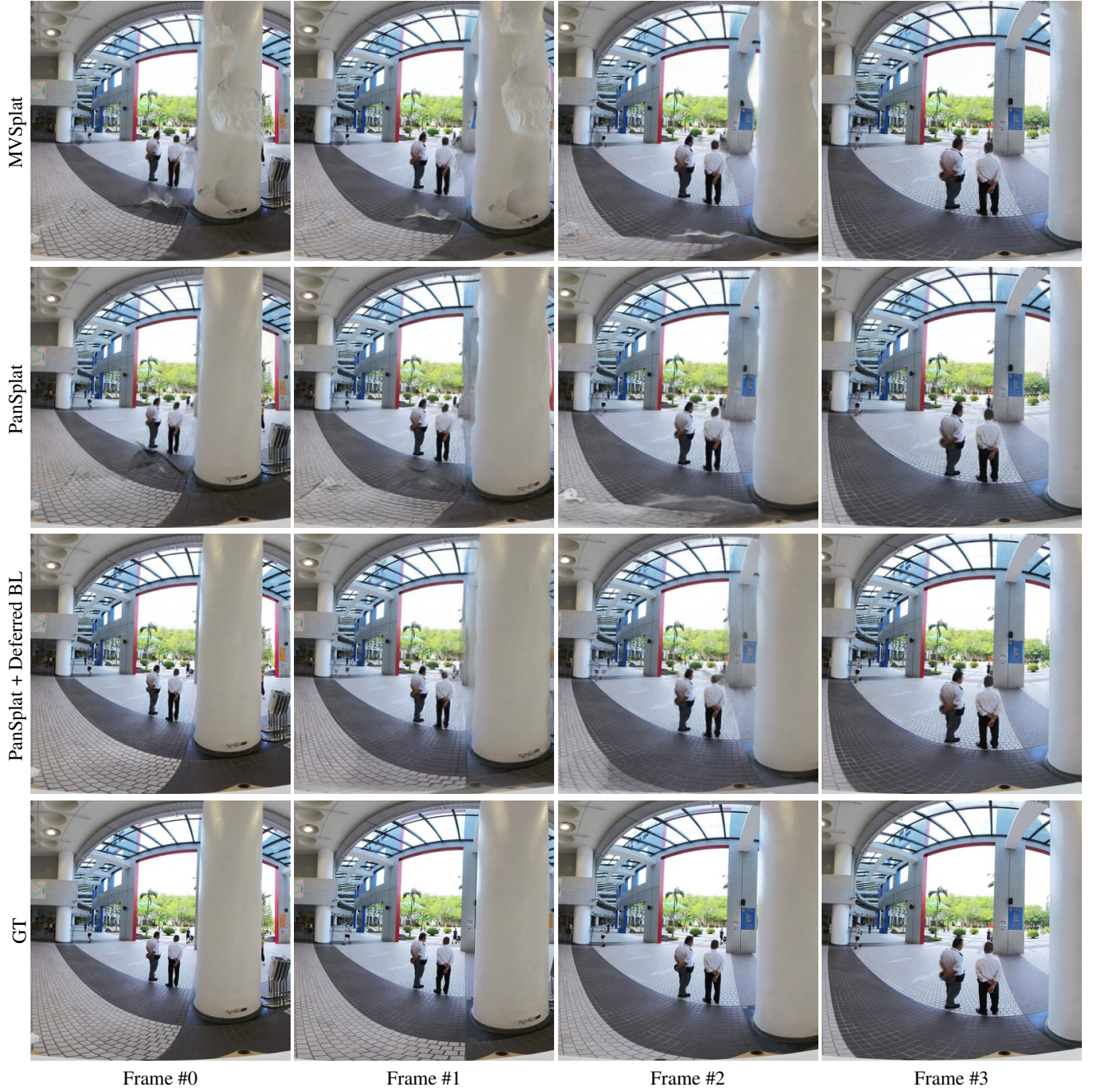


Figure F.2. **Qualitative comparisons on 360Loc dataset.** We show zoomed-in regions of the generated images by MV/Splat and PanSplat, with (PanSplat + Deferred BL) and without (PanSplat) deferred blending, compared to the ground truth (GT). The different columns represent different frames in the sequence, where Frame #0 and Frame #3 of GT are input views. We render the images across all four views to visualize different frame distances.

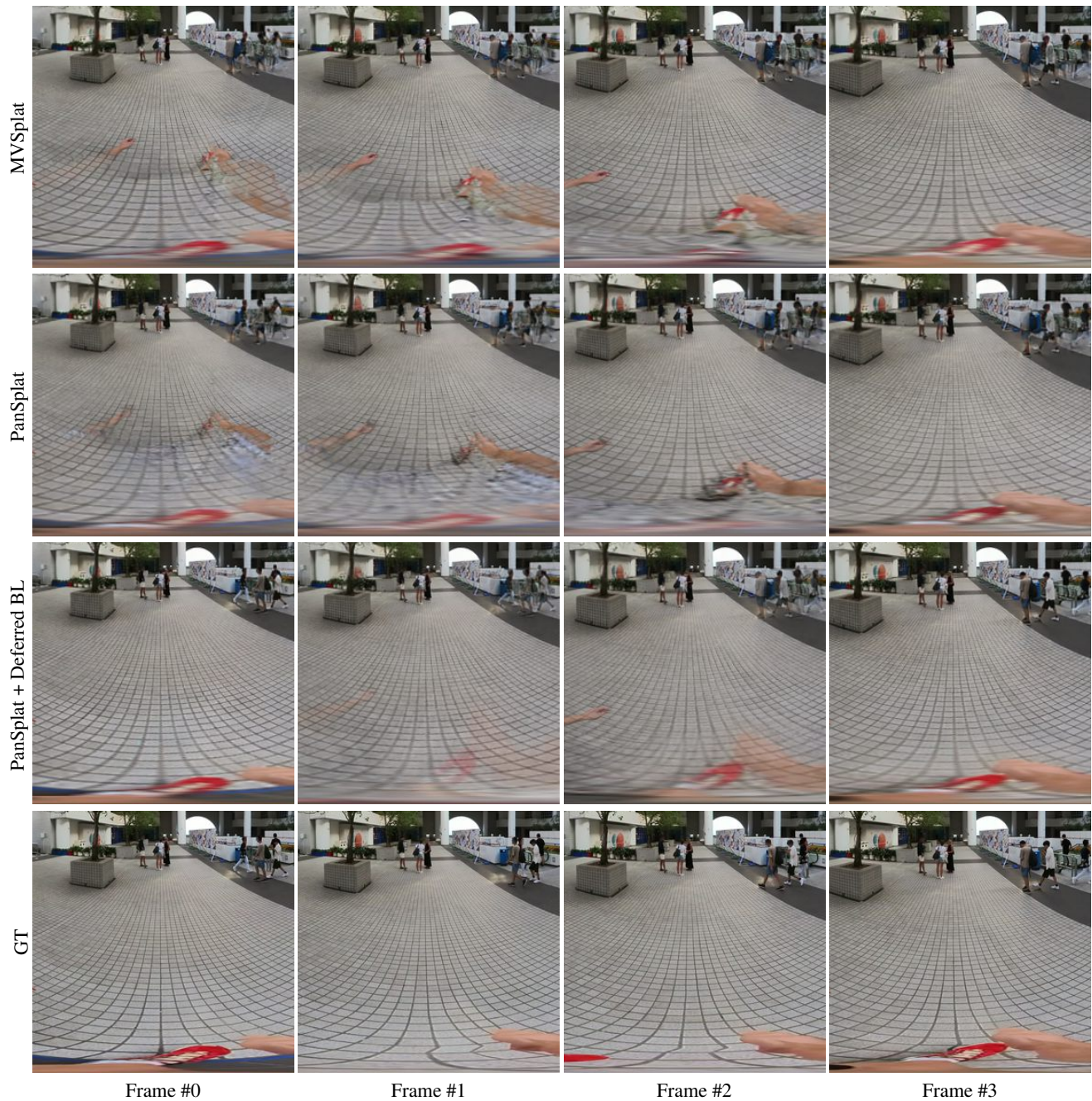


Figure F.3. **Qualitative comparisons on 360Loc dataset.** We show zoomed-in regions of the generated images by MV/Splat and PanSplat, with (PanSplat + Deferred BL) and without (PanSplat) deferred blending, compared to the ground truth (GT). The different columns represent different frames in the sequence, where Frame #0 and Frame #3 of GT are input views. We render the images across all four views to visualize different frame distances.

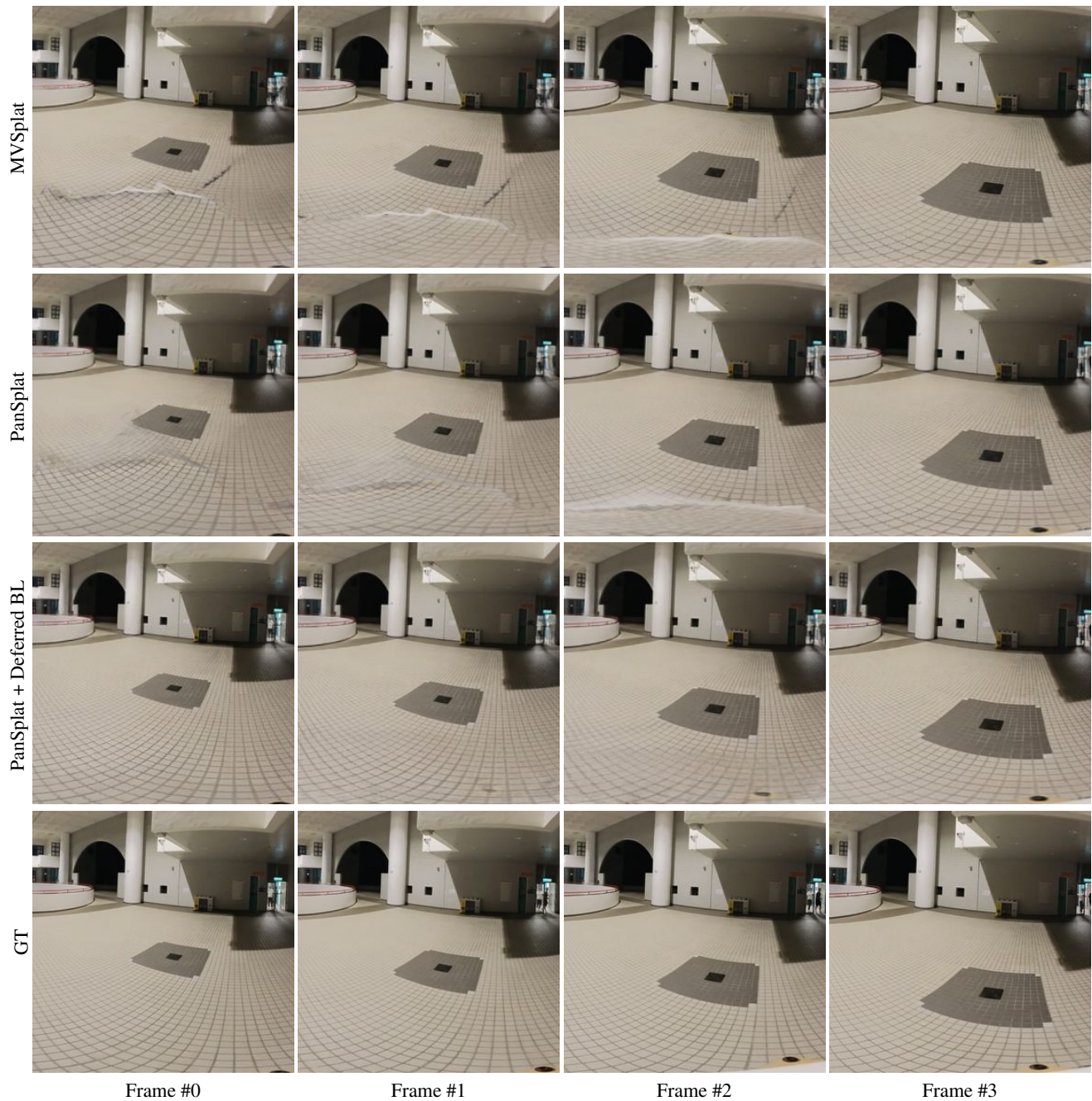


Figure F.4. **Qualitative comparisons on 360Loc dataset.** We show zoomed-in regions of the generated images by MV-Splat and PanSplat, with (PanSplat + Deferred BL) and without (PanSplat) deferred blending, compared to the ground truth (GT). The different columns represent different frames in the sequence, where Frame #0 and Frame #3 of GT are input views. We render the images across all four views to visualize different frame distances.

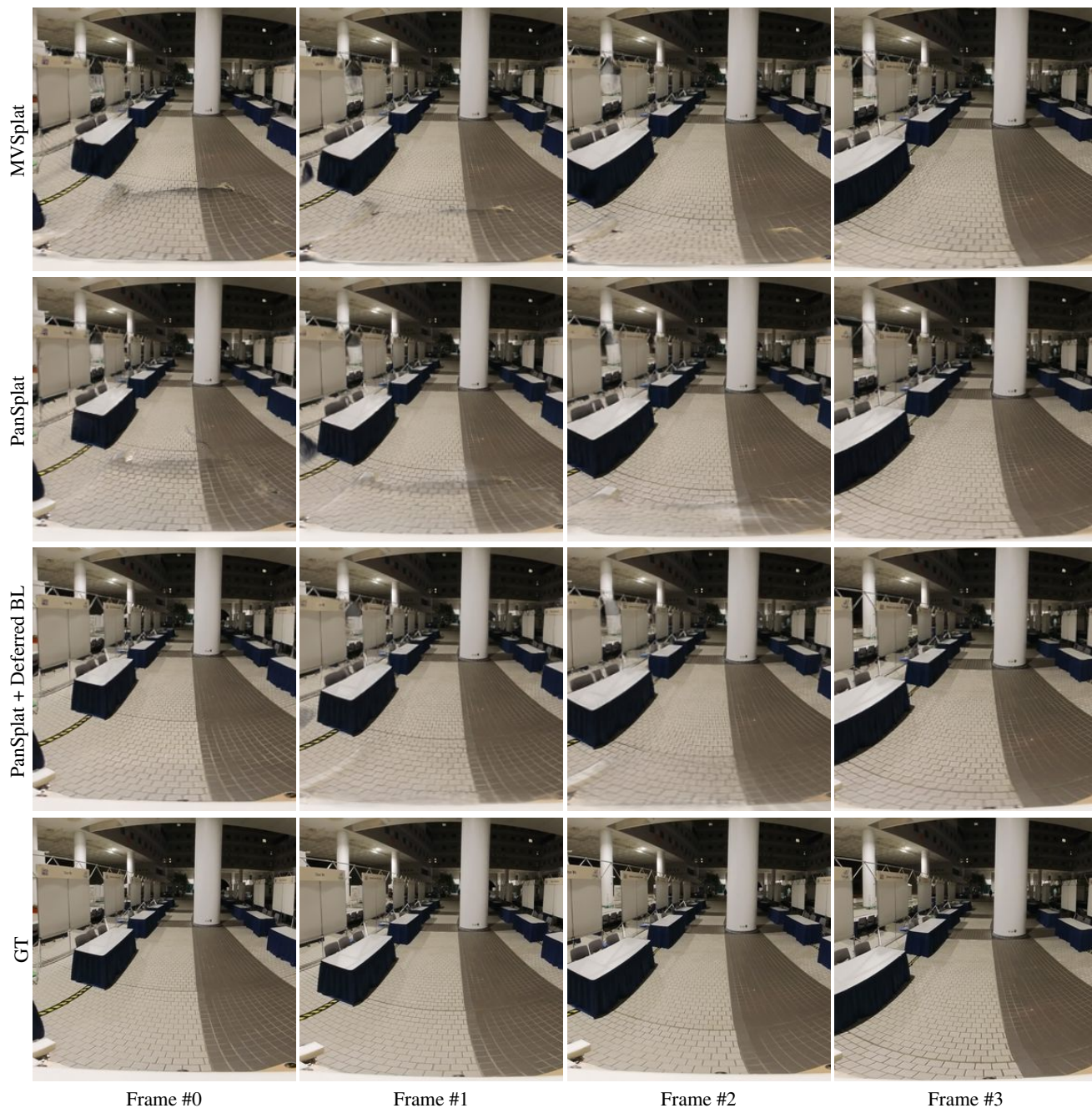


Figure F.5. **Qualitative comparisons on 360Loc dataset.** We show zoomed-in regions of the generated images by MV/Splat and PanSplat, with (PanSplat + Deferred BL) and without (PanSplat) deferred blending, compared to the ground truth (GT). The different columns represent different frames in the sequence, where Frame #0 and Frame #3 of GT are input views. We render the images across all four views to visualize different frame distances.

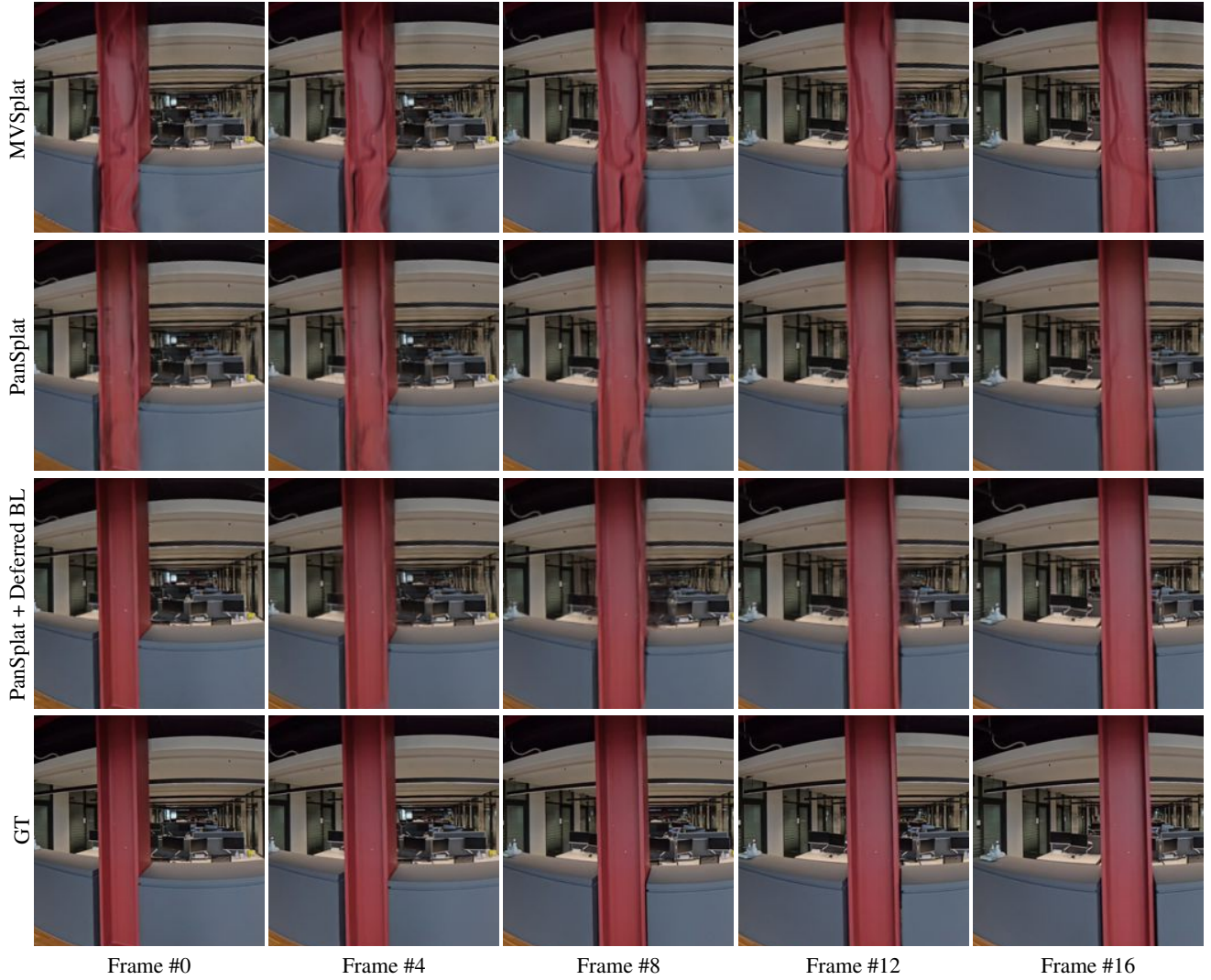


Figure F.6. **Qualitative comparisons on Insta360 dataset.** We show zoomed-in regions of the generated images by MV+Splat and PanSplat, with (PanSplat + Deferred BL) and without (PanSplat) deferred blending, compared to the ground truth (GT). The different columns represent different frames in the sequence, where Frame #0 and Frame #16 of GT are input views. We render the images across five evenly-spaced intermediate views to visualize different frame distances.

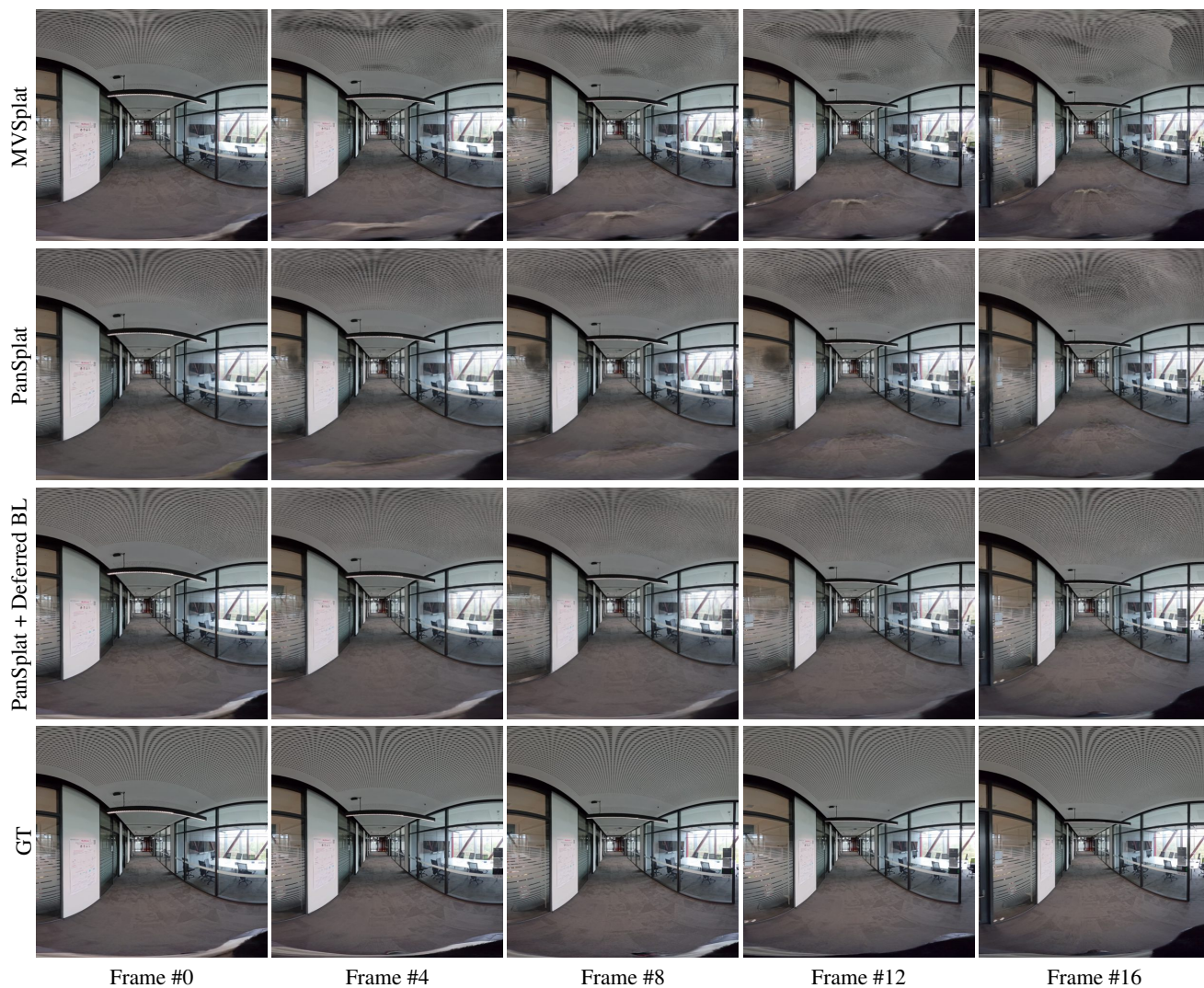


Figure F.7. **Qualitative comparisons on Insta360 dataset.** We show zoomed-in regions of the generated images by MV-Splat and PanSplat, with (PanSplat + Deferred BL) and without (PanSplat) deferred blending, compared to the ground truth (GT). The different columns represent different frames in the sequence, where Frame #0 and Frame #16 of GT are input views. We render the images across five evenly-spaced intermediate views to visualize different frame distances.

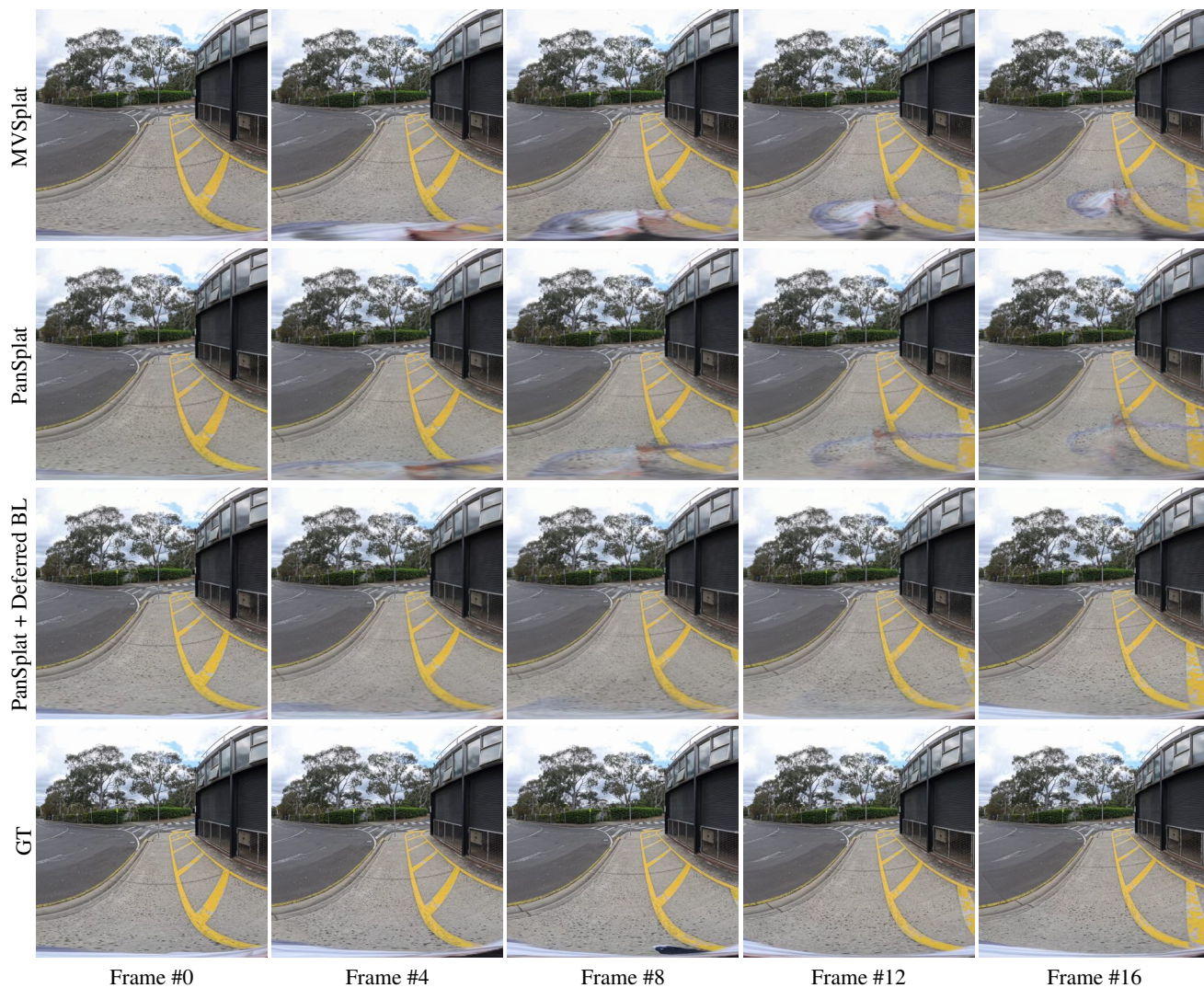


Figure F.8. **Qualitative comparisons on Insta360 dataset.** We show zoomed-in regions of the generated images by MVSplat and PanSplat, with (PanSplat + Deferred BL) and without (PanSplat) deferred blending, compared to the ground truth (GT). The different columns represent different frames in the sequence, where Frame #0 and Frame #16 of GT are input views. We render the images across five evenly-spaced intermediate views to visualize different frame distances.

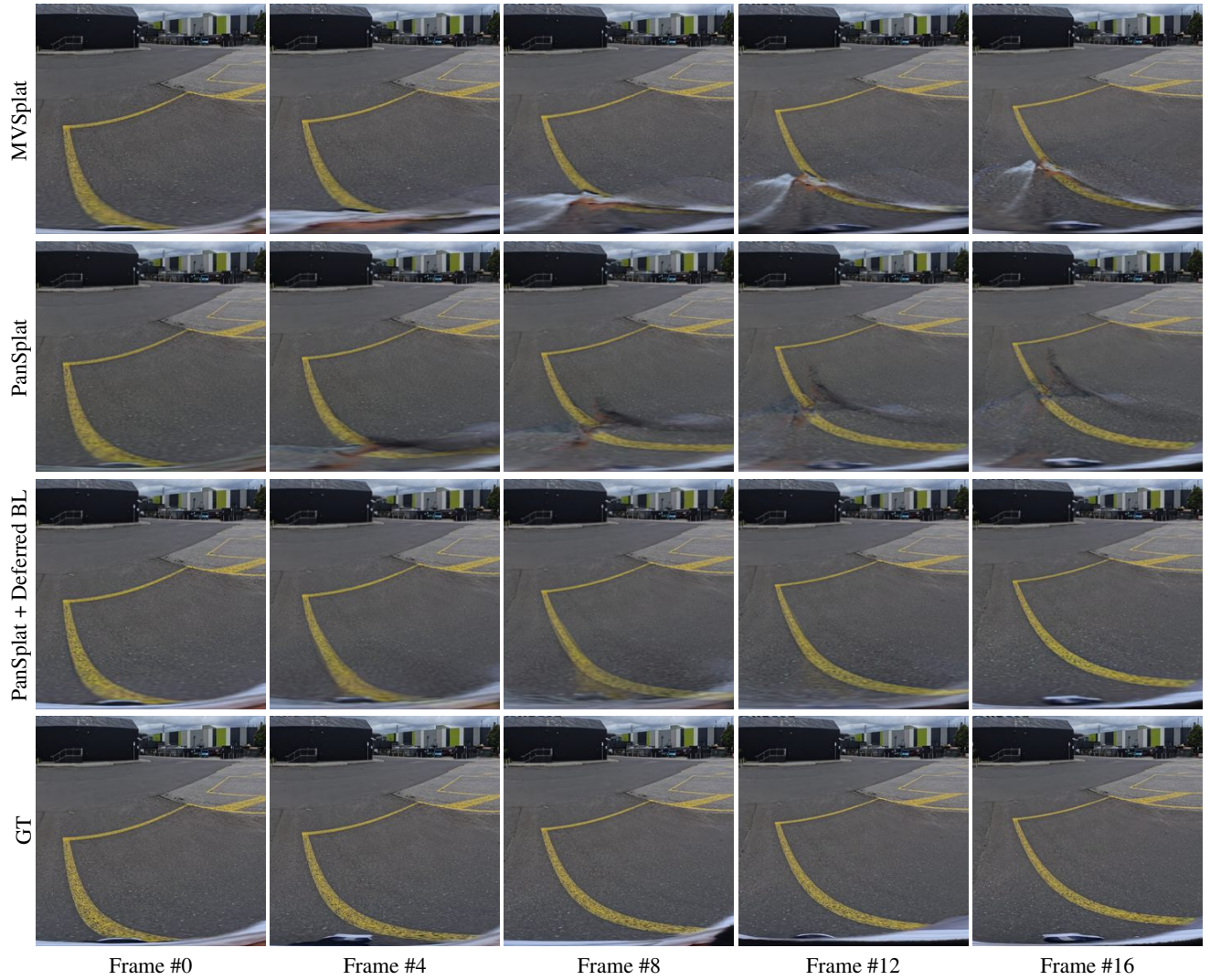


Figure F.9. **Qualitative comparisons on Insta360 dataset.** We show zoomed-in regions of the generated images by MVSplat and PanSplat, with (PanSplat + Deferred BL) and without (PanSplat) deferred blending, compared to the ground truth (GT). The different columns represent different frames in the sequence, where Frame #0 and Frame #16 of GT are input views. We render the images across five evenly-spaced intermediate views to visualize different frame distances.