TokenSplat: Token-aligned 3D Gaussian Splatting for Feed-forward Pose-free Reconstruction

Supplementary Material

A1. More Implementation Details of Experiments

More Training Details. Our model is implemented using PyTorch, and all models are trained on a single A800 GPU. Comparable results can also be achieved on a single A6000 or A100 GPU by adjusting to a smaller batch size and increasing the number of training iterations. Following strategies similar to those in [44] and [23], we train with a batch size of 16, a learning rate of 2×10^{-5} for the backbone, and a learning rate of 2×10^{-4} for the other components for 30,000 iterations under the 2-view settings. Subsequently, for multi-view scenarios, we fine-tune the model for 100,000 iterations with a backbone learning rate of 4×10^{-6}, a learning rate of 4×10^{-5} for the other components, and a batch size of 1. For the 28-view test on ScanNet, we use the model trained under the 10-view setting to evaluate the model's ability to generalize to a larger number of views. All other models, except for AnySplat, adopt a similar setup.

More Details of MVSplat and FreeSplat Results. All settings follow those specified in the official FreeSplat repository. For the 28-view results of FreeSplat on ScanNet, we use the model trained with the "fvt" setting (with the maximum number of views during training still set to 10), as this configuration offers better scalability and leads to higher performance for FreeSplat.

More Details of NoPoSplat Results. We use the official 2-view checkpoint from the repository and further fine-tune NoPoSplat with the exact same settings as our method to ensure a fair comparison. Additionally, all methods use the same view alignment protocol as NoPoSplat for evaluation to ensure complete fairness.

More Details of SPFSPlat Results. According to the official training configuration provided for SPFSplat, 200,000 training iterations are required. We use the official 2-view checkpoint from the repository and further fine-tune them for 200,000 iterations on the multi-view setting to ensure sufficient training of SPFSplat.

More Details of VicaSplat Results. For VicaSplat, the official code specifies the training requirements for each step on RE10K, and we strictly follow the training and fine-tuning procedures provided by the official code. For training on ScanNet, we follow our training and fine-tuning settings, and, as specified by the official guidelines, we perform point distillation before training.

More Details of AnySplat Results. Since the training dataset of AnySplat is approximately four times larger than the number of scenes in our RE10K training set and 95 times larger than the number of images in ScanNet, we report both the zero-shot results of the generic model provided by AnySplat (denoted as AnySplat), as well as the results obtained by training on RE10K and ScanNet following the official instructions (denoted as AnySplat$*$). Since AnySplat itself is trained on images with a resolution of 448, using different input resolutions leads to severe artifacts. Therefore, we use 448-resolution inputs for zero-shot testing with AnySplat. It is worth noting that the input image content is kept exactly the same as in other models, with only the resolution being different. For novel view synthesis tests, we render the 3DGS outputs of the models at a resolution of 256×256, ensuring that all models are evaluated at the same resolution. (Rendering at 448 resolution and then resampling to 256 yields even worse results.) Moreover, since AnySplat uses VGGT as its backbone and only supports inputs at 14× resolution, we train AnySplat$*$ at a resolution of 266×266, and adopt a testing strategy similar to that of AnySplat.

Results show that while the visual quality of AnySplat is acceptable, its reconstructed scenes exhibit some distortions and deformations. This distortion is not easily noticeable when viewing AnySplat results in isolation, but becomes much more apparent when test views are aligned with ground-truth views for evaluation (following the protocol in NoPoSplat). In many scenes, such distortions are clearly observed after this alignment. Even after fine-tuning on the corresponding scenes, achieving improvements and reaching the performance reported by the official implementation, there remains a significant gap compared to our method.

A2. More Implementation Details of Modulation in Asymmetric Dual-Flow Decoder

Inspired by the Adaptive Layer Normalization in DiT, we adopt a similar modulation strategy. Specifically, an MLP predicts three modulation parameters, *i.e. scale*, *shift*, and *gate*, from the camera token at each step. Since the camera tokens are continuously updated within the decoder blocks, after self-attention layer, a separate MLP predicts a new set of modulation parameters from the updated camera token. The normalization parameters (scale and shift) and an additional scaling parameter (gate) are applied as follows: before attention, we perform $\hat{t}_i^{\mathbf{I}} \leftarrow \hat{t}_i^{\mathbf{I}} \times (1 + \text{scale}) + \text{shift}$; after attention, we update as $\hat{t}_i^{\mathbf{I}} \leftarrow (1 + \text{gate}) \times \hat{t}_i^{\mathbf{I}}$.

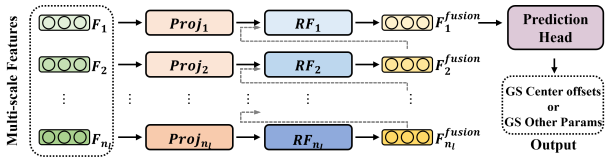

Figure A1. Structure of our Gaussian Prediction Head.

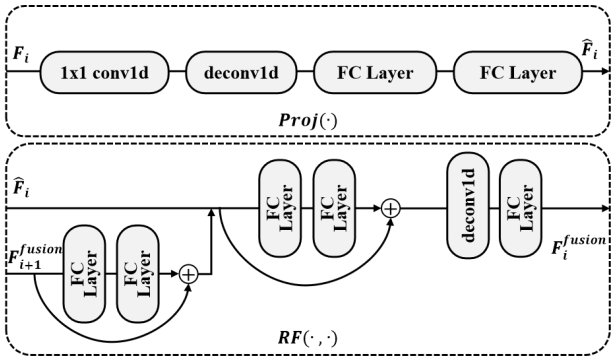

Figure A2. Network details in our Gaussian Prediction Head.

A3. More Implementation Details of Gaussian Prediction Heads

In this section, we further elaborate on the detailed design of our Gaussian Prediction heads, as illustrated in Fig. A1. Our Gaussian Prediction heads take the multi-scale features from the fused token decoder outputs as input. In our implementation, we follow the DPT head by selecting features from the 0-th, 6-th, 9-th, and 12-th layers of the decoder as inputs F_1, F_2, F_3, F_4 to the Gaussian Prediction head.

The multi-scale features are first individually processed by a projection module, denoted as $Proj(\cdot)$, as illustrated in the upper part of Fig. A2. This module consists of a 1×1 Conv1D layer for channel transformation, followed by a deconvolution layer for upsampling. Subsequently, two fully connected layers are used for linear mapping to get $\hat{F}_i$. Ultimately, all features are unified to the same number of channels, which is set to 256 in our implementation.

Subsequently, we perform deep-to-shallow fusion from feature $\hat{F}_4$ to feature $\hat{F}_1$, as shown in the lower part of Fig. A2. For feature $\hat{F}_4$, we directly apply a residual module composed of two fully connected layers for further feature transformation. For shallower features $\hat{F}_i$, we first add the fused feature from the deeper layer F_{i+1}^{fusion} after passing it through a residual module consisting of two fully connected layers, and then apply another residual module with two fully connected layers for feature transformation. Finally, the output is upsampled using a deconvolution module and further processed by a fully connected layer to obtain F_i^{fusion}.

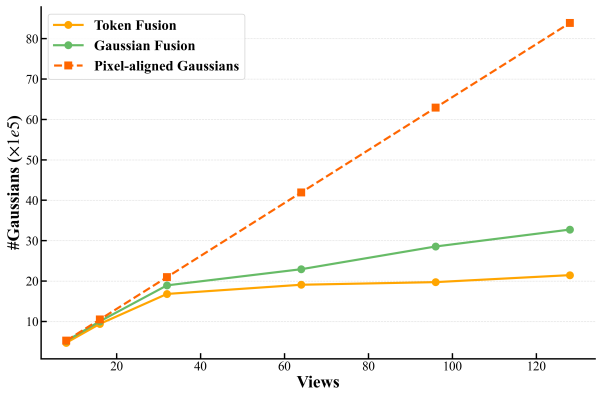

Figure A3. Number of Gaussians with increasing input views for pixel-aligned GS, Gaussian fusion methods, and our Token Fusion operation.

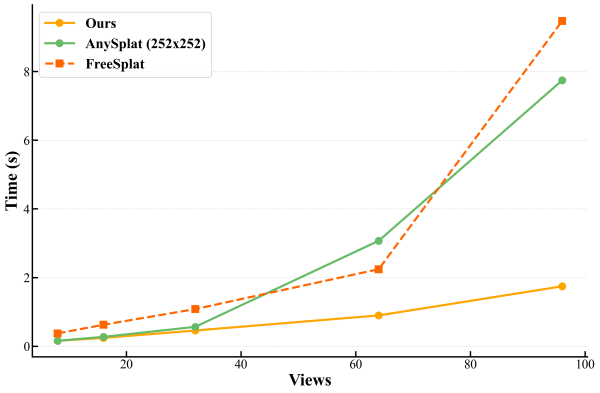

Figure A4. Inference time with increasing number of input views.

A4. More Implementation Details of Pose Heads

The pose head takes the camera token output from the Asymmetric Dual-Flow Decoder as input and consists of activation functions and fully connected layer. Specifically, we use a ReLU activation function and a fully connected layer with dimensions $(768, 7)$ in this work. The pose head outputs camera rotation (as a quaternion) and translation, which are then converted to unit dual quaternions for loss computation. Our pose head is designed with a simple structure, as the Asymmetric Dual-Flow Decoder effectively disentangles camera features, allowing accurate predictions with minimal complexity.

A5. More Experimental Analysis

A5.1. Efficiency

We investigate the impact of different fusion methods on the number of Gaussian primitives and show the result in

Figure A5. Visualization Results with increasing number of input views.

Fig. A3. As the number of input views increases, the number of pixel-aligned Gaussian primitives grows linearly. In contrast, using Gaussian fusion modules in 3D space directly or our Token Fusion operation helps stabilize the growth of Gaussians. It can be observed that as the number of views increases, our Token Fusion operation leads to a much slower growth in the number of Gaussians, making the count more stable compared to Gaussian fusion. We investigate how the model's inference time changes as the number of input views increases and show the result in Fig. A4. As other models experience excessive growth in model size and memory usage with increasing views, we mainly compare with FreeSplat and AnySplat. Since AnySplat only supports up to $14\times$ resolution, we use a resolution of 252×252, while FreeSplat and ours use 256×256. Our method maintains more stable inference time as the number of views increases. This is because, unlike Gaussian fusion methods where computation in the Gaussian prediction stage remains proportional to the number of input images, our approach performs fusion first, fully decoupling the number of Gaussians from the number of images, resulting in more stable inference time.

A5.2. Visualization Results with Increasing Number of Views

We further demonstrate the impact of increasing the number of input views on novel view synthesis. Starting with the first and last views, we gradually add more intermediate views. As shown in Fig. A5, additional input views progressively enhance scene completeness and visual details, especially in challenging regions such as railings.

A6. More Visualization Results

We present additional qualitative comparisons with baselines on the RE10K and ScanNet datasets in Fig. A6 to Fig. A10, further demonstrating the effectiveness and improvements of our method. Our approach achieves stable and superior performance across varying numbers of input images and diverse input data.

A7. Video Demo

The project webpage includes comparison videos between our approach and state-of-the-art methods. Whether in rendered videos or when observing the entire scene from a distance, our method consistently exhibits more complete and coherent structures, whereas existing methods suffer from blurring or defocus-like artifacts, and some even exhibit severe structural issues. For example, when running SPFSplat and VicaSplat with more input views than used during training, we observe significant scene compression at the boundaries, making it difficult to effectively extend outward. FreeSplat projects Gaussians into 3D space using depth maps, resulting in a large number of floating artifacts. In the left-side close-up rendering video of AnySplat, noticeable cracks caused by unsuccessful Gaussian fusion can be observed near the mirror at the beginning, and multiple issues arising from 3D Gaussian fusion are also clearly visible in the long-range scene. Furthermore, our method exhibits remarkable generalization ability, as evidenced by the reconstruction results on scenes from different datasets and **in-the-wild data casually captured with mobile phones**.

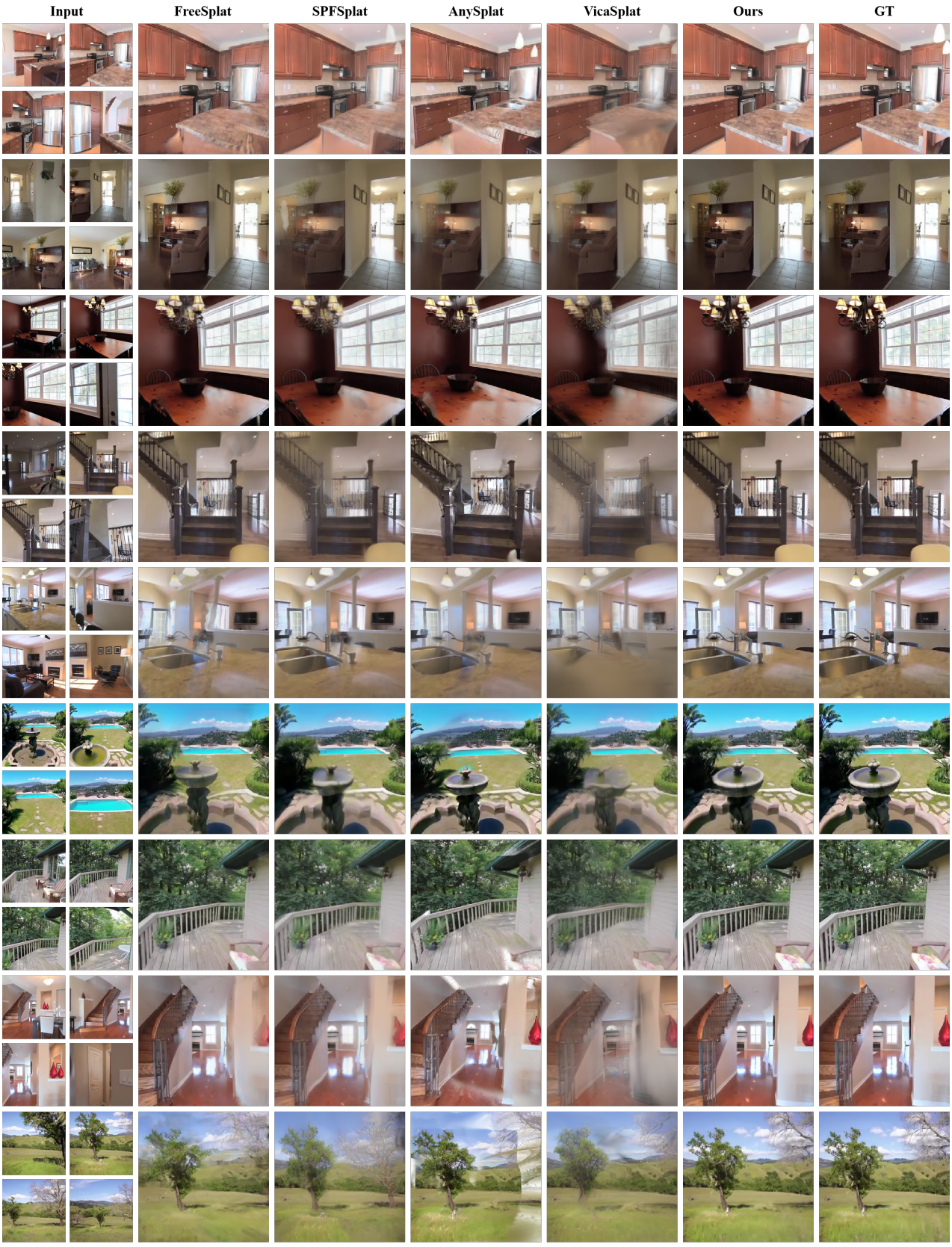

Figure A6. More qualitative comparisons on RE10K with 4 input images.

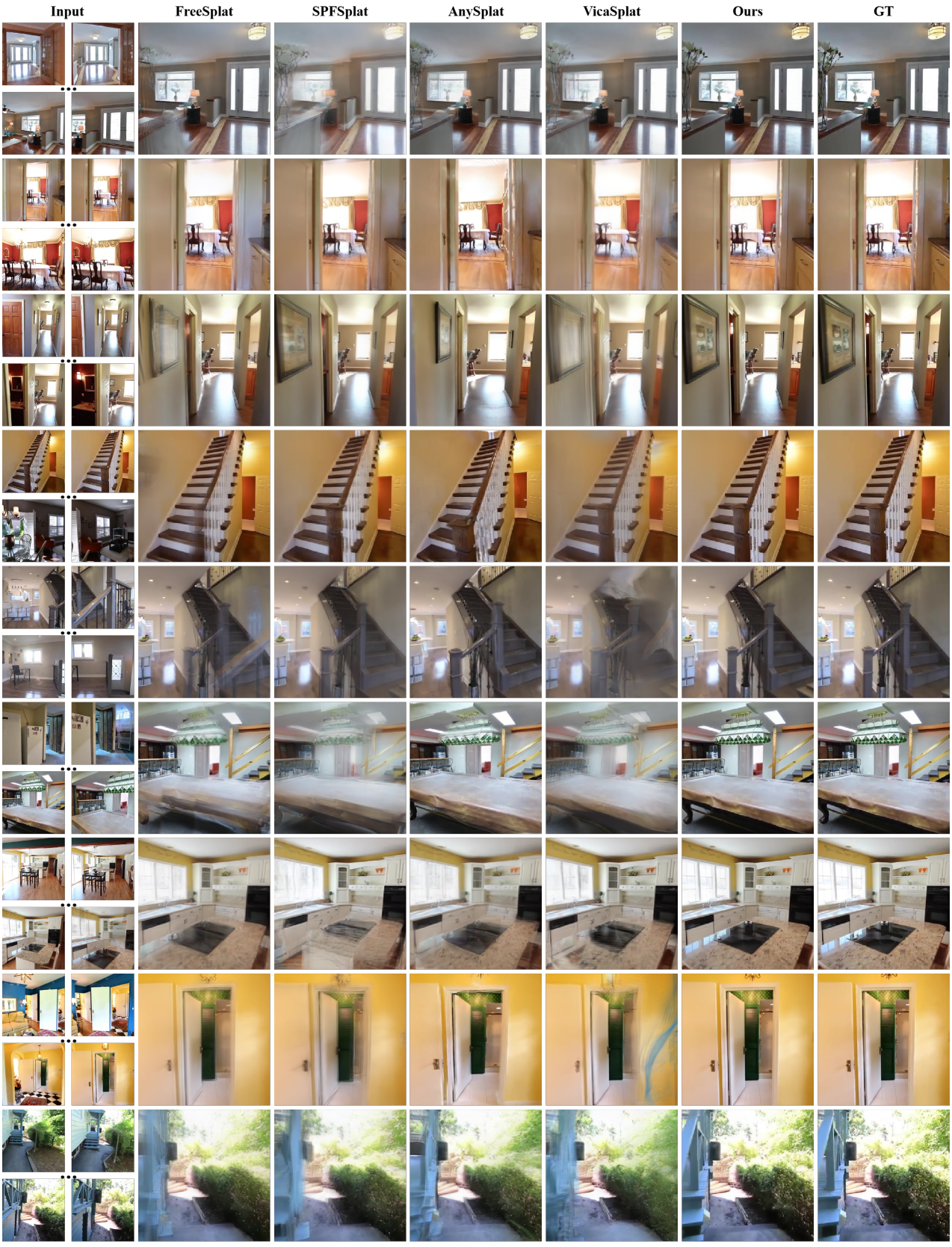

Figure A7. More qualitative comparisons on RE10K with 8 input images.

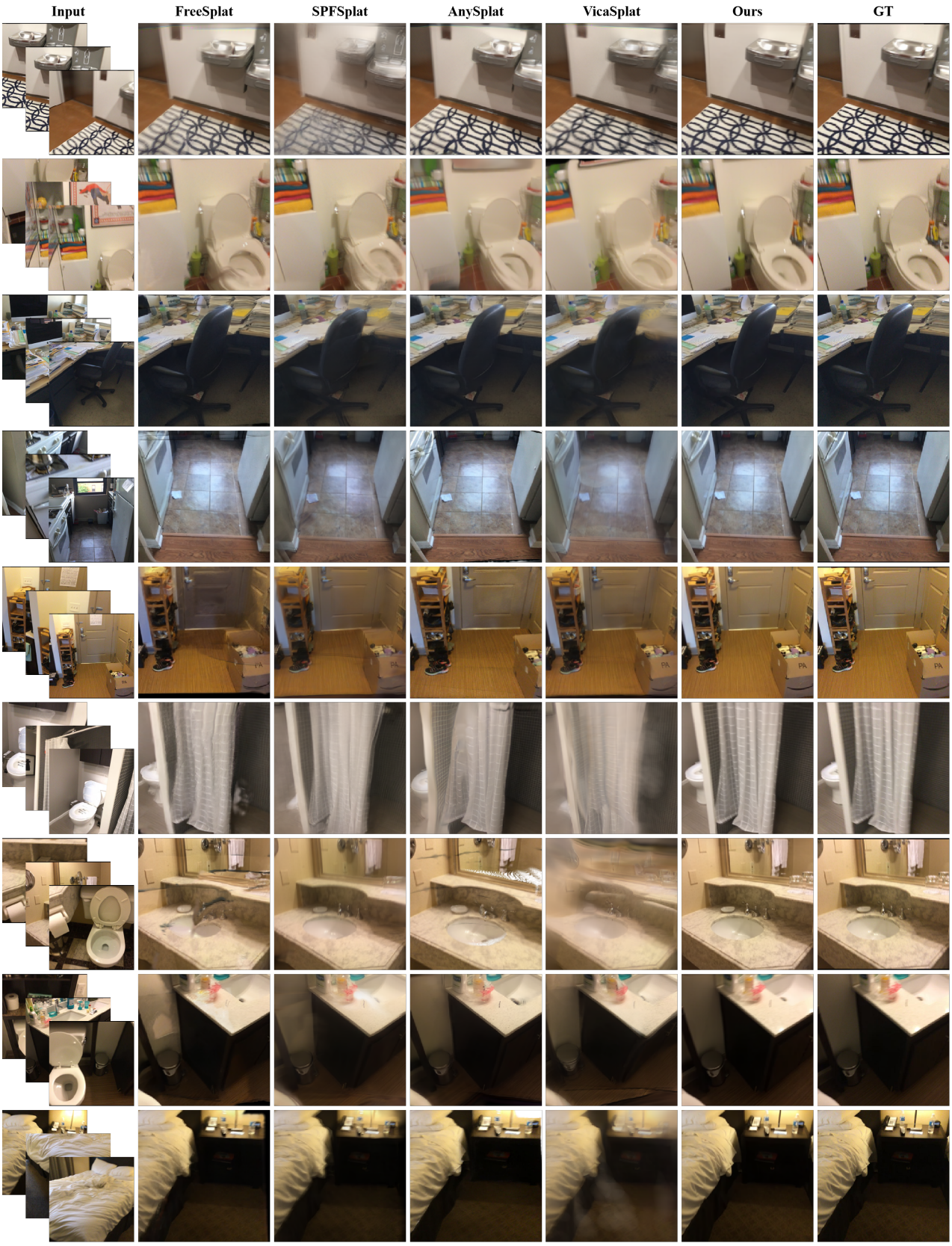

Figure A8. More qualitative comparisons on ScanNet with 3 input images.

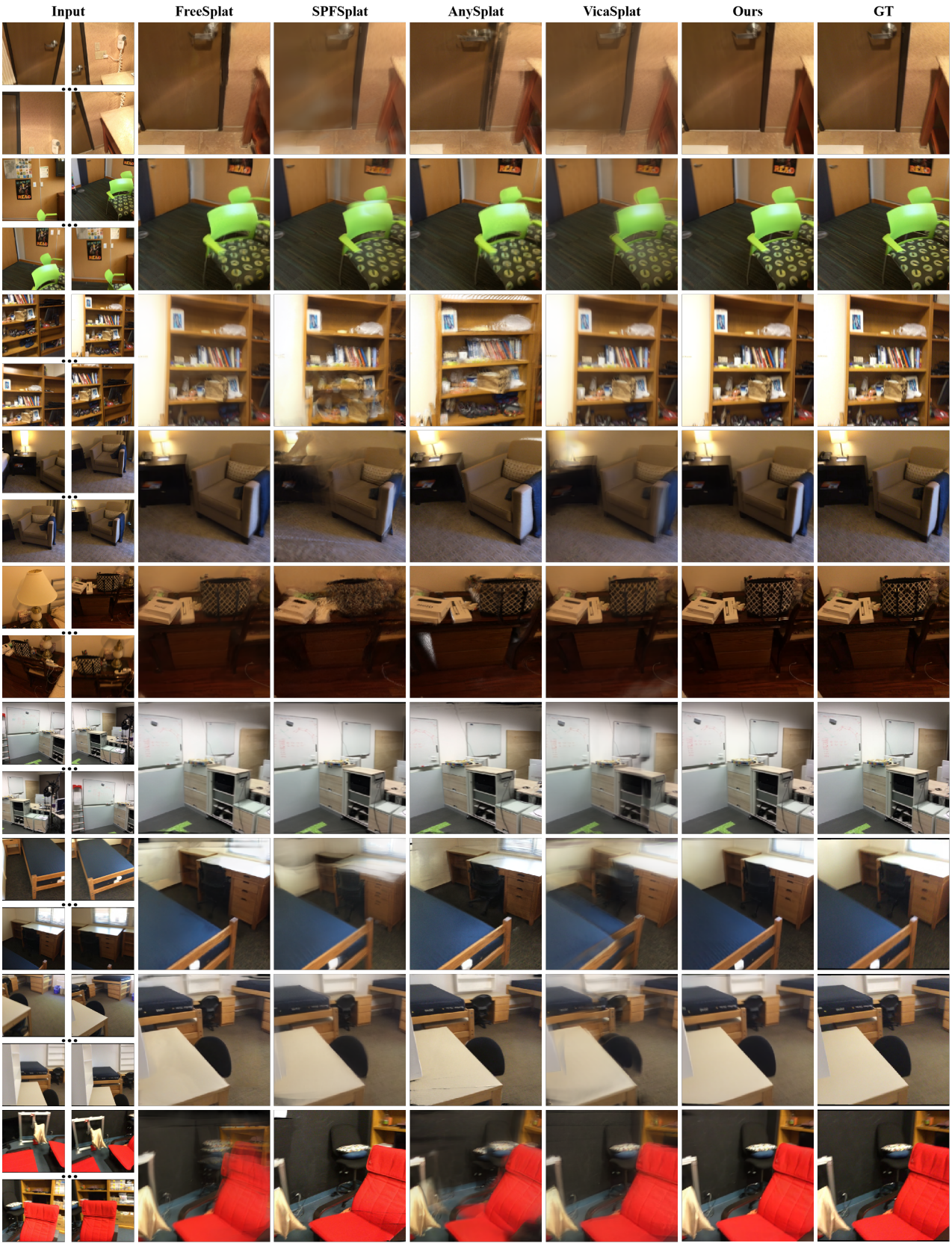

Figure A9. More qualitative comparisons on ScanNet with 10 input images.

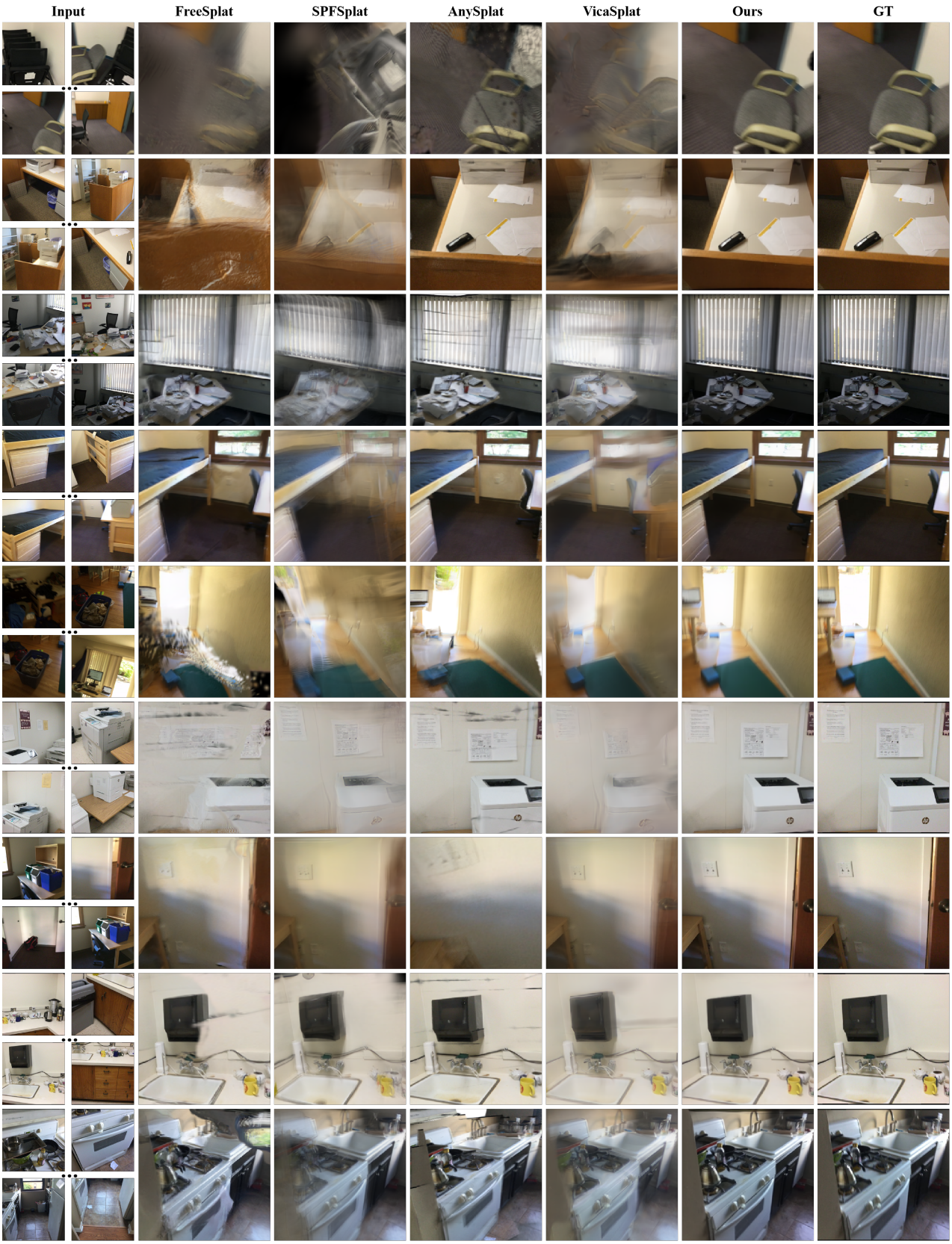

Figure A10. More qualitative comparisons on ScanNet with 28 input images.