Appendix

We provide additional qualitative and quantitative results and discuss training and evaluation details.

A. Background

A.1. Diffusion Models

Diffusion Models (DMs) transform data to noise by learning to sequentially denoise their inputs $\mathbf{z_T} \sim \mathcal{N}(\mathbf{0}, \sigma_T)$ [18, 54, 55]. This approximates the reverse ODE to a stochastic forward process which transforms the data distribution $p_{\text{data}}(\mathbf{z}_0)$ to an approximately Gaussian distribution $p(\mathbf{z}; \sigma_T)$ by adding i.i.d. Gaussian noise with sufficiently large σ_T and can be written as

$$d\mathbf{z} = -\dot{\sigma(t)}\sigma(t)\nabla_z \log p(\mathbf{x}; \sigma(t))dt. \tag{8}$$

DM training approximates the score function $\nabla_z \log p(\mathbf{x}; \sigma(t))$ with a neural network $\mathbf{s}_\theta(\mathbf{z}; \sigma)$ with parameters θ. In practice, the network can be parameterized as $\mathbf{s}_\theta(\mathbf{z}; \sigma) = (D_\theta(\mathbf{z}; \sigma) - \mathbf{z})/\sigma^2$ and trained via denoising score matching

$$\mathbb{E}_{(z_0,y)\sim p_{\text{data}}(z_0,y),(\sigma,n)\sim p(\sigma,n)} \left[\lambda_\sigma \left\| D_\theta(\mathbf{z}_0 + \mathbf{n}; \sigma, \mathbf{y}) - \mathbf{z}_0 \right\|_2^2\right] \tag{9}$$

where $p(\sigma, \mathbf{n}) = p(\sigma)\mathcal{N}(\mathbf{n}; \mathbf{0}, \sigma^2)$, $p(\sigma)$ is a distribution over noise levels σ, $\lambda_\sigma : \mathbb{R}^+ \to \mathbb{R}^+$ is a weighting function, and y is an arbitrary conditioning signal. We follow the EDM-preconditioning framework [23] and use

$$D_\theta(\mathbf{z}; \sigma) = c_{\text{skip}}(\sigma)\mathbf{z} + c_{\text{out}}(\sigma)F_\theta(c_{\text{in}}(\sigma)\mathbf{z}; c_{\text{noise}}(\sigma)),$$

where F_θ is the trained neural network and c_{skip}, c_{out}, c_{in}, and c_{noise} are scalar weights.

A.2. Gaussian Splatting

In their seminal work, Kerbl et al. [24] propose to represent a 3D scene as a set of scaled 3D Gaussian primitives $\{\mathbf{G}_k\}_{k=1}^K$ and render an image using volume splatting. Each 3D Gaussian $\mathbf{G}_k$ is parameterized by an opacity $\alpha_k \in [0, 1]$, color $\mathbf{c}_k$, a center (mean) $\mathbf{p}_k \in \mathbb{R}^{3\times 1}$ and a covariance matrix $\mathbf{\Sigma}_k \in \mathbb{R}^{3\times 3}$ defined in world space:

$$\mathbf{G}_k(\mathbf{x}) = e^{-\frac{1}{2}(\mathbf{x}-\mathbf{p}_k)^T\Sigma_k^{-1}(\mathbf{x}-\mathbf{p}_k)} \tag{10}$$

In practice, the covariance matrix is calculated from a predicted scaling vector $s \in \mathbb{R}^3$ and a rotation matrix $\mathbf{O} \in \mathbb{R}^{3\times 3}$ to constrain it to the space of valid covariance matrices, i.e.,

$$\mathbf{\Sigma}_k = \mathbf{O}_k\mathbf{s}_k\mathbf{s}_k^T\mathbf{O}_k^T. \tag{11}$$

The color c_k is parameterized with spherical harmonics to model view-dependent effects. To render this 3D representation from a given viewpoint with camera rotation $\mathbf{R} \in \mathbb{R}^{3\times 3}$ and translation $\mathbf{t} \in \mathbb{R}^3$, the Gaussians $\{G_k\}$ are first transformed into camera coordinates

$$\mathbf{p}'_k = \mathbf{R}\mathbf{p}_k + \mathbf{t}, \quad \mathbf{\Sigma}'_k = \mathbf{R}\mathbf{\Sigma}_k\mathbf{R}^T \tag{12}$$

and susequently projected to ray space using a local affine transformation

$$\mathbf{\Sigma}''_k = \mathbf{J}_k\mathbf{\Sigma}'_k\mathbf{J}_k^T, \tag{13}$$

where the Jacobian matrix $\mathbf{J}_k$ is an affine approximation to the projective transformation defined by the center of the 3D Gaussian $\mathbf{p}'_k$. The Gaussians are projected onto a plane by skipping the third row and column of $\mathbf{\Sigma}''_k$, yielding the 2D covariance matrix $\Sigma_{2D,k}$ of the projected 2D Gaussian $\mathbf{G}_{2D,k}$. The rendered color is obtained via alpha blending according to the primitive's depth order $1, \ldots, K$:

$$\mathbf{c}(x) = \sum_{k=1}^{K} \mathbf{c}_k\alpha_k\mathbf{G}_{2D,k}(x)\prod_{j=1}^{k-1}(1 - \alpha_j\mathbf{G}_{2D,j}(x)). \tag{14}$$

B. Implementation Details

Model architecture. Our epipolar transformer consists of 2 attention blocks with 4 heads each. Similar to [5], it samples 32 values per feature map on each pixel's epipolar line. As the transformer already operates on a low-resolution latent space, we do not apply any spatial downsampling. The depth predictor consists of 2 linear layers with ReLU and sigmoid activation that predict mean and variance of the per-pixel disparity. The disparity is further mapped to an opacity with 4 additional linear layers with ReLU activation, followed by a sigmoid activation. In parallel, the remaining Gaussian parameters, i.e., scale, rotation, color and feature values, as well as a per-pixel offset are predicted with a linear layer from the feature maps predicted by the epipolar transformer. For efficiency, we use a 32 channels for the feature values per Gaussian and omit view-dependent effects, i.e., predict rgb values instead of spherical harmonics.

The architecture of the 3D decoder consists of a 2D upsampler and architecturally similar layers to the aforementioned depth predictor and mapping to Gaussian parameters. The 2D upsampler consists of multiple blocks 2D convolutions with replication padding and nearest neighbor upsampling, resulting in a total of 1.5M parameters for the 3D decoder.

C. Baselines

PNVS We use the official implementation of the authors https://github.com/YorkUCVIL/Photoconsistent-NVS.git and the provided checkpoint on RealEstate10K.

MultiDiff We run code and a checkpoint for RealEstate10K, both provided by the authors, using our evaluation splits.

CameraCtrl We use the official implementation of the authors https://github.com/hehao13/CameraCtrl.git and the provided checkpoint on RealEstate10K. Since the original implementation is trained to generate 14 frames, we pad the camera trajectory by duplicating frames and subsequently subsampling the generated images.

ViewCrafter We use the official evaluation scripts provided by the authors https://github.com/Drexubery/ViewCrafter.git. Since the original implementation is trained to generate 25 frames, we pad the camera trajectory by duplicating frames and subsequently subsampling the generated images.

PixelSplat We run the official RealEstate10K-checkpoint and inference implementation https://github.com/dcharatan/pixelsplat.git using our evaluation splits. We remark that the results on our evaluation split are lower than the originally reported numbers in [5] on the full testset. Since evaluating generative methods on such a large quantity of scenes is computationally expensive and slow, we decided to only report numbers on 128 randomly sampled scenes from the testset, following [68]. We verified that we obtain the originally reported performance when using their original evaluation split to ensure we run the method correctly and note that another work also measured lower performance for PixelSplat on a slightly different evaluation split [69].

LatentSplat We evaluate the official checkpoint on RealEstate10K using the official inference implementation https://github.com/Chrixtar/latentsplat.git together with our evaluation splits.

4DiM While we designed our evaluation setting for RealEstate10K approximately similar to 4DiM [68], a quantitative comparison is difficult because no official code or evaluation splits are available. We observe that reconstruction quality on RealEstate10K can vary significantly between scenes, as indicated by a large standard deviation for both reconstruction metrics: 19.2 ± 4.2 for PSNR and 0.277 ± 0.113 for LPIPS in our single-view setting with 128 randomly sampled scenes. For reference, 4DiM reports a PSNR of 18.09 and LPIPS of 0.263 for their best model. We also measure a slightly lower TSED when using ground truth data: 0.993 whereas 4DiM obtains 1.000 on their evaluation split. Note that a TSED below 1.0 can indeed happen on ground truth data, as poses in the data are noisy and do not achieve a perfect score [68]. Considering the results for our approach $0.992(0.993)$ and $0.997(1.000)$ for 4DiM, both methods saturate the metric wrt. the corresponding evaluation split. Lastly, we remark that FID and FVD depend strongly on the number of real samples that were used for comparison, as well as any preprocessing of the data. 4DiM does not provide these evaluation details and their numbers for FID and FVD are not directly comparable with our results.

D. Experimental Setting

D.1. ScanNet++

The camera trajectories for ScanNet++ follow a scan-pattern and viewpoints often change rapidly with large camera motion. When evaluating our method, we hence ensure that the sampled target views have an average overlap of at least 50% with the reference views. Specifically, we sample one or two reference views randomly and then compute the overlap for each view in the scene with the reference views using the provided ground truth depth. The overlap score of each view is computed as the average score over all reference frames. We select the views with the largest overlap as target views and discard cases in which any of the selected target views has less than 50% average overlap.

D.2. Metrics

For FID, we take all generated views and compare their distribution to the same number of views for 20K scenes of the training set for RealEstate10K and 171 scenes for ScanNet++. For FVD, we use all generated views and the same number of views from 2048 RealEstate10K scenes and 171 scenes from ScanNet++. Since the feature extractor for FVD requires a minimum number of 9 frames, we use reflection padding to pad real and generated sequences.
We use a guidance scale of 2.0 for all our results.

E. Additional Results

E.1. Teaser Images

The images in Fig. 1 of the main paper show the generated splats with a subsequent per-scene optimization, running the default Splatfacto method of [59] with 5,000 iterations. To encourage a fewer splats, we initialize the scale based on the average distance to the three nearest neighbors instead of directly using the predicted scale. We also visualize the generated splats in feature space, i.e., the 3D representation generated by our model. Since the features are high-dimensional, we show the first three principal components of feature space. We use the same visualization for the images shown in this appendix.

E.2. Additional Quantitative Results

We provide additional quantitative results for 3D scene synthesis using two reference images in Table 5. We further conduct an additional ablations study, for which we only train a 3D decoder on top of a frozen diffusion model, i.e., Ours-No3D. As shown in Table 6, this approach consistently performs worse than GGS and GGS with 3D decoder,

	RE10K		ScanNet++	
	FID↓	FVD↓	FID↓	FVD↓
ViewCrafter	70.8	784.6	119.8	553.3
Ours-No3D	57.4	678.7	149.2	635.5
GGS (Ours)+depth	57.4	**440.6**	127.6	523.8
GGS (Ours)+depth+FT	**47.6**	468.4	**119.3**	**513.7**

Table 5. **3D Scene Synthesis:** We report FID and FVD for rendered views between the training images at image resolution 576x320 pixels.

	Interpolation			Extrapolation		
	PSNR↑	LPIPS↓	TSED↑	FID↓	FVD↓	TSED↑
GGS	22.5	0.214	0.977	51.3	317.0	0.972
No $\mathcal{L}_{\text{nv,LR}}$	19.6	0.294	0.868	65.8	320.8	0.865
No f_v, x_0-prediction	14.9	0.611	0.753	239.4	1131.6	0.821
GGS +depth	23.0	0.204	0.973	51.0	336.3	0.963
f_ϕ: Discrete ϕ	22.0	0.211	0.960	53.7	348.9	0.956
$\mathcal{D}_{3D}$: convolutional	21.2	0.287	0.995	66.9	317.9	0.990
$\mathcal{D}_{3D}$: transformer	21.3	0.342	0.996	92.2	343.5	0.990
$\mathcal{D}_{3D}$: frozen backbone	19.5	0.312	0.942	70.1	320.7	0.934

Table 6. **Ablation Studies:** We investigate the effectiveness of our design choices on RealEstate10K using two reference images.

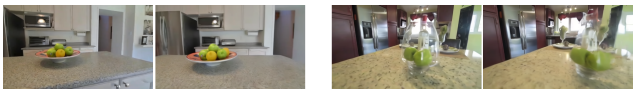

Figure 7. **View extrapolation:** Left: GGS can extrapolate the input view, e.g. the fridge. Right: Failure case for challenging transparent object close to the camera.

Figure 8. **Long-Range Autoregressive Synthesis** Generated views of GGS between the 12 reference images.

corroborating our design choice to include the 3D representation directly in the diffusion model to synthesize consistent results.

E.3. Additional Qualitative Results

We show additional baseline comparisons for synthesis from a single view and two views in Fig. 9 and Fig. 10, respectively. We showcase additional view extrapolations in Fig. 7, where for the first example the fridge on the left side is generated, and one failure case on the right, a broken glass vase. Fig. 11 provides more generated 3D scenes from a single image, using our GGS model and Fig. 12 depicts additional comparisons to ViewCrafter [76] for 3D scene synthesis. Lastly, we include more autoregressive scene synthesis results in Fig. 13 and Fig. 8. We extend the autoregressive synthesis to 12 views in Fig. 8 and find that GGS still performs well in this setting. For more views, since GGS generates per-pixel splats, the number of splats becomes computationally too expensive. A possible solution would be to include intermediate pruning steps to reduce the number of total splats.

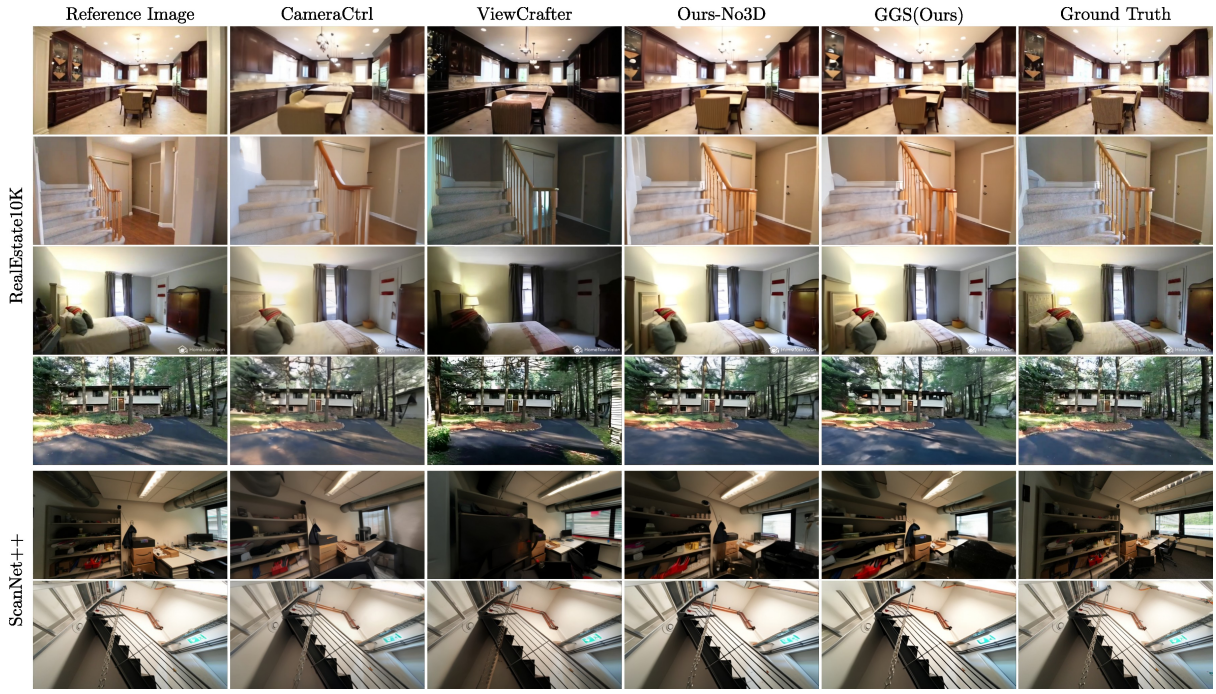

Figure 9. **Baseline Comparison Given One Reference Image:** We show results for the strongest baselines CameraCtrl [15] and ViewCrafter[76] together with our approach without (Ours-No3D) and with 3D representation (GGS). Best viewed zoomed in.

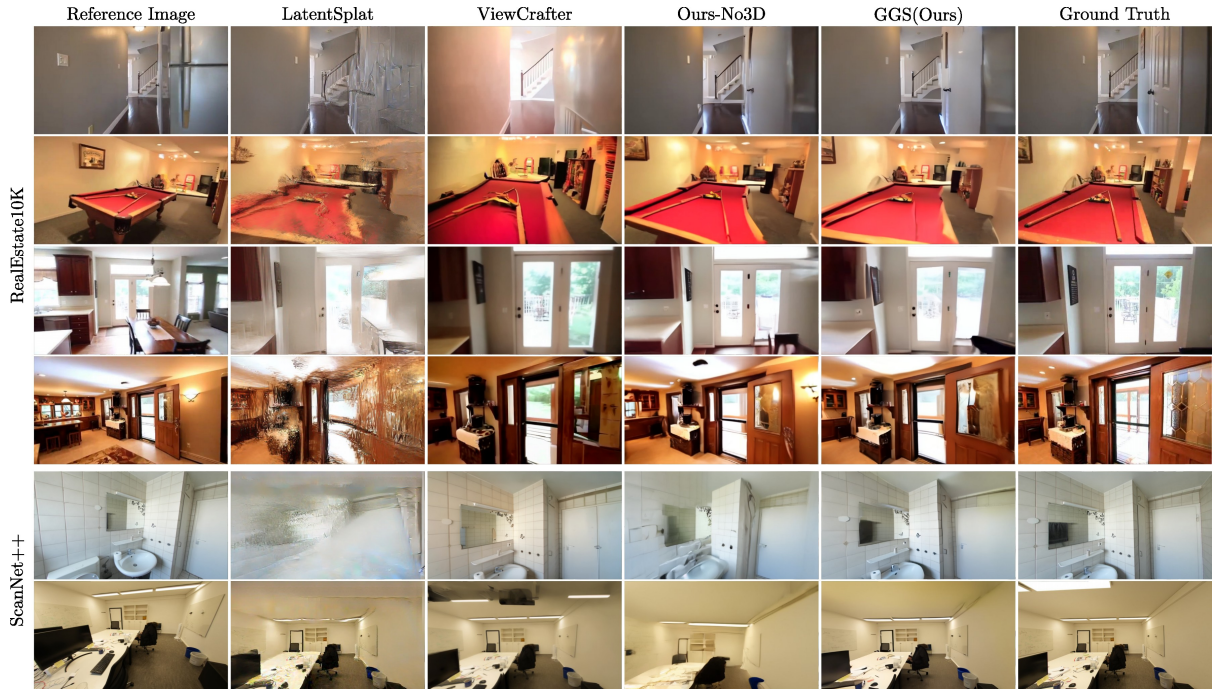

Figure 10. **Baseline Comparison For View Extrapolation Given Two Reference Images:** We show results for the strongest baselines LatentSplat [69] and ViewCrafter[76] together with our approach without (Ours-No3D) and with 3D representation (GGS). As both reference views are close together, we only include one image for reference. Best viewed zoomed in.

Figure 11. **3D Scene From a Single Image:** . We show generated Gaussian splats in image and feature space and the reference image.

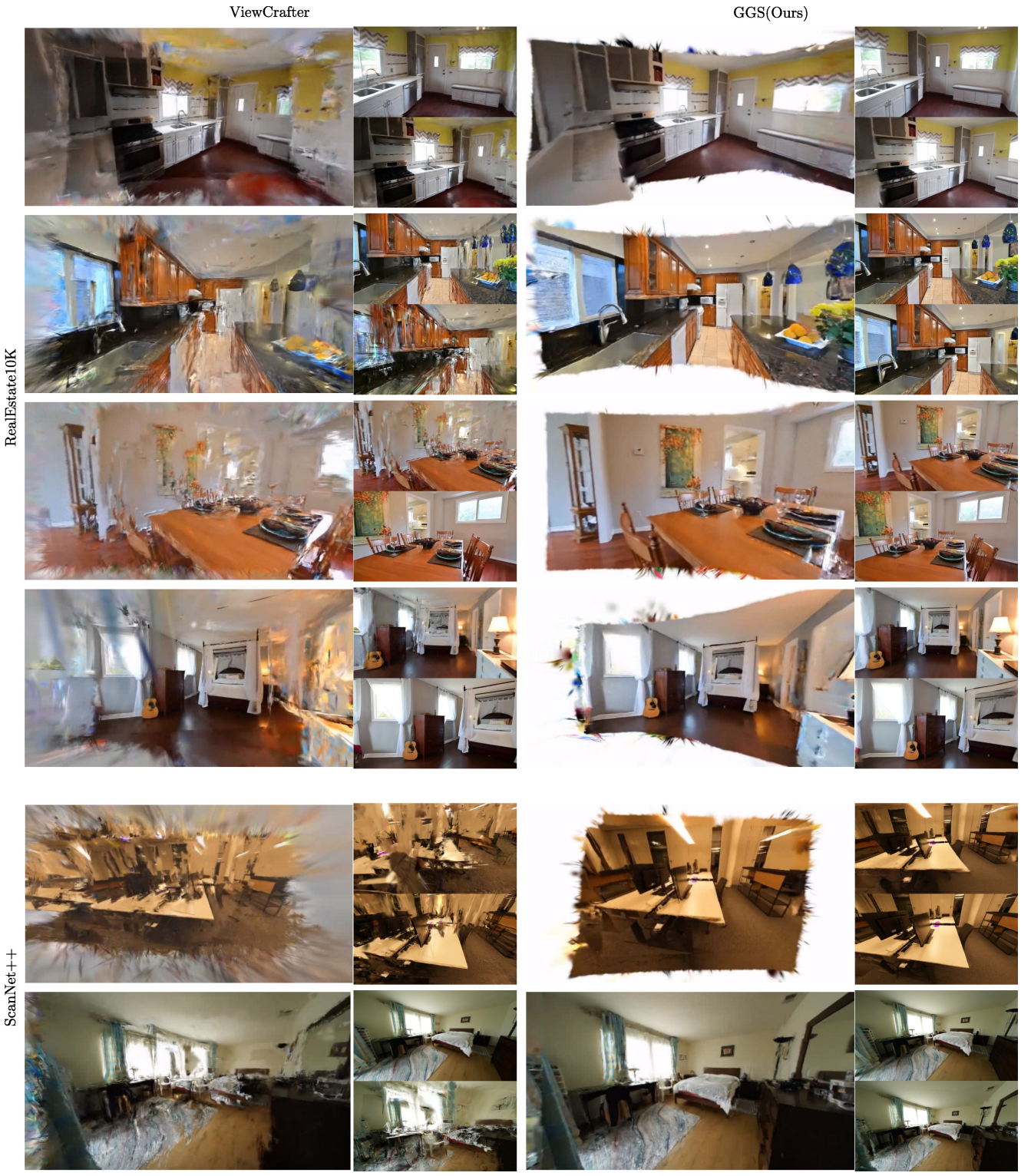

Figure 12. **3D Reconstruction Results From Generated Images:** We run an off-the-shelf 3DGS optimization on the generated multi-view images of ViewCrafter and GGS (Ours). For ViewCrafter, we use 15,000 optimization steps. For our approach, we only refine the generated splats with the generated multi-view images, using 5,000 iterations. The resulting 3D representation is shown on the left and two rendered views from novel viewpoints are included on the right.

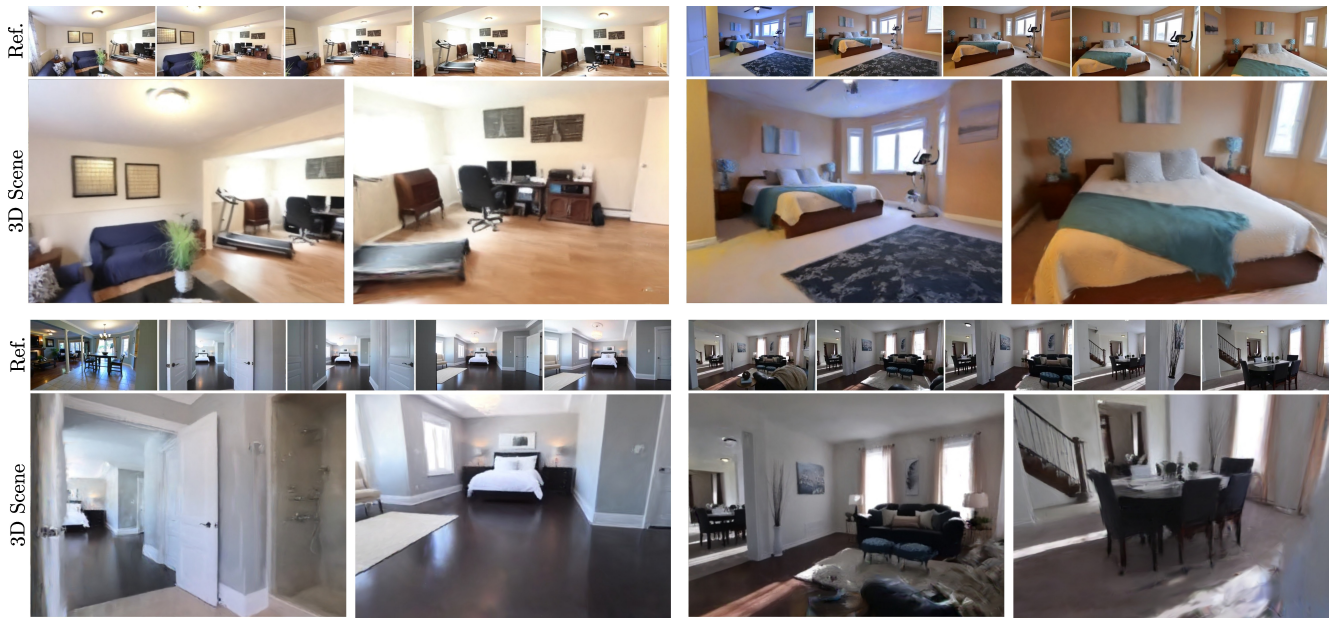

Figure 13. **Autoregressive Scene Synthesis with GGS:** By generating consistent views between the reference images and from additional viewpoints, GGS can augment the set of 5 reference images and generate larger 3D scenes autoregressively.