pixelSplat: 3D Gaussian Splats from Image Pairs for Scalable Generalizable 3D Reconstruction

Supplementary Material

1. Training Details

We train all methods using our data loaders, which gradually increase the distance between reference views during training. Specifically, over the first 150,000 training steps, we linearly increase the minimum distance between reference views from 25 to 45 and the maximum distance from 45 to 192.

1.1. Our Method

We train our method for 300,000 steps using a batch size of 7, which requires about 80 GB of VRAM on a single GPU. We use an MSE loss for the first 150,000 iterations and supplement it with an LPIPS loss with weight 0.05 starting at 150,000 steps. For each batch element (scene), we render 4 target views. To allow our method to produce gradients for multiple estimated depths during each forward pass, we place 3 Gaussians along each ray and determine their positions by independently sampling from the ray's probability distribution 3 times. We then divide each Gaussian's alpha value by 3 such that $\alpha = \frac{\phi_z}{3}$, which ensures that the Gaussians placed along any particular ray have a total opacity of roughly 1.

Depth regularization. To generate the point clouds shown in the main paper, we fine-tune our model for 50,000 steps with a depth regularization loss $\mathcal{L}_{\text{reg. depth}}$. To compute this loss, for each rendered view, we generate a corresponding depth map D. We use D to compute the loss as follows, where $D[\mathbf{u}_x, \mathbf{u}_y]$ represents indexing:

$$
\begin{aligned}
D_x^{\Delta}[\mathbf{u}] &= D[\mathbf{u}_{x-1}, \mathbf{u}_y] - 2D[\mathbf{u}_x, \mathbf{u}_y] + D[\mathbf{u}_{x+1}, \mathbf{u}_y] \\
D_y^{\Delta}[\mathbf{u}] &= D[\mathbf{u}_x, \mathbf{u}_{y-1}] - 2D[\mathbf{u}_x, \mathbf{u}_y] + D[\mathbf{u}_x, \mathbf{u}_{y+1}] \\
\mathbf{I}_x^{\Delta} &= \max(|\mathbf{I}[\mathbf{u}_{x-1}, \mathbf{u}_y] - \mathbf{I}[\mathbf{u}_x, \mathbf{u}_y]|, |\mathbf{I}[\mathbf{u}_{x+1}, \mathbf{u}_y] - \mathbf{I}[\mathbf{u}_x, \mathbf{u}_y]|) \\
\mathbf{I}_y^{\Delta} &= \max(|\mathbf{I}[\mathbf{u}_x, \mathbf{u}_{y-1}] - \mathbf{I}[\mathbf{u}_x, \mathbf{u}_y]|, |\mathbf{I}[\mathbf{u}_x, \mathbf{u}_{y+1}] - \mathbf{I}[\mathbf{u}_x, \mathbf{u}_y]|) \\
\mathcal{L}_{\text{reg. depth}}[\mathbf{u}] &= D_x^{\Delta}[\mathbf{u}] \exp\left(8 * \mathbf{I}_x^{\Delta}[\mathbf{u}]\right) + D_y^{\Delta}[\mathbf{u}] \exp\left(8 * \mathbf{I}_y^{\Delta}[\mathbf{u}]\right)
\end{aligned}
\tag{1}
$$

Transformation of Gaussians into world space. Because our model predicts Gaussian parameters in camera space, these parameters must be transformed into world space before rendering. Given a 4×4 camera-to-world extrinsics matrix $\mathbf{T}$ containing a 3×3 rotation $\mathbf{R}$ and a 3×1 translation $\mathbf{t}$, we transform them as follows:

$$
\begin{aligned}
\boldsymbol{\mu}_{\text{world}} &= \mathbf{T}\boldsymbol{\mu}_{\text{cam.}} \\
\boldsymbol{\Sigma}_{\text{world}} &= \mathbf{R}\boldsymbol{\Sigma}_{\text{cam.}}\mathbf{R}^T \\
\alpha_{\text{world}} &= \alpha_{cam.} \\
\mathbf{s}_{\text{world}} &= \mathbf{D}_{\mathbf{R}}\mathbf{s}_{\text{cam.}}
\end{aligned}
\tag{2}
$$

Here, $\mathbf{D}_{\mathbf{R}}$ is a block-diagonal matrix consisting of Wigner D matrices, which rotates the spherical harmonics coefficients $\mathbf{s}_{\text{cam.}}$. In practice, we use the e3nn library to compute these matrices and recompile the original 3D Gaussian splatting code base to follow e3nn's conventions.

1.2. Method of Du et al.

We train the method of Du et al. [10] for 300,000 iterations using a total batch size of 32 spread across 4 GPUs, which requires around 44 GB of VRAM per GPU. We train using the authors' default hyperparameters and enable the LPIPS loss after 150,000 iterations.

1.3. pixelNeRF

We train pixelNeRF [58] for 500,000 iterations using a batch size of 12, which requires about 20 GB of VRAM on a single GPU. We use the authors' default hyperparameters for the NMR dataset, in which the first pooling layer of pixelNeRF's ResNet is disabled to increase feature resolution. Following Du et al. [10], we set the near and far planes to be 0.1 and 10.0 respectively.

1.4. GPNR

We train GPNR [46] for 250,000 iterations using a batch size of 4098 spread across 6 GPUs, which requires about 67 GB of VRAM per GPU. We use the authors' default hyperparameters but reduce the learning rate to $1 * 10^{-4}$, since several attempts at using the default learning rate of $3 * 10^{-4}$ yielded sudden training collapses on our dataset.

2. Additional Results

We present additional results on the following pages.

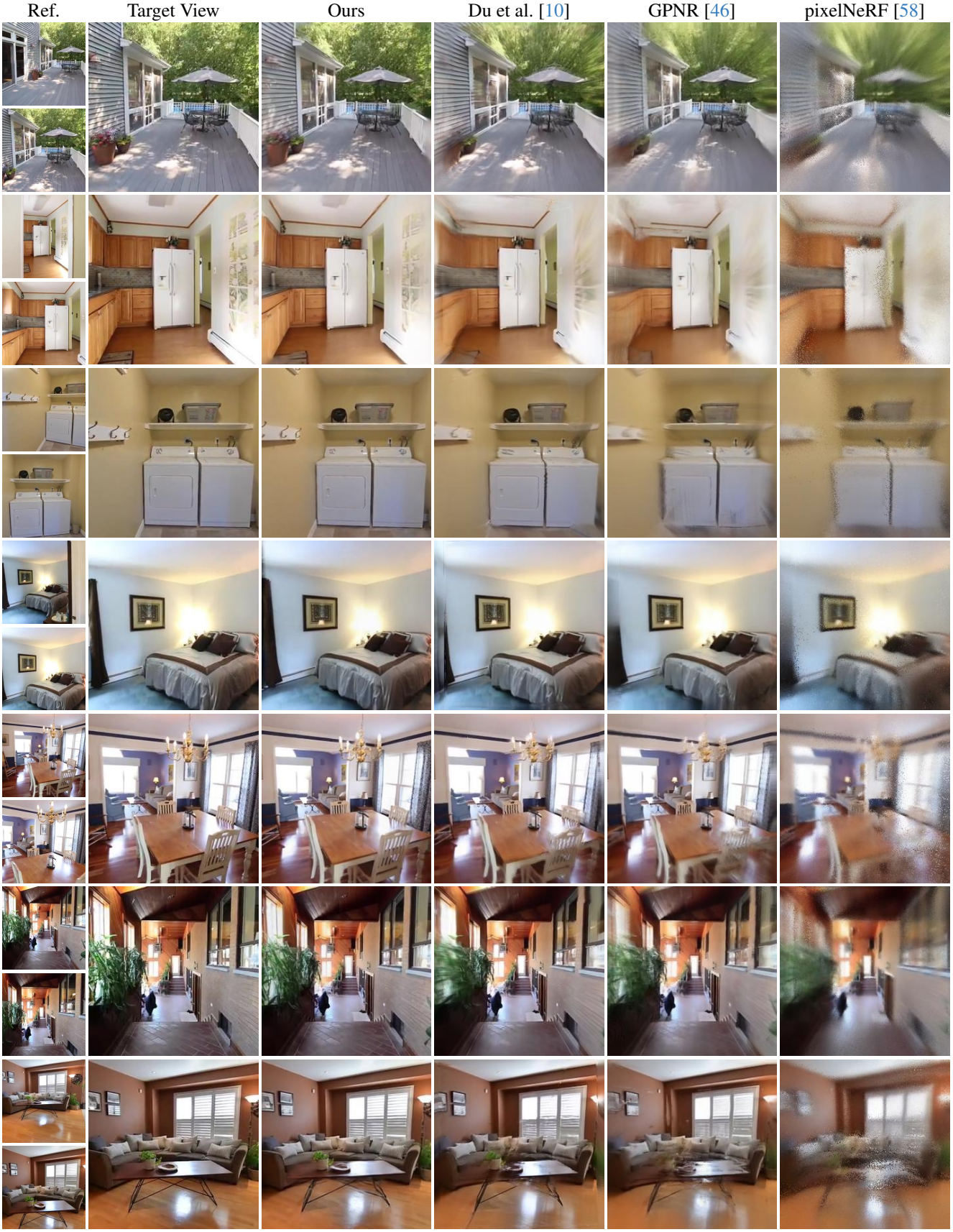

Figure 1. More results on the Real Estate 10k dataset.

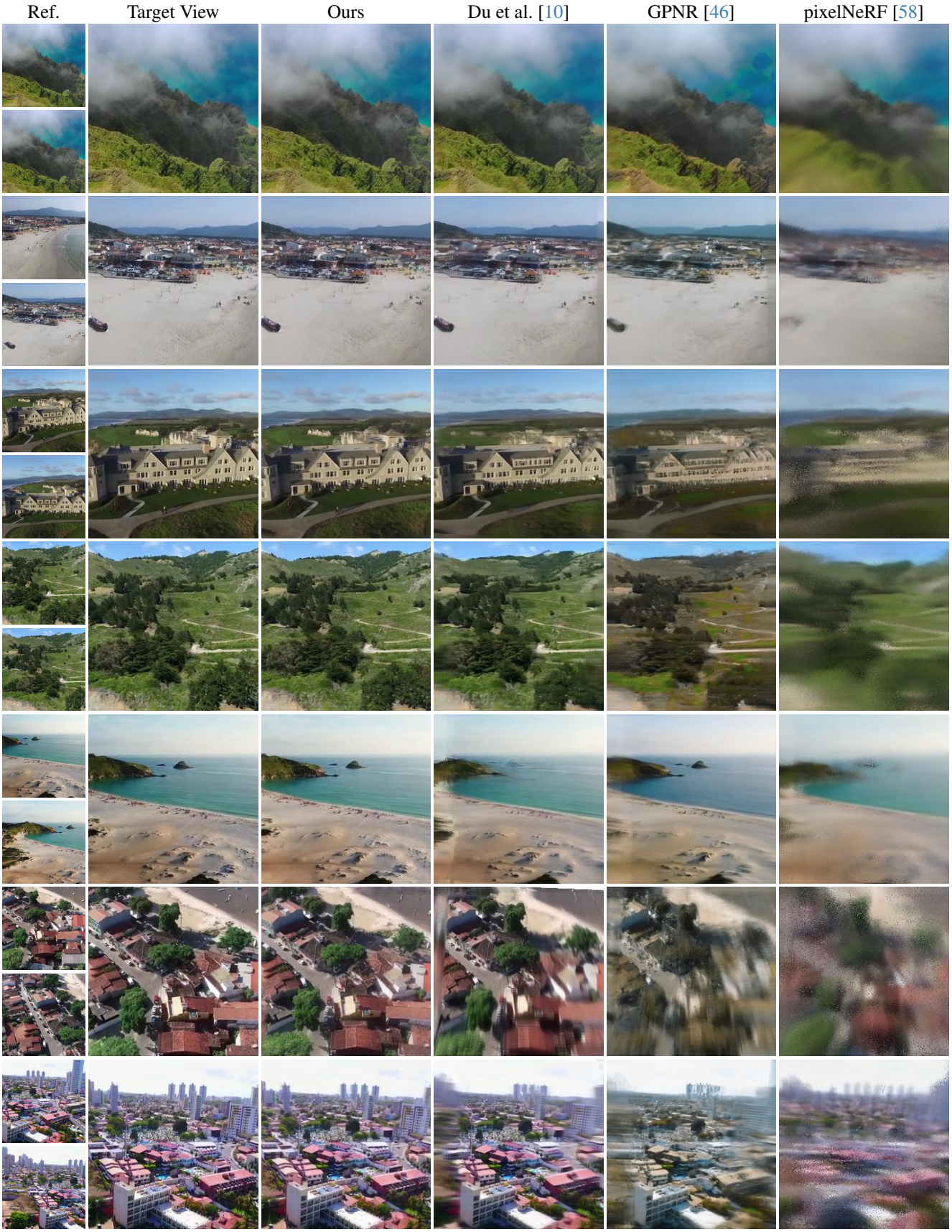

Figure 2. More results on the ACID dataset.