Splatter Image: Ultra-Fast Single-View 3D Reconstruction

Supplementary Material

	PSNR ↑	SSIM ↑	LPIPS ↓
Full model	**24.11**	**0.92**	**0.087**
w/o cross-view attn	23.68	**0.92**	0.091
w/o cam embed	23.91	**0.92**	0.088
w/o warping	23.84	**0.92**	0.088

Table 8. **Ablations: Multi-View Reconstruction.**

A. Additional results

Additional qualitative results Our project website contains a short summary of Splatter Image, videos of comparisons of our method to baselines and additional results from our method on the 4 object classes and the 2 multi-class datasets. Moreover, we present static comparisons of our method to PixelNeRF [55] and VisionNeRF on ShapeNet-SRN Cars and Chairs in Fig. 8, as well as static comparisons of our method to PixelNeRF on CO3D Hydrants and Teddybears in Fig. 9. In Fig. 10 we present additional static comparisons of our method to OpenLRM on the Google Scanned Objects dataset.

Multi-view model ablation. Table 8 ablates the multi-view model. We individually remove the multi-view attention blocks, the camera embedding and the warping component of the multi-view model and find that they all are important to achieve the final performance.

B. Data details

B.1. ShapeNet-SRN Cars and Chairs

We follow standard protocol in the ShapeNet-SRN datasets. We use the images, camera intrinsics, camera poses and data splits as provided by the dataset [45] at 128×128 resolution and train our method using *relative* camera poses: the reconstruction is done in the view space of the conditioning camera. For single-view reconstruction, we use view 64 as the conditioning view and in two-view reconstruction we use views 64 and 128 as conditioning. All other available views are used as target views in which we compute novel view synthesis merics.

B.2. CO3D

We use the first frame as input and all other frames as target frames. We use all testing sequences in the Hydrant and Teddybear classes where the first conditioning frame has a valid foreground mask (with probability $p > 0.8$). In practice, this means evaluating on 49 'Hydrant' and 93 'Teddybear' sequences.

Image center-cropping. Similarly to recent methods [4, 49] we take the largest crop in the original images centered on the principal point and resize to 128×128 resolution with Lanczos interpolation. Similarly to many single- and few-view reconstruction methods [24, 55, 59] we also remove backgrounds. We adjust the focal length accordingly with the resulting transformations. This is the only preprocessing we do – CO3D objects already have their point clouds normalised to zero-mean and unit variance.

Predicting Gaussian positions. Estimating the distance between the object and the camera from visual information alone is a challenging problem in this dataset: focal lengths vary between and within sequences, objects are partially cropped, and global scene parameters such as distance to the object, camera trajectory and the angle at which objects are viewed all vary, posing a challenge to both our and baseline methods. Thus, for both PixelNeRF and our method we set the center of prediction to the center of the object.

In our method we achieve this by setting $z_{\text{near}} = z_{\text{gt}} - w$ and $z_{\text{far}} = z_{\text{gt}} + w$, where z_{gt} is the ground truth distance from the object to the source camera and w is a fixed scalar $w = 2.0$. In PixelNeRF, we provide the network with $x = x_v - z_{\text{gt}}$ where x is the sample location at which we query the network and x_v is the sample location in camera view space. z_{gt} is computed as the perpendicular distance (along camera z-axis) to the world origin, which coincides with the center of the point cloud in CO3D.

B.3. Multi-class ShapeNet.

Identically to prior work, we use images, splits and camera parameters from NMR [21] which provides 64×64 renders from cameras at fixed elevations. For direct comparison with prior work [26, 55] we use the same source and target views for evaluation.

B.4. Objaverse and GSO data details.

We use renders from Zero-1-to-3 [29], filtered by the objects which appear in the LVIS subset to use only high-quality assets. The data is rendered at 512×512 resolution with focal length 560px with cameras pointing at the center of the object at randomly sampled distances. We resize data to 128×128 resolution with Lanczos interpolation, adjusting the focal length accordingly. At training and testing time we rescale the ground truth camera positions so that the distance from the object to the camera is a fixed scalar $d = 2$. GSO renders provided by Free3D [57] were rendered with the same parameters (resolution, distances, fo-

Figure 8. **ShapeNet-SRN.** Our method (fourth column) outputs reconstructions which are better than PixelNeRF (second column) and more or equally accurate than VisionNeRF (third column) while rendering 3 orders of magnitude faster (rendering speed in Frames Per Second denoted underneath method name).

Figure 9. **CO3D.** Our method (third column) outputs reconstructions which are sharper than PixelNeRF (second column) while rendering 3 orders of magnitude faster (rendering speed in Frames Per Second denoted underneath method name).

Figure 10. **Google Scanned Objects.** Our method (third column) outputs reconstructions which are comparable in quality to OpenLRM (second column) while requiring $\times 50$ less resources to train.

cal length) and we apply the same resolution scaling, focal length adjustment and camera scale adjustment at evaluation time.

C. Implementation details.

C.1. Splatter Image training.

We train our model (based on SongUNet [46]) with $\mathcal{L}_2$ reconstruction loss (Eq.4 main paper) on 3 unseen views and the conditioning view for 800,000 iterations. We use the network implementation from [20]. For single-class models, we use the Adam optimizer [23] with learning rate 5×10^{-5} and batch size 8. For multi-class ShapeNet model we use the same learning rate and batch size 32. Batch sizes are mainly dictated by GPU memory limits. For rasterization, we use the Gaussian Splatting implementation of [22]. After 800,000 iterations we decrease the learning rate by a factor of 10 and train for a further 100,000 (Cars, Hydrants, Teddybears), 150,000 (multi-class ShapeNet) or 200,000 (Chairs) iterations with the loss $\mathcal{L} = (1-\alpha)\mathcal{L}_2 + \alpha\mathcal{L}_{\text{LPIPS}}$ and $\alpha = 0.01$. Training done is on a single NVIDIA A6000 GPU and takes around 7 days.

Large dataset training. Training on Objaverse is done with Mixed Precision and effective batch size 32. We train first for $350,000$ iterations with learning rate 5×10^{-5} and $\alpha = 0$, followed by $40,000$ iterations with learning rate 6.3×10^{-5} and $\alpha = 0.338$. Training takes place on two NVIDIA A6000 GPUs for around 3.5 days.

Regularizers. For CO3D we additionally use regularisation losses to prevent exceedingly large or vanishingly small Gaussians for numerical stability. We regularize large Gaussians with the mean of their activated scale $s = \exp \hat{s}$ when it is bigger than a threshold scale $s_{\text{big}} = 20$.

$\mathcal{L}_{\text{big}} = (\sum_i s_i \mathbb{1}(s_i > s_{\text{big}}))/(\sum_i \mathbb{1}(s_i > s_{\text{big}}))$.

Small Gaussians are regularized with a mean of their negative deactivated scale $\hat{s}$ when it is smaller than a threshold $\hat{s}_{\text{small}} = -5$: $\mathcal{L}_{\text{small}} = (\sum_i -\hat{s}_i \mathbb{1}(\hat{s}_i < \hat{s}_{\text{smal 1}}))/(\sum_i \mathbb{1}(\hat{s}_i < \hat{s}_{\text{small}}))$.

Ablations. Due to computational costs, ablation models are trained at a shorter schedule 100k iterations with $\mathcal{L}_2$ and further 25k with $\mathcal{L}_2$ and $\mathcal{L}_{\text{LPIPS}}$ with $\alpha = 0.1$.

C.2. PixelNeRF.

For ShapeNet (single-class and multi-class) we use the scores reported in the original paper [55], as we train and evaluate on the same data. For training on CO3D, we use the official PixelNeRF implementation [55]. We use the same preprocessed data as for our method. We modify the activation function of opacity from ReLU to Softplus with the β parameter $\beta = 3.0$ for improved training stability. Parametrization of the sampling points to be centered about the ground truth distance to the camera z_{gt} as discussed in Appendix B.2 is available as default in the official implementation. As in original work, we train for $400,000$ iterations.

Method	GPU	Memory	# GPUs	Days	GPU × Days
VisionNeRF	A100	80G	16	5	80
NeRFDiff	A100	80G	16*	3	48
ViewDiff	A40	48G	2	3	6
PixelNeRF	TiRTX	24G	1	6	6
Ours - small scale	A6000	48G	1	7	7
LRM / OpenLRM*	A100	40G	128	3	384
Ours - Objaverse	A6000	48G	2	3.5	7

Table 9. **Training resources**. Ours, Viewset Diffusion and PixelNeRF have significantly lower compute costs than VisionNeRF and NeRFDiff. Our method is $\times 50$ cheaper to train than LRM. Memory denotes the memory capacity of the GPU. * denotes estimates.

C.3. OpenLRM.

OpenLRM was trained assuming distance to the object $d = 1.9$ and field-of-view $FOV = 40°$. To match this, we rescale the ground truth cameras so that the source camera was at distance $d = 1.9$ from the object. For exact comparison we use the same data for the baselines as for our method. For a fair comparison, we pass the 128×128 image as an input and render novel views at 128×128 too. Through experimentation we found that the best quantitative results were achieved by assuming the same field-of-view as at training time $FOV = 40°$.

D. Training resource estimate

We compare the compute resources needed at training time by noting the GPU used, its capacity, the number of GPUs and the number of days needed for training in Tab. 9. We report the compute resources reported in original works, where available. NeRFDiff only reports the resources needed to train their 'Base' models and the authors did not respond to our clarification emails about their 'Large' models which we compare against in the main paper. We thus report an estimate of such resources which we obtained by multiplying the number of GPUs used in the 'Base' models by a factor of 2. Our method is significantly cheaper than VisionNeRF and NeRFDiff. The resources required are similar to those of Viewset Diffusion and PixelNeRF, while we achieve better performance and do not require absolute camera poses. The difference between our method and prior works is even more striking on large datasets like Objaverse, where our method is $\times 50$ cheaper than LRM.

E. Covariance warping implementation

As described in Sec. 3.4 in the main paper, the 3D Gaussians are warped from one view's reference frame to another

with $\tilde{\Sigma} = R\Sigma R^{\top}$ where R is the relative rotation matrix of the reference frame transformation. The covariance is predicted using a 3-dimensional scale and quaternion rotation so that $\Sigma = R_q S R_q^{\top}$ where $S = \mathrm{diag}\left(\exp(\hat{s})\right)^2$. Thus the warping is applied by applying rotation matrix R to the orientation of the Gaussian $\tilde{R}_q = RR_q$. In practice this is implemented in the quaternion space with the composition of the predicted quaternion q and the quaternion representation of the relative rotation $p = m2q(R)$ where $m2q$ denotes the matrix-to-quaternion transformation, resulting in $\tilde{q} = pq$.