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

Stanislaw Szymanowicz   Christian Rupprecht   Andrea Vedaldi

Visual Geometry Group — University of Oxford

{stan,chrisr,vedaldi}@robots.ox.ac.uk

Figure 1. The **Splatter Image** is an ultra-efficient method for single- and few-view 3D reconstruction. It uses an image-to-image neural network to map the input image to another image that holds the parameters of one coloured 3D Gaussian per pixel. Splatter Image achieves excellent 3D reconstruction quality on synthetic, real and large-scale datasets while using a single GPU for training.

Abstract

We introduce the Splatter Image, an ultra-efficient approach for monocular 3D object reconstruction. Splatter Image is based on Gaussian Splatting, which allows fast and high-quality reconstruction of 3D scenes from multiple images. We apply Gaussian Splatting to monocular reconstruction by learning a neural network that, at test time, performs reconstruction in a feed-forward manner, at 38 FPS. Our main innovation is the surprisingly straightforward design of this network, which, using 2D operators, maps the input image to one 3D Gaussian per pixel. The resulting set of Gaussians thus has the form an image, the Splatter Image. We further extend the method take several images as input via cross-view attention. Owning to the speed of the renderer (588 FPS), we use a single GPU for training while generating entire images at each iteration to optimize perceptual metrics like LPIPS. On several synthetic, real, multi-category and large-scale benchmark datasets, we achieve better results in terms of PSNR, LPIPS, and other metrics while training and evaluating much faster than prior works. Code, models, demo and more results are available at https://szymanowiczs.github.io/splatter-image.

1. Introduction

We contribute Splatter Image, a method that achieves ultra-fast single-view reconstruction of the 3D shape and appearance of objects. Splatter Image uses a set of 3D Gaussians as the 3D representation, taking advantage of the rendering quality and speed of Gaussian Splatting [22]. Splatter Image works by predicting a 3D Gaussian for each of the input image pixels, using an image-to-image neural network. Remarkably, the predicted 3D Gaussians provide 360° reconstructions of quality comparable or superior to much slower methods (Fig. 1).

We formulate monocular 3D reconstruction as the problem of designing a neural network that takes an image of an object as input and produces as output a corresponding Gaussian mixture that represents all sides of it. While a Gaussian mixture is a set, *i.e.*, an unordered collection, it can still be stored in an ordered data structure. Splatter Image takes advantage of this fact by using a *2D image* as the container of the 3D Gaussians, storing the parameters of one Gaussian (*i.e.*, its opacity, position, shape, and colour) per pixel. The Gaussians predominantly lie on the rays from the camera to the object, but they can also be placed off the rays (Fig. 2), enabling 360° object representation.

The advantage of storing a set of 3D Gaussians in an image is that it reduces the reconstruction problem to learning

an image-to-image neural network. In this manner, the reconstructor can be implemented utilizing only efficient 2D operators (*e.g.*, 2D convolution instead of 3D convolution). We use in particular a U-Net [42] as those have demonstrated excellent performance in image generation [41]. In our case, their ability to capture small image details [55] helps to obtain higher-quality reconstructions.

Since the 3D representation in Splatter Image is a mixture of 3D Gaussians, it enjoys the rendering and space efficiency of Gaussian Splatting, which benefits inference and training. In particular, rendering stops being a training bottleneck [34] and we can afford to generate complete views of the object to optimize perceptual metrics like LPIPS [56]. More importantly, the efficiency is such that our model can be trained on a *single GPU* on standard benchmarks of 3D objects or two GPUs on large datasets such as Objaverse [10], whereas alternative methods typically require distributed training on dozens [26] or even hundreds [17] of GPUs. We also extend Splatter Image to take several views as input. This is achieved by taking the union of the Gaussian mixtures predicted from individual views, after registering them to a common coordinate frame. The different views communicate during prediction via lightweight cross-view attention layers in the architecture.

Empirically, we show that, while the network only sees one side of the object, it can still produce a 360° reconstruction of it by using the prior acquired during training. The 360° information is encoded in the 2D image by allocating different Gaussians in a given 2D neighbourhood to different parts of the 3D object.

We validate Splatter Image by comparing it to alternative, slower reconstructors on standard benchmark datasets like ShapeNet [5] and CO3D [38]. To assess scalability and generalization, we also apply Splatter Image to multi-category reconstruction and train it on Objaverse [10], and evaluate it on Google Scanned Objects [12]. We obtain results of quality comparable to the recent Large Reconstruction Model of [15, 17], which is 50× more expensive to train. In fact, in several cases we even outperform slower methods in PSNR and LPIPS. We argue that this is because the very efficient design allows training the model very effectively, including using image-level losses like LPIPS.

To summarise, our contributions are: (1) to port Gaussian Splatting to learning-based monocular reconstruction; (2) to do so with the Splatter Image, a straightforward, efficient and performant 3D reconstruction approach that operates at 38 FPS on a standard GPU and affords single-GPU training; (3) to also extend the method to multi-view reconstruction; (4) and to obtain state-of-the-art reconstruction performance in multiple standard benchmarks, including synthetic, real, multi-category and large-scale datasets, in terms of reconstruction quality and speed.

2. Related work

Representations for single-view 3D reconstruction. In recent years, implicit representations like NeRF [34] have dominated learning-based few-view reconstruction, parameterising the MLP in NeRF using global [19, 39], local [55] or both global and local codes [26]. However, implicit representations, particularly MLP-based ones, are notoriously slow to render, up to 2s for a single 128×128 image.

Follow-up works [14, 48] used faster, explicit, voxel grid representations that encode opacities and colours directly. Similar to DVGO [47], they achieve significant speed-ups, but, due to their voxel-based representation, they scale poorly with resolution. They also assume the knowledge of the absolute viewpoint of each object image.

The triplane representation [3, 7] was proposed as a compromise between rendering speed and memory consumption. While they are not as fast to render as explicit representations, they allow view-space reconstruction [13] and are fast enough to be effectively used for single-view reconstruction [1, 13]. Triplane-based reconstructors were shown to scale to large datasets like Objaverse [9, 10], albeit at the cost of hundreds of GPUs for multiple days [17, 50].

In contrast to these works, our method predicts a mixture of 3D Gaussians in a feed-forward manner. As a result, our method is cheap to train (1-2 GPUs), fast at inference and achieves real-time rendering speeds while achieving state-of-the-art image quality across multiple metrics on multiple standard single-view reconstruction benchmarks, including single- [45] and multi-category ShapeNet [5, 21].

When more than one view is available at the input, one can use them to estimate the scene geometry [6, 31], learn a view interpolation function [51] or optimize a 3D representation of a scene using priors [18]. Our method is primarily a single-view reconstruction network, but we do show how Splatter Image can be extended to fuse multiple views. However, we focus our work on object-centric reconstruction rather than on generalising to unseen scenes.

3D Reconstruction with Point Clouds. PointOutNet [11] took image encoding as input and trained point cloud prediction networks [37] using 3D point cloud supervision. PVD [58] and PC^2 [32] extended this approach using Diffusion Models [16] by conditioning the denoising process on partial point clouds and RGB images, respectively. These approaches require ground truth 3D point clouds, limiting their applicability. Other works [27, 40, 53] use point clouds as intermediate 3D representations for conditioning 2D inpainting or generation networks. However, these point clouds are assumed to correspond to only visible object points. In contrast, our Gaussians can model any part of the object, and thus afford 360° reconstruction.

Point cloud-based representations have also been used for high-quality reconstruction from multi-view images.

Novel views can be rendered with 2D inpainting networks for hole-filling [43], or by using non-isotropic 3D Gaussians with variable scale [22]. While showing high-quality results, Gaussian Splatting [22] requires many images per scene and has not yet been used in a learning-based reconstruction framework as we do here.

Our method also uses 3D Gaussians as an underlying representation but predicts them from as few as a single image. Moreover, it outputs a full $360°$ 3D reconstruction without using 2D or 3D inpainting networks.

Probabilistic 3D Reconstruction. Single-view 3D reconstruction is an ambiguous problem, so recently it has been tackled as a conditional generation task. Diffusion Models have been employed for conditional novel view synthesis [4, 29, 52]. Due to generating images without underlying geometries, the output images exhibit noticeable flicker. This can be mitigated by simultaneously generating multi-view images [30, 44], reconstructing a geometry at every step of the denoising process [48, 49, 54] or training a robust reconstructor [25, 28]. Other works build and use a 3D [8, 35] or 2D [13, 33] prior which can be used in an image-conditioned auto-decoding framework.

Here, we focus on deterministic reconstruction. However, few-view reconstruction is required to output 3D geometries from feed-forward methods [30, 44, 48, 49, 54]. Our method is capable of few-view 3D reconstruction, thus it is complementary to these generative methods and could lead to improvements in generation speed and quality.

3. Method

We discuss Gaussian Splatting in Sec. 3.1 for background, and then describe the Splatter Image in Secs. 3.2 to 3.6.

3.1. Overview of Gaussian Splatting

A *radiance field* [34] is a pair of functions, assigning an opacity $\sigma(\boldsymbol{x}) \in \mathbb{R}_+$ and a colour $c(\boldsymbol{x}, \boldsymbol{\nu}) \in \mathbb{R}^3$ to each 3D point $\boldsymbol{x} \in \mathbb{R}^3$ and viewing direction $\boldsymbol{\nu} \in \mathbb{S}^2$. Gaussian Splatting [60] represents the two functions σ and c as a mixture θ of G colored 3D Gaussians

$$g_i(\boldsymbol{x}) = \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_i)^\top \Sigma_i^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_i)\right),$$

where $1 \leq i \leq G$, $\boldsymbol{\mu}_i \in \mathbb{R}^3$ is the Gaussian mean or center and $\Sigma_i \in \mathbb{R}^{3 \times 3}$ is its covariance, specifying its shape and size. Each Gaussian has also an opacity $\sigma_i \in [0, 1]$ and a view-dependent colour $c_i(\boldsymbol{v}) \in \mathbb{R}^3$. Together, they define a radiance field as follows:

$$\sigma(\boldsymbol{x}) = \sum_{i=1}^{G} \sigma_i g_i(\boldsymbol{x}), \quad c(\boldsymbol{x}, \boldsymbol{\nu}) = \frac{\sum_{i=1}^{G} c_i(\boldsymbol{\nu})\sigma_i g_i(\boldsymbol{x})}{\sum_{j=1}^{G} \sigma_i g_i(\boldsymbol{x})}. \tag{1}$$

The mixture of Gaussians is thus given by the *set*

$$\theta = \{(\sigma_i, \boldsymbol{\mu}_i, \Sigma_i, c_i), i = 1, \ldots, G\}.$$

A raidance field is rendered into an image $I(\boldsymbol{u})$ by integrating the colors observed along the ray $\boldsymbol{x}_\tau = \boldsymbol{x}_0 - \tau\boldsymbol{\nu}$, $\tau \in \mathbb{R}_+$ that passes through each image pixel $\boldsymbol{u}$ via the equation:

$$I(\boldsymbol{u}) = \int_0^\infty c(\boldsymbol{x}_\tau, \boldsymbol{\nu})\sigma(\boldsymbol{x}_\tau)e^{-\int_0^\tau \sigma(\boldsymbol{x}_\mu)\,d\mu}\,d\tau. \tag{2}$$

Gaussian Splatting [22, 60] provides a very fast differentiable renderer $I = \mathcal{R}(\theta, \pi)$ that approximates Eq. (2), mapping the mixture θ and the viewpoint π to an image I.

3.2. The Splatter Image

To perform monocular reconstruction we seek for a function $\theta = \mathcal{S}(I)$ which is the 'inverse' of the renderer $\mathcal{R}$, mapping an image I to a mixture of 3D Gaussians θ. Our key innovation is to propose an extremely simple and yet effective design for such a function. Specifically, we predict a Gaussian for each pixel of the input image I using a standard image-to-image neural network architecture. We call its output image M the Splatter Image.

In more detail, Let $\boldsymbol{u} = (u_1, u_2, 1)$ denote one of the $H \times W$ image pixels. This corresponds to ray $\boldsymbol{x} = \boldsymbol{u}d$ in camera space, where d is the depth of the ray point. Our network f takes as input the $H \times W \times 3$ RGB image I, and outputs directly a $H \times W \times K$ tensor M, where each pixel is associated to the K-dimensional feature vector packing the parameters $M_{\boldsymbol{u}} = (\sigma, \boldsymbol{\mu}, \Sigma, c)$ of a corresponding Gaussian.

We assume that Gaussians are expressed in the same reference frame of the camera. As illustrated in Fig. 2, the network predicts the depth d and offset $(\Delta_x, \Delta_y, \Delta_y)$, setting

$$\boldsymbol{\mu} = \begin{bmatrix} u_1 d + \Delta_x \\ u_2 d + \Delta_y \\ d + \Delta_z \end{bmatrix}. \tag{3}$$

The network also predicts the opacity σ, the shape Σ and the colour c. For now, we assume that the colour is Lambertian, *i.e.*, $c(\nu) = c \in \mathbb{R}^3$, and relax this assumption in Sec. 3.5. Section 3.6 provides more detail on the network architecture.

Discussion. One may wonder how this design can predict a full $360°$ reconstruction of the object when the reconstruction is aligned a single input view. We find that the network adjusts the 3D offsets Δ and depths d to allocate some of the 3D Gaussians to reconstruct the input view, and some to reconstruct unseen portions of the object, automatically. The network can also decide to switch off any Gaussian by simply predicting $\sigma = 0$, if needed. These points are then not rendered and can be culled in post-processing.
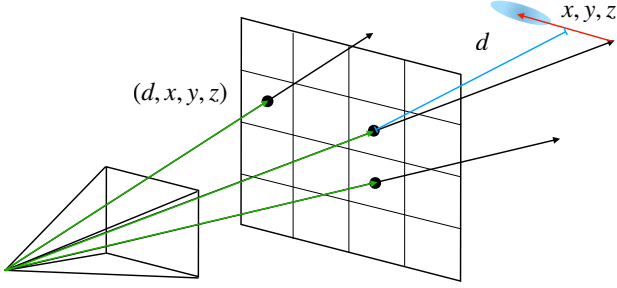
Figure 2. **Predicting locations.** The location of each Gaussian is parameterised by depth d and a 3D offset $\Delta = (\Delta_x, \Delta_y, \Delta_z)$. The 3D Gaussians are projected to depth d (blue) along camera rays (green) and moved by the 3D offset Δ (red).

Our design can also be seen as an extension of depth prediction networks which only predicting the depth of each pixel. Here, we also predict unobserved parts of the geometry, as well as the shape and appearance of each Gaussian.

3.3. Learning formulation

Learning to predict the Splatter Image is simple and efficient. It can be done on a single GPU using at most 20GB of memory at training time in most of our single-view reconstruction experiments (except for Objaverse, where we use 2 GPUs and using 26GB of memory on each). For training, we assume a multi-view dataset, either real or synthetic. The dataset $\mathcal{D}$ consists of triplets (I, J, π), where I is a source image, J a target image, and π the viewpoint change between the source and target cameras. Then we simply feed the source I as input to Splatter Image, and minimize the average reconstruction loss of target view J:

$$\mathcal{L}(\mathcal{S}) = \frac{1}{|\mathcal{D}|} \sum_{(I,J,\pi)\in\mathcal{D}} \|J - \mathcal{R}(\mathcal{S}(I), \pi)\|^2. \quad (4)$$

Image-level losses. A main advantage of the speed and efficiency of our method is that it allows for rendering entire images at each training iteration, even for relatively large batches (this differs from NeRF [34], which only generates a certain number of pixels in a batch). In particular, this means that, in addition to decomposable losses like the $L2$ loss above, we can use image-level losses like LPIPS [56], which do not decompose into per-pixel losses. In practice, we experiment with a combination of such losses.

Regularisations. We also add generic regularisers to prevent parameters from taking on unreasonable values (*e.g.*, Gaussians which are larger than the reconstructed objects, or vanishingly small). Please see the sup. mat. for details.

3.4. Extension to multiple input viewpoints

If two or more input views I_j, $j \in \{1, \ldots, N\}$ are provided, we can apply network $\mathcal{S}$ multiple times to obtain multiple Splatter Images M_j, one per view. If (R, T) is the relative camera pose change from an additional view to the reference view, we can take the mixture of 3D Gaussians θ defined in the additional view's coordinates and warp it to the reference view. Specifically, a Gaussian of parameters $(\sigma, \boldsymbol{\mu}, \Sigma, c)$ maps to Gaussian of parameters $(\sigma, \tilde{\boldsymbol{\mu}}, \tilde{\Sigma}, \tilde{c})$ where $\tilde{\boldsymbol{\mu}} = R\boldsymbol{\mu} + T$, $\tilde{\Sigma} = R\Sigma R^\top$, $\tilde{c} = c$. We use the symbol $\phi[\theta]$ to denote the Gaussian mixture obtained by warping each Gaussian in θ. Here we have also assumed a Lambertian colour model and will discuss in Sec. 3.5 how more complex models transform.

Given N different views I_j and corresponding warps ϕ, we can obtain a composite mixture of 3D Gaussians simply by taking their union $\Theta = \bigcup_{j=1}^{N} \phi_j[\mathcal{S}(I_j)]$. Note that this set of 3D Gaussians is defined in the coordinate system of the reference camera.

3.5. View-dependent colour

Generalising beyond the Lambertian colour model, we use *spherical harmonics* [22] to represent view-dependent colours. For a particular Gaussian $(\sigma, \boldsymbol{\mu}, \Sigma, c)$, we then define $[c(\boldsymbol{\nu}; \boldsymbol{\alpha})]_i = \sum_{l=0}^{L} \sum_{m=-L}^{L} \alpha_{ilm} Y_l^m(\boldsymbol{\nu})$ where α_{ilm} are coefficients predicted by the network and Y_l^m are spherical harmonics, L is the order of the expansion, and $\boldsymbol{\nu} \in \mathbb{S}^2$ is the viewing direction.

The viewpoint change of Sec. 3.4 transforms a viewing direction $\boldsymbol{\nu}$ in the source camera to the corresponding viewing direction in the reference frame as $\tilde{\boldsymbol{\nu}} = R\boldsymbol{\nu}$. We can then find the transformed colour function by finding the coefficients $\tilde{\boldsymbol{\alpha}}$ such that $c(\boldsymbol{\nu}; \boldsymbol{\alpha}) = c(\tilde{\boldsymbol{\nu}}; \tilde{\boldsymbol{\alpha}})$. This is possible because (each order of) spherical harmonics are closed under rotation. However, the general case requires the computation of Wigner matrices. For simplicity, we only consider orders $L = 0$ (Lambertian) and $L = 1$. Hence, the first level has one constant component Y_0^0 and the second level has three components which we can write collectively as $Y_1 = [Y_1^{-1}, Y_1^0, Y_1^1]$ such that

$$Y_1(\boldsymbol{\nu}) = \sqrt{\frac{3}{4\pi}} \Pi \boldsymbol{\nu}, \quad \Pi = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

We can then conveniently rewrite $[c(\boldsymbol{\nu}; \boldsymbol{\alpha})]_i = \alpha_{i0} + \boldsymbol{\alpha}_{i1}^\top Y_1(\boldsymbol{\nu})$. From this and $c(\boldsymbol{\nu}; \alpha_0, \boldsymbol{\alpha}_1) = c(\tilde{\boldsymbol{\nu}}; \tilde{\alpha}_0, \tilde{\boldsymbol{\alpha}}_1)$ we conclude that $\tilde{\alpha}_{i0} = \tilde{\alpha}_{i0}$, and $\tilde{\boldsymbol{\alpha}}_{i1} = \Pi^{-1} R \Pi \boldsymbol{\alpha}_{i1}$.

3.6. Neural network architecture

The bulk of the predictor $\mathcal{S}$ mapping the input image I to the mixture of Gaussians θ is architecturally identical to the SongUNet of [46]. The last layer is replaced with a 1×1 convolutional layer with $12 + k_c$ output channels, where $k_c \in \{3, 12\}$ depending on the colour model. Given $I \in \mathbb{R}^{3 \times H \times W}$ as input, the network thus produces

a $(12 + k_c) \times H \times W$ tensor as output, coding, for each pixel $\boldsymbol{u}$ channels, the parameters $(\hat{\sigma}, \Delta, \hat{d}, \hat{\boldsymbol{s}}, \hat{\boldsymbol{q}}, \boldsymbol{\alpha})$ which are then transformed to opacity, offset, depth, scale, rotation and colour, respectively. These are activated by non-linear functions to obtain the Gaussian parameters. Specifically, the opacity is obtained using the sigmoid operator as $\sigma = \mathrm{sigmoid}(\hat{\sigma})$. The depth is obtained as $d = (z_{\mathrm{far}} - z_{\mathrm{near}})\,\mathrm{sigmoid}(\hat{d}) + z_{\mathrm{near}}$. The mean $\boldsymbol{\mu}$ is then obtained using Eq. (3). Following [22], the covariance is obtained as $\Sigma = R(\boldsymbol{q})\,\mathrm{diag}(\exp \hat{\boldsymbol{s}})^2 R(\boldsymbol{q})^\top$ where $R(\boldsymbol{q})$ is the rotation matrix with quaternion $\boldsymbol{q} = \hat{\boldsymbol{q}}/\|\hat{\boldsymbol{q}}\|$ and $\hat{\boldsymbol{q}} \in \mathbb{R}^4$.

For multi-view reconstruction, we apply the same network to each input view and then use the approach of Sec. 3.4 to fuse the individual reconstructions. In order to allow the network to coordinate and exchange information between views, we apply two modifications to it.

First, we condition the network with the corresponding camera pose (R, T) (we only assume access to the *relative* camera pose to a common but otherwise arbitrary reference frame). In fact, since we consider cameras in a turn-table-like configuration, we only pass vectors $(R\boldsymbol{e}_3, T)$ where $\boldsymbol{e}_3 = (0, 0, 1)$. We do so by encoding each entry via a sinusoidal positional embedding of order 9, resulting in 60 dimensions in total. Finally, these are applied to the U-Net blocks via FiLM [36] embeddings.

Second, we add cross-attention layers to allow communication between the features of different views. We do so in a manner similar to [44], but only at the lowest UNet resolution, which maintains the computational cost very low.

4. Experiments

We evaluate our method extensively for single-view reconstruction on six standard benchmarks. Next, we assess the quality of multi-view reconstruction, and finish with an evaluation of the speed of the method.

Datasets. The standard benchmark for evaluating single-view 3D reconstruction is ShapeNet-SRN [45]. We train our method in the single-class setting and report results on the "Car" and "Chair" classes, following prior work. Moreover, we challenge our method with two classes of real objects from the CO3D [38] dataset: Hydrants and Teddybears. In this challenging dataset ripe with ambiguities we set z_{far} and z_{near} to depend on ground truth distance z_{gt} between the object and camera.

We further test our method on two multi-category datasets. First, we use the standard benchmark of multi-category ShapeNet (with objects from 13 largest categories), and use the renderings, standard splits and target views from NMR [21]. Secondly, we train one model on renderings of objects from Objaverse-LVIS [10] which contains over 1k object categories, using the renderings from Zero-1-to-3 [29]. We evaluate this model on all objects

Method	RC	1-view Cars			1-view Chairs		
		PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
SRN	✗	22.25	0.88	0.129	22.89	0.89	0.104
CodeNeRF	✗	23.80	0.91	0.128	23.66	0.90	0.166
FE-NVS	✗	22.83	0.91	0.099	23.21	0.92	0.077
ViewsetDiff w/o $\mathcal{D}$	✗	23.21	0.90	0.116	24.16	0.91	0.088
PixelNeRF	✓	23.17	0.89	0.146	23.72	0.90	0.128
VisionNeRF	✓	22.88	0.90	0.084	24.48	0.92	0.077
NeRFDiff w/o NGD	✓	23.95	**0.92**	0.092	**24.80**	**0.93**	0.070
Ours	✓	**24.00**	**0.92**	**0.078**	24.43	**0.93**	**0.067**

Table 1. **ShapeNet-SRN: Single-View Reconstruction.** Our method achieves State-of-the-Art reconstruction quality on all metrics on the Car dataset and on two metrics in the Chair dataset, while performing reconstruction in the camera view-space. 'RC' indicates if a method can operate using only relative camera poses.

from the Google Scanned Objects dataset [12], using the same renderings as used for evaluation in Free3D [57]. We train and evaluate all models at 128×128 resolution, apart from multi-category ShapeNet which is at 64×64. Finally, we use the ShapeNet-SRN Cars dataset for the evaluation of the two-view reconstruction quality. For more details on datasets see supp. mat.

Baselines. For ShapeNet (both single-class and multi-class), we compare against implicit [19, 26, 45, 55], hybrid implicit-explicit [13] and explicit methods [2, 14, 48]. We use the deterministic variants of [13, 48] by using their reconstruction network in a single forward pass. For CO3D we compare against PixelNeRF which we train for 400,000 iterations with their officially released code on the same data as used for our method. Finally, on Objaverse-LVIS we compare to OpenLRM [15] (open-source version of LRM [17]): a large triplane-based reconstructor, trained on the full Objaverse dataset. Since we are proposing a deterministic reconstruction method, we do not compare to methods that employ Score Distillation [29, 59] or feed-forward diffusion models [4, 28, 52, 54].

Implementation details can be found in the supp. mat.

4.1. Evaluation of reconstruction quality

In line with related works [26, 55], we assess the quality of the reconstructions by measuring novel view synthesis quality and report Peak Signal-to-Noise Ratio (PSNR), Structural Similarity (SSIM) and a perceptual loss (LPIPS). We perform reconstruction from a given source view and render the 3D shape to unseen target views following standard protocols as detailed in the supp. mat.

4.1.1 Single-view 3D reconstruction

ShapeNet. In Tab. 1 and Fig. 3 we compare the single-view reconstruction quality on the ShapeNet-SRN benchmark.
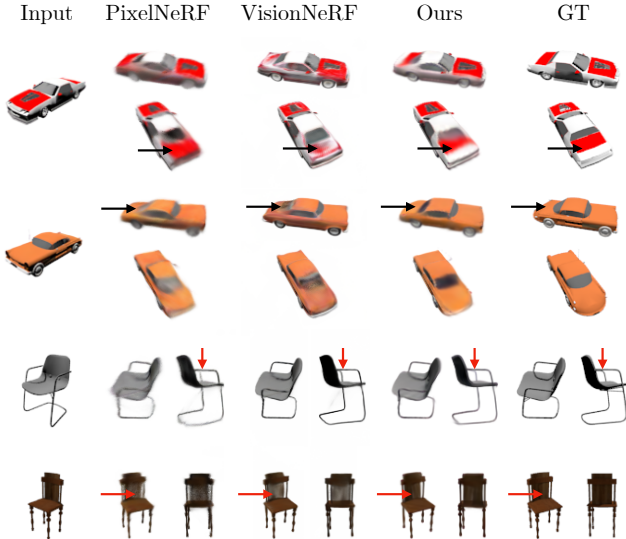
Figure 3. **ShapeNet-SRN Comparison.** Our method outputs more accurate reconstructions (cars' backs, top chair) and better represents thin regions (bottom chair).

	PixelNeRF [55]	FE-NVS [14]	FWD [2]	VisionNeRF [26]	Ours
PSNR ↑	26.80	27.08	26.66	28.76	**29.38**
SSIM ↑	0.91	0.92	0.91	0.93	**0.95**
LPIPS ↓	0.108	0.082	0.055	0.065	**0.047**

Table 2. Our method achieves State-of-the-Art quality of single-view reconstruction on multi-class ShapeNet dataset.

Object	Method	PSNR ↑	SSIM ↑	LPIPS ↓
Hydrant	PixelNeRF	21.76	0.78	0.203
Hydrant	**Ours**	**21.80**	**0.80**	**0.150**
Teddybear	PixelNeRF	19.38	0.65	0.290
Teddybear	**Ours**	**19.44**	**0.73**	**0.231**

Table 3. **CO3D: Single-View.** Our method outperforms Pixel-NeRF on this challenging benchmark across all metrics.

Our method outperforms all deterministic reconstructors in SSIM and LPIPS, obtaining sharper new views. Furthermore, our method requires only relative camera poses instead of absolute/canonical ones. Qualitatively, our method does well in challenging situations with limited visibility and thin structures. In Tab. 2, we use instead the multi-category ShapeNet protocol we observe that our method outperforms more expensive baselines [26] across all metrics in the multi-category ShapeNet setting.

CO3D. On CO3D bears and hydrants, our model outperforms PixelNeRF on all metrics (Tab. 3), and qualitatively produces sharper images (Fig. 4) while being 1,000× faster.

Objaverse-LVIS and Google Scanned Objects. We compare our method to OpenLRM [15], an open-source ver-
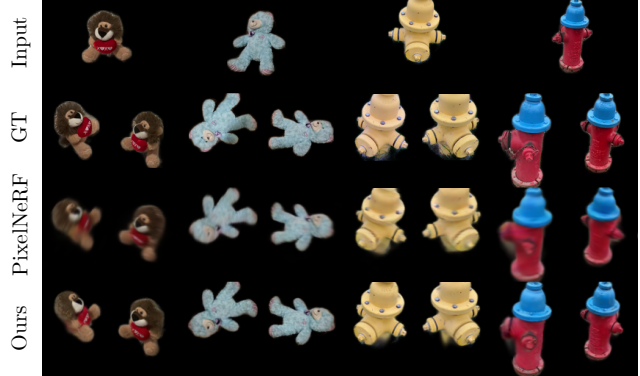

Figure 4. **CO3D Hydrants and Teddybears.** Our method outputs sharper reconstructions than PixelNeRF while being 100x faster in inference.

Method	PSNR ↑	SSIM ↑	LPIPS ↓
OpenLRM	18.06	0.84	0.129
Ours	**21.06**	**0.88**	**0.111**

Table 4. **Google Scanned Objects: Single-View.** Our method outperforms the much more expensive LRM [15, 17] on single-view open-world reconstruction.

Figure 5. **Google Scanned Objects.** On large datasets our model has similar quality to much more expensive baselines (shoe). Our reconstructions have more accurate lighting (Jenga), object pose (horse) and shape (toy).

sion of the LRM model [17], on the Google Scanned Objects [12] evaluation renderings from Free3D [57]. Quantitatively, in Tab. 4 Splatter Image outperforms OpenLRM and, qualitatively (Fig. 5), it is comparable. Our models perform well even on images collected from the Internet Fig. 6, after removing backgrounds and resizing. Remarkably, our method, trained for 7 GPU days, is able to compete with OpenLRM, which uses hundreds of GPUs for several days [15, 17].
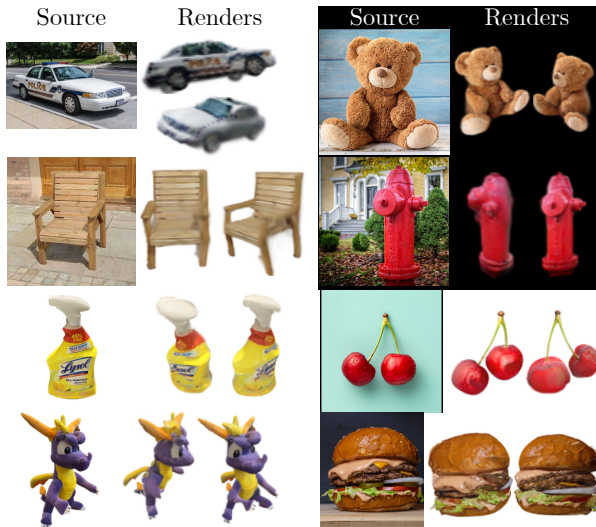
Figure 6. Our models trained on single classes (top) and on Obja-verse (bottom) can be used on **in-the-wild** Internet images (right).

Method	Relative Pose	2-view Cars	
		PSNR ↑	SSIM ↑
SRN	✗	24.84	0.92
CodeNeRF	✗	25.71	0.91
FE-NVS	✗	24.64	0.93
PixelNeRF	✓	25.66	**0.94**
Ours	✓	**26.01**	**0.94**

Table 5. Two-view reconstruction on ShapeNet-SRN Cars.

4.1.2 Two-view 3D reconstruction

We compare our multi-view reconstruction model on ShapeNet-SRN Cars by training it for two-view predictions (see Tab. 5). Prior work often relies on absolute camera pose conditioning, meaning that the model learns to rely on the canonical orientation of the object in the dataset. This limits the applicability of these models, as in practice for a new image of an object, the absolute camera pose is of course unknown. Here, only ours and PixelNeRF can deal with relative camera poses as input. Interestingly, our method shows not only better performance than PixelNeRF in both real and synthetic data but also improves over SRN, CodeN-eRF, and FE-NVS that rely on absolute camera poses.

4.1.3 Ablations

We evaluate the influence of individual components of our method, using a shorter training schedule than models in Tab. 1 for efficiency. Ablations of the multi-view model are given in the supp. mat.

We show the results of our ablation study for the single-view model in Tab. 6. We train a model (w/o image) that uses a fully connected, unstructured output instead of a Splatter Image. This model cannot transfer image informa-
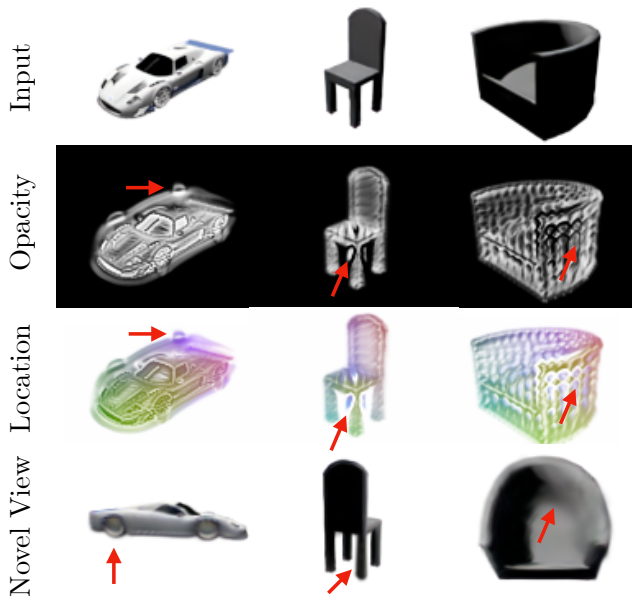

Figure 7. **Analysis.** Splatter Images represent full $360°$ of objects by allocating background pixels to appropriate 3D locations (third row) to predict occluded elements like wheels (left) or chair legs (middle). Alternatively, it predicts offsets in the foreground pixels to represent occluded chair parts (right).

tion directly to their corresponding Gaussians and does not achieve good performance. We also ablate predicting the depth along the ray by simply predicting 3D coordinates for each Gaussian. This version also suffers from its inability to easily align the input image with the output. Removing the 3D offset prediction mainly harms the backside of the object while leaving the front faces the same. This results in a lower impact on the overall performance of this component. Changing the degrees of freedom of appearance predictions (by fixing Gaussians to be isotropic or removing view-dependence) also reduced the image fidelity. Finally, removing perceptual loss (w/o $\mathcal{L}_{\text{LPIPS}}$) results in a significant worsening of LPIPS, indicating this loss is important for perceptual sharpness of reconstructions. Being able to use LPIPS in optimisation is a direct consequence of employing a fast-to-render representation and being able to render full images at training time.

Analysis. In Fig. 7, we analyse how 3D information is stored inside a Splatter Image. Since all information is arranged in an image format, we can visualise each of the modalities: opacity, depth, and location. Pixels of the input image that belong to the object tend to describe their corresponding 3D structure, while pixels outside of the object wrap around to close the object on the back.

4.2. Evaluation of reconstruction efficiency

A key advantage of the Splatter Image is its training and test time efficiency, which we assess below.

	PSNR ↑	SSIM ↑	LPIPS ↓
Full model	**22.25**	**0.90**	**0.115**
w/o image	20.60	0.87	0.152
w/o depth	21.21	0.88	0.145
w/o view dir.	21.77	0.89	0.121
isotropic	22.01	0.89	0.118
w/o offset	22.06	**0.90**	0.119
w/o $\mathcal{L}_{\text{LPIPS}}$	22.22	0.89	0.141

Table 6. **Ablations: Single-View Reconstruction.**

	RP	E ↓	R ↓	Forward ↓	Test ↓
NeRFDiff	✓	(0.031)	(0.0180)	(0.103)	(4.531)
FE-NVS	✗	(0.015)	(0.0032)	(0.028)	(0.815)
VisionNeRF	✓	0.008	2.4312	9.733	607.8
PixelNeRF	✓	**0.003**	1.8572	7.432	463.3
ViewsetDiff	✗	0.025	0.0064	0.051	1.625
Ours 2-view	✓	0.030	**0.0017**	0.037	0.455
Ours 1-view	✓	0.026	**0.0017**	**0.033**	**0.451**

Table 7. **Speed.** Time required for image encoding (E), rendering (R), the *'Forward'* time, indicative of train-time efficiency and the *'Test'* time, indicative of test-time efficiency. Our method is the most efficient in both train and test time across open-source available methods and only requires relative camera poses. 'RP' indicates if a method can operate using only relative camera poses.

Test-time efficiency. First, we assess the *'Test'* time speed, *i.e.*, the time it takes for the trained model to reconstruct an object and generate a certain number of images. We reference the evaluation protocol of the standard ShapeNet-SRN benchmark [45] and render 250 images at 128^2 resolution.

Assessing wall-clock time fairly is challenging as it depends on many factors. All measurements reported here are done on a single NVIDIA V100 GPU. We use officially released code of Viewset Diffusion [48], PixelNeRF [55] and VisionNeRF [26] and rerun those on our hardware. NeRFDiff [13] and FE-NVS [14] do not have code available, so we use their self-reported metrics. According to the authors, FE-NVS was evaluated on the same type of GPU, while NeRFDiff does not include information about the hardware used and we were unable to obtain more information. Since we could not perfectly control these experiments, the comparisons to NeRFDiff and FE-NVS are only indicative. For Viewset Diffusion and NeRFDiff we report the time for a single pass through the reconstruction network.

Tab. 7 reports the *'Encoding'* (E) time, spent by the network to compute the object's 3D representation from an image, and the *'Rendering'* (R) time, spent by the network to render new images from the 3D representation. From those, we calculate the *'Test'* time, equal to the *'Encoding'* time plus 250 *'Rendering'* time. As shown in the last column of Tab. 7, our method is more than $1000\times$ faster in testing than PixelNeRF and VisionNeRF (while achieving equal or superior quality of reconstruction in Tab. 1). Our method is also faster than voxel-based Viewset Diffusion even though it does not require knowing the absolute camera pose. The efficiency of our method is very useful to iterate quickly in research; for instance, evaluating our method on the full ShapeNet-Car validation set takes **less than 10 minutes** on a single GPU. In contrast, PixelNeRF takes **45 GPU-hours**.

Train-time efficiency. Next, we assess the efficiency of the method during training. Here, the encoding time becomes more significant because one typically renders only a few images to compute the reconstruction loss and obtain a gradient (*e.g.*, because there are only so many views available in the training dataset, or because generating more views provides diminishing returns in terms of supervision). As typical values (and as used by us in this work), we assume that the method is tasked with generating 4 new views at each iteration instead of 250 as before. We call this the *'Forward'* time and measure it the same way. As shown in the *'Forward'* column of Tab. 7, our method is $246\times$ faster at training time than implicit methods and $1.5\times$ than Viewset Diffusion, which uses an explicit representation. With this, we can train models achieving state-of-the-art quality on a **single A6000 GPU** in 7 days, while VisionNeRF requires **16 A100 GPUs** for 5 days. What is even more remarkable, we can train models on large datasets such as Objaverse on **two A6000 GPUs in 3.5 days**, while triplane-based methods such as LRM require **128 A100 GPUs for 3 days** [17].

5. Conclusion

We have presented Splatter Image, a simple method for single- or few-view 3D reconstruction. The method uses an off-the-shelf 2D image-to-image network and predicts a pseudo-image containing one colored 3D Gaussian per pixel. By combining fast inference with fast rendering via Gaussian Splatting, Splatter Image can be trained and evaluated quickly on synthetic and real benchmarks. Splatter Image achieves state-of-the-art reconstruction performance without requiring absolute/canonical camera poses at test time, is simple to implement, and can be trained and tested much faster than many alternatives.

Ethics. We use various datasets in a manner compatible with their terms. There is no processing of personal data. For further details on ethics, data protection, and copyright please see https://www.robots.ox.ac.uk/~vedaldi/research/union/ethics.html.

References

[1] Titas Anciukevicius, Zexiang Xu, Matthew Fisher, Paul Henderson, Hakan Bilen, Niloy J. Mitra, and Paul Guerrero. RenderDiffusion: Image diffusion for 3d reconstruction, inpainting and generation. In *Proc. CVPR*, 2022. 2

[2] Ang Cao, Chris Rockwell, and Justin Johnson. Fwd: Real-time novel view synthesis with forward warping and depth. In *Proc. CVPR*, 2022. 5, 6

[3] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J. Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3D generative adversarial networks. In *Proc. CVPR*, 2022. 2

[4] Eric R. Chan, Koki Nagano, Matthew A. Chan, Alexander W. Bergman, Jeong Joon Park, Axel Levy, Miika Aittala, Shalini De Mello, Tero Karras, and Gordon Wetzstein. GeNVS: Generative novel view synthesis with 3D-aware diffusion models. In *Proc. ICCV*, 2023. 3, 5, 1

[5] Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet an information-rich 3d model repository. *arXiv.cs*, abs/1512.03012, 2015. 2

[6] Anpei Chen, Zexiang Xu, Fuqiang Zhao, Xiaoshuai Zhang, Fanbo Xiang, Jingyi Yu, and Hao Su. MVSNeRF: Fast generalizable radiance field reconstruction from multi-view stereo. In *Proc. ICCV*, 2021. 2

[7] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. TensoRF: Tensorial radiance fields. In *arXiv*, 2022. 2

[8] Hansheng Chen, Jiatao Gu, Anpei Chen, Wei Tian, Zhuowen Tu, Lingjie Liu, and Hao Su. Single-stage diffusion nerf: A unified approach to 3d generation and reconstruction. In *Proc. ICCV*, 2023. 3

[9] Matt Deitke, Ruoshi Liu, Matthew Wallingford, Huong Ngo, Oscar Michel, Aditya Kusupati, Alan Fan, Christian Laforte, Vikram Voleti, Samir Yitzhak Gadre, Eli VanderBilt, Aniruddha Kembhavi, Carl Vondrick, Georgia Gkioxari, Kiana Ehsani, Ludwig Schmidt, and Ali Farhadi. Objaverse-XL: A universe of 10M+ 3D objects. *CoRR*, abs/2307.05663, 2023. 2

[10] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3D objects. In *Proc. CVPR*, 2023. 2, 5

[11] Haoqiang Fan, Hao Su, and Leonidas Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proc. ICCV*, 2017. 2

[12] Anthony G. Francis, Brandon Kinman, Krista Ann Reymann, Laura Downs, Nathan Koenig, Ryan M. Hickman, Thomas B. McHugh, and Vincent Olivier Vanhoucke. Google scanned objects: A high-quality dataset of 3d scanned household items. In *Proc. ICRA*, 2022. 2, 5, 6

[13] Jiatao Gu, Alex Trevithick, Kai-En Lin, Joshua M Susskind, Christian Theobalt, Lingjie Liu, and Ravi Ramamoorthi. Nerfdiff: Single-image view synthesis with nerf-guided distillation from 3d-aware diffusion. In *Proc. ICML*, 2023. 2, 3, 5, 8

[14] Pengsheng Guo, Miguel Angel Bautista, Alex Colburn, Liang Yang, Daniel Ulbricht, Joshua M. Susskind, and Qi Shan. Fast and explicit neural view synthesis. In *Proc. WACV*, 2022. 2, 5, 6, 8

[15] Zexin He and Tengfei Wang. Openlrm: Open-source large reconstruction models. https://github.com/3DTopia/OpenLRM, 2023. 2, 5, 6

[16] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Proc. NeurIPS*, 2020. 2

[17] Yicong Hong, Kai Zhang, Jiuxiang Gu, Sai Bi, Yang Zhou, Difan Liu, Feng Liu, Kalyan Sunkavalli, Trung Bui, and Hao Tan. LRM: Large reconstruction model for single image to 3D. In *Proc. ICLR*, 2024. 2, 5, 6, 8

[18] Ajay Jain, Matthew Tancik, and Pieter Abbeel. Putting nerf on a diet: Semantically consistent few-shot view synthesis. In *Proc. ICCV*, pages 5885–5894, 2021. 2

[19] Wonbong Jang and Lourdes Agapito. CodeNeRF: Disentangled neural radiance fields for object categories. In *Proc. ICCV*, 2021. 2, 5

[20] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. In *Proc. NeurIPS*, 2022. 5

[21] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proc. CVPR*, 2018. 2, 5, 1

[22] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian Splatting for Real-Time Radiance Field Rendering. *Proc. SIGGRAPH*, 42(4), 2023. 1, 3, 4, 5

[23] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[24] Jonáš Kulhánek, Erik Derner, Torsten Sattler, and Robert Babuška. ViewFormer: NeRF-free neural rendering from few images using transformers. In *Proc. ECCV*, 2022. 1

[25] Jiahao Li, Hao Tan, Kai Zhang, Zexiang Xu, Fujun Luan, Yinghao Xu, Yicong Hong, Kalyan Sunkavalli, Greg Shakhnarovich, and Sai Bi. Instant3D: Fast text-to-3D with sparse-view generation and large reconstruction model. *Proc. ICLR*, 2024. 3

[26] Kai-En Lin, Lin Yen-Chen, Wei-Sheng Lai, Tsung-Yi Lin, Yi-Chang Shih, and Ravi Ramamoorthi. Vision transformer for nerf-based view synthesis from a single input image. In *Proc. WACV*, 2023. 2, 5, 6, 8, 1

[27] Andrew Liu, Richard Tucker, Varun Jampani, Ameesh Makadia, Noah Snavely, and Angjoo Kanazawa. Infinite nature: Perpetual view generation of natural scenes from a single image. In *Proc. ICCV*, 2021. 2

[28] Minghua Liu, Chao Xu, Haian Jin, Linghao Chen, Mukund Varma T, Zexiang Xu, and Hao Su. One-2-3-45: Any single image to 3D mesh in 45 seconds without per-shape optimization. In *Proc. NeurIPS*, 2023. 3, 5

[29] Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. Zero-1-to-3: Zero-shot one image to 3D object. In *Proc. ICCV*, 2023. 3, 5, 1

[30] Yuan Liu, Cheng Lin, Zijiao Zeng, Xiaoxiao Long, Lingjie Liu, Taku Komura, and Wenping Wang. Syncdreamer: Learning to generate multiview-consistent images from a single-view image. *Proc. ICLR*, 2024. 3

[31] Xiaoxiao Long, Cheng Lin, Peng Wang, Taku Komura, and Wenping Wang. SparseNeuS: Fast generalizable neural surface reconstruction from sparse views. In *Proc. ECCV*, 2022. 2

[32] Luke Melas-Kyriazi, Christian Rupprecht, Iro Laina, and Andrea Vedaldi. PC2: Projection-conditioned point cloud diffusion for single-image 3d reconstruction. In *Proc. CVPR*, 2023. 2

[33] Luke Melas-Kyriazi, Christian Rupprecht, Iro Laina, and Andrea Vedaldi. Realfusion: 360° reconstruction of any object from a single image. In *Proc. CVPR*, 2023. 3

[34] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *Proc. ECCV*, 2020. 2, 3, 4

[35] Norman Müller, Yawar Siddiqui, Lorenzo Porzi, Samuel Rota Bulò, Peter Kontschieder, and Matthias Nießner. DiffRF: Rendering-guided 3D radiance field diffusion. In *Proc. CVPR*, 2023. 3

[36] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. FiLM: Visual reasoning with a general conditioning layer. In *AAAI*, 2018. 5

[37] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proc. CVPR*, 2017. 2

[38] Jeremy Reizenstein, Roman Shapovalov, Philipp Henzler, Luca Sbordone, Patrick Labatut, and David Novotny. Common objects in 3d: Large-scale learning and evaluation of real-life 3d category reconstruction. In *Proc. ICCV*, 2021. 2, 5

[39] Konstantinos Rematas, Ricardo Martin-Brualla, and Vittorio Ferrari. ShaRF: Shape-conditioned radiance fields from a single view. In *Proc. ICML*, 2021. 2

[40] Chris Rockwell, David F. Fouhey, and Justin Johnson. Pixelsynth: Generating a 3d-consistent experience from a single image. In *Proc. ICCV*, 2021. 2

[41] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proc. CVPR*, 2022. 2

[42] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Proc. MICCAI*, 2015. 2

[43] Darius Rückert, Linus Franke, and Marc Stamminger. Adop: Approximate differentiable one-pixel point rendering. In *ACM Trans. on Graphics (TOG)*, 2022. 3

[44] Yichun Shi, Peng Wang, Jianglong Ye, Long Mai, Kejie Li, and Xiao Yang. Mvdream: Multi-view diffusion for 3d generation. *arXiv:2308.16512*, 2023. 3, 5

[45] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Proc. NeurIPS*, 2019. 2, 5, 8, 1

[46] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *Proc. ICLR*, 2021. 4, 5

[47] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proc. CVPR*, 2022. 2

[48] Stanislaw Szymanowicz, Christian Rupprecht, and Andrea Vedaldi. Viewset diffusion: (0-)image-conditioned 3D generative models from 2D data. In *Proc. ICCV*, 2023. 2, 3, 5, 8

[49] Ayush Tewari, Tianwei Yin, George Cazenavette, Semon Rezchikov, Joshua B. Tenenbaum, Frédo Durand, William T. Freeman, and Vincent Sitzmann. Diffusion with forward models: Solving stochastic inverse problems without direct supervision. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2023. 3, 1

[50] Dmitry Tochilkin, David Pankratz, Zexiang Liu, Zixuan Huang, Adam Letts, Yangguang Li, Ding Liang, Christian Laforte, Varun Jampani, and Yan-Pei Cao. TripoSR: fast 3D object reconstruction from a single image. 2403.02151, 2024. 2

[51] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul P. Srinivasan, Howard Zhou, Jonathan T. Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas A. Funkhouser. Ibrnet: Learning multi-view image-based rendering. In *Proc. CVPR*, 2021. 2

[52] Daniel Watson, William Chan, Ricardo Martin-Brualla, Jonathan Ho, Andrea Tagliasacchi, and Mohammad Norouzi. Novel view synthesis with diffusion models. In *Proc. ICLR*, 2023. 3, 5

[53] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. Synsin: End-to-end view synthesis from a single image. In *Proc. CVPR*, 2020. 2

[54] Yinghao Xu, Hao Tan, Fujun Luan, Sai Bi, Peng Wang, Jiahao Li, Zifan Shi, Kalyan Sunkavalli, Gordon Wetzstein, Zexiang Xu, and Kai Zhang. DMV3D: Denoising multi-view diffusion using 3D large reconstruction model. In *Proc. ICLR*, 2024. 3, 5

[55] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. PixelNeRF: Neural radiance fields from one or few images. In *Proc. CVPR*, 2021. 2, 5, 6, 8, 1

[56] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proc. CVPR*, 2018. 2, 4

[57] Chuanxia Zheng and Andrea Vedaldi. Free3d: Consistent novel view synthesis without 3d representation. In *Proc. CVPR*, 2024. 5, 6, 1

[58] Linqi Zhou, Yilun Du, and Jiajun Wu. 3d shape generation and completion through point-voxel diffusion. In *Proc. ICCV*, 2021. 2

[59] Zhizhuo Zhou and Shubham Tulsiani. Sparsefusion: Distilling view-conditioned diffusion for 3d reconstruction. In *Proc. CVPR*, 2023. 5, 1

[60] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus H. Gross. EWA volume splatting. In *Proc. IEEE Visualization Conference,*, 2001. 3

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 'Teddy-

bear' 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 pre-processing 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 $\hat{s}_{\text{small}} = -5$: $\mathcal{L}_{\text{small}} = (\sum_i -\hat{s}_i \mathbb{1}(\hat{s}_i < \hat{s}_{\text{small}}))/(\sum_i \mathbb{1}(\hat{s}_i < \hat{s}_{\text{small}}))$.

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

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 ×50 cheaper to train than LRM. Memory denotes the memory capacity of the GPU. * denotes estimates.

in Appendix B.2 is available as default in the official implementation. As in original work, we train for $400,000$ iterations.

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 ×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 $\tilde{R}_q = R R_q$. 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 $\tilde{q} = pq$.