arXiv:2312.13150v1 [cs.CV] 20 Dec 2023

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

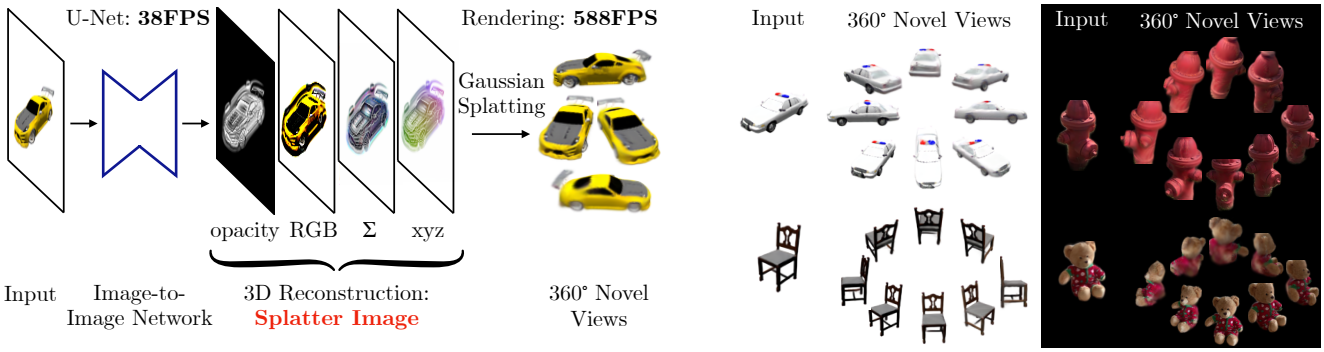

Figure 1. The **Splatter Image** is an ultra-fast method for single- and few-view 3D reconstruction. It works by applying an image-to-image neural network to the input view and obtain, as output, another image that holds the parameters of one coloured 3D Gaussian per pixel. The resulting Gaussian mixture can be rendered very quickly into an arbitrary view of the object by using Gaussian Splatting.

Abstract

We introduce the Splatter Image[1]*, an ultra-fast approach for monocular 3D object reconstruction which operates at 38 FPS. Splatter Image is based on Gaussian Splatting, which has recently brought real-time rendering, fast training, and excellent scaling to multi-view reconstruction. For the first time, we apply Gaussian Splatting in a monocular reconstruction setting. Our approach is learning-based, and, at test time, reconstruction only requires the feed-forward evaluation of a neural network. The main innovation of Splatter Image is the surprisingly straightforward design: it uses a 2D image-to-image network to map the input image to one 3D Gaussian per pixel. The resulting Gaussians thus have the form of an image, the Splatter Image. We further extend the method to incorporate more than one image as input, which we do by adding cross-view attention. Owning to the speed of the renderer (588 FPS), we can use a single GPU for training while generating entire images at each iteration in order to optimize perceptual metrics like LPIPS. On standard benchmarks, we demonstrate not only fast reconstruction but also better results than recent and much more expensive baselines in terms of PSNR, LPIPS, and other metrics.*

[1]Website: szymanowiczs.github.io/splatter-image

1. Introduction

Single-view 3D reconstruction poses a fundamental challenge in computer vision. In this paper, we contribute Splatter Image, a method that achieves ultra-fast single-view reconstruction of the 3D shape and appearance of objects. This method uses Gaussian Splatting [14] as the underlying 3D representation, taking advantage of its rendering quality and speed. It works by predicting a 3D Gaussian for each of the input image pixels, using an image-to-image neural network. Remarkably, the 3D Gaussians in the resulting 'Splatter Image' provide 360° reconstructions (Fig. 1) of quality matching or outperforming much slower methods.

The key challenge in using 3D Gaussians for monocular reconstruction is to design a network that takes an image of an object as input and produces as output a corresponding Gaussian mixture that represents all sides of it. We note that, 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 a container for the 3D Gaussians, so that each pixel contains in turn the parameters of one Gaussian, including its opacity, shape, and colour.

The advantage of storing sets 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 [32] as those have demonstrated excellent performance in image generation [31]. In our case, their ability to capture small image details [44] helps to obtain higher-quality reconstructions.

Since the 3D representation in Splatter Image is a mixture of 3D Gaussians, it enjoys the rendering speed and memory efficiency of Gaussian Splatting, which is advantageous both in inference and training. In particular, rendering stops being a training bottleneck and we can afford to generate complete views of the object to optimize perceptual metrics like LPIPS [45]. Possibly even more remarkably, the efficiency is such that our model can be trained on a *single GPU* on standard benchmarks of 3D objects, whereas alternative methods typically require distributed training on several 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 reference. Furthermore, we allow different views to communicate during prediction by injecting lightweight cross-view attention layers in the architecture.

Empirically, we study several properties of Splatter Image. First, we note 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 coded in the 2D image by allocating different Gaussians in a given 2D neighbourhood to different parts of the 3D object. We also show that many Gaussians are in practice inactivated by setting their opacity to zero, and can thus be culled in post-processing. We further validate Splatter Image by comparing it to alternative, slower reconstructors on standard benchmark datasets like ShapeNet [4] and CO3D [28]. Compared to these slower baselines, Splatter Image is not only competitive in terms of quality, but in fact state-of-the-art in several cases, improving both reconstruction PSNR and LPIPS. We argue that this is because the very efficient design allows training the model more 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; (3) to also extend the method to multi-view reconstruction; (4) and to obtain state-of-the-art reconstruction performance in standard benchmarks in terms of reconstruction quality and speed.

2. Related work

Representations for single-view 3D reconstruction. In recent years, implicit representations like NeRF [24] have dominated learning-based few-view reconstruction. Some works have approached this problem by parameterising the MLP in NeRF using global [12, 29], local [44] or both global and latent codes [16]. However, implicit representations, particularly MLP-based ones, are notoriously slow to render, up to 2s for a single 128×128 image.

Some follow-up works [8, 38] have used faster implicit representations based on voxel grids that encode opacities and colours directly [8, 38] — similar to DVGO [37], they can thus achieve significant speed-ups. However, due to their voxel-based representation, they scale poorly with resolution. They also assume the knowledge of the absolute viewpoint of each object image.

A hybrid implicit-explicit triplane representation [2] has been proposed as a compromise between rendering speed and memory consumption. Triplanes can be predicted by networks in the camera view space, instead of a global reference frame, thus allowing reconstruction in the view-space [7]. While they are not as fast to render as explicit representations, they are fast enough to be effectively used for single-view reconstruction [1, 7].

Finally, several works predict multi-view images directly [3, 19, 39, 42]. 3D models can then be obtained with test-time multi-view optimisation. The main disadvantage of image-to-image novel view generators is that they exhibit noticeable flicker and are 3D inconsistencies, thus limiting the quality of obtained reconstructions. In addition, test-time optimisation is an additional overhead, limiting the overall reconstruction speed.

In contrast to these works, our method predicts a mixture of 3D Gaussians in a feed-forward manner. As a result, our method is fast at inference and achieves real-time rendering speeds while achieving state-of-the-art image quality across multiple metrics on the standard single-view reconstruction benchmark ShapeNet-SRN [35].

When more than one view is available at the input, one can learn to interpolate between available views in the 3D space to estimate the scene geometry [5, 21], learn a view interpolation function [41] or optimize a 3D representation of a scene using semantic priors [11]. 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 [9] adapted PointNet [27] to take image encoding as input and trained point cloud prediction networks using 3D point cloud supervision. PVD [46] and PC^2 [23] extended this approach using Diffusion Models [10] 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 [17, 30, 43] train networks for Novel Views Synthesis purely from videos and use point clouds as intermediate 3D representations for conditioning 2D inpainting or generation net-

works. 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. ADOP [33] used points with a fixed radius and also used 2D inpainting networks for hole-filling. Gaussian Splatting [14] used non-isotropic 3D Gaussians with variable scale, thus removing the need for 2D inpainting networks. While showing high-quality results, Gaussian Splatting 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 several authors argue that it should be tackled as a conditional generative problem. Diffusion Models have been employed for conditional Novel View Synthesis [3, 18, 19, 42]. Due to generating images without underlying geometries, the output images exhibit noticeable flicker. This can be mitigated by simultaneously generating multi-view images [20, 34], or guaranteed by reconstructing a geometry at every step of the denoising process [38, 40]. Other works build and use a 3D [6, 25] or 2D [7, 22] 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 [20, 34, 38, 40]. Our method is capable of few-view 3D reconstruction, thus it is complimentary to these generative methods and could lead to improvements in generation speed and quality.

3. Method

We provide background information on Gaussian Splatting in Sec. 3.1, and then describe the Splatter Image in Secs. 3.2 to 3.6.

3.1. Overview of Gaussian Splatting

A *radiance field* [24] is given by the opacity function $\sigma(\boldsymbol{x}) \in \mathbb{R}_+$ and the colour function $c(\boldsymbol{x}, \boldsymbol{\nu}) \in \mathbb{R}^3$, where $\boldsymbol{\nu} \in \mathbb{S}^2$ is the viewing direction of the 3D point $\boldsymbol{x} \in \mathbb{R}^3$.

The field is rendered onto 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 pixel $\boldsymbol{u}$:

$$I(\boldsymbol{u}) = \int_0^\infty c(\boldsymbol{x}_\tau, \boldsymbol{\nu})\sigma(\boldsymbol{x}_\tau) e^{-\int_0^\tau \sigma(\boldsymbol{x}_\mu)\, d\mu}\, d\tau. \quad (1)$$

Gaussian Splatting [48] represents these two functions as a mixture θ of G colored 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 $\sigma_i \in \mathbb{R}_+$ and a view-dependent colour $c_i(\boldsymbol{v}) \in \mathbb{R}^3$. Together, they define a radiance field as follows:

$$\sigma(\boldsymbol{x}) = \sum_{i=1}^{G} \sigma_i g_i(\boldsymbol{x}), \quad c(\boldsymbol{x}, \boldsymbol{\nu}) = \frac{\sum_{i=1}^{G} c_i(\boldsymbol{\nu})\sigma_i g_i(\boldsymbol{x})}{\sum_{j=1}^{G} \sigma_i g_i(\boldsymbol{x})}. \quad (2)$$

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

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

Gaussian Splatting [14, 48] provides a very fast differentiable renderer $I = \mathcal{R}(\theta, \pi)$ that approximates Eq. (1), mapping the mixture θ to a corresponding image I given a viewpoint π.

3.2. The Splatter Image

The renderer $\mathcal{R}$ maps the set of 3D Gaussians θ to an image I. We now seek for an inverse function $\theta = \mathcal{S}(I)$ which reconstructs the mixture of 3D Gaussians θ from an image I, thereby performing single-view 3D reconstruction.

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 that outputs an 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, and outputs directly a $H \times W \times K$ tensor, 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

$$\boldsymbol{\mu} = \begin{bmatrix} u_1 d + \Delta_x \\ u_2 d + \Delta y \\ d + \Delta_z \end{bmatrix}. \quad (3)$$

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 explains in detail how these quantities are predicted.

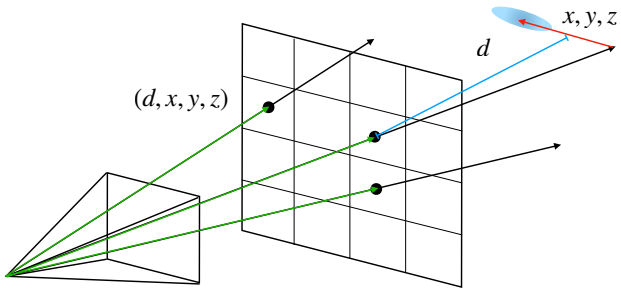

Figure 2. **Predicting locations.** The location of each Gaussian is parameterised by depth d and a 3D offset $\Delta = (\Delta_x, \Delta_y, \Delta_z)$. The 3D Gaussians are projected to depth d (blue) along camera rays (green) and moved by the 3D offset Δ (red).

Discussion. One may wonder how this design can predict a full 360° reconstruction of the object when the reconstruction is aligned to only one of its views. In practice, the network learns to use some of the Gaussians to reconstruct the given view, and some to reconstruct unseen portions of the scene, automatically. Note also that 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.

Our design can also be seen as an extension of depth prediction networks, where the network is only tasked with predicting the depth of each pixel. Here, we also predict invisible parts of the geometry, as well as the appearance.

3.3. Learning formulation

Learning to predict the Splatter Image is simple and efficient — we carry it out on a single GPU using at most 20GB of memory at training time in all our single-view reconstruction experiments. For training, we assume a multi-view dataset, either real or synthetic. At a minimum, this 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 the target camera. Then we simply feed the source I as input to Splatter Image, and minimize the average reconstruction loss of target view J:

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

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 [24], 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 [45], which do not decompose into per-pixel losses. In practice, we experiment with a combination of such losses.

Scale normalization. Estimating the scale of an object from a single view is ambiguous, and this ambiguity will be challenging to resolve for a network trained with a loss like one in Eq. (4). In synthetic datasets this is not an issue because all objects are at a fixed distance from the camera and rendered with the same camera intrinsics, thus removing the ambiguity. However, in real datasets like CO3D [28], this ambiguity is present. We apply pre-processing following the protocol of [38], thus approximately fixing the scale of all objects.

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, $j \in \{1, \dots, N\}$ are provided, we can apply network $\mathcal{S}$ multiple times to obtain multiple Splatter Images M_j, one per view.

Warping 3D Gaussians. 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 g of parameters $(\sigma, \boldsymbol{\mu}, \Sigma, c)$ maps to Gaussian $\tilde{g}$ of parameters $(\sigma, \tilde{\boldsymbol{\mu}}, \tilde{\Sigma}, \tilde{c})$ where

$$\tilde{\boldsymbol{\mu}} = R\boldsymbol{\mu} + T, \quad \tilde{\Sigma} = R\Sigma R^\top, \quad \tilde{c} = c.$$

We use the symbol $\phi[\theta]$ to denote the Gaussian Splat 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.

Predicting Composite 3D Gaussian Mixtures. 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* [14] 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.

Warping the colour model. 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 spherical harmonics are closed under rotation; in fact, each order is closed. 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 [36]. 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 $\sigma = \operatorname{sigmoid}(\hat{\sigma})$. The depth is obtained as $d = (z_{\text{far}} - z_{\text{near}})\operatorname{sigmoid}(\hat{d}) + z_{\text{near}}$. The mean $\boldsymbol{\mu}$ is then obtained using Eq. (3). Following [14], the covariance is obtained as $\Sigma = R(\boldsymbol{q})\operatorname{diag}(\exp\hat{\boldsymbol{s}})^2 R(\boldsymbol{q})^\top$ 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 consider 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 [26] embeddings.

Second, we add cross-attention layers to allow communication between the features of different views. We do so in a manner similar to [34], but only at the lowest UNet resolution, which maintains the computational cost very low.

4. Experiments

We begin our experiments with an evaluation of the speed of the method, followed by an evaluation of single-view reconstruction on two synthetic and two real object categories, and finish with an assessment of multi-view reconstruction.

Datasets. The standard benchmark for evaluating single-view 3D reconstruction is ShapeNet-SRN [35]. We train our method in the single-class setting and report results on the "Car" and "Chair" classes, following prior work. We use the images, camera intrinsics, camera poses and data splits as provided by the dataset and train our method using *relative* camera poses: the reconstruction is done in the view space of the conditioning camera.

In addition, we challenge our method with two classes from the CO3D dataset: Hydrants and Teddybears. Similarly to recent methods [3, 40] we take the largest center-crop in the original images and resize to 128×128 resolution with Lanczos interpolation. Similarly to many single- and few-view reconstruction methods [44, 47**?**] we also remove backgrounds. To reduce the scale ambiguity in single-view reconstruction, we follow the pre-processing protocol of Viewset Diffusion [38] and adjust bounds z_{far} and z_{near}. See supplement for details.

Finally, we use the ShapeNet-SRN Cars dataset for the evaluation of the two-view reconstruction quality.

Baselines. For ShapeNet, we compare against implicit [12, 16, 35, 44], hydrid implicit-explicit [7] and explicit methods [8, 38] methods. While Viewset Diffusion [38] and NeRFDiff [7] were introduced as diffusion-based generative methods, one can use their deterministic variants by using their reconstruction network in a single forward pass. Since we are proposing a deterministic reconstruction method, we do not compare to methods that employ Score Distillation [47] or feed-forward diffusion models [3, 42]. For CO3D we compare against PixelNeRF which we train for 400,000 iterations with their officially released code on data preprocessed with the aforementioned protocol.

Implementation details can be found in the supp. mat.

4.1. Assessing quality

We measure the quality of our reconstructions by evaluating the quality of Novel View Synthesis, in line with related works [16, 44]. We perform reconstruction from a given source view and render the 3D shape to unseen target views. We measure the Peak Signal-to-Noise Ratio (PSNR), Structural Similarity (SSIM) and perceptual quality (LPIPS). We follow standard protocol in the ShapeNet-SRN datasets: in

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.

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 unseen views are used for the computation of metrics. We also evaluate single-view reconstruction quality in CO3D. We use the first frame as conditioning 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.

4.1.1 Single-view 3D reconstruction

ShapeNet-SRN Cars and Chairs. In Tab. 1 we compare the single-view reconstruction quality on the ShapeNet-SRN benchmark. Our method outperforms all deterministic reconstruction methods in Structural Similarity (SSIM) and Perceptual Quality (LPIPS), indicating it provides sharper reconstructions. Moreover, it also outperforms all deterministic baselines on the car dataset on Peak Signal-to-Noise Ratio (PSNR), suggesting that the reconstructions are also more accurate. In addition to strong quantitative performance, our method only needs relative camera poses (does not require canonical camera poses) and is more efficient in both training and testing (see Sec. 4.2).

In Fig. 3 we qualitatively show that our method is able to capture interesting and thin geometries and capture the details of the conditioning views well. Fig. 4 shows that our method produces sharper and more accurate reconstructions than prior works in challenging situations with limited visibility and thin structures.

CO3D. Qualitative results and a comparison to PixelNeRF are shown in Fig. 5. Splatter Image predicts sharper images while also being 1000× faster. Quantitatively (Tab. 2) our model outperforms PixelNeRF for both categories on SSIM and LPIPS and results in PSNR on par with PixelNeRF.

Figure 3. **ShapeNet-SRN.** Our method is able to represent difficult geometries (*e.g.*, windshield in yellow car), and preserve details of the conditioning image (front of red car), occluded parts (left chair), thin structures (middle chair) and complex shapes (right chair).

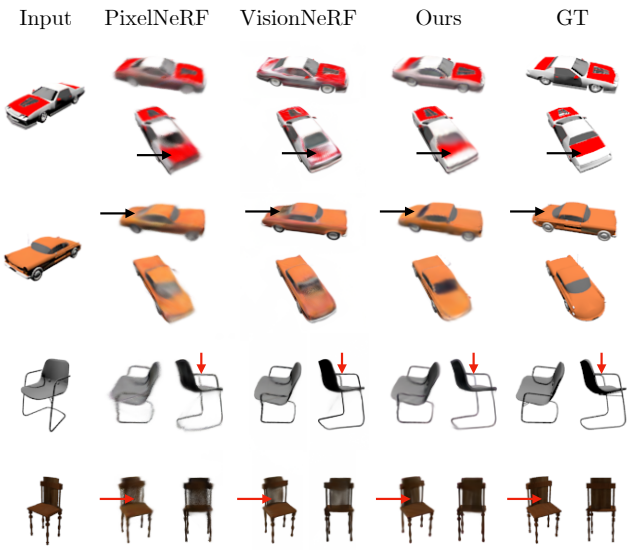

Figure 4. **ShapeNet-SRN Comparison.** Our method outputs more accurate reconstructions (cars' backs, top chair) and better represents thin regions (bottom chair).

Object	Method	PSNR ↑	SSIM ↑	LPIPS ↓
Hydrant	PixelNeRF	21.76	0.78	0.207
Hydrant	**Ours**	**22.10**	**0.81**	**0.148**
Teddybear	PixelNeRF	**19.57**	0.67	0.297
Teddybear	**Ours**	19.51	**0.73**	**0.236**

Table 2. **CO3D: Single-View.** Our method outperforms PixelNeRF on this challenging benchmark across most metrics.

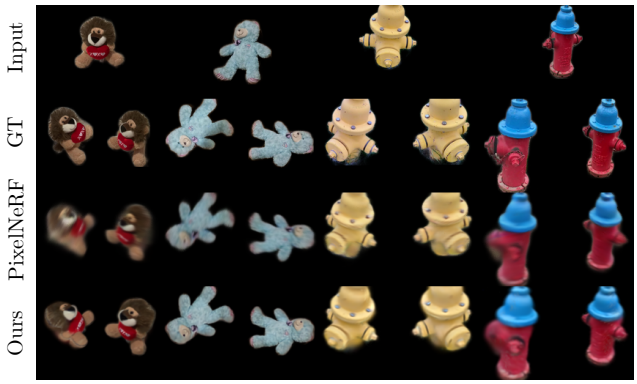

Figure 5. **CO3D Hydrants and Teddybears.** Our method outputs sharper reconstructions than PixelNeRF while being 100x faster in inference.

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 3. 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 a model for two-view predictions (see Tab. 3). 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 but also improves over SRN, CodeNeRF, and FE-NVS that rely on absolute camera poses.

4.1.3 Ablations

We evaluate the influence of individual components of our method on the final performance. Due to computational cost, we train these models at a shorter training schedule for 100k iterations with $\mathcal{L}_2$ and further 25k with $\mathcal{L}_2$ and $\mathcal{L}_{\text{LPIPS}}$.

Single-View Model. We show the results of our ablation study for the single-view model in Tab. 4. We test the impact of all components. We train a model (w/o image) that uses a fully connected, unstructured output instead of a Splatter Image. This model cannot transfer image information 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 the predicted covariance matrix to be isotropic or removing view-dependent appearance (by removing Spherical Harmonics prediction) also reduced the image fidelity. Finally, removing perceptual loss (w/o $\mathcal{L}_{\text{LPIPS}}$) results in a very small drop in PSNR but 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.

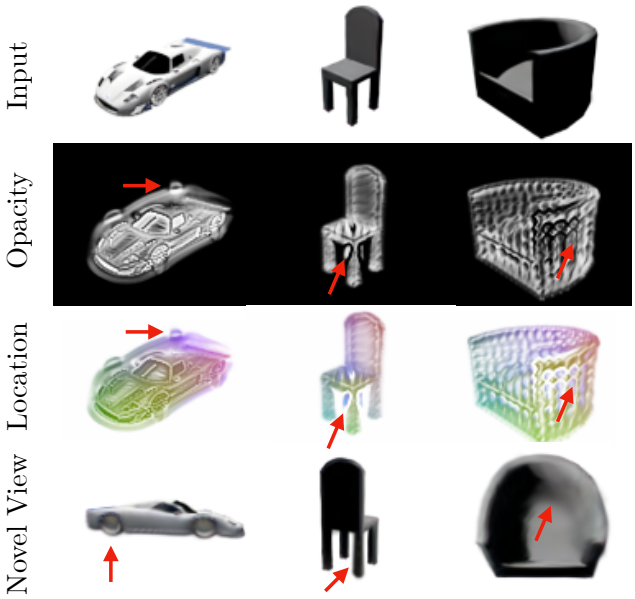

Figure 6. **Analysis.** Splatter Images represent full 360° of objects by allocating background pixels to appropriate 3D locations (third row) to predict occluded elements like wheels (left) or chair legs (middle). Alternatively, it predicts offsets in the foreground pixels to represent occluded chair parts (right).

Multi-View Model. Table 5 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.

Analysis. In Fig. 6, 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

	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 4. **Ablations: Single-View Reconstruction.**

	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 5. **Ablations: Multi-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 6. **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.

modalities: opacity, depth, and location offset. 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. Assessing efficiency

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

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 [35] 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 [38], PixelNeRF [44] and VisionNeRF [16] and rerun those on our hardware. NeRFDiff [7] and FE-NVS [8] do not have code available, so we use their self-reported metrics. FE-NVS was evaluated ostensibly 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. 6 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. 6, 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 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. 6, 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.

5. Conclusion

We have presented Splatter Image, a simple and fast method for single- or few-view 3D reconstruction. The method processes images efficiently using an off-the-shelf 2D CNN architecture 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 performance, does not require canonical camera poses, is simple to implement and offers significant computational savings in both training and inference.

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.

Acknowledgements. S. Szymanowicz is supported by an EPSRC Doctoral Training Partnerships Scholarship (DTP) EP/R513295/1 and the Oxford-Ashton Scholarship. A. Vedaldi is supported by ERC-CoG UNION 101001212.

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*, 2023. 2

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

[3] 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. 2, 3, 5, 1

[4] 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

[5] 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

[6] 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 *ICCV*, 2023. 3

[7] 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*, pages 11808–11826. PMLR, 2023. 2, 3, 5, 8

[8] 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, 8

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

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

[11] 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

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

[13] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. In *Proc. NeurIPS*, 2022. 1

[14] 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

[15] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1

[16] 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, 8

[17] 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

[18] 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. *arXiv.cs*, abs/2306.16928, 2023. 3

[19] 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. 2, 3

[20] 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. *arXiv preprint arXiv:2309.03453*, 2023. 3

[21] 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

[22] 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

[23] Luke Melas-Kyriazi, Christian Rupprecht, and Andrea Vedaldi. PC2: Projection-conditioned point cloud diffusion for single-image 3d reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 2

[24] 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

[25] 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

[26] 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

[27] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016. 2

[28] 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. CVPR*, 2021. 2, 4

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

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

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

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

[33] Darius Rückert, Linus Franke, and Marc Stamminger. Adop: Approximate differentiable one-pixel point rendering. 2022. 3

[34] 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

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

[36] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *Proc. ICLR*, 2021. 5, 1

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

[38] Stanislaw Szymanowicz, Christian Rupprecht, and Andrea Vedaldi. Viewset diffusion: (0-)image-conditioned 3D generative models from 2D data. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2023. 2, 3, 4, 5, 8, 1

[39] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Multi-view 3d models from single images with a convolutional network. In *ECCV*, 2016. 2

[40] 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. *arXiv.cs*, abs/2306.11719, 2023. 3, 5, 1

[41] 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

[42] Daniel Watson, William Chan, Ricardo Martin-Brualla, Jonathan Ho, Andrea Tagliasacchi, and Mohammad Norouzi. Novel view synthesis with diffusion models. In *Proc. ICLR*, 2023. 2, 3, 5

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

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

[45] 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

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

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

[48] 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

A. Additional results

The project website szymanowiczs.github.io/splatter-image.html includes additional results from our method on 4 object classes, comparisons of our method to baselines, a demonstration of running our method on a video frame-by-frame and a recording with a short summary of Splatter Image. Code is available via the project website. Moreover, we present static comparisons of our method to PixelNeRF [44] and VisionNeRF on ShapeNet-SRN Cars and Chairs in Fig. 7, as well as static comparisons of our method to PixelNeRF on CO3D Hydrants and Teddybears in Fig. 8.

B. CO3D Data preprocessing

Image center-cropping. The renderer we use assumes the image's principal point to be at the image centre. We pre-process image data to satisfy this constraint, similarly to [3, 40]. We take the largest square crop in the original images centred on the principal point. We then reshape the images to 128×128 resolution. We adjust the focal length accordingly with the resulting transformations.

Addressing scale/depth ambiguity. Single-view reconstruction is an ambiguous problem. In particular, given a single image, the scale of the object and its distance from the camera are ambiguous. First, we normalise the scale of all scenes in the dataset following the protocol from [38] which normalises each scene using the ground truth point cloud so that the objects have the same scale. However, even after this transformation estimating the distance between the object and the camera from visual information alone remains an ambiguous and 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 scalar bound that is larger than the post-normalization object size. In PixelNeRF, we follow the default setting for turntable-style datasets and 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.

C. Implementation details.

We train our model (based on SongUNet [36]) 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 [13]. We use the Adam optimizer [15] with learning rate 2×10^{-5} and batch size 8. For rasterization, we use the Gaussian Splatting implementation of [14]. After 800,000 iterations we decrease the learning rate by a factor of 10 and train for a further 100,000 (Cars, Hydrants, Teddybears) 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.

In addition, for CO3D we 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 $\hat{s}_{\text{small}} = -5$: $\mathcal{L}_{\text{small}} = (\sum_i -\hat{s}_i \mathbb{1}(\hat{s}_i < \hat{s}_{\text{smal l}}))/(\sum_i \mathbb{1}(\hat{s}_i < \hat{s}_{\text{small}}))$.

D. Implementation details — PixelNeRF.

We use the official PixelNeRF implementation [44]. 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 is available as default in the official implementation.

E. 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. 7. We report the compute resources reported in original works, where available. NeRFDiff only reports the resources needed to train their 'Base' models and we were unable to obtain more information 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

Figure 7. **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 8. **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).

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	A6000	48G	1	7	7

Table 7. **Training resources**. Ours, Viewset Diffusion and PixelNeRF have significantly lower compute costs than VisionNeRF and NeRFDiff. Memory denotes the memory capacity of the GPU. * denotes estimates.

of Viewset Diffusion and PixelNeRF, while we achieve better performance and do not require absolute camera poses.

F. 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 $\tilde{q} = pq$.