Lightplane: Highly-Scalable Components for Neural 3D Fields

Ang Cao[1, 2] Justin Johnson[2] Andrea Vedaldi[1] David Novotny[1]
Meta AI[1] University of Michigan, Ann Arbor[2]

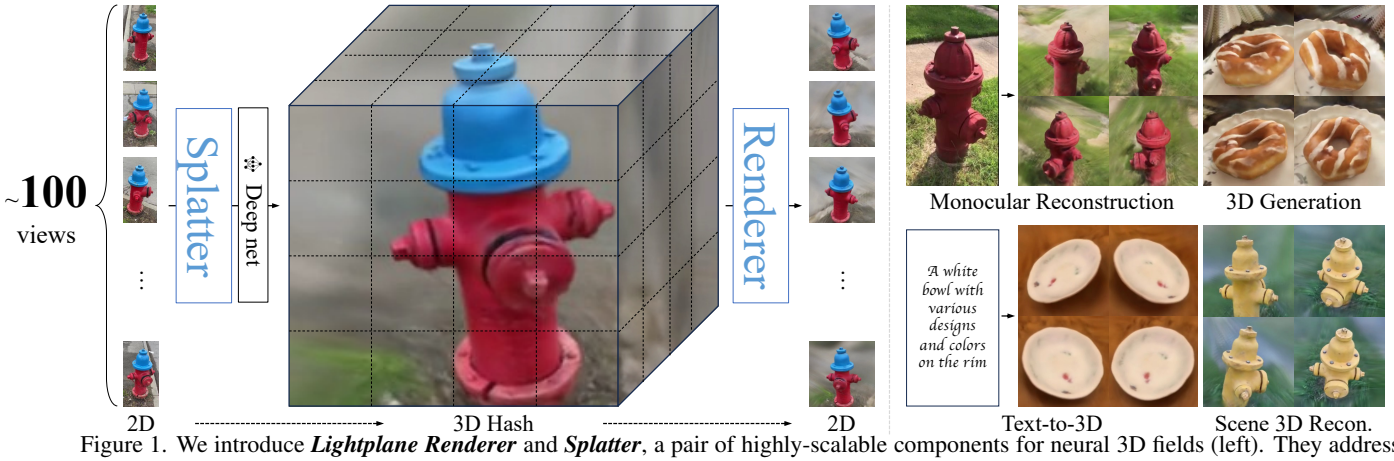

Figure 1. We introduce ***Lightplane Renderer*** and ***Splatter***, a pair of highly-scalable components for neural 3D fields (left). They address the key memory bottleneck of 2D-3D mapping (*i.e.* rendering and lifting), and reduce memory usage by over **three orders of magnitude**, which dramatically increases the number of images that can be processed. We showcase how they can boost various 3D applications (right).

Abstract

Contemporary 3D research, particularly in reconstruction and generation, heavily relies on 2D images for inputs or supervision. However, current designs for these 2D-3D mapping are memory-intensive, posing a significant bottleneck for existing methods and hindering new applications. In response, we propose a pair of highly scalable components for 3D neural fields: Lightplane Renderer and Splatter, which significantly reduce memory usage in 2D-3D mapping (over $\mathbf{1000}\times$*). These innovations enable the processing of vastly more and higher resolution images with significantly small memory and computational costs. We demonstrate their utility across various applications, from optimizing with image-level losses to enabling a versatile pipeline for scaling 3D reconstruction and generation.*[1]

1. Introduction

Recent advancements in neural rendering and generative modeling have significantly advanced 3D reconstruction and generation. However, these methods often rely heavily on 2D images for inputs or supervision, necessitating effective mapping between 2D and 3D spaces. For instance, Neural Radiance Fields (NeRFs) [50] rely on photometric loss on rendered images without 3D supervision. Similarly, many methods [11, 13, 91] employ 2D images as inputs and lift them into 3D space for feedforward processing. This 2D-to-3D mapping is crucial in 3D research due to the scarcity of 3D data for training versatile models and the relative ease of acquiring large-scale 2D image datasets.

Despite its crucial role and widespread use, the current 2D-3D mapping incurs a high resource cost, especially in *neural 3D fields* with volumetric rendering, which underpins many of the most powerful 3D representations. *Neural 3D fields* are continuous functions that assign values (e.g., density, color) to *any* point in 3D space, regardless of the presence of a physical surface. This makes them powerful and flexible, avoiding the need for initialization in point rendering [37] or topology constraints for meshes [46]. However, they require operations across numerous 3D points spanning an entire volume. While these operations can be relatively simple (*e.g.* computing a tiny MLP or average input features), performing them at all points in a differentiable manner is *extremely* memory intensive as all intermediate values must be kept in memory for backpropagation.

Even in modest settings, existing 2D-3D mapping methods can easily consume dozens of GBs of memory, taking a significant portion of available resources. This substantial

[1] work done during internship

memory demand poses a critical bottleneck for many 3D models and a formidable barrier to new applications. For example, the memory requirements to render even a single low-resolution image of a neural 3D field can be prohibitive enough to prevent the application of image-level losses such as LPIPS [95] or SDS [57]. Omitting such losses leads to a massive performance loss, as e.g. demonstrated by the SOTA Large Reconstruction Model [3, 29]. Additionally, memory inefficiencies limit the number of input views and the size of the 3D model that can be used.

We aim to solve this bottleneck by introducing two highly scalable components for neural 3D fields: *Lightplane Renderer* and *Splatter*. These innovations enable 2D-3D mapping with more than $\mathbf{1000}\times$ memory saving while maintaining comparable speed. Moreover, their designs can be easily extended to various 'hashed' 3D representations (*e.g.* voxel grids, triplanes and more), providing a universal and practical method for 2D-3D mapping across various 3D representations, especially for lifting images to 3D.

Lightplane Renderer renders 2D images from 3D models via the standard emission-absorption equations popularized by NeRF [50]. To save memory, we creatively re-configure inner computations and fuse operations over casted rays instead of 3D points. Specifically, it sequentially calculates features (*e.g.*, colors) and densities of points along the ray, updating rendered pixels and transmittance on-the-fly without storing intermediate tensors. This design significantly saves memory at the cost of a challenging backpropagation, which we solve by efficiently recomputing forward activations as needed. Note that the latter differs from the standard "checkpointing" trick, whose adoption here would be of little help, as it still entails caching many intermediate ray-point values as we march along each ray.

Conversely, *Lightplane Splatter* lifts (splats) 2D inputs into 3D structures, which are then processed by neural networks. This innovative design directly stores splatted features into the hash structure underlying the 3D model, without emitting a value for each 3D point. Unlike existing methods [11, 92] that require 3D positions for projection, *Splatter* can directly lift inputs into 3D hash structures even without clearly defined 3D positions, such as triplane or hash-table. Like *Lightplane Renderer*, its computations are reconfigured for significant memory savings. We implemented these components in Triton [75], a GPU programming language that is efficient, portable, and easy to modify. The code will be released as an open-source package.

Like flash-attention [17], our components are designed as building blocks to boost a variety of 3D models and applications, from optimizing with image-level loss to improving performance of the state-of-the-art Large Reconstruction Model [29]. Empowered by the *Lightplane*, we devise a pipeline taking up to input 100 images, significantly scaling the communication between 2D and 3D. We extensively evaluate on the CO3Dv2 dataset, reporting significant performance improvements in color and geometry accuracy for 3D reconstruction, and better 3D generation in FID/KID.

2. Related Work

3D reconstruction using neural 3D fields. Traditional 3D reconstruction models represented shapes as meshes [23, 79], point clouds [19, 88], or voxel grids [16, 22]. With the introduction of NeRF [50], however, the focus has shifted to *implicit* 3D representations, often utilizing MLPs to represent occupancy and radiance functions defined on a 3D domain. NeRF has been refined in many ways [4–6, 38, 43, 78, 94], including replacing the opacity function with a signed-distance field to improve the reconstruction of surfaces [7, 44, 56, 63, 67, 80, 83, 89, 93].

Storing an entire scene in a single MLP requires evaluating a complex function at every 3D point, which is very costly for both time and memory. Many authors propose represent radiance fields with smaller, more local components to improve speed, including using point clouds [87], tetrahedral meshes [39] or voxel grids [33, 47, 58, 71, 90]. Voxel grids could be further replaced by more compact structures like low-rank tensor decompositions [15], triplanes [10], hashing [52], and their combination [59].

Instead of focusing on speed, *Lightplane* significantly reduces memory demands for neural 3D fields with volumetric rendering. Our approach specifically targets neural 3D fields, whereas point-based rendering methods like 3DGS [12, 37, 72] fall outside this scope, as they don't model every 3D point in the space and rely on rasterization rather than volumetric rendering. While 3DGS offers fast convergence, it requires careful surface initialization. In contrast, NeRFs can converge from random initialization, making them more versatile for single-scene optimization.

Amortized 3D reconstruction. Amortized (Generalizable) 3D reconstruction utilizing implicit shape representations was initially approached in [28, 53, 61, 76, 82, 91] by warping/pooling features from source views to a target to estimate the color of the underlying scene surfaces. [64, 84] introduces latent transformer tokens to support the reconstruction. Generalizable triplanes [29, 30, 42], groundplanes [66], and voxel grids [32] were also explored.

These methods are memory-intensive, limited to *few-view* ($\leq$10), leading to suboptimal geometries and consistency. *Lightplane* supports over 100 input views, enabling large-scale training and more accurate reconstructions.

Image-supervised 3D generators. With the advent of Generative Adversarial Networks [24] (GAN), many methods attempted to learn 3D generative models given large uncurated image datasets. [20, 27, 54] learned to generate

voxel grids whose renders were indistinguishable from real object views according to an image-based deep discriminator. The same task was later tackled with Neural Radiance Fields [10, 25, 55, 65, 68], and meshes [21, 86]. The success of 2D generative diffusion models [18] led to image-supervised models such as [26, 35, 36, 48, 49, 74, 85], which directly model the distribution of 3D models. Similarly, RenderDiffusion [1] and ViewsetDiffusion [73] learn a 2D image denoiser by means of a 3D deep reconstructor. GeNVS [11] and HoloFusion [34] proposed 3D generators with 2D diffusion rendering post-processors. We demonstrate that *Lightplane* brings a strong performance boost to ViewsetDiffusion and generates realistic 3D scenes.

3. Method

We introduce *Lightplane Renderer* and *Splatter*, two core components for differentiable 2D-3D mapping that significantly reduce memory usage. First,we discuss the memory bottlenecks in existing methods used for rendering and lifting (Sec. 3.1). Then, we define the hashed 3D representations used in our framework (Sec. 3.2) and outline the functionality of the proposed components (Sec. 3.3). Finally, we discuss our memory-efficient designs (Sec. 3.4).

3.1. Preliminary

2D-3D Mapping. Mapping between 2D images and 3D models is a major practical bottleneck of many algorithms (Sec. 1), particularly when using powerful implicit 3D representations such as neural 3D fields. The memory bottleneck comprises a large number of 3D points from rendering rays and their intermediate features, which are cached in GPU memory for the ensuing backpropagation.

In rendering (*3D to 2D mapping*), an *entire ray* of 3D points contributes to the color of a single pixel in the rendered image. With M pixels and R points per ray, $M{\times}R$ MLP evaluations are needed to obtain colors and opacities. All intermediate results, including outputs of all MLP layers for every 3D point, are stored in memory for backpropagation, leading to huge memory usage. Using a tiny MLP with $L{=}6$ layers and $K{=}64$ hidden units, $M{\times}R{\times}L{\times}K$ memory is required to *just* store the MLP outputs, which totals 12 GB for a 256^2 image with $R{=}128$ points per ray.

Similarly, to lift N input features to 3D (*2D to 3D mapping*), models like PixelNeRF [91] and GeNVS [11] project each 3D point to N input views individually and average N sampled features as the point feature. Without considering any MLPs, this process uses $N{\times}|\mathcal{M}|$ memory, where $|\mathcal{M}|$ is the size of the 3D structure $\mathcal{M}$. For a 128^3 voxel grid with 64-dimensional features, $|\mathcal{M}|$ requires 512 MB in FP32, resulting in 5 GB of memory for 10 input views.

Moreover, this lifting process requires 3D structures' 3D positions for projection and cannot be easily used with other popular hash-based structures like triplanes or hash-tables. Elements in such structures (e.g., 2D grids on triplanes, or buckets in hash-table) lack clearly defined 3D positions and cannot be easily projected back to input images for features. Lifting multi-view inputs to them remains an open problem.

The memory bottleneck impacts multiple aspects. For 3D to 2D mapping (*i.e.* rendering), methods like NeRF [50] and PixelNeRF [91] are limited to a few low-resolution images per training iteration (even using 40GB GPUs) or to subsample rendered pixels, which prohibits image-level losses such as LPIPS [95] and SDS [57]. For 2D to 3D mapping, memory demands restrict the number of input views and the size of 3D representations. This substantial memory usage not only occupies resources that could enhance model sizes and capacities but also limits model training and inference on devices with limited memory.

Neural 3D fields. For 3D point $\boldsymbol{x} \in \mathbb{R}^3$, a *neural 3D field* is a volumetric function f that maps each point $\boldsymbol{x}$ to a vector $f(\boldsymbol{x}) \in \mathbb{R}^C$. NeRF [50] represented such functions using a single MLP. While simple, the MLP must represent entire 3D scenes, requiring it to be large and costly to evaluate.

Several methods are proposed to solve this problem, by decomposing information into local buckets (optionally with tiny MLP), which is more efficient than evaluating the global MLP. Most famously, [51] utilizes hash tables, but other representations such as voxel grids [70] and various low-rank decompositions such as triplanes [10], TensoRF [14] and HexPlane [8, 9] also follow this pattern.

3.2. Hybrid representation with 3D hash structure

Following the idea in Sec. 3.1, we use a hybrid representation for neural 3D fields f, and decompose $f = g \circ h$, where $h : \mathbb{R}^3 \to \mathbb{R}^K$ is a hashing scheme (sampling operation) for 3D hash structure θ, and $g : \mathbb{R}^K \to \mathbb{R}^C$ is a tiny MLP, inputing hashing features and outputting final values.

In this paper, we generalize the concept of 3D hash structures to structures like voxel grids [70], triplanes [10], HexPlane [8, 9] and actual hash table [51], as obtaining information from these structures only requires accessing and processing the small amount of information stored in a particular bucket. The associated hashing scheme h typically samples 3D point features from hash structure θ via interpolation, which is highly efficient. In practice, we operationalize θ with voxel grids and triplanes as they are easy to process by neural networks, although other structures with a differentiable hashing scheme could be easily supported.

In more detail, for voxel-grids, θ is a $H{\times}W{\times}D{\times}K$ tensor and h is the tri-linear interpolation on θ given position $\boldsymbol{x}$. For triplane, θ is a list of three tensors of dimensions $H{\times}W{\times}K$, $W{\times}D{\times}K$, and $D{\times}H{\times}K$. Then, $h(x, y, z)$ is obtained by bilinear interpolation of each plane at (x, y), (y, z), (z, x), respectively, followed by summing the resulting three feature vectors. Again, this design could be easily

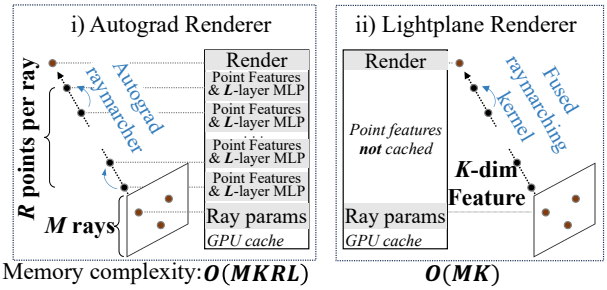

Figure 2. **GPU memory usage** of our *Lightplane Renderer* vs. a standard autograd NeRF renderer.

generalized to other hashed 3D structures θ with their corresponding hashing scheme (sampling operation) h.

3.3. Rendering and splatting

We now detail *Lightplane Renderer* and *Splatter*, two components using hybrid representations with 3D hash structures. They are mutually dual as one maps 3D information to 2D via rendering, and the other maps 2D images to 3D.

Renderer. *Renderer* outputs pixel features $\boldsymbol{v}$ (*e.g.* colors, depths) in a differentiable way from a hybrid representation $f{=}g{\circ}h$, given M rays $\{\boldsymbol{r}_i\}_{i=1}^{M}$ and $R{+}1$ points per ray. We make its high-level design consistent with existing hybrid representations [8, 10, 51, 70] as they have proven to be powerful, while re-designing the computations in Sec. 3.4 to achieve significant memory savings.

Following NeRF [50], *Renderer* uses a generalized Emission-Absorption (EA) model and calculates transmittance T_{ij}, which is the probability that a photon emitting at $\boldsymbol{x}_{ij}$ (j-th sampling points on the i-th ray) reaches the sensor. Accordingly, the rendered feature $\boldsymbol{v}_i$ of ray $\boldsymbol{r}_i$ is:

$$\boldsymbol{v}_i = \sum_{j=1}^{R} (T_{i,j-1} - T_{ij}) f_v(\boldsymbol{x}_{ij}). \tag{1}$$

where $f_v(\boldsymbol{x}_{ij})$ is the feature (*e.g.* color) of the 3D point $\boldsymbol{x}_{ij}$, obtained from the hybrid representation f_v; $T_{ij} = \exp(-\sum_{n=0}^{j} \Delta \cdot \sigma(\boldsymbol{x}_{in}))$, Δ is the distance between two sampled points, and $\sigma(\boldsymbol{x}_{in})$ is the opacity of the n-th sampled point; $(T_{i,j-1} - T_{ij}) \in [0, 1]$ is the visibility of the point $\boldsymbol{x}_{ij}$. Given a 3D point, *Renderer* samples its feature from the 3D representation and feeds the feature to an MLP g_σ to calculate the opacity. $f_v(\boldsymbol{x}_{ij})$ is calculated by another MLP g_v, with sampled feature and view directions as inputs.

Splatter. Lifting 2D inputs to 3D structures is crucial in 3D, particularly for feed-forward models. Existing works [11, 34, 35, 73] project 3D points (*voxel grid centers*) back to input views and interpolate (*pull*) input features based on these projected positions to obtain point features. After this lifting, a neural network (e.g., 3D-UNet) can be applied to the lifted 3D structures (e.g., voxel grids) for further processing, such as reconstruction. However, this lifting operation is memory-intensive and relies on the points' 3D positions for projection, making it incompatible with other popular 3D hashed structures, like triplanes or hash tables, where buckets (e.g., points on triplane or hash keys for hash-table) lack clearly defined 3D positions.

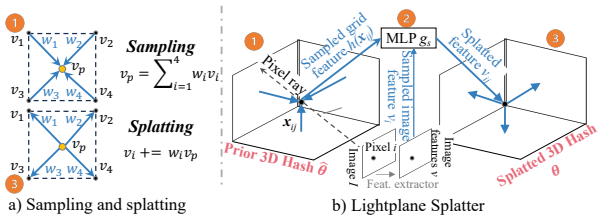

Figure 3. ***Lightplane Splatter* illustration.** (a) On a hash grid with vertex features $\boldsymbol{v}_i$: *sampling* gets point features $\boldsymbol{v}_p$ by interpolating vertex features weighted by inverse distance; *splatting* updates vertex features by accumulating point features to vertex features using the same weights. (b) *Lightplane Splatter* involves three steps. For each 3D point along the ray, *Splatter* samples its features from prior 3D hash $\hat{\theta}$ (1), calculates features to be splatted using MLP (2), and splats them to zero-initialized θ (3).

Instead of looping over 3D points and pulling information from inputs, *Splatter* is designed to loop over *input pixels/rays* and directly *push* information to 3D structures. This approach makes *Splatter* a reverse of *Renderer*, eliminating the reliance on 3D points' positions and enabling a unified lifting operation compatible with various 3D hashed structures, as long as they have a clearly defined renderer function. Additionally, since *Renderer* also loops over *pixels/rays*, *Splatter* can leverage similar memory optimization strategies, significantly reducing memory usage.

Starting from a zero-initialized 3D structure θ, *Splatter* samples $R{+}1$ equispaced 3D points $\boldsymbol{x}_{ij}$ along the ray $\boldsymbol{r}_i$ (cast from an input pixel), and splats (pushes) these 3D points' features into θ. 3D point's feature $\boldsymbol{v}_{ij}$ is inherited from the pixel's feature $\boldsymbol{v}_i$, and the splatting operation is inverse to the sampling operation used in rendering. Specifically, this is achieved by accumulating $\boldsymbol{v}_{ij}$ into hash cells containing $\boldsymbol{x}_{ij}$, with the accumulation weighted by splatting weights (Fig3(a)). After looping over all M rays, each hash cell is normalized by the sum of all splatting weights assigned to that cell. For triplanes, a hash cell is a 2D grid on a plane, and splatting weights are the normalized inverse distances between the 2D point and the four grid vertices.

Assigning the same pixel feature to all points along the ray works well for voxel grids, while fails for more compact representations like triplanes, which have less 3D spatial awareness when reducing from 3D space to 2D planes. To this end, we introduce another MLP g_s to regress a more expressive feature vector $\boldsymbol{v}_{ij}$ for points $\boldsymbol{x}_{ij}$, taking pixel feature $\boldsymbol{v}_i$, interpolated prior shape encoding $h_{\hat{\theta}}(\boldsymbol{x}_{ij})$, and the positional encoding $\mathrm{direnc}(\boldsymbol{r}_i)$ of ray direction $\boldsymbol{r}_i$ as input.

For each sample $\boldsymbol{x}_{ij}$, the splatted feature $\tilde{\boldsymbol{v}}_{ij}$ is

$$\tilde{\boldsymbol{v}}_{ij} = g_s(\boldsymbol{v}_i, h_{\hat{\theta}}(\boldsymbol{x}_{ij}), \text{direnc}(\boldsymbol{r}_i)) \tag{2}$$

$\hat{\theta}$ is a another hashed 3D structure, and $h_{\hat{\theta}}(\cdot)$ is the corresponding hashing scheme used to obtain features given $\boldsymbol{x}_{ij}$. This MLP allows points along the same ray to have distinct spatial-aware features, effectively improving the model's expressive capability. In practice, $\hat{\theta}$ can either be a randomly initialized learnable 3D structure (serving as a learnable 3D positional encoding) or the splatted results from previous steps. This enables iterative refinement of 3D representations based on the current input and previous results. We provide illustrations in Figure 3 and detailed pseudocode in the supplementary material.

3.4. Memory-efficient Designs

We discuss the practical designs of *Lightplane Renderer* and *Splatter*, which makes them memory-efficient and scalable.

Fusing operations along the ray. As analyzed, treating 3D points as basic entities and storing intermediate results for each point is extremely memory-intensive. Alternatively, we treat rays as the basic entities and reconfigure the computations to process sequentially along the ray without storing intermediate results, leading to significant memory savings. Operations along the ray are fused into a single GPU kernel, which is parallelly spawned for each ray.

As Eq. 1, a *Renderer* kernel sequentially conducts feature sampling, MLP evaluation, and rendered result updating along the ray without saving any intermediate results. In the example from Sec. 3.1, this approach reduces memory usage from $O(MKRL)$ to $O(MK)$, dropping from 12 GB to 2 KB for a 256^2 image with $R = 128$ samples per ray in FP32. This is less than 0.02% of the memory required by the naïve implementation. Since *Splatter* is designed to process rays emanating from input pixels as well (Sec. 3.3), it benefits from the same optimization practice.

Recalculation for backpropagation. Skipping the storage of intermediate results during forward propagation drastically reduces memory usage, though these tensors are crucial for backpropagation. To manage this, we recompute intermediate results during backpropagation for gradient calculation. While recalculating the MLP increases floating-point operations by less than 50 compared to the naïve implementation, this cost is incurred only during backpropagation and leads to significant memory savings.

Using GPU memory hierarchy. The speed can be further optimized by leveraging the GPU's memory hierarchy. By fusing operations into a single GPU kernel, we maximize on-chip SRAM usage and minimize slower HBM access, which often bottlenecks performance. This approach keeps our kernel efficient, even with the overhead of recalculating intermediate results during backpropagation. For more on GPU memory hierarchy, see flash-attention [17].

Emission-absorption backpropagation. *Renderer* and *Splatter* are dual not only in functionality but also in their implementation. *Splatter* 's backpropagation mirrors *Renderer* 's forward pass, as it samples 3D point gradients from 3D structures' gradient field and aggregates them along the ray to form input pixel gradients. Conversely, *Renderer* 's backward process resembles *Splatter* 's forward pass.

Notably, *Renderer*'s backpropgation is more complex as the visibility of 3D points is influenced by the transmittance of previous points in the EA model. During the forward pass, we sequentially calculate each 3D point's visibility and implement the rendering equation Eq. (1) by summing in order $j{=}1, 2, \ldots$, as visibility T_j is easily derived from T_{j-1} (ray index omitted for simplicity). For the backward pass on a ray $\boldsymbol{r}$, the vector-Jacobian product (i.e., the quantity computed during backpropagation) is derived as:

$$\boldsymbol{p}^\top \frac{d\boldsymbol{v}}{df_\sigma(\boldsymbol{x}_q)} = -\Delta \frac{d\sigma(\boldsymbol{x}_q)}{df_\sigma(\boldsymbol{x}_q)} \left(\sum_{j=q+1}^{R} (T_{j-1} - T_j)\boldsymbol{a}_j - T_q \boldsymbol{a}_q \right) \tag{3}$$

where $\boldsymbol{a}_j{=}\boldsymbol{p}^\top f_v(\boldsymbol{x}_j)$, $\boldsymbol{p}$ is the propogated gradient vector.

To backpropagate through *Renderer* efficiently, we compute Eq. (3) by marching along each rendering ray in the reverse order $q = R, R{-}1, \ldots$, since the vectors $\boldsymbol{a}_j$ are accumulated from sample q onwards, and the opacity $f_\sigma(\boldsymbol{x}_q)$ affects only the visibility of successive samples $\boldsymbol{x}_q, \boldsymbol{x}_{q+1}, \ldots$. To make this possible, we cache the final transmittance T_R, which is computed in the forward pass(this amounts to one scalar per ray). In backpropagation, we sequentially compute $\sigma(x_j)$ for every 3D point along the ray, and calculate $T_{j-1} = T_j \cdot \exp(\Delta\sigma(x_j))$ from T_j. In this way, similar to the forward pass, the kernel only stores the accumulation of per-point features instead of keeping them all in memory.

Difference from checkpointing. Note that the latter is very different from "checkpointing" which can be trivially enabled for the naive renderer implementation in autograd frameworks such as PyTorch. This is because, unlike our memory-efficient backward pass from Eq. (3), a checkpointed backward pass still entails storing all intermediate features along rendering rays in memory.

4. Example applications

Similar to flash-attention [17], *Lightplane Renderer* and *Splatter* serve as highly memory-efficient building blocks for rendering and lifting, which can be easily integrated into existing models and pipelines. We demonstrate various 3D

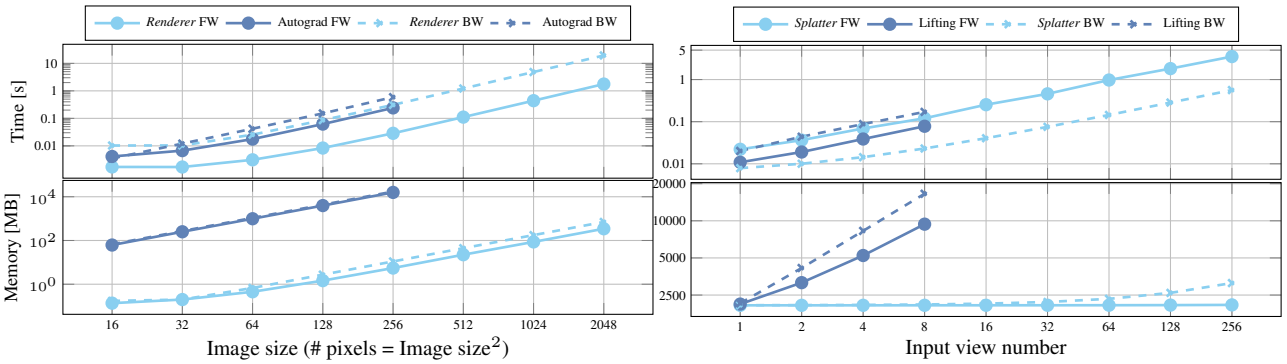

Figure 4. ***Lightplane* memory & speed benchmark** showing the forward (FW and backward (BW) passes of *Lightplane Renderer* (left) and *Splatter* (right), compared to the Autograd renderer and lifter from [11, 91]. *Lightplane* exhibits over **1000**× lower memory consumption at comparable speed. Note the size of lifting/*Splatter* outputs is 1GB, leading to smaller saving ratio. All axes are log-scaled.

applications that can benefit from these components, ranging from single-scene optimization with image-level losses to a versatile framework for large-scale 3D reconstruction and generation. Results are presented in Sec. 5.

Single-scene optimization with image-level losses. Due to the intensive memory usage in rendering, existing volumetric methods are often limited to using pixel-level losses (*e.g.* MSE) on a subset of rays, or image-level losses on very low-resolution renderings 64^2 [45, 57], often requiring complicated and inefficient deferred techniques. In contrast, *Lightplane Renderer* enables direct rendering of high-resolution images (*e.g.* 512^2) in a differentiable way, with minimal memory overhead and without the need for additional tricks. It seamlessly supports image-level looses (*e.g.* perceptual loss [31], LPIPS [95], or SDS [57],) on high-resolution full images We show 3D neural style transfer results on Figure 5 and discuss more applications in Supp.

Multi-view reconstruction. Combining *Renderer* and *Splatter*, we introduce a versatile pipeline for 3D reconstruction and generation, as shown in Fig 1 (left). Given a set of views (viewset) $\mathcal{V}=\{I_i\}_{i=1}^N$ and corresponding cameras $\{\pi_i\}_{i=1}^N$, we train a large-scale model Φ, which directly outputs the 3D representations $\theta=\Phi(\mathcal{V},\pi)$ of the scene by learning 3D priors from large-scale data. Reconstruction begins by extracting a pixel-wise feature map $\boldsymbol{v}=\psi(I_i)$ from each image I_i and lifting them into the 3D representation $\tilde{\theta}$ with *Splatter*. Model Φ takes $\tilde{\theta}$ as input and outputs the final 3D representations $\theta=\Phi_\theta(\tilde{\theta})$. Finally, *Renderer* outputs novel view images $\hat{I}=\mathcal{R}(\theta,\pi)$ from θ, and the model is trained by minimizing the loss $\mathcal{L}$ between the novel rendered image and the corresponding ground truth I.

3D generation using viewset diffusion. Following recent works [2, 73], above pipeline can be easily extended into a diffusion-based 3D generator with minimal changes. This is done by introducing a noised viewset as input to the network and training the model to denoise the viewset. Specifically, each image I_i is replaced with $I_{it}=\alpha_t I_i+\sigma_t\epsilon_i$ where t is the noising schedule, α_t and $\sigma_t=\sqrt{1-\alpha_t^2}$ are the noise level, and ϵ_i is a random normal noise vector. During inference, the model initializes the viewset with Gaussian noise and iteratively denoises it by applying the reconstruction model. This process simultaneously generates multiple views of the object as well as its 3D model.

Org 3D Scene Stylization 1 Stylization 2

Figure 5. **Single-scene optimization with image-level losses.** The memory efficiency of *Lightplane* allows rendering high resolution images in a differentiable way and backpropagating image-level losses. We show pre-optimized 3D scenes (in unseen views) and their stylizations with perceptual losses.

5. Experiments

We first benchmark the proposed components' performance and then demonstrate their usage on various 3D tasks. The scalability boost comes from improvements in both *input-size* and *modeling*. For input-size, the components significantly increase the amount of 2D information lifted to 3D by enabling more input views and larger output sizes. For modeling, the memory savings allow for increased model complexity and larger batch sizes during training.

5.1. Memory & Speed Benchmark

We benchmark components' speed and memory in Figure 4 and show more benchmarks in Supp. Overall, *Lightplane* offers over **1000**× memory savings with comparable speed to baselines. *Renderer* (left column) is tested on a triplane with 256 points per ray and compared to a PyTorch Autograd renderer, adapted from [8, 10]. It easily supports large images with very low memory usage, which is unfeasible for the Autograd renderer. *Splatter* (right col.) is tested on lifting N input feature maps into a 160^3 voxel grid. We benchmark it against the lifting operations from GeNVS [11], disabling the MLPs in *Splatter* for a fair com-

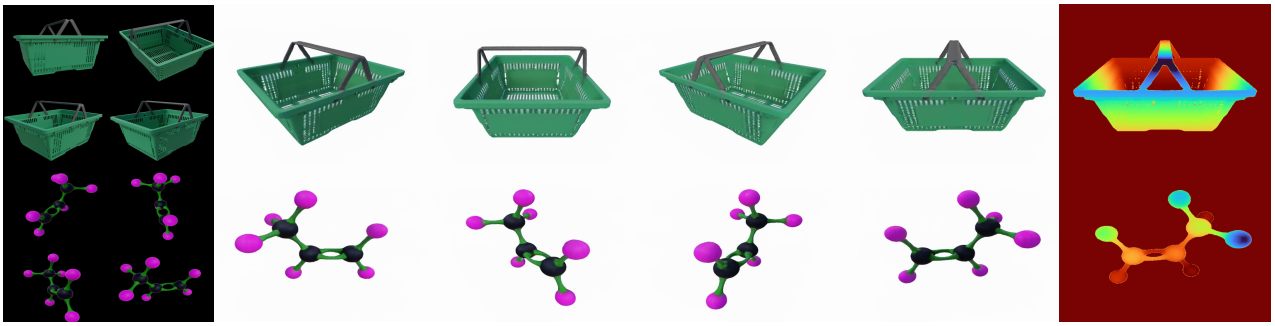

Figure 6. **Multi-view Large Reconstruction Model (LRM) with *Lightplane*.** Taking four views as input (leftmost column), we show the RGB renders (mid) and depth (rightmost column) of the 3D reconstruction.

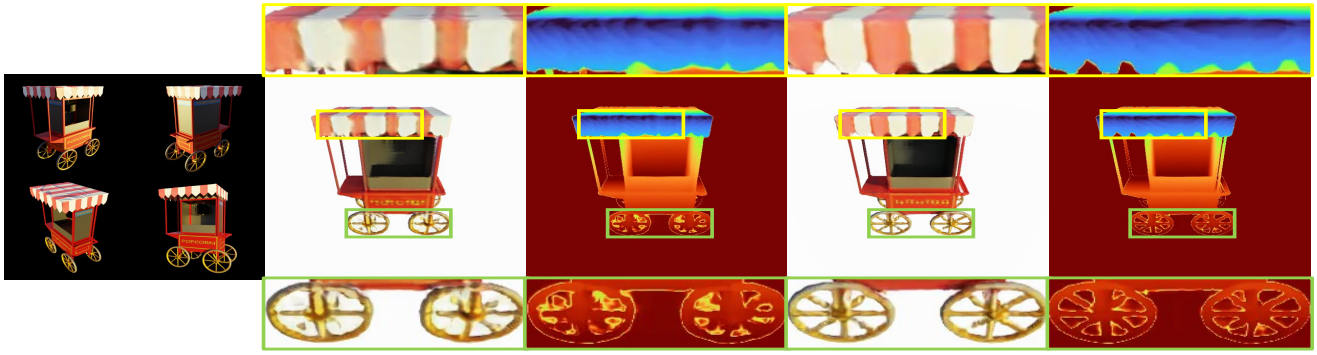

Figure 7. **Adding *Lightplane* to LRM** gives more accurate geometry and appearance with little additional computation and memory cost.

parison. As shown, *Splatter* can handle over a hundred views efficiently, while existing methods are restricted to just a few views. Notably, even for modest settings (*i.e.* 4 input views, 128^2 rendered image size), existing designs still consume over 20 GB of memory, which constitutes a significant portion (25%) of an A100 GPU's memory.

Table 1. **Quantitative results of LRM.** *Lightplane* effectively improves the reconstruction results, especially geometry (depth L1).

Method	PSNR↑	LPIPS↓	IOU↑	Depth L1↓
LRM [29]	23.7	0.113	0.904	0.208
Lightplane +LRM	**24.1**	**0.106**	**0.916**	**0.168**

Figure 9. **Single Image 3D Recon.** With a clean image (1st col.), our model generates realistic 3D structures matching the input.

5.2. Multi-view LRM with *Lightplane*

We first validate *Lightplane* 's efficacy by integrating it with the Large Reconstruction Model (LRM)[29], with results presented in Table1 and Figures 6, 7. LRM takes four images as input and outputs a triplane as the 3D representation through a series of transformer blocks. Every 3 transformer blocks (*i.e.* 5 blocks in total), we insert the *Splatter* layer, which splats source view features into a new triplane, using the previous block's outputs (triplane) as a prior shape encoding (Eq 2). Incorporating *Splatter* into LRM adds minimal computational overhead while clearly enhancing performance. Also, replacing the original renderer with our *Renderer* enables LPIPS loss without the added complexity of the deferred backpropagation used in LRM [29].

Table 2. **Amortized 3D Reconstruction.** *Lightplane* trained on the *whole* CO3Dv2 significantly outperforms baseline ViewFormer [41]. We compare overfitting baselines (Voxel, NeRF [50]) to *Lightplane*, and to their scene-tuned versions ("Feedforward + Overfit"). Initializing from *Lightplane*-feedforward removes defective geometry leading to better depth error.

Method	Mode	#views	PSNR↑	LPIPS↓	Depth corr.↑	Time↓
ViewFormer [41]	Feedforward	9	16.4	0.274	N/A	N/A
Lightplane	Feedforward	10	20.7	0.141	0.356	1.6 sec
Lightplane	Feedforward	20	20.9	0.136	0.382	1.9 sec
Lightplane	Feedforward	40	21.4	0.131	0.405	2.5 sec
Lightplane	Feedforward + Overfit	160	26.2	**0.086**	**0.449**	5 min
Voxel	Overfit from scratch	160	**26.5**	**0.086**	0.373	35 min
NeRF	Overfit from scratch	160	26.3	0.108	0.658	1 day

Table 3. **Unconditional 3D Generation on CO3Dv2.** Our *Lightplane* significantly outperforms HoloDiffusion [35] and Viewset Diffusion [73]. It even beats HoloFusion [34], a distillation-based method, which takes 30 mins for one generation.

Method	Feed-forward	Hydrant FID↓	Hydrant KID↓	Teddybear FID↓	Teddybear KID↓	Apple FID↓	Apple KID↓	Donut FID↓	Donut KID↓	Mean FID↓	Mean KID↓	Inference Time
HoloFusion [34]	×	**66.8**	**0.047**	**87.6**	0.075	69.2	0.063	109.7	0.098	83.3	0.071	30 min
HoloDiffusion [35]	✓	100.5	0.079	109.2	0.106	94.5	0.095	115.4	0.085	122.5	0.102	< 2 min
Viewset Diffusion [73]	✓	150.5	0.124	219.7	0.178	-	-	-	-	-	-	< 2 min
Lightplane	✓	75.1	0.058	87.9	**0.070**	**32.6**	**0.019**	**44.0**	**0.019**	**59.9**	**0.042**	< 2 min

Figure 8. **Unconditional 3D Generation** displaying samples from our *Lightplane*-augmented Viewset Diffusion trained on CO3Dv2 [60].

5.3. Large-Scale 3D Recon. and Gen.

As illustrated in Sec. 4 and Fig 1, we build a large-scale 3D reconstruction and generation model based on *Lightplane*. As a versatile pipeline, it could support various 3D hash structures, *e.g*. voxel grids or triplanes. Without loss of generality, we utilize a UNet [62] with attention layers [77] to process these 3D hash structures (*Deep net* in Fig 1). We use CO3Dv2 [60] as our primary dataset, a collection of real-world videos capturing objects across 51 common categories. We contract ray-point's coordinates [5] to support unbounded backgrounds. See more details in Supp..

Amortized 3D Reconstruction. Existing amortized 3D reconstruction and novel view synthesis methods [60, 76, 81, 91, 96] are typically limited to a few input views (up to 10) due to memory constraints. In contrast, we significantly increase the number of input views. Unlike category-specific models, we train a *single* model across *all* CO3Dv2 categories, aiming for a reconstruction model capable of handling various object types and providing valuable 3D priors for subsequent 3D optimizations. During training, 20 source images from each scene are used as inputs, with MSE losses calculated on five additional novel views.

We evaluate our model in two ways: (1) comparing it to other feedforward and single-scene overfitting methods to assess performance; (2) finetuning feedforward results using training views from a single scene to to show its efficacy as a learned 3D prior. As few feedforward Nerf model work across all categories, we use ViewFormer [40] as the baseline, which directly outputs novel views from a Transformer. In (2), we input 80 views to the feedforward model for initialization and report results of vanilla NeRF [50], and voxel-grid overfits (*i.e*., trained from scratch). Results are evaluated on novel views of unseen scenes.

Our model achieves compelling reconstructions with a single forward pass, as shown in Tab. 2. After finetuning, it matches overfitting baselines in color accuracy (PSNR, LPIPS) and largely outperforms hash-based baselines (Voxel) in depth error. CO3D's limited viewpoint coverage leads to geometric defects in hashed representations (see Supp.). Leveraging *Lightplane* 's memory-efficient pre-training, our model learns a robust surface prior, ensuring defect-free geometry. NeRF excels in depth accuracy but requires 50× longer training.

Generation. Our model supports unconditional generation with minimal modifications, specifically by accepting noisy input images and rendering clean images through a denoising process. Using *Splatter* and *Renderer*, we can denoise more views (10 in our experiments) per iteration, enhancing process stability and significantly improving results. During inference, we input 10 instances of pure noise and proceed with 50 Denoising Diffusion Implicit Model (DDIM) [69] sampling steps. We compare our method to Viewset Diffusion [73] and HoloFusion [34] quantitatively in Tab. 3 and evaluate qualitatively in Fig. 8. Our results significantly outperform other feedforward generation models and are comparable to distillation-based methods, which are much more time-consuming. We can also introduce a clean image as conditioning, enabling single-view reconstruction. Our framework can also be extended to a text-conditioned model by using captions as inputs. We present results and comparisons in Figure 9 and the supplementary materials.

6. Conclusion

We have introduced *Lightplane*, a versatile framework that provide two novel components, *Splatter* and *Renderer*, which address the key memory bottleneck in network that manipulate neural fields. We have showcased the potential of these primitives in a number of applications, boosting models for reconstruction, generation and more. Once released to the community, we hope that these primitives will be used by many to boost their own research as well. [2]

7. Acknowledgement

Ang Cao is supported by a grant from LG AI Research. We thank Roman Shapovalov, Jianyuan Wang, and Mohamed El Banani for their valuable help and discussions.

[2] We discuss limitations and potential negative impact in Supp.

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. *arXiv.cs*, abs/2211.09869, 2022. 3

[2] Titas Anciukevičius, Zexiang Xu, Matthew Fisher, Paul Henderson, Hakan Bilen, Niloy J Mitra, and Paul Guerrero. Renderdiffusion: Image diffusion for 3d reconstruction, inpainting and generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12608–12618, 2023. 6

[3] Anonymous. Instant3d: Fast text-to-3d with sparse-view generation and large reconstruction model. *Under Review*, 2023. 2

[4] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021. 2

[5] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022. 8

[6] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Zip-nerf: Anti-aliased grid-based neural radiance fields. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 19640–19648, 2023. 2

[7] Raphael Bensadoun, Tom Monnier, Yanir Kleiman, Filippos Kokkinos, Yawar Siddiqui, Mahendra Kariya, Omri Harosh, Roman Shapovalov, Benjamin Graham, Emilien Garreau, et al. Meta 3d gen. *arXiv preprint arXiv:2407.02599*, 2024. 2

[8] Ang Cao and Justin Johnson. Hexplane: A fast representation for dynamic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 130–141, 2023. 3, 4, 6

[9] 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 *arXiv*, 2021. 3

[10] 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, et al. Efficient geometry-aware 3d generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16123–16133, 2022. 2, 3, 4, 6

[11] 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 *ICCV*, 2023. 1, 2, 3, 4, 6

[12] David Charatan, Sizhe Lester Li, Andrea Tagliasacchi, and Vincent Sitzmann. pixelsplat: 3d gaussian splats from image pairs for scalable generalizable 3d reconstruction. *ArXiv*, abs/2312.12337, 2023. 2

[13] 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 *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14124–14133, 2021. 1

[14] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision (ECCV)*, 2022. 3

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

[16] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII 14*, pages 628–644. Springer, 2016. 2

[17] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022. 2, 5

[18] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021. 3

[19] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613, 2017. 2

[20] Matheus Gadelha, Subhransu Maji, and Rui Wang. 3D shape induction from 2D views of multiple objects. In *arXiv*, 2016. 2

[21] Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. GET3D: A generative model of high quality 3d textured shapes learned from images. *arXiv.cs*, abs/2209.11163, 2022. 3

[22] Rohit Girdhar, David F Fouhey, Mikel Rodriguez, and Abhinav Gupta. Learning a predictable and generative vector representation for objects. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VI 14*, pages 484–499. Springer, 2016. 2

[23] Georgia Gkioxari, Justin Johnson, and Jitendra Malik. Mesh R-CNN. In *Proc. ICCV*, 2019. 2

[24] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Proc. NeurIPS*, 2014. 2

[25] Jiatao Gu, Lingjie Liu, Peng Wang, and Christian Theobalt. Stylenerf: A style-based 3d-aware generator for high-resolution image synthesis. *arXiv preprint arXiv:2110.08985*, 2021. 3

[26] Xianglong He, Junyi Chen, Sida Peng, Di Huang, Yangguang Li, Xiaoshui Huang, Chun Yuan, Wanli Ouyang, and

Tong He. Gvgen: Text-to-3d generation with volumetric representation. *ArXiv*, abs/2403.12957, 2024. 3

[27] Philipp Henzler, Niloy J. Mitra, and Tobias Ritschel. Escaping plato's cave using adversarial training: 3D shape from unstructured 2D image collections. In *Proc. ICCV*, 2019. 2

[28] Philipp Henzler, Jeremy Reizenstein, Patrick Labatut, Roman Shapovalov, Tobias Ritschel, Andrea Vedaldi, and David Novotny. Unsupervised learning of 3d object categories from videos in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 2

[29] 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. *arXiv preprint arXiv:2311.04400*, 2023. 2, 7

[30] Muhammad Zubair Irshad, Sergey Zakharov, Katherine Liu, Vitor Guizilini, Thomas Kollar, Adrien Gaidon, Zsolt Kira, and Rares Ambrus. Neo 360: Neural fields for sparse view synthesis of outdoor scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9187–9198, 2023. 2

[31] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part II 14*, pages 694–711. Springer, 2016. 6

[32] Abhishek Kar, Christian Häne, and Jitendra Malik. Learning a multi-view stereo machine. *Advances in neural information processing systems*, 30, 2017. 2

[33] Animesh Karnewar, Tobias Ritschel, Oliver Wang, and Niloy Mitra. Relu fields: The little non-linearity that could. In *ACM SIGGRAPH 2022 Conference Proceedings*, pages 1–9, 2022. 2

[34] Animesh Karnewar, Niloy J Mitra, Andrea Vedaldi, and David Novotny. Holofusion: Towards photo-realistic 3d generative modeling. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22976–22985, 2023. 3, 4, 7, 8

[35] Animesh Karnewar, Andrea Vedaldi, David Novotny, and Niloy Mitra. Holodiffusion: Training a 3D diffusion model using 2D images. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023. 3, 4, 7

[36] Animesh Karnewar, Andrea Vedaldi, Roman Shapovalov, Tom Monnier, Niloy Jyoti Mitra, and David Novotny. Goembed: Gradient origin embeddings for representation agnostic 3d feature learning. 2023. 3

[37] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42 (4), 2023. 1, 2

[38] Justin Kerr, Chung Min Kim, Ken Goldberg, Angjoo Kanazawa, and Matthew Tancik. Lerf: Language embedded radiance fields. *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 19672–19682, 2023. 2

[39] Jonas Kulhanek and Torsten Sattler. Tetra-nerf: Representing neural radiance fields using tetrahedra. *arXiv preprint arXiv:2304.09987*, 2023. 2

[40] Jonáš Kulhánek, Erik Derner, Torsten Sattler, and Robert Babuška. ViewFormer: NeRF-free neural rendering from few images using transformers. In *Proc. ECCV*, 2022. 8

[41] Jon'avs Kulh'anek, Erik Derner, Torsten Sattler, and Robert Babuvska. Viewformer: Nerf-free neural rendering from few images using transformers. In *European Conference on Computer Vision*, 2022. 7

[42] 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. *arXiv preprint arXiv:2311.06214*, 2023. 2

[43] Ruilong Li, Sanja Fidler, Angjoo Kanazawa, and Francis Williams. Nerf-xl: Scaling nerfs with multiple gpus. *ArXiv*, abs/2404.16221, 2024. 2

[44] Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. Neuralangelo: High-fidelity neural surface reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8456–8465, 2023. 2

[45] Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3d: High-resolution text-to-3d content creation. *arXiv preprint arXiv:2211.10440*, 2022. 6

[46] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7708–7717, 2019. 1

[47] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *arXiv preprint arXiv:1906.07751*, 2019. 2

[48] Luke Melas-Kyriazi, Christian Rupprecht, and Andrea Vedaldi. Pc^2 projection-conditioned point cloud diffusion for single-image 3d reconstruction, 2023. 3

[49] Luke Melas-Kyriazi, Iro Laina, C. Rupprecht, Natalia V. Neverova, Andrea Vedaldi, Oran Gafni, and Filippos Kokkinos. Im-3d: Iterative multiview diffusion and reconstruction for high-quality 3d generation. *ArXiv*, abs/2402.08682, 2024. 3

[50] 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 *ECCV*, 2020. 1, 2, 3, 4, 7, 8

[51] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics (ToG)*, 41(4):1–15, 2022. 3, 4

[52] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. In *Proc. SIGGRAPH*, 2022. 2

[53] Phong Nguyen-Ha, Animesh Karnewar, Lam Huynh, Esa Rahtu, and Janne Heikkila. Rgbd-net: Predicting color and depth images for novel views synthesis. In *Proceedings of the International Conference on 3D Vision*, 2021. 2

[54] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. HoloGAN: Unsupervised

learning of 3D representations from natural images. *arXiv.cs*, abs/1904.01326, 2019. 2

[55] Michael Niemeyer and Andreas Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021. 3

[56] Jeong Joon Park, Peter R. Florence, Julian Straub, Richard A. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 165–174, 2019. 2

[57] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. *arXiv preprint arXiv:2209.14988*, 2022. 2, 3, 6

[58] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. KiloNeRF: Speeding up neural radiance fields with thousands of tiny MLPs. *arXiv.cs*, abs/2103.13744, 2021. 2

[59] Christian Reiser, Rick Szeliski, Dor Verbin, Pratul Srinivasan, Ben Mildenhall, Andreas Geiger, Jon Barron, and Peter Hedman. Merf: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes. *ACM Transactions on Graphics (TOG)*, 42(4):1–12, 2023. 2

[60] 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 *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10901–10911, 2021. 8

[61] 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

[62] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015. 8

[63] Radu Alexandru Rosu and Sven Behnke. Permutosdf: Fast multi-view reconstruction with implicit surfaces using permutohedral lattices. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8466–8475, 2023. 2

[64] Mehdi S. M. Sajjadi, Henning Meyer, Etienne Pot, Urs Bergmann, Klaus Greff, Noha Radwan, Suhani Vora, Mario Lucic, Daniel Duckworth, Alexey Dosovitskiy, Jakob Uszkoreit, Thomas A. Funkhouser, and Andrea Tagliasacchi. Scene representation transformer: Geometry-free novel view synthesis through set-latent scene representations. *CoRR*, abs/2111.13152, 2021. 2

[65] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. *Advances in Neural Information Processing Systems*, 33:20154–20166, 2020. 3

[66] Prafull Sharma, Ayush Tewari, Yilun Du, Sergey Zakharov, Rares Ambrus, Adrien Gaidon, William T. Freeman, Frédo Durand, Joshua B. Tenenbaum, and Vincent Sitzmann. Seeing 3d objects in a single image via self-supervised static-dynamic disentanglement. *arXiv.cs*, abs/2207.11232, 2022. 2

[67] Yawar Siddiqui, Tom Monnier, Filippos Kokkinos, Mahendra Kariya, Yanir Kleiman, Emilien Garreau, Oran Gafni, Natalia V. Neverova, Andrea Vedaldi, Roman Shapovalov, and David Novotny. Meta 3d assetgen: Text-to-mesh generation with high-quality geometry, texture, and pbr materials. *ArXiv*, abs/2407.02445, 2024. 2

[68] Ivan Skorokhodov, Sergey Tulyakov, Yiqun Wang, and Peter Wonka. Epigraf: Rethinking training of 3d gans. *arXiv preprint arXiv:2206.10535*, 2022. 3

[69] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020. 8

[70] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *CVPR*, 2022. 3, 4

[71] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5459–5469, 2022. 2

[72] Stanislaw Szymanowicz, C. Rupprecht, and Andrea Vedaldi. Splatter image: Ultra-fast single-view 3d reconstruction. *ArXiv*, abs/2312.13150, 2023. 2

[73] Stanislaw Szymanowicz, Christian Rupprecht, and Andrea Vedaldi. Viewset diffusion:(0-) image-conditioned 3d generative models from 2d data. *arXiv preprint arXiv:2306.07881*, 2023. 3, 4, 6, 7, 8

[74] 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 *arXiv*, 2023. 3

[75] Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19, 2019. 2

[76] Alex Trevithick and Bo Yang. Grf: Learning a general radiance field for 3d scene representation and rendering. In *arXiv:2010.04595*, 2020. 2, 8

[77] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 8

[78] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T Barron, and Pratul P Srinivasan. Ref-nerf: Structured view-dependent appearance for neural radiance fields. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5481–5490. IEEE, 2022. 2

[79] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proceedings of the European conference on computer vision (ECCV)*, pages 52–67, 2018. 2

[80] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. NeuS: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. *arXiv.cs*, abs/2106.10689, 2021. 2

[81] 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. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4688–4697, 2021. 8

[82] 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

[83] Yiming Wang, Qin Han, Marc Habermann, Kostas Daniilidis, Christian Theobalt, and Lingjie Liu. Neus2: Fast learning of neural implicit surfaces for multi-view reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3295–3306, 2023. 2

[84] Chao-Yuan Wu, Justin Johnson, Jitendra Malik, Christoph Feichtenhofer, and Georgia Gkioxari. Multiview compressive coding for 3d reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9065–9075, 2023. 2

[85] Rundi Wu, Ben Mildenhall, Philipp Henzler, Keunhong Park, Ruiqi Gao, Daniel Watson, Pratul P. Srinivasan, Dor Verbin, Jonathan T. Barron, Ben Poole, and Aleksander Holynski. Reconfusion: 3d reconstruction with diffusion priors. *ArXiv*, abs/2312.02981, 2023. 3

[86] Shangzhe Wu, Christian Rupprecht, and Andrea Vedaldi. Unsupervised learning of probably symmetric deformable 3d objects from images in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1–10, 2020. 3

[87] Qiangeng Xu, Zexiang Xu, Julien Philip, Sai Bi, Zhixin Shu, Kalyan Sunkavalli, and Ulrich Neumann. Point-NeRF: Point-based neural radiance fields. *arXiv.cs*, abs/2201.08845, 2022. 2

[88] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4541–4550, 2019. 2

[89] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. *Advances in Neural Information Processing Systems*, 34:4805–4815, 2021. 2

[90] Alex Yu, Sara Fridovich-Keil, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. *arXiv preprint arXiv:2112.05131*, 2021. 2

[91] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4578–4587, 2021. 1, 2, 3, 6, 8

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

[93] Zehao Yu, Songyou Peng, Michael Niemeyer, Torsten Sattler, and Andreas Geiger. Monosdf: Exploring monocular geometric cues for neural implicit surface reconstruction. *Advances in neural information processing systems*, 35:25018–25032, 2022. 2

[94] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020. 2

[95] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 2, 3, 6

[96] Zhizhuo Zhou and Shubham Tulsiani. Sparsefusion: Distilling view-conditioned diffusion for 3d reconstruction. *arXiv preprint arXiv:2212.00792*, 2022. 8