Flexible Techniques for Differentiable Rendering with 3D Gaussians

Leonid Keselman   Martial Hebert
Carnegie Mellon University
Pittsburgh, PA, USA

{lkeselma,hebert}@cs.cmu.edu

Abstract

Fast, reliable shape reconstruction is an essential ingredient in many computer vision applications. Neural Radiance Fields demonstrated that photorealistic novel view synthesis is within reach, but was gated by performance requirements for fast reconstruction of real scenes and objects. Several recent approaches have built on alternative shape representations, in particular, 3D Gaussians. We develop extensions to these renderers, such as integrating differentiable optical flow, exporting watertight meshes and rendering per-ray normals. Additionally, we show how two of the recent methods are interoperable with each other. These reconstructions are quick, robust, and easily performed on GPU or CPU. For code and visual examples, see https://leonidk.github.io/fmb-plus.

1. Introduction

As computer vision systems are more widely deployed in society, either on robots or via mixed reality headsets, users will desire that reconstructions of their many regular everyday objects. While classic techniques from multiview scene reconstruction could be used [61], modern approaches strive for more photorealistic scene generation, such as those created by Neural Radiance Fields (NeRF) [48] and the large array of follow-up work [64]. Of note, these NeRF approaches could be seen as differentiable renderers [39, 73], where an underlying scene representation is optimized for view synthesis. However, NeRF methods are demanding in their computational requirements, even with speed-up methods such as Instant-NGP [51].

Two recent papers explored similar approaches to fast NeRF alternatives – Fuzzy Metaballs [34] in ECCV 2022 and 3D Gaussian Splatting in SIGGRAPH 2023 [33]. Both approaches represent geometry using a set of classic primitives, specifically 3D Gaussians [21, 56]. The former built a differentiable raytracer for 3D Gaussians, connected it to the metaball literature [4], developed a sort-free rendering function, focused on fast CPU runtimes, used dozens of

(a) **Fuzzy Metaballs** [34] raytraced dozens of Gaussians, used random initialization, and estimated geometry and pose for objects.

(b) **3D Gaussian Splatting** [33] splatted millions of Gaussians, used SfM [61] initialization, and synthesized novel views for scenes.

Figure 1. **Recent approaches in differentiable rendering use 3D Gaussians as an underlying representation.**. These approaches enable fast reconstruction of 3D objects and scenes. This paper demonstrates how both approaches use the same underlying representation and how to enhance and export such reconstructions.

Gaussians, and performed quantitative experiments for object reconstruction and pose estimation. A year layer, the latter designed a fast rasterizer with a custom CUDA kernel for splatting [46, 54], used millions of Gaussians, and focused their system on reconstructing entire scenes to approach NeRF levels of fidelity. We demonstrate connections between and extensions to both methods.

1

This paper is focused on extending the Fuzzy Metaballs renderer to make it simpler, more robust, and to add additional features. We show that these recent papers are interoperable and render the same underlying representation. Since most well-established rendering techniques are built on triangle meshes, we demonstrate a reliable way to transform 3D Gaussian representations into meshes.

We summarize our contributions as follows:

- Develop a simplified version of Fuzzy Metaballs [34] for shape reconstruction from Gaussians (Section 3.2).

- Show how Fuzzy Metaballs [34] can be rendered without hyperparameters (Section 3.3).

- Show how to get per-pixel, differentiable optical flow and its benefits in reconstruction (Section 5).

- Demonstrate how to export meshes from shapes defined by 3D Gaussians (Section 6).

- Show that existing 3D Gaussian rendering methods [33, 34] are interoperable (Section 7).

- Develop a loss-based approach to reparameterizing Gaussians by splitting components (Section 8).

These techniques enable more powerful, flexible uses of these 3D Gaussian shape representations for applications involving shape reconstruction. Of note, these approaches require no pretraining and can be optimized directly on the scene of interest without dataset bias, making them applicable to robotics applications where robots may be interacting with novel objects, and are limited by onboard compute.

2. Related Work

A comprehensive overview of related work can be found in prior papers using 3D Gaussians, including Fuzzy Metaballs [34], 3D Gaussian Splatting [33] and VoGE [66].

Some early methods of building models from partial observations used generalized cylinders [2]. More commonly, methods build on top of triangle meshes, point clouds and surfels [57]. Differentiable renders have been built for these representations, initially for meshes [30, 39, 40]. These include custom backends that allow for fast GPU-based results [36], and high-quality results [53]. Other works focus on point clouds [26, 73]. Pulsar [37] uses spheres as its representation, which are equivalent to isotropic 3D Gaussians. Primitives-based rendering methods have benefits for both for composition [74] and tracking [43].

The earliest work using 3D Gaussians in rendering came from Blinn [4], originally called atoms, blobs or metaballs, and were the birth of implicit surfaces. Several methods these techniques [18, 23, 52, 63, 67–69]. Some renderers used rays while others used splatting [1], and some recent

differentiable renders build screen space Gaussians [50]. Others use Gaussians as a primary representation in computer vision [9–11, 16, 17, 22, 44] or render them via projection, search, or other techniques [25, 55, 62].

Gaussians can be seen as a fundamental building block that only uses the 1st (μ) and 2nd (Σ) order moments [27]. Point clouds only use μ. Oriented point clouds add a covariance eigenvector [9], and Gaussian Mixtures [11] use all the information. Connections in this space include error metrics [15, 45] and physics simulations [49].

There is work on connecting NeRF-style differentiable renderers to mesh representations. MobileNeRF [6] used the rasterization pipeline of commodity hardware to perform rendering of NeRF-like objects. VMesh [19] constructed a hybrid representation of volume and mesh. NeRFMeshing [58] learned a Signed Surface Approximation Network to distill NeRF representations into meshes.

3. Ray-Shape Intersections

Existing methods for intersecting 3D Gaussians and rays typically take two forms. Methods taking inspiration from NeRF family methods [33, 66] typically sort all intersections and use closer Gaussians to attenuate the contributions of further Gaussians. Fuzzy Metaballs [34] introduced a heuristic technique for blending intersections which did does not require sorting. Section 3.1 summarizes this technique, Section 3.2 presents a simplification, and Section 3.3 proposes a variation without hyperparameters. All three correctly render the same shapes, so objects optimized by one can be reproduced with the other.

3.1. Weighted Blending

In Fuzzy Metaballs [34], intersections between each ray ($\vec{v}$) and each Gaussian are computed separately and then combined with a weighted average. Each Gaussian is parameterized as mean ($\mu \in \mathbb{R}^3$), inverse root precision ($\Sigma^{-\frac{1}{2}}$) and a weight ($\lambda_i \geq 0$) where the $\sum_i \lambda_i = 1$.

Multivariate Gaussians are defined as

$$P(\vec{x}) = |\Sigma|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\vec{x} - \mu)^T \Sigma^{-1}(\vec{x} - \mu)\right), \quad (1)$$

and the unnormalized log distance for each Gaussian is

$$d(\vec{v}t) = -\frac{1}{2}\left[(vt - \mu)^T \Sigma^{-1}(vt - \mu)\right] + \log(\lambda_i), \quad (2)$$

which we will refer to as d_i when referring to evaluating the i-th Gaussian for a ray $\vec{v}$ (here we use the point of maximum likelihood, $t_i = \frac{\mu_i^T \Sigma_i^{-1} v}{v^T \Sigma_i^{-1} v}$, the linear approach [34]).

For each Gaussian, its intersection (t_i) and distance (d_i) are used to obtain weights (w_i). The final intersection is obtained as t_f, similar to OIT [12, 47]:

$$t_f = \frac{1}{\sum_i w_i} \sum_i w_i t_i. \quad (3)$$

Per-ray estimation of other properties can continue to use Equation (3). For example, t_i (distance from camera) can be replaced with $\vec{n_i}$ (normal) or $\vec{c_i}$ (color) and the same blending functions can be reused. For more details about normal computation, see Section 6. It is also helpful to think of the unnormalized Gaussian density as:

$$\delta_i = \exp(d_i), \tag{4}$$

The original Fuzzy Metaballs approach uses 5 hyperparameters $(\beta_1, \beta_2, \beta_3, \beta_4, \beta_5)$ to compute the weights and the transparency. There is also a shape scale (η) to account for shapes of different scales and return identical results. The following are the weight and α (quality/opacity) functions:

$$w_i = \exp\left(\beta_1 d_i \sigma\left(\frac{\beta_3}{\eta} t_i\right) - \frac{\beta_2}{\eta} t_i\right), \tag{5}$$

$$\alpha = \sigma\left(\beta_4 \sum_i \delta_i + \beta_5\right). \tag{6}$$

3.2. Two Parameter Model

In shape reconstruction from video, 3 of these hyperparameters are not necessary and we develop a simplified two parameter model that works in reconstruction settings.

First, β_3 was used to give some minor contribution to intersections behind the camera. While this does make the renderer more differentiable, it is not used in practice, so the σ function can be removed, leaving us with

$$w_i = \exp\left(\beta_1 d_i - \frac{\beta_2}{\eta} t_i\right). \tag{7}$$

Since the original paper [34] focused on rendering full, proper Gaussian Mixture Models (where $\sum_i \lambda_i = 1$), it required a normalizing factor β_4 to account for how much opaque a GMM should be. In the case of reconstruction, there is no need for strict GMMs, so learning β_4 can be the responsibilities of unnormalized λ_i. Additionally, the GMM focus suggested a smooth step function (sigmoid) for computing α, but shape reconstruction can simplify this to a decaying exponential, letting us drop the intercept term β_5. This allows us to have a simple, perhaps familiar [48, 65], expression for α:

$$\alpha = 1 - \exp\left(-\sum_i \delta_i\right). \tag{8}$$

We use this simplified two parameter model in most of our experiments. While we noticed some small differences due to losing the global normalization condition, they were minor and mostly in the space of pose estimation, which is not the focus of our experiments here for shape estimation.

3.3. Zero Parameter Model

With the simplified design given above, and the NeRF-style approaches in other 3D Gaussian papers [33, 66], we also investigate an alpha compositing variant of Fuzzy Metaball rendering that requires sorting all intersections and then computing transmission. The transmissions can be seen as weights and Equation (3) can be used to compute depth estimates (as well as normal and color estimates):

$$w_i = T_i \left(1 - \exp(-\delta_i)\right) \tag{9}$$

where T_i is the accumulated transmissions of earlier intersections

$$T_i = \exp\left(-\sum_j \delta_j 1[t_j < t_i]\right) \tag{10}$$

These equations create a version of differentiable rendering with metaballs that is hyperparameter-free, doesn't require splatting, and enables the same usage of weights for depth, normal and color computation as prior work [34]. As our experiments in Table 2 and Figure 7 show, this approach is slower but performs nearly identically.

4. Shape Reconstruction

The input to the system is a video and a single masked frame. COLMAP [61] produces poses, XMem [7] propagates masks, Unimatch [70] produces flow, and 3D Gaussians optimized to fit the data. The shape converges quickly since we use a ray-based differentiable renderer and are able to sample minibatches that includes pixels across all frames.

Our differentiable rendering code is based on Fuzzy Metaballs [34], which is built in JAX [5] and allows for reconstructions on both the CPU and GPU. With an Nvidia GTX 1080, we can do memory for image sequences of roughly 250×130, while our CPU experiments are typically closer to 125×65. Both sets of experiments typically converge in less than a minute on commodity hardware. We use 40 Gaussians as in prior work, for ease of comparison.

Differentiable renders provide flexibility for many loss functions in shape reconstruction. For our experiments, we combine of cross-entropy loss (L_M) for objects segmentation masks, and L_1 losses for color and flow (Section 5), weighted by the object mask. Estimated alpha is clipped to $[10^{-6}, 1 - 10^{-6}]$. For α, color (c), and optical flow (f) we use the following loss function:

$$L_M = \alpha \cdot \log(\hat{\alpha}) + (1 - \alpha) \cdot \log(1 - \hat{\alpha}) \tag{11}$$

$$L_C = \alpha \cdot ||c - \hat{c}||_1 \tag{12}$$

$$L_F = \alpha \cdot ||f - \hat{f}||_1 \tag{13}$$

$$L = L_M + \lambda_C L_C + \lambda_F L_F \tag{14}$$

In practice, we use colors that have had gamma corrected back to linear intensity, and use a sigmoid to map from an

Figure 2. **Shape reconstruction** using 40 3D Gaussians and converging in under one minute, with color. (See Section 4 for details). All objects are reconstructed from videos in the CO3D [60] dataset.

unconstrained parameterization to $[0, 1]$. Flow is measured in the scale of half the shorter image dimension. This lets us set $\lambda_C = 4.5$ and $\lambda_F = 210$ for all of our experiments. We used identical settings for all our experiments: initialization is from a randomized small sphere of Gaussians, and we use fixed parameters for learning rate, canonical rescaling, batch size, and all other known parameters. Learning rates are automatically decayed based on statistical criteria [34].

We use the Adam [35] optimizer and rescale all scenes to a canonical size based on camera pose distances to balance the optimization of means and precisions [34] that occurs when using re-scaling optimizers. We randomly sample minibatches of 50,000 rays from across the entire sequence, which leads to extreme fast convergence, often getting a reasonable shape in well under an epoch (Figure 7). We use the per-Gaussian simple colors used in prior work [34] for simplicity, but extensions to Spherical Harmonics are possible for greater color fidelity per Gaussian [14, 33].

Examples of our reconstructions can be seen Figure 2. Even though the optimization only takes minutes, operates on reasonably low resolution images, with reasonably few Gaussians, we can still see good results. Depth estimates are able to capture small geometric details (notice the kickstand and both mirrors on the motorcycle). Despite the initialization being a small, invisible sphere of Gaussians, the optimization procedure is able to reconstruct shapes with rich geometry (such as the bicycle and the plant). Lastly, the color results look reasonably realistic. Although separate Gaussians must be used to paint 2D flat textures onto surfaces, reasonable results are obtained for the toy truck and the skateboard reconstructions. In the toy truck, grey side and roof panel details appear in the reconstruction. In the skateboard, the painted curve shape is also approximately modeled in the reconstruction, as are both wheels.

We obtained similarly good results with both Section 3.1 and Section 3.3, but all results in Figure 2 use the faster, two parameter model for optimization and the resulting visuals.

5. Reconstructing with Optical Flow

Many approaches to 3D reconstruction focus only on reconstructing the independent images from the given sequence [33], including all the 3D Gaussian methods [33, 34, 66]. However, in practice, these image sequences are often collected by cell phone videos [60] and have a strong temporal prior. Inspired by work in reconstructing 4D scenes [38, 71, 72], we leverage optical flow in producing more precise 3D reconstructions of static objects.

Optical flow provides a hypothesis of surface correspondence, which regularizes the shape reconstruction. Correspondence can be essential in classic techniques for shape estimation [29, 42]. Sparse particle video trackers extend this, and obtain long-term video correspondences [20].

Optical flow is a useful signal since it is a local estimator, and is robust to the long-term lighting changes that occur when typical users capture scenes under auto-exposure [28]. This makes it perhaps a more appropriate loss term than color models, which would be sensitive to lighting changes. Additionally, color often means texture, which implies high frequency texture details that can be difficult to reconstruct [48], and perhaps efficient shape estimation can do without.

The benefits of flow can be from our experiments, qualitatively in Figure 4 and quantitatively in Table 1. One interesting result is that even classic optical flow [13] provides helpful cues to the shape optimization, even though it is very noisy (see Figure 4a). Even better, state-of-the-art flow methods such as Unimatch [70] are extremely fast (real-time on GPU hardware) and have learned good priors even in texture-less areas. Using such learned flow maps (Figure 4b) can help shape estimation greatly. After optimization, predicted model flow (Figure 4d) is very similar to the given flow estimate from the network that was used to supervise it. Incorporating flow also produces significantly smoother depth maps, as can be seen in Fig. 3.
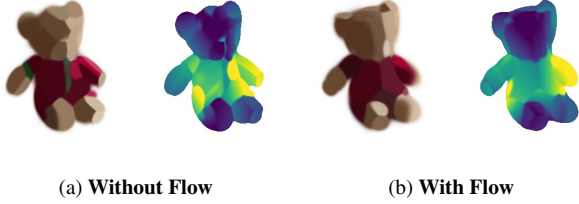
(a) **Without Flow**   (b) **With Flow**

Figure 3. **Optical flow improves reconstruction**. We show the results of fitting a set of 3D Gaussians to a CO3D [60] sequence. Without a flow loss term, colors are estimated well but the shape is not. After adding flow, color fidelity is sacrificed but shape estimation improves.

In our experiments, optical flow slightly hurts the color fidelity of the reconstructed models, but provides much more accurate shape reconstruction, as can be seen in Figure 3. After adding a flow loss term, this surrogate estimate of surface correspondence helps resolve concavity/convexity ambiguity from the silhouette and color loss terms. For example, the body of the teddy bear becomes smooth and its arms become well defined.

With the ray-based differentiable rendering of 3D Gaussians, it is reasonably easy to compute per-pixel optical flow. It only requires taking the 3D coordinate obtained from Equation (3), and transforming it with adjacent camera poses and projecting back into camera coordinates.

In our implementation, we pass all camera poses into the rendering function and represent the camera with a single inverse focal length parameter (making the assumptions that the images lack distortion, the projection is in the center of the image, and that the pixels are square). After computing the depth image for a given frame, we transform the point cloud (forwards and backwards) and project the transformed coordinates into the image. The changes in coordinates is the direct, per-ray optical flow estimate. Without using another source to regularize the optical flow, we get reasonable estimates but with some artifacts due to shape uncertainty (as can be see in Figure 4c).

We include estimates for both forward flow (pose i to pose $i + 1$) and backward flow (pose i to pose $i - 1$) from our differentiable renderer, for each ray.

	Depth Error	Runtime (seconds)
No Color or Flow	0.271	17
Color	0.262	15
Color & Classic Flow [13]	0.237	**14**
Color & Learned Flow [70]	**0.155**	15

Table 1. **Optical flow helps reconstruction** of CO3D sequences [60]. For details see Section 5.

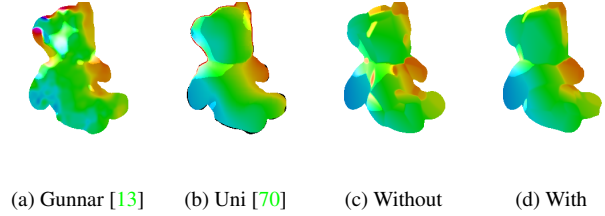(a) Gunnar [13]   (b) Uni [70]   (c) Without   (d) With

Figure 4. **Optical Flow Quality**. (a) shows a classic optical flow estimate. (b) shows a modern, learned optical flow estimate. (c) and (d) show the estimated flow for the same frame after fitting 3D Gaussian model; respectively without adding a flow loss and after adding a flow loss term to fit the estimate from (a). Using standard optical flow coloring [3].

6. Exporting Meshes

Many successful differentiable renderers realize that fast, efficient, robust differentiable rendering often require smooth, fuzzy and indefinite surface representations [33, 34, 48]. For optimizing shapes from videos, it is helpful to have some degree of softness to aid gradient flow.

On the other hand, commercial rasterization pipelines in most desktop and mobile GPUs typically operate on triangle meshes [6, 19]. Additional, the field of shape processing often prefers not just definite surfaces in the form of meshes, but watertight meshes [24, 75]. Differentiable mesh renderers typically bridge this divide with an explicit spatial smoothness term [39, 59].

Instead, 3D Gaussians, neural surfaces, and other similar methods are implicit surface methods, where the genus of the object can change during optimization. Many existing works export meshes using marching cubes [34, 41, 48], where a volumetric grid is evaluated and triangles are produced at edges crossing a particular level set threshold. While fast and simple, this can produce non-watertight meshes. Additionally, it can require searching over an ideal threshold to find which level set of the implicit surface matches the explicit surface best [34].

In this work, we leverage the surface definition used by the underlying differentiable renderer, and then solve the appropriate Poisson equation [31, 32]. In Poisson surface reconstruction, oriented point sets (points with normals indicating the local tangent plane of the surface) are used as input to solve a Poisson equation:

$$\nabla \cdot \nabla \chi = \nabla \cdot \vec{V}. \tag{15}$$

Solving these equations can be done via well-conditioned sparse linear systems [31]. Each point in the oriented point cloud provides an estimate of the local gradient of the indicator function (which points are inside the surface of the object). A nice property of Poisson Surface solvers is that their solutions always produce watertight meshes, as they solve for an indicator volume, which produces a surface that is a

5

Figure 5. **Colored Mesh Exports** reconstructed from CO3D sequences [60] with 40 3D Gaussians and exported to Blender (Section 6).

$\mathbb{R}^2$ manifold folded in $\mathbb{R}^3$. This process can be done over a basis function set of B-splines with compact support.

To produce an oriented point set, we simply go over all of our training views and render the point cloud of the object, to produce point locations $p = (x, y, z)$. The orientation of each point can be produced in two different ways (shown in Section 6). The first, and most general, is to use the rasterization locality by taking a horizontal (p_x) and a vertical (p_y) screen-space neighbor for each point (p) and taking their cross product:

$$\vec{n} = \frac{(p - p_x) \times (p - p_y)}{||(p - p_x) \times (p - p_y)||}. \quad (16)$$

This technique works for any differentiable renderer producing images on a grid (hence rasterization), but produces poor results on discontinuities.

An alternative approach for 3D Gaussians is to re-use Equation (3) to blend all the local estimates of the normal ($\vec{n_i}$ instead of t_i) into a final estimate. The local estimate of the normal is given by the derivative of Equation (1).

$$\vec{n_i} = \frac{\Sigma^{-1}(vt_i - \mu_i)}{||\Sigma^{-1}(vt_i - \mu_i)||}. \quad (17)$$

In general, the sign and scale of the normal is fixed: the sign of the normal should face the camera created it, and the $||\vec{n_i}|| = ||\vec{n_f}|| = 1$. Both of these techniques can be seen in Section 6. In practice, the blended definition is preferred as it requires no neighbors.

For producing meshes from our renderer, we typically perform the faster optimization with Section 3.1. and the rendering equation from Section 3.3 can directly render that representation with no changes. The alpha compositing definition is preferred for mesh exporting as it is more view consistent than the heuristic blending. Lastly, we can reject any intersections that fail a quality threshold of a direct intersection, $\max_i(w_i) \leq \epsilon$ which is typically set to 0.9. The Poisson surface reconstruction produces a surface interpolation, and so the sparsity of point samples is not a problem. Additionally, we can export a colored, oriented point cloud, where the color is either the reconstructed color or the color of the images themselves at those points (see Figure 5). We obtained object reconstructions by solving Equation (15) with Dirichlet boundary constraints [32].

(a) **Ficus via Weighted Blending (Section 3.1)**

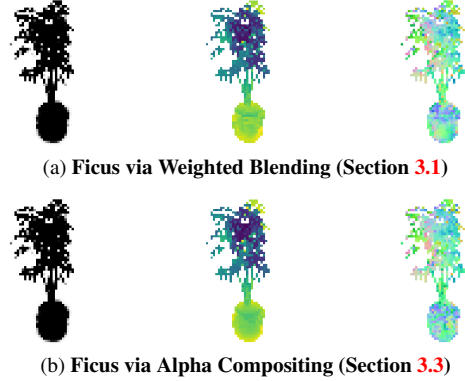(b) **Ficus via Alpha Compositing (Section 3.3)**

Figure 6. **3D Gaussians are Fuzzy Metaballs** Reconstructed ficus from Gaussian Splatting [33], rendered with Fuzzy Metaballs [34]. Shown are opacity, depth & normals.

(a) **Optimization via Weighted Blending (Section 3.1)**

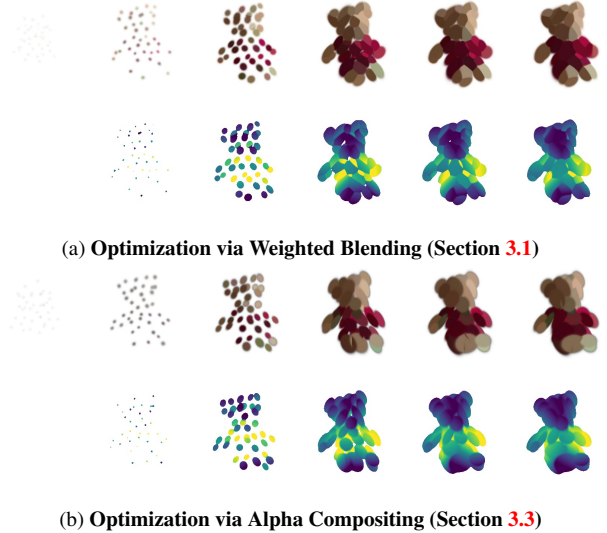(b) **Optimization via Alpha Compositing (Section 3.3)**

Figure 7. **Both forms of rendering behave similarly.** Shown are 1%, 4%, 7%, 12%, 16% and 20% of the first epoch.

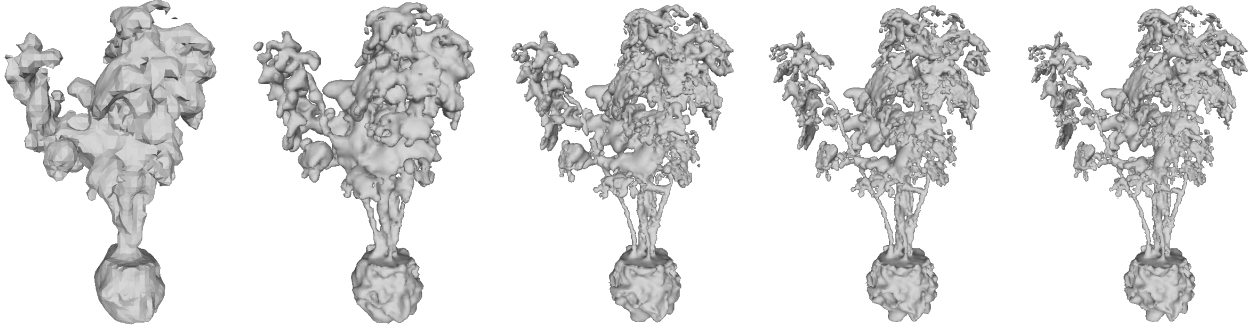Figure 8. **Visualization of Normals** for 3D Gaussians.

Figure 9. **Visualization of Mesh Export** This ficus shape was reconstructed using the Gaussian Splatting [33] code but rendered as an oriented point cloud with Fuzzy Metaballs [34] and reconstructed with a Poisson Solver [31, 32]. Left to right are Poisson tree depths of 6 through 10. While deeper tree depths produce more detailed reconstruction, noise and artifacts are also amplified. The mesh export is based on an oriented point cloud exported from 100 images of size 80×60. It is interesting to see that precise details like the various steps become visible, even when the forward passes were low resolution (see Figure 6 for a visual example of the coarseness generated by the renderer).

7. Interoperability

We describe how Fuzzy Metaballs [34] and 3D Gaussian Splatting [33] share a similar underlying shape representation. We demonstrate this by showing results in Figures 6 and 9 where the initial shape reconstruction was performed using the 3D Gaussian Splatting paper [33], and then directly converted and rendered using the methods in Sections 3.1 and 3.3 and then exported using the oriented points using Section 6.

Our experiments in Figure 7 and Section 6 show that Sections 3.1 and 3.3 use mutually compatible definitions of shape representation. However, the Gaussian Splatting [33] work uses an entirely different code base, in a different framework, optimizing scenes instead of objects, with no object masks, with a custom CUDA kernel, and preferring α to δ estimates. However, since both are using 3D Gaussians, we can render one with the other.

We convert the Gaussian Splatting method to be compatible with our approach with only a few steps. Means are used directly, each Σ are converted to $\Sigma^{-\frac{1}{2}}$, and the α for each Gaussian is replaced. For simplicity, we ignore $\alpha_i < 0.5$ and set $\lambda_i = \log(80)$ for the remaining 3D Gaussians, which we found works reasonably well[1]. About 90% of Gaussians had insufficient α, creating a ten times speedup in our experiments. For weighted blending experiments, we reused the settings of $\beta_1 = 21.4$ and $\beta_2 = 3.14$ from our prior experiments.

As can be seen in Figure 6, both techniques are able to capture the fine stem and leaf structures in the reconstructed ficus plant. The weighted blending technique demonstrates

smoother normals, but the mesh export uses the alpha compositing method and reasonable meshes can be obtained.

In Figure 9 we show Poisson surface reconstructions of the ficus, where the oriented point cloud used for solving the equation was generated by our renderer, but the original reconstruction was made with 3D Gaussian splatting. We show reconstructions at different tree depths, showing an increase in details, and an increase in noise, at finer scales. The solver can recover reasonably fine details considering considering the low resolution of the oriented point clouds.

This extends the utility of the 3D Gaussian Splatting approach to be rendered with our fast methods that have viable JAX [5] CPU and GPU backends. Our approach enables per-ray depth computation, normals and mesh exporting (Section 6). The ficus data was provided without color, so we show colored exports in Figure 5. Mesh exports provides an interconnect with most 3D creation tools.

	CPU	**GPU**
Weighted Blending (Section 3.1)	4.94 μs	226.6 ns
Alpha Compositing (Section 3.3)	12.7 μs	377.6 ns

Table 2. **Runtime per ray, for an entire iteration** with a i5-7267U CPU and a GTX 1080 GPU. This was done with a 40 Gaussian model and these times include memory transfer times and forwards and backwards passes. Alpha compositing drops the need for hyper-parameters, in exchange for 200% slower runtimes on CPU and 50% slower runtimes on GPU. Since both approaches behave similarly (Fig. 7) and are interoperable (Sec. 6), this creates a trade-off between performance and simplicity.

---

[1]Inverse sigmoid conversion of $\lambda = -C \log(1 - \alpha)$ maybe also be reasonable with appropriate C, which we did not search for.

N = 40 N = 45 N = 51

Figure 10. **Re-parameterized 3D Gaussians** to minimize apparent loss. N is the number of mixtures used. The appearance of the tie and the sleeve in green can be seen with added Gaussians.

8. Splitting Gaussians

While the 3D Gaussian Splatting [33] explores some heuristic techniques for merging and splitting of Gaussians, here we develop an alternative, deterministic approach to modifying the number of Gaussians in the reconstruction.

We split 3D Gaussians after our initial model has converged according to statistical criteria [34]. And we then repeat the optimization process. Two steps of this are shown in Figure 10. The splitting and removal process is based on removing Gaussians that contribute minimally to the reconstruction and splitting Gaussians that are given too much reconstruction responsibility based on the chosen loss.

To compute the set of Gaussians that minimally contribute, we compute the means and standard deviations of the weights assigned to each Gaussian μ_λ and σ_λ and remove Gaussians that satisfy

$$\lambda_i \leq \mu_\lambda - z_\lambda \sigma_\lambda, \tag{18}$$

where z_λ is a z-score typically set to 2.

To compute the splitting criteria, we take a random batch of rays (typically about 5% of the dataset) and compute the per-ray loss (e.g. Eq. (14) but any reconstruction loss is viable). Each ray also has an associated set of computed weights w_i (Eqs. (7) and (9)). Over this batch of M rays, we estimate the average loss associated with each Gaussian as $\bar{l}_i = \frac{1}{M} \sum_j^M = L_j \cdot w_i$. We compute the means and standard deviations of these losses as $\mu_{\bar{l}}$ and $\sigma_{\bar{l}}$ and split Gaussians that satisfy

$$\bar{l}_i \geq \mu_{\bar{l}} + z_{\bar{l}} \sigma_{\bar{l}}, \tag{19}$$

where $z_{\bar{l}}$ is a z-score typically set to 1.

Gaussians are split by forming two Gaussians by using the properties of the half-normal distribution [8]. The two new means are generated by shifting the initial mean (μ_i) in opposite directions by the direction of the dominant eigenvector ($\vec{v_1}$), and by the appropriate factor of the dominant eigenvalue (σ_1) of the covariance matrix (Σ_i):

$$\mu_{a,b} = \mu \pm \sigma_1 \vec{v_1} \sqrt{\frac{2}{\pi}}. \tag{20}$$

Both new Gaussians are given identical covariance matrices, reconstructed from the initial eigenvalues and eigenvectors but with a scaled dominant eigenvalue:

$$\sigma_{1_{a,b}} = \sigma_1 \sqrt{1 - \frac{2}{\pi}}. \tag{21}$$

The scaling factors $\sqrt{\frac{2}{\pi}} \approx 0.8$ and $\sqrt{1 - \frac{2}{\pi}} \approx 0.6$ are based on the properties of the half-normal distribution.

We replicate the initial weights (λ) and colors (c_i) for the new Gaussians, with noise ($\epsilon_c = \epsilon_w = 0.1$) in their unconstrained parameterization space to avoid issues with coupled gradients during further optimization:

$$\log(\lambda_{a,b}) = \log(\lambda) + \mathcal{N}(0, \epsilon_w), \tag{22}$$
$$\sigma^{-1}(c_{a,b}) = \sigma^{-1}(c) + \mathcal{N}(0, \epsilon_c). \tag{23}$$

This approach to splitting is deterministic and allows for increasing detail in an iterative way, as shown in Fig. 10.

9. Discussion

With the increased deployment of vision systems in everyday environments, there is a need for flexible, efficient, and easily computed shape reconstructions. The recent developments in photorealistic differentiable rendering make close the dream of virtual systems accurately capturing everyday objects in the virtual world. 3D Gaussians, or metaballs, are a simple and powerful representation for shapes that enables easy reconstruction, as has been shown by prior work [33, 34, 66]. These techniques extend and interconnect these approaches and provide them additional flexibility.

The equivalence of weighted blending and alpha compositing approaches provides a wider array of options — one being significantly faster and the other lacking hyperparameters. The computation of per-ray blended normals allows for reliable mesh exporting, connecting to other 3D techniques and methods, without the need to pick thresholds for marching cubes [41], and producing watertight meshes via Poisson reconstruction [32]. We show a more deterministic, grounded approach to performing reparameterization of Gaussians. Lastly, optical flow as a regularizer and prior for surface correspondences can easily and reliably improve the quality of shape reconstructions.

10. Conclusion

We have extended existing approaches for differentiable rendering of 3D Gaussians for speed, simplicity and flexibility. This expanded flexibility should allow these representations to be used in more places, for more applications, and potentially across a wider array of computational platforms than before.

References

[1] Bart Adams, Toon Lenaert, and Philip Dutré. Particle splatting: Interactive rendering of particle-based simulation data. Report CW 453, KU Leuven, July 2006. 2

[2] Gerald Jacob Agin. *Representation and Description of Curved Objects*. PhD thesis, Stanford University, CA, USA, 1972. 2

[3] Simon Baker, Stefan Roth, Daniel Scharstein, Michael J. Black, J.P. Lewis, and Richard Szeliski. A database and evaluation methodology for optical flow. In *International Conference on Computer Vision*, pages 1–8, 2007. 5

[4] James F. Blinn. A generalization of algebraic surface drawing. *ACM Trans. Graph.*, 1(3):235–256, July 1982. 1, 2

[5] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. 3, 7

[6] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *The Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 2, 5

[7] Ho Kei Cheng and Alexander G. Schwing. XMem: Long-term video object segmentation with an atkinson-shiffrin memory model. In *ECCV*, 2022. 3

[8] Cuthbert Daniel. Use of half-normal plots in interpreting factorial two-level experiments. *Technometrics*, 1(4):311–341, 1959. 8

[9] Ben Eckart, Kihwan Kim, and Jan Kautz. Hgmr: Hierarchical gaussian mixtures for adaptive 3d registration. In *ECCV 2018*, pages 730–746, 2018. 2

[10] Ben Eckart, Kihwan Kim, Alejandro Troccoli, Alonzo Kelly, and Jan Kautz. MLMD: Maximum Likelihood Mixture Decoupling for Fast and Accurate Point Cloud Registration. In *3DV*, pages 241–249, 2015. 2

[11] Ben Eckart, Kihwan Kim, Alejandro Troccoli, Alonzo Kelly, and Jan Kautz. Accelerated Generative Models for 3D Point Cloud Data. In *CVPR*, pages 5497–5505, 2016. 2

[12] Eric Enderton, Erik Sintorn, Peter Shirley, and David Luebke. Stochastic transparency. In *I3D '10: Proceedings of the 2010 symposium on Interactive 3D graphics and games*, pages 157–164, New York, NY, USA, 2010. 2

[13] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In Josef Bigun and Tomas Gustavsson, editors, *Image Analysis*, pages 363–370, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. 4, 5

[14] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *CVPR*, 2022. 4

[15] Michael Garland and Paul S Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH*, pages 209–216, 1997. 2

[16] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. Local deep implicit functions for 3d shape, 2020. 2

[17] Kshitij Goel, Nathan Michael, and Wennie Tabib. Probabilistic point cloud modeling via self-organizing gaussian mixture models. *IEEE Robotics and Automation Letters*, 8(5):2526–2533, 2023. 2

[18] Olivier Gourmel, Anthony Pajot, Mathias Paulin, Loic Barthe, and Pierre Poulin. Fitted BVH for Fast Raytracing of Metaballs. *Computer Graphics Forum*, 3:7 – 288, 2010. 2

[19] Yuan-Chen Guo, Yan-Pei Cao, Chen Wang, Yu He, Ying Shan, Xiaohu Qie, and Song-Hai Zhang. Vmesh: Hybrid volume-mesh representation for efficient view synthesis, 2023. 2, 5

[20] Adam W. Harley, Zhaoyuan Fang, and Katerina Fragkiadaki. Particle video revisited: Tracking through occlusions using point trajectories. In *ECCV*, 2022. 4

[21] Paul S. Heckbert. Fun with gaussians. *SIGGRAPH '86 Advanced Image Processing seminar notes*, 1986. 1

[22] Amir Hertz, Rana Hanocka, Raja Giryes, and Daniel Cohen-Or. Pointgmm: a neural gmm network for point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020. 2

[23] Robert Horvath. Image-space metaballs using deep learning. Master's thesis, Faculty of Informatics, TU Wien, July 2019. 2

[24] Shi-Min Hu, Zheng-Ning Liu, Meng-Hao Guo, Junxiong Cai, Jiahui Huang, Tai-Jiang Mu, and Ralph R. Martin. Subdivision-based mesh convolution networks. *ACM Trans. Graph.*, 41(3):25:1–25:16, 2022. 5

[25] H. Huang, H. Ye, Y. Sun, and M. Liu. Gmmloc: Structure consistent visual localization with gaussian mixture models. *IEEE Robotics and Automation Letters*, 5(4):5043–5050, 2020. 2

[26] Eldar Insafutdinov and Alexey Dosovitskiy. Unsupervised learning of shape and pose with differentiable point clouds, 2018. 2

[27] S. Julier, J. Uhlmann, and H.F. Durrant-Whyte. A new method for the nonlinear transformation of means and covariances in filters and estimators. *IEEE Transactions on Automatic Control*, 45(3):477–482, 2000. 2

[28] Kim Jun-Seong, Kim Yu-Ji, Moon Ye-Bin, and Tae-Hyun Oh. Hdr-plenoxels: Self-calibrating high dynamic range radiance fields. In *ECCV*, 2022. 4

[29] Takeo Kanade, Peter Rander, and P.J. Narayanan. Virtualized reality: constructing virtual worlds from real scenes. *IEEE MultiMedia*, 4(1):34–47, 1997. 4

[30] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer, 2017. 2

[31] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson Surface Reconstruction. In Alla Sheffer and Konrad Polthier, editors, *Symposium on Geometry Processing*. The Eurographics Association, 2006. 5, 7

9

[32] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):1–13, 2013. 5, 6, 7, 8

[33] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (ToG)*, 42(4), July 2023. 1, 2, 3, 4, 5, 6, 7, 8

[34] Leonid Keselman and Martial Hebert. Approximate differentiable rendering with algebraic surfaces. In *European Conference on Computer Vision (ECCV)*, Oct 2022. 1, 2, 3, 4, 5, 6, 7, 8

[35] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015. 4

[36] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics*, 39(6), 2020. 2

[37] Christoph Lassner and Michael Zollhöfer. Pulsar: Efficient sphere-based neural rendering. *arXiv:2004.07484*, 2020. 2

[38] Zhengqi Li, Simon Niklaus, Noah Snavely, and Oliver Wang. Neural scene flow fields for space-time view synthesis of dynamic scenes. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 4

[39] 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 (ICCV)*, October 2019. 1, 2, 5

[40] Matthew M. Loper and Michael J. Black. Opendr: An approximate differentiable renderer. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *ECCV*, pages 154–169, Cham, 2014. 2

[41] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987. 5, 8

[42] David G. Lowe and Thomas O. Binford. Interpretation Of Geometric Structure From Image Boundaries. In James J. Pearson, editor, *Techniques and Applications of Image Understanding*, volume 0281, pages 224 – 231. International Society for Optics and Photonics, SPIE, 1981. 4

[43] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. 2023. 2

[44] Martin Magnusson. *The three-dimensional normal-distributions transform: an efficient representation for registration, surface analysis, and loop detection*. PhD thesis, Örebro universitet, 2009. 2

[45] Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, pages 49–55, 1936. 2

[46] Petr Man. Generating and real-time rendering of clouds. *Central European seminar on computer graphics*, 2006. 1

[47] Morgan McGuire and Louis Bavoil. Weighted blended order-independent transparency. *Journal of Computer Graphics Techniques (JCGT)*, 2(2):122–141, December 2013. 2

[48] 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 Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 405–421, Cham, 2020. Springer International Publishing. 1, 3, 4, 5

[49] Brian Mirtich. Fast and accurate computation of polyhedral mass properties. *JGT*, 1(2):31–50, 1996. 2

[50] Jan U. Müller, Michael Weinmann, and Reinhard Klein. Unbiased gradient estimation for differentiable surface splatting via poisson sampling. In *European Conference on Computer Vision (ECCV)*, 2022. 2

[51] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022. 1

[52] Shigeru Muraki. Volumetric shape description of range data using "blobby model". In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, page 227–235, New York, NY, USA, 1991. Association for Computing Machinery. 2

[53] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: A retargetable forward and inverse renderer. *ACM Trans. Graph.*, 38(6), nov 2019. 2

[54] Manjushree Nulkar and Klaus Mueller. Splatting with shadows. In Klaus Mueller and Arie E. Kaufman, editors, *Volume Graphics 2001*, pages 35–49, Vienna, 2001. Springer Vienna. 1

[55] C. O'Meadhra, W. Tabib, and N. Michael. Variable resolution occupancy mapping using gaussian mixture models. *IEEE Robotics and Automation Letters*, 4(2):2015–2022, 2019. 2

[56] Karl Pearson. Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London. A*, 185:71–110, 1894. 1

[57] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, page 335–342, USA, 2000. 2

[58] Marie-Julie Rakotosaona, Fabian Manhardt, Diego Martin Arroyo, Michael Niemeyer, Abhijit Kundu, and Federico Tombari. Nerfmeshing: Distilling neural radiance fields into geometrically-accurate 3d meshes, 2023. 2

[59] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501*, 2020. 5

[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 *International Conference on Computer Vision*, 2021. 4, 5, 6

[61] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016. 1, 3

[62] Kumar Shaurya Shankar and Nathan Michael. Mrfmap: On-line probabilistic 3d mapping using forward ray sensor models. In *Robotics: Science and Systems*, 2020. 2

[63] László Szécsi and Dávid Illés. Real-time metaball ray casting with fragment lists. In *Eurographics*, 2012. 2

[64] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David Mcallister, Justin Kerr, and Angjoo Kanazawa. Nerfstudio: A modular framework for neural radiance field development. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Proceedings*. ACM, 2023. 1

[65] Delio Vicini, Wenzel Jakob, and Anton Kaplanyan. A non-exponential transmittance model for volumetric scene representations. *ACM Trans. Graph.*, 40(4), jul 2021. 3

[66] Angtian Wang, Peng Wang, Jian Sun, Adam Kortylewski, and Alan Yuille. VoGE: A differentiable volume renderer using gaussian ellipsoids for analysis-by-synthesis. In *The Eleventh International Conference on Learning Representations*, 2023. 2, 3, 4, 8

[67] Lee Westover. Footprint evaluation for volume rendering. *SIGGRAPH Comput. Graph.*, 24(4):367–376, sep 1990. 2

[68] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Data structure forsoft objects. *The Visual Computer*, 2(4):227–234, Aug 1986. 2

[69] Geoff Wyvill and Andrew Trotman. Ray-tracing soft objects. In Tat-Seng Chua and Tosiyasu L. Kunii, editors, *CG International*, pages 469–476, Tokyo, 1990. Springer Japan. 2

[70] Haofei Xu, Jing Zhang, Jianfei Cai, Hamid Rezatofighi, Fisher Yu, Dacheng Tao, and Andreas Geiger. Unifying flow, stereo and depth estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023. 3, 4, 5

[71] Gengshan Yang, Deqing Sun, Varun Jampani, Daniel Vlasic, Forrester Cole, Huiwen Chang, Deva Ramanan, William T Freeman, and Ce Liu. Lasr: Learning articulated shape reconstruction from a monocular video. In *CVPR*, 2021. 4

[72] Gengshan Yang, Deqing Sun, Varun Jampani, Daniel Vlasic, Forrester Cole, Ce Liu, and Deva Ramanan. Viser: Video-specific surface embeddings for articulated 3d shape reconstruction. In *NeurIPS*, 2021. 4

[73] Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. Differentiable surface splatting for point-based geometry processing. *ACM Transactions on Graphics*, 38(6):1–14, Nov 2019. 1, 2

[74] Xiaoshuai Zhang, Abhijit Kundu, Thomas Funkhouser, Leonidas Guibas, Hao Su, and Kyle Genova. Nerflets: Local radiance fields for efficient structure-aware 3d scene representation from 2d supervision. *CVPR*, 2023. 2

[75] Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10, 000 3d-printing models. *CoRR*, abs/1605.04797, 2016. 5