Approximate Differentiable Rendering
with Algebraic Surfaces

Leonid Keselman and Martial Hebert

Carnegie Mellon University, Pittsburgh PA, USA
{lkeselma,hebert}@cs.cmu.edu

Abstract. Differentiable renderers provide a direct mathematical link between an object's 3D representation and images of that object. In this work, we develop an approximate differentiable renderer for a compact, interpretable representation, which we call Fuzzy Metaballs. Our approximate renderer focuses on rendering shapes via depth maps and silhouettes. It sacrifices fidelity for utility, producing fast runtimes and high-quality gradient information that can be used to solve vision tasks. Compared to mesh-based differentiable renderers, our method has forward passes that are 5x faster and backwards passes that are 30x faster. The depth maps and silhouette images generated by our method are smooth and defined everywhere. In our evaluation of differentiable renderers for pose estimation, we show that our method is the only one comparable to classic techniques. In shape from silhouette, our method performs well using only gradient descent and a per-pixel loss, without any surrogate losses or regularization. These reconstructions work well even on natural video sequences with segmentation artifacts.
Project page: https://leonidk.github.io/fuzzy-metaballs

Keywords: differentiable rendering, metaballs, implicit surfaces, pose estimation, shape from silhouette, gaussian mixture models

1   Introduction

Rendering can be seen as the inverse of computer vision: turning 3D scene descriptions into plausible images. There are countless classic rendering methods, spanning from the extremely fast (as used in video games) to the extremely realistic (as used in film and animation). Common to all of these methods is that the rendering process for opaque objects is discontinuous; rays that hit no objects have no relationship to scene geometry and when intersections do occur, they typically only interact with the front-most component of geometry.

Differentiable Rendering is a recent development, designing techniques (often sub-gradients) that enable a more direct mathematical relationship between an image and the scene or camera parameters that generated it. The easy access to derivatives allows for statistical optimization and natural integration with gradient-based learning techniques. There exist several recent differentiable renderers which produce images comparable in fidelity to classic, non-differentiable, photorealistic rendering methods [33,46,49,75].

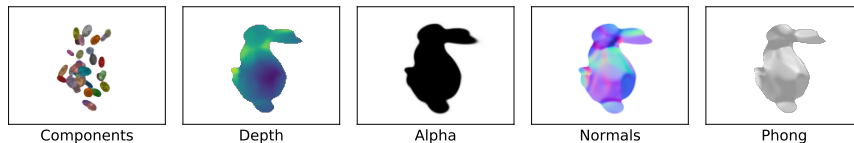| Components | Depth | Alpha | Normals | Phong |

Fig. 1: **Our differentiable renderer producing images of Stanford bunny**, using a representation with 400 parameters. From left to right: the 40 components at one standard deviation, followed by our differentiable renderer generating depth, alpha, surface normals and a shaded image.

Our paper presents a different approach: a differentiable renderer focused on utility for computer vision tasks. We are interested in the quality and computability of gradients, not on matching the exact image formation task. We willingly sacrifice fidelity for computational simplicity (and hence speed). Our method focuses on a rendering-like process for shapes which generates good gradients that rapidly lead to viable solutions for classic vision problems. Other methods may produce more pleasing images, but we care about the quality of our local minima and our ability to easily find those minima. Our experiments show how, compared to classic methods, differentiable renderers can be used to solve classic vision problems using only gradient descent, enabling a high degree of robustness to noise such as under-segmented masks or depth sensor artifacts.

Our approach is built on a specific 3D representation. Existing representations often have undesirable properties for rendering or optimization. Point clouds require splatting or calculating precise point sizes [74]. Meshes explicitly represent the object surface, making changes of genus difficult. Other representations require optimization or numerical estimation of ray-shape intersections [6,46]. Our proposed method is formulated with independent rays, represents object surfaces implicitly and computes ray termination in closed form.

Most existing differentiable renders focus on GPU performance. However, GPUs are not always available. Many robotics platforms do not have a GPU [64] or find it occupied running object detection [73], optical flow [61] or a SLAM method [47]. While a single method may claim to be real-time on a dedicated GPU [62], an autonomous system requires a sharing of resources. To run in parallel with the countless GPU-friendly techniques of today, CPU-friendly methods are desirable. Thus, while our method is implemented in JAX [8], supporting CPU and GPU backends, our focus is typically on CPU runtimes.

Lastly, in the era of deep learning, techniques which support gradient-based optimization are desirable. Since our objects have an explicit algebraic form, gradients are simple and easy to compute. Importantly, every pixel has a non-zero (if very slight) relationship with each piece of geometry in the scene (even those behind the camera!). This allows for gradient flow (up to machine precision), even when objects start far from their initialization. While this can also true of large over-parmaterized implicit surfaces (such as NeRF [46]), our representation is extremely compact and each parameter has approximate geometric meaning.

2   Related Work

Early work in 3D shape representation focused on building volumes from partial observations [3] but most modern methods instead focus on surface representation. Meshes, point clouds and surfels [51] focus on representing the exterior of an object. In contrast, our method works by representing volumes, and obtaining surface samples is implicit; similar to recent work on implicit neural surfaces [46].

In using low-fidelity representations, our work is hardly unique. Often learning-based methods settle for pseudorendering [37] or even treating images as layers of planar objects [67]. Settling for low fidelity contrasts sharply with a wide array of differentiable renderers focused on accurate light transport, which are slower but can simulate subtle phenomena [4,75]. High-quality results can also be obtained by using learning methods and dense voxel grids [39].

Differentiable Rendering has many recent works. OpenDR [40] demonstrated pose updates for meshes representing humans. Neural Mesh Renderer [28] developed approximate gradients and used a differentiable renderer for a wide array of tasks. SoftRasterizer [38] developed a subgradient function for meshes with greatly improved gradient quality. Modular Primitives [33] demonstrated fast, GPU-based differentiable rendering for meshes with texture mapping. Differentiable Surface Splatting [74] developed a differentiable renderer for point clouds by building upon existing rendering techniques [79]. Conversion of point clouds to volumes is also differentiable [26]. Pulsar [34] uses spheres as the primary primitive and focuses on GPU performance. PyTorch3D [52] implements several of these techniques for mesh and point cloud rendering. Some methods exploit sampling to be generic across object representation [12]. Many methods integrate with neural networks for specific tasks, such as obtaining better descriptors [36] or predicting 3D object shape from a single images [10,63].

The use of an algebraic surface representation, which came to be known as *metaballs* can be attributed to Blinn [6]. These algebraic representations were well studied in the 1980s and 1990s. These include the development of ray-tracing approximations [22,70,71] and building metaball representations of depth images [48]. Non-differentiable rendering metaballs has many methods, involving splatting [2], data structures [20,59] or even a neural network [24].

Metaballs, especially in our treatment of them, are related to the use of Gaussian Mixture Models (GMMs) for surface representation. Our method could be considered a *differentiable renderer for GMMs*. Gaussian Mixture Models as a shape representation has some appeal to roboticists [50,60]. Methods developed to render GMMs include search-based methods [57] and projection for occupancy maps [50]. Projection methods for GMMs have also found application in robot pose estimation [25]. In the vision community, GMMs have been studied as a shape representation [16] and used for pose estimation [14,15]. In the visual learning space, GMMs [23], or their approximations [19] have also been used.

Concurrent work also uses Gaussians for rendering. VoGE [68] uses existing volume rendering techniques [44,46]. Others use a DGT to build screen-space Gaussians for point clouds [1]. In contrast, our contribution is the development of an approximate differentiable renderer that produces fast & robust results.

3   Fuzzy Metaballs

Our proposed object representation, dubbed Fuzzy Metaballs, is an algebraic, implicit surface representation. Implicit surfaces are an object representations where the surface is represented as

$$F(x, y, z) = 0. \tag{1}$$

While some methods parameterize F with neural networks [46], Blinn's algebraic surfaces [6], also known as blobby models or metaballs, are defined by

$$F(x, y, z) = \sum_i \lambda_i P(x, y, z) - T, \tag{2}$$

where $P(x, y, z)$ is some geometric component and T is a threshold for sufficient density. While Blinn used isotropic Gaussians (hence balls), in our case, we use general multidimensional Gaussians that form ellipsoids:

$$P(\vec{x}) = |\Sigma|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\vec{x} - \mu)^T \Sigma^{-1}(\vec{x} - \mu)\right). \tag{3}$$

To obtain Fuzzy Metaballs, we relax the restriction on T being a hard threshold set by the user and instead develop a ray-tracing formulation for Gaussians. To achieve this, we develop two components: a way of defining intersections between Gaussians and rays (Section 4.1), and a way of combining intersections across all Gaussians (Section 4.2). In our definition, all rays always intersect all Gaussians, leading to smooth gradients.

Our implementation is in JAX [8], enabling CPU and GPU acceleration as well as automatic backpropogation. The rendering function that takes camera pose, camera rays and geometry is 60 lines of code. To enable constraint-free backpropogation, we parameterize Σ^{-1} with its Cholesky decomposition: a lower triangular matrix with positive diagonal components. We ensure that the diagonal elements are positive and at least 10^{-6}. The determinant is directly computed from a product of the diagonal of L. When analyzing ray intersections, one can omit the $|\Sigma|^{-\frac{1}{2}}$ term as maximizing requires only the quadratic form. For example, $\vec{x}$ is replaced with a ray intersection of $\vec{v}t$ with $\vec{v} \in \mathbf{R}^3$ and $t \in \mathbf{R}$:

$$s(vt) = (vt - \mu)^T \Sigma^{-1}(vt - \mu), \tag{4}$$

giving a Mahlanobis distnance [42] that is invariant to object scale and allows us to use constant hyperparameters, irrespective of object distance. Using probabilities would be scale-sensitive as equivalent Gaussians that are further are also larger and would have smaller likelihoods at the same points.

To produce an alpha-mask, we simply have two hyperparameters for scale and offset and use a standard sigmoid function:

$$\alpha = \sigma\left(\beta_4\left[\sum_i \lambda_i \exp(-\frac{1}{2}s(vt))\right] + \beta_5\right). \tag{5}$$

4 Approximate Differentiable Rendering

Instead of using existing rendering methods, we develop an approximate renderer that produces smooth, high-quality gradients. While inexact, our formulation enables fast and robust differentiable rendering usable in an analysis by synthesis pipeline [5]. We split the process into two steps: intersecting each component independently in Section 4.1 and combining those results smoothly in Section 4.2.

4.1 Intersecting Gaussians

What does it mean to have a particular intersection of a ray with a Gaussian? We propose three methods. The *linear* method is where the ray intersects the Gaussian at the point of highest probability. Maximizing Eq. (4) is solved by

$$t = \frac{\mu^T \Sigma^{-1} v}{v^T \Sigma^{-1} v}. \tag{6}$$

An alternative view is a volume model, intersecting at the maximum magnitude of the gradient of the Gaussian:

$$||\nabla p(tv)||^2 = P(tv)^2 = (tv - \mu)^T \Sigma^{-1} \Sigma^{-1} (tv - \mu). \tag{7}$$

Obtaining the gradient of Eq. (7) and setting it equal to zero leads to a cubic equation, hence the *cubic* method. Defining $m = \Sigma\mu$ and $r = \Sigma v$ leads to:

$$0 = -t^3 (r^T r)(v^T r)$$
$$+t^2 \left[(m^T r + r^T m)(v^T r) + (r^T r)(v^T m) \right]$$
$$-t \left[(m^T m)(v^T r) + (m^T r + r^T m)(v^T m) - r^T r \right]$$
$$+(m^T m)(v^T m) - r^T m.$$

While standard formulas exist for the cubic, the higher order polynomial all-but-ensures that numerical issues will arise. We implement a numerically stable solver for the cubic [7]. However, even the numerically stable version produces problematic pixels in 32bit floating point. Errors at a rate of about 1 in 1,000 produce NaNs and make backpropagation impossible.

The *quadratic* method approximates the cubic by intersecting the Gaussian at the one standard deviation ellipsoid. Clipping the inside of square roots to be non-negative leads to reasonable results when the ray misses the ellipsoid.

$$t^2 v^T \Sigma^{-1} v - 2t v^T \Sigma^{-1} \mu + \mu^T \Sigma^{-1} \mu = 1$$
$$a = v^T \Sigma^{-1} v \qquad b = -2 v^T \Sigma^{-1} \mu \qquad c = \mu^T \Sigma^{-1} \mu - 1$$

Figures 2 and 3 illustrate all three methods. The linear method produces smooth surfaces and the quadratic surface shows the individual ellipsoids protruding from the surface of the object and the cubic shows artifacts.

In 3D evaluation on objects, for a forward pass, the linear method is the fastest, the quadratic method takes 50% longer and the cubic method takes twice as long as the linear method. The quadratic method has the lowest errors in depth and mask errors. However, due to its stability, in all evaluation outside this section, we use the linear method.
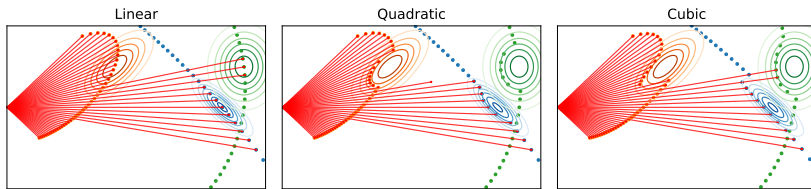
Fig. 2: Two dimensional version of our approximate renderer with camera rays cast from the center left. Three components are shown by their contour maps and their intersections with dots. The blended results are shown with red rays.

Linear	Quadratic	Cubic

errors: mask (28)   errors: mask (28)   errors: mask (31)
           depth (14)   depth (12)   depth (13)
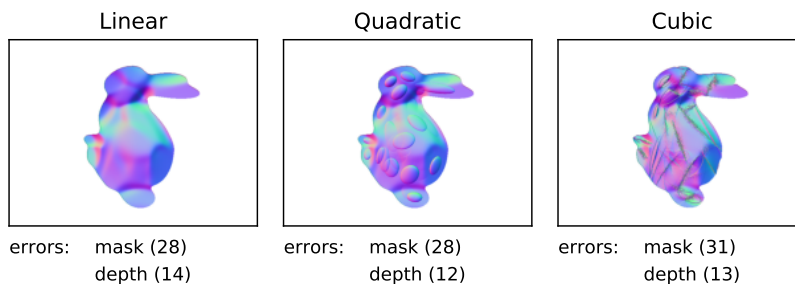
Fig. 3: Visual examples of normal maps from different methods of ray intersection, along with the respective mask and depth errors. See Section 4.1 for details

4.2   Blending intersections

We present a particular solution to the hidden-surface problem [58]. Our method is related to prior work on *Order Independent Transparency (OIT)* [17,45] but extended to 3D objects with opaque surfaces. We combine each pixel's ray-Gaussian intersections with a weighted average

$$t_f = \frac{1}{\sum_i w_i} \sum_i w_i t_i. \tag{8}$$

The weights are an exponential function with two hyperparameters β_1 and β_2 balancing high-quality hits versus hits closer closer to the camera:

$$w_i = \exp\left(\beta_1 s(vt_i)h(t_i) - \frac{\beta_2}{\eta}t_i\right). \tag{9}$$

We include a term (η) for the rough scale of the object. This, along with use of Eq. (4) allows our rendering to be invariant to object scale. We also include an extra term to down-weight results of intersections behind the camera with a simple sigmoid function:

$$h(t) = \sigma\left(\frac{\beta_3}{\eta}t\right). \tag{10}$$

Our blending solution requires only $O(N)$ evaluations of Gaussians for each ray.

Table 1: **Runtimes in milliseconds** with $\mu \pm \sigma$ for rendering images and performing gradient updates in pose estimation with comparable fidelity (Section 6). CPU performance may be a fairer comparison as our method is 60 lines of JAX [8] code and lacks a custom CUDA kernel. CUDA numbers use 160 x 120 images on a Quadro P2000, while CPU use 80 x 60 images on an i5-7287U.

Method	Forward CUDA	Backwards CUDA	Forward CPU	Backwards CPU
Point Cloud [52]	12.1 ± 0.5	23.4 ± 0.5	18.0 ± 1.0	23.8 ± 4.0
Pulsar [34]	7.8 ± 0.3	11.2 ± 0.4	16.4 ± 1.4	63.6 ± 7.9
SoftRas Mesh [38,52]	17.0 ± 0.4	27.2 ± 0.5	21.5 ± 2.0	384.7 ± 93.8
Fuzzy Metaballs	$\mathbf{3.0 \pm 0.2}$	$\mathbf{9.6 \pm 0.5}$	$\mathbf{3.0 \pm 0.15}$	$\mathbf{13.2 \pm 1.4}$

4.3 Obtaining Fuzzy Metaballs

A representation can be limited in utility by how easily one can convert to it. We propose that, unlike classic Metaballs, Fuzzy Metaballs have reasonably straightforward methods for conversion from other formats.

Since we've developed a differentiable renderer, one can optimize a Fuzzy Metaball representation from a set of images. One could use several different losses, but experiments with silhouettes are described in Section 7.2.

If one has a mesh, the mathematical relationship of Fuzzy Metaballs and Gaussian Mixture Models can be exploited by fitting a GMM with Expectation-Maximization [13]. With Fuzzy Metaballs being between a surface and volume representation, there are two forms of GMM one could fit. The first is a surface GMM (sGMM) as used by many authors [16,30,60], where a GMM is fit to points sampled from the surface of the object. The second is to build a volumetric GMM (vGMM). To build a vGMM, one takes a watertight mesh [29], and samples points from the interior of the object. Fitting a GMM to these interior points is what we call a volumetric GMM. Both representations can then further be optimized using the differentiable renderer. Our experiments show that both forms of GMM initialization work well, but we use vGMMs in our experiments.

Extraction is also straightforward. Point clouds can easily be sampled from our proper probability distributions. Extracting a mesh is possible by running marching cubes [41] with an optimized iso-surface level. The details of these experiments can be found in the supplementary material.

5 Data

We use ten models for evaluation: five from the Stanford Model Repository [35] (arma, buddha, dragon, lucy, bunny), three from Thingi10K [78] (gear, eiffel, rebel) and two from prior rendering literature (yoga, plane). All ten are used for reconstruction, and seven are used for pose estimation. We selected objects with different scales, genus, and variety in features. We choose 40 component FMs based on prior literature suggesting 20 to 60 GMMs for object representation [14].
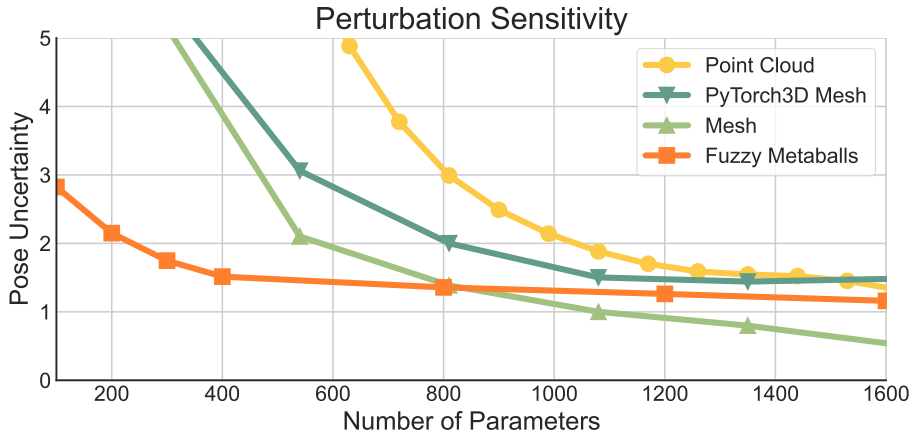
Fig. 4: **Perturbation sensitivity** is the average error in pose when registration is performed with ground truth pose as initialization. See Section 6 for details. The underlying ground truth is a decimated mesh, so only the mesh representation approaches exactly zero error while other asymptote at a higher mark.

6   Comparing Representations

Fairly comparing object representations requires some notion of what to hold constant. As the parameter counts of each representation increase, so do their representational ability. It would be unfair to compare a million point point-cloud against a 100 face triangle mesh. Since our goal is utility in vision tasks, our definition of fidelity will also be task-centric.

In this case, our metric of fidelity will be a representation's *perturbation sensitivity*. We define this as the pose error obtained when optimizing an object's camera pose given a depth map, when the optimization process was initialized with ground truth camera pose. The given depth map is of the full representation object, but the methods are evaluated using lower fidelity versions, leading to perturbations of optimal pose and our fidelity metric. Pose errors are reported using the geometric mean of rotation error and translation error.

Results of our fidelity experiments can be seen in Fig. 4. We evaluate point clouds and meshes using a standard Iterative Closest Point (ICP) method [77], with the point clouds randomly subsampled and the meshes undergoing decimation [18]. We also use PyTorch3D [52], a differential mesh renderer, and obtain its perturbation curve. These experiments are conditional on an experimental setup and methods used, and thus these results may change under different conditions.

In our experiments, a 40 component Fuzzy Metaball (the size we throughout across this paper) produces a pose uncertainty equivalent to a 470 point point cloud (roughly triple the parameter count of a fuzzy metaball) and 85 vertex, 170 triangle mesh (roughly twice the parameter count). These are the sizes use throughout the rest of the paper, in our attempt to keep comparisons fair.

7   Experiments

For comparison against other Differentiable Renderers, we use the methods implemented in PyTorch3D [52], which has a wide variety of techniques with well-optimized routines. The mesh rendering method is an implementation of the *SoftRasterizer* [38]. For point clouds, PyTorch3D cites *Direct Surface Splatting* [74], while also implementing the recent *Pulsar* [34].

With the fidelity of different object representations normalized out (Section 6), we can compare the runtime performance in a fair way, with times shown in in Table 1. On the CPU, where comparisons are more equal (due to lacking a custom CUDA kernel), our renderer is 5 times faster for a forward pass, and significantly faster (30x) for a backwards pass compared to the mesh rendering methods. The point cloud renderer is more comparable in runtime to ours but need a pre-specified point size, often producing images with lots of holes (when points are too small) or a poor silhouette (when points are too big).

To the demonstrate the ability our differentiable renderer to solve classic computer vision tasks, we look at pose estimation (Section 7.1) and 3D reconstruction from silhouettes (Section 7.2). Our renderer is a function that takes camera pose and geometry, and produces images. It seems natural to take images and evaluate how well either camera pose or geometry can be reconstructed, when the other is given. All five hyperparameters for our rendering algorithm ($\beta_{1,2,3,4,5}$) were held constant throughout all experiments.

Since pose estimation and shape from silhouette (SFS) are classic computer vision problems, there are countless methods for both tasks. We do not claim to be the best solution to these problems, as there are many methods specifically designed for these tasks under a variety of conditions. Instead, we seek to demonstrate how our approximate differentiable renderer is comparable in quality to typical solutions, using only gradient descent, without any regularization.

Table 2: **Pose Estimation Results**. Pose Errors are reported with a geometric mean of rotation and translation error. The reported numbers are mean $\pm$ IQR. We report results clean data and data with simulated depth and silhouette noise.

	Parameters	Noise-Free Error	Noisy Error
Initialization		20.2 $\pm$ 18	20.2 $\pm$ 18
Pulsar [34]	1,200	20.2 $\pm$ 18	20.2 $\pm$ 18
Point Cloud [52]	1,200	18.5 $\pm$ 16	18.4 $\pm$ 16
SoftRas Mesh [38]	750	14.9 $\pm$ 15	17.0 $\pm$ 17
Equal Fidelity ICP (Plane) [77]	1,200	10.8 $\pm$ 12	8.2 $\pm$ 3.3
Equal Fidelity ICP (Point) [77]	1,200	7.6 $\pm$ 9.9	8.7 $\pm$ 6.6
High Fidelity ICP (Plane) [77]	120,000	8.2 $\pm$ 0.8	8.0 $\pm$ 3.6
High Fidelity ICP (Point) [77]	120,000	6.2 $\pm$ 3.7	6.8 $\pm$ 3.3
Fuzzy Metaballs	400	**4.0 $\pm$ 1.5**	**4.2 $\pm$ 2.1**

7.1   Pose Estimation

Many differential renderers show qualitative results of pose estimation [38,74]. We instead perform quantitative results over our library of models rendered from random viewpoints. Methods are given a perturbed camera pose ($\pm 45°$ rotation and a random translation up to 50% of model scale) and the ground truth depth image from the original pose. The methods are evaluated by their ability to recover the original pose from minimizing image-based errors. The resulting pose is evaluated for rotation error and translation error. We quantify the score for a model as the geometric mean of the two errors. All methods are tested on the same random viewpoints and with the same random perturbations.

For Fuzzy Metaballs, we establish projective correspondence [54] and optimize silhouette cross-entropy loss averaged over all pixels:

$$CE(\alpha, \hat{\alpha}) = \alpha \log(\hat{\alpha}) + (1 - \alpha) \log(1 - \hat{\alpha}). \tag{11}$$

Estimated alpha is clipped to $[10^{-6}, 1 - 10^{-6}]$ to avoid infinite error. We also evaluate with an additional depth loss of $MSE(z, \hat{z})$ where $|z|$ normalizes the errors to be invariant to object scale and comparable in magnitude to $CE(\alpha, \hat{\alpha})$.

$$MSE(z, \hat{z}) = \left\| \frac{(z - \hat{z})}{|z|} \right\|_2 \tag{12}$$

There is a subtle caveat in the gradients of Fuzzy Metaballs. The gradient of the translation scales by the inverse of model scale.. We correct for this by scaling the gradients by η^2. Alternatively one could scale the input data to always be of some canonical scale [72]. To maintain scale invariance, we limit our use of adaptive learning rate methods to SGD with Momentum.

We provide point cloud ICP results for point-to-point and point-to-plane methods [54] as implemented by Open3D [77]. For the differentiable rendering experiments, we use PyTorch3D [52] and tune its settings (see supplementary). All differentiable rendering methods use the same loss, learning rate decay criteria and are run until the loss stops reliably decreasing.

Pose Estimation Results Overall results are found in Table 2 and a more detailed breakdown in Fig. 5. All methods sometimes struggle to find the correct local minima in this testing setup. Prior differentiable renderers significantly under-performed classic baselines like ICP, while our approximate renderer even outperforms the ICP baselines under realistic settings with synthetic noise.

ICP on noise-free data had bimodal results: it typically either recovered the correct pose to near machine precision or it fell into the wrong local minima. Despite having a higher mean error, ICP's median errors on noise-free data were $\frac{1}{10}$ of Fuzzy Metaballs (FMs). With noisy data, this bimodal distribution disappears and Fuzzy Metaballs outperform on all tested statistical measures. FMs even outperformed ICP with high-fidelity point clouds, suggesting a difference in method not just fidelity. This may be due to our inclusion of a silhouette loss, the benefits of projective correspondence over the nearest neighbors used by this ICP variant [77] or the strengths of visual loss over geometric loss [65].
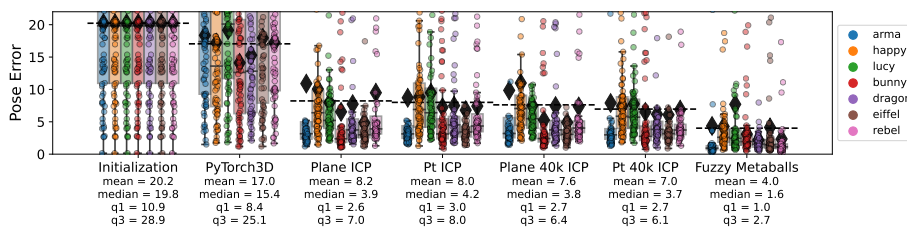
Fig. 5: **Noisy Pose Estimation** Dashed lines are averages for the method, while the black diamonds show the average for that method and model. Here Fuzzy Metaballs win in all statistical measures, typically by a factor of ≈ 2.

7.2   3D Reconstruction

Reconstruction experiments are common in the differential rendering literature [34,74]. However, instead of optimizing with annotations of color [33] or normals [74], we instead only optimize only over silhouettes, as in the classic Shape From Silhouette (SFS) [11]. Unlike many prior examples in the literature, which require fine-tuning of several regularization losses [52,74], we use no regularization in our experiments and can keep constant settings for all objects.

We initialize with a sphere (isosphere for meshes, an isotropic Gaussian of points for point clouds and a small blobby sphere for Fuzzy Metaballs). Given a set of silhouette images and their camera poses, we then optimize silhouette loss for the object. In our experiments, we use 64 x 64 pixel images and have 32 views. For these experiments, we resize all models to a canonical scale and use the Adam [32] optimizer. For baseline hyperparameters, we use the PyTorch3D settings with minimal modification. For SoftRas, we use a twice subdivided icosphere. For NeRF [46], we use a two layer MLP with 30 harmonic function embedding with 128 hidden dimension and the same early exit strategy as FMs.

Inspired by artifacts seen in real videos (Fig. 9), we produce a noisy silhouette dataset where training data had $\frac{1}{8}$ of each silhouette under-segmented (Fig. 8) in 16 of 32 images by clustering silhouette coordinates [56] and removing a cluster.
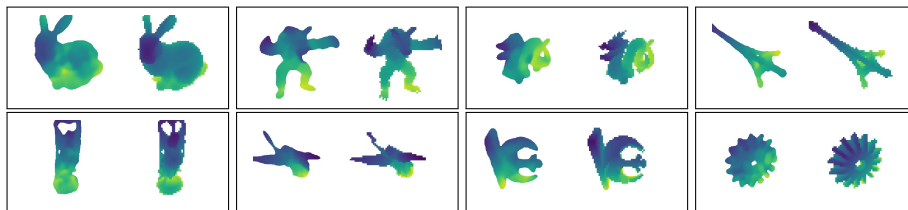

Fig. 6: **Shape from Silhouette (SFS)** reconstructions. On the left is a 40 component Fuzzy Metaball result and the right is the mesh ground-truth of about 2,500 faces, both colored by depth maps.

Table 3: **Shape from Silhouette reconstruction fidelity** as measured by cross-entropy silhouette loss on 32 novel viewpoints for each of 10 sample models. Runtimes were the average per model and performed on CPU. Results show $\mu\pm\sigma$.

	Time (s)	Noise-Free Recon. Error	Noisy Recon. Error
Voxel Carving [43,77]	<u>82</u>	0.31 ± 0.100	1.119 ± 0.367
PyTorch3D Point [52]	185	0.075 ± 0.066	0.100 ± 0.079
PyTorch3D Mesh [38]	3008	0.062 ± 0.049	0.072 ± 0.051
NeRF [46]	7406	$\mathbf{0.032 \pm 0.022}$	<u>0.062 ± 0.063</u>
Fuzzy Metaballs	**68**	<u>0.040 ± 0.015</u>	$\mathbf{0.055 \pm 0.016}$

Shape From Silhouette Results We show qualitative reconstructions from Fuzzy Metaballs (Fig. 6), along with quantitative results against baselines (Table 3) and some example reconstructions from all methods (Fig. 8).

Overall, we found that our method was significantly faster than the other differentiable renderers, while producing the best results in the case of noisy reconstructions. Classic voxel carving [43] with a 384^3 volume was reasonably fast, but the 32 views of low resolution images didn't produce extremely sharp contours (see supplementary). With under-segmentation noise, voxel carving fails completely while the differentiable renderers reasonably reconstruct all models.

Among the differentiable renders, we can see how the mesh-based approach struggles to change genus from a sphere to the Eiffel tower. The point cloud renderer lacks the correct gradients to successfully pull spurious points into the model. NeRF [46] performs reasonably well in shape from silhouette, even with spurious masks. In fact, it was the best performer for noise-free data, and in a majority of the reconstructions in noisy data (its mean performance was hurt by results on `eiffel` and `lucy` with long thin surfaces). NeRF is a sophisticated model with many settings, and it may have a configuration where it successfully reconstructs all the models, but due to its dense volumetric rendering and use of an MLP, it is 100x slower than our low degree of freedom representation.
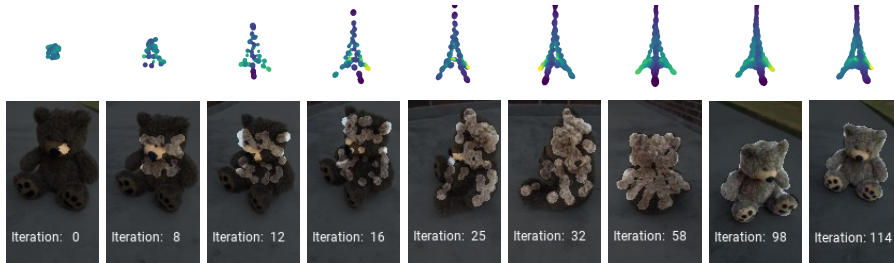

Fig. 7: **Shape from Silhouette steps** Top row shows synthetic data with reconstructed depth. Bottom row shows reconstructed masks for a CO3D video [53].
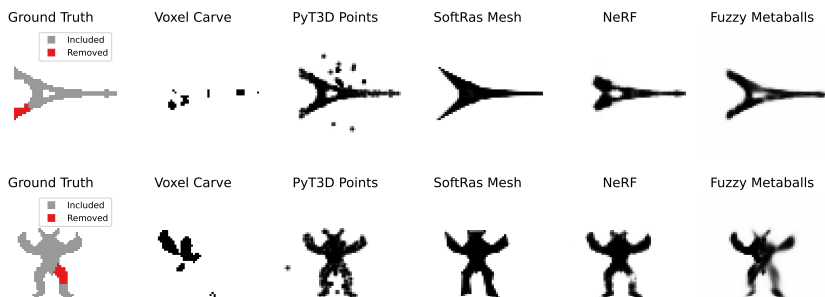
Fig. 8: **Shape from Silhouette Results** with simulated under-segmentation.

8   Discussion

The focus of our approximate differentiable rendering method has been on shape. While it is possible to add per-component colors to Fuzzy Metaballs (see supplementary), that has not been the focus of our experiments. Focusing on shape allows us to circumvent modeling high-frequency color textures, as well as ignoring lighting computations. This shape-based approach can use data from modern segmentation methods [21] and depth sensors [31]. Low-degree of freedom models have a natural robustness and implicit regularization that allows for recovery from significant artifacts present in real systems. For example, Fig. 9 shows robust recovery from real over/under-segmentation artifacts in video sequences.

Our approximate approach to rendering by using OIT-like methods creates a trade-off. The downside is that small artifacts can be observed since the method coarsely approximates correct image formation. The benefits are good gradients, speed & robustness, all of which produce utility in vision tasks.

Compared to prior work [34,38], our results do not focus on the same areas of differentiable rendering. Unlike other work, we do not perform GPU-centric optimizations [33]. Additionally, prior work focuses on producing high-fidelity color images (and using them for optimization). Unlike prior work, we benchmark our method across a family of objects and report quantitative results against classic baselines. Unlike some popular implicit surface methods such as the NeRF [46] family, our object representation is low degree of freedom, quick to optimize from scratch, and all the parameters are interpretable with geometric meaning.

While our experiments focus on classic computer vision tasks such as pose estimation or shape from silhouette, the value of efficiently rendering interpretable, low degree of freedom models may have the biggest impact outside of classic computer vision contexts. For example, in scientific imaging it is often impossible to obtain high-quality observations since the sensors are limited. For example, in non-light-of-sight imaging [66], sonar reconstruction [69], lightcurve inversion [27] and CryoEM [9,76]. In all these contexts, getting good imaging information is extremely hard and low degree of freedom models could be desirable.

FM SFS Depth   FM SFS Mask   Mask-RCNN Mask

(a) **Depth and silhouette** from a shape-from-silhouette reconstruction.

FM SFS Mask 10%   FM SFS Mask 50%   Mask-RCNN Mask

(b) **Recovering from undersegmentation in the ground truth masks.** While a 50% threshold does a good job recovering the head, better recovery can be shown with a 10% threshold, also recovering the leg.

FM SFS Mask 10%   FM SFS Mask 50%   Mask-RCNN Mask

(c) **Recovering from oversegmentation in ground truth masks.** Even the $\alpha = 10\%$ threshold only leads to minor over-segmentation in the mask, suggesting a setting that be appropriate in general.

Fig. 9: **Shape from silhouette reconstruction on natural images** from a handheld cell phone video, using COLMAP [55] and Mask RCNN [21] for automatic camera poses and silhouettes. The low degree of freedom leads to natural regularization and recovery from errors in ground truth.

9   Conclusion

Approximate differentiable rendering with algebraic surfaces enables fast analysis-by-synthesis pipelines for vision tasks which focus on shapes, such as pose estimation and shape from silhouette. For both tasks, we show results with realistic, simulated noise. The robustness of our approach enables it to runs naturally on silhouettes extracted from real video sequences without any regularization. Whereas classic methods can struggle once noise is introduced, differentiable renderers naturally recovery by using stochastic optimization techniques. By using gradient-based optimization, differentiable rendering techniques provide a robust solution to classic vision problems. Fuzzy Metaballs can enable low-latency differential rendering on CPUs. Our formulation connects algebraic surfaces [6] used in graphics with Gaussian Mixture Models [16] used in vision. These provide a compact, interpretable representation for shape with many uses.

References

1. Unbiased gradient estimation for differentiable surface splatting via poisson sampling. In: European Conference on Computer Vision (ECCV) (2022)
2. Adams, B., Lenaert, T., Dutré, Ph.: Particle splatting: Interactive rendering of particle-based simulation data. Report CW 453, KU Leuven (July 2006), http://www.cs.kuleuven.be/publicaties/rapporten/cw/CW453.abs.html
3. Agin, G.J.: Representation and Description of Curved Objects. Ph.D. thesis, Stanford University, CA, USA (1972)
4. Bangaru, S., Li, T.M., Durand, F.: Unbiased warped-area sampling for differentiable rendering. ACM Trans. Graph. **39**(6), 245:1–245:18 (2020)
5. Bell, C.G., Fujisaki, H., Heinz, J.M., Stevens, K.N., House, A.S.: Reduction of speech spectra by analysis-by-synthesis techniques. The Journal of the Acoustical Society of America **33**(12), 1725–1736 (1961). https://doi.org/10.1121/1.1908556
6. Blinn, J.F.: A generalization of algebraic surface drawing. ACM Trans. Graph. **1**(3), 235–256 (Jul 1982). https://doi.org/10.1145/357306.357310
7. Blinn, J.F.: How to solve a cubic equation, part 5: Back to numerics. IEEE Computer Graphics and Applications **27**(3), 78–89 (2007). https://doi.org/10.1109/MCG.2007.60
8. Bradbury, J., Frostig, R., Hawkins, P., Johnson, M.J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., Zhang, Q.: JAX: composable transformations of Python+NumPy programs (2018), http://github.com/google/jax
9. Brubaker, M., Punjani, A., Fleet, D.: Building proteins in a day. CVPR (June 2015). https://doi.org/10.1109/cvpr.2015.7298929
10. Chen, W., Ling, H., Gao, J., Smith, E., Lehtinen, J., Jacobson, A., Fidler, S.: Learning to predict 3d objects with an interpolation-based differentiable renderer. Advances in Neural Information Processing Systems **32** (2019)
11. Cheung, K.m.G., Baker, S., Kanade, T.: Shape-from-silhouette across time part i: Theory and algorithms. International Journal of Computer Vision **62**(3), 221–247 (2005)
12. Cole, F., Genova, K., Sud, A., Vlasic, D., Zhang, Z.: Differentiable surface rendering via non-differentiable sampling (2021)
13. Dempster, A., Laird, N., Rubin, D.B.: Maximum likelihood from incomplete data via the EM algorithm. J. Royal Stat. Soc. (B) **39**(1), 1–38 (1977). https://doi.org/http://dx.doi.org/10.2307/2984875
14. Eckart, B., Kim, K., Kautz, J.: Hgmr: Hierarchical gaussian mixtures for adaptive 3d registration. In: ECCV 2018. pp. 730–746 (2018)
15. Eckart, B., Kim, K., Troccoli, A., Kelly, A., Kautz, J.: MLMD: Maximum Likelihood Mixture Decoupling for Fast and Accurate Point Cloud Registration. In: 3DV. pp. 241–249 (2015). https://doi.org/10.1109/3DV.2015.34
16. Eckart, B., Kim, K., Troccoli, A., Kelly, A., Kautz, J.: Accelerated Generative Models for 3D Point Cloud Data. In: CVPR. pp. 5497–5505 (2016). https://doi.org/10.1109/CVPR.2016.593
17. Enderton, E., Sintorn, E., Shirley, P., Luebke, D.: Stochastic transparency. In: I3D '10: Proceedings of the 2010 symposium on Interactive 3D graphics and games. pp. 157–164. New York, NY, USA (2010)
18. Garland, M., Heckbert, P.S.: Surface simplification using quadric error metrics. In: SIGGRAPH. pp. 209–216 (1997). https://doi.org/10.1145/258734.258849

19. Genova, K., Cole, F., Sud, A., Sarna, A., Funkhouser, T.: Local deep implicit functions for 3d shape (2020)
20. Gourmel, O., Pajot, A., Paulin, M., Barthe, L., Poulin, P.: Fitted BVH for Fast Raytracing of Metaballs. Computer Graphics Forum **3**, 7 – 288 (2010). https://doi.org/10.1111/j.1467-8659.2009.01597.x, https://hal.archives-ouvertes.fr/hal-01516266
21. He, K., Gkioxari, G., Dollár, P., Girshick, R.B.: Mask r-cnn. 2017 IEEE International Conference on Computer Vision (ICCV) pp. 2980–2988 (2017)
22. Heckbert, P.S.: Fun with gaussians. SIGGRAPH '86 Advanced Image Processing seminar notes (1986)
23. Hertz, A., Hanocka, R., Giryes, R., Cohen-Or, D.: Pointgmm: a neural gmm network for point clouds. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (2020)
24. Horvath, R.: Image-Space Metaballs Using Deep Learning. Master's thesis, Faculty of Informatics, TU Wien (Jul 2019), https://www.cg.tuwien.ac.at/research/publications/2019/horvath-2018-ism/
25. Huang, H., Ye, H., Sun, Y., Liu, M.: Gmmloc: Structure consistent visual localization with gaussian mixture models. IEEE Robotics and Automation Letters **5**(4), 5043–5050 (2020). https://doi.org/10.1109/LRA.2020.3005130
26. Insafutdinov, E., Dosovitskiy, A.: Unsupervised learning of shape and pose with differentiable point clouds (2018)
27. Kaasalainen, M., Torppa, J.: Optimization methods for asteroid lightcurve inversion. Icarus **153**(1), 24–36 (2001)
28. Kato, H., Ushiku, Y., Harada, T.: Neural 3d mesh renderer (2017)
29. Kazhdan, M., Bolitho, M., Hoppe, H.: Poisson Surface Reconstruction. In: Sheffer, A., Polthier, K. (eds.) Symposium on Geometry Processing. The Eurographics Association (2006). https://doi.org/10.2312/SGP/SGP06/061-070
30. Keselman, L., Hebert, M.: Direct fitting of gaussian mixture models. In: 2019 16th Conference on Computer and Robot Vision (CRV). pp. 25–32 (2019). https://doi.org/10.1109/CRV.2019.00012
31. Keselman, L., Woodfill, J.I., Grunnet-Jepsen, A., Bhowmik, A.: Intel realsense stereoscopic depth cameras. CoRR **abs/1705.05548** (2017), http://arxiv.org/abs/1705.05548
32. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: ICLR (Poster) (2015), http://arxiv.org/abs/1412.6980
33. Laine, S., Hellsten, J., Karras, T., Seol, Y., Lehtinen, J., Aila, T.: Modular primitives for high-performance differentiable rendering. ACM Transactions on Graphics **39**(6) (2020)
34. Lassner, C., Zollhöfer, M.: Pulsar: Efficient sphere-based neural rendering. arXiv:2004.07484 (2020)
35. Levoy, M., Gerth, J., Curless, B., Pull, K.: The Stanford 3D scanning repository **5**(10) (2005), http://graphics.stanford.edu/data/3Dscanrep/
36. Li, L., Zhu, S., Fu, H., Tan, P., Tai, C.L.: End-to-end learning local multi-view descriptors for 3d point clouds (2020)
37. Lin, C.H., Kong, C., Lucey, S.: Learning efficient point cloud generation for dense 3d object reconstruction. In: AAAI Conf. on Artificial Intelligence (AAAI) (2018)
38. Liu, S., Li, T., Chen, W., Li, H.: Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (October 2019)

39. Lombardi, S., Simon, T., Saragih, J., Schwartz, G., Lehrmann, A., Sheikh, Y.: Neural volumes: Learning dynamic renderable volumes from images. ACM Trans. Graph. **38**(4), 65:1–65:14 (Jul 2019)
40. Loper, M.M., Black, M.J.: Opendr: An approximate differentiable renderer. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) Computer Vision – ECCV 2014. pp. 154–169. Springer International Publishing, Cham (2014)
41. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm. ACM siggraph computer graphics **21**(4), 163–169 (1987)
42. Mahalanobis, P.C.: On the generalized distance in statistics. Proceedings of the National Institute of Sciences (Calcutta) pp. 49–55 (1936)
43. Martin, W.N., Aggarwal, J.K.: Volumetric descriptions of objects from multiple views. IEEE Transactions on Pattern Analysis and Machine Intelligence **PAMI-5**(2), 150–158 (1983). https://doi.org/10.1109/TPAMI.1983.4767367
44. Max, N.: Optical models for direct volume rendering. IEEE Transactions on Visualization and Computer Graphics **1**(2), 99–108 (1995). https://doi.org/10.1109/2945.468400
45. McGuire, M., Bavoil, L.: Weighted blended order-independent transparency. Journal of Computer Graphics Techniques (JCGT) **2**(2), 122–141 (December 2013), http://jcgt.org/published/0002/02/09/
46. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.M. (eds.) Computer Vision – ECCV 2020. pp. 405–421. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-58452-8_24
47. Miller, I.D., Cladera, F., Cowley, A., Shivakumar, S.S., Lee, E.S., Jarin-Lipschitz, L., Bhat, A., Rodrigues, N., Zhou, A., Cohen, A., Kulkarni, A., Laney, J., Taylor, C.J., Kumar, V.: Mine tunnel exploration using multiple quadrupedal robots (2020)
48. Muraki, S.: Volumetric shape description of range data using "blobby model". In: Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques. p. 227–235. SIGGRAPH '91, Association for Computing Machinery, New York, NY, USA (1991). https://doi.org/10.1145/122718.122743
49. Nimier-David, M., Vicini, D., Zeltner, T., Jakob, W.: Mitsuba 2: A retargetable forward and inverse renderer. ACM Trans. Graph. **38**(6) (nov 2019). https://doi.org/10.1145/3355089.3356498
50. O'Meadhra, C., Tabib, W., Michael, N.: Variable resolution occupancy mapping using gaussian mixture models. IEEE Robotics and Automation Letters **4**(2), 2015–2022 (2019). https://doi.org/10.1109/LRA.2018.2889348
51. Pfister, H., Zwicker, M., van Baar, J., Gross, M.: Surfels: Surface elements as rendering primitives. In: Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques. p. 335–342. SIGGRAPH '00, ACM Press/Addison-Wesley Publishing Co., USA (2000). https://doi.org/10.1145/344779.344936
52. Ravi, N., Reizenstein, J., Novotny, D., Gordon, T., Lo, W.Y., Johnson, J., Gkioxari, G.: Accelerating 3d deep learning with pytorch3d. arXiv:2007.08501 (2020)
53. Reizenstein, J., Shapovalov, R., Henzler, P., Sbordone, L., Labatut, P., Novotny, D.: Common objects in 3d: Large-scale learning and evaluation of real-life 3d category reconstruction. In: International Conference on Computer Vision (2021)
54. Rusinkiewicz, S., Levoy, M.: Efficient variants of the icp algorithm. In: Proceedings Third International Conference on 3D Digital Imaging & Modeling. pp. 145–152 (2001). https://doi.org/10.1109/IM.2001.924423

55. Schönberger, J.L., Zheng, E., Pollefeys, M., Frahm, J.M.: Pixelwise view selection for unstructured multi-view stereo. In: European Conference on Computer Vision (ECCV) (2016)
56. Sculley, D.: Web-scale k-means clustering. In: Proceedings of the 19th International Conference on World Wide Web. p. 1177–1178. WWW '10, Association for Computing Machinery, New York, NY, USA (2010). https://doi.org/10.1145/1772690.1772862
57. Shankar, K.S., Michael, N.: Mrfmap: Online probabilistic 3d mapping using forward ray sensor models. In: Robotics: Science and Systems (2020)
58. Sutherland, I.E., Sproull, R.F., Schumacker, R.A.: A characterization of ten hidden-surface algorithms. ACM Comput. Surv. **6**(1), 1–55 (3 1974). https://doi.org/10.1145/356625.356626
59. Szécsi, L., Illés, D.: Real-time metaball ray casting with fragment lists. In: Eurographics (2012)
60. Tabib, W., O'Meadhra, C., Michael, N.: On-manifold gmm registration. IEEE Robotics and Automation Letters **3**(4), 3805–3812 (Oct 2018). https://doi.org/10.1109/LRA.2018.2856279
61. Teed, Z., Deng, J.: Raft: Recurrent all-pairs field transforms for optical flow. In: 16th European Conference on Computer Vision. pp. 402–419. Germany (2020). https://doi.org/10.1007/978-3-030-58536-5_24
62. Teed, Z., Deng, J.: Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras (2021)
63. Tewari, A., Zollhöfer, M., Kim, H., Garrido, P., Bernard, F., Pérez, P., Theobalt, C.: Mofa: Model-based deep convolutional face autoencoder for unsupervised monocular reconstruction. In: 2017 IEEE International Conference on Computer Vision (ICCV). pp. 3735–3744 (2017). https://doi.org/10.1109/ICCV.2017.401
64. Tomic, T., Schmid, K., Lutz, P., Domel, A., Kassecker, M., Mair, E., Grixa, I.L., Ruess, F., Suppa, M., Burschka, D.: Toward a fully autonomous uav: Research platform for indoor and outdoor urban search and rescue. IEEE Robotics Automation Magazine **19**(3), 46–56 (2012). https://doi.org/10.1109/MRA.2012.2206473
65. Triggs, B., McLauchlan, P.F., Hartley, R.I., Fitzgibbon, A.W.: Bundle adjustment — a modern synthesis. In: Triggs, B., Zisserman, A., Szeliski, R. (eds.) Vision Algorithms: Theory and Practice. pp. 298–372. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)
66. Tsai, C., Sankaranarayanan, A., Gkioulekas, I.: Beyond volumetric albedo. In: CVPR (June 2019)
67. Tucker, R., Snavely, N.: Single-view view synthesis with multiplane images. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 551–560 (2020)
68. Wang, A., Wang, P., Sun, J., Kortylewski, A., Yuille, A.: Voge: A differentiable volume renderer using gaussian ellipsoids for analysis-by-synthesis. arXiv preprint arXiv:2205.15401 (2022)
69. Westman, E., Gkioulekas, I., Kaess, M.: Volumetric albedo framework for 3d imaging sonar. In: ICRA (2020)
70. Wyvill, G., McPheeters, C., Wyvill, B.: Data structure forsoft objects. The Visual Computer **2**(4), 227–234 (Aug 1986). https://doi.org/10.1007/BF01900346
71. Wyvill, G., Trotman, A.: Ray-tracing soft objects. In: Chua, T.S., Kunii, T.L. (eds.) CG International. pp. 469–476. Springer Japan, Tokyo (1990)
72. Yang, J., Li, H., Jia, Y.: Go-icp: Solving 3d registration efficiently and globally optimally. In: 2013 IEEE International Conference on Computer Vision. pp. 1457–1464 (2013). https://doi.org/10.1109/ICCV.2013.184

73. Yang, S., Scherer, S.: Cubeslam: Monocular 3-d object slam. IEEE Transactions on Robotics **35**(4), 925–938 (2019). https://doi.org/10.1109/TRO.2019.2909168

74. Yifan, W., Serena, F., Wu, S., Öztireli, C., Sorkine-Hornung, O.: Differentiable surface splatting for point-based geometry processing. ACM Transactions on Graphics **38**(6), 1–14 (Nov 2019). https://doi.org/10.1145/3355089.3356513

75. Zhang, C., Miller, B., Yan, K., Gkioulekas, I., Zhao, S.: Path-space differentiable rendering. ACM Trans. Graph. **39**(4), 143:1–143:19 (2020). https://doi.org/10.1145/3386569.3392383

76. Zhong, E.D., Lerer, A., Davis, J.H., Berger, B.: Cryodrgn2: Ab initio neural reconstruction of 3d protein structures from real cryo-em images. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV). pp. 4066–4075 (October 2021)

77. Zhou, Q.Y., Park, J., Koltun, V.: Open3D: A modern library for 3D data processing. arXiv:1801.09847 (2018)

78. Zhou, Q., Jacobson, A.: Thingi10k: A dataset of 10, 000 3d-printing models. CoRR **abs/1605.04797** (2016), http://arxiv.org/abs/1605.04797

79. Zwicker, M., Pfister, H., van Baar, J., Gross, M.: Surface splatting. In: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques. p. 371–378. SIGGRAPH '01 (2001). https://doi.org/10.1145/383259.383300