Sparse 3D Reconstruction via Object-Centric Ray Sampling

Llukman Cerkezi Paolo Favaro Computer Vision Group, Institute of Computer Science, University of Bern, Switzerland

{llukman.cerkezi, paolo.favaro}@unibe.ch



Figure 1. Left: Sparse view setting of a 360° camera rig with 8 views. Right: 3D reconstructions with existing SotA methods. Due to the sparsity and wide spacing of the camera views, methods such as NeRS [65] and RegNeRF [40] reconstruct surfaces with visible artifacts. COLMAP* [46, 47] returned a valid mesh only with 50 views (so is used only as a reference). Methods such as DS [16] obtain better reconstructions, but with fewer details than with our approach. Most methods make use of masks to segment the object in each view. In contrast, our method can work without this additional supervision and still obtain accurate 3D reconstructions (compare to the GT).

Abstract

We propose a novel method for 3D object reconstruction from a sparse set of views captured from a 360-degree calibrated camera rig. We represent the object surface through a hybrid model that uses both an MLP-based neural representation and a triangle mesh. A key contribution in our work is a novel object-centric sampling scheme of the neural representation, where rays are shared among all views. This efficiently concentrates and reduces the number of samples used to update the neural model at each iteration. This sampling scheme relies on the mesh representation to ensure also that samples are well-distributed along its normals. The rendering is then performed efficiently by a differentiable renderer. We demonstrate that this sampling scheme results in a more effective training of the neural representation, does not require the additional supervision of segmentation masks, yields state of the art 3D reconstructions, and works with sparse views on the Google's Scanned Objects, Tank and Temples and MVMC Car datasets. Code available at: https://github. com/llukmancerkezi/ROSTER

1. Introduction

The task of reconstructing the 3D surface of an object from multiple calibrated views is a well-established problem with a long history of methods exploring a wide range of 3D representations and optimization methods [11–13, 53]. Recent approaches have focused their attention on deep learning models [6, 21, 27, 34, 35, 39, 50, 65]. In particular, methods based on neural rendering such as NeRF and its variants [3, 5, 35, 42, 61], have not only shown impressive view interpolation capabilities, but also the ability to output 3D reconstructions as a byproduct of their training.

NeRF's neural rendering drastically simplifies the generation of images given a new camera pose. It altogether avoids the complex modeling of the light interaction with surfaces in the scene. A neural renderer learns to output the color of a pixel as a weighted average of 3D point samples from the NeRF model. Current methods choose these samples along the ray defined by the given pixel and the camera center (see Figure 2 left). Because each camera view defines a separate pencil of rays, the 3D samples rarely overlap. Thus, each view will provide updates for mostly independent sets of parameters of the NeRF model, which can lead to data overfitting. In practice, overfitting means that views used for training will be rendered correctly, but new camera views will give unrealistic images. Such overfitting is particularly prominent when training a NeRF on only a sparse set of views with a broad object coverage (e.g., see the 360° camera rig in Figure 1).

In this work, we address overfitting when working



Figure 2. **Sampling schemes.** Left: NeRF view-centric sampling scheme. Right: Our object-centric sampling scheme. The view-centric sampling scheme uses separate sets of 3D samples for each camera view. This leads to overfitting when views are sparse. Object-centric sampling instead shares the same 3D samples across multiple views.

with sparse views by proposing an object-centric sampling scheme that is shared across all views (see Figure 2 right and Figure 3). We design the scheme so that all (visible) views can provide an update for the same 3D points on a given sampling ray. To do so, we introduce a hybrid 3D representation, where we simultaneously update a Multi Layer Perceptron (MLP) based implicit surface model (similarly to a NeRF) and an associated triangle mesh. The MLP model defines an implicit 3D representation of the scene, while the mesh is used to define the sampling rays. These rays are located at each mesh vertex and take the direction of the normal to the mesh. Then, the mesh vertex of the current object surface is updated by querying the MLP model at 3D samples on the corresponding ray. We use a similar deep learning model to associate color to the mesh. We then image the triangle mesh in each camera view via a differentiable renderer. Because of our representation, the queried 3D samples can be shared across multiple views and thus avoid the overfitting shown by NeRF models in our settings.

Notice that a common practice to handle overfitting in NeRF models trained on sparse views is to constrain the 3D reconstruction through object masks. Masks provide a very strong 3D cue. In fact, a (coarse) reconstruction of an object can even be obtained from the masks alone, a technique known as shape from silhouette [24]. We show that our method yields accurate 3D reconstructions even without mask constraints. This confirms experimentally the effectiveness of our sampling scheme in avoiding overfitting.

To summarize, our contributions are

- A novel object-centric sampling scheme that efficiently shares samples across multiple views and avoids overfitting; the robustness of our method is such that it does not need additional constraints, such as 2D object masks;
- A 3D reconstruction method that yields state of the art results with a sparse set of views from a 360-degree camera rig (on the Google's Scanned Objects [10], Tank and



Figure 3. View vs object-centric sampling (see Figure 2). Computational efficiency: The view-centric approach uses $8 \times K$ samples per mesh vertex, with K camera views. In contrast, the object-centric approach uses only 8 samples per vertex regardless of the number of camera views. Object-centric sampling is not only more efficient but also avoids overfitting. For more details, please check Section 3.

Temples [23] and MVMC Car datasets [65]).

2. Prior Work

Mesh-based methods. With the development of differentiable renderers [6, 18, 20, 52], object reconstruction is now possible through gradient descent (or backpropagation in the context of deep learning). A common approach to predict the shape of an object using differentiable rendering is to use category level image collections [15, 19, 36, 49, 55, 63]. Recently, some methods aim to estimate the shape of an object in a classic multi-view stereo setting and without any prior knowledge of the object category [16, 37, 59, 60, 65]. Several methods also propose different ways to update the surface of the reconstructed object. The methods proposed by Goel et al. [16] and Worchel et al. [59] update the mesh surface by predicting vertex offsets to the template mesh. Zhang et al. [65] use a neural displacement field over a canonical sphere, but restrict the geometry to model only genus-zero topologies. Xu et al. [60], after getting a smooth initial shape via [62], proposes surface-based local MLPs to encode the vertex displacement field for the reconstruction of surface details. Munkberg et al. [37] use a hybrid representation as we do in our method. They learn the signed distance field (SDF) of the reconstructed object. The SDF is defined on samples on a fixed tetrahedral grid and then converted to a surface mesh via deep marching tetrahedra [48]. In contrast, we adapt the samples to the surface of the object as we reconstruct it.

Implicit representations for volume rendering. Recently, Neural Radiance Field (NeRF)-based methods have shown great performance in novel view synthesis tasks [2, 26, 32, 35, 45, 51, 56, 66]. However, these methods require a dense number of training views and camera poses to render realistic views. Methods that tackle novel view rendering from a small set of training views usually exploit two directions. The methods in the first group pre-train their models on large scale calibrated multiview datasets of diverse scenes [5, 7, 25, 28, 43, 58, 64]. In our approach, however, we consider training only on a small set of images.

The methods in the second group add an additional regularization term to their optimization cost to handle the limited number of available views. Diet-Nerf [17] incorporates an additional loss that encourages the similarity between the pre-trained CLIP features between the training images and rendered novel views. RegNerf [40] incorporates two additional loss terms: 1) color regularization to avoid color shifts between different views and 2) geometry regularization to enforce smoothness between predicted depths of neighboring pixels. InfoNerf [22] adds a ray entropy loss to minimize the entropy of the volume density along each ray (thus, they encourage the concentration of the volume density on a single location along the ray). This is not suitable for sparse 360-degree camera rigs, where the camera positions lie at the same elevation angle (as in our case) as a ray can be shared by two opposite camera centers. They also add a loss that minimizes the KL-divergence between the normalized density profiles of two neighboring rays. DS-Nerf [9] instead improves the reconstruction quality by adding depth supervision. As they report, its performance is only as good as the estimates of depth obtained by COLMAP [46, 47]. Common to all of the above methods is that they require some sort of additional training (except InforNerf [22]), while our method reconstructs objects without any additional pre-training.

Implicit representations for surface rendering. [54] provides an overview of methods that use implicit representations for either volume or surface rendering. This family of approaches uses a neural SDF or an occupancy field as an implicit surface representation. DVR [39] and IDR [61] are pioneering SDF methods that use only images for training. They both provide a differentiable rendering formulation using implicit gradients. However, both methods require accurate object masks as well as appropriate weight initialization due to the difficulty of propagating gradients. IRON [67] proposes a method to estimate edge derivatives to ease the optimization of neural SDFs. Some of the methods combine implicit surface models with volumetric ones [41, 57, 62] and also implicit surface models with explicit ones [8, 33, 44]. One advantage of the methods that combine implicit surface models with volumetric ones is that they do not require mask supervision and are more stable. However, they heavily depend on a large number of training images. SparseNeuS [29] can work in the sparse view setting, but requires pre-training on a multi-view dataset of multiple scenes. Additionally, it is pretrained only for the narrow view setup, as opposed to the 360-degree one.

3. Sparse 3D Reconstruction

3.1. Problem Formulation

Our goal is to reconstruct the 3D surface of the object depicted in N images $I = \{I_1, I_2, \dots, I_N\}$ given their corresponding calibrated camera views $\Pi = \{\pi_1, \pi_2, \dots, \pi_N\}$, where π_i denotes the 3D camera pose and intrinsic camera calibration parameters. We consider the *sparse setting*, *i.e.*, when N is small (*e.g.*, 8–15 views). We mostly use camera views distributed uniformly in a 360° rig (see Figure 1), but our method can also work for the narrow view setup (see the supplementary material for experiments with this setting). We pose the 3D reconstruction task as the problem of finding the 3D surface and texture such that the images $I^r = \{I_1^r, \dots, I_N^r\}$ rendered with the given camera views II, best match the corresponding set of captured images I.

3.2. 3D Representation

We describe the surface of the 3D object via a hybrid model that maintains both an implicit (density-based) and explicit (mesh-based) representation. The two representations serve different purposes. The explicit one is used to efficiently render views of the object and is directly obtained from the implicit representation. The advantage of the implicit representation is that it can smoothly transition through a general family of 3D shapes (e.g., from a sphere to a torus). This is especially useful when the 3D reconstruction is achieved through iterative gradient-based optimization algorithms. Such transition is typically much more difficult to achieve with a lone explicit representation. More specifically, the implicit representation is based on the Implicit Surface Neural Network (ISNN) $F_{\text{shape}} : \mathbb{R}^3 \to \mathbb{R}$, that outputs the object density value $\sigma \doteq F_{\text{shape}}(X)$ at a 3D point $X \in \mathbb{R}^3$. The explicit representation is obtained by converting the implicit representation in F_{shape} to a triangle mesh $\mathcal{G} = (\mathcal{V}, \mathcal{F})$ consisting of M vertex locations \mathcal{V} and a face list \mathcal{F} . A triangle in \mathcal{F} is the triplet of indices of the vertices in \mathcal{V} that form that triangle. The conversion of F_{shape} to the explicit representation mesh is based on the selection of a finite set of 3D points, which we call samples and discuss in detail in the next section.

3.3. Object-Centric Sampling

In Figure 4, we show a 2D slice of the implicit representation of the COW 3D shape. The implicit representation will have a density σ that is close to 1 at the surface of the object and 0 elsewhere. In an iterative procedure, we can assume that we already have some existing mesh that is sufficiently close to the current surface of the implicit representation (recall that the implicit representation will be updated through the optimization procedure). To also update the mesh, we use its existing mesh vertices and normals to define segments that are approximately normal to the updated implicit surface and to select samples on the segments in equal number on either side of the surface.

More formally, for each vertex $V_i \in \mathcal{V}, i = 1, \dots, M$, in the current *out-of-date* mesh, we define a sampling ray R_i , such that $R_i \propto N_i$, where N_i is the surface normal at the vertex V_i . Along the ray R_i we draw K 3D samples $X_{i,1}, \ldots, X_{i,K}$ (in Figure 4 we show K = 4). We define outward and inward point samples by drawing K equally spaced 3D points from the segments $[V_i, V_i + t_i^{out}N_i]$ and $[V_i - t_i^{\text{in}} N_i, V_i]$, where $t_i^{\text{in}}, t_i^{\text{out}} > 0$. The range factors t_i^{in} and t_i^{out} are defined independently so that samples on either one of the two segments stay always either inside or outside the mesh, with a maximum possible range. This choice allows to deal with the reconstruction of thin structures of the mesh (e.g., the leg of a horse). For each 3D point we obtain corresponding densities $\sigma_{i,1}, \cdots, \sigma_{i,K}$ from the ISNN via $\sigma_{i,j} = F_{\text{shape}}(X_{i,j})$. We then compute normalized weights via the softmax function as $w_{i,j} \propto \exp(\sigma_{i,j})$, such that $\sum_{j=1}^{K} w_{i,j} = 1$. Finally, we define the updated mesh vertex V_i as the following weighted sum

$$\hat{V}_{i} = \sum_{k=1}^{K} w_{i,k} X_{i,k}.$$
(1)

See Figures 6 and 4 for an illustration of these steps.

Remark. In view-centric methods, such as NeRF, the sampling rays are defined via the camera directions. The view-centric approach presents two drawbacks: Firstly, the number of points grows linearly with the number of cameras. Secondly, when using view-centric (VC) sampling, the surface can only evolve within the subspace determined by the camera poses, resulting in elongated shapes (as observed in Figure 3). This limitation becomes particularly challenging in scenarios with sparse camera views.

Adding Texture. Instead of obtaining color directly from the ISNN as in NeRF models [2, 32, 35, 51, 66], we introduce a separate model, the Texture Neural Network (TNN) $F_{\text{texture}} : \mathbb{R}^P \mapsto \mathbb{R}^3$, where P is the size of the 3D position embedding. Given the updated 3D vertex location \hat{V}_i , we compute its positional embedding $\gamma(\hat{V}_i)$, where $\gamma(\cdot)$ denotes the positional encoding operator, and then obtain the color $C_i \doteq F_{\text{texture}}(\gamma(\hat{V}_i))$.

Image Rendering. The above procedure yields an updated triangle mesh $\hat{\mathcal{G}} = (\hat{\mathcal{V}}, \mathcal{F})$, where $\hat{\mathcal{V}} = {\hat{V}_1, \ldots, \hat{V}_M}$, with corresponding vertex colors $C = {C_1, \ldots, C_M}$. We render the image viewed by the *j*-th camera with calibration $\pi_j, j \in {1, \ldots, N}$, by feeding $\hat{\mathcal{G}}$ and the vertex colors C to a differentiable renderer [6, 18, 20, 52]. This yields the rendered image $I_j^r = \text{Renderer}(\hat{\mathcal{G}}, C, \pi_j)$ (see Figure 5).

Reconstruction Loss. F_{shape} and F_{texture} are parametrized as Multi Layer Perceptron (MLP) networks (more details of their architectures are in section 4.1). We train their param-



Figure 4. Detailed model representation. We feed the object-centric points to ISNN and obtain a density value. Then, we update the vertex location via eq. (1) using the points sampled along the vertex normal. We repeat this operation for all vertices to get the updated mesh surface.



Figure 5. We assign a color to each vertex of the mesh by querying the TNN model at that vertex. Then, we feed the textured mesh and a camera viewpoint as input to a differentiable renderer to synthesize a view of the scene. The reconstruction task is based on minimizing the difference between the synthesized view and a captured image (with the same viewpoint) in both L_1 and perceptual norms.

eters by minimizing the following loss on the images I

$$L = L_{\text{images}}(I, I^r) + L_{\text{perceptual}}(I, I^r) + \lambda L_{\text{laplacian}}(\hat{\mathcal{G}}) \quad (2)$$

where $L_{\text{images}}(I, I^r) = \sum_{i=1}^N |I_i - I_i^r|_1$ is the L_1 loss between the rendered images and captured images. $L_{\text{perceptual}}$ is the same loss as L_{images} , but where instead of the L_1 loss we use the perceptual loss [68] and $L_{\text{Laplacian}}$ is the Laplacian loss of the mesh $\hat{\mathcal{G}}$ [38], which we use to regularize the reconstructed 3D vertices $\hat{\mathcal{V}}$ through the parameter $\lambda > 0$. We optimize the loss L using the AdamW optimizer [31].

3.4. Technical Details

We employ several ideas to make the optimization robust and accurate.

Mesh Initialization. We use a robust initialization procedure to obtain a first approximate surface mesh. We start from a predefined sphere mesh with radius ρ . This sphere defines vertices and triangles of the mesh \mathcal{G} . Then, for each vertex $V_i \in \mathcal{V}$ we cast a ray R_i in the radial direction from



Figure 6. Mesh initialization. The points in the radial point cloud are fed to ISNN to obtain a density value. Then, for each ray we update the vertex location using eq. (1). By repeating this operation for all rays, we obtain the mesh surface.

the origin of the sphere towards the vertex V_i and draw K equally spaced points $X_{i,1}, \cdots, X_{i,K}$ along the ray R_i , such that $|X_{i,K}|_2 = \rho$ (see Figure 6). These 3D points are never changed throughout this initial model training. The mesh vertices are then computed as in eq. (1). Because each linear combination considers only samples $X_{i,1}, \cdots, X_{i,K}$ along a ray, the initial representation $F_{\text{shape}}^{\text{init}}$ can only move the updated vertices radially. Although the reconstructed mesh can model only genus zero objects and only describe a radial structure, it gives us a very reliable initial mesh.

Surface Normals. Surface normals are computed by averaging the normals of the faces within the second order neighbourhood around V_i .

Re-meshing. Every 100 iterations during the first 2500 iterations, and then every 250 iterations afterwards, we apply a re-meshing step that regularizes and removes self-intersections from the mesh \mathcal{G} . This is a separate step that is not part of the optimization of the loss eq. (2) (*i.e.*, there is no backpropagation through these operations). Through re-meshing, the mesh can have genus different from zero and its triangles are adjusted so that they have similar sizes. We use the implementation available in PyMesh - Geometry Processing Library [1]. For more details, see the supplementary material. Notice that the total time for the above calculations during training is almost negligible as we do not apply these operations at every iteration and they are highly optimized.

Inside/Outside 3D Points. For the identification of which 3D samples are inside/outside the 3D surface we use the generalized *winding number* technique, which is also available in the PyMesh - Geometry Processing Library [1].

Texture Refinement. During training, we observe that the TNN model does not learn to predict sharp textures. Therefore, we run a final phase during which the mesh is kept constant and we fine-tune the TNN separately. Following IDR [61], we feed the vertex location, the vertex normal and the camera viewing direction to the TNN so that it can describe a more general family of reflectances. These quan-

Table 1. Reported 3D metrics on the GSO Dataset. *Note that we obtained COLMAP-reconstructed point clouds using 50 views. (**All scores have been multiplied by 10^4). Notice the robustness of our method even when not using the mask constraints.

Method	Mask	CH-L2*↓	CH-L1↓	Normal ↑	F@10↑
NeRS	yes	18.58	0.052	0.54	98.10
RegNeRF	yes	60.19	0.107	0.30	91.44
Munkberg	yes	13.32	0.047	0.56	98.65
DS	yes	13.21	0.042	0.71	98.58
NeuS	no	1217.00	0.495	0.37	32.40
NeuS	yes	13.85	0.049	0.70	98.79
COLMAP*	yes	34.35	0.049	-	99.11
Our wo/BCG	yes	8.69	0.034	0.75	99.24
Our w/BCG	no	11.08	0.038	0.75	98.85

Table 2. GSO Dataset: Quantitative evaluation of generated views on the test set. Notice the robustness of our method even when not using the mask constraints.

Method	Mask	$PSNR\uparrow$	$MSE\downarrow$	SSIM \uparrow	LPIPS \downarrow
NeRS	yes	20.108	0.0185	0.874	0.126
RegNeRF	yes	20.217	0.013	0.882	0.143
Munkberg et al.	yes	26.838	0.002	0.955	0.067
DS	yes	24.649	0.004	0.944	0.081
Our wo/BCG	yes	29.029	0.001	0.967	0.028
Our w/BCG	no	27.370	0.002	0.964	0.038

tities are concatenated to $\gamma(\hat{V}_i)$ and then fed as input to F_{texture} . As described earlier on, in this final phase, we optimize eq. (2) only with respect to the parameters of F_{texture} .

Handling the Background. So far, we have not discussed the presence of a background in the scene and have focused instead entirely on the surface of the object. Technically, unless a mask for each view is provided, there is no explicit distinction between the object and the background. Masks give a strong 3D cue about the reconstructed surface, so much so that they can do most of the heavy-lifting in the 3D reconstruction. Thus, to further demonstrate the strength of our sampling and optimization scheme, we introduce a way to avoid the use of user pre-defined segmentation masks. We extend our model with an approximate background mesh representation. For simplicity, we initialize the background on a fixed mesh (a cuboid) that is sufficiently separated from the volume of camera frustrums' intersections. When we reconstruct the background, we only optimize the texture assigned to each vertex of the background. Note that we add a separate TNN to estimate the texture of the background and the texture is viewindependent at all stages. See also the supplementary material for further details.

Table 3. MVMC Car Dataset: Quantitative evaluation of generated views on the test set. Our wo/BCG* and Our w/BCG are trained with the same pre-processing as in NeRS.

Method	Mask	$PSNR\uparrow$	$MSE\downarrow$	SSIM \uparrow	LPIPS \downarrow
NeRS	yes	18.381	0.015	0.852	0.080
RegNeRF	yes	15.776	0.028	0.751	0.259
Munkberg et al.	yes	15.145	0.031	0.761	0.239
DS	yes	18.608	0.015	0.835	0.139
Our w/BCG*	no	18.301	0.015	0.855	0.132
Our wo/BCG*	yes	20.030	0.010	0.867	0.095
Our w/BCG	no	18.450	0.014	0.865	0.142
Our wo/BCG	yes	21.563	0.007	0.883	0.091

4. Experiments

In this section, we present implementation details and results obtained on the standard datasets. For more comprehensive ablation studies as well as for more visual results, we refer to the supplementary material.

4.1. Implementation Details

We parameterize $F_{\text{shape}}^{\text{init}}$, F_{shape} and F_{texture} as MLPs with 5, 5, and 3 layers respectively and a hidden dimension of 256 for all. The initialization mesh uses 2500 vertices, while the mesh for the object reconstruction uses a maximum of 10K vertices. For the background mesh we use a vertex resolution of 10K. When training the texture network TNN in the final step, we upsample the mesh resolution to 250K vertices. The scale t_{in} and t_{out} for the detailed model reconstruction are both set to 0.15. The number of samples along the rays for $F_{\text{shape}}^{\text{init}}$ and F_{shape} is 16 and 8 respectively. The learning rate for the training of the initialization, shape, and texture models is 10^{-5} , 5×10^{-5} , and 10^{-3} . The Laplacian regularization may change across datasets. For objects with non-Lambertian surfaces, e.g., specular surfaces, we use a higher Laplacian regularization, as in this case the F_{shape} network can overfit and generate spiky surfaces due to the lack of multiview consistency across the views. We will release the code of all components of our work to facilitate further research and allow others to reproduce our results.

4.2. Datasets

Google's Scanned Objects (GSO) [10]. We test our algorithm on 14 different objects. For training, we use 8 views, and for validation 100 views. Camera poses are uniformly spread out around the object where the elevation angle is uniformly sampled in $[0^{\circ}, 15^{\circ}]$. The background image is generated by warping a panorama image onto a sphere.

MVMC Car dataset [65]. We run our algorithm on 5 different cars from the MVMC dataset. We use the optimized camera poses provided in the dataset. Although they are optimized, we find that some of them are not correct. Thus, we

Table 4. Tank and Temples Dataset: Quantitative evaluation of generated views on the test set.

	1.4.1.1		DOMD 4	LOF 1	agen ()	I DIDG
Scene	Method	Mask	PSNR \uparrow	MSE↓	SSIM ↑	LPIPS ↓
	RegNeRF	yes	18.078	0.018	0.657	0.254
	Munkberg et al.	yes	18.398	0.018	0.673	0.245
Truck	Our wo/BCG	yes	18.315	0.017	0.701	0.252
	Our w/BCG	no	13.168	0.0508	0.666	0.343
	RegNeRF	yes	21.123	0.105	0.866	0.108
	Munkberg et al.	yes	22.720	0.007	0.873	0.073
Ignatius	Our wo/BCG	yes	23.022	0.007	0.896	0.080
	Our w/BCG	no	17.845	0.021	0.875	0.153

eliminated those views for both training and testing. We follow a leave-one-out cross-validation setup, where we leave one view for validation and the rest is used for training. We repeat this 5 times for each car. Note that this dataset is more challenging than the GSO Dataset [10] as the camera locations are not spread out uniformly around the object. Most of the views are placed mainly on two opposite sides of the cars. Furthermore, the surface of cars is not-Lambertian, and there are many light sources present in the scene too.

Tank and Temple dataset [23]. We evaluate our method on images from 2 objects, *Truck* and *Ignatius*. We use 15 images for training and the rest as the test set. We obtain the image masks of each object by rendering its corresponding laser-scanned ground-truth 3D point cloud. The camera poses are computed via COLMAP's SfM pipeline [46].

4.3. Evaluation

Metrics. For the datasets with a 3D ground truth, we compare the reconstructed meshes with the ground-truth meshes or point clouds. More specifically, we report the L2-Chamfer and L1-Chamfer distances, normal consistency, and F1 score, following [14]. We also report texture metrics to evaluate the quality of the texture on unseen views. More specifically, we employ Mean-Square Error (MSE), Peak Signal-to-Noise Ratio (PSNR), and Structural Similarity Index Measure (SSIM), and Learned Perceptual Image Patch Similarity (LPIPS) [68].

Baselines. We compare our method with the following methods: (1) RegNerf [40], a volume rendering method, (2) Munkberg et al. [37], a hybrid-based method, (3) DS [16], a mesh-based method, (4) COLMAP [46, 47], a multi-view stereo method, (5) NeUS [57], neural surface reconstruction method, and (6) NeRS [65] a neural reflectance surface method. Further details about these baseline methods can be found in the supplementary material.

4.4. Results

We run our algorithm under two settings: with background (*w/BCG*) and by removing the background (*wo/BCG*), the



Figure 7. Qualitative Results on the GSO Dataset. Note that in our w/BCG* column we remove the background rendering in the w/BCG column to simplify the visual comparisons.



Figure 8. Qualitative Results on the MVMC Car dataset. NeRS*: the original NeRS implementation crops the images before training and thus changes their aspect ratio during training. Thus, the rendered images have an aspect ratio of 1, while the original ones do not. For more information, see section 4.3. Note that we remove the background rendering in the w/BCG* column to simplify the visual comparisons.

latter of which is akin to using a mask. We present both qualitative (Figure 17 and Figure 9) and quantitative (Table 1 and Table 2) result on the GSO dataset. We observe that our proposed method for both *w/BCG* and *wo/BCG* is able to recover the original shape with high accuracy. DS, Munkberg et al. [37] and NeuS (with mask supervision) show the closest performance to ours. We observe that NeuS without mask supervision struggles to accurately reconstruct the original shape for seven out of the fourteen objects. NeRS is also able to recover the shape, but cannot

recover genus 1 objects. RegNeRF shows blur artifacts as a result of the inherent ambiguity of sparse input data and also may miss some parts of the original object, *e.g.*, the leg of the cow object. RegNeRF does not always recover the thin parts of the object, *e.g.*, legs of the horse, and thus the reconstructed geometry is not fully accurate. When we run COLMAP with 8 views we find that for most objects the reconstructed point cloud is mostly empty and the object is not recognizable (see also the supplementary material for visual results). This is not surprising since we have only 8



Figure 9. Reconstructed meshes for the COW object in the GSO Dataset.

Table 5. Tank and Temples Dataset: 3D metrics. *Note that we obtained the COLMAP*-reconstructed point clouds using 50 views.

Scene	Method	Mask	Chamfer-Chamfer- F@1		- F@10
			$L2\downarrow$	$L1\downarrow$	\uparrow
	RegNeRF-clean	yes	0.059	0.342	42.56
	Munkberg et al.	yes	0.072	0.355	50.11
Truck	DS	yes	0.110	0.479	31.73
	COLMAP*	yes	0.056	0.298	57.74
	NeuS	no	3.342	2.417	6.50
	NeuS	yes	0.629	1.253	11.93
	Our wo/BCG	yes	0.094	0.406	48.14
	Our w/BCG	no	0.225	0.613	45.78
	RegNeRF-clean	yes	0.106	0.423	43.45
	Munkberg et al.	yes	0.022	0.189	81.75
Ignatius	DS	yes	0.024	0.207	77.68
	COLMAP*	yes	0.013	0.166	86.90
	NeuS	no	0.155	0.572	31.95
	NeuS	yes	0.061	0.3878	37.72
	Our wo/BCG	yes	0.018	0.147	87.12
	Our w/BCG	no	0.139	0.480	55.32

views covering 360-degrees of the object. Furthermore, in this setting, any surface is visible at most from 3 views and the objects does not have a rich texture. Because of this reason, we run COLMAP with 50 views and present the results in all tables just for reference.

In Figure 24 and Table 3 we present qualitative and quantitative results for car objects on MVMC Car dataset. We qualitatively observe that our method wo/BCG shows better view renderings than other methods. Our method fails to recover the texture around transparent surfaces, e.g., the car window. The model in the w/BCG case is able to recover the main shape of the car, but it misses some parts, e.g., the tires of the car that are attached to the ground due to overlaps with the background mesh. Additionally, the tires contain poor texture, e.g., mostly they are black, so this can be easily captured by the background texture network and thus introduce an ambiguity in the reconstruction. We note that the performance for NeRS is consistent across different cars. This is not surprising as they use the mask and their initial template is also car-like. We do not run COLMAP on this dataset as the dataset has a limited number of views. DS has weaker performance on this dataset compared to the GSO. The main reason for poor texture quality is that texture is obtained by 3D back-projections of the mesh to the input views. Thus, incorrect geometry leads to poor texture quality. We observe that the quality of the reconstructions from RegNeRF and Munkberg et al. [37] lack realism. There are two main reasons for this. Firstly, the camera locations are not uniformly spread out around the object. Most of them are located on two sides of the cars. In this case, the methods struggle to recover the original shape. Secondly, because of the many light sources present in the scene and non-Lambertian surfaces, the multi-view consistency across the views is not satisfied. As can be observed, our method is more robust to the above issues. The main reason for that is that during the training of ISNN, the output color of TNN does not depend on the camera view and thus it is less prone to overfitting.

In Table 5 and Table 4 we present quantitative results for two objects in the Tank and Temple datasets. Note that the performance of all methods is drastically decreased especially in the recovered 3D shape compared to the GSO dataset. This is expected as the ground-truth point clouds are hollow (without the bottom), *e.g. Truck*, and the reported numbers only approximate the quality of the shape. Our *wo/ BCG* has a higher Chamfer distance compared to the others although it looks visually better. This is because the corresponding ground-truth shape does not only include the target object, but also some other components from the background, as, *e.g.*, in the *Truck* scene. For visual results and more details, see the supplementary material.

5. Conclusion

We have introduced a novel multi-view stereo method that works with sparse views from a 360 rig. The method can handle this extreme setting by using a novel object-centric sampling scheme and a corresponding hybrid surface representation. The sampling scheme allows to concentrate the updates due to multiple camera views to the same components of the surface representation and to structure the updates so that they result in useful surface changes (along its normals, rather than its tangent space). We have demonstrated the robustness of this method by working without the common mask supervision constraint, by using datasets with diverse 3D objects (GSO Dataset), on scenes with complex illumination sources and with non-Lambertian surfaces (MVMC Car).

Acknowledgements. This work was supported by grant 188690 of the Swiss National Science Foundation

References

- Pymesh geometry processing library for python. https: //pymesh.readthedocs.io/en/latest/index. html. Accessed: 2023-03-08. 5
- [2] Jonathan T. Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P. Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. *ICVV*, 2021. 2, 4
- [3] Jonathan T. Barron, Ben Mildenhall, Dor Verbin, Pratul P. Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. *CVPR*, 2022. 1
- [4] P.J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992. 11
- [5] Anpei Chen, Zexiang Xu, Fuqiang Zhao, Xiaoshuai Zhang, Fanbo Xiang, Jingyi Yu, and Hao Su. Mvsnerf: Fast generalizable radiance field reconstruction from multi-view stereo. In *ICCV*, 2021. 1, 3
- [6] Wenzheng Chen, Huan Ling, Jun Gao, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. In Advances in Neural Information Processing Systems, 2019. 1, 2, 4
- [7] Julian Chibane, Aayush Bansal, Verica Lazova, and Gerard Pons-Moll. Stereo radiance fields (srf): Learning view synthesis from sparse views of novel scenes. In *CVPR*, 2021.
 3
- [8] Chong Bao and Bangbang Yang, Zeng Junyi, Bao Hujun, Zhang Yinda, Cui Zhaopeng, and Zhang Guofeng. Neumesh: Learning disentangled neural mesh-based implicit field for geometry and texture editing. In ECCV, 2022. 3
- [9] Kangle Deng, Andrew Liu, Jun-Yan Zhu, and Deva Ramanan. Depth-supervised NeRF: Fewer views and faster training for free. In CVPR, 2022. 3
- [10] Laura Downs, Anthony Francis, Nate Koenig, Brandon Kinman, Ryan Hickman, Krista Reymann, Thomas B. McHugh, and Vincent Vanhoucke. Google scanned objects: A highquality dataset of 3d scanned household items. In *International Conference on Robotics and Automation*, 2022. 2, 6, 13
- [11] David A Forsyth and Jean Ponce. Computer vision: a modern approach. Prentice Hall Professional Technical Reference, 2002. 1
- [12] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multiview stereopsis. *PAMI*, 2010.
- [13] Yasutaka Furukawa, Carlos Hernández, et al. Multi-view stereo: A tutorial. *Foundations and Trends*® *in Computer Graphics and Vision*, 2015. 1
- [14] Justin Johnson Georgia Gkioxari, Jitendra Malik. Mesh rcnn. In *ICCV*, 2019. 6

- [15] Shubham Goel, Angjoo Kanazawa, and Jitendra Malik. Shape and viewpoints without keypoints. In ECCV, 2020.
- [16] Shubham Goel, Georgia Gkioxari, and Jitendra Malik. Differentiable stereopsis: Meshes from multiple views using differentiable rendering. In *CVPR*, 2022. 1, 2, 6, 11
- [17] Ajay Jain, Matthew Tancik, and Pieter Abbeel. Putting nerf on a diet: Semantically consistent few-shot view synthesis. In *ICCV*, 2021. 3
- [18] Justin Johnson, Nikhila Ravi, Jeremy Reizenstein, David Novotny, Shubham Tulsiani, Christoph Lassner, and Steve Branson. Accelerating 3d deep learning with pytorch3d. In SIGGRAPH Asia 2020 Courses, 2020. 2, 4
- [19] Angjoo Kanazawa, Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. Learning category-specific mesh reconstruction from image collections. In ECCV, 2018. 2
- [20] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In CVPR, 2018. 2, 4
- [21] Tejas Khot, Shubham Agrawal, Shubham Tulsiani, Christoph Mertz, Simon Lucey, and Martial Hebert. Learning unsupervised multi-view stereopsis via robust photometric consistency. In CVPR, 2019. 1
- [22] Mijeong Kim, Seonguk Seo, and Bohyung Han. Infonerf: Ray entropy minimization for few-shot neural volume rendering. In CVPR, 2022. 3
- [23] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. ACM Trans. Graph., 2017. 2, 6
- [24] Aldo Laurentini. The visual hull concept for silhouette-based image understanding. PAMI, 1994. 2
- [25] Jiaxin Li, Zijian Feng, Qi She, Henghui Ding, Changhu Wang, and Gim Hee Lee. Mine: Towards continuous depth mpi with nerf for novel view synthesis. In *ICCV*, 2021. 3
- [26] Chen-Hsuan Lin, Wei-Chiu Ma, Antonio Torralba, and Simon Lucey. Barf: Bundle-adjusting neural radiance fields. In *ICCV*, 2021. 2
- [27] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. In *NeurIPS*, 2020. 1
- [28] Yuan Liu, Sida Peng, Lingjie Liu, Qianqian Wang, Peng Wang, Christian Theobalt, Xiaowei Zhou, and Wenping Wang. Neural rays for occlusion-aware image-based rendering. In CVPR, 2022. 3
- [29] Xiaoxiao Long, Cheng Lin, Peng Wang, Taku Komura, and Wenping Wang. Sparseneus: Fast generalizable neural surface reconstruction from sparse views. ECCV, 2022. 3
- [30] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. page 163–169, 1987. 11
- [31] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019. 4
- [32] Ricardo Martin-Brualla, Noha Radwan, Mehdi S. M. Sajjadi, Jonathan T. Barron, Alexey Dosovitskiy, and Daniel Duckworth. NeRF in the Wild: Neural Radiance Fields for Unconstrained Photo Collections. In CVPR, 2021. 2, 4
- [33] Ishit Mehta, Manmohan Chandraker, and Ravi Ramamoorthi. A level set theory for neural implicit evolution under explicit flows. In *ECCV*, 2022. 3

- [34] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. ACM Trans. Graph., 2019. 1
- [35] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In ECCV, 2020. 1, 2, 4
- [36] Tom Monnier, Matthew Fisher, Alexei A. Efros, and Mathdieu Aubry. Share With Thy Neighbors: Single-View Reconstruction by Cross-Instance Consistency. In *ECCV*, 2022. 2, 15
- [37] Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Müller, and Sanja Fidler. Extracting Triangular 3D Models, Materials, and Lighting From Images. In *CVPR*, 2022. 2, 6, 7, 8, 11, 16, 17, 18, 19
- [38] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Laplacian mesh optimization. In International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia, 2006. 4
- [39] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *CVPR*, 2020. 1, 3
- [40] Michael Niemeyer, Jonathan T. Barron, Ben Mildenhall, Mehdi S. M. Sajjadi, Andreas Geiger, and Noha Radwan. Regnerf: Regularizing neural radiance fields for view synthesis from sparse inputs. In *CVPR*, 2022. 1, 3, 6, 11
- [41] Michael Oechsle, Songyou Peng, and Andreas Geiger. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *ICCV*, 2021. 3
- [42] Xuran Pan, Zihang Lai, Shiji Song, and Gao Huang. Activenerf: Learning where to see with uncertainty estimation. In ECCV, 2022. 1
- [43] Konstantinos Rematas, Ricardo Martin-Brualla, and Vittorio Ferrari. Sharf: Shape-conditioned radiance fields from a single view. In *ICML*, 2021. 3
- [44] Edoardo Remelli, Artem Lukoianov, Stephan Richter, Benoit Guillard, Timur Bagautdinov, Pierre Baque, and Pascal Fua. Meshsdf: Differentiable iso-surface extraction. In *NeurIPS*, 2020. 3
- [45] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In CVPR, 2022. 2
- [46] Johannes L. Schonberger and Jan-Michael Frahm. Structurefrom-motion revisited. In CVPR, 2016. 1, 3, 6, 11
- [47] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *ECCV*, 2016. 1, 3, 6, 11
- [48] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. In *NeurIPS*, 2021. 2
- [49] Alessandro Simoni, Stefano Pini, Roberto Vezzani, and Rita Cucchiara. Multi-category mesh reconstruction from image collections. In *3DV*, 2021. 2

- [50] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3dstructure-aware neural scene representations. In *NeurIPS*, 2019. 1
- [51] Vincent Sitzmann, Semon Rezchikov, William T. Freeman, Joshua B. Tenenbaum, and Fredo Durand. Light field networks: Neural scene representations with single-evaluation rendering. In *NeurIPS*, 2021. 2, 4
- [52] Attila Szabó, Givi Meishvili, and Paolo Favaro. Unsupervised generative 3d shape learning from natural images. *arXiv*:1910.00287, 2019. 2, 4
- [53] Richard Szeliski. Computer vision: algorithms and applications. Springer Nature, 2022. 1
- [54] Ayush Tewari, Justus Thies, Ben Mildenhall, Pratul Srinivasan, Edgar Tretschk, Yifan Wang, Christoph Lassner, Vincent Sitzmann, Ricardo Martin-Brualla, Stephen Lombardi, Tomas Simon, Christian Theobalt, Matthias Niessner, Jonathan T. Barron, Gordon Wetzstein, Michael Zollhoefer, and Vladislav Golyanik. Advances in neural rendering. arXiv:2111.05849, 2021. 3
- [55] Shubham Tulsiani, Nilesh Kulkarni, and Abhinav Gupta. Implicit mesh reconstruction from unannotated image collections. arXiv:2007.08504, 2020. 2
- [56] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T. Barron, and Pratul P. Srinivasan. Ref-NeRF: Structured view-dependent appearance for neural radiance fields. In CVPR, 2022. 2
- [57] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. In *NeurIPS*, 2021. 3, 6, 11
- [58] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul Srinivasan, Howard Zhou, Jonathan T. Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. Ibrnet: Learning multi-view image-based rendering. In CVPR, 2021. 3
- [59] Markus Worchel, Rodrigo Diaz, Weiwen Hu, Oliver Schreer, Ingo Feldmann, and Peter Eisert. Multi-view mesh reconstruction with neural deferred shading. In CVPR, 2022. 2
- [60] Jiamin Xu, Zihan Zhu, Hujun Bao, and Weiwei Xu. A hybrid mesh-neural representation for 3d transparent object reconstruction. arXiv:2203.12613, 2023. 2
- [61] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. *NeurIPS*, 2020. 1, 3, 5
- [62] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. In *NeurIPS*, 2021. 2, 3
- [63] Yufei Ye, Shubham Tulsiani, and Abhinav Gupta. Shelfsupervised mesh prediction in the wild. In CVPR, 2021. 2
- [64] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images. In CVPR, 2021. 3
- [65] Jason Y. Zhang, Gengshan Yang, Shubham Tulsiani, and Deva Ramanan. NeRS: Neural reflectance surfaces for sparse-view 3d reconstruction in the wild. In *NeurIPS*, 2021. 1, 2, 6, 11, 12

- [66] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. arXiv:2010.07492, 2020. 2, 4, 15
- [67] Kai Zhang, Fujun Luan, Zhengqi Li, and Noah Snavely. Iron: Inverse rendering by optimizing neural sdfs and materials from photometric images. In *CVPR*, 2022. 3
- [68] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 4, 6

A. Comparing with Baselines

For all baselines, we run them using their default hyperparameters unless explicitly specified.

RegNeRF [40]¹. The released implementation does not include the appearance regularization loss. We recover the geometry from the predicted density by running the marching cubes [30] algorithm. We note that the extracted mesh contains noisy parts at times. For a fair comparison, we clean out these parts manually and report the results for the manually cleaned mesh. We employ the iterative-closest-point (ICP) [4] algorithm to align the predicted mesh to the ground truth. We train the models by removing the background.

Munkberg et al. $[37]^2$. We run the method without any modification.

NeRS [65]³. The original implementation of the code crops the objects around their mask and thus changes the original aspect ratio of the original images. For a fair comparison, we run our method for both original images and modified images.

DS [16]⁴. We optimize only the mesh and the texture as the camera poses are given.

COLMAP [46, 47]⁵ We run the COLMAP by removing the background as our aim is to estimate the 3D of the object in the scene.

NeuS [57]⁶. We run the NeuS under two settings: with mask supervision and without mask supervision. As our aim is to recover the 3D of an object we define the regions of interest by pointcloud of COLMAP output for both settings as it is explained in the original repository.

B. Ablations

C. Ablations

Narrow view reconstruction. As described in the main paper, our method can also work in the narrow view configuration. In Figure 10 we present reconstructions given *only two*



Figure 10. Reconstructions with narrow camera views. Input views (top), reconstructed meshes (middle) and GT meshes (bottom).

Table 6. 3D metrics on the GSO dataset (without background) for a different number of training views. Note that the result with 6 views is better than other baselines (see Table 1 in paper) and the result with 5 views is better than NerS [57]. *All scores have been multiplied by 10^4 .

# of view	wsCH-L2*↓	CH-L1↓	Normal 1	`F@10↑
3	59.88	0.090	0.53	93.89
4	22.04	0.063	0.61	98.26
5	16.09	0.053	0.67	98.85
6	12.55	0.048	0.68	99.17
7	11.88	0.048	0.68	99.26
8	8.69	0.034	0.75	99.24

views each with a relative pose difference of 30 degree and 90 degree. The algorithm successfully reconstructs complete shapes, albeit with noticeable surface deformations.

Performance vs. the number of views. In this experiment, we measure the performance of our algorithm for a different number of training views in the GSO dataset (see Table 6 and Figure 15). By observing the Figure 15 we see that our algorithm can produce approximately correct shapes even with 5 camera views. Note that for 3 camera views, the coarse shape estimate is reasonable, but the shapes either miss some parts (*e.g.*, the back leg of the horse, the sword of the ninja) or some parts are not distinguishable enough (*e.g.*, face of Mario) due to the high ambiguity of potential valid 3D models.

Number of samples along the ray. In this ablation study, we explore the influence of the number of ray samples. To achieve this, we applied our algorithm to the *Ninja* object (with background) using 2, 4, and 8 samples along the ray. Figure 11 showcases the reconstructions. Notably, our algorithm manages to reconstruct almost all the shape even with only 4 samples along the ray. Nonetheless, there are

¹https://github.com/google-research/googleresearch/tree/master/regnerf ²https://github.com/NVlabs/nvdiffrec

³https://github.com/jasonyzhang/ners

⁴https://github.com/shubham-goel/ds

⁵https://colmap.github.io/

⁶https://github.com/Totoro97/NeuS

Table 7. 3D metrics on the GSO dataset where we skip the first stage. *All scores have been multiplied by 10^4 . **We run for more iterations, *i.e.*, 15K in total.

Method	CH-L2*	¢CH-L1↓	Normal 1	`F@10↑
Our w/BCG	13.75	0.041	0.75	98.79
Our w/BCG**	11.44	0.040	0.76	98.99
Our w/BCG-Ful	1 11.08	0.038	0.75	98.85

still noticeable artifacts, such as the division of the sword into two parts.

Using a sphere as a coarse shape initialization. In Figure 23 we show the shape evolution during the training. As we mention in the paper, the coarse model reconstruction provides us with the robust initialization for the detailed model stage. However, one can start directly from the sphere mesh by skipping the first stage and still obtain decent shapes. To verify this, we start directly from the detailed model stage by skipping the first stage in the GSO dataset. In Table 7 we present our quantitative results. Although the performance dropped by a little amount the performance is still satisfactory.

Figure 14 shows reconstructed meshes, where the initial template mesh was a sphere. We observe that the detailed model representation is strong enough to recover the correct shape even from a basic initial shape (a sphere in this case). However, reconstructions with such initialization show some artifacts. More tuning and longer training might help to remove these artifacts. To verify this assumption, we run the algorithm for more iterations and observe that the performance noticeably improves. On the other hand, starting from the coarse shape that our method produces leads to a more stable training and a faster convergence.

The role of the mesh resolution. In Figure 12 we show the effect of the mesh resolution on the reconstruction of objects with thin parts. Recovering these parts requires a higher mesh resolution, *i.e.*, more triangles. Note that for this case the training is longer.

The role of the Laplacian. In Figure 13, we show the effect of the Laplacian regularizer on an object from the MVMC Car [65] dataset. To recover a smooth shape of objects with non-Lambertian surfaces, one needs to use more regularization. Note that for the lower regularization the F_{shape} network overfits and generates spiky surfaces around the windows of the car.

Calculation of the surface normal. In the paper, we calculate the normal to the vertex V_i in the detailed model representation by averaging the normals of the faces within the second order neighborhood around V_i . In Figure 16, we show reconstructions obtained by calculating the normal to a vertex by averaging the normals of the faces within the *first, second* and *third* order neighbourhoods around V_i . We



Figure 11. Number of samples along the ray.



Figure 12. We show our reconstructions using 10K and 20K vertices for the watch object. Using 10K vertices is insufficient to recover the circle on top of the watch. Increasing the mesh resolution, e.g., to 20K vertices, can fix this easily.



Figure 13. Reconstructed cars for different values λ of the Laplacian regularization. The two rows show the mesh of the same object from two different views.

see that using the third order neighbourhood to calculate the normals does not allow the optimization to fully recover thinner parts of the shape. This is expected since the nor-



Figure 14. Reconstructed meshes of some objects in the GSO [10] dataset by starting from a sphere (*i.e.*, we skip the Coarse Model Reconstruction stage). Recovered meshes are shown from two different views (columns 1 and 3) with their corresponding GT view on their right hand side (columns 2 and 4). The reconstructions in the first and the third rows are satisfactory. However, the reconstructions on the second and the fourth rows have some artifacts, *e.g.*, spiky surfaces, which do not appear with the coarse model initialization.

mal field is smoother in this case (there is more averaging). One can handle this by increasing the mesh resolution. A first order neighbourhood does not allow to recover the thinner parts of the original shape too. On the other hand, the second order neighbourhood allows to recover thin object parts.

D. Implementation Details

For remeshing we use the pseudocode 1 described here.⁷ The algorithm receives the mesh and the target edge length as input. It refines/fixes the mesh by applying the following sequence of operations: 1) removing degenerated triangles, 2) splitting long edges, 3) collapsing short edges, 4) removing obtuse triangles, and 5) computing the outer hull of the mesh. Notice that computing the outer hull allows us to change the topology of the mesh. In our experiments, our meshes have around 10K vertices. In order to match this resolution, we start from the initial target edge length and call the pseudocode. If the new mesh has more than 10K vertices we increase the target edge length threshold and run

it until we have the mesh that has around 10K vertices. For the objects in the GSO dataset, we observe that the remeshing and the classification of inside/outside of the 3D points take around 15% of the total time in the detailed model reconstruction stage. The method takes about 2 hours for training on NVIDIA RTX A5000. Specifically, the coarse shape reconstruction accounts for roughly 15% of the total time, while the fine reconstruction takes up about 35%, and the remaining 50% is for texture refinement.

E. Additional Results

In Figure 19 and Figure 20 we show qualitative results for the Tank and Temple datasets. Note that, for the truck object our w/BCG is able to reconstruct the object, but not the bottom part around it as this was part of the background mesh during training. For the ignatius object, our w/BCG could not recover the bottom (seddle) part and our wo/BCG could recover it partially. The part that was not recovered has a high brightness (almost white color) and there is not enough signal (feedback) from the RGB loss to recover that part. Observe that the performance of NeUS shows a significant decrease when the background is present (without mask supervision). We have smoother meshes for both objects than the baselines. One can increase the vertex resolution to recover more details. In Figures 21, 18 and 24 we show reconstructions for more objects in the GSO dataset and MVMC car dataset respectively. Figure 22 shows the reconstructed mashes for NeuS w/BCG. Notice that the method cannot accurately estimate the mesh for most of the objects.

COLMAP results. In order to increase the quality of the reconstructions we eliminate the background from the raw images and run the COLMAP with the white background. Figure 25 shows dense point clouds reconstructed via COLMAP from 8 views in the GSO dataset. We observe that the reconstructed point clouds are either indistinguishable or miss many parts from the object. This is due to the sparse view setting since any part of the object is seen from at most 3 views and as well as the poor texture of the objects. In Figure 26 and 27 we show reconstructed dense point clouds and meshes with 50 views in the GSO dataset and Tank and Temple dataset respectively. The reconstructed point clouds are noticeably better than the ones with 8 views. However, they still might have artifacts, *i.e.*, holes and also it might still not work for objects with uniform texture, i.e., the porcelain white pitcher (third row-last column). Note that also the reconstructed meshes are not smooth and not watertight. One needs to post-process them via remeshing to remove many of these artifacts.

F. Discussion

Even though our proposed method performs well, it has limitations. For objects with thinner surfaces, one needs to in-

⁷https://github.com/PyMesh/PyMesh/blob/ 384ba882b7558ba6e8653ed263c419226c22bddf/scripts/ fix_mesh.py#L14



Figure 15. Reconstructed object with different number of input views for objects in the GSO dataset.



Figure 16. Reconstructed meshes for the *ninja* object using averaging of the face normals in the first, second and third order neighbourhood around a vertex.

Algorithm 1:	Python	style	pseudocode	remeshing
--------------	--------	-------	------------	-----------

```
def remesh(mesh, target_len):
                 target edge length
     It removes triangles having collinear
    vertices i.e. zero areas
   mesh =
    pymesh.remove_degenerated_triangles(mesh)
    # It split long edges into 2 or more shorter
    edges
   mesh = pymesh.split_long_edges(mesh,
     target_len)
    # It collapses short edges
   mesh = pymesh.collapse_short_edges(mesh, 1e-6)
   mesh = pymesh.collapse_short_edges(mesh,
     target_len, preserve_feature = True
    # It removes obtuse triangles i.e. triangles
     having one of the interior angles more than
    90 degrees
   mesh = pymesh.remove_obtuse_triangles(mesh.
    150, 100)
   mesh = pymesh.resolve_self_intersection(mesh)
   # It removes vertices with nearly same
   mesh = pymesh.remove_duplicated_faces(mesh)
    # It computer the outer hull of the input mesh
   mesh = pymesh.compute_outer_hull(mesh, 100)
   mesh = pymesh.remove_duplicated_faces(mesh)
   mesh = pymesh.remove_obtuse_triangles (mesh,
    179, 5)
    # It removes vertices not referred by any face
   mesh = pymesh.remove_isolated_triangles(mesh)
   return mesh
```

crease the mesh resolution (see Figure 12) and this affects the computational load of the method. Although in our experiments we found that 10K vertices works fine for most of the objects, one needs to adjust the mesh resolution for objects having more details.

The method can easily update its topology via remeshing during the learning. This allows reconstructing objects that are not homeomorphic to spheres. However if there is a hole at the center of the ground truth object our model would not handle that as our initial shape is either a sphere or a coarse shape that is homeomorphic to a sphere.

We handle the background by fixing its geometry to a simple shape, e.g., a cuboid, and by updating only its texture. Thus, the background is multiview consistent as well. Note that, in this study our goal is to model only the main object in the scene, not the whole scene. We observed that the method is working sufficiently well if the background geometry is locally correct around the main object in the scene. One can consider different options, *i.e.*, generating separate background images for each training image like Monnier et al. [36] or model the background like NeRF++ [66]. In both cases, the background texture would not be multiview consistent and this would break the training. The current model might have an issue when the background is close to the foreground object, as, e.g., in the MVMC Car dataset. Handling this case is quite challenging and still an open problem. To our knowledge, we are the first among the explicit mesh-based representation methods to propose including the background in the pipeline. We leave handling more complex background models to future work.



Figure 17. Qualitative Results on the GSO Dataset. Note that in our w/BCG* column we remove the background from our w/BCG for better visualization. Differences can be better appreciated by zooming in.



Figure 18. Additional qualitative results on the GSO Dataset.



Figure 19. Reconstructed meshes for the *truck* and *ignatius*. Note that our meshes are smoother due to the vertex resolution.



Figure 20. Qualitative results on the Tank and Temple Dataset. The reconstructed objects are rendered from three views. Note that in our w/BCG* column we remove the background from our w/BCG for better visualization.



Figure 21. Reconstructed meshes for more objects in the GSO Dataset.



Figure 22. Reconstructed meshes for NeuS with background. The corresponding GT meshes are shown in Figure 21 (from top to bottom)



Figure 23. Evolution of the shape over iteration time for coarse and detailed model training on the *Ninja Turtles Leonardo* (top) and *Breyer Horse* (bottom) from the GSO dataset with 8 views and with the background. We show the mesh normals in false colors.



Figure 24. Qualitative results on the MVMC Car dataset. NeRS*: the original NeRS implementation crops the images before training and thus changes their aspect ratio during training. Thus, the rendered images have an aspect ratio of 1, while the original ones do not. Note that in our w/BCG* column we remove the background from our w/BCG results for better visualization.



Figure 25. COLMAP-reconstructed dense point clouds with 8 input views from known ground-truth cameras for objects in the GSO Dataset. Better viewed by zooming in.



Figure 26. COLMAP-reconstructed dense point clouds with 50 input views from known ground-truth cameras for objects in the GSO Dataset. Better viewed by zooming in.



Figure 27. COLMAP-reconstructed dense point clouds with 50 input views from known ground-truth cameras for *Truck* and *Ignatius* objects. Better viewed by zooming in.