Common Objects in 3D:
Large-Scale Learning and Evaluation of Real-life 3D Category Reconstruction

Jeremy Reizenstein[1] Roman Shapovalov[1] Philipp Henzler[2] Luca Sbordone[1]
Patrick Labatut[1] David Novotny[1]

{reizenstein,romansh,lsbordone,plabatut,dnovotny}@fb.com {p.henzler}@cs.ucl.ac.uk

[1]Facebook AI Research [2]University College London

CO3D https://github.com/facebookresearch/co3d

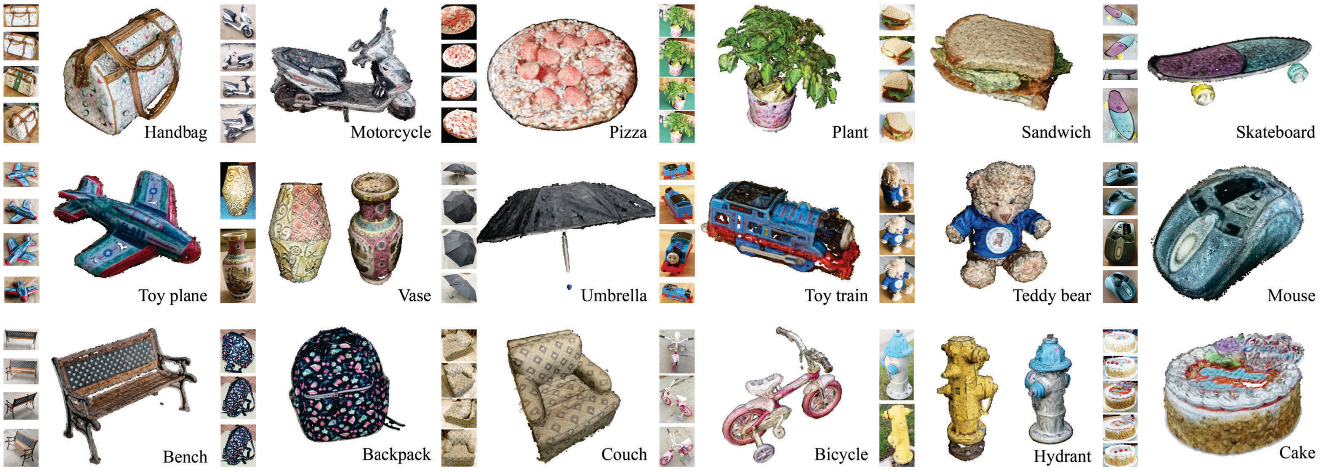

Figure 1: We introduce the **Common Objects in 3D (CO3D)** dataset comprising 1.5 million multi-view images of almost 19k objects from 50 MS-COCO categories annotated with accurate cameras and 3D point clouds (visualized above).

Abstract

Traditional approaches for learning 3D object categories have been predominantly trained and evaluated on synthetic datasets due to the unavailability of real 3D-annotated category-centric data. Our main goal is to facilitate advances in this field by collecting real-world data in a magnitude similar to the existing synthetic counterparts. The principal contribution of this work is thus a large-scale dataset, called Common Objects in 3D*, with real multi-view images of object categories annotated with camera poses and ground truth 3D point clouds. The dataset contains a total of 1.5 million frames from nearly 19,000 videos capturing objects from 50 MS-COCO categories and, as such, it is significantly larger than alternatives both in terms of the number of categories and objects.*

We exploit this new dataset to conduct one of the first large-scale "in-the-wild" evaluations of several new-view-synthesis and category-centric 3D reconstruction methods. Finally, we contribute NerFormer *- a novel neural rendering method that leverages the powerful Transformer to reconstruct an object given a small number of its views.*

1. Introduction

Recently, the community witnessed numerous advances in deeply learning to reconstruct category-centric 3D models. While a large variety of technical approaches was proposed [56, 62, 13, 25, 26, 46, 8], they are predominantly trained and benchmarked either on synthetic data [6], or on real datasets of specific object categories such as birds [62] or chairs [37]. The latter is primarily a consequence of a lack of relevant real-world datasets with 3D ground truth.

Our main goal is therefore to collect a large-scale open real-life dataset of common objects in the wild annotated with 3D ground truth. While the latter can be collected with specialized hardware (turn-table 3D scanner, dome [29]), it is challenging to reach the scale of synthetic datasets [6] comprising thousands of instances of diverse categories.

Instead, we devise a photogrammetric approach only requiring object-centric multi-view RGB images. Such data can be effectively gathered in huge quantities by means of crowd-sourcing “turn-table” videos captured with smartphones, which are nowadays a commonly owned accessory. The mature Structure-from-Motion (SfM) framework then provides 3D annotations by tracking cameras and reconstructs a dense 3D point cloud capturing the object surface.

To this end, we collected almost 19,000 videos of 50 MS-COCO categories with 1.5 million frames, each annotated with camera pose, where 20% of the videos are annotated with a semi-manually verified high-resolution 3D point cloud. As such, the dataset exceeds alternatives [9, 1, 21] in terms of number of categories and objects.

Our work is an extension of the dataset from [26]. Here, we significantly increase the dataset size from less than 10 categories to 50 and, more importantly, conduct a human-in-the-loop check ensuring reliable accuracy of all cameras. Finally, the dataset from [26] did not contain any point cloud annotations, the examples of which are in fig. 1.

We also propose a novel *NerFormer* model that, given a small number of input source views, learns to reconstruct object categories in our dataset. NerFormer mates two of the main workhorses of machine learning and 3D computer vision: Transformers [65] and neural implicit rendering [43]. Specifically, given a set of 3D points along a rendering ray, features are sampled from known images and stacked into a tensor. The latter is in fact a ray-depth-ordered sequence of sets of sampled features which admits processing with a sequence-to-sequence Transformer. Therefore, by means of alternating feature pooling attention and ray-wise attention layers, NerFormer learns to jointly aggregate features from the source views and raymarch over them.

Importantly, NerFormer outperforms a total of 14 baselines which leverage the most common shape representations to date. As such, our paper conducts one of the first truly large-scale evaluations of learning 3D object categories in the wild.

2. Related Work

In this section we review current 3D datasets and related methods in the areas of single-image reconstruction, generative modelling and novel-view synthesis.

3D object datasets The main enabler of early works in 3D reconstruction was the synthetic ShapeNet [6] dataset. Pascal3D [72] introduced a real world dataset providing pose estimation for images, but only approximate 3D models. Choi et al. [9] provide a large set of realobject-centric RGB-D videos, however only a small subset is annotated with 3D models and cameras. Increasing the number of categories and objects, Objectron [1] contains object-centric videos, object/camera poses, point clouds, surface planes and 3D bounding boxes. Unfortunately, only a limited number of object-centric videos cover full 360 degrees. Our dataset further increases the number of categories by a factor of 5 and covers the full 360 degree range. Requiring 3D scanners, GSO [21] provides clean full 3D models including textures of real world objects. Due to the requirement of 3D scanning, it contains less objects. A detailed comparison of the aforementioned datasets is presented in tab. 1.

Dataset	[6]	[72]	[9]	[1]	[21]	[26]	**Ours**
Real	×	✓	✓	✓	✓	✓	✓
# Categories	55	12	9	9	NA	7	50
# Objects	51k	36k	2k	15k	2k	2k	19k
3D GT	full	approx.	depth	pcl, box	full	pcl	pcl
Multi-view	full	none	full	limited	full	full	full

Table 1: Common Objects in 3D compared to alternatives. The “pcl” abbreviation stands for “point clouds”.

3D reconstruction A vast amount of methods studied fully supervised 3D reconstruction from 2D images making use of several different representations: voxel grids [10, 18], meshes [19, 67], point clouds [14, 73], signed distance fields, [49, 2] or continuous occupancy fields [42, 8, 17, 16].

Methods overcoming the need for 3D supervision are based on differentiable rendering allowing for comparison of 2D images rather than 3D shapes [51, 63, 31]. Generating images from meshes was achieved via soft rasterization in [32, 40, 61, 30, 7, 36, 77, 20, 67]. Volumetric represenations are projected to 2D via differentiable raymarching [25, 15, 41, 43, 39, 55] or in a similar fashion via sphere tracing for signed distance fields [45, 74]. [28] introduce differentiable point clouds. Another line of work focuses on *neural rendering*, i.e. neural networks are trained to approximate the rendering function [44, 56]. [35, 34] map pixels to object-specific derformable template shapes. [47, 48] canonically align point clouds of object-categories in an unsupervised fashion, but do not reason about colour. Exploiting symmetries and reasoning about lighting [71], compose appearance, shape and lighting.

Similar to us, [52, 75, 26, 60] utilize per-pixel warp-conditioned embedding [26]. In contrast to our method, multi-view aggregation is handled by averaging over encodings which is prone to noise. Instead, we learn the aggregation by introducing the NerFormer module. Finally, a very recent IBRNet [68] learns to copy existing colors from known views in an IBR fashion [24, 5], whereas our method can hallucinate new colors, which is crucial since we leverage far fewer source views (at most 9).

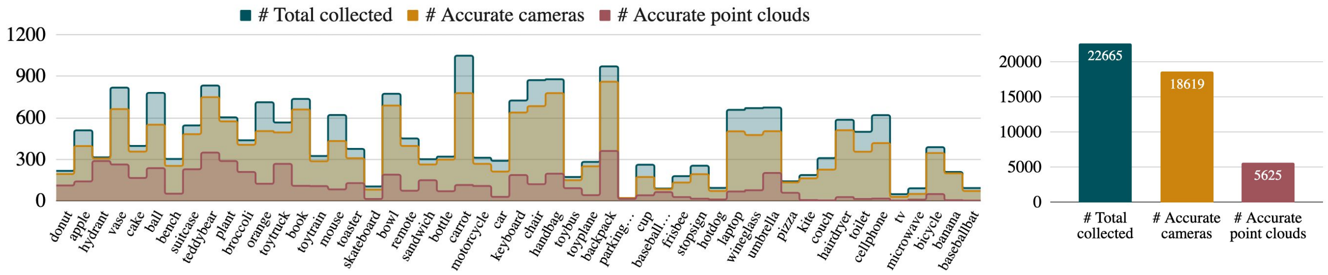

Figure 2: **Statistics of the Common Objects in 3D dataset** reporting the total (right) and per-category (left) numbers of collected videos, and the number of videos with accurate cameras and point clouds.

3. Common Objects in 3D

In this section we describe the dataset collection process.

AMT video collection In order to scale the collection of object-centric videos, we crowd-sourced it on Amazon Mechanical Turk (AMT). Each AMT task asks a worker to select an object of a given category, place it on a solid surface and take a video where they keep the whole object in view while moving full circle around it. We pre-selected 50 MS-COCO [38] categories (listed in fig. 2) comprising stationary objects that are typically large enough to be reconstructed. The workers were instructed to avoid actions that would hinder the ensuing reconstruction stage such as abrupt movements leading to motion blur. Each video was reviewed to ensure that it fulfills the requirements.

Generating 3D ground-truth As explained below, we use off-the-shelf software to produce object masks, camera tracking, and 3D reconstructions for each video. We then semi-automatically filter out poor reconstructions.

1) Sparse reconstruction Given the set of valid object-centric videos, we reconstruct the extrinsic (3D location and orientation) and intrinsic (calibration) properties of the cameras that captured the videos. To this end, each video is first converted into a time-ordered sequence of images $\mathcal{V} = (I^i \mid I^i \in \mathbb{R}^{3\times H\times W})_{i=1}^{N_I}$ by extracting $n_I = 100$ frames uniformly spaced in time. The frames are then fed to the COLMAP SfM pipeline [53] which annotates each image with camera projection matrices $\mathcal{P} = (P^i \mid P^i \in \mathbb{R}^{4\times 4})_{i=1}^{N_I}$. Fig. 3 shows example camera tracks together with estimated sparse scene geometries.

2) Object segmentation We segment the object in each image I^i with PointRend [33], state-of-the-art instance segmentation method, resulting in a sequence of soft binary masks $\mathcal{M} = (M^i \mid M^i \in [0,1]^{H\times W})_{i=1}^{N_I}$ per video. Note that, while masks aid further dense reconstruction, we have not used them for camera tracking which is typically anchored to the background regions.

3) Semantic dense reconstruction Having obtained the camera motions and segmentations, we now describe the process of annotating the captured objects with a 3D surface. We first execute the multi-view stereo (MVS) algorithm of COLMAP [54] to generate per-frame dense depth maps $(D^i \mid D^i \in \mathbb{R}_+^{H\times W})_{i=1}^{N_I}$. We then run COLMAP's point cloud fusion algorithm, which back-projects the depth values masked with $\mathcal{M}$ and retains the points that are consistent across frames, to get a point cloud $\mathcal{P}(\mathcal{V}) = \{\mathbf{x}_i\}_{i=1}^{N_\mathcal{P}}$. Example dense point clouds are visualized in fig. 3.

4) Labelling reconstruction quality with Human-in-the-loop. Since our reconstruction pipeline is completely automated, any of the aforementioned steps can fail, resulting in unreliable 3D annotations. We thus incorporate a semi-manual check that filters inaccurate reconstructions.

To this end, we employ an active learning [11] pipeline which cycles between: a) manually labelling the point-cloud and camera-tracking quality; b) retraining a "quality" SVM classifier; and c) automatically estimating the shape/tracking quality of unlabelled videos. Details of this process are deferred to the supplementary.

5) The dataset The active-learned SVM aids the final dataset filtering step. As accurate camera annotations are crucial for the majority of recent 3D category reconstruction methods, the first filtering stage completely removes all scenes with camera tracking classified as "inaccurate" (18% of all videos). While all videos that pass the camera check are suitable for training, the scenes that pass both camera and point cloud checks (30% of the videos with accurate cameras) comprise the pool from which evaluation videos are selected. Note, a failure to pass the point cloud check does not entail that the corresponding scene is irreconstructible and should therefore be removed from training – instead this merely implies that the MVS method [54] failed, while other alternatives (see sec. 5) could succeed.

Fig. 2 summarizes the size of Common Objects in 3D. Reconstructing a single video took on average 1h 56 minutes with the majority of execution time spent on GPU-accelerated MVS and correspondence estimation. This amounts to the total of 43819 GPU-hours of reconstruction time distributed to a large GPU cluster.

4. Learning 3D categories in the wild

Here, we describe the problem set, give an overview of implicit shape representations, and explain our main techni-

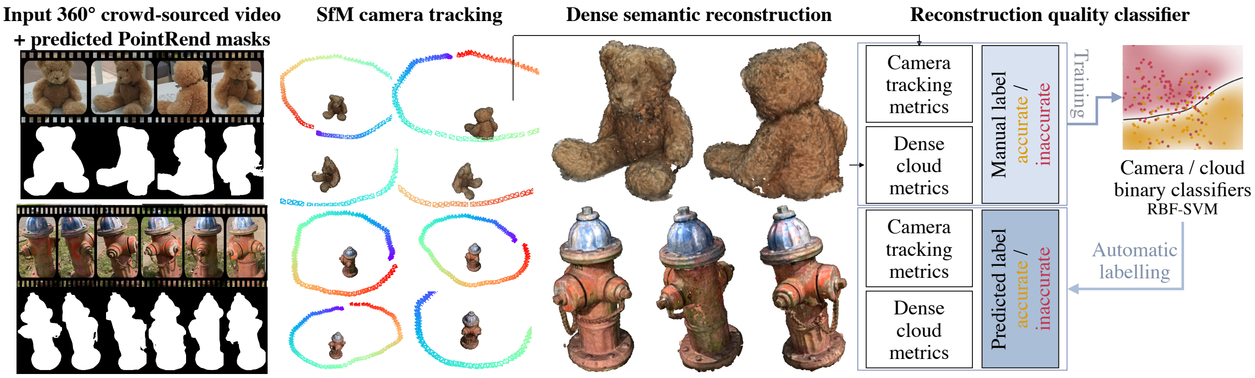

Figure 3: **3D ground truth for Common Objects in 3D** was generated with active learning. Videos are first annotated with cameras and dense point clouds with COLMAP [53]. Given several reconstruction metrics and manual binary annotations ("accurate"/"inaccurate") of a representative subset of reconstructions, we train an SVM that automatically labels all videos.

cal contribution: the NerFormer neural rendering model.

Problem setting We tackle the task of generating a representation of appearance and geometry of an object given a small number of its observed source RGB views $\{I_i^{\text{src}}\}_{i=1}^{N_{\text{src}}}$ and their cameras $\{P_i^{\text{src}}\}_{i=1}^{N_{\text{src}}}$ which, in our case, are samples from the set of frames $\mathcal{V}_v$ and cameras $\mathcal{P}_v$ of a video v.

In order to visualize the generated shape and appearance, we differentiably render it to a *target* view for which only the camera parameters $P^{\text{tgt}} \in \mathbb{R}^{4\times 4}$ are known. This results in a render $\hat{I}^{\text{tgt}}$ which is, during training, incentivised to match the ground truth target view I^{tgt}.

Methods are trained on a dataset of category-centric videos $\{\mathcal{V}_v\}_{v=1}^{n_\mathcal{V}}$, which allows to exploit the regular structure of the category for better generalization to previously unseen objects. Note that, in our experiments (sec. 5), we additionally consider "overfitting" with $n_\mathcal{V} = 1$.

4.1. Representing common objects in 3D

In order to predict novel views, all methods need to reconstruct 3D shape in some form. The latter can be represented with a plethora of recent frameworks such as voxel grids [64, 41], implicit surfaces [43, 46, 45], point clouds [28, 69], or meshes [40, 50]. Among those, implicit representations have been successfully applied to reconstructing real scenes and object categories [43, 25, 75], therefore we choose to use them in our pipeline.

Implicit surface An implicit surface is defined as a level set $\mathcal{S}_f = \{\mathbf{x} \mid f(\mathbf{x}, \mathbf{z}) = C, C \in \mathbb{R}\}$ of a function $f : \mathbb{R}^3 \times \mathbb{D}_z \mapsto \mathbb{R}$ that accepts a 3D point $\mathbf{x} \in \mathbb{R}^3$ and D_z-dimensional latent code $\mathbf{z} \in \mathbb{R}^{D_z}$. In addition to f, which represents geometry, a second function $c : \mathbb{R}^3 \times \mathbb{S}^2 \times \mathbb{D}_z \mapsto \mathbb{R}^3$ assigns colors $c(\mathbf{x}, \mathbf{r}, \mathbf{z})$ to the input points $\mathbf{x}$. Note that, in line with recent work [43, 74], c is further conditioned on the 3D direction vector $\mathbf{r} \in \mathbb{S}^2$ from which $\mathbf{x}$ is imaged in order to model viewpoint-dependent effects such as specularities. Finally, both functions f and c depend on the latent code $\mathbf{z}$ encoding geometry and appearance of the scene. Changes in $\mathbf{z}$ alter the level set of f and colors c allowing for representing different instances of object categories.

Following recent successes of [55, 26], we model category-specific 3D shapes with opacity functions f_o. Specifically, f_o assigns $f_o(\mathbf{x}, \mathbf{z}) = 0$ to the unoccupied points $\mathbf{x} \notin \mathcal{S}_f$, and $f(\mathbf{x}', \mathbf{z}) > 0$ to surface points $\mathbf{x}' \in \mathcal{S}_f$.

Neural implicit surfaces Recent methods implement functions f_o and c as multi-layer perceptrons (MLP) f_{MLP} and c_{MLP} [43, 45, 22, 2]. They typically learn shallow specialized networks on top of the shared deep feature extractor f'_{MLP}: $f_{\text{MLP}} = f_{\text{HEAD}} \circ f'_{\text{MLP}}$ and $c_{\text{MLP}} = c_{\text{HEAD}} \circ f'_{\text{MLP}}$.

We depart from representing occupancies with plain MLPs as they process input 3D points $\mathbf{x}$ independently, without any form of spatial reasoning, which is crucial in our case where input source views provide only a partial information about the reconstructed shape.

Positional embedding Following [43, 66], avoiding loss of detail, we pre-process the raw 3D coordinates $\mathbf{x}$ with a positional embedding (PE) $\gamma(\mathbf{x}) = [\sin(\mathbf{x}), \cos(\mathbf{x}), ..., \sin(2^{N_f}\mathbf{x}), \cos(2^{N_f}\mathbf{x})] \in \mathbb{R}^{2N_f}$ before feeding to $f_{\text{MLP}}(\mathbf{x}, \mathbf{z})$. While [43] was the first to demonstrate benefits of PE, in sec. 5 we also combine PE with other pre-NeRF methods, such as SRN [56] or DVR [45].

Rendering an implicit surface In order to admit image-supervised learning, implicit surfaces are converted into an explicit representation of appearance and geometry with a rendering function r. Formally, given a target camera P^{tgt}, the goal is to generate the target image $I^{\text{tgt}} = r(f, c, P^{\text{tgt}})$ which depicts the scene from P^{tgt}'s viewpoint.

We render opacity fields with the Emission-Absorption model (EA). EA renders the RGB value $I^{\text{tgt}}_{\mathbf{u}} \in \mathbb{R}^3$ at a pixel $\mathbf{u} \in \{1, \dots, W\} \times \{1, ..., H\}$, by evaluating the opacity function $f_o(\mathbf{x}, \mathbf{z})$ for an ordered point-tuple $(\mathbf{x}_i^{\mathbf{r}_\mathbf{u}})_{i=1}^{N_S}$ sampled along $\mathbf{u}$'s projection ray $\mathbf{r}_\mathbf{u}$ at approximately equidistant intervals Δ. The color $I^{\text{tgt}}_{\mathbf{u}}(\mathbf{r}_\mathbf{u}, \mathbf{z}) = \sum_{i=1}^{N_S} w_i(\mathbf{x}_i^{\mathbf{r}_\mathbf{u}}, \mathbf{z}) c(\mathbf{x}_i^{\mathbf{r}_\mathbf{u}}, \mathbf{r}_\mathbf{u}, \mathbf{z})$ is a sum of per-point colors

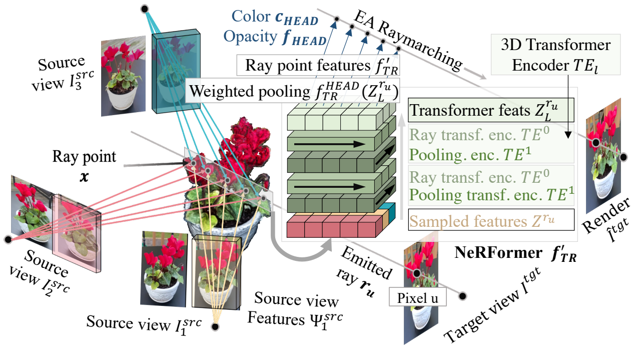
Color $\boldsymbol{c}_{HEAD}$
Opacity $\boldsymbol{f}_{HEAD}$
EA Raymarching
3D Transformer Encoder TE_l
Source view I_3^{src}
Ray point features f'_{TR}
Weighted pooling $f_{TR}^{HEAD}(Z_L^{r_u})$
Transformer feats $Z_L^{r_u}$
Ray transf. enc. TE^0
Pooling. enc. TE^1
Ray transf. enc. TE^0
Pooling transf. enc. TE^1
Sampled features Z^{r_u}
Ray point $\boldsymbol{x}$
Render $\hat{I}^{tgt}$
NeRFormer $\boldsymbol{f}'_{TR}$
Emitted ray $\boldsymbol{r_u}$
Source view I_2^{src}
Source view Features Ψ_1^{src}
Pixel u
Target view I^{tgt}
Source view I_1^{src}

Figure 4: **We propose NerFormer** which jointly learns to pool features from source views and to raymarch by means of a series of transformers alternating between attention along the ray and pooling dimensions.

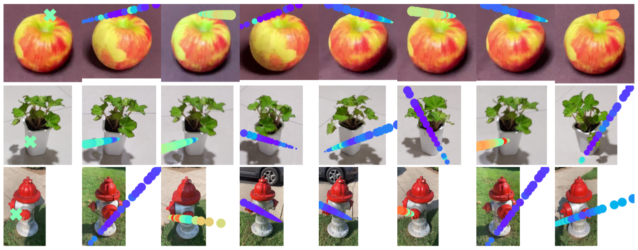

Figure 5: **NerFormer learns to attend to features from source images**. A ray is emitted from a target-image pixel (1st column), and its points are projected to the source views (columns 2-8) from which features are sampled. For each source feature, NerFormer predicts attention (red=high, blue=low) for aggregating to a single source embedding per point. Note how the attention model implicitly learns to pool features from nearby source views.

$c(\mathbf{x}_i^{\mathbf{r_u}}, \mathbf{r_u}, \mathbf{z})$ weighted by the emission-absorption product $w_i = \left(\prod_{j=0}^{i-1} T_j\right)(1 - T_i)$ with $T_i = \exp(-\Delta f_o(\mathbf{x}_i^{\mathbf{r}_u}, \mathbf{z}))$.

4.2. Latent shape encoding z

A crucial part of a category-centric reconstructor is the latent embedding $\mathbf{z}$. Early methods [18, 70, 58, 61, 40] predicted a global scene encoding $\mathbf{z}_{\text{global}} = \Phi_{\text{CNN}}(I^{\text{src}})$ with a deep convolutional network Φ_{CNN} that solely analyzed the colors of source image pixels. While this approach was successful for the synthetic ShapeNet dataset where shapes are rigidly aligned, it has been recently shown in [26, 75] that such approach is infeasible in real world settings where objects are arbitrarily placed in the 3D space. This is because, unlike in the former case where color-based shape inference is possible since similar images of aligned scenes generate similar 3D shapes, in the latter case, similarly looking images of unaligned scenes can generate vastly different 3D.

Warp-conditioned embedding To fix the latter, [26] proposed Warp-Conditioned Embedding (WCE): given a world coordinate point $\mathbf{x}$ and a source view I^{src} with camera P^{src}, the warp-conditioned embedding $\mathbf{z}_{\text{WCE}} \in \mathbb{R}^{D_z}$

$$\mathbf{z}_{\text{WCE}}(\mathbf{x}, I^{\text{src}}, P^{\text{src}}) = \Psi_{\text{CNN}}(I^{\text{src}})[\pi_{P^{\text{src}}}(\mathbf{x})],$$

is formed by sampling a tensor of source image descriptors $\Psi_{\text{CNN}}(I^{\text{src}}) \in \mathbb{R}^{D_z \times H \times W}$ at a 2D location $\pi_{P^{\text{src}}}(\mathbf{x})$. Here, $\pi_{P^{\text{src}}}(\mathbf{x}) = P^{\text{src}}[\mathbf{x}; 1] = d_{\mathbf{u}}[\mathbf{u}; 1]$ expresses perspective camera projection of a 3D point $\mathbf{x}$ to a pixel $\mathbf{u}$ with depth $d_{\mathbf{u}} \in \mathbb{R}$. Intuitively, since $\mathbf{z}_{\text{WCE}}$ is a function of the world coordinate $\mathbf{x}$, the ensuing implicit f can perceive the specific 3D location of the sampled appearance element in the world coordinates, which in turn enables f to learn invariance to rigid scene misalignment.

In the common case where multiple source views are given, the aggregate WCE $\mathbf{z}^*_{\text{WCE}}(\mathbf{x}, \{I_i^{\text{src}}\}, \{P_i^{\text{src}}\})$ is defined as a concatenation of the mean and standard deviation of the set of view-specific source embeddings $\{\mathbf{z}_{\text{WCE}}(\mathbf{x}, I_i^{\text{src}}, P^{\text{src}})\}_{i=1}^{N_{\text{src}}}$.

Boosting baselines with WCE The vast majority of existing methods for learning 3D categories leverage global shape embeddings $\mathbf{z}_{\text{global}}$, which renders them inapplicable to our real dataset. As WCE has been designed to alleviate this critical flaw, and because of its generic nature, in this paper we endow state-of-the-art category-centric 3D reconstruction methods with WCE in order to enable them for learning category-specific models on our real dataset.

To this end, we complement SRN [56], NeuralVolumes [41], and the Implicit Point Cloud (discussed later) with WCE. These extensions are detailed in the supplementary.

4.3. Attention is all you nerf

Limitations of WCE While [26] has demonstrated that the combination of NeRF and WCE (termed NeRF-WCE) leads to performance improvements, our experiments indicated that a major shortcoming of NeRF-WCE is its inability to deal with cases where parts of the 3D domain are labelled with noisy WCE. This is because NeRF's MLP f'_{MLP} independently processes each 3D point and, as such, cannot detect failures and recover from them via *spatial reasoning*. The 3D deconvolutions of Neural Volumes [41] are a potential solution, but we found that the method ultimately produces blurry renders due to the limited resolution of the voxel grid. The LSTM marcher of SRN [56] is capable of spatial reasoning, which is however somewhat limited due to the low-capacity of the LSTM cell. Last but not least, a fundamental flaw is that simple averaging of the source-view WCE embeddings can suppress important features.

Our main technical contribution aims to alleviate these issues and follows a combination of two main design guidelines: 1) We replace the f'_{MLP} with a more powerful architecture *capable of spatial reasoning*. 2) Instead of engineering the WCE aggregation function, we propose to *learn* it.

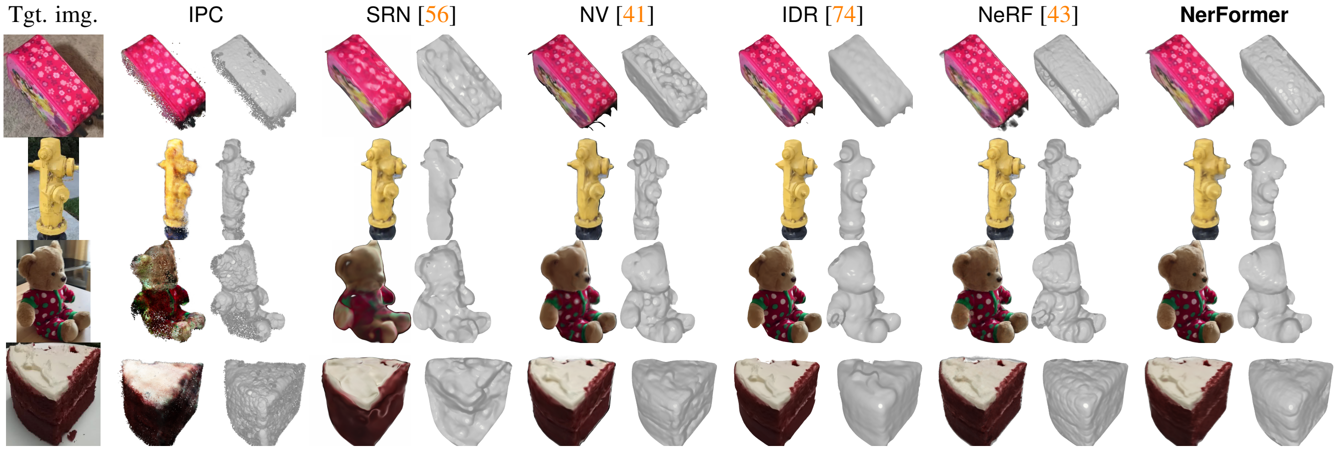

Figure 6: **Single-scene new-view synthesis on Common Objects in 3D** depicting a target image from the training video (left), and corresponding synthesized view generated by IPC, SRN [56], NV [41], IDR [74], NeRF [43], and our NerFormer .

NerFormer As a solution, we propose to leverage the popular Transformer architecture [65]. Employing a sequence-to-sequence model is intuitive since the set of all WCE embeddings along a projection ray is in fact a depth-ordered descriptor sequence along the ray dimension, and an unordered sequence along the feature pooling dimension.

Formally, given a ray $\mathbf{r_u}$, we define $Z^{\mathbf{r_u}} \in \mathbb{R}^{N_S \times N_{\text{src}} \times D_{\mathbf{z}}}$ as a stacking of un-aggregated WCEs of all ray-points $\mathbf{x}_j^{\mathbf{r}_u}$:

$$Z^{\mathbf{r_u}} = \left(\left\{ \mathbf{z}_{\text{WCE}}(\mathbf{x}_j^{\mathbf{r}_u}, I_i^{\text{src}}, P_i^{\text{src}}) \right\}_{i=1}^{n_{\text{src}}} \right)_{j=1}^{N_S}. \quad (1)$$

The *NerFormer* module $f'_{\text{TR}}(Z^{\mathbf{r_u}}) = f_{\text{TR}}^{\text{HEAD}} \circ \text{TE}_L \circ \cdots \circ \text{TE}_1(Z^{\mathbf{r_u}})$ replaces the feature backbone f'_{MLP} (sec. 4.1) with a series of L 3D transformer modules TE_l terminated by a weighted pooling head $f_{\text{TR}}^{\text{HEAD}}$ (fig. 4). Here, each 3D transformer module TE_l is a pairing of Transformer Encoder [65] layers $\text{TE}^0(Z)$ and $\text{TE}^1(Z)$:

$$\text{MHA}_l^d(Z_l) = Z'_l = \text{LN}(\text{MHA}_l(Z_l, \text{dim}{=}d) + Z_l) \quad (2)$$
$$\text{TE}_l^d(Z_l) = \text{LN}(\text{MLP}_l(Z'_l) + Z'_l) \quad (3)$$
$$\text{TE}_l(Z_l) = \text{TE}_l^0(\text{TE}_l^1(Z_l)) = Z_{l+1}, \quad (4)$$

$\text{MHA}(Z, \text{dim}{=}d)$ is a multi-head attention layer [65] whose attention vectors span the d-th dimension of the input tensor Z, MLP is a two-layer MLP with ReLU activation, and LN is Layer Normalization [3]. Intuitively, the alternation between ray and pooling attention of $\text{TE}^0(Z)$ and $\text{TE}^1(Z)$ facilitates learning to jointly aggregate WCE features from the source views and ray-march over them respectively.

Finally, f'_{TR} is terminated by a weighted pooling head $f_{\text{TR}}^{\text{HEAD}}(Z_L)$ that aggregates the second dimension of Z_L output by the final L-th 3D transformer module TE_L: $f_{\text{TR}}^{\text{HEAD}}(Z_L) = \sum_{i=1}^{n_{\text{src}}} \omega_i(Z_L) Z_L[:, i, :] \in \mathbb{R}^{N_S \times D_{\mathbf{z}}}$, where the weights $\omega_i \in [0, 1], \sum_i \omega_i = 1$ are output by a linear layer with softmax activation. We show ω_i in fig. 5.

method	PSNR	LPIPS	ℓ_1^{depth}	IoU	method	PSNR	LPIPS	ℓ_1^{depth}	IoU
NerFormer	23.3	0.17	0.40	**0.96**	SRN[56]	20.4	0.21	0.60	0.93
NeRF+WCE[26]	21.0	0.19	0.74	0.91	SRN+WCE+γ	16.9	0.30	0.60	0.75
NeRF[43]	**23.6**	0.17	**0.38**	0.95	SRN+WCE	15.8	0.26	0.64	0.80
NV[41]	22.2	0.20	0.91	0.91	SRN+γ	16.9	0.30	0.59	0.75
NV+WCE	18.7	0.25	0.85	0.90	DVR[45]	15.0	0.33	0.89	0.68
IDR[74]	18.5	**0.15**	0.81	0.92	DVR+γ	15.8	0.38	0.74	0.65
IPC+WCE	13.9	0.25	1.59	0.83	P3DMesh[50]	17.3	0.22	0.69	0.91
IPC	13.8	0.25	1.58	0.83					

Table 2: **Single-scene new-view synthesis results on Common Objects in 3D** comparing the baseline approaches [43, 41, 74, 56, 45, 67], IPC, their variants with Warp-conditioned Embeddings (+**WCE**) or Positional Embedding (+γ), and our NerFormer (the **best** / 2nd best result).

The final opacity and coloring functions of NerFormer are thus $f_o = f_{\text{HEAD}} \circ f'_{\text{TR}}$, $c_o = c_{\text{HEAD}} \circ f'_{\text{TR}}$ respectively, which are rendered with the EA raymarcher (sec. 4.1).

Technical details Training minimizes, with Adam (learning rate $5 \cdot 10^{-4}$), a sum of the RGB MSE error $\| I^{\text{tgt}} - \hat{I}^{\text{tgt}} \|^2$ and the binary cross entropy between the rendered alpha mask $\hat{M}^{\text{tgt}}$ and the ground truth mask M^{tgt}. We iterate over batches comprising a randomly sampled target view and 1 to 9 source views of a training video until convergence.

5. Experiments

Datasets In order to evaluate and train reconstruction methods on our dataset, we split the 18,619 collected CO3D videos into 4 different sets as follows. For each of the 50 categories, we split its videos to a `train` and a `test` set in 9:1 ratio. For each video, we further define a set of frames that are removed from the training set by randomly dividing each `train` video in a 8:2 ratio to 80 `train-known` training and 20 `train-unseen` holdout frames. `test` video frames are split according to the same protocol resulting in `test-known` and `test-unseen` sets. As all base-

	(a) Average statistics								(b) PSNR @ # source views										(c) PSNR @ target view difficulty					
	train-unseen				test-unseen				train-unseen					test-unseen					train-unseen			test-unseen		
Method	PSNR	LPIPS	ℓ_1^{depth}	IoU	PSNR	LPIPS	ℓ_1^{depth}	IoU	9	7	5	3	1	9	7	5	3	1	easy	med.	hard	easy	med.	hard
NerFormer	**17.9**	0.26	0.87	0.82	**17.6**	0.27	0.91	0.81	**19.3**	**19.0**	**18.3**	17.4	15.6	**18.9**	**18.6**	**18.1**	**17.1**	**15.1**	**18.9**	15.5	14.6	**18.6**	**14.9**	14.7
SRN+WCE+γ	17.6	0.24	**0.28**	0.89	14.4	0.27	0.40	0.81	18.0	18.0	17.8	**17.6**	**16.8**	14.6	14.5	14.6	14.5	13.9	18.0	**16.8**	**16.0**	14.7	13.6	**15.1**
SRN+WCE	16.6	0.26	0.31	0.87	14.6	0.27	**0.36**	**0.82**	17.0	17.0	16.7	16.4	15.8	14.9	14.8	14.8	14.6	13.9	16.9	15.7	14.5	14.9	13.7	14.8
NeRF+WCE[26]	14.3	0.27	2.14	0.72	13.8	0.27	2.23	0.70	14.3	15.0	14.9	14.7	14.2	12.6	14.5	14.4	14.2	13.8	12.1	13.6	13.6	14.4	13.0	13.0
IPC+WCE	14.1	0.36	2.12	0.70	13.5	0.37	2.24	0.69	14.4	14.4	14.2	14.1	13.4	13.8	13.8	13.7	13.6	12.6	14.4	13.7	13.4	13.8	12.8	12.2
P3DMesh	17.2	**0.23**	0.50	**0.91**	12.4	**0.26**	2.49	0.69	17.6	17.5	17.4	17.1	16.2	12.6	12.5	12.5	12.5	12.1	17.5	**16.8**	**16.0**	12.6	11.8	13.2
NV+WCE	12.3	0.34	2.87	0.54	11.6	0.35	3.01	0.53	12.5	12.5	12.3	12.2	12.0	11.7	11.6	11.6	11.6	11.3	12.4	12.0	13.6	11.7	11.2	12.0
SRN+γ+AD	**21.7**	**0.21**	0.31	0.89	-	-	-	-	**21.7**	**21.7**	**21.7**	**21.7**	**21.8**	-	-	-	-	-	**21.7**	**21.3**	**19.7**	-	-	-
SRN[56]+AD	21.2	**0.21**	**0.23**	**0.94**	-	-	-	-	21.2	21.2	21.2	21.2	21.3	-	-	-	-	-	21.2	20.8	19.2	-	-	-
NV[41]+AD	19.7	0.23	0.41	0.93	-	-	-	-	19.7	19.7	19.6	19.6	19.7	-	-	-	-	-	19.7	19.3	17.7	-	-	-
NeRF+AD	17.1	0.25	0.55	0.92	-	-	-	-	17.1	17.0	17.0	17.0	17.1	-	-	-	-	-	17.1	16.9	15.8	-	-	-
P3DMesh[50]+AD	16.1	0.24	0.74	0.89	-	-	-	-	16.1	16.1	16.1	16.1	16.2	-	-	-	-	-	16.1	16.0	15.1	-	-	-
IPC+AD	14.0	0.37	2.20	0.69	-	-	-	-	14.4	14.3	14.1	14.0	13.3	-	-	-	-	-	14.3	13.6	13.2	-	-	-
IDR+AD	13.7	0.25	1.74	0.88	-	-	-	-	13.7	13.7	13.7	13.7	13.8	-	-	-	-	-	13.7	13.7	14.0	-	-	-
DVR+γ+AD	7.6	0.29	3.24	0.09	-	-	-	-	7.6	7.6	7.5	7.5	7.7	-	-	-	-	-	7.6	7.7	9.0	-	-	-
DVR[45]+AD	7.6	0.29	3.01	0.06	-	-	-	-	7.6	7.6	7.5	7.5	7.7	-	-	-	-	-	7.6	7.7	9.0	-	-	-

Table 3: **Category-centric new-view synthesis results on Common Objects in 3D** comparing mean performance over 10 object categories. Reported are: (a) the main metrics averaged over the `train-unseen`/`test-unseen` set; (b) PSNR averaged over test samples with a specific number of source views and; (c) the samples within one of three viewpoint difficulty bins. Methods with +AD are autodecoders and thus not applicable to the `test-unseen` set. (**best** / 2nd best)

lines require a bounded scene coordinate frame, we zero-center each point cloud and normalize the scale by dividing with an average over per-axis standard deviations.

Evaluation protocol All evaluations focus on new RGB+D-view synthesis where a method, given a known target camera P^{tgt} and a set of source views $\{I_i^{\text{src}}\}_{i=1}^{n_{\text{src}}}$, renders new target RGB view $\hat{I}^{\text{tgt}}$ and depth $\hat{D}^{\text{tgt}}$. Four metrics are reported: The peak-signal-to-noise ratio (PSNR) and the LPIPS distance [76] between $\hat{I}^{\text{tgt}}$ and the ground truth target RGB frame I^{tgt}, the Jaccard index between the rendered/g.t. object mask $\hat{M}^{\text{tgt}}/M^{\text{tgt}}$ and the ℓ_1 distance ℓ_1^{depth} between the rendered depth $\hat{D}^{\text{tgt}}$ and the ground truth point cloud depth render D^{tgt}. Note that PSNR and ℓ_1^{depth} are only aggregated over the foreground pixels $\{\mathbf{u}_{\text{fg}}|M^{\text{tgt}}[\mathbf{u}_{\text{fg}}] = 1\}$. Metrics are evaluated at an image resolution of 800x800 and 400x400 pixels for single-scene (sec. 5.2) and category-specific reconstruction (sec. 5.3) respectively.

5.1. Evaluated methods

We implemented a total of 15 approaches that leverage the majority of common shape representation types.

Implicit surfaces Among implicit surface estimators we selected the seminal opacity-based NeRF; and IDR [74] and DVR [45] expressing shapes as a SDF f_d and render with sphere-tracing. We further benchmark SRN [56] which implements an implicit learned LSTM renderer. Importantly, following secs. 4.1 and 4.2, we evaluate the modifications SRN-γ, DVR-γ that endow SRN and DVR respectively with positional embedding γ. Furthermore, SRN-γ-WCE, SRN-WCE, NeRF-WCE [26] complement SRN and NeRF with the Warp-conditioned Embedding [26]. Finally, we also evaluate our NerFormer method.

Voxel grids Neural Volumes (NV) [41] represents the State of the Art among voxel grid predictors. Similar to implicit methods, we also combine NV with WCE.

Point clouds In order to compare with a point cloud-based method, we devised an Implicit Point Cloud (IPC) baseline which represents shapes with a colored set of 3D points, converts the set into an implicit surface and then renders it with the EA raymarcher. We note that IPC is strongly inspired by SynSin [50, 69] (see supplementary).

Meshes We benchmark P3DMesh - the best-performing variant of PyTorch3D's soft mesh rasterizer from [50] inspired by Pixel2Mesh [67], which deforms an initial spherical mesh template with a fixed topology with a series of convolutions on the mesh graph.

5.2. Single-scene reconstruction

We first task the approaches to independently reconstruct individual object videos. More specifically, given a `test` video, every baseline is trained to reproduce the video's `test-known` frames (by minimizing method-specific loss functions such as ℓ_2 RGB error), and evaluated by comparing the renders from the given `test-known` camera viewpoints to the corresponding ground truth images/depths/masks. Since training on all ~2k `test` videos is prohibitively expensive (each baseline trains at least for 24 hours on a single GPU), we test on 40 randomly selected `test` videos (two from each of 20 random classes). Quantitative / qualitative results are in fig. 6 / Tab. 2.

NerFormer is either the best or the second best across all metrics. NeRF+WCE is beaten by vanilla NeRF which suggests that the noisy WCE embeddings can hurt performance without NerFormer 's spatial reasoning. Interestingly, IDR's realistic, but less detailed renders win in terms of LPIPS, but are inferior in PSNR. Furthermore, we observed a large

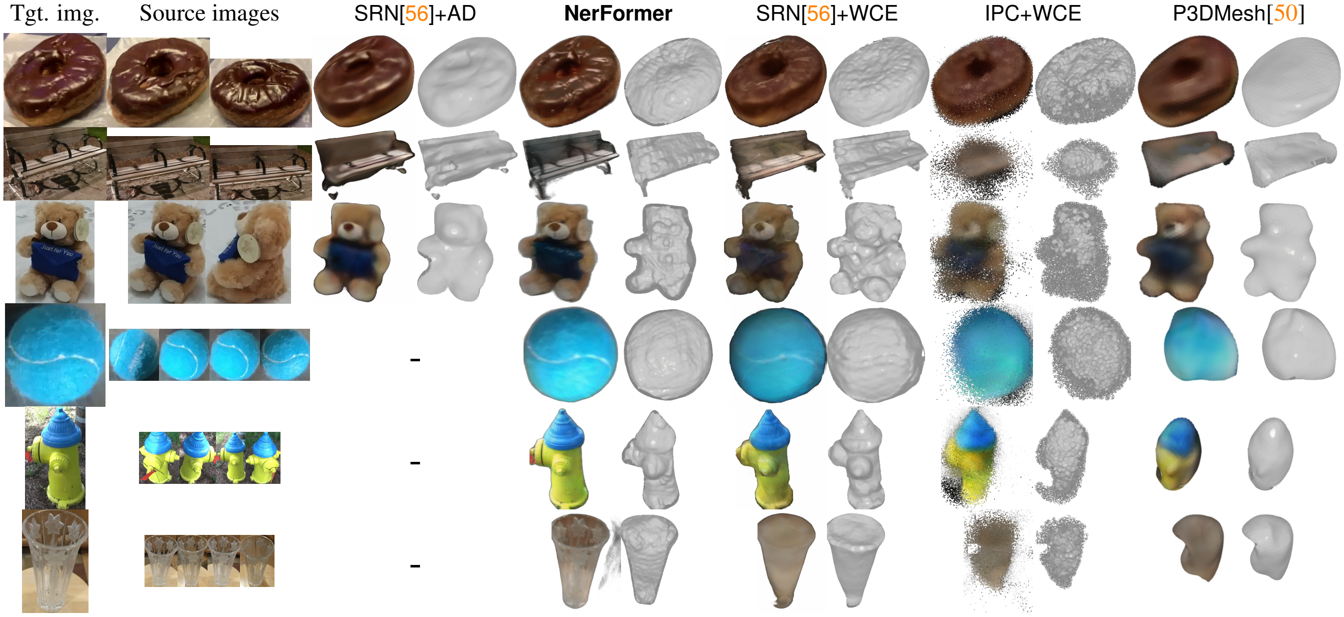

Figure 7: **Category-centric 3D reconstruction on Common Objects in 3D** depicting a target image, known source images $\{I_i^{\text{src}}\}$, and a synthesized new view. The first/last 3 rows are from the `train`/`test` set (SRN+AD is not applicable to `test`).

discrepancy between the train and test PSNR of γ/WCE-endowed SRN. This shows that, for the single-scene setting, increasing the model expressivity with WCE or γ can lead to overfitting to the training views.

5.3. Learning 3D Object Categories

Our main task is learning category-centric 3D models. In more detail, a single model is trained on all `train-known` frames of an object category and evaluated on 1000 randomly generated test samples from `train-unseen` and `test-unseen` sets of the category. Each train or test sample is composed of a single target image I^{tgt} and randomly selected source views $\{I^{\text{src}}\}_i^{n_{\text{src}}}$, where n_{src} is randomly picked from $\{1, 3, 5, 7, 9\}$. Since training a model for each of the 50 object categories is prohibitively expensive (each method takes at least 7 days to train), we chose a subset of 10 categories for evaluation.

Baselines For each method we evaluate the ability to represent the shape space of training object instances by turning it into an *autodecoder* [4] which, in an encoder-less manner, learns a separate latent embedding $\mathbf{z}_{\text{scene}}(\text{sequence}_{\text{ID}})$ for each `train` scene as a free training parameter. Since autodecoders (abbreviated with the **+AD** suffix) only represent the `train` set, we further compare to all WCE-based methods which can additionally reconstruct `test` videos. Note that, as remarked in [26] and sec. 4.2, the alternative encoding $\mathbf{z}_{\text{global}} = \Phi_{\text{CNN}}(I^{\text{src}})$ is deemed to fail due to the world coordinate ambiguity of the training SfM reconstructions.

Results Quantitative and qualitative comparisons are shown in tab. 3 and fig. 7 respectively. Besides average metrics over both test sets, we also analyze the dependence on the number of available source views, and on the difficulty of the target view. For the latter, test frames are annotated with a measure of their distance to the set of available source views, and then split into 3 different difficulty bins. Average per-bin PSNR is reported. The supplementary details the distance metric and difficulty bins.

While SRN+AD has the best performance across all metrics on `train-unseen`; on the `test-unseen` set, where SRN+AD is inapplicable, our NerFormer is the best for most color metrics. Among implicit methods, the SDF-based DVR and IDR are outperformed by the opacity-based NeRF, with both DVR+WCE and IDR+WCE failing to converge. This is likely because regressing SDFs is more challenging than classifying 3D space with binary labels opaque/transparent. Finally, we observed poor performance of P3DMesh, probably due to the inability of meshes to represent complicated real geometries and textures.

6. Conclusion

We have introduced Common Objects in 3D (CO3D), a dataset of in-the-wild object-centric videos capturing 50 object categories with camera and point cloud annotations.

We further contributed NerFormer which is a marriage between Transformer and neural implicit rendering that can reconstruct 3D object categories from CO3D with better accuracy than a total of 14 other tested baselines.

The CO3D collection effort still continues at a steady pace of ~500 videos per week which we plan to release in the near future.

Common Objects in 3D:
Large-Scale Learning and Evaluation of Real-life 3D Category Reconstruction

Supplementary material

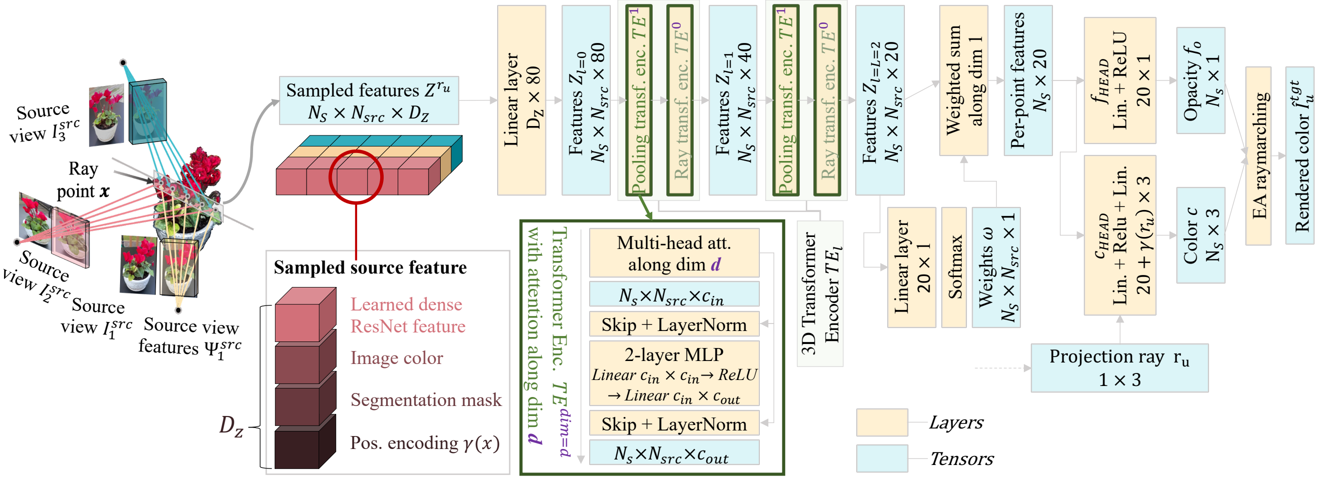
Source view I_3^{src}
Ray point $\boldsymbol{x}$
Source view I_2^{src}
Source view I_1^{src}
Source view features Ψ_1^{src}
Sampled features Z^{r_u}
$N_S \times N_{src} \times D_Z$
Sampled source feature
Learned dense ResNet feature
Image color
Segmentation mask
Pos. encoding $\gamma(x)$
D_Z
Linear layer $D_Z \times 80$
Features $Z_{l=0}$ $N_S \times N_{src} \times 80$
Pooling transf. enc. TE^1
Ray transf. enc. TE^0
Features $Z_{l=1}$ $N_S \times N_{src} \times 40$
Pooling transf. enc. TE^1
Ray transf. enc. TE^0
Features $Z_{l=L=2}$ $N_S \times N_{src} \times 20$
Transformer Enc. $TE^{dim=d}$ with attention along dim $\boldsymbol{d}$
Multi-head att. along dim $\boldsymbol{d}$
$N_S \times N_{src} \times c_{in}$
Skip + LayerNorm
2-layer MLP
$Linear\ c_{in} \times c_{in} \rightarrow ReLU \rightarrow Linear\ c_{in} \times c_{out}$
Skip + LayerNorm
$N_S \times N_{src} \times c_{out}$
3D Transformer Encoder TE_l
Weighted sum along dim 1
Linear layer 20×1
Softmax
Weights ω $N_S \times N_{src} \times 1$
Per-point features $N_S \times 20$
f_{HEAD} Lin. + ReLU 20×1
c_{HEAD} Lin. + Relu + Lin. $20 + \gamma(r_u) \times 3$
Opacity f_o $N_S \times 1$
Color c $N_S \times 3$
EA raymarching
Rendered color I_u^{tgt}
Projection ray r_u 1×3
Layers
Tensors

Figure I: A detailed illustration of the architecture of NerFormer .

In what follows, we provide additional quantitative results (sec. A), technical details of NerFormer and the baselines (sec. B), and details of the Human-in-the-loop 3D annotation process (sec. C).

A. Additional results

A.1. Results on all 50 categories

While tab. 3 in the main paper provides results on a subset of 10 object categories for all baselines, for completeness, in tab. I, we provide evaluation on all 50 object classes for 4 best-performing baselines according to results reported in tab. 3: NerFormer, SRN+WCE, SRN+γ+WCE, and NeRF+WCE.

Similar to tab. 3, on the `test-unseen` set, NerFormer is the best in all color-based metrics, suggesting that SRN and NeRF+WCE are prone to overfitting to the training scenes. While SRN outperforms NerFormer in some cases on `train-unseen`, we note that the autodecoders would likely yield superior performance on `train-unseen` due to their ability to capture the information from all views of a training scene in the latent scene-specific encoding.

A.2. Convergence speed

Fig. II further analyzes training convergence on the single-scene new-view synthesis task. For each method and epoch, we plot the average and standard deviation over of the per-epoch mean PSNRs of each of the 40 test scenes. The fastest converging methods are SRN, IDR, NerFormer , NeRF which also top the performance in tab. 2, indicating a significant correlation between convergence rate and performance. Furthermore, there is a large discrepancy between the train and test PSNR of γ/WCE-endowed SRN. This shows that, for the single-scene setting, increasing the model expressivity with WCE or γ can lead to overfitting to the training views.

A.3. Execution speed

Tab. II contains an evaluation of execution times for all methods from tab. 2. Here, each row reports an average time to render an 800x800 pixel image on NVIDIA Tesla V100 GPU.

A.4. Test-time autodecoder optimization

In tab. III, we provide an extension of tab. 3 containing the evaluation of the best-performing autodecoding methods at test-time. First, each method is first trained on `train-known`. Then, during evaluation, the latter freezes the trained weights and optimizes the input latent code for a given set of source frames from a test sequence. The latent codes are optimized with Adam until convergence, decaying the learning rate 10-fold whenever the optimization

	(a) Average statistics								(b) PSNR @ # source views										(c) PSNR @ target view difficulty					
	train-unseen				test-unseen				train-unseen					test-unseen					train-unseen			test-unseen		
Method	PSNR	LPIPS	ℓ_1^{depth}	IoU	PSNR	LPIPS	ℓ_1^{depth}	IoU	9	7	5	3	1	9	7	5	3	1	easy	med.	hard	easy	med.	hard
NerFormer	16.5	**0.24**	3.67	0.76	**15.7**	**0.24**	1.82	0.75	**17.5**	17.3	16.9	16.3	14.8	**16.7**	**16.4**	**16.1**	**15.5**	**13.9**	17.3	14.7	12.8	**16.5**	**13.7**	11.2
SRN+WCE	16.3	0.25	0.37	**0.81**	14.2	0.27	**0.47**	**0.77**	16.6	16.6	16.5	16.2	15.6	14.4	14.3	14.3	14.2	13.5	16.6	15.5	12.9	14.4	13.4	**11.4**
SRN+WCE+γ	**17.1**	0.25	**0.35**	**0.81**	13.7	0.28	**0.47**	0.73	17.4	**17.4**	**17.3**	**17.0**	**16.3**	14.0	13.8	13.9	13.7	13.2	**17.4**	**16.3**	**14.4**	14.0	13.1	10.6
NeRF+WCE[26]	12.6	0.27	6.21	0.54	11.6	0.27	4.54	0.51	13.0	13.0	12.8	12.6	11.6	11.9	11.8	11.8	11.6	10.8	12.9	12.1	9.4	11.9	11.1	8.7

Table I: **Results on all 50 classes from CO3D** comparing the 4 best-performing methods from tab. 3.

method	time [sec]	method	time [sec]
NerFormer	178.41	SRN[56]	1.00
NeRF+WCE[26]	113.82	SRN+γ	1.19
NeRF[43]	23.82	SRN+WCE+γ	4.20
NV[41]	0.37	SRN+WCE	5.34
NV+WCE	0.41	DVR[45]	196.94
IDR[74]	69.11	DVR+γ	204.39
IPC	0.15	P3DMesh[50]	0.09
IPC+WCE	0.16		

Table II: **Average rendering time** of an 800x800 pixel image comparing all methods from tab. 2.

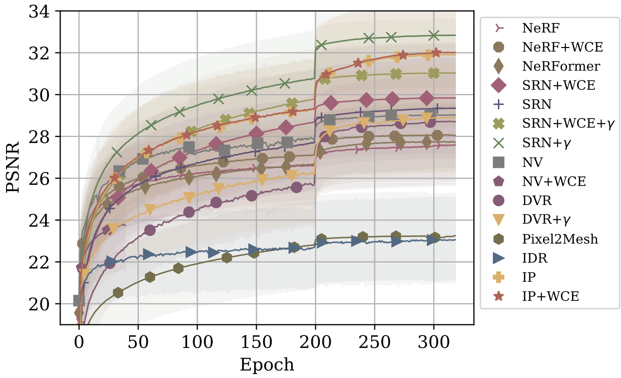

Figure II: **Convergence speed on single-scene new-view-synthesis** showing the mean and std. dev. over the per-scene training PSNRs for each method.

objective plateaus.

We observed that the latent code optimization was mostly failing for NeRF and NV. On the other hand, SRN gave slightly better performance, which we attribute to the higher smoothness of the implicit function compared to the NeRF and NV (SRN contains normalization layers while the other baselines are bare MLPs interleaving linear layers and ReLUs).

A.5. Estimating new-view difficulty

Tab. 3 contains metrics evaluated separately for three viewpoint difficulty bins. Here, we detail the process of estimating the difficulty of a testing target view.

Camera difficulty $\mathcal{D}$ Given a target camera P^{tgt} and a set of available source views $\{P_i^{\text{src}}\}_{i=1}^{N_{\text{src}}}$, the difficulty of the target view $\mathcal{D}(P^{\text{tgt}}) \in [0,1]$ is quantified as the average of the two lowest distances $d(P^{\text{tgt}}, P_i^{\text{src}})$ between the target view and each of the source views.

Camera distance d^{cam} The distance $d^{\text{cam}}(P_i, P_j) \in [0,1]$ between two cameras P_i and P_j is defined as follows. We first generate a cubical voxel grid of size 32^3 in the center of the scene with the voxel size set such that the majority of the grid is observed by all cameras in the scene. Each point $\mathbf{x}_k$, denoting the coordinates of the center of a cell in the voxel grid, is then projected to both cameras leading to a pair of projection rays $\mathbf{r}_k^i, \mathbf{r}_k^j \in \mathbb{S}^2$. We then define the similarity $s(\mathbf{r}_k^i, \mathbf{r}_k^j) = \delta[\pi_{P_i}(\mathbf{x}_k) \in \Omega_i \wedge \pi_{P_j}(\mathbf{x}_k) \in \Omega_j](1 + \mathbf{r}_k^i \cdot \mathbf{r}_k^j)$ as a dot product between the pair of rays weighted by an indicator that checks whether the projection of $\mathbf{x}_k$ simultaneously lands in the rasters $\Omega_i, \Omega_j \in [0,W] \times [0,H]$ of both cameras P_i and P_j. The camera distance d^{cam} is then defined as one minus the intersection-over-union of the similarities between all pairs of rays generated by each voxel grid point $\mathbf{x}_k$:

$$d^{\text{cam}}(P_i, P_j) = 1 - \frac{\sum_k s(\mathbf{r}_k^i, \mathbf{r}_k^j)}{\sum_k s(\mathbf{r}_k^i, \mathbf{r}_k^i) + s(\mathbf{r}_k^j, \mathbf{r}_k^j) - s(\mathbf{r}_k^i, \mathbf{r}_k^j)}.$$

Intuitively, the camera distance is proportional to the angle between the camera heading vectors adjusted by the overlap between the voxels observed by both cameras. However, merely considering the heading vectors would not take into account the intrinsics of the cameras (focal length / principal point). We thus devised d^{cam} which leverages angles between projection rays, which are a function of both the intrinsics and extrinsics.

In order to understand d^{cam}, consider the following two examples: Two cameras observing the same set of voxels at a relative angle of 0.5π would have $d^{\text{cam}}(P_i, P_j) \approx \frac{2}{3}$, while opposite-facing cameras would yield a maximum possible $d^{\text{cam}}(P_i, P_j) \approx 1$.

Camera difficulty bins Each testing target camera P^{tgt} is then assigned into one of 3 difficulty bins (*easy, medium, hard*) depending on its difficulty measure $\mathcal{D}(P^{\text{tgt}})$. More specifically, the easy cameras satisfy $0 \leq \mathcal{D}(P^{\text{tgt}}) < \frac{1}{6}$, medium $\frac{1}{6} \leq \mathcal{D}(P^{\text{tgt}}) < \frac{1}{3}$, and hard $\mathcal{D}(P^{\text{tgt}}) \geq \frac{1}{3}$.

	(a) Average statistics				(b) PSNR @ # src. views					(c) PSNR @ tgt. difficulty		
	`test-unseen`				`test-unseen`					`test-unseen`		
Method	PSNR	LPIPS	ℓ_1^{depth}	IoU	9	7	5	3	1	easy	medium	hard
NerFormer	**17.6**	**0.27**	0.91	**0.81**	**18.9**	**18.6**	**18.1**	**17.1**	**15.1**	**18.6**	**14.9**	**14.7**
SRN+γ+AD	13.2	0.29	0.48	0.71	13.6	13.5	13.3	13.1	12.4	13.5	11.6	11.8
SRN[56]+AD	13.8	0.28	**0.45**	0.74	14.3	14.3	14.0	13.6	12.6	14.2	12.5	11.1
NV[41]+AD	11.4	0.53	1.29	0.47	11.5	11.2	11.3	11.5	11.5	11.4	11.3	8.0
NeRF+AD	10.6	0.32	4.42	0.49	10.7	10.5	10.4	10.7	10.4	10.7	10.3	3.6

Table III: **Autodecoder latent optimization on `test-unseen`** extending the results in tab. 3. Each method labelled with +AD is first trained on `train-known`. During evaluation the latter fixes the trained weights and optimizes the input latent code for a given set of source frames from a test sequence. For context, we also compare to NerFormer , which is not an autodecoder.

B. Additional technical details

In this section we provide additional details of NerFormer and of the benchmarked baseline approaches which were outlined in sec. 5.

B.1. NerFormer

Source image features Ψ The dense pixel-wise descriptor $\Psi_{\text{CNN}}(I^{\text{src}})$ (sec. 4.2) of a source image I^{src} is a stacking of 3 types of feature tensors along the channel dimension after differentiably upsampling to a common spatial resolution $H \times W$. The feature types are: 1) Intermediate activations extracted from the source image I^{src} after "layer1", "layer2", and "layer3" layers of the ResNet34 [23] network, 2) the source segmentation mask M^{src}, 3) the raw source image I^{src}. Note that we separately map the output of each of the ResNet34 layers to a 32-dimensional feature with a 1x1 convolution followed by ℓ_2 normalization of the feature column at every spatial location.

NerFormer architecture In fig. I we provide a more detailed visualisation of the NerFormer architecture.

Rendering details Similar to NeRF [43], NerFormer optimizes loss functions for 800 randomly sampled image rays $\mathbf{r_u}$ in each training target image. Following [43], NerFormer implements a coarse and fine rendering network f_{TR}. The former, given a ray $\mathbf{r_u}$, samples 32 points $\mathbf{x}_i \in \mathbf{r_u}$ at uniform depth intervals between predefined lower and upper depth bounds. The fine rendering network then samples 16 points on $\mathbf{r_u}$ with importance sampling from the distribution proportional to the coarse rendering weights w_i.

Training details For a randomly sampled pixel $\mathbf{u}$ we thus render the color $\hat{I}^{\text{tgt}}_{\mathbf{u}}$ and an alpha value $\hat{M}^{\text{tgt}}_{\mathbf{u}} = 1 - \prod_i (1 - \exp(-\Delta f_o(\mathbf{x}, \mathbf{z}))) \in [0, 1]$, where the latter denotes the total amount of light absorbed by the implicit surface ($\hat{M}^{\text{tgt}}_{\mathbf{u}} = 1$ for complete absorption).

As noted in sec. 4.3, the optimized loss is a sum of the RGB squared error $\sum_{\mathbf{u}} \|\hat{I}^{\text{tgt}}_{\mathbf{u}} - I^{\text{tgt}}_{\mathbf{u}}\|^2$ and the segmentation binary cross entropy (BCE) $\sum_{\mathbf{u}} M^{\text{tgt}}_{\mathbf{u}} \log \hat{M}^{\text{tgt}}_{\mathbf{u}} + (1 - M^{\text{tgt}}_{\mathbf{u}}) \log(1 - \hat{M}^{\text{tgt}}_{\mathbf{u}})$. The latter ensures that rays that do not intersect the object of interest do not terminate in the scene and vice versa. Following [43], we evaluate the losses for the fine and coarse renders and optimize their sum.

B.2. NeRF

We use the implementation of NeRF [43] from PyTorch3D [50] which closely follows the original paper. Similar to NerFormer , we also add to the original losses of NeRF the BCE loss between the rendered alpha mask and the ground truth target mask. The coloring function c_{MLP} and the opacity function f_{MLP} have their architecture identical to the original implementation.

B.3. SDF methods - DVR, IDR

Here we detail the two baseline methods that represent shapes with signed distance fields (SDF). We start with introducing the SDF and a method for their rendering.

Signed distance fields While opacity functions f_o represent shapes with a measure of opaqueness of 3D spatial elements, *signed distance fields* $f_d(\mathbf{x}, \mathbf{z}) \in \mathbb{R}$, express the signed euclidean distance to the nearest point $\mathbf{x}' \in \mathcal{S}_f$ on the implicit surface $\mathcal{S}_f$.

Sphere tracing (ST) While EA is the most popular method for rendering opacity fields f_o, ST is its analogue for signed distance fields f_d. Specifically, ST renders a pixel $\mathbf{u}$ by seeking the minimum of the signed distance function f_d on the domain of 3D points belonging to the ray $\mathbf{r_u}$. ST, during its t-th iteration, refines the current estimate $\mathbf{x}^{\mathbf{r_u}}_t$ of the ray-surface intersection by moving $\Delta_t = f_d(\mathbf{x}^{\mathbf{r_u}}_t, \mathbf{z})$ units in the direction of the projection ray: $\mathbf{x}^{\mathbf{r_u}}_{t+1} = \mathbf{x}_t + \Delta_t \mathbf{r_u}$. Upon convergence at time T, the rendered color $\hat{I}^{\text{tgt}}_u = c(\mathbf{x}^{\mathbf{r_u}}_T, \mathbf{r_u}, \mathbf{z})$ comprises the response of the coloring function at the estimated ray-surface intersection.

DVR natively supports our supervisory scenario and hence no alternations were required for the training protocol of both DVR+AD and DVR. In order to implement DVR+γ, we simply convert the input coordinates to positional embeddings and adjust the number of input channels of the first layer of DVR's implicit function accordingly. The released code [45] supports DVR+AD so no changes were required here. As mentioned in the paper, unfortunately, all our attempts to merge DVR with WCE lead to a non-converging model.

IDR Similar to DVR, IDR [74] supports our supervisory setup by default. In order to implement IDR+AD, we append the latent code $\mathbf{z}$ to the positional embeddings that are input to the implicit function, and we adjust the number of input channels of the first layer of the implicit function accordingly. IDR already takes as input the positional embeddings γ, so the extension IDR+γ does not apply here. As mentioned in the main paper, we could not obtain a converging version of IDR+WCE.

B.4. SRN

Here, we first give a brief overview of the learned SRN raymarcher, followed by describing the WCE extension of SRN.

Neural raymarching Contrasted to the explicit formulations of sphere-tracing or EA, recently, SRN [56] proposed to learn to march along the projection rays with a recurrent deep network. Similar to sphere-tracing, SRN decides at iteration t on the length of the raymarching step Δ_t by evaluating a function at the current intersection estimate $\mathbf{x}_t^{\mathbf{r}_u}$. However, instead of querying the SDF, SRN utilizes an LSTM [27] cell $f_{\text{LSTM}}(\mathbf{x}_t^{\mathbf{r_u}}, \mathbf{r_u}, \mathbf{z}, \mathbf{h}_t) = (\Delta_t, \mathbf{h}_{t+1})$ which is additionally conditioned on the ray direction $\mathbf{r_u}$ and a temporal hidden state $\mathbf{h}_t$. In this manner, the raymarcher adapts the step-size prediction based on the past marching observations.

SRN+WCE The WCE exension of SRN is straightforwardly implemented by replacing the iterative invocation of the global-encoding-conditioned implicit $f_{\text{LSTM}}(\mathbf{x}_t^{\mathbf{r_u}}, \mathbf{r_u}, \mathbf{z}_{\text{global}}, \mathbf{h}_t)$ of the SRN's raymarcher with the WCE-conditioned implicit $f_{\text{LSTM}}(\mathbf{x}_t^{\mathbf{r_u}}, \mathbf{r_u}, \mathbf{z}^{\star}_{\text{WCE}}(\mathbf{x}_t^{\mathbf{r_u}}, \{I_i^{\text{src}}\}, \{P_i^{\text{src}}\}), \mathbf{h}_t)$ This way, the learned raymarcher can "tap"" into the source views during every iteration to receive a more direct triangulation signal. As apparent from tabs. 3 and I, our WCE extension of SRN provides a very strong baseline that in fact achieves the best depth prediction performance.

Mask prediction The learned raymarcher of the original version of SRN does not render an alpha mask of the foreground object. In order to enable the latter, we extend the last layer of the SRN's coloring function c with an additional channel that is terminated with a sigmoid activation and represents the alpha value of the corresponding pixel $\mathbf{u}$. This channel is then supervised by minimizing the DICE coefficient between its output and the ground truth segmentation masks M^{tgt}.

B.5. P3DMesh

As mentioned in the paper, P3DMesh [50] deforms an initial spherical mesh template with a fixed topology with a series of convolutions on the mesh graph. As in [67], the graph convolutions accept features sampled from the source images at the 2D projections of the mesh vertices. Since P3DMesh supports conditioning only on a single-source-view, we extend to the multi-view setting by averaging over the per-vertex features sampled from each of the source views. Furthermore, note that the implementation in [50] differentiably renders the mesh with a memory-efficient version of the Soft Rasterizer [40]. The training protocol, including the employed losses and their weighting, closely follows [50].

B.6. Neural Volumes (NV)

Neural Volumes [41] is a method that represents implicit surfaces as voxel grids. In what follows, we first briefly describe voxel grids, their specific implementation in NV, and its extension with warp-conditioned embedding (NV+WCE).

Voxel grids While MLPs can label an arbitrary element of the 3D domain, a voxel grid can be seen as an implicit surface restricted to a subset of $\mathbb{R}^3$ which is uniformly subdivided to a lattice $V(\mathbf{z}) \in \mathbb{R}^{R^3}$ of $R^3; R \in \mathbb{N}_+$ cuboid elements of the same size. Note that the lattice $V(\mathbf{z})$ is a function (typically a 3D deconvnet) of $\mathbf{z}$ which allows for representing different 3D shapes. The implicit function $f_{\text{voxel}}(\mathbf{x}, \mathbf{z}) = \zeta(V(\mathbf{z}), \mathbf{x})$ is then evaluated by sampling $V(\mathbf{z})$ at the corresponding world coordinate $\mathbf{x}$, with a grid-sampling function $\zeta : \mathbb{R}^{R^3} \times \mathbb{R}^3 \mapsto \mathbb{R}^{\dim(f)}$, such as trilinear interpolation. Voxel grids also admit coloring via a volume $C(\mathbf{z}) \in \mathbb{R}^{3 \times R^3}$ which can be sampled in an analogous manner.

Neural Volumes A notable voxel-grid-based method is Neural Volumes [41], which proposed an improved sampling function $\zeta_{\text{warp}}(\zeta(W(\mathbf{z}), \mathbf{x}) + \mathbf{x}, V(\mathbf{z}))$ which refines the sampling location $\mathbf{x}$ with an offset vector $\zeta(W(\mathbf{z}), \mathbf{x}) \in \mathbb{R}^3$ sampled from a warping lattice $W(\mathbf{z}) \in \mathbb{R}^{R^3}$. Here both W and V are implemented as a 3D deconvolutional network.

NV and NV+AD NV+AD is in fact the vanilla version of [41] whose 3D deconvnets V and W accept the scene-specific latent code $\mathbf{z}_{\text{scene}}(\text{sequence}_{\text{ID}})$ (described in sec. 5.3). The "overfitting" version of NV from tab. 2 is a special case of NV+AD with a single latent code.

NV+WCE The WCE extension of Neural Volumes (NV+WCE) appends the WCE to the feature of each voxel after the second 3D deconvolution layer of the 3D convnets V and W. Here, the WCE of a voxel is generated by expressing the world coordinate $\mathbf{x}_{V_i}$ of the center of the correspoding voxel V_i and calculating the aggregate WCE

$\mathbf{z}^\star_{\text{WCE}}(\mathbf{x}_{V_i}, \{I_i^{\text{src}}\}, \{P_i^{\text{src}}\})$ for a set of source views $\{I_i^{\text{src}}\}$ and their cameras $\{P_i^{\text{src}}\}$. Note that a similar approach has been proposed in [31].

Training All versions of NV optimize the losses from [41] with the original weights. Furthermore, we exploit the known ground truth segmentation masks and minimize the binary cross entropy between the alpha mask returned by the raymarcher of NV and the ground truth mask M^{tgt}.

B.7. Implicit Point Cloud (IPC)

As mentioned in sec. 5, IPC represents shapes by converting a point cloud to an implicit function which is later rendered with EA raymarching.

Formally, let a point cloud $\mathcal{P}(\mathbf{z}) = \{\mathbf{x}_i\}_{i=1}^{N_{\text{pts}}}$ be an N_{pts}-sized unordered set of points, where $\mathcal{P}$ is a point cloud predictor (detailed later in this section) which accepts the latent code $\mathbf{z}$. $\mathcal{P}(\mathbf{z})$ then admits an occupancy function $f_{\mathcal{P}\epsilon}$ defined as follows:

$$f_{\mathcal{P}\epsilon}(\mathbf{x}', \mathbf{z}) = \delta[\|\text{NN}_{\mathcal{P}(\mathbf{z})}(\mathbf{x}') - \mathbf{x}'\| < \epsilon],$$

where $\text{NN}_{\mathcal{P}(\mathbf{z})}(\mathbf{x}') = \arg\min_{\mathbf{x} \in \mathcal{P}(\mathbf{z})} \|\mathbf{x} - \mathbf{x}'\|$ returns the nearest point from the point cloud $\mathcal{P}(\mathbf{z})$ to the query point $\mathbf{x}'$. Intuitively, $f_{\mathcal{P}\epsilon}$ yields zero everywhere except within an ϵ neighborhood of each point cloud point $\mathbf{x}_i \in \mathcal{P}(\mathbf{z})$, where $f_{\mathcal{P}}$ yields 1. As we describe later, anchoring the implicit function on the set of cloud points allows for faster and more memory-efficient EA raymarching than in the case of the neural implicit occupancy f_{MLP} (described in sec. 4.1).

In order to color the implicit point cloud, we define its coloring function c_{IPC}:

$$c_{\text{IPC}}(\mathbf{x}', \mathbf{r}, \mathbf{z}) = c_{\text{MLP}}(\text{NN}_{\mathcal{P}(\mathbf{z})}(\mathbf{x}'), \mathbf{r}, \mathbf{z}).$$

Here c_{IPC} attaches to an arbitrary point $\mathbf{x}'$ the response of the coloring MLP c_{MLP} at $\mathbf{x}'$'s nearest point cloud neighbor $\text{NN}_{\mathcal{P}(\mathbf{z})}(\mathbf{x}')$.

Rendering IPC IPC is rendered efficiently with the PyTorch3D point cloud renderer [50, 69]. More specifically, given a target camera P^{tgt}, each point from the predicted point cloud $\mathcal{P}(\mathbf{z})$ is projected to the camera plane to form a set of 2D projections $\{\pi_{P^{\text{tgt}}}(\mathbf{x}_i) | \mathbf{x}_i \in \mathcal{P}(\mathbf{z})\}$. For each pixel coordinate $\mathbf{u} \in \{1, ..., W\} \times \{1, ..., H\}$ in the rendering lattice of the target render $\hat{I}^{\text{tgt}} \in \mathbb{R}^{3 \times H \times W}$, the renderer records the ordered set

$$\mathsf{\Pi}_\epsilon^{\mathbf{u}}(\mathcal{P}(\mathbf{z})) = \big(\mathbf{x}_i | \mathbf{x}_i \in \mathcal{P}(\mathbf{z}); \|\pi_{P^{\text{tgt}}}(\mathbf{x}_i) - \mathbf{u}\| \leq \epsilon \mathsf{f}_{P^{\text{tgt}}};$$
$$d_{P^{\text{tgt}}}(\mathbf{x}_i) \leq d_{P^{\text{tgt}}}(\mathbf{x}_{i+1})\big),$$

of point cloud points $\mathbf{x}_i \in \mathcal{P}(\mathbf{z})$ whose 2D projections $\pi_{P^{\text{tgt}}}(\mathbf{x}_i)$ land within the $\epsilon \mathsf{f}_{P^{\text{tgt}}}$ distance from the pixel $\mathbf{u}$, and which is ordered by the depth $d_{P^{\text{tgt}}}(\mathbf{x}_i)$ of each point in the target camera P^{tgt}. $\mathsf{f}_{P^{\text{tgt}}} \in \mathbb{R}$ is the focal length of the target camera P^{tgt}.

Intuitively, $\mathsf{\Pi}_\epsilon^{\mathbf{u}}(\mathcal{P}(\mathbf{z}))$ denotes the set of point cloud points whose ϵ neighborhoods are intersected by the rendering ray $\mathbf{r}_{\mathbf{u}}$ emitted from pixel $\mathbf{u}$. Note that this is an approximation: comparing the 2D camera-plane distance $\|\pi_{P^{\text{tgt}}}(\mathbf{x}_i) - \mathbf{u}\|$ to the constant $\epsilon \mathsf{f}_{P^{\text{tgt}}}$ corresponds to orthographic projections of the point neighborhoods, whereas our cameras are perspective. However, the orthographic approximation is mild in our case, since the distance of the point cloud points from the camera is relatively large compared to its focal length.

The EA raymarching then takes the set of $\mathbf{u}$'s 3D points $\mathsf{\Pi}_\epsilon^{\mathbf{u}}(\mathcal{P}(\mathbf{z}))$ in order to render the color $\hat{I}_{\mathbf{u}}^{\text{tgt}} \in \mathbb{R}^3$:

$$\hat{I}_{\mathbf{u}}^{\text{tgt}}(\mathbf{r}_{\mathbf{u}}, \mathbf{z}) = \sum_{\mathbf{x}_i \in \mathsf{\Pi}_\epsilon^{\mathbf{u}}(\mathcal{P}(\mathbf{z}))} w_i(\mathbf{x}_i, \mathbf{z}, \mathbf{u}) c_{\text{IPC}}(\mathbf{x}_i, \mathbf{r}_{\mathbf{u}}, \mathbf{z}).$$

For IPC, the weight $w_i(\mathbf{x}_i, \mathbf{z}, \mathbf{u}) = \left(\prod_{j=0}^{i-1} T_j^{\text{IPC}}(\mathbf{x}_i, \mathbf{z}, \mathbf{u})\right) \left(1 - T_i^{\text{IPC}}(\mathbf{x}_i, \mathbf{z}, \mathbf{u})\right)$ is the product of emission and absorption functions with the transmission term T_i^{IPC} defined as

$$T_i^{\text{IPC}}(\mathbf{x}_i, \mathbf{z}, \mathbf{u}) = \underbrace{f_{\mathcal{P}\epsilon}(\mathbf{x}_i, \mathbf{z})}_{=1} \frac{\|\mathbf{u} - \pi_{P^{\text{tgt}}}(\mathbf{x}_i)\|}{\epsilon \mathsf{f}_{P^{\text{tgt}}}},$$

which approximately measures the amount of light transmitted through the spherical ϵ neigborhood of a point $\mathbf{x}_i$ which intersects the projection ray $\mathbf{r}_{\mathbf{u}}$. To demonstrate this, observe that for a pixel $\mathbf{u}^{\text{intersect}} = \pi_{P^{\text{tgt}}}(\mathbf{x}_i)$ which coincides with the projection of the 3D point $\mathbf{x}_i$, the transmission $T_i^{\text{IPC}}(\mathbf{x}_i, \mathbf{z}, \mathbf{u}^{\text{intersect}}) = 0$, *i.e.* no light is transmitted through $\mathbf{x}_i$ and the corresponding color $c_{\text{IPC}}(\mathbf{x}_i, \mathbf{r}_{\mathbf{u}^{\text{intersect}}}, \mathbf{z})$ is fully rendered. On the contrary, for a pixel $\mathbf{u}^{\text{outside}} = \pi_{P^{\text{tgt}}}(\mathbf{x}_i + \epsilon)$ outside the epsilon neighborhood, the unit transmission $T_i^{\text{IPC}}(\mathbf{x}_i, \mathbf{z}, \mathbf{u}^{\text{outside}}) = 1$ signifies that all light passes through and the point's color is ignored during rendering. Note that the above equation is very similar to the top-k point cloud rasterizer of SinSyn [69].

Point cloud predictor $\mathcal{P}(\mathbf{z})$ The point cloud predictor $\mathcal{P}(\mathbf{z})$ is the same for both IPC+AD and IPC+WCE. More specifically, $\mathcal{P}(\mathbf{z}) = \{\bar{\mathbf{x}}_i + o_{\text{MLP}}(\bar{\mathbf{x}}_i, \mathbf{z})\}_{i=1}^{N_{\text{pts}}}$ offsets a fixed set of template points $\bar{\mathcal{P}} = \{\bar{\mathbf{x}}_i\}_{i=1}^{N_{\text{pts}}}$ with an offset function $o : \mathbb{R}^3 \times \mathbb{R}^{D_{\mathbf{z}}} \mapsto \mathbb{R}^3$ implemented as an MLP with the same architecture as f_{MLP}. Therefore, o alters the template point cloud to match a specific shape given its latent shape code $\mathbf{z}$.

IPC+AD and IPC+WCE For IPC+AD, the offset function o accepts the video-specific latent code $\mathbf{z}_{\text{scene}}(\text{sequence}_{\text{ID}})$ described in sec. 5.3, while for IPC+WCE, o takes as input the aggregate warp-conditioned embedding $\mathbf{z}^\star_{\text{WCE}}(\bar{\mathbf{x}}_i, \{I_i^{\text{src}}\}, \{P_i^{\text{src}}\})$ evaluated at each template point $\bar{\mathbf{x}}_i \in \bar{\mathcal{P}}$. Finally, the single-scene version, abbreviated simply as IPC in tab. 2, is a special case of IPC+AD with $\mathbf{z}_{\text{scene}}(\text{sequence}_{\text{ID}}) := \mathbf{0}$ set to a constant zero vector.

Metric	Domain	Description
$BA_{\text{final cost}}$	$\mathbb{R}$	The final value of the Bundle Adjustment (BA) cost function.
$BA_{\text{termination}}$	$\{0, 1\}$	The termination state of BA (converged/not converged).
$\mu_{\text{det score}}$	$[0, 1]$	An average over per-frame detection scores of the PointRend object detector.
$\mu_{\text{perc detected}}$	$[0, 100]$	Percentage of frames in which the category of interest is detected with PointRend.
N_{cameras}	$\mathbb{N}$	The number of cameras registered during BA.
$N_{\text{sparse pts}}$	$\mathbb{N}$	Number of points in the sparse point cloud.
$PCL^{\text{render}}_{\ell\text{depth}}$	$\mathbb{R}$	The average ℓ_1 depth error between the renders of the fused pointcloud $\mathcal{P}(\mathcal{V})$ into each camera P_i of a video $\mathcal{V}$ and the corresponding dense depth map P_i.
$PCL^{\text{render}}_{\ell\text{rgb}}$	$\mathbb{R}$	The average ℓ_1 RGB error between the renders of the fused pointcloud $\mathcal{P}(\mathcal{V})$ into each camera P_i of a video $\mathcal{V}$ and the corresponding frame I_i.
$PCL^{\text{render}}_{IoU}$	[0, 1]	The average Jaccard Index between the renders of the fused point cloud $\mathcal{P}(\mathcal{V})$ into each camera P_i of a video $\mathcal{V}$ and the corresponding PointRend segmentation M_i.
$PCL^{\text{direction cover}}$	$\mathbb{N}$	Measures the coverage of the views of the point cloud $\mathcal{P}(\mathcal{V})$ with the number of occupied bins in the azimuth/elevation map of projection rays corresponding to each dense point cloud point $\mathbf{x}_j$ and a camera P_{i}.

Table IV: **The list of SfM and point-cloud reconstruction metrics** that serve as a set of features for training the active-SVM that labels camera and reconstruction quality.

Training All versions of IPC optimize the MSE between the rendered image $\hat{I}^{\text{tgt}}$ and the ground truth colors I^{tgt}. Furthermore, we make use of the ground truth segmentation masks and minimize the Chamfer distance between the set of 2D projections of the predicted point cloud points $\mathcal{P}(\mathbf{z})$, and the 2D points of the ground truth segmentation mask [36]. Note that a standard segmentation loss, such as DICE [57] or Binary Cross Entropy between the rendered alpha mask and the ground truth segmentation mask, do not apply here. This is because the gradients generated by the alpha mask renders of IPC are not well-defined and do not lead to convergence.

C. 3D annotations with Human-in-the-loop

In sec. 3, we outlined the process of annotating the AMT-collected videos with 3D ground truth. Here, we further detail the semi-automated process of labelling the quality of camera tracking and the 3D dense point cloud of the captured videos (Paragraph 4 in sec. 3).

We initialize the process by annotating an initial set of several hundreds of reconstructions with a binary label ”accurate / inaccurate” by visually inspecting both the camera tracks $(P_i | P_i \in R^{4\times4})_{i=1}^{N_I}$ and the scene point cloud $\mathcal{P}(\mathcal{V})$. From each video, we then extract various metrics that are indicative of the reconstruction quality such as a per-pixel RGB and depth error of the rendered point cloud, the number of registered cameras, final bundle adjustment energy etc. The full set of metrics is outlined in tab. IV. We then train a binary Support Vector Machine (SVM [12]) with an RBF kernel that regresses the binary label given the reconstruction metrics as input.

Afterwards, the trained SVM classifies all previously unlabelled videos. In line with the uncertainty principle [59], we manually annotate a subset of previously unlabelled samples that are the closest to the SVM decision boundary. We further correct significant classification errors by inspecting the highest/lowest scoring samples. In this manner, we alternate between SVM training and manual annotation until 1.5k labels are collected (8 % of the whole dataset).

In order to validate the SVM’s performance, we conduct a 5-fold cross-validation on the set of annotated videos. The cross-validation indicates that the SVM has 90% and 78% accuracy for classifying the camera tracking and point cloud quality respectively.

References

[1] Adel Ahmadyan, Liangkai Zhang, Jianing Wei, Artsiom Ablavatski, and Matthias Grundmann. Objectron: A large scale dataset of object-centric videos in the wild with pose annotations. *Proc. CVPR*, 2021. 2

[2] Matan Atzmon and Yaron Lipman. Sal: Sign agnostic learning of shapes from raw data. In *Proc. CVPR*, 2020. 2, 4

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv*, 2016. 6

[4] Piotr Bojanowski, Armand Joulin, David Lopez-Paz, and Arthur Szlam. Optimizing the latent space of generative networks. *arXiv*, 2017. 8

[5] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 425–432, 2001. 2

[6] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv*, 2015. 1, 2

[7] Wenzheng Chen, Jun Gao, Huan Ling, Edward J. Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to Predict 3D Objects with an Interpolation-based Differentiable Renderer. In *Proc. NeurIPS*, 2019. 2

[8] 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 *Proc. NIPS*, 2019. 1, 2

[9] Sungjoon Choi, Qian-Yi Zhou, Stephen Miller, and Vladlen Koltun. A large dataset of object scans. *arXiv*, 2016. 2

[10] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *Proc. ECCV*, 2016. 2

[11] David Cohn, Les Atlas, and Richard Ladner. Improving generalization with active learning. *Machine learning*, 15(2):201–221, 1994. 3

[12] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3), 1995. 14

[13] Alexey Dosovitskiy and Thomas Brox. Generating images with perceptual similarity metrics based on deep networks. In *Proc. NeurIPS*, 2016. 1

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

[15] Matheus Gadelha, Subhransu Maji, and Rui Wang. 3d shape induction from 2d views of multiple objects. In *2017 International Conference on 3D Vision (3DV)*, pages 402–411. IEEE, 2017. 2

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

[17] Kyle Genova, Forrester Cole, Daniel Vlasic, Aaron Sarna, William T Freeman, and Thomas Funkhouser. Learning shape templates with structured implicit functions. In *Proc. ICCV*, 2019. 2

[18] Rohit Girdhar, David F Fouhey, Mikel Rodriguez, and Abhinav Gupta. Learning a predictable and generative vector representation for objects. In *Proc. ECCV*, 2016. 2, 5

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

[20] Shubham Goel, Angjoo Kanazawa, and Jitendra Malik. Shape and viewpoint without keypoints. *Proc. ECCV*, 2020. 2

[21] GoogleResearch. Google scanned objects, September. 2

[22] David Ha, Andrew M. Dai, and Quoc V. Le. HyperNetworks. 2017. 4

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015. 11

[24] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (SIGGRAPH Asia Conference Proceedings)*, 37(6), November 2018. 2

[25] Philipp Henzler, Niloy Mitra, and Tobias Ritschel. Escaping plato's cave using adversarial training: 3d shape from unstructured 2d image collections. In *Proc. ICCV*, 2019. 1, 2, 4

[26] Philipp Henzler, Jeremy Reizenstein, Patrick Labatut, Roman Shapovalov, Tobias Ritschel, Andrea Vedaldi, and David Novotny. Unsupervised learning of 3d object categories from videos in the wild. *Proc. CVPR*, 2021. 1, 2, 4, 5, 6, 7, 8, 10

[27] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8), 1997. 12

[28] Eldar Insafutdinov and Alexey Dosovitskiy. Unsupervised learning of shape and pose with differentiable point clouds. In *Proc. NIPS*, 2018. 2, 4

[29] Hanbyul Joo, Tomas Simon, Xulong Li, Hao Liu, Lei Tan, Lin Gui, Sean Banerjee, Timothy Godisart, Bart C. Nabbe, Iain A. Matthews, Takeo Kanade, Shohei Nobuhara, and Yaser Sheikh. Panoptic studio: A massively multiview system for social interaction capture. *PAMI*, 41(1), 2019. 2

[30] Angjoo Kanazawa, Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. Learning category-specific mesh reconstruction from image collections. In *Proc. ECCV*, 2018. 2

[31] Abhishek Kar, Christian Häne, and Jitendra Malik. Learning a multi-view stereo machine. In *Proc. NIPS*, 2017. 2, 13

[32] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proc. CVPR*, 2018. 2

[33] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. Pointrend: Image segmentation as rendering. In *Proc. CVPR*, 2020. 3

[34] Nilesh Kulkarni, Abhinav Gupta, David F. Fouhey, and Shubham Tulsiani. Articulation-aware canonical surface mapping. In *Proc. CVPR*, pages 449–458, 2020. 2

[35] Nilesh Kulkarni, Abhinav Gupta, and Shubham Tulsiani. Canonical surface mapping via geometric cycle consistency. In *Proc. ICCV*, 2019. 2

[36] Xueting Li, Sifei Liu, Kihwan Kim, Shalini De Mello, Varun Jampani, Ming-Hsuan Yang, and Jan Kautz. Self-supervised single-view 3d reconstruction via semantic consistency. *Proc. ECCV*, 2020. 2, 14

[37] Joseph J. Lim, Hamed Pirsiavash, and Antonio Torralba. Parsing IKEA Objects: Fine Pose Estimation. *Proc. ICCV*, 2013. 1

[38] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In *Proc. ECCV*, 2014. 3

[39] Lingjie Liu, Jiatao Gu, Kyaw Zaw Lin, Tat-Seng Chua, and Christian Theobalt. Neural sparse voxel fields. *Proc. NIPS*, 2020. 2

[40] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3D reasoning. *arXiv.cs*, Proc. CVPR, 2019. 2, 4, 5, 12

[41] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *ACM Trans. Graph.*, 38(4):65:1–65:14, July 2019. 2, 4, 5, 6, 7, 10, 11, 12, 13

[42] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy net-

works: Learning 3d reconstruction in function space. In *Proc. CVPR*, 2019. 2

[43] 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. *Proc. ECCV*, 2020. 2, 4, 6, 10, 11

[44] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. HoloGAN: Unsupervised learning of 3D representations from natural images. *Proc. ICCV*, abs/1904.01326, 2019. 2

[45] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proc. CVPR*, pages 3504–3515, 2020. 2, 4, 6, 7, 10, 12

[46] Michael Niemeyer, Lars M. Mescheder, Michael Oechsle, and Andreas Geiger. Occupancy flow: 4d reconstruction by learning particle dynamics. In *Proc. ICCV*, 2019. 1, 4

[47] David Novotny, Diane Larlus, and Andrea Vedaldi. Learning 3d object categories by looking around them. In *Proc. ICCV*, 2017. 2

[48] David Novotný, Diane Larlus, and Andrea Vedaldi. Capturing the geometry of object categories from video supervision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018. 2

[49] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proc. CVPR*, 2019. 2

[50] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv*, 2020. 4, 6, 7, 8, 10, 11, 12, 13

[51] Danilo Jimenez Rezende, SM Ali Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg, and Nicolas Heess. Unsupervised learning of 3d structure from images. In *Proc. NIPS*, 2016. 2

[52] Shunsuke Saito, , Zeng Huang, Ryota Natsume, Shigeo Morishima, Angjoo Kanazawa, and Hao Li. Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. *Proc. ICCV*, 2019. 2

[53] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proc. CVPR*, 2016. 3, 4

[54] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *Proc. ECCV*, 2016. 3

[55] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. *Proc. NIPS*, 2020. 2, 4

[56] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. *CoRR*, abs/1906.01618, 2019. 1, 2, 4, 5, 6, 7, 8, 10, 11, 12

[57] Carole H Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M Jorge Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. In *Deep learning in medical image analysis and multimodal learning for clinical decision support*. 2017. 14

[58] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Multi-view 3d models from single images with a convolutional network. In *Proc. ECCV*, 2016. 5

[59] Simon Tong and Edward Y. Chang. Support vector machine active learning for image retrieval. In *ACM Multimedia*, 2001. 14

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

[61] Shubham Tulsiani, Alexei A Efros, and Jitendra Malik. Multi-view consistency as supervisory signal for learning shape and pose prediction. In *Proc. CVPR*, 2018. 2, 5

[62] Shubham Tulsiani, Abhishek Kar, Joao Carreira, and Jitendra Malik. Learning category-specific deformable 3D models for object reconstruction. *PAMI*, 39(4):719–731, 2017. 1

[63] Shubham Tulsiani, Tinghui Zhou, Alexei A Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *Proc. CVPR*, 2017. 2

[64] Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *Proc. CVPR*, pages 209–217, 2017. 4

[65] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Proc. NeurIPS*, 2017. 2, 6

[66] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017. 4

[67] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *Proc. ECCV*, 2018. 2, 6, 7, 12

[68] 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. *Proc. CVPR*, 2021. 2

[69] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. Synsin: End-to-end view synthesis from a single image. In *Proc. CVPR*, pages 7467–7477, 2020. 4, 7, 13

[70] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In *Proc. NeurIPS*, 2016. 5

[71] Shangzhe Wu, Christian Rupprecht, and Andrea Vedaldi. Unsupervised learning of probably symmetric deformable 3d objects from images in the wild. In *Proc. CVPR*, pages 1–10, 2020. 2

[72] Yu Xiang, Roozbeh Mottaghi, and Silvio Savarese. Beyond PASCAL: A benchmark for 3D object detection in the wild. In *Proc. WACV*, 2014. 2

[73] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. In *Proc. ICCV*, 2019. 2

[74] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. *Proc. NIPS*, 2020. 2, 4, 6, 7, 10, 12

[75] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images. *Proc. ECCV*, 2020. 2, 4, 5

[76] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proc. CVPR*, 2018. 7

[77] Yuxuan Zhang, Wenzheng Chen, Huan Ling, Jun Gao, Yinan Zhang, Antonio Torralba, and Sanja Fidler. Image gans meet differentiable rendering for inverse graphics and interpretable 3d neural rendering. *Proc. ICLR*, 2021. 2