Neural-Sim: Learning to Generate Training Data with NeRF

Yunhao Ge[1], Harkirat Behl[2*], Jiashu Xu[1*], Suriya Gunasekar[2], Neel Joshi[2], Yale Song[2], Xin Wang[2], Laurent Itti[1], and Vibhav Vineet[2]

[1] University of Southern California
[2] Microsoft Research

Abstract. Training computer vision models usually requires collecting and labeling vast amounts of imagery under a diverse set of scene configurations and properties. This process is incredibly time-consuming, and it is challenging to ensure that the captured data distribution maps well to the target domain of an application scenario. Recently, synthetic data has emerged as a way to address both of these issues. However, existing approaches either require human experts to manually tune each scene property or use automatic methods that provide little to no control; this requires rendering large amounts of random data variations, which is slow and is often suboptimal for the target domain. We present the first fully differentiable synthetic data pipeline that uses Neural Radiance Fields (NeRFs) in a closed-loop with a target application's loss function. Our approach generates data on-demand, with no human labor, to maximize accuracy for a target task. We illustrate the effectiveness of our method on synthetic and real-world object detection tasks. We also introduce a new "YCB-in-the-Wild" dataset and benchmark that provides a test scenario for object detection with varied poses in real-world environments. Code and data could be found at https://github.com/gyhandy/Neural-Sim-NeRF.

Keywords: Synthetic data, NeRF, Bilevel optimization, Detection

1 Introduction

The traditional pipeline for building computer vision models involves collecting and labelling vast amounts of data, training models with different configurations, and deploying it to test environments [24,37,42]. Key to achieving good performance is collecting training data that mimics the test environment with similar properties relating to the object (pose, geometry, appearance), camera (pose and angle), and scene (illumination, semantic structures)[2].

However, the traditional pipeline does not work very well in many real-world applications as collecting large amounts of training data which captures all variations of objects and environments is quite challenging. Furthermore, in many

[*] Equal contribution as second author

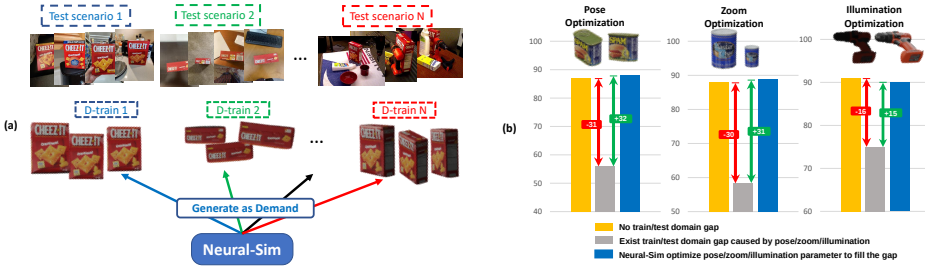

Fig. 1: (a) On-demand synthetic data generation: Given a target task and a test dataset, our approach "Neural-sim" generates data on-demand using a fully differentiable synthetic data generation pipeline which maximises accuracy for the target task. (b) Train/test domain gap causes significant detection accuracy drop (yellow bar to gray bar). We dynamically optimize the render parameters (pose/zoom/illumination) to generate the best data to fill the gap (blue bar).

applications, users may want to learn models for unique objects with novel structures, textures, or other such properties. Such scenarios are very common particularly in business scenarios where there is desire to create object detectors for new products introduced in the market.

Recent advances in rendering, such as photo-realistic renderers [10,21] and generative models (GANs [6], VAEs [12,25]), have brought the promise of generating high-quality images of complex scenes. This has motivated the field to explore synthetic data as source of training data [13,28,14,23,27,38,40,44,18,52,19]. However, doing so in an offline fashion has similar issues as the traditional pipeline. While it alleviates certain difficulties, e.g., capturing camera/lighting variations, it create dependency on 3D asset creation, which is time-consuming.

Recently, a new image generation technique called the Neural Radiance Field (NeRF) [34] was introduced as a way to replace the traditional rasterization and ray-tracing graphics pipelines with a neural-network based renderer. This approach can generate high-quality novel views of scenes without requiring explicit 3D understanding. More recent advancements in NeRFs allow to control other rendering parameters, like illumination, material, albedo, appearance, etc. [43,33,51,5,29]. As a result, they have attracted significant attention and have been widely adopted in various graphics and vision tasks [16,5,43,36]. NeRF and their variants possess some alluring properties: (i) differentiable rendering, (ii) control over scene properties unlike GANs and VAEs, and (iii) they are data-driven in contrast to traditional renderers which require carefully crafting 3D models and scenes. These properties make them suitable for generating the optimal data on-demand for a given target task.

To this end, we propose a bilevel optimization process to jointly optimize neural rendering parameters for data generation and model training. Further, we

also propose a reparameterization trick, sample approximation, and patch-wise optimization methods for developing a memory efficient optimization algorithm.

To demonstrate the efficacy of the proposed algorithm, we evaluate the algorithm on three settings: controlled settings in simulation, on the YCB-video dataset [47], and in controlled settings on YCB objects captured in the wild. This third setting is with our newly created "YCB-in-the-wild" dataset, which involves capturing YCB objects in real environments with control over object pose and scale. Finally, we also provide results showing the interpretability of the method in achieving high performance on downstream tasks. Our key contributions are as follows:

(1) To the best of our knowledge, for the first time, we show that NeRF can substitute the traditional graphics pipeline and synthesize useful images to train downstream tasks (object detection).

(2) We propose a novel bilevel optimization algorithm to automatically optimize rendering parameters (pose, zoom, illumination) to generate optimal data for downstream tasks using NeRF and its variants.

(3) We demonstrate the performance of our approach on controlled settings in simulation, controlled settings in YCB-in-wild and YCB-video datasets. We release YCB-in-wild dataset for future research.

2 Related work

Traditional Graphics rendering methods can synthesize high-quality images with controllable image properties, such as object pose, geometry, texture, camera parameters, and illumination [38,10,21,27,39]. Interestingly, NeRF has some important benefits over the traditional graphics pipelines, which make it more suitable for learning to generate synthetic datasets. First, NeRF learns to generate data from new views based only on image data and camera pose information. In contrast, the traditional graphics pipeline requires 3D models of objects as input. Getting accurate 3D models with correct geometry, material, and texture properties generally requires human experts (i.e. an artist or modeler). This, in turn, limits the scalability of the traditional graphics pipeline in large-scale rendering for many new objects or scenes. Second, NeRF is a differentiable renderer, thus allowing backpropagation through the rendering pipeline for learning how to control data generation in a model and scene-centric way.

Deep generative models, such as GANs [22,6], VAEs [12,25] and normalizing flows [9] are differentiable and require less human involvement. However, most of them do not provide direct control of rendering parameters. While some recent GAN approaches allow some control [48,1,20] over parameters, it is not as explicit and can mostly only change the 2D properties of images. Further, most generative models need a relatively large dataset to train. In comparison, NeRF can generate parameter-controllable high-quality images and requires a lesser number of images to train. Moreover, advancements in NeRF now allow the control of illumination, materials, and object shape alongside camera pose and

scale [43,33,51,5,29]. We use NeRF and their variants (NeRF-in-the-wild [33]) to optimize pose, zoom and illumination as representative rendering parameters.

Learning simulator parameters. Related works in this space focus on learning non-differentiable simulator parameters for e.g., learning-to-simulate (LTS) [41], Meta-Sim [30], Meta-Sim2 [11], Auto-Sim [4], and others [49,17,32]. Our work in contrast has two differences: (i) a difference in the renderer used (NeRF vs traditional rendering engines), and (ii) a difference in the optimization approach. We discuss the different renderers and their suitability for this task in the previous subsection.

LTS [41] proposed a bilevel optimization algorithm to learn simulator parameters that maximized accuracy on downstream tasks. It assumed both data-generation and model-training as a black-box optimization process and used REINFORCE-based [45] gradient estimation to optimize parameters. This requires many intermediate data generation steps. Meta-sim [30] is also a REINFORCE based approach, which requires a grammar of scene graphs. Our approach does not use scene grammar. Most similar to our work is the work of Auto-Simulate [4] that proposed a local approximation of the bilevel optimization to efficiently solve the problem. However, since they optimized non-differentiable simulators like Blender [10] and Arnold [21], they used REINFORCE-based [45] gradient update. Further, they have not shown optimization of pose parameter whose search space is very large. In comparison, our proposed Neural-Sim approach can learn to optimize over pose parameters as well.

3 Neural-Sim

The goal of our method is to automatically synthesize optimal training data to maximize accuracy for a target task. In this work, we consider object detection as our target task. Furthermore, in recent times, NeRF and its variants (NeRFs) have been used to synthesize high-resolution photorealistic images for complex scenes [43,33,51,5,29]. This motivates us to explore NeRFs as potential sources of generating training data for computer vision models. We propose a technique to optimize rendering parameters of NeRFs to generate the *optimal set* of images for training object detection models.

NeRF model: NeRF [34,50] takes as input the viewing direction (or camera pose) denoted as $V = (\phi, \rho)$, and renders an image $x = \text{NeRF}(V)$ of a scene as viewed along V. Note that our proposed technique is broadly applicable to differentiable renderers in general. In this work, we also optimize NeRF-in-the-wild (NeRF-w) [33] as it allows for appearance and illumination variations alongside pose variation. We first discuss our framework for optimizing the original NeRF model and later we discuss optimization of NeRF-w in Section 3.2.

Synthetic training data generation: Consider a parametric probability distribution p_ψ over rendering parameters V, where ψ denotes the parameters of the distribution. It should be noted that ψ corresponds to all rendering parameters including pose/zoom/illumination, here, for simplicity, we consider ψ to denote pose variable. To generate the synthetic training data, we first sample ren-

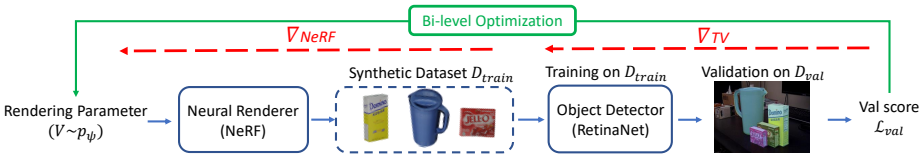

Fig. 2: Neural-Sim pipeline: Our pipeline finds the optimal parameters for generating views from a trained neural renderer (NeRF) to use as training data for object detection. The objective is to find the optimal NeRF rendering parameters ψ that can generate synthetic training data D_{train}, such that the model (RetinaNet, in our experiments) trained on D_{train}, maximizes accuracy on a downstream task represented by the validation set D_{val}.

dering parameters $V_1, V_2, ..., V_N \sim p_\psi$. We then use NeRF to generate synthetic training images $x_i = \text{NeRF}(V_i)$ with respective rendering parameters V_i. We use an off-the-shelf foreground extractor to obtain labels $y_1, y_2, \ldots, y_N$. the training dataset thus generated is denoted as $D_{train} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$.

Optimizing synthetic data generation Our goal is to optimize over the rendering distribution p_ψ such that training an object detection model on D_{train} leads to good performance on D_{val}. We formulate this problem as a bi-level optimization [8,15,4] as below:

$$\min_{\psi} \ \mathcal{L}_{val}(\hat{\theta}(\psi)); \qquad s.t. \ \ \hat{\theta}(\psi) \in \arg\min_{\theta} \ \mathcal{L}_{train}(\theta, \psi), \tag{1a}$$

where θ denotes the parameters of the object detection model, $\mathcal{L}_{train}(\theta, \psi) = \mathbb{E}_{V \sim p_\psi} l(x, \theta) \approx \frac{1}{N} \sum_{i=1}^{N} l(x_i, \theta)$ is the training loss over the synthetic dataset from NeRF, [3] and $\mathcal{L}_{val}$ is the loss on the task-specific validation set D_{val}.

The bi-level optimization problem in (1) is challenging to solve; for example, any gradient based algorithm would need access to an efficient approximation of $\nabla_\psi \hat{\theta}(\psi)$, which in turn requires propagating gradients through the entire training trajectory of a neural network. Thus, we look to numerical approximations to solve this problem. Recently, Behl et. al. [4] developed a technique for numerical gradient computation based on local approximation of the bi-level optimization. Without going into their derivation, we borrow the gradient term for the outer update, which at time step t takes the form:

$$\frac{\partial \mathcal{L}_{val}(\hat{\theta}(\psi))}{\partial \psi}\bigg|_{\psi=\psi_t} \approx - \overbrace{\frac{\partial}{\partial \psi}\Big[\frac{\partial \mathcal{L}_{train}\big(\hat{\theta}(\psi_t), \psi\big)}{\partial \theta}\Big]^T}^{\nabla_{NeRF}}\bigg|_{\psi=\psi_t} \underbrace{\mathcal{H}(\hat{\theta}(\psi_t), \psi)^{-1} \frac{d\mathcal{L}_{val}(\hat{\theta}(\psi_t))}{d\theta}}_{\nabla_{TV}}. \tag{2}$$

[3] For simplicity, we have dropped the dependence of loss ℓ on labels y

We have divided the gradient term into two parts: ∇_{NeRF} corresponds to backpropagation through the dataset generation from NeRF, and ∇_{TV} corresponds to approximate backpropagation through training and validation (Fig. 2). ∇_{TV} is computed using the conjugate gradient method [4]. However, [4] treated the data generation as a black box and used REINFORCE [46] to compute the approximate gradient because they used non-differentiable renderers for data generation. However, REINFORCE is considered noisy process and is known to lead to high-variance estimates of gradients. In contrast, NeRF is differentiable, which gives us tools to obtain more accurate gradients. We propose an efficient technique for computing ∇_{NeRF}, which we discuss in the next section.

3.1 Backprop through data generation from NeRF

A good gradient estimation should possess the following properties: (i) high accuracy and low noise, (ii) computational efficiency, (iii) low memory footprint. We leverage different properties of NeRF, i.e., its differentiability and pixel-wise rendering, to design a customized technique which satisfies the above properties.

In computation of ∇_{NeRF} in (2), we approximate $\mathcal{L}_{train}(\theta, \psi)$ using samples in D_{train} as $\mathcal{L}_{train}(\theta, \psi) \approx \frac{1}{N}\sum_{i=1}^{N} l(x_i, \theta)$. Using chain rule we then have partial derivative computation over $l(x, \theta)$ as follows:

$$\frac{\partial}{\partial \psi}\left[\frac{\partial l(x_i, \hat{\theta}(\psi_t))}{\partial \theta}\right] = \left[\frac{\partial(\frac{\partial l(x_i, \hat{\theta}(\psi_t))}{\partial \theta})}{\partial x_i}\right]\left[\frac{\partial x_i}{\partial V_i}\right]\left[\frac{dV_i}{d\psi}\right] \tag{3}$$

The first term is the second order derivative through a detection network and can be computed analytically for each image x_i. The second term is the gradient of the rendered image w.r.t NeRF inputs, which can be obtained by backpropagating through the differentiable NeRF rendering $x_i = \text{NeRF}(V_i)$. While both these terms have exact analytical expressions, naively computing and using them in (2) becomes impractical even for small problems (see below in Tool2 and Tool3 for details and proposed solutions). Finally the third term $\frac{dV_i}{d\psi}$ requires gradient computation over probabilistic sampling $V_i \sim p_\psi$. We consider p_ψ over discretized bins of pose parameters. For such discrete distributions $\frac{dV_i}{d\psi}$ is not well defined. Instead, we approximate this term using a reparameterization technique described below in Tool1. We summarize our technical tools below:

- For distributions p_ψ over a discrete bins of pose parameters, we propose a reparametrization of ψ that provides efficient approximation of $\frac{dV_i}{d\psi}$ (Tool1).
- We dramatically reduce memory and computation overhead of implementing the gradient approximation in (2) using a new *twice-forward-once-backward* approach (Tool2). Without this new technique the implementation would require high computation involving large matrices and computational graphs.
- Even with the above technique, the computation of first and second terms in (3) has a large overhead on GPU memory that depends on image size. We overcome this using a patch-wise gradient computation approach (Tool 3).

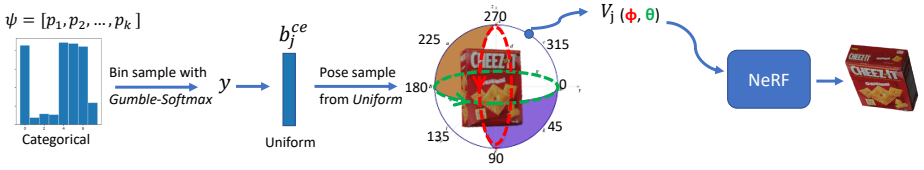

Fig. 3: A concrete example to one time sample, starting form a particular value of ψ, we can follow reparametrization sampling and obtain a pose. Each sample represents a pose that is input in NeRF to render one image.

Tool 1: Reparametrization of pose sampling NeRF renders images x_j using camera pose V_j=(ϕ_i, ρ_j), where $\phi_j \in [0, 360], \rho_j \in [0, 360]$. For simplicity we describe our method for optimizing over ϕ, while keeping ρ fixed to be uniform.

We discretize the pose into k equal sized bins over the range of ϕ as $B_1 = \left[0, \frac{360}{k}\right), B_2 = \left[\frac{360}{k}, \frac{360\times 2}{k}\right), \ldots$. and define the distribution over ϕ as the categorical distribution with p_i as the probability of ϕ belonging to B_i. This distribution is thus parametrized by $\psi \equiv p = [p_1, ..., p_k]$. To back propagate through the sampling process, we approximate the sample from the categorical distribution by using Gumble-softmax "reparameterization trick" with parameters $y \in \mathbb{R}^k$, where y_i are given as follows: $y_i = \mathrm{GS}_i(p) = \exp[(G_i + log(p_i))/\tau] / \sum_j \exp[(G_i + log(p_j))/\tau]$. Where $G_i \sim Gumbel(0, 1)$ are i.i.d. samples from the standard Gumbel distribution and τ is temperature parameter. The random vector y defined as above satisfies the property that the coordinate (index) of the largest element in $y \in \mathbb{R}^k$ follows the categorical distribution with parameter p.

We now approximate sampling from the categorical distribution (see Figure 3 for depiction). Denote the bin center of B_i as $\bar{B}_i^{ce} = 360(i - 0.5)/k$; and the bin range as $\bar{b}^{ra} = 360/k$. We generate $V_j = (\phi_j, \rho_j) \sim p_\psi$ as below:

- Generate y_i's for $i = 1, 2, \ldots k$
- Define $b_j^{ce} = \sum_i y_i \bar{B}_i^{ce}$ as the approximate bin center.
- Define the bin for the j^{th} sample centered around b_j^{ce} as $[b_j^{st}, b_j^{en}] = [b_j^{ce} - \bar{b}^{ra}/2, b_j^{ce} + \bar{b}^{ra}/2]$
- We sample ϕ_j from uniform distribution over $[b_j^{st}, b_j^{en}]$ which has a reparametrization for diffentiability: $\mathcal{U}(b_j^{st}, b_j^{en}) \equiv (1 - \epsilon) b_j^{st} + \epsilon b_j^{en}$ s.t. $\epsilon \sim \mathcal{U}(0, 1)$.
- $\rho_j \sim \mathcal{U}[0, 360]$, or can follow same process as ϕ_j.

Note that in general the approximate bin centers b_j^{ce} need not be aligned with original categorical distribution, however we can control the approximation using the temperature parameter τ. We now have the full expression for approximate gradient of ∇_{NeRF} using (3) and reparametrization as follows:

$$\nabla_{NeRF} \approx \frac{1}{N} \sum_{j=1}^{N} \frac{\partial(\frac{\partial l(x_j, \hat{\theta}(\psi_t))}{\partial \theta})}{\partial x_j} \frac{\partial x_j}{\partial V_j} \frac{\partial V_j}{\partial (b_j^{st}, b_j^{en})} \frac{\partial (b_i^{st}, b_i^{en})}{\partial y} \frac{\partial y}{\partial p}. \tag{4}$$

Below we present tools that drastically improve the compute and memory efficiency and are crucial for our pipeline.

Tool 2: Twice-forward-once-backward The full gradient update of our bi-level optimization problem involves using the approximation of ∇_{NeRF} in (4) and back in (2). This computation has three terms with the following dimensions: (1) $\frac{\partial(\frac{\partial l(x_j,\hat{\theta}(\psi_t))}{\partial\theta})}{\partial x_j} \in \mathbb{R}^{m\times d}$, (2) $\frac{\partial x_j}{\partial\psi} \in \mathbb{R}^{d\times k}$, (3) $\nabla_{TV} = \mathcal{H}(\hat{\theta}(\psi_t),\psi)^{-1}\frac{d\mathcal{L}_{val}(\hat{\theta}(\psi_t))}{d\theta} \in \mathbb{R}^{m\times 1}$, where $m = |\theta|$ is the # of parameters in object detection model, d is the # of pixels in x, and k is # of pose bins.

Implementing eq. (2) with the naive sequence of (1)-(2)-(3) involves computing and multiplying large matrices of sizes $m \times d$ and $d \times k$. Further, this sequence also generates a huge computation graph. These would lead to prohibitive memory and compute requirements as m is often in many millions. On the other hand, if we could follow the sequence of (3)-(1)-(2), then we can use the produce of $1 \times m$ output of (3) to do a weighted autograd which leads computing and storing only vectors rather than matrices. However, the computation of (3) needs the rendered image involving forward pass of (2) (more details in appendix.).

To take advantage of the efficient sequence, we propose a twice-forward-once backward method where we do two forward passes over NeRF rendering. In the first forward path, we do not compute the gradients, we only render images to form D_{train} and save random samples of y, ϕ_j used for rendering. We then compute (3) by turning on gradients. In the second pass through NeRF, we keep the same samples and this time compute the gradient (1) and (2).

Tool 3: Patch-wise gradient computation Even though we have optimized the computation dependence on $m = |\theta|$ with the tool described above, computing (1)-(2) sequence in the above description still scales with the size of images d. This too can lead to large memory footprint for even moderate size images. To optimize the memory further, we propose patch-wise computation, where we divide the image into S patches $x = (x^1, x^2, \ldots, x^S)$) and compute (3) as follows:

$$\frac{\partial}{\partial\psi}\frac{\partial l(x,\hat{\theta}(\psi_t))}{\partial\theta} = \sum_{c=1}^{S}\frac{\partial(\frac{\partial l(x^c,\hat{\theta}(\psi_t))}{\partial\theta})}{\partial x^c}\frac{\partial x^c}{\partial\psi}. \tag{5}$$

Since NeRF renders an image pixel by pixel, it is easy to compute the gradient of patch with respect to ψ in the memory efficient patch-wise optimization.

3.2 Nerf-in-the-wild

NeRF-in-the-wild (NeRF-w) extends NeRF model to allow image dependent appearance and illumination variations such that photometric discrepancies between images can be modeled explicitly. NeRF-w takes as input an appearance embedding denoted as ℓ alongside the viewing direction V to render an image as $x = \text{NeRF}(V, \ell)$. For NERF-w, the optimization of pose (V) remains the same as discussed above. For efficient optimization of lighting we exploit a noteworthy property of NeRF-w: it allows smooth interpolations between color and lighting. This enables us to optimize lighting as a continuous variable, where the lighting

(ℓ) can be written as an affine function of the available lighting embeddings (ℓ_i) as $\ell = \sum_i \psi_i * \ell_i$ where $\sum_i \psi_i = 1$. To calculate the gradient from Eq. 3, $\frac{\partial x_i}{\partial \ell}$ is computed in the same way as described above utilizing our tools 2 and 3, and the term $\frac{d\ell}{d\psi}$ is straightforward and is optimized with projected gradient descent.

4 Experiments

We now evaluate the effectiveness of our proposed Neural-Sim approach in generating optimal training data on object detection task. We provide results under two variations of our Neural-Sim method. In the first case, we use Neural-Sim without using bi-level optimization steps. In this case, data from NeRF are always generated from the same initial distribution. The second case involves our complete Neural-Sim pipeline with bi-level optimization updates (Eq. 2). In the following sections, we use terms NS and NSO for Neural-Sim without and Neural-Sim with bi-level optimization respectively.

We first demonstrate that NeRF can successfully generate data for downstream tasks as a substitute for a traditional graphic pipeline (e.g., BlenderProc) (see appendix for results) with similar performance. Then we conduct experiments to demonstrate the efficacy of Neural-Sim in three different scenarios: controllable synthetic tasks on YCB-synthetic dataset (Sec. 4.1); controllable real-world tasks on YCB-in-the-wild dataset (Sec. 4.2); general real-world tasks on YCB-Video dataset (Sec. 4.3). We also show the interpretable properties of the Neural-Sim approach (NSO) during training data synthesis (Sec. 4.4). All three datasets are based on the objects from the YCB-video dataset [47,26,7]. It contains 21 objects and provides high-resolution RGBD images with ground truth annotation for object bounding boxes. The dataset consists of both digital and physical objects, which we use to create both real and synthetic datasets.

Implementation details: We train one NeRF-w model for each YCB object using 100 images with different camera pose and zoom factors using BlenderProc. We use RetinaNet [31] as our downstream object detector. To accelerate the optimization, we fix the backbone during training. During bi-level optimization steps, we use Gumble-softmax temperature $\tau = 0.1$. In each optimization iteration, we render 50 images for each object class and train RetinaNet for two epochs. More details are in the appendix.

Baselines: We compare our proposed approach against two state-of-the-art approaches that learn simulator parameters. First is Learning to simulate (LTS) [41] which proposed a REINFORCE-based simulator optimization approach. Also note that the meta-sim [30] is a REINFORCE-based approach. Next, we consider Auto-Sim [4] which proposed an efficient method to learn simulator parameters. We implemented LTS and received code from the authors of Auto-Sim.

4.1 YCB-synthetic dataset

Next, we conduct experiments on a YCB-synthetic dataset to show how NSO helps to solve a drop in performance due to distribution shifts between the training and test data.

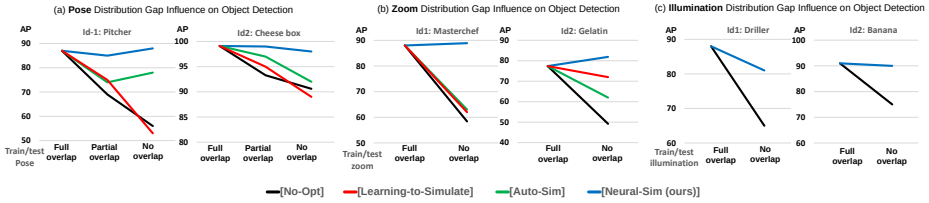

Fig. 4: Neural-Sim performance on YCB-Synthetic. When there are distribution gap between train and test sets ((a) pose (b) zoom (c) illumination gap), with the gap increase, object detection faces larger accuracy drop (black line). With the help of Neural-Sim (NSO) in blue line, the performance drop are filled. Observe improvement of NSO over LTS [41] (red line) and Auto-Sim [4] (green line).

Dataset setting We select six objects that are easily confused with each other: *masterchef* and *pitcher* are both blue cylinders and *cheezit*, *gelatin*, *mug* and *driller* are all red colored objects. To conduct controlled experiments, we generate data with a gap in the distribution of poses between the training and test sets. For this, we divide the object pose space into k= 8 bins. For each objects o_j and pose bin i combination, we use BlenderProc [10] to synthesize 100 images. These images of the six selected objects with pose bin-labels form YCB-synthetic data.

Train/test biasness We create controlled experiments by varying the degree of pose distribution overlap between the training and test sets. For each object (e.g. *pitcher*) we fix its pose distribution in the test set (e.g. images are generated with pose from bin 1) and change its pose distribution in training set in three ways. First, images are generated with pose with same distribution as test set (bin1 is dominant), uniform distribution (pose values uniformly selected from bin1 to bin 8) and totally different distribution from the test set (other bins are dominant except bin 1). We introduce such pose biasness in two of the six objects, *pitcher* and *driller*. For other four objects, test images are generated from an uniform distribution. The test set has 600 images (100 images per object).

Results Quantitative results are shown in Fig. 4. First, we show the performance of our NS based training images rendered using three initial distributions described earlier. We observe that the object detection performance drops by almost 30% and 10% for *pitcher* and *driller* objects respectively when there is object pose gap between training and test distributions.

Our NSO is able to automatically find the optimal pose distribution of the test set. NeRF then uses the optimal distribution to synthesize training data. The object detection model trained on the optimal data helps improve performance significantly; average precision accuracy for the *pticher* and *driller* objects have been improved by almost 30% and 10%, respectively. The blue lines in Fig. 4 represent the performance of NSO which fill the gap caused by distribution mismatch. Note there is similar significant improvement in experiments where there is gap in camera zoom when using the proposed NSO approach.

Table 1: Large scale YCB-synthetic experiments

Objects	**mAP**	master chef can	cracker box	sugar box	tomato soup can	mustard bottle	tuna fish can	pudding box	gelatin box	potted meat can	banana
NS	68.4	93.5	96.6	58.3	83.9	78.4	44.3	78.0	65.2	55.3	89.4
Auto-Sim	69.3	96.0	82.5	92.3	37.4	81.3	52.0	80.6	**79.4**	74.4	83.4
NSO	**82.1**	**98.5**	**98.4**	**98.2**	**81.8**	**90.5**	**64.6**	**84.1**	57.6	**92.2**	**91.6**
Objects	pitcher base	bleach cleanser	bowl	mug	power drill	wood block	scissor	large marker	large clamp	extra large clamp	foam brick
NS	29.0	49.9	78.7	46.8	89.3	97.8	**67.9**	42.9	47.8	72.7	**69.6**
Auto-Sim	7.7	81.5	78.3	60.0	83.2	95.6	64.1	41.5	46.6	**79.0**	57.9
NSO	**83.5**	**93.4**	**98.5**	**87.9**	**93.6**	**98.7**	55.3	**56.9**	**50.8**	78.6	68.2

We compare our NSO with LTS [41] and Auto-Sim [4] that use REINFORCE for non-differentiable simulator optimization (Fig. 4(a)(b)). We observe that on pose optimization, NSO achieves almost 34% and 11% improvement over LTS and Auto-Sim respectively on the pitcher object. We observe similar behaviour on zoom optimization. This highlights the gradients from differentiable NSO are more effective and generate better data than LTS and Auto-Sim.

Experiments on illumination optimization. To verify the effectiveness of Neural-Sim on illumination, we substitute vanilla NeRF model with NeRF-w. We conduct similar experiments as the pose and zoom experiments in Sec. 4.1 on illumination with YCB-synthetic dataset. The results show in Fig. 4(c). NSO has great performance on illumination optimization with 16% and 15% improvements on driller and banana objects respectively.

Large scale YCB-Synthetic dataset experiments Here we highlight the results of our large-scale experiments on the YCB-synthetic dataset. Experiments demonstrate that our proposed NSO approach helps to solve a drop in performance due to distribution shifts between the train and test sets. We use the same setting as previous experiment except we conduct object detection on all 21 objects on the YCB-Synthetic dataset. The test set has 2100 images (100 images per object). The experiment results are shown in Table. 1. Note that our proposed NSO achieves improvements of almost 14 % and 13 % points over NS and Auto-Sim baselines respectively.

4.2 YCB-in-the-wild dataset

To evaluate the performance of the proposed NS and NSO approaches on a real world dataset, we have created a real world *YCB-in-the-wild* dataset. The dataset has 6 YCB objects in it, which are same as in the *YCB-synthetic* dataset: *masterchef, cheezit, gelatin, pitcher, mug* and *driller*. We manually labelled both the object bounding box and the object pose. (More details in the appendix.)

To explore the performance of the NS and NSO under the training and test distribution gap on the *YCB-in-the-wild*, we use the same experiment setup as in Sec. 4.1. The test images are selected from *YCB-in-the-wild* and training images are synthesized by NeRF. The training data is generated under two categorical distributions: uniform distribution and a random bin as dominant bin.

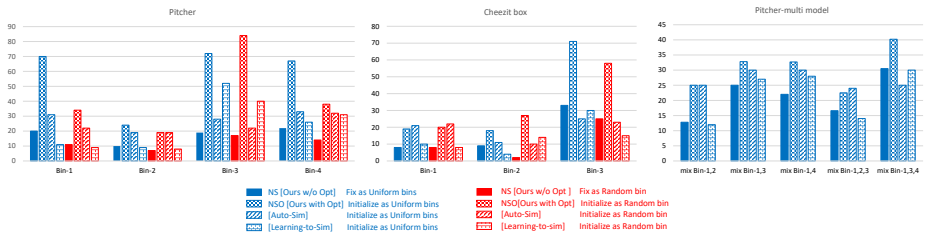

Fig. 5: Performance of Neural-Sim on the YCB-in-the-wild dataset. We observe that the Neural-Sim optimization (NSO) can consistently achieve 20% to 60% improvement in accuracy over our method without optimization (NS) case and large improvements over LTS (up to 58%) and Auto-Sim (up to 60%). Here each bin on $x-$axis represents bin from which test data is generated. We observe large improvement in both single-modal and multi-modal test data.

Quantitative results are provided in the Fig. 5. First we highlight the performance achieved by our NS approach to generate data according two different initial pose distributions. We observe that NS generated data helps achieve up to 30% in object detection accuracy on different objects starting from two different initial distributions. Moreover, our NSO approach achieves remarkable improvement in every experimental setup. For example, on *pitcher*, starting from uniform and random distributions, our optimization improve performance by almost 60%. Compared with other optimization methods LTS and Auto-Sim, we observe large improvement upto 58% improvement over LTS and 60% improvement over Auto-Sim on the pitcher object. This highlights three points. First, NeRF can be used to generate good data to solve object detection task in the wild; far more importantly, our Neural-Sim with bi-level optimization (NSO) approach can automatically find the optimal data that can help achieve remarkable improvements in accuracy on images captured in the wild. Third, the gradients from NSO are more effective and generate better data than LTS and Auto-Sim.

4.3 YCB Video dataset

To show the performance of the proposed NS and NSO approaches on a general real world dataset, we also conduct experiments on the YCB-Video dataset [47,26]. Each image in this dataset consists of multiple YCB objects (usually 3 to 6 different objects) in a real world scene. After sampling frames from 80 videos, $YCBV_{train}$ consists of over 2200 images, YCB-Video testset contains 900 images. Both train and test sets have all 21 YCB objects. In order to show the benefit of synthetic data, we create two different training scenarios (1) **Few-shot setting**, where we randomly select 10 and 25 images from ($YCBV_{train}$) to form different few shot training sets. (2) **Limited dataset setting**, where we randomly select 1%, 5%, 10% images from ($YCBV_{train}$) to form limited training sets.

Few-shot setting	0-shot	10-shot	25-shot
Only YCBV-train	N/A	0.45	0.49
train(pre)+ours (w/o opt)	2.3	3.9	4.6
train(pre)+ours (with opt)	4.5	4.9	4.9
Learning-to-sim (com)	N/A	12.4	22.5
Auto-Sim (com)	N/A	12.9	22.2
train(com)+ours (w/o opt)	N/A	12.2	21.0
train(com)+ours (with opt)	N/A	**13.1**	**23.0**

(a) Zero and few-shot setting (YCB-Video).

Percent of $YCBV_{train}$	0.01	0.05	0.1
Only YCBV-train	5.77	8.88	12.5
Only images to train NeRF	3.9	3.9	3.9
train(pre)+ours (w/o opt)	7.9	11.8	14.4
train(pre)+ours (with opt)	8.9	12.4	14.5
Learning-to-sim (com)	36.9	44.1	48.2
Auto-Sim (com)	37.1	43.7	48.3
train(com)+ours (w/o opt)	36.7	43.6	47.9
train(com)+ours (with opt)	**37.4**	**44.9**	**48.9**

(b) limited data setting (YCB-Video)

Table 2: YCB-Video performance. Observe large improvement of the proposed Neural-Sim approaches before and after optimization over the baselines.

Using a similar setting as in Sec. 4.2, we demonstrate performance of NS and NSO approaches starting from uniform distributions and compare with four baselines. First baseline-1 involves training RetinaNet using few-shot or limited training images from $YCBV_{train}$ data, and baseline-2 involves training RetinaNet using the images that were used to train NeRF. Baseline-3 is LTS and baseline-4 is Auto-Sim. Further, we combine the real-world few-shot or limited training images along with NeRF synthesized images during our Neural-Sim optimization steps. This *Combined* setting reduces the domain gap between synthetic and real data. All the models have been evaluated on YCB-Video testset.

For the normal Few-shot setting (rows 2, 3, 4 in Tab. 2(a)), NS starting from the uniform distribution achieves almost 3.45 and 4.11% improvement over the baseline-1 in 10 and 25 shots settings, respectively. Further, when we use NSO, we observe improvements of $4.45, 4.41\%$ over the baseline-1 and $1.0, 0.3\%$ improvements over the NS case in $10, 25$ shot settings respectively. We also observe almost 1.8% improvement in the zero-shot case. In addition, for the *Combined* Few-shot setting (rows 5,6,7,8 in Table. 2(a)), we observe similar large improvements in accuracy. We observe similar large performance improvements in the limited data settings (Table. 2(b)). Please refer to the appendix for more results and discussion including the results on ObjectNet[3] dataset.

4.4 Interpretability of Neural-Sim

We raise a question: does the Neural-Sim optimization provide interpretable results? In order to demonstrate this behavior, we conduct experiment on *YCB-in-the-wild* dataset illustrated in Fig 6. Generally, we find that no matter what the starting distributions the Neural-Sim approach used, the learned optimal ψ^* is always aligned with the test distribution. More visualizations in the appendix.

5 Discussion and Future Work

It has been said that "Data is food for AI"[35]. While computer vision has made wondrous progress in neural network models in the last decade, the data side has

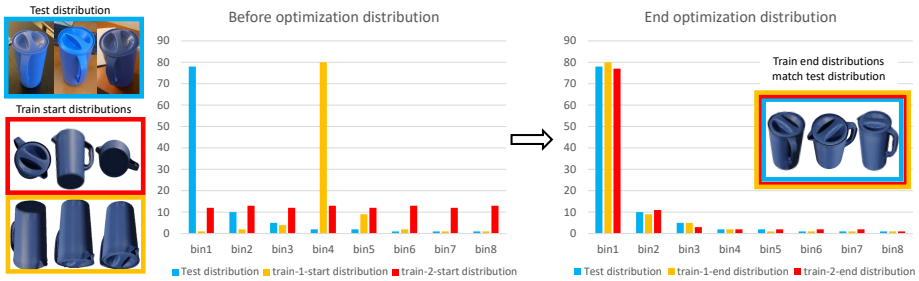

Fig. 6: NSO generates interpretable outputs. In the shown example, test images are sampled from distribution bin 1 as dominant bin. For Neural-Sim optimization (NSO), initial training pose distributions are uniform and bin 4 as dominant bin. Observe the bin distribution at the optimization - the final bin distribution at the end of Neural-Sim training matches with the test bin distribution.

seen much less advancement. There has been an explosion in the number and scale of datasets, but the **process** has evolved little, still requiring a painstaking amount of labor. Synthetic data is one of the most promising directions for transforming the data component of AI. While it has been used to show some impressive results, its wide-spread use has been limited, as creating good synthetic data still requires a large investment and specialized expertise.

We believe we have taken a big step towards making synthetic data easier to use for a broader population. By optimizing for how to synthesize data for training a neural network, we have shown big benefits over current synthetic data approaches. We have shown through extensive experiment that the data found by our system is better for training models. We have removed the need for any 3D modeling and for an expert to hand-tune the rendering parameters. This brings the promise of synthetic data closer for those that don't have the resources to use the current approaches.

We have handled camera pose, zoom and illumination; and our approach can be extended to other parameters (such as materials, etc.), by incorporating new advances in neural rendering. For future work, we hope to improve the ease of use of our approach, such as performing our optimization using lower quality, faster rendering using a smaller network for the neural rendering component, and then using the learned parameters to generate high quality data to train the final model. We hope that our work in this space will inspire future research.

Acknowledgments We thank Yen-Chen Lin for help on using the nerf-pytorch code. This work was supported in part by C-BRIC (one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA), DARPA (HR00112190134) and the Army Research Office (W911NF2020053). The authors affirm that the views expressed herein are solely their own, and do not represent the views of the United States government or any agency thereof.

References

1. Ali Jahanian, Lucy Chai, P.I.: On the "steerability" of generative adversarial networks. CoRR (2019)
2. Barbu, A., Mayo, D., Alverio, J., Luo, W., Wang, C., Gutfreund, D., Tenenbaum, J., Katz, B.: Objectnet: A large-scale bias-controlled dataset for pushing the limits of object recognition models. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019), https://proceedings.neurips.cc/paper/2019/file/97af07a14cacba681feacf3012730892-Paper.pdf
3. Barbu, A., Mayo, D., Alverio, J., Luo, W., Wang, C., Gutfreund, D., Tenenbaum, J., Katz, B.: Objectnet: A large-scale bias-controlled dataset for pushing the limits of object recognition models. Advances in neural information processing systems **32** (2019)
4. Behl, H.S., Baydin, A.G., Gal, R., Torr, P.H., Vineet, V.: Autosimulate:(quickly) learning synthetic data generation. In: European Conference on Computer Vision. pp. 255–271. Springer (2020)
5. Bi, S., Xu, Z., Srinivasan, P., Mildenhall, B., Sunkavalli, K., Hašan, M., Hold-Geoffroy, Y., Kriegman, D., Ramamoorthi, R.: Neural reflectance fields for appearance acquisition. arXiv preprint arXiv:2008.03824 (2020)
6. Brock, A., Donahue, J., Simonyan, K.: Large scale GAN training for high fidelity natural image synthesis. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=B1xsqj09Fm
7. Calli, B., Walsman, A., Singh, A., Srinivasa, S., Abbeel, P., Dollar, A.M.: Benchmarking in manipulation research: The ycb object and model set and benchmarking protocols. arXiv preprint arXiv:1502.03143 (2015)
8. Colson, B., Marcotte, P., Savard, G.: An overview of bilevel optimization. Annals of operations research **153**(1), 235–256 (2007)
9. Danilo Jimenez Rezende, S.M.: Variational inference with normalizing flows. In: ICML (2015)
10. Denninger, M., Sundermeyer, M., Winkelbauer, D., Zidan, Y., Olefir, D., Elbadrawy, M., Lodhi, A., Katam, H.: Blenderproc. arXiv preprint arXiv:1911.01911 (2019)
11. Devaranjan, J., Kar, A., Fidler, S.: Meta-sim2: Unsupervised learning of scene structure for synthetic data generation. In: European Conference on Computer Vision. pp. 715–733. Springer (2020)
12. Diederik Kingma, M.W.: Autoencoding variational bayes. In: ICLR (2014)
13. Doersch, C., Zisserman, A.: Sim2real transfer learning for 3d human pose estimation: motion to the rescue. In: NeurIPS (2019)
14. Dwibedi, D., Misra, I., Hebert, M.: Cut, paste and learn: Surprisingly easy synthesis for instance detection. In: ICCV (2017)
15. Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., Pontil, M.: Bilevel programming for hyperparameter optimization and meta-learning. In: International Conference on Machine Learning. pp. 1568–1577. PMLR (2018)
16. Gafni, G., Thies, J., Zollhofer, M., Nießner, M.: Dynamic neural radiance fields for monocular 4d facial avatar reconstruction. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 8649–8658 (2021)
17. Ganin, Y., Kulkarni, T., Babuschkin, I., Eslami, S.M.A., Vinyals, O.: Synthesizing programs for images using reinforced adversarial learning. In: ICML (2018)

18. Ge, Y., Abu-El-Haija, S., Xin, G., Itti, L.: Zero-shot synthesis with group-supervised learning. arXiv preprint arXiv:2009.06586 (2020)
19. Ge, Y., Xu, J., Zhao, B.N., Itti, L., Vineet, V.: Dall-e for detection: Language-driven context image synthesis for object detection. arXiv preprint arXiv:2206.09592 (2022)
20. Ge, Y., Zhao, J., Itti, L.: Pose augmentation: Class-agnostic object pose transformation for object recognition. In: European Conference on Computer Vision. pp. 138–155. Springer (2020)
21. Georgiev, I., Ize, T., Farnsworth, M., Montoya-Vozmediano, R., King, A., Lommel, B.V., Jimenez, A., Anson, O., Ogaki, S., Johnston, E., et al.: Arnold: A brute-force production path tracer. TOG (2018)
22. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. pp. 2672–2680 (2014)
23. Handa, A., Patraucean, V., Badrinarayanan, V., Stent, S., Cipolla, R.: Understanding real world indoor scenes with synthetic data. In: CVPR (2016)
24. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: ICCV (2017)
25. Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., Lerchner, A.: beta-vae: Learning basic visual concepts with a constrained variational framework. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France (2017)
26. Hodan, T., Michel, F., Brachmann, E., Kehl, W., GlentBuch, A., Kraft, D., Drost, B., Vidal, J., Ihrke, S., Zabulis, X., et al.: Bop: Benchmark for 6d object pose estimation. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 19–34 (2018)
27. Hodaň, T., Vineet, V., Gal, R., Shalev, E., Hanzelka, J., Connell, T., Urbina, P., Sinha, S., Guenter, B.: Photorealistic image synthesis for object instance detection. ICIP (2019)
28. Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A., Brox, T.: Flownet 2.0: Evolution of optical flow estimation with deep networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017. pp. 1647–1655 (2017). https://doi.org/10.1109/CVPR.2017.179, https://doi.org/10.1109/CVPR.2017.179
29. Jang, W., Agapito, L.: Codenerf: Disentangled neural radiance fields for object categories. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 12949–12958 (2021)
30. Kar, A., Prakash, A., Liu, M.Y., Cameracci, E., Yuan, J., Rusiniak, M., Acuna, D., Torralba, A., Fidler, S.: Meta-sim: Learning to generate synthetic datasets. In: ICCV (2019)
31. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. In: Proceedings of the IEEE international conference on computer vision. pp. 2980–2988 (2017)
32. Louppe, G., Cranmer, K.: Adversarial variational optimization of non-differentiable simulators. In: AISTATS (2019)
33. Martin-Brualla, R., Radwan, N., Sajjadi, M.S., Barron, J.T., Dosovitskiy, A., Duckworth, D.: Nerf in the wild: Neural radiance fields for unconstrained photo collections. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 7210–7219 (2021)
34. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis. In: European conference on computer vision. pp. 405–421. Springer (2020)

35. Ng, A.: Mlops: From model-centric to data-centric ai. https://www.deeplearning.ai/wp-content/uploads/2021/06/MLOps-From-Model-centric-to-Data-centric-AI.pdf
36. Park, K., Sinha, U., Barron, J.T., Bouaziz, S., Goldman, D.B., Seitz, S.M., Martin-Brualla, R.: Nerfies: Deformable neural radiance fields. ICCV (2021)
37. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: towards real-time object detection with region proposal networks. PAMI (2017)
38. Richter, S.R., Hayder, Z., Koltun, V.: Playing for benchmarks. In: ICCV (2017)
39. Richter, S.R., Vineet, V., Roth, S., Koltun, V.: Playing for data: Ground truth from computer games. In: European conference on computer vision. pp. 102–118. Springer (2016)
40. Ros, G., Sellart, L., Materzynska, J., Vázquez, D., López, A.M.: The SYNTHIA dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In: CVPR (2016)
41. Ruiz, N., Schulter, S., Chandraker, M.: Learning to simulate. In: ICLR (2019)
42. Shelhamer, E., Long, J., Darrell, T.: Fully convolutional networks for semantic segmentation. PAMI (2017)
43. Srinivasan, P.P., Deng, B., Zhang, X., Tancik, M., Mildenhall, B., Barron, J.T.: Nerv: Neural reflectance and visibility fields for relighting and view synthesis. In: CVPR (2021)
44. Tremblay, J., To, T., Birchfield, S.: Falling things: A synthetic dataset for 3d object detection and pose estimation. In: CVPR (2018)
45. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning (1992)
46. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning **8**(3), 229–256 (1992)
47. Xiang, Y., Schmidt, T., Narayanan, V., Fox, D.: Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. arXiv preprint arXiv:1711.00199 (2017)
48. Xiaogang Xu, Ying-Cong Chen, J.J.: View independent generative adversarial network for novel view synthesis. In: ICCV (2019)
49. Yang, D., Deng, J.: Learning to generate synthetic 3d training data through hybrid gradient. In: CVPR (2020)
50. Yen-Chen, L.: Nerf-pytorch. https://github.com/yenchenlin/nerf-pytorch/ (2020)
51. Zhang, X., Srinivasan, P.P., Deng, B., Debevec, P., Freeman, W.T., Barron, J.T.: Nerfactor: Neural factorization of shape and reflectance under an unknown illumination. ACM Transactions on Graphics (TOG) **40**(6), 1–18 (2021)
52. Zhang, Y., Song, S., Yumer, E., Savva, M., Lee, J., Jin, H., Funkhouser, T.A.: Physically-based rendering for indoor scene understanding using convolutional neural networks. In: CVPR (2017)