Robust and Scalable in-the-wild 3D Reconstruction for Realistic and Controllable Sensor Simulation

Anonymous Author(s)
Affiliation
Address
email

Abstract: Realistic simulation is key to enabling safe and scalable development of self-driving vehicles. A core component is simulating the sensors so that the entire autonomy system can be tested and learned in simulation. Sensor simulation involves modeling traffic participants, such as vehicles, with high quality appearance and articulated geometry, and rendering them in real time. The self-driving industry has typically employed artists to build these assets. However, this is expensive, slow and may not reflect reality. Instead, reconstructing assets automatically from sensor data collected in the wild would provide a better path to generating a diverse and large set that provides good real-world coverage. Nevertheless, current reconstruction approaches struggle on in-the-wild sensor data, due to its sparsity and noise. To tackle these issues, we present *CADSim*, which combines part-aware object-class priors via a small set of CAD models with differentiable rendering to automatically reconstruct vehicle geometry, including articulated wheels, with high-quality appearance. Our experiments show our approach recovers more accurate shapes from sparse data compared to existing approaches. Importantly, it also trains and renders efficiently. We demonstrate our reconstructed vehicles in several applications, including accurate testing of autonomy perception systems.

Keywords: 3D Reconstruction, CAD models, Sensor Simulation, Self-Driving

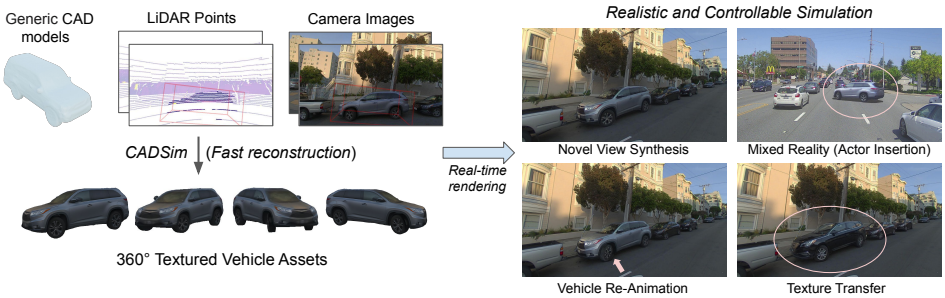

Figure 1: **CADSim recovers shape, appearance and illumination in a robust and scalable way from sensor observations**. The reconstructed assets are of high fidelity, part-aware, geometry-aligned and compatible with graphics engine, enabling efficient, realistic and controllable simulation.

1 Introduction

Robots, such as self-driving vehicles (SDVs), learn to navigate environments and interact safely with other agents through experience. It is critical for robots to handle rare or safety-critical situations which are challenging or dangerous to observe in the real world [1]. Simulation provides a solution for the robot to efficiently experience edge-case scenarios for both learning and evaluation in a safe and cost-effective manner. For proper testing and learning that covers the full space of possible scenarios, the simulator should have a diverse and large set of traffic participants. For example, vehicles have a wide variety of shapes, sizes, and appearances. We want to ensure that the perception and autonomy systems can detect them and act appropriately in all possible situations. Existing self-driving simulators have a limited set of objects, as each object is typically designed manually, with fully specified shape, appearance, etc. This is a time-consuming and expensive process that does

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.

not scale. In this work, we propose to leverage real sensor data collected by the SDV in the wild to automatically build an asset library for realistic sensor simulation. In-the-wild data is cost efficient and scalable to obtain, enabling collection of a diverse set of objects.

Specifically, given a sequence of sensor data (camera images and LiDAR point clouds) collected by an SDV, our goal is to automatically create digital replicas for all nearby vehicles in the scene and directly use them in a high-fidelity simulator to create many scenarios that never existed before. To achieve high quality sensor simulation, the reconstructed assets should (i) have precise shape and photorealistic appearance; (ii) be easily editable to create new variations from the original assets; (iii) be controllable, such as the ability to manipulate the vehicle to generate new behaviors (*e.g.*, wheels steering) and (iv) allow for real-time rendering. Out of past works using various 3D representations for reconstruction, such as point clouds [2, 3, 4, 5], voxels [6, 7, 8], or neural implicit representations [9, 10, 11, 12], the mesh representation best achieves the desired properties [13, 14]. In this paper we utilize mesh representations since they are widely used throughout simulation [15, 16, 17] and content creation systems [18, 19], are easy to manipulate [20, 21], suitable for rigging and animation [22, 23, 24, 25], support texturing and material properties [26, 27], and are fast to render [28].

Generating meshes of nearby objects from in-the-wild sensor data is a challenging task. The sensor data is sparse (< 30 views) and may be noisy. Furthermore, the objects of interest are observed from limited viewpoints and portions of the objects are unobserved, whereas we want to reconstruct the complete shape for simulation. Previous mesh reconstruction methods tackle this task as an energy minimization problem, where given an initial sphere [29] or "mean-shape" mesh template [30, 31], the goal is to optimize the mesh to be consistent with sensor data, along with some regularization. However, when operating on in-the-wild sensor data, these methods generate overly smoothed meshes that are inaccurate [29, 31], or have severe artifacts and self-intersecting edges [32]. We demonstrate in our experiments that this is due to poor initialization. Moreover, these approaches generate rigid meshes that cannot be controlled and lack articulation. We therefore propose a simple and effective solution—to leverage CAD models of the object class to provide priors for reconstruction. Not only do CAD models have high quality shape, they encode rich semantic information, such as where the wheels in a vehicle are and how they move. This enables better initialization and reconstruction of high-quality *articulated* meshes from real-world data.

CAD models alone are not sufficient for realistic sensor simulation, as we we want to create digital twins of any possible vehicle. Instead, we optimize our vehicle CAD model representation with a state-of-the-art differentiable renderer in an energy minimization framework. This allows us to faithfully reconstruct a diverse set of real-world vehicles, including their animatable parts (*e.g.*, wheels), from very sparse and noisy observations. We compare against a wide range of reconstruction methods, including recent neural rendering-based approaches, and show improved reconstruction performance on both camera and LiDAR sensor data. We show that our reconstructed vehicles can be used for a variety of applications such as novel view synthesis, texture transfer, mixed reality via injecting simulated actors into real data, and vehicle re-animation (Fig. 1). We also demonstrate that our approach creates a more realistic simulation environment for robots such as SDVs by evaluating perception models on simulated sensor data and showing small domain gap compared to real data.

2 Related Work

3D Representations for Sensor Simulation. Triangular meshes are currently the de facto standard in real-time graphics [13, 14, 15]. While fast, flexible, compact, and widely adopted in game engines and simulators [15], mesh-based assets are usually regarded as expensive to create due to the need for human artists. Surfels [33] have also been leveraged in LiDAR simulation for robotic applications [34, 35]. However, their lack of geometric structure makes them difficult to animate, edit, and compress. Recently, volumetric representations have experienced a surge in popularity [36, 37, 38] thanks to their simplicity, differentiability, and flexibility in modeling non-Lambertian effects [39, 40, 41]. These works can roughly be divided into neural volumetric rendering, such as NeRFs [39, 42, 43, 44, 45], and implicit surfaces such as signed distance fields [46, 47, 48] or occupancy fields [9, 49, 50]. Various methods have been proposed to speed up surface and volumetric rendering [51, 52, 53, 54, 55]. However, these approaches still lag behind traditional rasterization pipeline in terms of rendering speed and throughput. In our work, we use triangular meshes optimized from CAD models for efficient and accurate geometry modelling.

Multi-View 3D Reconstruction in the Wild. Despite the incredible success achieved by recent neural volumetric representations, these works are mainly demonstrated on synthetic datasets with

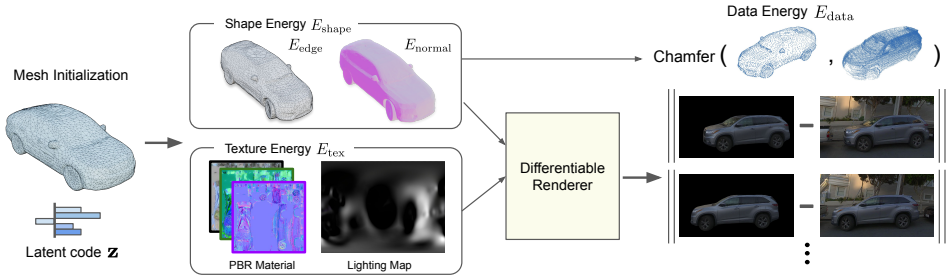

Figure 2: Overview of CADSim for 3D Reconstruction on in-the-wild data.

hundreds of (noise-free) images and dense coverage on the camera viewpoints [39, 56, 29]. However, the input is usually much sparser (*e.g.*, few views of single object with similar view points) and noisier (*e.g.*, localization, calibration error and LiDAR noise) for real world applications such as robotics and self-driving. To address this issue, recent works [57, 58, 59] proposed conditional models that require expensive pre-training on other scenes with multi-view images and camera pose annotations. Another recent line of work proposes learning mesh representations directly, either by deforming a generic mesh [26, 60, 32, 61, 62], or by leveraging a differentiable mesh extraction algorithm on top of an implicit representation [63, 64, 65]. Different from previous works, we show that simple mesh optimization with differentiable rendering is sufficient for high-fidelity reconstruction.

CAD Priors. Retrieving a 3D model conditioned on a single observation has been shown to be a strong baseline which outperforms more sophisticated early approaches to single-view reconstruction [66]. This motivates the use of imperfect but readily available 3D CAD models as a powerful prior for reconstruction [30, 67, 68, 69, 70]. Uy et al. [68, 69] encode sparse observations and use them to retrieve the most similar candidate from a CAD asset bank, which is then deformed to match the input. While promising, none of the approaches in this area have been demonstrated to work outside of synthetic [71] or motion-captured [72] datasets. Most do not model realistic texture, nor are they part-aware. Lu et al. [70] conduct a two-stage CAD-based deformation and handle the wheel separately. However, they recover the appearance only from single images without physics based material and lighting models, and do not optimize the articulation from data.

3 Method

Given camera images and/or LiDAR point clouds collected by a self-driving car, our goal is to automatically create digital replicas for all nearby vehicles in the scene and use them in a high-fidelity simulator. We build our model based on the observation that existing CAD models are equipped with detailed geometry and animatable parts, which can be used as a prior during the reconstruction process. Towards this goal, we propose an energy-based formulation that exploits CAD models, as well as visual and geometric cues from images and/or LiDAR for 3D reconstruction. We first describe our model representation and how it leverages CAD priors (Sec. 3.1). Then we present our energy-based model (Sec. 3.2), and how we conduct inference to generate the vehicle mesh (Sec. 3.3).

3.1 Model Representation

To faithfully simulate an object, we need to recover not only its geometry, but also its material. Towards this goal, we exploit *meshes* as our underlying representation since they best meet the desired properties of modern high-fidelity simulators, by being efficient to animate, edit, and render.

Mesh representation. A mesh $\mathcal{M} = (\mathbf{V}, \mathbf{F})$ is composed of a set of vertices $\mathbf{V} \in \mathbb{R}^{|V| \times 3}$ and faces $\mathbf{F} \in \mathbb{N}^{|F| \times 3}$, where the faces define the connectivity of the vertices. Our goal is to deform a mesh to match the observations from the sensor data. Typically, during deformation, the topology (*i.e.*, connectivity) of the mesh is fixed and only the vertices are "moving" [22, 31, 73, 20]. This strategy greatly simplifies the deformation process, yet at the same time constrains its representation power. For instance, if the original mesh topology is relatively simple (*e.g.*, a sphere [73], a "mean-shape" mesh template [31], etc.) and non-homeomorphic to the object of interest, the mesh may struggle to capture the fine-grained geometry (*e.g.*, side mirrors of a car).

CAD models as priors. To circumvent such limitations, we propose to incorporate shape priors from CAD models into the mesh reconstruction. One straightforward approach is to directly exploit the CAD model to initialize the mesh $\mathcal{M}_{\text{CAD}} = (\mathbf{V}_{\text{CAD}}, \mathbf{F}_{\text{CAD}})$. Since CAD models, by design,

respect the topology of real-world objects, the mesh may be able to model various details. One obvious caveat, however, is that there is no structure among the vertices. Each vertex may move freely in the 3D space during the optimization; and there is no guarantee that vertices belonging to a wheel will continue to form a cylinder. We thus further incorporate the *part information* from CAD models into the parameterization. This allows us to *inherently* preserve part geometry, as well as enable part manipulation (*e.g.*, changing the steering of the wheels).

We leverage the semantic part information from CAD models to partition the vehicle mesh $\mathcal{M}_{\text{CAD}}$ into a vehicle body $\{\mathbf{V}_{\text{body}}, \mathbf{F}_{\text{body}}\}$ and wheels $\{\mathbf{V}^{(k)}_{\text{wheel}}, \mathbf{F}^{(k)}_{\text{wheel}}\}_{i=1}^{K}$. K indicates the number of wheels a vehicle has. $K = 4$ for most vehicles but may be 6 or more for large trucks. We employ the fact that the wheels are usually the same within the same vehicle, and model all wheels with the same underlying mesh $\{\mathbf{V}_{\text{wheel}}, \mathbf{F}_{\text{wheel}}\}$, composited with their relative pose $\mathbf{T}^k$ to the vehicle origin. As there are a wide variety of vehicles with different wheel sizes and different relative positions to the vehicle body, we further add a scale factor $\mathbf{r} = [r_w, r_h, r_w]$ (wheel radius and thickness), and per-axle translation offsets $\mathbf{t}_{\text{front}}, \mathbf{t}_{\text{back}} \in \mathbb{R}^3$ with respect to the wheel origin. We can thus drastically reduce the degrees of freedom, enforce the shape of the wheel, and guarantee the relative positions of the wheels to be symmetric. Importantly, the front-axle wheels can be steered and do not necessarily align with the body. We hence parameterize the front wheels to have a yaw-relative orientation ρ. The full vehicle mesh model (see Figure 3) can thus be written as:

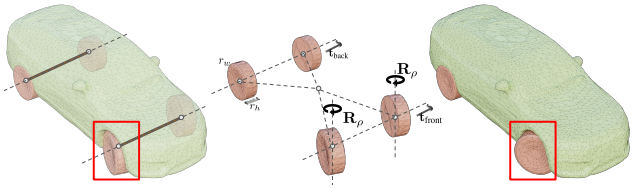

Figure 3: **Vehicle model used in CADSim**. For simplicity, we only show the offsets along the axle axis.

$$
\begin{aligned}
\mathbf{V}^k_{\text{wheel}}(\mathbf{r}, \mathbf{t}_{\text{front}}, \rho; \mathbf{V}_{\text{wheel}}) &= \mathbf{T}^{\text{k}}(\mathbf{R}_\rho \mathbf{r} \mathbf{V}_{\text{wheel}} + \mathbf{t}_{\text{front}}) \quad k \in \{1, 2\}, \quad \text{(front wheels)} \\
\mathbf{V}^k_{\text{wheel}}(\mathbf{r}, \mathbf{t}_{\text{back}}; \mathbf{V}_{\text{wheel}}) &= \mathbf{T}^{\text{k}}(\mathbf{r} \mathbf{V}_{\text{wheel}} + \mathbf{t}_{\text{back}}) \quad k \in \{3, 4, ..., K\}, \quad \text{(remaining wheels)} \\
\mathbf{V} &= \{\mathbf{V}_{\text{body}}, \mathbf{V}^{(k)}_{\text{wheel}}\} \quad k \in \{1, \ldots, K\},
\end{aligned} \tag{1}
$$

with $\mathbf{V}_{\text{body}}, \mathbf{r}, \mathbf{t}_{\text{back}}, \mathbf{t}_{\text{front}}, \rho$ being the free variables.

Extending to a CAD library. While a CAD model provides a useful prior for mesh initialization, it is likely that a single template mesh may not sufficiently cover a wide range of objects. We thus resort to a CAD model library that consists of various vehicle types. Inspired by [30, 70], we represent a large variety of vehicles with a compact, low dimensional code, which allows us to quickly determine the best CAD model to initialize from. We first parameterize a collection of the CAD models with Eq. (1), simplify them so that they have the same number of vertices and topology, and align them in a shared coordinate space with dense correspondences. Finally, we apply principal component analysis on vertex coordinates to obtain a shared low-dimensional code $\mathbf{z}$. Note that since all CAD models are parameterized with Eq. (1), the deformed mesh will reflect this part structure as well. Please refer to the supp. material for more details.

Appearance representation. In addition to geometry, our mesh representation must accurately capture how light interacts with its surface so that we can realistically reproduce sensor observations. Towards this goal, we parameterize the mesh appearance using a physically-based appearance representation. Specifically, we represent appearance using a micro-facet BRDF model with the differentiable split sum environment lighting [74] proposed by Munkberg et al. [65]. Since the topology of the mesh is fixed, we can build a one-to-one mapping relationship between each point on the mesh surface and each point in the 2D (u, v) space. We use a physically-based (PBR) material model from Disney [75], which contains a diffuse lobe $\mathbf{k}_d$ with an isotropic, specular GGX lobe [76].

3.2 Energy Formulation

The next step after the CAD model initialization is to optimize both the geometry and the appearance of the mesh such that it best matches the captured sensor data. Our approach builds upon recent success on differentiable rendering [77, 78, 79, 80], leveraging a differentiable renderer that takes as input variables like the sensor pose and our textured mesh representation, and outputs a realistic simulation of the object. Differentiable rendering allows all variables to be optimized based on the supervision provided by the input images.

Notation. Let $\mathcal{I} = \{\mathbf{I}_i\}_{1 \leq i \leq N}$ be the images captured at different timestamps and $\mathcal{P}$ be the aggregated LiDAR point clouds captured by a data collection platform, in our case a self-driving vehicle driving in the real-world. Let $\{\mathbf{M}_i\}_{1 \leq i \leq N}$ be the foreground segmentation mask of $\{\mathbf{I}_i\}_{1 \leq i \leq N}$ obtained from a off-the-shelf algorithm [29, 81]. Let $\mathcal{A} = \{\mathcal{D}, \mathcal{R}\}$ be the variables directly related to the appearance model (*i.e.*, the material and the lighting). We denote $\Pi = \{\mathbf{K}_i^{\text{Cam}}, \boldsymbol{\xi}_i^{\text{Cam}}, \boldsymbol{\xi}^{\text{LiDAR}}\}$ as the intrinsics and the extrinsics of the sensors, where $\boldsymbol{\xi} \in \mathfrak{se}(3)$ is the Lie-algebra associated with SE(3). We assume all cameras are pre-calibrated with known intrinsics. Let $\psi : (\mathcal{M}, \mathcal{A}, \Pi) \rightarrow (\mathbf{I}_\psi, \mathbf{M}_\psi)$ be the differentiable renderer where $\mathbf{I}_\psi$ and $\mathbf{M}_\psi$ denote the rendered RGB image and object mask.

Overall energy. We design an energy function with complementary terms which measure the geometry and appearance agreement between the observations and estimations (E_{data}), while regularizing the shape (E_{shape}) and appearance (E_{app}) to obey known priors:

$$\underset{\mathcal{M},\Pi,\mathcal{A}}{\text{argmin}}\{E_{\text{data}}(\mathcal{M}, \Pi, \mathcal{A}; \mathcal{I}, \mathcal{P}) + \lambda_{\text{shape}} E_{\text{shape}}(\mathcal{M}) + \lambda_{\text{app}} E_{\text{app}}(\mathcal{M}, \Pi, \mathcal{A}; \mathcal{I}, \mathcal{P})\}. \tag{2}$$

We now describe in more detail each energy term.

Data term. This energy encourages the estimated textured mesh to match the sensor data as much as possible. Specifically, it consists of three components:

$$E_{\text{data}}(\mathcal{M}, \Pi, \mathcal{A}; \mathcal{I}, \mathcal{P}) = E_{\text{color}}(\mathcal{M}, \Pi, \mathcal{A}; \mathcal{I}) + \lambda_{\text{mask}} E_{\text{mask}}(\mathcal{M}, \Pi; \mathcal{I}) + \lambda_{\text{lidar}} E_{\text{lidar}}(\mathcal{M}, \Pi; \mathcal{P}). \tag{3}$$

E_{color}, E_{mask} both enforce the rendering of the textured mesh to be close to the image observations. They, however, are complementary. E_{color} encourages the appearance of the rendered image to match the RGB observation and propagates the gradients to all variables including appearance variables $\mathcal{A}$, while E_{mask} measures the mask difference and only depends on the shape of the mesh as well as the camera poses. Following previous work [29, 65], we exploit smooth-ℓ_1 distance to measure the difference in RGB space and squared ℓ_2 for object masks:

$$E_{\text{color}} = \frac{1}{N} \sum_i^N \bar{\ell}_1 \left(\mathbf{I}_\psi(\mathcal{M}, \mathbf{K}_i, \boldsymbol{\xi}_i, \mathcal{A}), \mathbf{I}_i\right) \qquad E_{\text{mask}} = \frac{1}{N} \sum_i^N \|\mathbf{M}_\psi(\mathcal{M}, \mathbf{K}_i, \boldsymbol{\xi}_i) - \mathbf{M}_i\|_2^2, \tag{4}$$

with $\bar{\ell}_1$ the smooth-ℓ_1 norm [82] and N the number of images available.

E_{LiDAR} encourages the geometry of our mesh to match the aggregated LiDAR point clouds. Since minimizing point-to-surface distance is computationally expensive in practice, we adopt the popular Chamfer Distance (CD) to measure the similarity instead. We randomly select L points ($\mathcal{P}_{\text{s}}$) from the current mesh and compute the asymmetric CD of $\mathcal{P}_{\text{s}}$ with respect to the aggregated point cloud $\mathcal{P}$:

$$E_{\text{lidar}} = \text{CD}(\mathcal{P}_{\text{s}}, \mathcal{P}) = \frac{1}{|\mathcal{P}_{\text{s}}|} \sum_{x \in \mathcal{P}_{\text{s}}} \alpha_x \min_{y \in \mathcal{P}} \|x - y\|_2^2, \tag{5}$$

where α is an indicator function representing which LiDAR point is an outlier. We refer the reader to Sec. 3.3 for a discussion on how to estimate α.

Shape term. This energy encourages the deformed mesh to be smooth and the faces of the mesh to be uniformly distributed among the surfaces (so that the appearance would be less likely distorted). Specifically, E_{shape} is an addition of two shape regularizations [80, 61, 29]: a normal consistency term $E_{\text{normal}}(\mathbf{V})$ and an average edge length term $E_{\text{edge}}(\mathbf{V})$:

$$E_{\text{normal}}(\mathbf{V}) = \frac{1}{N_{\mathbf{F}}} \sum_{\mathbf{f} \in \mathbf{F}} \sum_{\mathbf{f}' \in \mathcal{N}(\mathbf{f})} ||\mathbf{n}(\mathbf{f}) \cdot \mathbf{n}(\mathbf{f}')||_2^2 \quad E_{\text{edge}}(\mathbf{V}) = \frac{1}{N_{\mathbf{E}}} \sum_{\mathbf{v} \in \mathbf{V}} \sum_{\mathbf{v}' \in \mathcal{N}(\mathbf{v})} ||\mathbf{v} - \mathbf{v}'||_2^2. \tag{6}$$

Here, $\mathbf{f} \in \mathbf{F}$ and $\mathbf{v} \in \mathbf{V}$ refer to a single face and a single vertex. $\mathcal{N}(\mathbf{f})$ and $\mathcal{N}(\mathbf{v})$ denote the neighboring faces of $\mathbf{f}$ and the neighboring vertices of $\mathbf{v}$ respectively. $N_{\mathbf{F}}$ and $N_{\mathbf{E}}$ are the number of neighboring face pairs and edges.

Appearance term. This energy exploits the facts: (1) the appearance of a vehicle will not change abruptly in most cases; instead, it varies in a smooth fashion; (2) it is primarily neutral, white lighting in the real world. Following prior art in intrinsic decomposition [83, 84], we adopt a sparsity term to penalize frequent color changes on the diffuse $\mathbf{k}_d$ and specular $\mathbf{k}_s$ terms. We also add a regularizer to penalize the environment light in gray scale.

$$E_{\text{app}} = \lambda_{\text{mat}} \left(\|\nabla \mathbf{k}_d\|_1 + \|\nabla \mathbf{k}_s\|_1\right) + \lambda_{\text{light}} \sum_{i=1}^{3} \|\mathbf{c}_i - \bar{\mathbf{c}}_i\|_1, \tag{7}$$

where $\nabla \mathbf{k}_d$ and $\nabla \mathbf{k}_s$ are the image gradients approximated by Sobel-Feldman operator [85]. $\mathbf{c}_i$ and $\bar{\mathbf{c}}_i$ are the light intensity values at R,G,B channels and the per-channel average intensities.

3.3 Inference

Our goal is to find the optimal mesh $\mathcal{M}^*$, appearance representation $\mathcal{A}^*$, and sensor poses Π^* such that the total energy is minimized.We adopt the following strategy for better inference convergence.

Initialization. A good initialization is required for non-convex optimization based methods to achieve good performance. If the energy model is initialized from CAD models that are very different from the observations, the mesh may not be able to fit the sensor data well. To handle a wide variety of vehicles, we leverage the learned low dimensional latent code $\mathbf{z}$ to estimate the initial coarse mesh $\mathcal{M}_{\text{init}} = (\mathbf{V}_{\text{init}}, \mathbf{F})$, where $\mathbf{V}_{\text{init}}$ is reconstructed from the optimized latent code $\mathbf{z}^*$ and

$$z^* = \underset{z}{\operatorname{argmin}}\, \lambda_{\text{mask}} E_{\text{mask}}(\mathcal{M}, \Pi; \mathcal{I}) + \lambda_{\text{lidar}} E_{\text{lidar}}(\mathcal{M}, \Pi; \mathcal{P}) + \lambda_{\text{shape}} E_{\text{shape}}(\mathbf{V}). \tag{8}$$

We note that we focus on the geometry and only optimize the latent code. The latent code is initialized from $\mathbf{0}$, and the sensor poses are obtained from coarse calibration and fixed. Since the low dimensional latent code already smooths out instance-specific details, the optimization is much more robust and less sensitive to initialization.

Efficient and robust inference. Given the initialization $\mathcal{M}_{\text{init}}$, we jointly optimize the vertices $\mathbf{V}$, appearance variables $\mathcal{A}$, and sensor poses Π as described in Eq. (2). To save computation, we uniformly sample L points on the current mesh at each iteration to compute the LiDAR energy $E_{\text{LiDAR}} = \text{CD}(\mathcal{P}_s, \mathcal{P})$, where $\mathcal{P}_s \sim \mathcal{M}$ and $|\mathcal{P}| = L$. To handle LiDAR outliers, we estimate the indicator function α by only calculating the Chamfer distance for the top p percentage of point pairs with the smallest distance. We find it is sufficient to handle point outliers in most cases. For the sensor extrinsics, we optimize the 6D rotation representation [86] and 3D translation. It is a well known issue that optimizing vertex positions with adverse gradient descent steps can cause self-intersections that is irrecoverable or even exacerbated in further steps [61, 32]. Therefore, following Nicolet et al. [32], we use preconditioned gradient descent steps to bias towards smooth solutions at each iterations, which enables faster convergence and higher mesh quality (*i.e.*, more fine-grained details).

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate our approach on two *in-the-wild* datasets: Multi-view Marketplace Cars (MVMC) dataset [29] and PandaVehicle dataset [87]. MVMC consists of a diverse set of vehicles captured under various illumination conditions. Each vehicle comes with around 10 views and rough camera pose. Following [29], we evaluate our model as well as the baselines on the selected 20 vehicles. PandaVehicle is a dataset we derived from PandaSet [87]. It consists of 200+ vehicles, along with their corresponding multi-view images and aggregated LiDAR point clouds. We manually inspect and select 10 high-quality vehicles for evaluation based on the accuracy of LiDAR-camera calibration. On average, each vehicle is captured from 24 camera views.

Baselines. We compare our apporach with three typese of 3D reconstruction methods: (1) *Neural radiance fields*: NeRF++ [42], and Instant-NGP [45]. (2) *Implicit surface representations*: NeRS [29], NVDiffRec [65], and NeuS [47]. (3) *Explicit geometry-based approaches*: single-image view-warping (SI-ViewWarp) [88], which warps the source image to the novel target view using depth estimated by LiDAR points; and multi-image view-warping (MI-ViewWarp), which blends multiple source images to the target view. We also compare against SAMP [30], a CAD model mesh optimization approach which leverages SDF-aligned CAD models for joint pose and shape optimization. See supp. material.

Evaluation settings and metrics. We evaluate reconstruction quality by performing camera and LiDAR simulation with the reconstructed vehicle. For camera simulation (*i.e.*, novel view synthesis (NVS)), we adopt mean-square error (MSE), peak signal-to-noise ratio (PSNR), SSIM [89], LPIPS [90] and cleaned FID [91]. To focus on the reconstructed vehicle itself, we only evaluate the foreground pixels using the predicted segmentation mask [81]. For LiDAR simulation, we place the reconstructed vehicle mesh in its original location and perform ray-casting to generate a simulated point cloud. We compare against held out real LiDAR points. We evaluate what fraction of real LiDAR points do not have a corresponding simulated point (*i.e.*, coverage), the average per-ray ℓ_2 error, Chamfer and Hausdorff distance. See supp. for additional details.

4.2 Reconstruction Quality

Novel View Synthesis on PandaVehicle and MVMC. We report NVS results on PandaVehicle in Tab. 1 and on MVMC in Tab. 2, along with reconstruction time (T) and rendering speed (*i.e.*,

Method	MSE ↓	PSNR ↑	SSIM ↑	LPIPS ↓	T (hour)	FPS
NeRF++ [Zhang et al., 2020]	0.0138	20.86	0.611	0.300	4.70	0.05
Instant-NGP [Müller et al., 2022]	0.0095	21.68	0.641	0.319	0.05	1.14
NeRS [Zhang et al., 2021]	0.0176	18.49	0.562	0.265	1.37	3.23
NVDiffRec [Munkberg et al., 2021]	0.0114	20.46	0.593	0.396	1.07	51.2
NeuS [Wang et al., 2021]	0.0115	21.37	0.640	0.247	6.25	0.02
SI-ViewWarp [Tulsiani et al., 2018]	0.0233	17.51	0.514	0.371	–	1.67
SAMP [Engelmann et al., 2017]	0.0144	19.52	0.628	0.283	0.09	71.4
CADSim (ours)	**0.0087**	**21.72**	**0.674**	**0.220**	0.13	49.6

Table 1: Evaluation of novel-view synthesis (left → front-left camera) on PandaSet. We report the average GPU hour T for reconstruction per actor and rendering FPS on single RTX A5000. Underline denotes fast reconstruction (*i.e.*, $<$ 10 minutes) and real-time rendering (FPS $>$ 30).

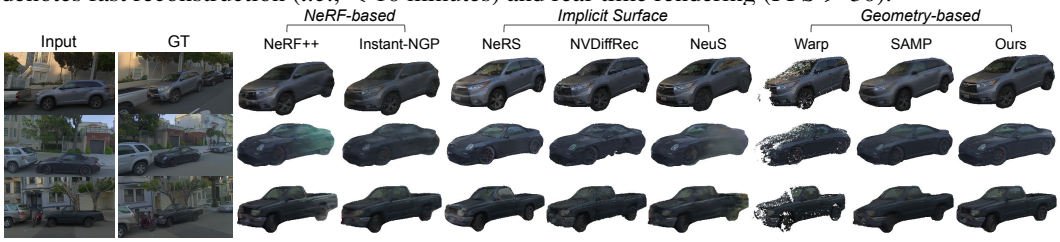

Figure 4: **Qualitative results on PandaVehicle for novel view synthesis**. Compared to existing reconstruction approaches, CADSim produces more robust and realistic results on large extrapolation.

frame per second (FPS)). For PandaVehicle we train each approach using images only from the left camera, and evaluate on front-left and front cameras. For MVMC we evaluate each approach via a cross-validation approach, where we hold out one image each time. CADSim achieves better performance across all image quality metrics, especially LPIPS. LPIPS measures how neural nets perceive images. This suggests that our simulation results are perceived by the ML models to be more similar to real-world images (see Sec. 4.3 for more analysis).

With CAD priors, we can reconstruct from much fewer views compared to NeRF-based or implicit-surface based approaches. Compared to geometry-based warping methods, our differentiable renderer and material model enable much higher photorealism. Qualitative results on PandaVehicle in Fig. 4 further confirms this. NeRF-based approaches usually lead to visible artifacts with large view extrapolation and sparse image observations. Implicit surface approaches perform better by alleviating shape-radiance ambiguity [42], but still have artifacts due to lower quality geometries. The other geometry-based approaches have better geometry, but have blurry appearance quality or missing pixels compared to CADSim. To further demonstrate the value of our improved geometry, we compare the NVS results using our reconstructed mesh against a NeRS mesh with various appearance representations in the supp. material. Additionally, neural rendering based approaches such as NeuS and NeRF++ are usually slow to train and render. While Instant-NGP [45] reduces training time significantly, it does not support real-time rendering.

Method	MSE ↓	PSNR ↑	SSIM ↑	LPIPS ↓	FIDS ↓
NeRS [29]	0.0254	16.5	0.720	0.172	60.9
CADSim	**0.0188**	**17.7**	**0.751**	**0.147**	**48.1**

Table 2: Evaluation of NVS on MVMC [29].

Geometry	L_2 error ↓	Hit rate ↑	Chamfer↓	Hausdorff↓
NeRS [29]	0.171	94.3%	0.249	1.028
NVDiffRec [65]	0.320	**98.2**%	0.439	1.708
NeuS [47]	0.367	90.3%	0.424	1.151
SAMP [30]	0.158	94.8%	0.256	1.043
CADSim (ours)	**0.151**	96.3%	**0.245**	**0.972**

Table 3: Comparison of LiDAR rendering metrics.

LiDAR simulation on PandaVehicle. We now evaluate the geometry performance of the reconstructed meshes for LiDAR simulation. As shown in Tab. 3, our approach is better than or comparable to prior art across all metrics, suggesting that our meshes are more complete and more accurate than existing methods. NVDiffRec has the highest hit rate, yet performs bad on the rest of the metrics. This is because their generated meshes are noisy, hence covering many LiDAR points.

Ablation study. To showcase the importance of CAD initalization, we initalize the mesh with the following alternatives: a unit sphere, a rescaled ellipsoid, and geometries from either SAMP [30] or NeRS [29]. As shown in Fig. 5, exploiting CAD priors leads to faster convergence and better performance. Our reconstruction is able to capture fine-grained details such as wheel size and rotation, rear-view mirrors, etc. We also evaluate our reflectance model choice [76] in the supp. material. Our physics-based material and lighting model performs the best and generalizes well to novel views.

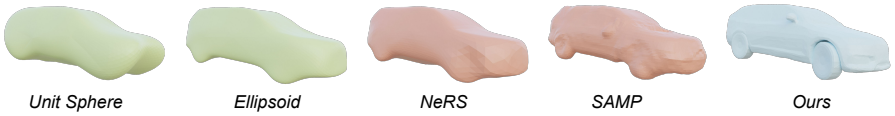

Figure 5: CAD priors are crucial for achieving high-fidelity shape in non-convex optimization.

	Left camera → Front-left camera				*Left camera → Front camera*			
	Blending [16]		Copy-Paste		Blending [16]		Copy-Paste	
	Det. (IoU)	Segm. (IoU)	Det. (IoU)	Segm. (IoU)	Det. (IoU)	Segm. (IoU)	Det. (IoU)	Segm. (IoU)
Instant-NGP	86.77	86.86	80.57	80.81	66.37	65.80	41.65	40.13
NeuS	93.97	**94.22**	92.82	93.63	66.26	65.09	66.01	64.66
SAMP	90.39	89.58	90.04	89.73	82.82	79.78	83.32	**81.79**
CADSim (ours)	**94.15**	93.92	**93.71**	**93.72**	**83.99**	**80.94**	**83.45**	81.42

Table 4: Evaluation of downstream perception tasks (*i.e.*, object detection, instance segmentation) on camera simulation, metric agreement with real data. Our assets lead to smaller domain gaps.

4.3 Realistic and Controllable Simulation

We now showcase applying CADSim for accurate camera simulation to evaluate perception models, and to perform realistic vehicle insertion. CADSim also supports texture transfer naturally, allowing us to expand simulation assets easily. Please see additional details and results in supp. material.

Downstream evaluation on camera simulation. To verify if CADSim helps reduce domain gap for downstream perception tasks, we evaluate object detection and instance segmentation algorithms [92, 81] on simulated camera images at novel views. In particular, we consider two setups: rendered and blended [16] and copy-pasted. We compute the instance-level Intersection over Union (IoU) of predicted bounding box and segmentation mask between simulated image and real images. This detection/segmentation agreement metric (the larger the better) indicates how well we can use the simulation to test existing perception systems. As shown in Tab. 4, using CADSim assets usually leads to the largest agreement with real images under different settings.

Realistic animated vehicle insertion. Given the original scenario and sensor data and an inserted actor trajectory, we produce consistent multi-sensor simulation with the rendered asset. In Fig. 6, we generate safety-critical scenarios by inserting a new actor turning into our lane (left half) and an actor slowing down in the adjacent lane, preventing the SDV from lane-changing (right half). The rendered asset is seamlessly blended into the original scenario and has physically realistic wheel rotation and movement. This enables diverse scenario generation for end-to-end autonomy testing.

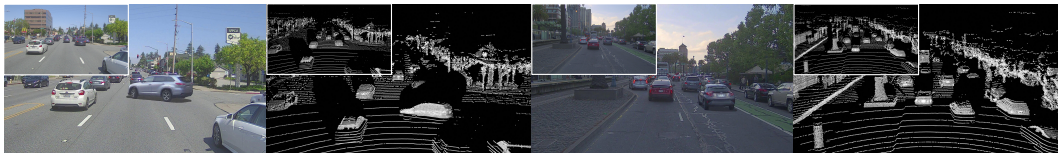

Figure 6: **Multi-sensor simulation on safety-critical scenarios with animated vehicle insertion.** The original camera and LiDAR observations captured in the real world are at the top-left corners.

5 Limitations and Conclusion

CADSim's main assumption is that it requires CAD models for the object class of interest. We note that CAD models are readily available for most object classes, and that our approach only requires encoding semantic priors for a single CAD asset, as our energy-model optimization allows for transfer of these priors to other assets of the same class. We demonstrate our approach on other classes (*e.g.*, motorcycles and cones) in the supplementary material. Additionally, our approach relies on either good segmentation masks or LiDAR points to reconstruct accurate shapes. While CADSim is more robust than other reconstruction approaches, it might still produce unexpected results on extremely noisy data (*e.g.*, large calibration misalignment, highly corrupted object masks).

In this paper, we proposed to leverage in-the-wild camera and LiDAR data to reconstruct objects such as vehicles. Towards this goal, we designed CADSim, which leverages geometry and semantic cues from CAD models with differentiable rendering, to generate meshes with high quality geometry and appearance. These cues also enable our approach to generate *articulated* and *editable* meshes, enabling endless creation of new shapes, textures, and animations for simulation. We demonstrated that integrating our meshes into a camera simulation system can more effectively evaluate robot perception algorithms than existing reconstruction methods, bringing smaller domain gap in simulation.

References

[1] S. Tan, K. Wong, S. Wang, S. Manivasagam, M. Ren, and R. Urtasun. SceneGen: Learning to generate realistic traffic scenes. In *CVPR*, 2021.

[2] W. Yuan, T. Khot, D. Held, C. Mertz, and M. Hebert. PCN: Point completion network. In *3DV*, pages 728–737. IEEE, 2018.

[3] E. Insafutdinov and A. Dosovitskiy. Unsupervised learning of shape and pose with differentiable point clouds. *NeurIPS*, 2018.

[4] C.-H. Lin, C. Kong, and S. Lucey. Learning efficient point cloud generation for dense 3d object reconstruction. In *AAAI*, 2018.

[5] J. Gu, W.-C. Ma, S. Manivasagam, W. Zeng, Z. Wang, Y. Xiong, H. Su, and R. Urtasun. Weakly-supervised 3D shape completion in the wild. In *ECCV*, 2020.

[6] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *NeurIPS*, 2016.

[7] J. Wu, Y. Wang, T. Xue, X. Sun, B. Freeman, and J. Tenenbaum. MarrNet: 3D shape reconstruction via 2.5D sketches. *NeurIPS*, 2017.

[8] J. Wu, C. Zhang, X. Zhang, Z. Zhang, W. T. Freeman, and J. B. Tenenbaum. Learning shape priors for single-view 3d completion and reconstruction. In *ECCV*, 2018.

[9] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *CVPR*, 2019.

[10] Q. Xu, W. Wang, D. Ceylan, R. Mech, and U. Neumann. Disn: Deep implicit surface network for high-quality single-view 3d reconstruction. 2019.

[11] C.-H. Lin, C. Wang, and S. Lucey. Sdf-srn: Learning signed distance 3d object reconstruction from static images. *NeurIPS*, 2020.

[12] S. Duggal, Z. Wang, W.-C. Ma, S. Manivasagam, J. Liang, S. Wang, and R. Urtasun. Mending neural implicit modeling for 3d vehicle reconstruction in the wild. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2022.

[13] A. Sanders. *An introduction to Unreal engine 4*. AK Peters/CRC Press, 2016.

[14] B. O. Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. URL http://www.blender.org.

[15] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In *CoRL*, 2017.

[16] Y. Chen, F. Rong, S. Duggal, S. Wang, X. Yan, S. Manivasagam, S. Xue, E. Yumer, and R. Urtasun. Geosim: Realistic video simulation via geometry-aware composition for self-driving. In *CVPR*, 2021.

[17] J. Tu, M. Ren, S. Manivasagam, M. Liang, B. Yang, R. Du, F. Cheng, and R. Urtasun. Physically realizable adversarial examples for lidar object detection. In *CVPR*, 2020.

[18] H.-T. D. Liu, M. Tao, and A. Jacobson. Paparazzi: surface editing by way of multi-view image processing. *ACM TOG*, 2018.

[19] H.-T. D. Liu and A. Jacobson. Cubic stylization. *ACM TOG*, 2019.

[20] W. Yifan, N. Aigerman, V. G. Kim, S. Chaudhuri, and O. Sorkine-Hornung. Neural cages for detail-preserving 3d deformations. In *CVPR*, 2020.

[21] T. Jakab, R. Tucker, A. Makadia, J. Wu, N. Snavely, and A. Kanazawa. Keypointdeformer: Unsupervised 3d keypoint discovery for shape control. In *CVPR*, 2021.

[22] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black. SMPL: A skinned multi-person linear model. *SIGGRAPH Asia*, 2015.

[23] S. Zuffi, A. Kanazawa, D. W. Jacobs, and M. J. Black. 3D menagerie: Modeling the 3D shape and pose of animals. In *CVPR*, 2017.

[24] G. Pavlakos, V. Choutas, N. Ghorbani, T. Bolkart, A. A. A. Osman, D. Tzionas, and M. J. Black. Expressive body capture: 3D hands, face, and body from a single image. In *CVPR*, 2019.

[25] Z. Yang, S. Manivasagam, M. Liang, B. Yang, W.-C. Ma, and R. Urtasun. Recovering and simulating pedestrians in the wild. In *CoRL*, 2020.

[26] H. Kato, Y. Ushiku, and T. Harada. Neural 3D mesh renderer. In *CVPR*, 2018.

[27] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson, and G. Gkioxari. Accelerating 3d deep learning with PyTorch3D. *arXiv preprint arXiv:2007.08501*, 2020.

[28] J. D. Foley, F. D. Van, A. Van Dam, S. K. Feiner, J. F. Hughes, and J. Hughes. *Computer graphics: principles and practice*. Addison-Wesley, 1996.

[29] J. Y. Zhang, G. Yang, S. Tulsiani, and D. Ramanan. NeRS: Neural reflectance surfaces for sparse-view 3d reconstruction in the wild. In *NeurIPS*, 2021.

[30] F. Engelmann, J. Stückler, and B. Leibe. SAMP: shape and motion priors for 4d vehicle reconstruction. In *WACV*, 2017.

[31] A. Kanazawa, S. Tulsiani, A. A. Efros, and J. Malik. Learning category-specific mesh reconstruction from image collections. In *ECCV*, 2018.

[32] B. Nicolet, A. Jacobson, and W. Jakob. Large steps in inverse rendering of geometry. *ACM TOG*, 40(6):1–13, 2021.

[33] H. Pfister, M. Zwicker, J. Van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 335–342, 2000.

[34] Z. Yang, Y. Chai, D. Anguelov, Y. Zhou, P. Sun, D. Erhan, S. Rafferty, and H. Kretzschmar. Surfelgan: Synthesizing realistic sensor data for autonomous driving. In *CVPR*, 2020.

[35] S. Manivasagam, S. Wang, K. Wong, W. Zeng, M. Sazanovich, S. Tan, B. Yang, W.-C. Ma, and R. Urtasun. LiDARsim: Realistic lidar simulation by leveraging the real world. In *CVPR*, 2020.

[36] V. Sitzmann. Awesome implicit representations - a curated list of resources on implicit neural representations.

[37] A. Tewari, J. Thies, B. Mildenhall, P. Srinivasan, E. Tretschk, W. Yifan, C. Lassner, V. Sitzmann, R. Martin-Brualla, S. Lombardi, et al. Advances in neural rendering. In *Computer Graphics Forum*, volume 41, pages 703–735. Wiley Online Library, 2022.

[38] Y. Xie, T. Takikawa, S. Saito, O. Litany, S. Yan, N. Khan, F. Tombari, J. Tompkin, V. Sitzmann, and S. Sridhar. Neural fields in visual computing and beyond. In *Computer Graphics Forum*, volume 41, pages 641–676. Wiley Online Library, 2022.

[39] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[40] V. Sitzmann, J. Martel, A. Bergman, D. Lindell, and G. Wetzstein. Implicit neural representations with periodic activation functions. *NeurIPS*, 2020.

[41] M. Boss, A. Engelhardt, A. Kar, Y. Li, D. Sun, J. T. Barron, H. Lensch, and V. Jampani. SAMURAI: Shape and material from unconstrained real-world arbitrary image collections. *arXiv preprint arXiv:2205.15768*, 2022.

[42] K. Zhang, G. Riegler, N. Snavely, and V. Koltun. NeRF++: Analyzing and improving neural radiance fields. *arXiv*, 2020.

[43] X. Zhang, P. P. Srinivasan, B. Deng, P. Debevec, W. T. Freeman, and J. T. Barron. NeRFactor: Neural factorization of shape and reflectance under an unknown illumination. *arXiv*, 2021.

[44] M. Tancik, B. Mildenhall, T. Wang, D. Schmidt, P. P. Srinivasan, J. T. Barron, and R. Ng. Learned initializations for optimizing coordinate-based neural representations. In *CVPR*, 2021.

[45] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv preprint arXiv:2201.05989*, 2022.

[46] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *CVPR*, 2019.

[47] P. Wang, L. Liu, Y. Liu, C. Theobalt, T. Komura, and W. Wang. NeuS: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. In *NeurIPS*, 2021.

[48] K. Zhang, F. Luan, Q. Wang, K. Bala, and N. Snavely. PhySG: Inverse rendering with spherical gaussians for physics-based material editing and relighting. In *CVPR*, 2021.

[49] L. Yariv, Y. Kasten, D. Moran, M. Galun, M. Atzmon, R. Basri, and Y. Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. In *NeurIPS*, 2020.

[50] M. Oechsle, S. Peng, and A. Geiger. UNISURF: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *ICCV*, 2021.

[51] R. Chabra, J. E. Lenssen, E. Ilg, T. Schmidt, J. Straub, S. Lovegrove, and R. Newcombe. Deep local shapes: Learning local SDF priors for detailed 3d reconstruction. In *ECCV*, 2020.

[52] T. Takikawa, J. Litalien, K. Yin, K. Kreis, C. Loop, D. Nowrouzezahrai, A. Jacobson, M. McGuire, and S. Fidler. Neural geometric level of detail: Real-time rendering with implicit 3D shapes. In *CVPR*, 2021.

[53] P. Hedman, P. P. Srinivasan, B. Mildenhall, J. T. Barron, and P. Debevec. Baking neural radiance fields for real-time view synthesis. *ICCV*, 2021.

[54] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. *ICCV*, 2021.

[55] A. Yu, S. Fridovich-Keil, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa. Plenoxels: Radiance fields without neural networks. *arXiv preprint arXiv:2112.05131*, 2021.

[56] M. Niemeyer, J. T. Barron, B. Mildenhall, M. S. Sajjadi, A. Geiger, and N. Radwan. RegNeRF: Regularizing neural radiance fields for view synthesis from sparse inputs. In *CVPR*, pages 5480–5490, 2022.

[57] J. Chibane, A. Bansal, V. Lazova, and G. Pons-Moll. Stereo radiance fields (srf): Learning view synthesis for sparse views of novel scenes. In *CVPR*, pages 7911–7920, 2021.

[58] A. Chen, Z. Xu, F. Zhao, X. Zhang, F. Xiang, J. Yu, and H. Su. Mvsnerf: Fast generalizable radiance field reconstruction from multi-view stereo. In *ICCV*, pages 14124–14133, 2021.

[59] A. Trevithick and B. Yang. GRF: Learning a general radiance field for 3d representation and rendering. In *ICCV*, pages 15182–15192, 2021.

[60] W. Chen, H. Ling, J. Gao, E. Smith, J. Lehtinen, A. Jacobson, and S. Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. *NeurIPS*, 32, 2019.

[61] W. Chen, J. Litalien, J. Gao, Z. Wang, C. F. Tsang, S. Khamis, O. Litany, and S. Fidler. DIB-R++: Learning to predict lighting and material with a hybrid differentiable renderer. *arXiv*, 2021.

[62] T. Monnier, M. Fisher, A. A. Efros, and M. Aubry. Share with thy neighbors: Single-view reconstruction by cross-instance consistency. *arXiv preprint arXiv:2204.10310*, 2022.

[63] M. Guo, A. Fathi, J. Wu, and T. Funkhouser. Object-centric neural scene rendering. 2021.

[64] T. Shen, J. Gao, K. Yin, M.-Y. Liu, and S. Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3D shape synthesis. *NeurIPS*, 34, 2021.

[65] J. Munkberg, J. Hasselgren, T. Shen, J. Gao, W. Chen, A. Evans, T. Müller, and S. Fidler. Extracting triangular 3D models, materials, and lighting from images. *arXiv preprint arXiv:2111.12503*, 2021.

[66] M. Tatarchenko, S. R. Richter, R. Ranftl, Z. Li, V. Koltun, and T. Brox. What do single-view 3d reconstruction networks learn? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3405–3414, 2019.

[67] C. Jiang, J. Huang, A. Tagliasacchi, and L. J. Guibas. ShapeFlow: Learnable deformation flows among 3d shapes. *NeurIPS*, 33:9745–9757, 2020.

[68] M. A. Uy, J. Huang, M. Sung, T. Birdal, and L. Guibas. Deformation-aware 3d model embedding and retrieval. In *ECCV*, pages 397–413. Springer, 2020.

[69] M. A. Uy, V. G. Kim, M. Sung, N. Aigerman, S. Chaudhuri, and L. J. Guibas. Joint learning of 3d shape retrieval and deformation. In *CVPR*, pages 11713–11722, 2021.

[70] F. Lu, Z. Liu, X. Song, D. Zhou, W. Li, H. Miao, M. Liao, L. Zhang, B. Zhou, R. Yang, et al. PerMO: Perceiving more at once from a single image for autonomous driving. *arXiv preprint arXiv:2007.08116*, 2020.

[71] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. ShapeNet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[72] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black. SMPL: A skinned multi-person linear model. *ACM transactions on graphics (TOG)*, 34(6):1–16, 2015.

[73] S. Goel, A. Kanazawa, and J. Malik. Shape and viewpoint without keypoints. In *ECCV*, 2020.

[74] B. Karis. Real shading in unreal engine 4. *Proc. Physically Based Shading Theory Practice*, 4 (3):1, 2013.

[75] S. McAuley, S. Hill, N. Hoffman, Y. Gotanda, B. Smits, B. Burley, and A. Martinez. Practical physically-based shading in film and game production. In *ACM SIGGRAPH 2012 Courses*, pages 1–7. 2012.

[76] B. Walter, S. R. Marschner, H. Li, and K. E. Torrance. Microfacet models for refraction through rough surfaces. *Rendering techniques*, 2007:18th, 2007.

[77] H. Kato, Y. Ushiku, and T. Harada. Neural 3d mesh renderer. In *CVPR*, 2018.

[78] S. Liu, T. Li, W. Chen, and H. Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. *ICCV*, 2019.

[79] W. Chen, J. Gao, H. Ling, E. Smith, J. Lehtinen, A. Jacobson, and S. Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. In *NeurIPS*, 2019.

[80] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson, and G. Gkioxari. Accelerating 3D deep learning with PyTorch3D. *arXiv:2007.08501*, 2020.

[81] A. Kirillov, Y. Wu, K. He, and R. Girshick. PointRend: Image segmentation as rendering. In *CVPR*, 2020.

[82] R. Girshick. Fast R-CNN. In *ICCV*, pages 1440–1448, 2015.

[83] E. H. Land and J. J. McCann. Lightness and retinex theory. *Josa*, 1971.

[84] W.-C. Ma, H. Chu, B. Zhou, R. Urtasun, and A. Torralba. Single image intrinsic decomposition without a single intrinsic image. In *ECCV*, 2018.

[85] I. Sobel and G. Feldman. A 3x3 isotropic gradient operator for image processing. *a talk at the Stanford Artificial Project*, pages 271–272, 1968.

[86] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5745–5753, 2019.

[87] P. Xiao, Z. Shao, S. Hao, Z. Zhang, X. Chai, J. Jiao, Z. Li, J. Wu, K. Sun, K. Jiang, et al. Pandaset: Advanced sensor suite dataset for autonomous driving. In *ITSC*, 2021.

[88] S. Tulsiani, R. Tucker, and N. Snavely. Layer-structured 3d scene inference via view synthesis. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 302–317, 2018.

[89] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[90] G. Parmar, R. Zhang, and J.-Y. Zhu. On aliased resizing and surprising subtleties in gan evaluation. In *Computer Vision and Pattern Recognition*, 2022.

[91] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. *NeurIPS*, 2017.

[92] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *ICCV*, pages 2961–2969, 2017.