LatentFusion: End-to-End Differentiable Reconstruction and Rendering for Unseen Object Pose Estimation

Keunhong Park[1,2]* Arsalan Mousavian[2] Yu Xiang[2] Dieter Fox[1,2]
[1]University of Washington [2]NVIDIA

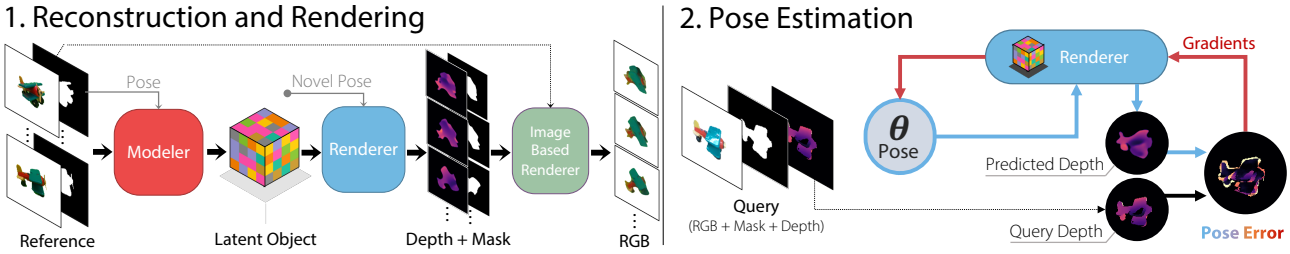

Figure 1: We present an end-to-end differentiable reconstruction and rendering pipeline. We use this pipeline to perform pose estimation on unseen objects using simple gradient updates in a render-and-compare fashion.

Abstract

Current 6D object pose estimation methods usually require a 3D model for each object. These methods also require additional training in order to incorporate new objects. As a result, they are difficult to scale to a large number of objects and cannot be directly applied to unseen objects.

We propose a novel framework for 6D pose estimation of unseen *objects. We present a network that reconstructs a latent 3D representation of an object using a small number of reference views* at inference time. *Our network is able to render the latent 3D representation from arbitrary views. Using this neural renderer, we directly optimize for pose given an input image. By training our network with a large number of 3D shapes for reconstruction and rendering, our network generalizes well to unseen objects. We present a new dataset for unseen object pose estimation–MOPED. We evaluate the performance of our method for unseen object pose estimation on MOPED as well as the ModelNet and LINEMOD datasets. Our method performs competitively to supervised methods that are trained on those objects. Code and data will be available at https://keunhong.com/publications/latentfusion/.*

1. Introduction

The *pose* of an object defines where it is in space and how it is oriented. An object pose is typically defined by a 3D orientation (rotation) and translation comprising six degrees of freedom (6D). Knowing the pose of an object is crucial for any application that involves interacting with real world objects. For example, in order for a robot to manipulate objects it must be able to reason about the pose of the object. In augmented reality, 6D pose estimation enables virtual interaction and re-rendering of real world objects.

In order to estimate the 6D pose of objects, current state-of-the-art methods [49, 8, 45] require a 3D model for each object. Methods based on renderings [42] usually need high quality 3D models typically obtained using 3D scanning devices. Although modern 3D reconstruction and scanning techniques such as [26] can generate 3D models of objects, they typically require significant effort. It is easy to see how building a 3D model for every object is an infeasible task.

Furthermore, existing pose estimation methods require extensive training under different lighting conditions and occlusions. For methods that train a single network for multiple objects [49], the pose estimation accuracy drops significantly with the increase in the number of objects. This is due to large variation of object appearances depending on the pose. To remedy this mode of degradation, some approaches train a separate network for each object [42, 41, 8]. This approach is not scalable to a large number of objects. Regardless of using a single or multiple networks, all model-based methods require extensive training for unseen test objects that are not in the training set.

In this paper, we investigate the problem of constructing a 3D object representations for 6D object pose estimation without 3D models and without extra training for unseen objects during test time. The core of our method is a novel neural network that takes a set of reference RGB images of a target object with known poses, and internally builds a 3D

*Work done while author was an intern at NVIDIA.

representation of the object. Using the 3D representation, the network is able to render arbitrary views of the object. To estimate object pose, the network compares the input image with its rendered images in a gradient descent fashion to search for the best pose where the rendered image matches the input image. Applying the network to an unseen object only requires collecting views with registered poses using traditional techniques [26] and feeding a small subset of those views with the associated poses to the network, instead of training for the new object which takes time and computational resources.

Our network design is inspired by space carving [18]. We build a 3D voxel representation of an object by computing 2D latent features and projecting them to a canonical 3D voxel space using a *de*projection unit inspired by [27]. This operation can be interpreted as space carving in latent space. Rendering a novel view is conducted by rotating the latent voxel representation to the new view and projecting it into the 2D image space. Using the projected latent features, a decoder generates a new view image by first predicting the depth map of the object at the query view and then assigning color for each pixel by combining corresponding pixel values at different reference views.

To reconstruct and render unseen objects, we train the network on the ShapeNet dataset [4] randomly textured with images from the MS-COCO dataset [21] under random lighting conditions. Our experiments show that the network generalizes to novel object categories and instances. For pose estimation, we assume that the object of interest is segmented with a generic object instance segmentation method such as [50]. The pose of the object is estimated by finding a 6D pose that minimizes the difference between a predicted rendering and the input image. Since our network is a differentiable renderer, we optimize by directly computing the gradients of the loss with respect to the object pose. Fig. 1 illustrates our reconstruction and pose estimation pipeline.

Some key benefits of our method are:

1. *Ease of Capture* – we perform pose estimation given just a few reference images rather than 3D scans;

2. *Robustness to Appearance* – we create a latent representation from images rather than relying on a 3D model with baked appearance; and

3. *Practicality* – our zero-shot formulation requires only one neural network model for all objects and requires no training for novel objects.

In addition, we introduce the Model-free Object Pose Estimation Dataset (MOPED) for evaluating pose estimation in a zero-shot setting. Existing pose estimation benchmarks provide 3D models and rendered training images sequences, but typically do not provide casual real-world reference images. MOPED provides registered reference and test images for evaluating pose estimation in a zero-shot setting.

2. Related Work

Pose Estimation. Pose estimation methods fall into three major categories. The first category tackles the problem of pose estimation by designing network architectures that facilitate pose estimation [25, 17, 15]. The second category formulates the pose estimation by predicting a set of 2D image features, such as the projection of 3D box corners [42, 45, 14, 33] and direction of the center of the object [49], then recovering the pose of the object using the predictions. The third category estimates the pose of objects by aligning the rendering of the 3D model to the image. DeepIM [20] trains a neural network to align the 3D model of the object to the image. Another approach is to learn a model that can reconstruct the object with different poses [41, 8]. These methods then use the latent representation of the object to estimate the pose. A limitation of this line of work is that they need to train separate auto-encoders for each object category and there is a lack of knowledge transfer *between* object categories. In addition, these methods require high-fidelity textured 3D models for each object which are not trivial to build in practice since it involves specialized hardware [37]. Our method addresses these limitations: our method works with a set of reference views with registered poses instead of a 3D model. Without additional training, our system builds a latent representation from the reference views which can be rendered to color and depth for arbitrary viewpoints. Similar to [41, 8], we seek to find a pose that minimizes the difference in latent space between the query object and the test image. Differentiable mesh renderers have been explored for pose estimation [29, 5] but still require 3D models leaving the acquisition problem unsolved.

3D shape learning and novel view synthesis. Inferring shapes of objects at the category level has recently gained a lot of attention. Shape geometry has been represented as voxels [6], Signed Distance Functions (SDFs) [30], point clouds [51], and as implicit functions encoded by a neural network [39]. These methods are trained at the category level and can only represent different instances within the categories they were trained on. In addition, these models only capture the *shape* of the object and do not model the *appearance* of the object. To overcome this limitation, recent works [28, 27, 39] decode appearance from neural 3D latent representations that respect projective geometry, generalizing well to novel viewpoints. Novel views are generated by transforming the latent representation in 3D and projecting it to 2D. A decoder then generates a novel view from the projected features. Some methods find a nearest neighbor shape proxy and infer high quality appearances but cannot handle novel categories [46, 32]. Differentiable

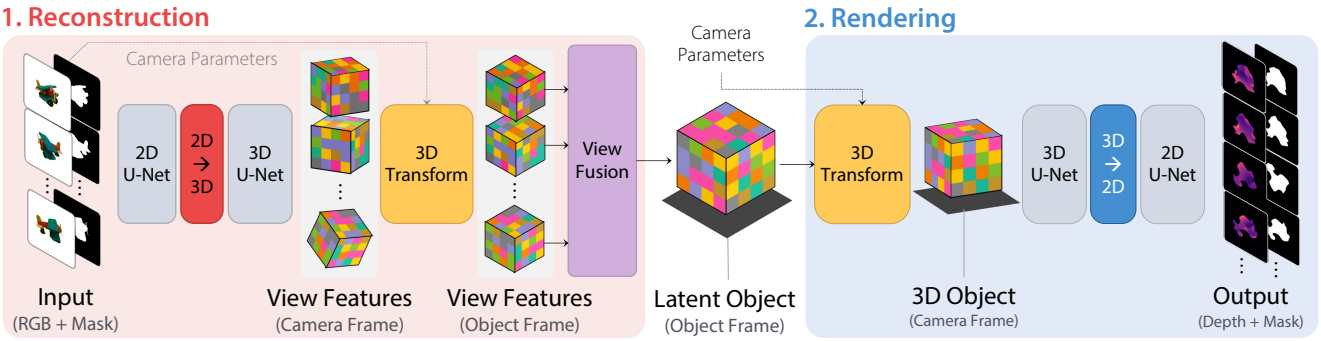

Figure 2: A high level overview of our architecture. 1) Our modeling network takes an image and mask and predicts a feature volume for each input view. The predicted feature volumes are then fused into a single canonical *latent object* by the fusion module. 2) Given the latent object, our rendering network produces a depth map and a mask for any camera pose.

rendering [19, 27, 22] systems seek to implement the rendering process (rasterization and shading) in a differentiable manner so that gradients can be propagated to and from neural networks. Such methods can be used to directly optimize parameters such as pose or appearance. Current differentiable rendering methods are limited by the difficulty of implemented complex appearance models and require a 3D mesh. We seek to combine the best of these methods by creating a differentiable rendering pipeline that does not require a 3D mesh by instead building voxelized latent representations from a small number of reference images.

Multi-View Reconstruction. Our method takes inspiration from multi-view reconstruction methods. It is most similar to space carving [18] and can be seen as a latent-space extension of it. Dense fusion methods such as [26, 47] generate dense point clouds of the objects from RGB-D sequences. Recent works [44, 43] have explored ways to learn object representations from unaligned views. These methods recover coarse geometry and pose given an image, but require large amounts of training data for a single object category. Our method builds on both approaches: we train a network to reconstruct an object; however, instead of training per-object or per-category, we provide multiple reference images at inference time to create a 3D latent representation which can be rendered from novel viewpoints.

3. Overview

We present an end-to-end system for novel view reconstruction and pose estimation. We present our system in two parts. Sec. 4 describes our reconstruction pipeline which takes a small collection of reference images as input and produces a flexible representation which can be rendered from novel viewpoints. We leverage multi-view consistency to construct a latent representation and do not rely on category specific shape priors. This key architecture decision enables generalization beyond the distribution of training objects. We show that our reconstruction pipeline can accurately reconstruct unseen object categories from real images. In Sec. 5, we formulate the 6D pose estimation problem using our neural renderer. Since our rendering process is fully differentiable, we directly optimize for the camera parameters without the need for additional training or codebook generation for new objects.

Camera Model. Throughout this paper we use a perspective pinhole camera model with an intrinsic matrix

$$\boldsymbol{K} = \begin{pmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{1}$$

and a homogeneous extrinsic matrix $\boldsymbol{E} = [\boldsymbol{R}|\boldsymbol{t}]$, where f_u and f_v are the focal lengths, u_0 and v_0 are the coordinates of the camera principal point, and $\boldsymbol{R}$ and $\boldsymbol{t}$ are rotation and translation of the camera, respectively. We also define a *viewport* cropping parameter $\boldsymbol{c} = (u_-, v_-, u_+, v_+)$ which represents a bounding box around the object in pixel coordinates. For brevity, we refer to the collection of these camera parameters as $\boldsymbol{\theta} = \{\boldsymbol{R}, \boldsymbol{t}, \boldsymbol{c}\}$.

4. Neural Reconstruction and Rendering

Given a set of N reference images with associated object poses and object segmentation masks, we seek to construct a representation of the object which can be rendered with arbitrary camera parameters. Building on the success of recent methods [28, 39], we represent the object as a latent 3D voxel grid. This representation can be directly manipulated using standard 3D transformations–naturally accommodating our requirement of novel view rendering. The overview of our method is shown in Fig. 2. There are two main components in our reconstruction pipeline: 1) *Modeling* the object by predicting per-view feature volumes and fusing them into a single canonical latent representation; 2) *Rendering* the latent representation to depth and color images.

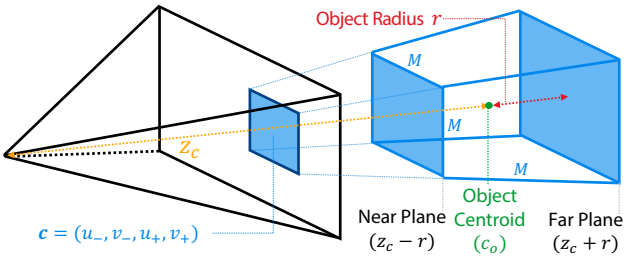

Figure 3: The $M \times M \times M$ per-view feature volumes computed in the modeling network corresponds a depth bounded camera frustum. The blue box on the image plane is determined by the camera crop parameter $\boldsymbol{c} = (u_-, v_-, u_+, v_+)$ and together with the depth determines the bounds of the frustum.

4.1. Modeling

Our modeling step is inspired by space carving [18] in that our network takes observations from multiple views and leverages multi-view consistency to build a canonical representation. However, instead of using photometric consistency, we use latent features to represent each view which allows our network to learn features useful for this task.

Per-View Features. We begin by generating a feature volume for each input view $\boldsymbol{\mathcal{I}}_i \in \{\boldsymbol{\mathcal{I}}_1, \ldots, \boldsymbol{\mathcal{I}}_N\}$. Each feature volume corresponds to the camera frustum of the input camera, bounded by the viewport parameter $\boldsymbol{c} = (u_-, v_-, u_+, v_+)$ and depth-wise by $z \in [z_c - r, z_c + r]$ where z_c is the distance to the object center and r is the radius of the object. Fig. 3 illustrates the generation of the per-view features. Similar to [38], we use U-Nets [34] for their property of preserving spatial structure. We first compute 2D features $g_{\text{pix}}(\boldsymbol{x}_i) \in \mathbb{R}^{C \times H \times W}$ by passing the input $\boldsymbol{x}_i$ (an RGB image $\boldsymbol{\mathcal{I}}_i$, a binary mask $\boldsymbol{\mathcal{M}}_i$, and optionally depth $\boldsymbol{\mathcal{D}}_i$) through a 2D U-Net. The deprojection unit ($p_\uparrow$) then lifts the 2D image features in $\mathbb{R}^{C \times H \times W}$ to 3D volumetric features in $\mathbb{R}^{(C/D) \times D \times H \times W}$ by factoring the 2D channel dimension into the 3D channel dimension $C' = C/D$ and depth dimension D. This deprojection operation is the exact opposite of the projection unit presented in [27]. The lifted features are then passed through a 3D U-Net g_{cam} to produce the volumetric features for the camera: $\boldsymbol{\Phi_i} = g_{\text{cam}} \circ p_\uparrow \circ g_{\text{pix}}(\boldsymbol{x}_i) \in \mathbb{R}^{C' \times M \times M \times M}$.

Camera to Object Coordinates. Each voxel in our feature volume represents a point in 3D space. Following recent works [27, 28, 38], we transform our feature volumes directly using rigid transformations. Consider a continuous function $\phi(\boldsymbol{x}) \in \mathbb{R}^{C'}$ defining our camera-space latent representation, where $\boldsymbol{x} \in \mathbb{R}^3$ is a point in camera coordinates. The feature volume $\boldsymbol{\Phi}$ is a discrete sample of this function. This representation in object space is given by $\psi(\boldsymbol{x}') = \phi(\boldsymbol{W}^{-1}\boldsymbol{x}')$ where $\boldsymbol{x}'$ is a point in object coordinates and $\boldsymbol{W} = [\boldsymbol{R}|\boldsymbol{t}]$ is an object-to-camera extrinsic matrix. We compute the object-space volume $\hat{\boldsymbol{\Psi}}$ by sampling $\phi(\boldsymbol{W}^{-1}\boldsymbol{x}'_{ijk})$ for each object-space voxel coordinate $\boldsymbol{x}'_{ijk}$. In practice, this is done by trilinear sampling the voxel grid and edge-padding values that fall outside. Given this transformation operation $T_{\text{c}\to\text{o}}$, the object-space feature volume is given by $\hat{\boldsymbol{\Psi}}_{\boldsymbol{i}} = T_{\text{c}\to\text{o}}(\boldsymbol{\Phi_i})$.

View Fusion. We now have a collection of feature volumes $\hat{\boldsymbol{\Psi}}_{\boldsymbol{i}} \in \{\hat{\boldsymbol{\Psi}}_{\boldsymbol{i}}, \ldots, \hat{\boldsymbol{\Psi}}_{\boldsymbol{N}}\}$, each associated with an input view. Our fusion module f fuses all views into a single canonical feature volume: $\boldsymbol{\Psi} = f(\hat{\boldsymbol{\Psi}}_{\boldsymbol{1}}, \ldots, \hat{\boldsymbol{\Psi}}_{\boldsymbol{N}})$.

Simple channel-wise average pooling yields good results but we found that sequentially integrating each volume using a Recurrent Neural Network (RNN) similarly to [38] slightly improved reconstruction accuracy (see Sec. 6.5). Using a recurrent unit allows the network to keep and ignore features from views in contrast to average pooling. This facilitates comparisons between different views allowing the network to perform operations similar to the photometric consistency criterion used in space carving [18]. We use a Convolutional Gated Recurrent Unit (ConvGRU) [1] so that the network can leverage spatial information.

4.2. Rendering

Our rendering module takes the fused object volume $\boldsymbol{\Psi}$ and renders it given arbitrary camera parameters $\boldsymbol{\theta}$. Ideally, the rendering module would directly regress a color image. However, it is challenging to preserve high frequency details through a neural network. U-Nets [34] introduce skip connections between equivalent-scale layers allowing high frequency spatial structure to propagate to the end of the network, but it is unclear how to add skip connections in the presence of 3D transformations. Existing works such as [38, 23] train a single network for each scene allowing the decoder to memorize high frequency information while the latent representation encodes state information. Trying to predict color without skip connections results in blurry outputs. We side-step this difficulty by first rendering depth and then using an image-based rendering approach to produce a color image.

Decoding Depth. Depth is a 3D representation, making it easier for the network to exploit the geometric structure we provide. In addition, depth tends to be locally smoother compared to color allowing more information to be compactly represented in a single voxel.

Our rendering network is a simple inversion of the reconstruction network and bears many similarities to RenderNet [27]. First, we pass the canonical object-space volume $\boldsymbol{\Psi}$ through a small 3D U-Net (h_{obj}) before transforming it to camera coordinates using the method described in Sec. 4.1. We perform the transformation with an object-to-camera extrinsic matrix $\boldsymbol{E}$ instead of the inverse $\boldsymbol{E}^{-1}$. A second

3D U-Net (h_{cam}) then decodes the resulting volume to produce a feature volume: $\boldsymbol{\Psi'} = h_{\text{cam}} \circ T_{\text{o}\to\text{c}} \circ h_{\text{obj}}(\boldsymbol{\Psi})$ which is then flattened to a 2D feature grid $\boldsymbol{\Phi'} = p_{\downarrow}(\boldsymbol{\Psi'})$ using the projection unit ($p_{\downarrow}$) from [27] by first collapsing the depth dimension into the channel dimension and applying a 1x1 convolution. The resulting features are decoded by a 2D U-Net (h_{pix}) with two output branches for depth (h_{depth}) and for a segmentation mask (h_{mask}). The outputs of the rendering network are given by $\boldsymbol{y}_{\text{depth}}(\boldsymbol{\Phi'}) = h_{\text{depth}} \circ h_{\text{pix}}(\boldsymbol{\Phi'})$ and $\boldsymbol{y}_{\text{mask}}(\boldsymbol{\Phi'}) = h_{\text{mask}} \circ h_{\text{pix}}(\boldsymbol{\Phi'})$.

Image Based Rendering (IBR). We use image-based rendering [36] to leverage the reference images to predict output color. Given the camera intrinsics $\boldsymbol{K}$ and depth map for an output view, we can recover the 3D object-space position of each output pixel (u, v) as $\boldsymbol{X} = \boldsymbol{E}^{-1}\left(\frac{u-u_0}{f_u}z, \frac{v-v_0}{f_v}z, z, 1\right)^T$, which can be transformed to the input image frame as $\boldsymbol{x'_i} = \boldsymbol{K_i W_i X}$ for each input camera $\boldsymbol{\theta}_i = \{\boldsymbol{K_i}, \boldsymbol{W_i}\}$. The output pixel can then copy the color of the corresponding input pixel to produce a reprojected color image.

The resulting reprojected image will contain invalid pixels due occlusions. There are multiple strategies to weighting each pixel including 1) weighting by reprojected depth error, 2) weighting by similarity between input and query cameras, 3) using a neural network. The first choice suffers from artifacts in the presence of depth errors or thin surfaces. The second approach yields reasonable results but produces blurry images for intermediate views. We opt for the third option. Following deep blending [10], we train a network that predicts blend weights $\boldsymbol{\mathcal{W}}_i$ for each reprojected input $\boldsymbol{\mathcal{I}}'_i$: $\boldsymbol{\mathcal{I}}_o = \sum_i \boldsymbol{\mathcal{W}}_i \odot \boldsymbol{\mathcal{I}}'_i$, where $\odot$ is an element-wise product. The blend weights are predicted by a 2D U-Net. The inputs to this network are 1) the depth predicted by our reconstruction pipeline, 2) each reprojected input image $\boldsymbol{\mathcal{I}}'_i$, and 3) a view similarity score s based on the angle between the input and query poses.

4.3. Implementation Details

Training Data. We train our reconstruction network on shapes from ShapeNet [4] which contains around 51,300 shapes. We exclude large models for efficient data loading resulting in around 30,000 models. We generate UV maps using Blender's smart UV projection [3] to facilitate texturing. We normalize all models to unit diameter. When rendering, we sample a random image from MS-COCO [21] for each component of the model. We render with the Beckmann model [2] with randomized parameters and also render uniformly colored objects with a probability of 0.5.

Network Input. We generate our training data at a resolution of 640×480. However, the input to our network is a fixed size 128×128. To keep our inputs consistent and our network scale-invariant, we 'zoom' into the object such that all images appear to be from the same distance. This is done by computing a bounding box size $(w_b, h_b) = (\frac{d'w'}{f_u dw}, \frac{d'h'}{f_v dh})$ where (w, h) is the current image width and height, d is the distance to the centroid c_o (See Fig. 3), (w', h') is the desired output size, and d' is the desired 'zoom' distance and cropping around object centroid projected to image coordinates (c_u, c_v). This defines the viewport parameter $\boldsymbol{c} = (c_u - w_b/2, c_v - h_b/2, c_u + w_b/2, c_v + h_b/2)$. The cropped image is scaled to 128×128.

Training. In each iteration of training, we sample a 3D model and then sample 16 random reference poses and 16 random target poses. Each pose is sampled by uniformly sampling a unit quaternion and translation such that the object stays within frame. We train our network using the Adam optimizer [16] with a fixed learning rate of 0.001 for 1.5M iterations. Each batch consists of 20 objects with 16 input views and 16 target views. We use an L_1 reconstruction loss for depth and binary cross-entropy for the mask. We apply the losses to both the input and output views. We randomly orient our canonical coordinate frame in each iteration by uniformly sampling a random unit quaternion. This prevents our network from overfitting to the implementation of our latent voxel transformations. We also add motion blur, color jitter, and pixel noise to the color inputs and add noise to the input masks using the same procedure as [24].

5. Object Pose Estimation

Given an image $\boldsymbol{\mathcal{I}}$, and a depth map $\boldsymbol{\mathcal{D}}$, a pose estimation system provides a rotation $\boldsymbol{R}$ and a translation $\boldsymbol{t}$ which together define an object-to-camera coordinate transformation $\boldsymbol{E} = [\boldsymbol{R}|\boldsymbol{t}]$ referred to as the *object pose*. In this section, we describe how we use our reconstruction pipeline described in Sec. 4 to directly optimize for the pose. We first find a *coarse* pose using only forward inference and then *refine* it using gradient optimization.

Formulation. Pose is defined by a rotation $\boldsymbol{R}$ and a translation $\boldsymbol{t}$. Our formulation also includes the viewport parameter $\boldsymbol{c}$ defined in Sec. 3. Defining a viewport allows us to efficiently pass the input to the reconstruction network while also providing scale invariance. We encode the rotation as a quaternion $\boldsymbol{q}$ and translation as $\boldsymbol{t}$. We assume we are given an RGB image $\boldsymbol{\mathcal{I}}$, an object segmentation mask $\boldsymbol{\mathcal{M}}$, and depth $\boldsymbol{\mathcal{D}}$ comprising the input $\boldsymbol{x} = \{\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{M}}, \boldsymbol{\mathcal{D}}\}$.

5.1. Losses

In order to estimate pose, we must provide a criterion which quantifies the quality of the pose. We use four loss functions. One is a standard L_1 depth reconstruction loss $\mathcal{L}_{\text{depth}}(\boldsymbol{\mathcal{D}}^*, \boldsymbol{\mathcal{D}}) = \|\boldsymbol{\mathcal{D}}^* - \boldsymbol{\mathcal{D}}\|_1$ which disambiguates the object scale and measures how well the predicted depth $\boldsymbol{\mathcal{D}}$ matches the input depth $\boldsymbol{\mathcal{D}}^*$. We also use

a pixel-wise binary cross entropy loss $\mathcal{L}_{\text{mask}}$ on the predicted mask as well as intersection over union (IoU) loss $\mathcal{L}_{\text{iou}}(\boldsymbol{\mathcal{M}}^*, \boldsymbol{\mathcal{M}}) = \log U - \log I$ where U is the sum of the pixels in the union and I is the sum of the pixels in the intersection of the masks $\boldsymbol{\mathcal{M}}^*$ and $\boldsymbol{\mathcal{M}}$. Finally, we introduce a novel latent loss which leverages our reconstruction network F. Given the input $\boldsymbol{x} = \{\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{M}}, \boldsymbol{\mathcal{D}}\}$, a latent object $\boldsymbol{\Psi}$, and a pose $\boldsymbol{\theta}$, the latent loss is defined as $\mathcal{L}_{\text{latent}}(\boldsymbol{x}, \boldsymbol{\theta}; \boldsymbol{\Psi}) = \|H_{\boldsymbol{\theta}}(G_{\boldsymbol{\theta}}(\boldsymbol{x})) - H_{\boldsymbol{\theta}}(\boldsymbol{\Psi})\|_1$, where $H_{\boldsymbol{\theta}}$ is the rendering network up to the projection layer and $G_{\boldsymbol{\theta}}$ is the modeling network as described in Sec. 4. This loss differs from auto-encoder based approaches such as [41, 8] in that 1) our network is not trained on the object, and 2) the loss is computed directly given the image and camera pose. Our pose estimation problem is given by:

$$\underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, \mathcal{L}_{\text{depth}} + \lambda \mathcal{L}_{\text{latent}} + \gamma \mathcal{L}_{\text{mask}} + \eta \mathcal{L}_{\text{iou}}, \qquad (2)$$

where λ, γ, η are the weights of the lossees. The parameters of the losses are omitted for clarity.

Parameterization. We parameterize the rotation in the log quaternion form $\boldsymbol{\omega} = (0, \omega_1, \omega_2, \omega_3)$ which ensures that all updates to the parameters result in a valid unit quaternion:

$$\boldsymbol{q} = \exp(\boldsymbol{\omega}) = \begin{pmatrix} \cos\|\boldsymbol{\omega}\| \\ \frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|} \sin\|\boldsymbol{\omega}\| \end{pmatrix}. \qquad (3)$$

Coarse Initialization. Although we have a differentiable renderer, the space of poses is non-convex which can lead to bad local minima when using gradient based optimization. We therefore bootstrap the pose by computing a coarse estimate. This also has the benefit of speeding up inference since it only requires forward evaluation.

We begin by estimating the translation of the object $\boldsymbol{t} = (x, y, z)$ as the centroid of the bounding cube defined by the mask bounding box $\boldsymbol{c}$ and corresponding depth values. We initialize k poses using the estimated translation. To get good coverage of possible orientations, we evenly sample azimuth and elevation angles using a Fibonacci lattice [9] then uniformly sample a random yaw angle. We use the cross entropy method [7] to optimize the translation and log quaternion parameters, and use a Gaussian Mixture Model as the probability distribution.

Pose Optimization. Our entire pipeline is differentiable end-to-end. We can therefore optimize Eq. (2) using gradient optimization. Given a latent object $\boldsymbol{\Psi}$ and a coarse pose estimate $\boldsymbol{\theta}$, we compute the loss and propagate the gradients back to the camera pose. This step only requires the rendering network and does not use the modeling network. The image-based rendering network is also not used in this step. We jointly optimize the rotation $\boldsymbol{q}$, translation $\boldsymbol{t}$, and viewport $\boldsymbol{c}$ using Adam [16].

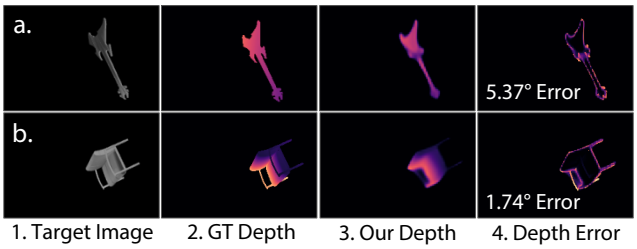

Figure 4: Two examples from the ModelNet experiment. (1) target image, (2) ground truth depth, (3) optimized predicted depth, and (4) L_1 error between the ground truth and our prediction. (a) illustrates how a pose with low depth error can still result in a relatively high angular error.

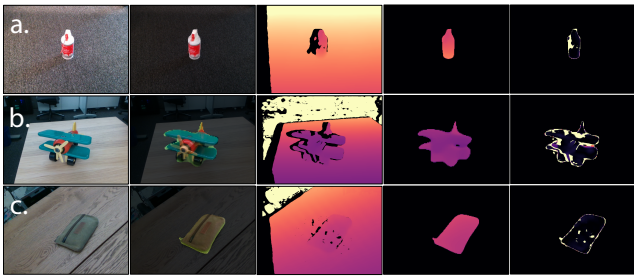

Figure 5: Qualitative results on the MOPED dataset.

6. Experiments

We evaluate our method on LINEMOD [11], ModelNet [48] and our new dataset MOPED. We aim to evaluate pose estimation accuracy on unseen objects.

6.1. Evaluation Metrics

We use four main evaluation metrics. 1) $(k^\circ, k\text{cm})$: a pose is considered correct if it is within k° and $k$cm of the ground truth target pose where the angular metric is the angle between the orientations. 2) ADD [12]: the average distance between points after being transformed by the ground truth and predicted poses. 3) ADD-S: A modification to ADD that computes the average distance to the closest point rather than the ground truth point to account for symmetric objects. 4) Proj.2D: the pixel distance between the projected points of the ground truth and predicted pose.

6.2. Experiments on the LINEMOD Dataset

We evaluate our method on the LINEMOD dataset. We compare our results with DeepIM [20] and pix2pose [31]. Both of these methods are trained on the LINEMOD dataset. DeepIM uses the 3D models in an online fashion to refine the pose. We *do not train* on the dataset. Instead, our network is given 16 reference views of each object during inference time. We use the provided segmentation masks during inference. We follow the train/test split of [13] and

Table 1: Evaluation on LINEMOD. We report the ADD recall metric [12], comparing against DeepIM and pix2pose. Symmetric objects are indicated by * and the pose is considered correct if it is flipped along the z axis.

Method	Input	ape	benchvice	camera	can	cat	driller	duck	eggbox*	glue*	holepunch	iron	lamp	phone	mean
Ours	RGB-D	**83.7**	91.3	74.3	83.4	**84.4**	90.6	71.3	39.3	94.7	53.7	70.0	87.8	**90.3**	78.0
DeepIM [20]	RGB	77.0	**97.5**	**93.5**	**96.5**	82.1	**95.0**	**77.7**	**97.1**	**99.4**	52.8	**98.3**	**97.5**	87.7	**88.6**
pix2pose [31]	RGB	58.1	91.0	60.9	84.4	65.0	76.3	43.8	96.8	79.4	**74.8**	83.4	82.0	45.0	72.4

sample our reference views from the training set. We follow the same evaluation methodology of [12], reporting the percentage of poses with ADD metric less than 10% of the object diameter. Table. 1 shows the results. Our experiments show that our method performs on-par with state of the art supervised methods despite having never seen the objects.

6.3. Experiments on the ModelNet Dataset

We conduct experiments on ModelNet [48] to evaluate the generalization of our method toward unseen object categories. To this end, we train our network on all the meshes in ShapeNetCore [4] excluding the categories we are going to evaluate on. We closely follow the evaluation protocol of [20] here. The model is evaluated on 7 unseen categories: *bathtub, bookshelf, guitar, range hood, sofa, wardrobe*, and *TV stand*. For each category, 50 pairs of initial and target object pose are sampled. We compared with [20] and [40] where all the methods initialized with the initial pose and evaluated on how successful they are on estimating the target pose. We report three metrics: $(5°, 5cm)$, *ADD* within 10% of the object diameter, and *Proj.2D* within 5 pixels.

Table 2 shows the quantitative results on the ModelNet dataset. On average, our method achieves state-of-the-art results on all the metrics thanks to our ability to perform continuous optimization on pose. However, for the $(5°, 5cm)$ metric, there are object categories that our method performs worse despite performing well on all other metrics. One reason is the image and spatial resolution. The input and output images to our network have resolution 128×128. The resolution of our voxel representation is $16 \times 16 \times 16$. The limited resolution can hinder the performance for small objects or objects that are distant from the camera. Small changes in the depth of each pixel may disproportionately affect the rotation of the object compared to our losses. Fig. 4 shows examples from the ModelNet experiment illustrating this limitation.

6.4. Experiments on the MOPED Dataset

We introduce the Model-free Object Pose Estimation Dataset (MOPED). MOPED consists of 11 objects, shown in Fig. 6. For each object, we take multiple RGB-D videos cover all views the object. We first use KinectFusion [26] to register frames from a single capture and then use a combination of manual annotation and automatic registration [52, 35, 53] to align separate captures. We generate object segmentation maps using [50]. For each object, we select reference frames with farthest point sampling to ensure good coverage of the object. For test sequences, we capture each object in 5 different environments. We sample every other frame for evaluation videos. This results in approximately 300 test images per object. We evaluate our method and baselines using three metrics for which we provide the Area Under Curve (AUC): 1) *ADD* with threshold between $0 - 10cm$, 2) *ADD-S* with threshold between

Table 2: ModelNet pose refinement experiments compared to DeepIM (DI) [20] and Multi-Path Learning (MP) [40].

	(5°, 5cm)			ADD (0.1d)			Proj2D (5px)		
	DI	MP	Ours	DI	MP	Ours	DI	MP	Ours
bathtub	71.6	**85.5**	85.0	88.6	91.5	**92.7**	73.4	80.6	**94.9**
bookshelf	39.2	**81.9**	80.2	76.4	85.1	**91.5**	51.3	76.3	**91.8**
guitar	50.4	69.2	**73.5**	69.6	80.5	**83.9**	77.1	80.1	**96.9**
range_hood	69.8	**91.0**	82.9	89.6	95.0	**97.9**	70.6	83.9	**91.7**
sofa	82.7	**91.3**	89.9	89.5	95.8	**99.7**	94.2	86.5	**97.6**
tv_stand	73.6	85.9	**88.6**	92.1	90.9	**97.4**	76.6	82.5	**96.0**
wardrobe	62.7	88.7	**91.7**	79.4	92.1	**97.0**	70.0	81.1	**94.2**
Mean	64.3	84.8	**85.5**	83.6	90.1	**94.3**	73.3	81.6	**94.7**

Table 3: AUC metrics on MOPED by reference view count.

# Views	1	2	4	8	16	32
ADD	15.91	25.00	40.38	55.35	**58.67**	55.81
ADD-S	63.14	75.91	85.62	87.72	87.45	**88.70**
Proj.2D	8.68	15.43	28.41	38.87	**43.35**	38.45

Table 4: AUC metrics for different view fusion strategies

	ADD	ADD-S	Proj.2D
Avg Pool	**56.78**	88.04	39.82
ConvGRU	56.36	**88.28**	**40.43**

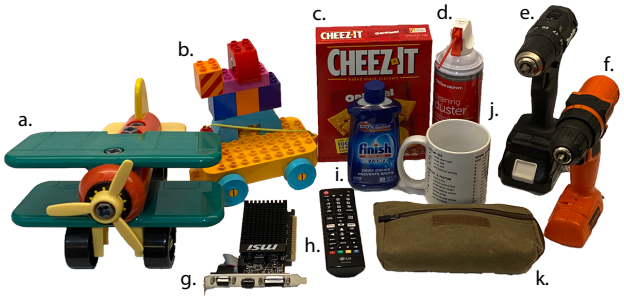

Figure 6: Objects in MOPED–a new dataset for model-free pose estimation. The objects shown are: (a) toy_plane, (b) duplo_dude, (c) cheezit, (d) duster, (e) black_drill, (f) orange_drill, (g) graphics_card, (h) remote, (i) rinse_aid, (j) vim_mug, and (k) pouch.

Table 5: Quantitative Results on MOPED Dataset. We report the Area Under Curve (AUC) for each metric.

	PoseRBPF [8]			Ours (LD)			Ours (D)			Ours (L)		
Pose Loss	-			$\mathcal{L}_{\text{latent}} + \mathcal{L}_{\text{depth}}$			$\mathcal{L}_{\text{depth}}$			$\mathcal{L}_{\text{latent}}$		
	ADD	**ADD-S**	**Proj.2D**	**ADD**	**ADD-S**	**Proj.2D**	**ADD**	**ADD-S**	**Proj.2D**	**ADD**	**ADD-S**	**Proj.2D**
black_drill	59.78	**82.94**	49.80	56.67	79.06	53.77	**62.15**	82.36	**59.36**	51.61	80.81	48.05
cheezit	57.78	82.45	48.47	**61.31**	**91.63**	**55.24**	44.56	90.24	35.10	23.98	88.20	15.92
duplo_dude	56.91	82.14	47.11	74.02	89.55	52.49	**76.81**	**90.50**	**59.83**	53.26	89.51	38.54
duster	**58.91**	82.78	**46.66**	49.13	91.56	19.33	51.13	**91.68**	24.78	39.05	81.57	20.82
graphics_card	59.13	83.20	49.85	**80.71**	**91.25**	**67.71**	79.33	90.90	60.35	60.11	87.91	41.92
orange_drill	**58.23**	82.68	**49.08**	51.84	70.95	46.12	55.52	73.68	45.46	44.20	68.39	41.68
pouch	57.74	82.16	49.01	**60.43**	**89.60**	**49.80**	58.51	89.15	44.40	22.03	82.94	20.19
remote	56.87	82.04	**48.06**	55.38	94.80	37.73	**63.18**	**94.96**	45.27	62.39	91.58	41.96
rinse_aid	57.74	82.53	**48.13**	65.63	92.58	28.61	**67.09**	**93.66**	27.62	57.54	87.44	19.00
toy_plane	**62.41**	85.10	49.81	60.18	**90.24**	**51.70**	56.80	88.54	40.16	34.29	87.22	35.07
vim_mug	**58.09**	**82.38**	**48.08**	30.11	80.76	14.38	49.89	77.79	32.85	27.49	78.59	10.51
mean	58.51	82.76	**48.55**	58.67	87.45	43.35	**60.45**	**87.59**	43.20	43.27	84.01	30.33

$0-10cm$, and 3) *Proj.2D* with threshold between $0-40px$. We compute all metrics for all sampled frames.

We compare our method with PoseRBPF [8], a state-of-the-art model-based pose estimation method. Since PoseRBPF requires textured 3D models, we reconstruct a mesh for each object by aggregating point clouds from reference captures and building a TSDF volume. The point clouds are integrated into the volume using KinectFusion [26]. The meshes have artifacts such as washed out high frequency details and shadow vertices due to slight misalignment (see supplementary materials). Table 5 shows quantitative comparisons on the MOPED dataset. Note that our method is not trained on the test objects while PoseRBPF has a separate encoder for each object. Our method achieves superior performance on both ADD and ADD-S. We evaluate different version of our method with different combinations of loss functions. Compared to our combined loss, optimizing only $\mathcal{L}_{\text{depth}}$ performs better for geometrically asymmetric objects but worse on textured objects such as the cheezit box. Optimizing both losses achieves better results on textured objects. Fig. 5 shows estimated poses for different test images. Please see supplementary materials for qualitative examples.

6.5. Ablation Studies

In this section, we analyze the effect of different design choices and how they affect the robustness of our method.

Number of reference views. We first evaluate the sensitivity of our method to the number of input reference views. Novel view synthesis is easier with more reference views because there is a higher chance that a query view will be close to a reference view. Table 3 shows that the accuracy increases with the number of reference views. In addition, having more than 8 reference views only yields marginal performance gains shows that our method does not require many views to achieve good pose estimation results.

View Fusion. We compare multiple strategies for aggregating the latent representations from each reference view. The naive way is to use a simple pooling function such as average/max pooling. Alternatively, we can integrate the volumes using an RNN such as a ConvGRU so that the network can reason across views. Table 4 shows the quantitative evaluation of these two variations. Although the average performance of the objects are very similar, the ConvGRU variation performs better than the average pooling variation. This indicates the importance of spatial relationship in the voxel representation for pose estimation.

7. Conclusion

We have presented a novel framework for building 3D object representations at inference time using a small number of reference images, as well as an accompanying neural renderer to render the 3D representation from arbitrary 6D viewpoints. Our networks are trained on thousands of shapes with random textures rendered under various lighting conditions allowing it to robustly generalize to unseen objects *without additional training*.

We leverage our reconstruction and rendering pipeline for zero-shot pose estimation. We perform pose estimation given just a small number of reference views and without needing to train any network. This greatly simplifies the process for performing pose estimation on novel objects as a detailed 3D model is not required. In addition, we have a single universal network which works for all objects including unseen ones. For future work, we plan to investigate unseen object pose estimation in cluttered scenes with occlusions. We also plan to speed up the pose estimation process by applying network optimization techniques.

Acknowledgements

We thank Xinke Deng for helpful discussions.

References

[1] Nicolas Ballas, Li Yao, Chris Pal, and Aaron Courville. Delving deeper into convolutional networks for learning video representations. *arXiv preprint arXiv:1511.06432*, 2015. 4

[2] Petr Beckmann and Andre Spizzichino. The scattering of electromagnetic waves from rough surfaces. *Norwood, MA, Artech House, Inc.*, 1987. 5

[3] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, 2019. 5

[4] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], 2015. 2, 5, 7

[5] Wenzheng Chen, Huan Ling, Jun Gao, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3D objects with an interpolation-based differentiable renderer. In *Advances in Neural Information Processing Systems (NuerIPS)*, pages 9605–9616, 2019. 2

[6] Christopher Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3D-R2N2: A unified approach for single and multi-view 3D object reconstruction. In *European Conference on Computer Vision (ECCV)*, pages 628–644, 2016. 2

[7] Pieter-Tjerk De Boer, Dirk Kroese, Shie Mannor, and Reuven Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134(1):19–67, 2005. 6

[8] Xinke Deng, Arsalan Mousavian, Yu Xiang, Fei Xia, Timothy Bretl, and Dieter Fox. PoseRBPF: A rao-blackwellized particle filter for 6D object pose tracking. In *Robotics: Science and Systems (RSS)*, 2019. 1, 2, 6, 8

[9] Álvaro González. Measurement of areas on a sphere using fibonacci and latitude–longitude lattices. *Mathematical Geosciences*, 42(1):49, 2010. 6

[10] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (TOG)*, 37(6):1–15, 2018. 5

[11] Stefan Hinterstoisser, Stefan Holzer, Cedric Cagniart, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *International Conference on Computer Vision (ICCV)*, pages 858–865, 2011. 6

[12] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes. In *Asian Conference on Computer Vision (ACCV)*, pages 548–562, 2012. 6, 7

[13] Tomas Hodan, Frank Michel, Eric Brachmann, Wadim Kehl, Anders GlentBuch, Dirk Kraft, Bertram Drost, Joel Vidal, Stephan Ihrke, Xenophon Zabulis, Caner Sahin, Federico Tombari Fabian Manhardt, Tae-Kyun Kim, Jiri Matas, and Carsten Rother. BOP: Benchmark for 6D object pose estimation. In *European Conference on Computer Vision (ECCV)*, pages 19–34, 2018. 6

[14] Yinlin Hu, Joachim Hugonot, Pascal Fua, and Mathieu Salzmann. Segmentation-driven 6D object pose estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3385–3394, 2019. 2

[15] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. SSD-6D: Making RGB-based 3D detection and 6D pose estimation great again. In *International Conference on Computer Vision (ICCV)*, pages 1521–1529, 2017. 2

[16] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5, 6

[17] Abhijit Kundu, Yin Li, and James M Rehg. 3D-RCNN: Instance-level 3D object reconstruction via render-and-compare. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3559–3568, 2018. 2

[18] Kiriakos Kutulakos and Steven Seitz. A theory of shape by space carving. *International Journal of Computer Vision (IJCV)*, 38(3):199–218, 2000. 2, 3, 4

[19] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018. 3

[20] Yi Li, Gu Wang, Xiangyang Ji, Yu Xiang, and Dieter Fox. DeepIM: Deep iterative matching for 6D pose estimation. In *European Conference on Computer Vision (ECCV)*, pages 683–698, 2018. 2, 6, 7

[21] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision (ECCV)*, pages 740–755, 2014. 2, 5

[22] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3D reasoning. *International Conference on Computer Vision (ICCV)*, 2019. 3

[23] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *ACM Transactions on Graphics (TOG)*, 38(4):65, 2019. 4

[24] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-Net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. In *Robotics: Science and Systems (RSS)*, 2017. 5

[25] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Kosecka. 3D bounding box estimation using deep learning and geometry. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[26] Richard Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, volume 11, pages 127–136, 2011. 1, 2, 3, 7, 8

[27] Thu Nguyen-Phuoc, Chuan Li, Stephen Balaban, and Yong-Liang Yang. RenderNet: A deep convolutional network for differentiable rendering from 3D shapes. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. 2, 3, 4, 5

[28] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. HoloGAN: Unsupervised learning of 3D representations from natural images. In *International Conference on Computer Vision (ICCV)*, 2019. 2, 3, 4

[29] Andrea Palazzi, Luca Bergamini, Simone Calderara, and Rita Cucchiara. End-to-end 6-DOF object pose estimation through differentiable rasterization. In *European Conference on Computer Vision (ECCV)*, pages 0–0, 2018. 2

[30] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 165–174, 2019. 2

[31] Kiru Park, Timothy Patten, and Markus Vincze. Pix2pose: Pixel-wise coordinate regression of objects for 6D pose estimation. In *International Conference on Computer Vision (ICCV)*, pages 7668–7677, 2019. 6, 7

[32] Keunhong Park, Konstantinos Rematas, Ali Farhadi, and Steven Seitz. Photoshape: photorealistic materials for large-scale shape collections. *ACM Transactions on Graphics (TOG)*, 37(6):192, 2019. 2

[33] Sida Peng, Yuan Liu, Qixing Huang, Xiaowei Zhou, and Hujun Bao. Pvnet: Pixel-wise voting network for 6dof pose estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4561–4570, 2019. 2

[34] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Conference on Medical Image Computing and Computer-assisted Intervention*, pages 234–241, 2015. 4

[35] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, 2001. 7

[36] Harry Shum and Sing Bing Kang. Review of image-based rendering techniques. In *Visual Communications and Image Processing*, volume 4067, pages 2–13, 2000. 5

[37] Arjun Singh, James Sha, Karthik S Narayan, Tudor Achim, and Pieter Abbeel. BigBIRD: A large-scale 3D database of object instances. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 509–516, 2014. 2

[38] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer. Deepvoxels: Learning persistent 3D feature embeddings. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 4

[39] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3D-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems (NuerIPS)*, 2019. 2, 3

[40] Martin Sundermeyer, Maximilian Durner, En Yen Puang, Zoltan-Csaba Marton, and Rudolph Triebel. Multi-path learning for object pose estimation across domains. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 7

[41] Martin Sundermeyer, Zoltan-Csaba Marton, Maximilian Durner, Manuel Brucker, and Rudolph Triebel. Implicit 3D orientation learning for 6D object detection from rgb images. In *European Conference on Computer Vision (ECCV)*, pages 699–715, 2018. 1, 2, 6

[42] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. In *Conference on Robot Learning (CoRL)*, 2018. 1, 2

[43] Shubham Tulsiani, Alexei Efros, and Jitendra Malik. Multi-view consistency as supervisory signal for learning shape and pose prediction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 3

[44] Shubham Tulsiani, Tinghui Zhou, Alexei Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 3

[45] Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martín-Martín, Cewu Lu, Li Fei-Fei, and Silvio Savarese. DenseFusion: 6D object pose estimation by iterative dense fusion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3343–3352, 2019. 1, 2

[46] Tuanfeng Wang, Hao Su, Qixing Huang, Jingwei Huang, Leonidas Guibas, and Niloy Mitra. Unsupervised texture transfer from images to model collections. *ACM Transactions on Graphics (TOG)*, 35(6):1–13, 2016. 2

[47] Thomas Whelan, Stefan Leutenegger, R Salas-Moreno, Ben Glocker, and Andrew Davison. ElasticFusion: Dense SLAM without a pose graph. In *Robotics: Science and Systems (RSS)*, 2015. 3

[48] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015. 6, 7

[49] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes. In *Robotics: Science and Systems (RSS)*, 2018. 1, 2

[50] Christopher Xie, Yu Xiang, Arsalan Mousavian, and Dieter Fox. The best of both modes: Separately leveraging RGB and depth for unseen object instance segmentation. In *Conference on Robot Learning (CoRL)*, 2019. 2, 7

[51] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. PointFlow: 3D point cloud generation with continuous normalizing flows. In *International Conference on Computer Vision (ICCV)*, 2019. 2

[52] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Fast global registration. In *European Conference on Computer Vision (ECCV)*, pages 766–782, 2016. 7

[53] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018. 7