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

Keunhong Park\textsuperscript{1,2}\textsuperscript{*} Arsalan Mousavian\textsuperscript{2} Yu Xiang\textsuperscript{2} Dieter Fox\textsuperscript{1,2}
\textsuperscript{1}University of Washington \textsuperscript{2}NVIDIA

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.

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 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 deprojection 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
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

\[
K = \begin{pmatrix}
    f_u & 0 & u_0 \\
    0 & f_v & v_0 \\
    0 & 0 & 1
\end{pmatrix},
\]

(1)

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

**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 to a depth bounded camera frustum. The blue box on the image plane is determined by the camera crop parameter $c = (u_-, v_-, u_+, v_+)$ 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 $I_i \in \langle I_1, \ldots, I_N \rangle$. Each feature volume corresponds to the camera frustum of the input camera, bounded by the viewport parameter $c = (u_-, v_-, u_+, v_+)$ 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 $g_{\text{pix}}(x_i)$ by passing the input $x_i$ (an RGB image $I_i$, a binary mask $M_i$, and optionally depth $D_i$) through a 2D U-Net. The deprojection unit $(p_t)$ 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 $g_{\text{cam}}$ to produce the volumetric features for the camera: 

$$\Phi_i = g_{\text{cam}} \circ p_t \circ g_{\text{pix}}(x_i) \in \mathbb{R}^{C' \times M \times M \times M}.$$

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 $\phi(x) \in \mathbb{R}^{C'}$ defining our camera-space latent representation, where $x \in \mathbb{R}^3$ is a point in camera coordinates. The feature volume $\Phi$ is a discrete sample of this function. This representation in object space is given by 

$$\psi(x') = \phi(W^{-1}x')$$

where $x'$ is a point in object coordinates and $W = [R|t]$ is an object-to-camera extrinsic matrix. We compute the object-space volume $\hat{\Psi}$ by sampling $\phi(W^{-1}x'_{ijk})$ for each object-space voxel coordinate $x'_{ijk}$. In practice, this is done by trilinear sampling the voxel grid and edge-padding values that fall outside. Given this transformation operation $T_{c \rightarrow o}$, the object-space feature volume is given by $\hat{\Psi}_i = T_{c \rightarrow o}(\Phi_i)$.

View Fusion. We now have a collection of feature volumes $\hat{\Psi}_i \in \{\hat{\Psi}_i, \ldots, \hat{\Psi}_N\}$, each associated with an input view. Our fusion module $f$ fuses all views into a single canonical feature volume: $\hat{\Psi} = f(\hat{\Psi}_1, \ldots, \hat{\Psi}_N)$.

Simple channel-wise average pooling yields good results but we found that sequentially integrating each view 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 $\hat{\Psi}$ and renders it given arbitrary camera parameters $\theta$. 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 $\hat{\Psi}$ through a small 3D U-Net ($h_{\text{obj}}$) 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 $E$ instead of the inverse $E^{-1}$. A second
3D U-Net \((h_{\text{cam}})\) then decodes the resulting volume to produce a feature volume: \(\Psi' = h_{\text{cam}} \circ T_{\text{w-cam}} \circ h_{\text{obj}}(\Psi)\) which is then flattened to a 2D feature grid \(\Phi' = p_l(\Psi')\) using the projection unit \((p_l)\) 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 \((h_{\text{pix}})\) with two output branches for depth \((h_{\text{depth}})\) and for a segmentation mask \((h_{\text{mask}})\). The outputs of the rendering network are given by \(y_{\text{depth}}(\Phi') = h_{\text{depth}} \circ h_{\text{pix}}(\Phi')\) and \(y_{\text{mask}}(\Phi') = h_{\text{mask}} \circ h_{\text{pix}}(\Phi')\).

**Image Based Rendering (IBR).** We use image-based rendering [36] to leverage the reference images to predict output color. Given the camera intrinsics \(K\) and depth map for an output view, we can recover the 3D object-space position of each output pixel \((u, v)\) as \(X = E^{-1} \begin{bmatrix} u - u_0 \\ v - v_0 \\ w \end{bmatrix}^T\), which can be transformed to the input image frame as \(x_i' = K_i W_i X\) for each input camera \(\theta_i = \{K_i, W_i\}\). 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 to 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 \(W_i\) for each reprojected input \(I_i'\): \(I_o = \sum_i W_i \circ I_i',\) where \(\circ\) 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 \(I_i'\), 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 Beemann 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 \((w_b, h_b) = (d_w h_w, d_h h_h)\) 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 \(c = (c_u - w_b/2, c_v - h_b/2, c_u + w_b/2, c_v + h_b/2)\). 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 \(I\), and a depth map \(D\), a pose estimation system provides a rotation \(R\) and a translation \(t\) which together define an object-to-camera coordinate transformation \(E = [R|t]\) 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 \(R\) and a translation \(t\). Our formulation also includes the viewport parameter \(c\) 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 \(q\) and translation as \(t\). We assume we are given an RGB image \(I\), an object segmentation mask \(M\), and depth \(D\) comprising the input \(x = \{I, M, D\}\).

### 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 \(L_{\text{depth}}(D^*, D) = \|D^* - D\|_1\) which disambiguates the object scale and measures how well the predicted depth \(D\) matches the input depth \(D^*\). We also use...
a pixel-wise binary cross entropy loss $L_{\text{mask}}$ on the predicted mask as well as intersection over union (IoU) loss $L_{\text{iou}}(\mathcal{M}^*, \mathcal{M}) = \log U - \log I$ 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 $\mathcal{M}^*$ and $\mathcal{M}$. Finally, we introduce a novel latent loss which leverages our reconstruction network $F$. Given the input $x = (\mathcal{I}, \mathcal{M}, \mathcal{D})$, a latent object $\Psi$, and a pose $\theta$, the latent loss is defined as $L_{\text{latent}}(x, \theta; \Psi) = \|H_{\theta}(G_{\theta}(x)) - H_{\theta}(\Psi)\|_1$, where $H_{\theta}$ is the rendering network up to the projection layer and $G_{\theta}$ 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:

$$\arg\min_\theta L_{\text{depth}} + \lambda L_{\text{latent}} + \gamma L_{\text{mask}} + \eta L_{\text{iou}},$$  \hspace{1cm} (2)

where $\lambda, \gamma, \eta$ are the weights of the losses. The parameters of the losses are omitted for clarity.

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

$$q = \exp(\omega) = \left(\frac{\cos \|\omega\|}{\|\omega\|}, \frac{\omega}{\|\omega\| \sin \|\omega\|}\right).$$  \hspace{1cm} (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 $t = (x, y, z)$ as the centroid of the bounding cube defined by the mask bounding box $c$ 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 $\Psi$ and a coarse pose estimate $\theta$, 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 $q$, translation $t$, and viewport $c$ 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.](image)

![Figure 5: Qualitative results on the MOPED dataset.](image)

### 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) $(k^o, kcm)$: a pose is considered correct if it is within $k^o$ and $kcm$ 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.

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

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

<table>
<thead>
<tr>
<th># Views</th>
<th>ADD (0.1d)</th>
<th>Proj.2D (5px)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DI</td>
<td>MP</td>
<td>Ours</td>
</tr>
<tr>
<td>bathtub</td>
<td>85.5</td>
<td>80.0</td>
</tr>
<tr>
<td>bookshelf</td>
<td>82.9</td>
<td>80.2</td>
</tr>
<tr>
<td>guitar</td>
<td>87.3</td>
<td>80.0</td>
</tr>
<tr>
<td>range_hood</td>
<td>89.6</td>
<td>85.0</td>
</tr>
<tr>
<td>sofa</td>
<td>89.3</td>
<td>89.5</td>
</tr>
<tr>
<td>tv_stand</td>
<td>86.6</td>
<td>90.9</td>
</tr>
<tr>
<td>wardrobe</td>
<td>87.7</td>
<td>91.7</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th># Views</th>
<th>ADD</th>
<th>ADD-S</th>
<th>Proj.2D</th>
</tr>
</thead>
<tbody>
<tr>
<td>DI</td>
<td>56.78</td>
<td>58.80</td>
<td>58.80</td>
</tr>
<tr>
<td>MP</td>
<td>57.36</td>
<td>59.84</td>
<td>59.84</td>
</tr>
<tr>
<td>Ours</td>
<td>58.67</td>
<td>60.81</td>
<td>60.81</td>
</tr>
</tbody>
</table>

Table 4: AUC metrics for different view fusion strategies.

<table>
<thead>
<tr>
<th>Method</th>
<th>ADD</th>
<th>ADD-S</th>
<th>Proj.2D</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConvGRU</td>
<td>56.78</td>
<td>58.80</td>
<td>58.80</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Pose Loss</th>
<th>PoseRBPF [8]</th>
<th>Ours (LD)</th>
<th>Ours (D)</th>
<th>Ours (L)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ADD</td>
<td>ADD-S</td>
<td>Proj.2D</td>
<td>ADD</td>
</tr>
<tr>
<td>black_drill</td>
<td>59.78</td>
<td>82.94</td>
<td>49.80</td>
<td>56.67</td>
</tr>
<tr>
<td>cheezit</td>
<td>57.78</td>
<td>82.45</td>
<td>48.47</td>
<td>61.31</td>
</tr>
<tr>
<td>duplo_dude</td>
<td>56.91</td>
<td>82.14</td>
<td>47.11</td>
<td>74.02</td>
</tr>
<tr>
<td>duster</td>
<td>58.91</td>
<td>82.78</td>
<td>46.66</td>
<td>49.13</td>
</tr>
<tr>
<td>graphics_card</td>
<td>59.13</td>
<td>83.20</td>
<td>49.85</td>
<td>80.71</td>
</tr>
<tr>
<td>orange_drill</td>
<td>58.23</td>
<td>82.68</td>
<td>49.08</td>
<td>51.84</td>
</tr>
<tr>
<td>pouch</td>
<td>57.74</td>
<td>82.16</td>
<td>49.01</td>
<td>60.43</td>
</tr>
<tr>
<td>remote</td>
<td>56.87</td>
<td>82.04</td>
<td>48.06</td>
<td>55.38</td>
</tr>
<tr>
<td>rinse Aid</td>
<td>57.74</td>
<td>82.53</td>
<td>48.13</td>
<td>65.63</td>
</tr>
<tr>
<td>toy plane</td>
<td>62.41</td>
<td>85.10</td>
<td>49.81</td>
<td>60.18</td>
</tr>
<tr>
<td>vim mug</td>
<td>58.09</td>
<td>82.38</td>
<td>48.08</td>
<td>30.11</td>
</tr>
<tr>
<td>mean</td>
<td>58.51</td>
<td>82.76</td>
<td>48.55</td>
<td>58.67</td>
</tr>
</tbody>
</table>

0 − 10 cm, and 3) Proj.2D with threshold between 0 − 40 px. 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 KineticFusion [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 \( L_{\text{depth}} \) 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.


[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


[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
