Supplementary Material for: CoCoNets: Continuous Contrastive 3D Scene Representations

Shamit Lal*, Mihir Prabhudesai*, Ishita Mediratta, Adam W. Harley, Katerina Fragkiadaki
Carnegie Mellon University
{shamitl, mprabhud, imedirat, aharley, katef} @cs.cmu.edu

1. Overview

The structure of this supplementary file is as follows: Section 2 covers the details on dataset preparation. Section 3 elucidates the implementation details, including inputs to our model, architecture details, and hyperparameter values used. Finally, Section 4 provides a more detailed analysis of the results.

2. Dataset Preparation

CARLA dataset. We use the CARLA simulator [2] to generate multi-view data for training, and tracking data for testing. CARLA is an open-source simulator for urban driving scenes, which allows multiple cameras to be placed at arbitrary positions. We generate episodes 300 frames long, with a framerate of 10 FPS. Each episode is captured from 6 random camera locations, sampled from a set of 18 cameras that lie on a hemisphere. We treat Towns 0-3 as the training set, and Towns 4-5 as the test set. In total, we use 388 scenes for self-supervised multi-view training and 188 scenes for tracking evaluation. The resolution of our RGB and depth images is 128x384.

KITTI dataset. We use the KITTI benchmark [3] to evaluate our model on the tracking task. We use the "tracking" subset of KITTI to test our model. This subset has 8008 frames across twenty labelled sequences. We create 8-frame sub-sequences of this data. We make sure that all the eight frames have a valid label for the target object. The egomotion information in the "tracking" data is only approximate.

Shapenet dataset. We use the meshes from ShapeNetCore [1]. We first normalize the ShapeNet meshes. We then render RGB-D data for them using Blender, where we place 24 cameras around the object. The cameras are placed at a distance of 2m, with azimuth ranging from 0° - 360° at intervals of 45°, and elevation ranging from 0° - 80° at intervals of 20°.

*Equal contribution

CLEVR dataset. We build upon the CLEVR Blender simulator [6]. Our scenes consist of cubes, spheres, and cylinders. We create scenes as follows: each object model is randomly rotated (0° to 360° along vertical axis), translated (randomly within a sphere of radius 10.5 units) and scaled (0.25 to 1.25 times the actual size). Each scene contains $2-10$ objects. We randomly vary the lighting of each scene. We render each scene by placing 28 RGB-D cameras at elevations ranging from 26° to 80° with 13° increments and azimuths ranging from 0° to 360° with 45° increments. Each camera is placed within a sphere of radius 1.5 metres from the center of the scene.

3. Implementation details

Code, training details and computation complexity. Our model is implemented in Python/Pytorch. We use a batch size of 2 and our learning rate is set at 10^{-4}. We use the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$. Our model takes 24 hours (approximately 100k iterations) of training for convergence on a single V100 GPU. As far as the running time of the proposed method for tasks in the experiments is concerned, extraction of point features from an RGB-D image takes 0.3 seconds for our model on a V100 GPU. A single iteration of Object tracking (Section 4.1) takes 5 seconds, Object Detection (Section 4.2) takes 0.4 seconds, Object Alignment (Section 4.3) takes 6 seconds and View Prediction using CoCoNets-*OccRGB* (Section 4.4) takes 0.6 seconds.

Inputs. We randomly select images from 2 views which includes the RGB-D and relative ground-truth egomotion (query view and target view) as input for training our model, while we use a single view for testing.

2.5D-to-3D lifting. Our 2.5D-to-3D unprojection module takes as input RGB-D images and converts it into a 4D tensor $\mathbf{U} \in \mathbb{R}^{w \times h \times d \times 4}$, where w, h, d is $64, 64, 64$. We use perspective (un)projection to fill the 3D grid with samples from 2D image. Specifically, using pinhole camera model

[4], we find the floating-point 2D pixel location that every cell in the 3D grid, indexed by the coordinate (i, j, k), projects onto from the current camera viewpoint. This is given by $[u, v]^T = \boldsymbol{K}\boldsymbol{S}[i, j, k]^T$, where $\boldsymbol{S}$, the similarity transform, converts memory coordinates to camera coordinates and $\boldsymbol{K}$, the camera intrinsics, convert camera coordinates to pixel coordinates. Bilinear interpolation is applied on pixel values to fill the grid cells. We obtain a binary occupancy grid $\mathbf{O} \in \mathbb{R}^{w \times h \times d \times 1}$ from the depth image $\mathbf{D}$ in a similar way. This occupancy is then concatenated with the unprojected RGB to get a tensor $[\mathbf{U}, \mathbf{O}] \in \mathbb{R}^{w \times h \times d \times 4}$. This tensor is then passed through a 3D encoder-decoder network, the architecture of which is as follows: 4-2-64, 4-2-128, 4-2-256, 4-0.5-128, 4-0.5-64, 1-1-F. Here, we use the notation k-s-c for kernel-stride-channels, and F is the feature dimension, which we set to $F = 32$. We concatenate the output of transposed convolutions in decoder with same resolution feature map output from the encoder. The concatenated tensor is then passed to the next layer in the decoder. We use leaky ReLU activation and batch normalization after every convolution layer, except for the last one in each network. We obtain our 3D feature map $\mathbf{M}$ as the output of this process. Even though we do not use multiple views as input in the paper, our system can handle that easily by orienting the feature map $\mathbf{M}_v$ obtained from each view v to a reference view to cancel the egomotion, and then fusing them to obtain the final feature map $\mathbf{M}$.

Implicit function parameterization. To predict the features, RGB, and occupancy values for different 3D physical points within a voxel grid, we use implicit functions parameterized by neural networks. As mentioned in the main paper, we denote the operation of obtaining point features, occupancies and colors by querying the feature map $\mathbf{M}$ at continuous locations (X, Y, Z) by $\mathcal{F}^f(\mathbf{M}, (X, Y, Z)), \mathcal{F}^o(\mathbf{M}, (X, Y, Z)), \mathcal{F}^c(\mathbf{M}, (X, Y, Z))$ respectively. We parameterize these three functions using neural networks with identical architectures, with the exception that the output of $\mathcal{F}^f$, $\mathcal{F}^o$, $\mathcal{F}^c$ is 32-D, 1-D and 3D, respectively. There is no weight sharing between these three neural networks. For brevity, we will use $\mathcal{F}$ when explaining properties common to all the three networks/functions.

The architecture of $\mathcal{F}$ is similar to that of [9]. Given the point (X, Y, Z) that we want to featurize, we first infer the trilinearly-interpolated feature vector at point (X, Y, Z) from the 8 neighbouring voxel features, which we denote as c. We encode the coordinate (X, Y, Z) into a 32-D feature vector using a linear layer. We denote this vector as p. The inputs c and p are then processed as follows:

$$out = RN_i(RN_{i-1}(\cdots RN_1(p + FC_1(c)) \cdots] + FC_{i-1}(c)) + FC_i(c)). \quad (1)$$

We set $i = 5$ for our usecase. FC_i is a linear layer which gives a 32-dimensional output. RN_i is a 2 layer ResNet block [5]. The architecture of ResNet block is as follows: ReLU, 32-32, ReLU, 32-32. Here, we use the notation $i_u - o_u$ to represent a linear layer, where i_u is the input dimension, and o_u is the output dimension of that layer. Each ResNet block finally generates the output by adding the input to the output of the above layers. out is then passed through a ReLU activation function followed by a linear layer, the output dimension of which depends on the property of the point we are trying to predict.

CoCoNets on RGB images during inference for 3D alignment. Our model can operate without depth maps at inference time. We achieve this by replacing our bottom-up 3D convolutional encoder-decoder architecture with a ResNet-18 [5] which directly operates on RGB image from the target view I_{target}. Our ResNet-18 encodes I_{target} to a 1D feature vector $\mathbf{x}_{target}$. During training, our top-down mapping of $(\mathrm{I}_{inp}, \boldsymbol{D}_{inp})$ to $\mathbf{M}^{\boldsymbol{V}}_{inp}$ and then to feature cloud $\{(X, Y, Z, \mathcal{F}(\mathbf{M}^{\boldsymbol{V}}_{inp}, (X, Y, Z)))\}$ remains the same, we instead replace the bottom-up feature point clouds obtained from $\mathbf{M}_{target}$ with feature point cloud $\{(X, Y, Z, \mathcal{F}(\mathbf{x}_{target}, (X, Y, Z)))\}$ obtained by interpolating the 1D feature vector $\mathbf{x}_{target}$. We then train these feature clouds using the same contrastive loss discussed in Section 3 of the main paper. Our bottom-up network gives us an option to operate without depth images during inference time. In Section 4.5, we show the comparison on 3D alignment of our model operating on RGB versus RGB-D images.

Volumetric rendering using CoCoNets on RGB images. We leverage our model's continuous representations on the task of rendering novel views on the ShapeNet dataset without depth as input and call it $CoCoNets^{NoDepthRGB}$. In this experiment, since we do not use depth information, we only pass the RGB image from the input view, I_{inp}, as an unprojected 3D tensor $U \in R^{w \times h \times d \times 3}$, to the top-down 3D-CNN encoder-decoder (Figure 1(a) in main file) to get M_{inp}. Then we take the camera pose change between the target (the view for which we want to render the image) and the input view, defined as V, and sample $w \times h$ number of rays, sampling 32 points on each ray, as per the method given in NeRF [8]. We use these sampled points, defined as D^V_{target}, to query M_{inp} using trilinear interpolation. Thereafter, we use the MLP architecture given in [8] and pass as input the positional encoding of the query point, its viewing

direction and the corresponding interpolated feature vector, getting (r, g, b, σ) as the output for each query point. Extracted (r, g, b, σ) of all query points on each ray are then passed into the volume rendering module of [8] to get the final RGB image, $\hat{I}_{target}$. We use MSE as the rendering loss between the ground truth RGB, I_{target} and the RGB image rendered by NeRF, $\hat{I}_{target}$. Figure 5 (main file) shows the qualitative results for this experiment. A concurrent work to this experiment is pixelNeRF [11], with the difference being that we operate on a 3D feature tensor, whereas [11] operates on a 2D feature tensor.

4. Results

In this section, we show more results on occupancy prediction, RGB view prediction, dense correspondence, vehicle tracking and pose alignment when operating on RGB input.

4.1. Occupancy prediction

Figure 1 shows the occupancies predicted by our model on ShapeNet objects. The first three columns show the renderings of the mesh created from predicted occupancies from three views. The last three columns show the corresponding ground truth. We also show multiview renderings of the predicted occupancies and the ground truth meshes as GIFs in the supplementary video on our project page and urge the readers to refer that. At inference time, we extract meshes by applying Multiresolution IsoSurface Extraction (MISE) [7]. Table 1 shows the IOU for occupancy prediction for our model on CARLA and ShapeNet datasets. For ShapeNet, we report the mean IOU over all the classes.

Dataset	IOU
CARLA	0.79
ShapeNet	0.56

Table 1: **Occupancy prediction evaluation.** Metric used is IOU.

4.2. RGB view prediction

Figure 2 shows qualitative results on the RGB novel view prediction task by our model on the ShapeNet dataset using depth as input. On the other hand, Figure 5 in the main file shows the views rendered without using depth via the NeRF volumetric renderer. Given an input view (first column), our model can render the scene from an arbitrary viewpoint (third column). We compare this rendering with the ground truth RGB for that view (second column).

4.3. Self-supervised 3D object tracking

Figure 3 shows more qualitative tracking results on CARLA and KITTI dataset. We have further attached videos of estimated and ground truth trajectories on our project page, and we urge the readers to have a look at those videos for a better understanding of the dataset complexity and our model's capabilities.

4.4. Dense correspondence

In this section, we evaluate our features on the task of establishing correspondence between two instances, I_1 (the Source entity) and I_2 (the Target entity), belonging to the same category. We use the keypoints provided by You *et al*. [10] on the ShapeNet dataset for qualitative evaluation. We assume that the input objects are already aligned. Given the dense pointcloud for both the objects, we featurize the points using the approach proposed in Section 3 of the main paper. Then, for a point (x, y, z) with feature $f_{x,y,z}$ in I_1, we find the corresponding point (x', y', z') in I_2 whose feature $f'_{x',y',z'}$ has the highest cosine similarity with $f_{x,y,z}$. Since the instances are already aligned, we limit this search to a local neighborhood of radius 10cm. Figure 4 shows the cross-object correspondence results achieved using the features learned by our model.

4.5. Using only RGB for pose estimation

In this section, we show the comparison for cross-scene and cross-object 3D alignment, while operating our model on RGB versus RGB-D input. For this we follow the same experimental setup as of Section 4.3 in the main paper. We refer our model operating on RGB as CoCoNets-RGB. We show the comparison with CoCoNetsin Table 2.

Method	cross-object	cross-view
CoCoNets	**0.18**	**0.58**
CoCoNets-RGB	0.16	0.39

Table 2: Cross-object and cross-view 3D alignment accuracies in ShapeNet dataset (mean over 4 classes: Aeroplane, Mug, Car, Chair).

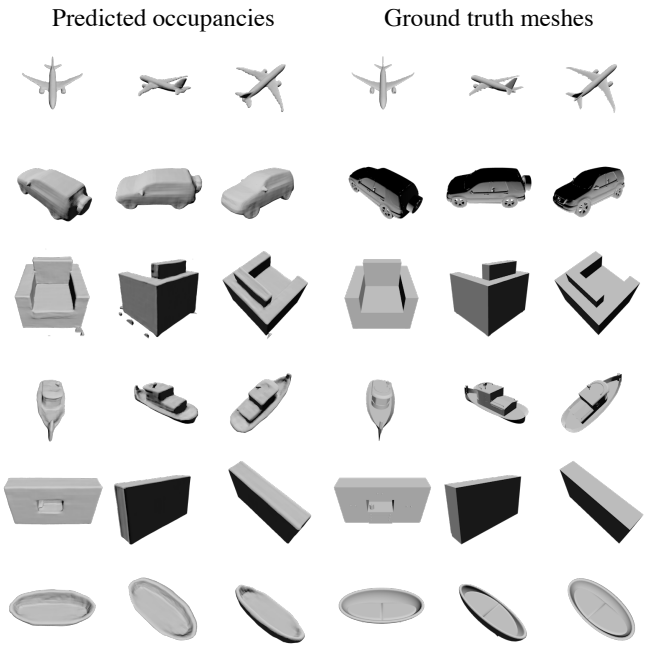

Figure 1: **Occupancies predicted by our model for ShapeNet objects.** First three columns show the renderings of the mesh created from predicted occupancies from three views. Last three columns show the same for ground truth mesh.

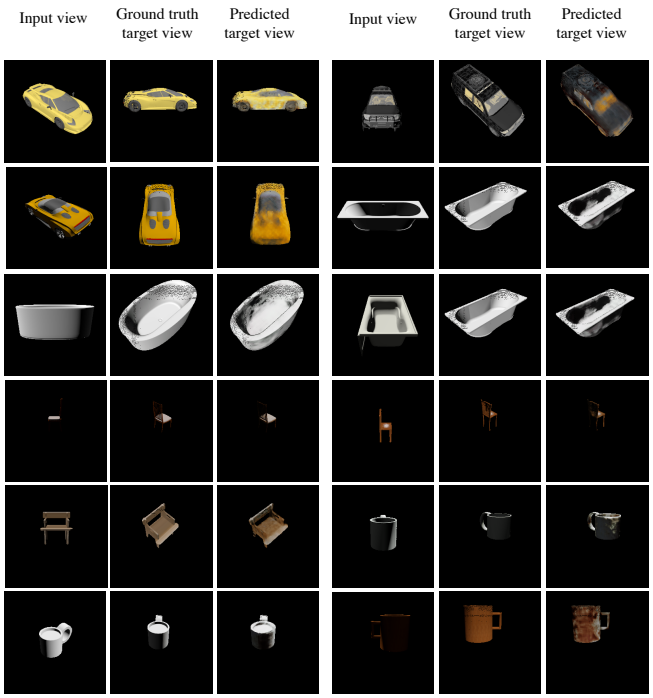

Figure 2: **Neural renders on ShapeNet dataset from CoCoNets-*OccRGB*.** The first column shows the view given as input to our model. The second column shows the ground truth target view. The third column shows the predicted target view. Columns 4-6 follow the same convention.

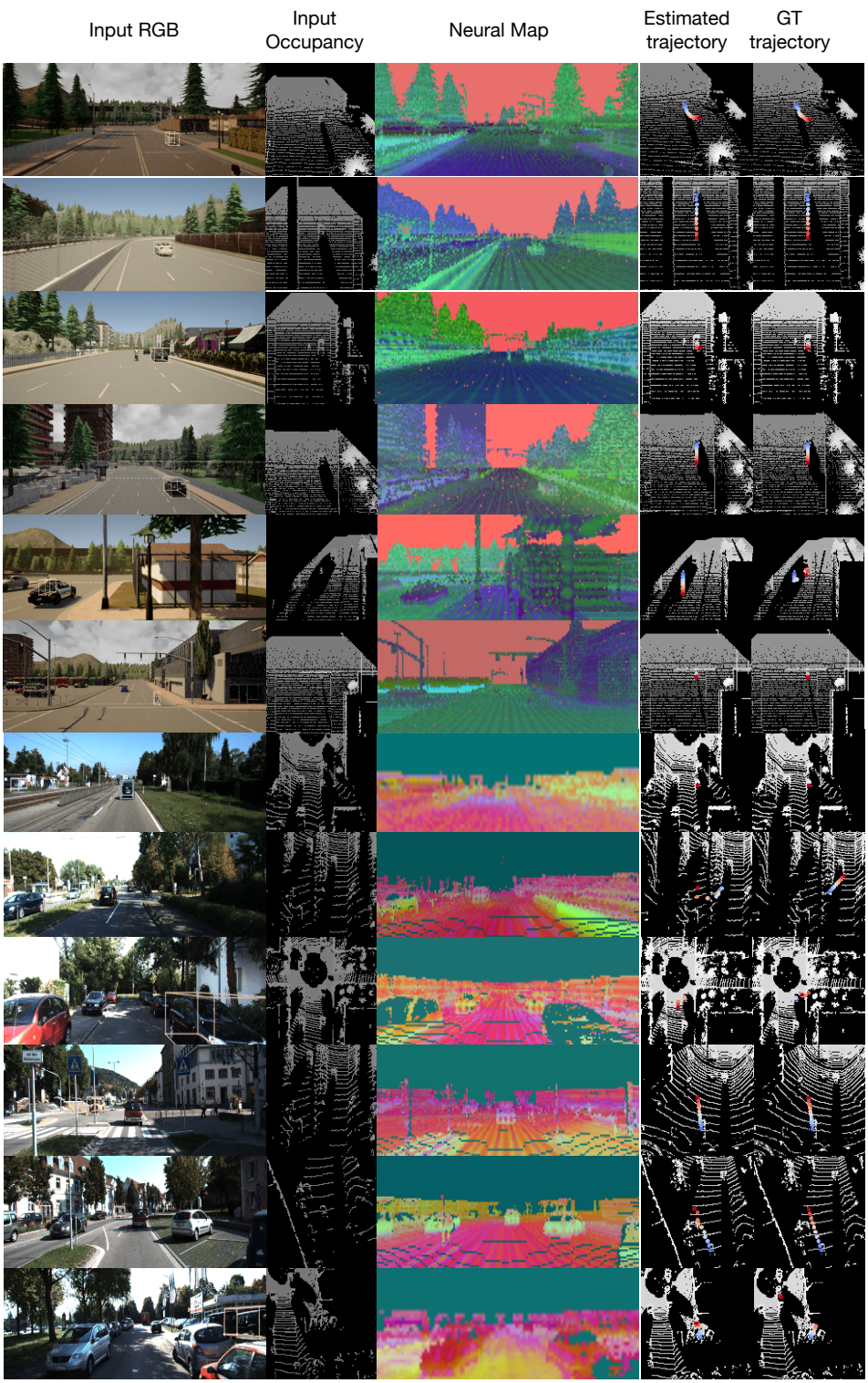

Figure 3: **Self-supervised 3D object tracking**. In the 1st and 2nd column, we visualize the RGB, the object to track and depth from the first time frame, which is given as input to our model. In the 3rd column, we visualize our inferred point features by projecting them to the same RGB image and then doing PCA compression. In the last two columns, we show the estimated and ground truth trajectories. The top six rows show our results on CARLA; the bottom six rows show our KITTI results.

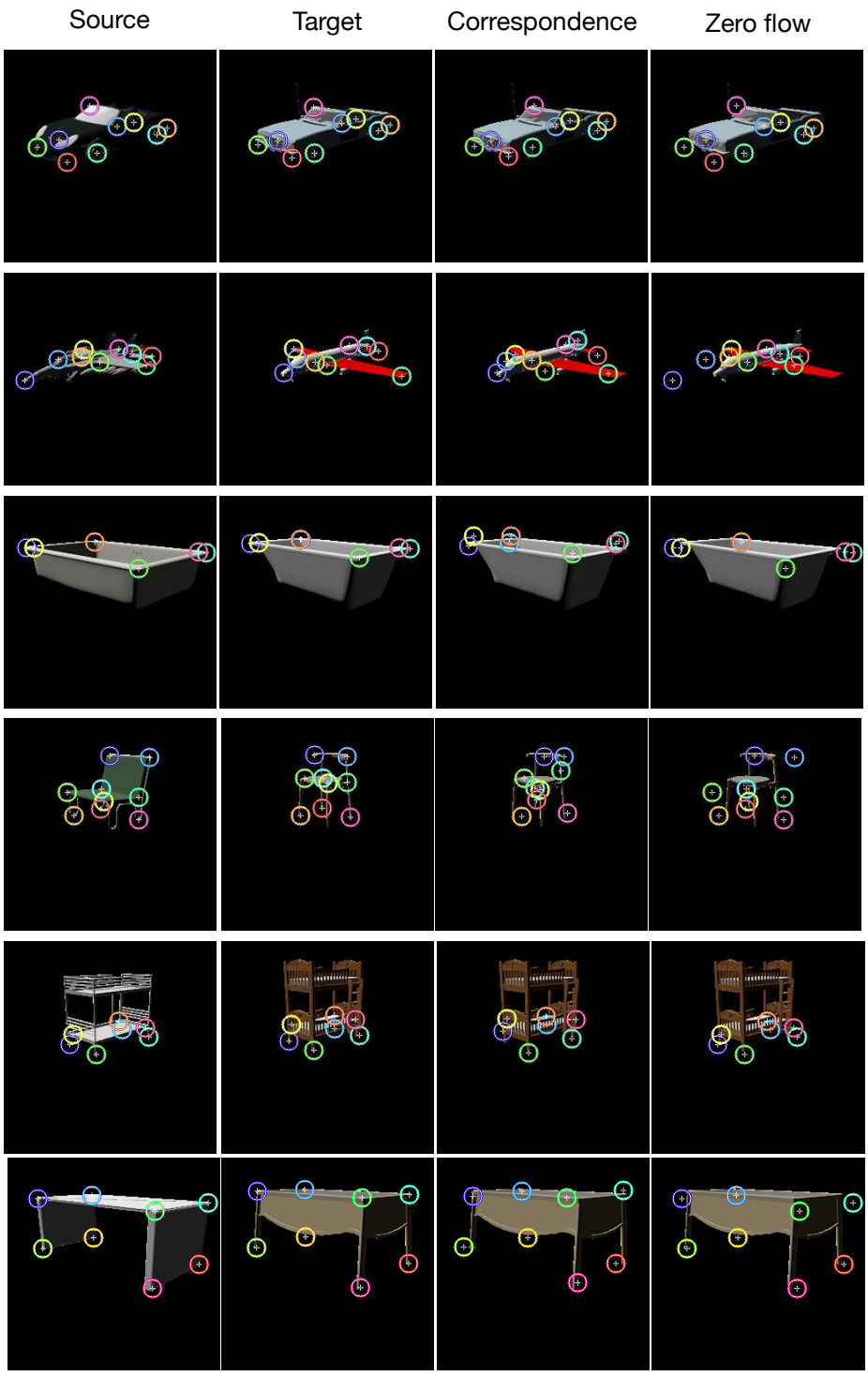

Figure 4: **Cross-object correspondence**. First column shows the Source entity. Second column shows the Target entity. The keypoints shown for these two columns are the ground truth keypoints. Third column shows the keypoints in Target entity inferred using correspondence from Source to Target. Last column shows the keypoint locations obtained assuming zero-flow correspondence from Source to Target, i.e. a point at location (x, y, z) in Source gets mapped to the same location in Target.

References

[1] 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. *arXiv:1512.03012*, 2015. 1

[2] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *CORL*, pages 1–16, 2017. 1

[3] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *CVPR*, 2012. 1

[4] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003. 2

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2

[6] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2901–2910, 2017. 1

[7] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019. 3

[8] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis, 2020. 2, 3

[9] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks, 2020. 2

[10] Yang You, Yujing Lou, Chengkun Li, Zhoujun Cheng, Liangwei Li, Lizhuang Ma, Cewu Lu, and Weiming Wang. Keypointnet: A large-scale 3d keypoint dataset aggregated from numerous human annotations. *arXiv preprint arXiv:2002.12687*, 2020. 3

[11] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelnerf: Neural radiance fields from one or few images, 2020. 3