Supplementary Material for Towards Unsupervised Learning of Generative Models for 3D Controllable Image Synthesis

Yiyi Liao[1,2,*] Katja Schwarz[1,2,*] Lars Mescheder[1,2,3,†] Andreas Geiger[1,2]
[1]Max Planck Institute for Intelligent Systems, Tübingen [2]University of Tübingen [3]Amazon, Tübingen
{firstname.lastname}@tue.mpg.de

Abstract

*In this **supplementary document**, we first provide details on the network architectures and the geometric consistency loss in Section 1. In Section 2, we present the implementation details for all baselines. Finally we show additional results and investigate how well our model captures the underlying 3D distribution in Section 3. The **supplementary video** shows synthesized animations by controlling the camera viewpoint and object poses.*

1. Implementation Details

1.1. 3D Generator

Fig. 1 illustrates the network architecture of our 3D generator g_θ^{3D}. As introduced in Section 3.1 of the main paper, $\mathcal{O} = \{\mathbf{o}_{bg}, \mathbf{o}_1, \dots, \mathbf{o}_N\}$ denotes the set of primitives that represent all objects in the scene. Each *foreground object* is described by its pose parameters $\{\mathbf{s}_i, \mathbf{R}_i, \mathbf{t}_i\}$ and a feature vector $\boldsymbol{\phi}_i$ determining its appearance. Considering that pose and appearance are independent variables, we split the pose generation and feature generation using two MLPs as shown in Fig. 1.

The background feature $\boldsymbol{\phi}_{bg}$ is a texture map attached to the inside of the background sphere. We generate $\boldsymbol{\phi}_{bg} \in \mathbb{R}^{F\times 64\times 128}$ using transposed convolutional layers. $\{\mathbf{s}_{bg}, \mathbf{R}_{bg}, \mathbf{t}_{bg}\}$ are not shown in the figure as they are constant and therefore not generated by the network.

1.2. 2D Generator

Our 2D generator g_θ^{2D} is an encoder-decoder structure based on ResNet [1]. Fig. 2 describes the network architecture in detail. Note that g_θ^{2D} is shared across the foreground and background primitives.

1.3. Alpha Composition

We fuse all outputs $(\mathbf{X}'_i, \mathbf{A}'_i, \mathbf{D}'_i)$ of our 2D generator using alpha composition in ascending order of depth at each pixel. For a single pixel, let $\{\mathbf{x}'_1, \dots, \mathbf{x}'_{N+1}\}$ denote the sorted RGB values of the N foreground objects and the background and let $\{\alpha'_1, \dots, \alpha'_{N+1}\}$ denote the corresponding alpha values. We calculate the composed pixel value $\hat{\mathbf{x}}$ as follows:

Algorithm 1 Alpha Composition

$\hat{\mathbf{x}} = \mathbf{x}'_1\alpha'_1$
$\alpha = \alpha'_1$
for $2 \le i \le N$ **do**
  $\hat{\mathbf{x}} = \hat{\mathbf{x}} + \mathbf{x}'_i\alpha'_i(1-\alpha)$
  $\alpha = \alpha + \alpha'_i(1-\alpha)$
end for

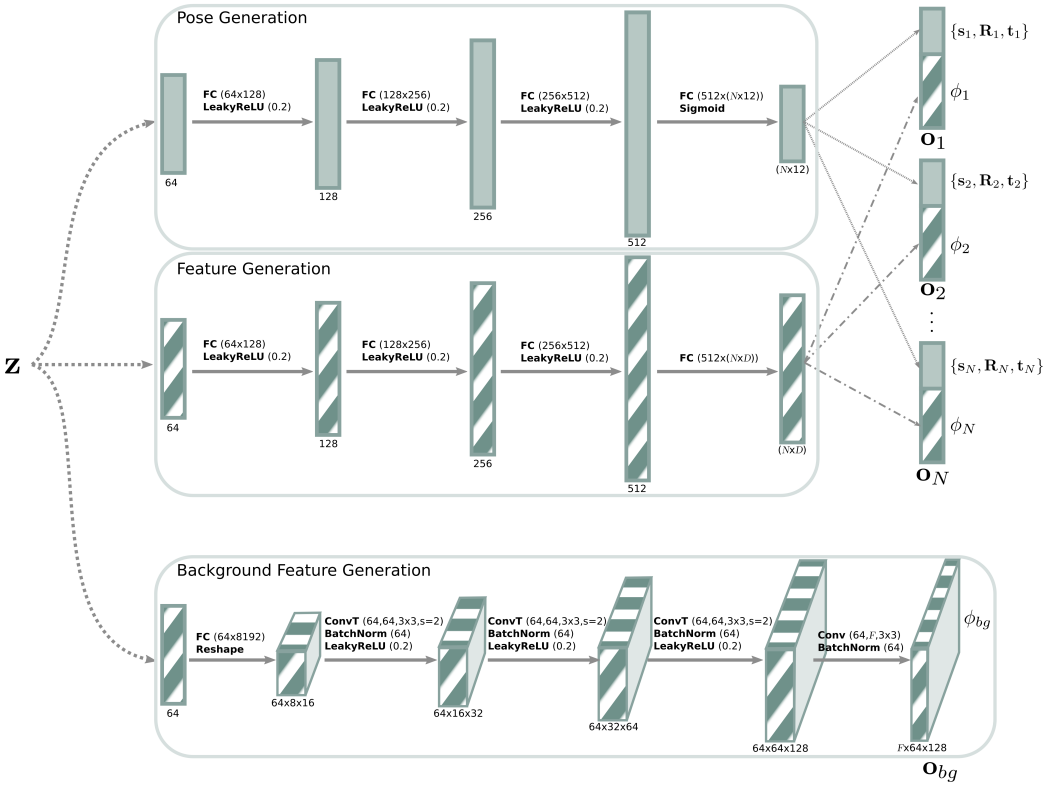

Figure 1: **Network Architecture of 3D Generator.** The first two branches generate the pose and features of the *foreground objects*. The last branch generates the texture map of the *background object*. **FC** (d_{in}, d_{out}) denotes fully connected layers, **ConvT** $(d_{in}, d_{out}, k \times k, stride)$ refers to transposed 2D convolutions, **BatchNorm** (d) refers to batch normalization [3] and **LeakyRelu** $(negative\_slope)$ is the non-linear activation function.

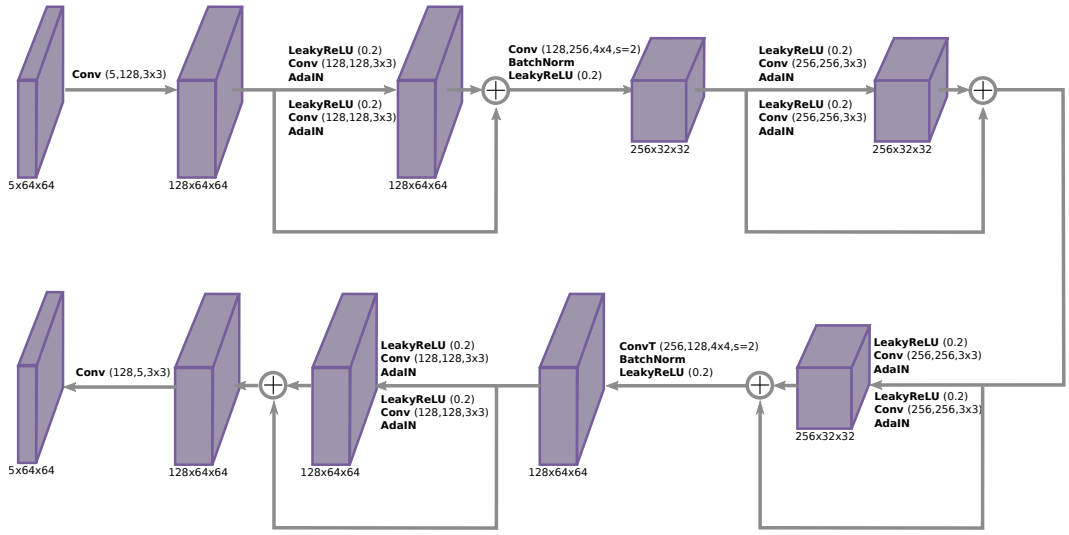

Figure 2: **Network Architecture of 2D Generator. Conv** $(d_{in}, d_{out}, k \times k, stride = 1)$ refers to 2D convolutions, **ConvT** $(d_{in}, d_{out}, k \times k, stride)$ refers to transposed 2D convolutions, **BatchNorm** (d) refers to batch normalization [3], **AdaIN** refers to adaptive instance normalizations [2] and **LeakyRelu** $(negative\_slope)$ is the non-linear activation function.

Applying this algorithm to all pixels yields the composite image $\hat{\mathbf{I}}$ which is the final output of our image synthesis pipeline. In practice, we unroll this iterative algorithm and backpropagate gradients through it in the backward pass.

1.4. Geometric Consistency Loss

In this section, we present the warping process of the geometric consistency loss in details. Our geometric consistency loss is implemented similar to [6]. In contrast to [6] that predicts the depth of the full image, we estimate the depth for each object and apply the geometric consistency loss to each object individually.

More specifically, we add random noise to the pose parameters $[\mathbf{R}_i; \mathbf{t}_i]$ of each $foreground$ primitive and observe a new pose $\left[\hat{\mathbf{R}}_i; \hat{\mathbf{t}}_i\right]$. Next, we render this primitive in both poses and generate two pairs of RGB and depth images, $(\mathbf{X}'_i, \mathbf{D}'_i)$ and $(\hat{\mathbf{X}}'_i, \hat{\mathbf{D}}'_i)$. Given the intrinsic matrix $\mathbf{K}$ and the camera extrinsic matrix $[\mathbf{R}; \mathbf{t}]$, we obtain the projection functions for both $(\mathbf{X}'_i, \mathbf{D}'_i)$ and $(\hat{\mathbf{X}}'_i, \hat{\mathbf{D}}'_i)$ respectively:

$$\begin{aligned} d_1[u_1, v_1, 1]^T &= \mathbf{K}\,[\mathbf{R}; \mathbf{t}]\,[\mathbf{R}_i; \mathbf{t}_i]\,[x, y, z, 1]^T \\ d_2[u_2, v_2, 1]^T &= \mathbf{K}\,[\mathbf{R}; \mathbf{t}]\left[\hat{\mathbf{R}}_i; \hat{\mathbf{t}}_i\right][x, y, z, 1]^T \end{aligned} \tag{1}$$

where $[u, v]$ denotes the coordinate of a pixel in the image space, i.e., $d_1 = \mathbf{D}'_i(u_1, v_1)$, $d_2 = \hat{\mathbf{D}}'_i(u_2, v_2)$ and $\mathbf{x}_1 = \mathbf{X}(u_1, v_1)$, $\mathbf{x}_2 = \mathbf{X}(u_2, v_2)$. When $[x, y, z]^T$ marks the same 3D point in the primitive coordinate, its projected RGB values in both images ($\mathbf{x}_1$ and $\mathbf{x}_2$) should be the same and the depth values (d_1 and d_2) should conform to multi-view geometry constraint (in case of no occlusion). To apply this constraint, we convert each pixel $[u_2, v_2; d_2]$ to a 3D point and project it into the other pose as:

$$d_{2\to1}[u_{2\to1}, v_{2\to1}, 1]^T = \mathbf{K}\,[\mathbf{R}; \mathbf{t}]\,[\mathbf{R}_i; \mathbf{t}_i]\left[\hat{\mathbf{R}}_i; \hat{\mathbf{t}}_i\right]^{-1}[\mathbf{R}; \mathbf{t}]^{-1}\,\mathbf{K}^{-1} d_2[u_2, v_2, 1]^T \tag{2}$$

this yields a projected RGB image $proj(\hat{\mathbf{X}}'_i)$ where $proj(\hat{\mathbf{X}}'_i)(u_{2\to1}, v_{2\to1}) = \mathbf{x}_{2\to1}$ and a projected depth image $proj(\hat{\mathbf{D}}'_i)$ where $proj(\hat{\mathbf{D}}'_i)(u_{2\to1}, v_{2\to1}) = d_{2\to1}$. Note that the pixel values of $proj(\hat{\mathbf{X}}'_i)$ and $proj(\hat{\mathbf{D}}'_i)$ are located on pixels $\{u_{2\to1}, v_{2\to1}\}$. We warp the projected images into image space $\{u_1, v_1\}$ using bilinear interpolation:

$$\begin{aligned} \tilde{\mathbf{X}}'_i &= warp(proj(\hat{\mathbf{X}}'_i)) \\ \tilde{\mathbf{D}}'_i &= warp(proj(\hat{\mathbf{D}}'_i)) \end{aligned} \tag{3}$$

We thus obtain our geometric consistency loss as (equation (6) in the main paper):

$$\begin{aligned} \mathcal{L}_{geo}(\theta) = \mathbb{E}_{p(\mathbf{z})}&\left[\sum_{i=1}^{N} \|\mathbf{A}'_i \odot (\mathbf{X}_i - \tilde{\mathbf{X}}'_i)\|_1\right] \\ + \mathbb{E}_{p(\mathbf{z})}&\left[\sum_{i=1}^{N} \|\mathbf{A}'_i \odot (\mathbf{D}_i - \tilde{\mathbf{D}}'_i)\|_1\right] \end{aligned} \tag{4}$$

where $\mathbf{A}'_i$ is the alpha map of the foreground objects, i.e. we compute the consistency loss only in foreground regions where the appearance and depth information is valid. We apply the consistency loss in both directions $1 \leftrightarrow 2$ in our implementation.

2. Baselines

We describe the implementation details of the baselines in this section, including state-of-the-art methods and variants of our method.

2.1. Vanilla GAN

We compare with [5] which is a typical representative of a vanilla 2D GAN method. We use the official implementation[1]. As we use spectral normalization in the discriminator which is not considered in [5], we train the vanilla GAN method both with and w/o spectral normalization and report the best performance. Empirically we observe that spectral normalization boosts the performance of [5].

[1] https://github.com/LMescheder/GAN_stability

2.2. Layout2Im

We use the official implementation of Layout2Im[2]. As this baseline requires 2D bounding boxes as input, we generate 2D bounding boxes and corresponding class labels for both the Car and Indoor dataset to train the model. We choose all hyperparameters as suggested in [7].

2.3. 2D baseline

The 2D baseline is a variant of our model in which the 3D generator is replaced by a 2D generator which generates 2D primitives thus not requiring a differentiable rendering layer. We now describe the details of the 2D baseline by discussing the differences to our full model.

3D Generator: The "3D Generator" of the 2D baseline generates a set of 2D primitives. Let $\mathcal{O}^{2D} = \{\mathbf{o}_{bg}^{2D}, \mathbf{o}_1^{2D}, \ldots, \mathbf{o}_N^{2D}\}$ denote the set of 2D primitives. Each foreground primitive $\mathbf{o}_i^{2D}$ is described by a set of attributes $\mathbf{o}_i^{2D} = (\mathbf{s}_i^{2D}, \mathbf{t}_i^{2D}, \boldsymbol{\phi}_i^{2D})$ where $\mathbf{s}_i^{2D} \in \mathbb{R}^2$ denote scale, $\mathbf{t}_i^{2D} \in \mathbb{R}^2$ denote translation and $\phi_i^{2D} \in \mathbb{R}^{F \times D \times D}$ denote a 2D feature map. We do not consider 2D in-plane rotation for this baseline.

Differentiable Rendering: Instead of "rendering", we use bilinear sampling to place the feature maps ϕ_i^{2D} on $\mathbf{X}_i$ according to its pose parameters $\{\mathbf{s}_i^{2D}, \mathbf{t}_i^{2D}\}$. Therefore the gradients are backpropagated to both the feature and the pose parameters. We generate both a feature map $\mathbf{X}_i$ and an alpha map $\mathbf{A}_i$, but do not predict depth information.

2D Generator: The 2D generator then generates $(\mathbf{X}'_i, \mathbf{A}'_i, \mathbf{D}'_i)$ from $(\mathbf{X}_i, \mathbf{A}_i)$, i.e. a photorealistic image $\mathbf{X}'_i$, a refined alpha map $\mathbf{A}'_i$ and a depth map $\mathbf{D}'_i$. Another difference of the 2D baseline is in alpha composition. The alpha composition we use for our method is not differentiable wrt. the depth, thus the object depths are adopted to provide a sorting order. As this is not possible for the 2D baseline, we apply differentiable soft composition [4] to learn $\mathbf{D}'_i$:

$$\begin{aligned} \mathbf{W}_i &= \frac{\mathbf{A}'_i \exp(\mathbf{D}'_i/\gamma)}{\sum_i \mathbf{A}'_i \exp(\mathbf{D}'_i/\gamma)} \\ \hat{\mathbf{I}} &= \sum_i \mathbf{W}_i \mathbf{X}'_i \end{aligned} \tag{5}$$

Here, $\gamma = 0.1$ controls the sharpness of the composition function.

2.4. Ours w/o c

For this baseline, we train our model without the background images, i.e. the model is only supervised with composite images $\mathbf{I}$. While the generator still generates both composite and background images, the discriminator is not conditioned on the class label c. Therefore, the adversarial loss for this baseline is formulated as follow:

$$\mathcal{L}_{adv}(\theta, \psi) = \mathbb{E}_{p(\mathbf{z})}[f(d_\psi(g_\theta(\mathbf{z}, c=1)))] + \mathbb{E}_{p_\mathcal{D}(\mathbf{I})}[f(-d_\psi(\mathbf{I}))] \tag{6}$$

where $g_\theta(\mathbf{z}, c=1)$ denotes the composite image from the generator.

3. Additional Experimental Results

We show additional qualitative results of all methods on the synthetic datasets in Fig. 3 and Fig. 4. We also analyze the distribution of our learned 3D parameters.

3.1. Background Supervision

As shown in Fig. 3, our method is able to disentangle the foreground objects without background supervision (ours w/o c) if the background appearance is simple. For the Indoor dataset Fig. 4 where the background appearance is more complex, our method fails to disentangle the foreground objects without any supervision. The foreground primitives vanish and the background object generates the entire image in this case. In contrast, our method is also able to disentangle foreground objects in this challenging scenario when provided with unpaired background images for supervision.

[2] https://github.com/zhaobozb/layout2im

3.2. Depth Maps

We visualize the predicted depth maps in Fig. 3 and Fig. 4. While recovering depth in an unsupervised fashion from unpaired images remains challenging, our results indicate that the overall geometric structure of the objects can be recovered. In addition, our depth prediction reflects the object translations precisely as it is refined from the depth of the projected 3D primitives.

3.3. 3D Distribution

We investigate if our 3D generator is able to capture the underlying 3D distribution of the real images given only 2D supervision on the Car w/o BG dataset. We randomly draw 3k samples from both the real images and our generated images for evaluation where each sample contains 1 to 3 cars.

Translation: We first analyze the distribution of the 3D translations $\mathbf{t}_1, \ldots, \mathbf{t}_N$ of the N foreground objects. For our ground truth data the distribution over translations is created as follows: The simulator sequentially draws the location of the foreground objects avoiding collisions among objects. Therefore, the translation distribution of of an object depends on its index in the set of foreground objects. More specifically, let $P(\mathbf{t})$ denote the discrete probability distribution of $\mathbf{t}$ that we obtain from our simulator. We assume that each $\mathbf{t}_i$ is drawn from $P(\mathbf{t})$ conditioned on its index i

$$P(\mathbf{t}) = \sum_{i=1}^{N} P(\mathbf{t}|I=i)P(I=i) \tag{7}$$

where I is a random variable denoting the indices of the foreground primitives. Similarly, let $Q(\mathbf{t})$ denote the distribution of $\mathbf{t}$ estimated by our 3D generator. For both $P(\mathbf{t})$ and $Q(\mathbf{t})$, we uniformly discretize $\mathbf{t}$ into K bins on each axis, resulting in $K \times K \times K$ possible values of $\mathbf{t}$. Note that an alternative is to consider each $\mathbf{t}_i$ as a random variable and compute the joint probability $P(\mathbf{t}_1, \ldots, \mathbf{t}_N)$, however, this leads to very high-dimensional sample space and thus hard to evaluate.

We quantitatively evaluate the distance between $P(\mathbf{t})$ and $Q(\mathbf{t})$ based on Jensen–Shannon divergence

$$\begin{aligned} \mathrm{JS}(P\|Q) &= \frac{1}{2}\,\mathrm{KL}(P\|M) + \frac{1}{2}\,\mathrm{KL}(Q\|M) \\ \mathrm{KL}(P\|Q) &= \sum_{\mathbf{t}} P(\mathbf{t}) \log\left(\frac{P(\mathbf{t})}{Q(\mathbf{t})}\right) \end{aligned} \tag{8}$$

where $M = \frac{1}{2}(P+Q)$ and $\mathrm{KL}(P\|Q)$ is the Kullback–Leibler divergence from Q to P.

Note that $P(\mathbf{t})$ and $Q(\mathbf{t})$ are not necessarily aligned due to the scale ambiguity and different camera parameters (we do not assume access to the camera parameters of our simulator for training). Specifically, there are several ambiguous parameters, including the global scale of $\mathbf{t}$, the rotation of $\mathbf{t}$ about the z-axis and the translation offset of $\mathbf{t}$ at the z-axis. Note that modifying these parameters does not change the underlying property of the distribution. Therefore, we apply grid search on these parameters to align $Q(\mathbf{t})$ to $P(\mathbf{t})$. The same transformation is applied to all $\mathbf{t}_1, \ldots, \mathbf{t}_N$ of all samples so that the spatial arrangement remains the same for each image.

Fig. 5 shows $P(\mathbf{t})$ and $Q(\mathbf{t})$, as well as $\mathrm{JS}(P\|Q)$. Note that $P(\mathbf{t})$ is not a uniform distribution due to the collision check in our simulator for rendering multiple objects. For $Q(\mathbf{t})$ we consider all 3D representation variants, including point cloud, cuboid and sphere. Our point cloud representation yields the best JS score while all the representations achieve very similar scores. We observe that our 3D generator learns to place objects on a ground plane. Moreover, it also learns a non-uniform distribution and avoids putting objects in the center which can reduce object collisions.

Rotation: We then compare the rotation distributions $P(\mathbf{R})$ and $Q(\mathbf{R})$. Here, we adopt the axis-angle representation which parameterizes the rotation matrix $\mathbf{R}$ by an axis of rotation $\mathbf{e}$ and the rotation angle ω about the axis. In contrast to $P(\mathbf{t})$ that is not uniform due to the collision check, $P(\mathbf{R})$ can be represented by a uniformly distributed $\omega \sim \mathcal{U}(0, 2\pi)$ with $\mathbf{e}$ denoting the z-axis, as the cars independently rotate about the z-axis in our simulator:

$$P(\mathbf{R}) \triangleq P(\omega) \tag{9}$$

It can be seen that the rotations in our real images have only one degree of freedom.

For comparing $Q(\mathbf{R})$ to $P(\mathbf{R})$, we first investigate if our estimated rotations follow this low degree of freedom or not. In principle the generator can create primitives with various rotations that match the coarse object shape. For instance, rotating

a cube 90 degrees around any of its symmetry axes will yield an identical shape but with another rotation. Therefore, the rotation axes $\mathbf{e}$ over all primitives are not aligned and we cannot compare them directly. However, we find that the network learns a consistent orientation over the ordered primitives, i.e. the first primitives in all samples have a similar orientation, the second and so forth. Thus, we compute the mean rotation axis $\bar{\mathbf{e}}_i$ for each index $i = 1, \ldots, N$ over all samples. We then evaluate the angles between the rotation axes $\mathbf{e}_i$ and the mean axis $\bar{\mathbf{e}}_i$. Fig. 6 (top) illustrates the distribution over these angles. As can be seen, the distribution peaks near 0 for all 3D representations (point cloud, cuboid and sphere), indicating that each primitive rotates about a stable rotation axis and mimics our ground truth distribution where the degree of freedom is 1.

Next, we evaluate the distribution of the rotation angle around these relatively fixed rotation axes

$$\begin{aligned} Q(\mathbf{R}|I=i) &\triangleq Q(\omega_i) \\ Q(\mathbf{R}) &= \sum_{i=1}^{N} Q(\mathbf{R}|I=i)Q(I=i) \end{aligned} \tag{10}$$

We uniformly discretize ω into K bins and compare $Q(\mathbf{R})$ in Fig. 6 (bottom). We observe that none of our 3D representations captures the uniform distribution $P(\mathbf{R})$. A possible explanation is that our randomly sampled camera rotations compensate for the rotation of the objects.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1

[2] X. Huang and S. J. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2017. 2

[3] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. of the International Conf. on Machine learning (ICML)*, 2015. 2

[4] S. Liu, T. Li, W. Chen, and H. Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2019. 4

[5] L. Mescheder, A. Geiger, and S. Nowozin. Which training methods for gans do actually converge? In *Proc. of the International Conf. on Machine learning (ICML)*, 2018. 3

[6] A. Noguchi and T. Harada. RGBD-GAN: unsupervised 3d representation learning from natural image datasets via RGBD image synthesis. *arXiv.org*, 1909.12573, 2019. 3

[7] B. Zhao, L. Meng, W. Yin, and L. Sigal. Image generation from layout. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019. 4, 7, 8

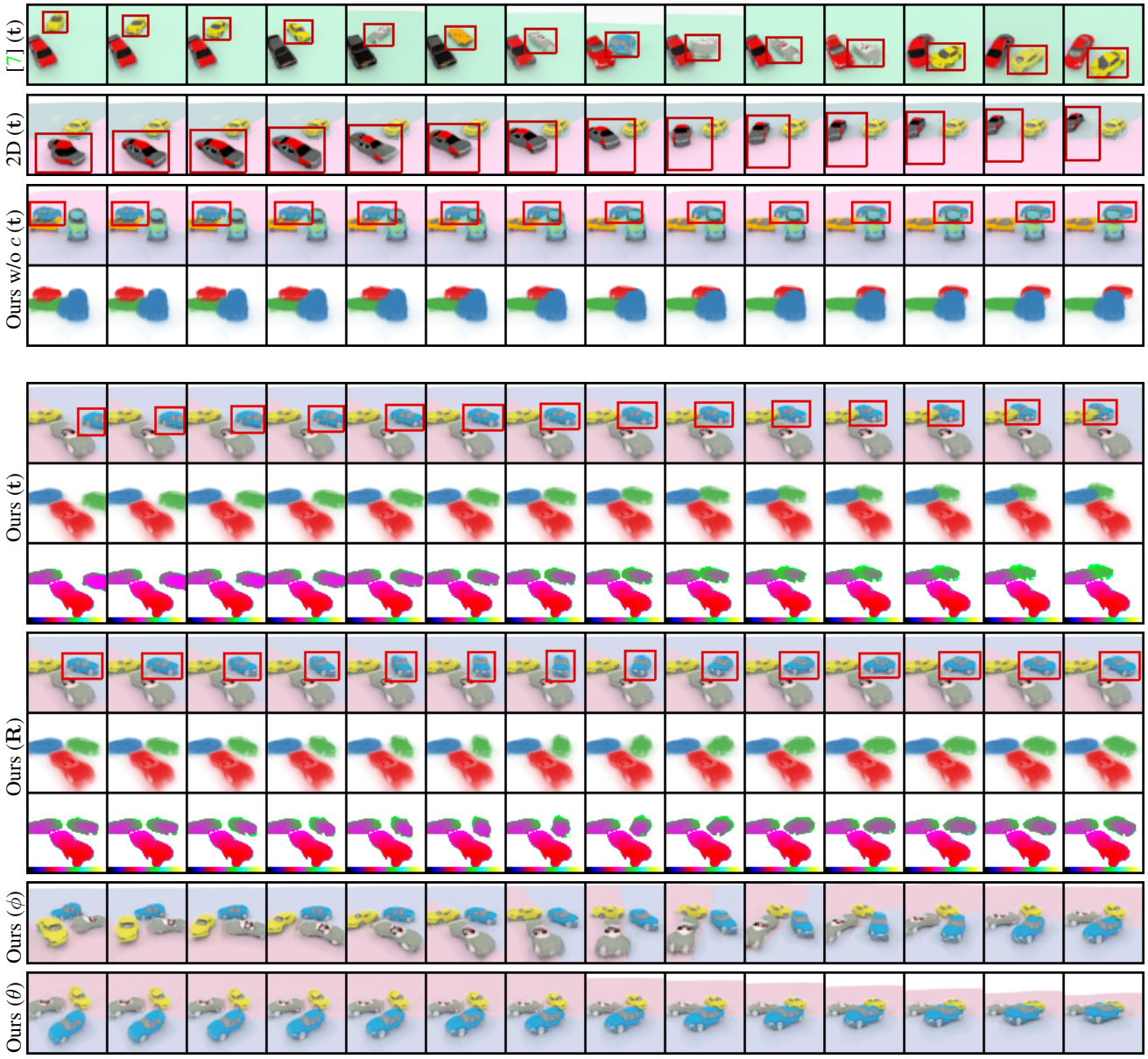

Figure 3: **Car Dataset.** We translate one object for all methods. Additionally, we rotate one object and manipulate camera poses with our method which cannot be achieved with the baselines. For **Ours w/o** c, we additionally visualize the composite alpha $\mathbf{A}'_i$. For **Ours** $(\mathbf{t})$ and **Ours** $(\mathbf{R})$, we additionally visualize the composite alpha $\mathbf{A}'_i$ and the composite depth maps $\mathbf{D}'_i$. The colorbar below each depth map illustrates the transition from zero to maximum depth. **Ours** (ϕ) refers to rotating the camera by updating the azimuthal angle ϕ. **Ours** (θ) refers to rotating the camera by updating the polar angle θ.

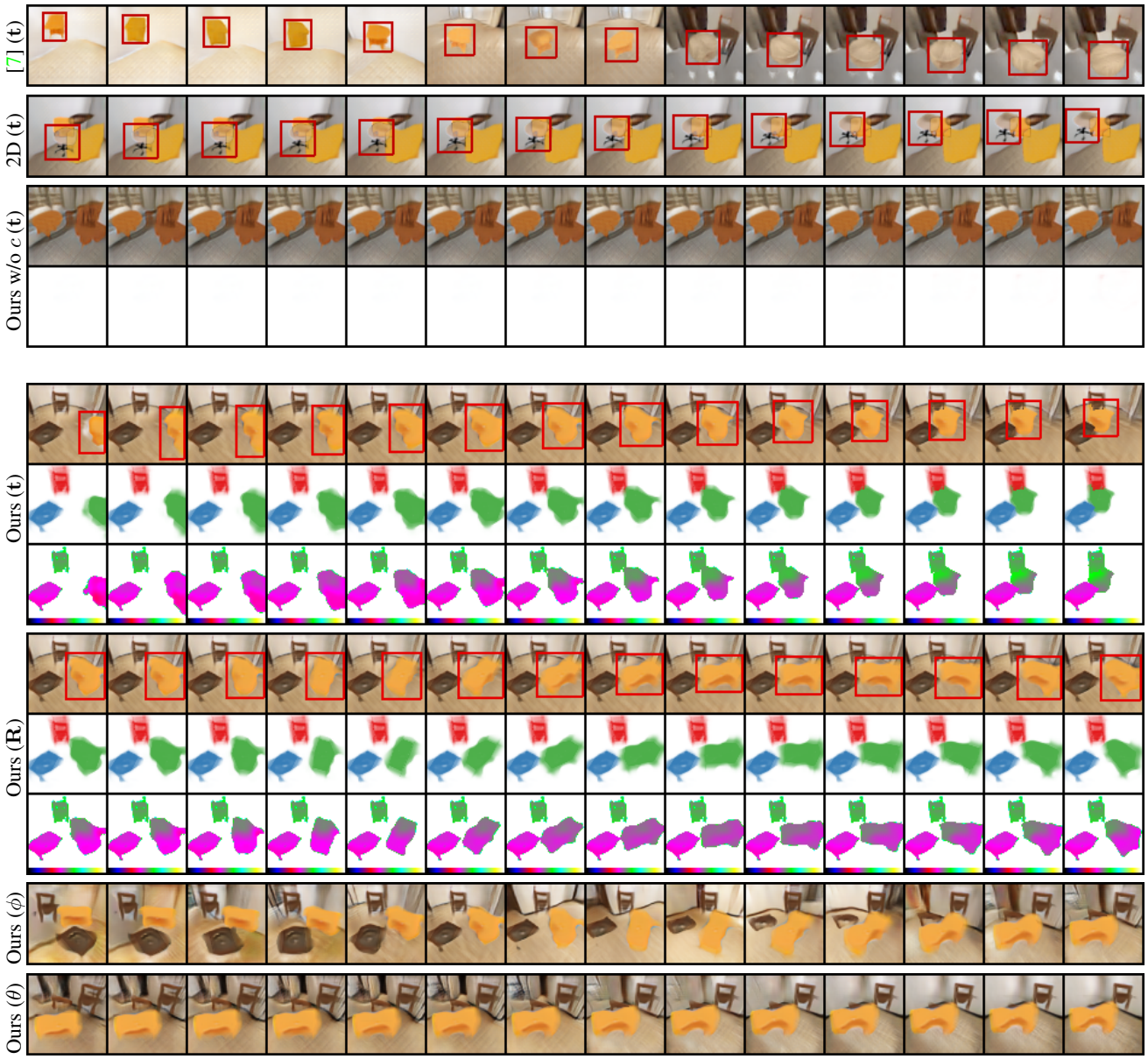

Figure 4: **Indoor Dataset.** We translate one object for all methods. Additionally, we rotate one object and manipulate camera poses with our method which cannot be achieved with the baselines. For **Ours w/o** c, we additionally visualize the composite alpha $\mathbf{A}_i'$. Note here $\mathbf{A}_i' = \mathbf{0}$ as the foreground primitive vanishes. For **Ours** $(\mathbf{t})$ and **Ours** $(\mathbf{R})$, we additionally visualize the composite alpha $\mathbf{A}_i'$ and the composite depth maps $\mathbf{D}_i'$. The colorbar below each depth map illustrates the transition from zero to maximum depth. **Ours** (ϕ) refers to rotating the camera by updating the azimuthal angle ϕ. **Ours** (θ) refers to rotating the camera by updating the polar angle θ.

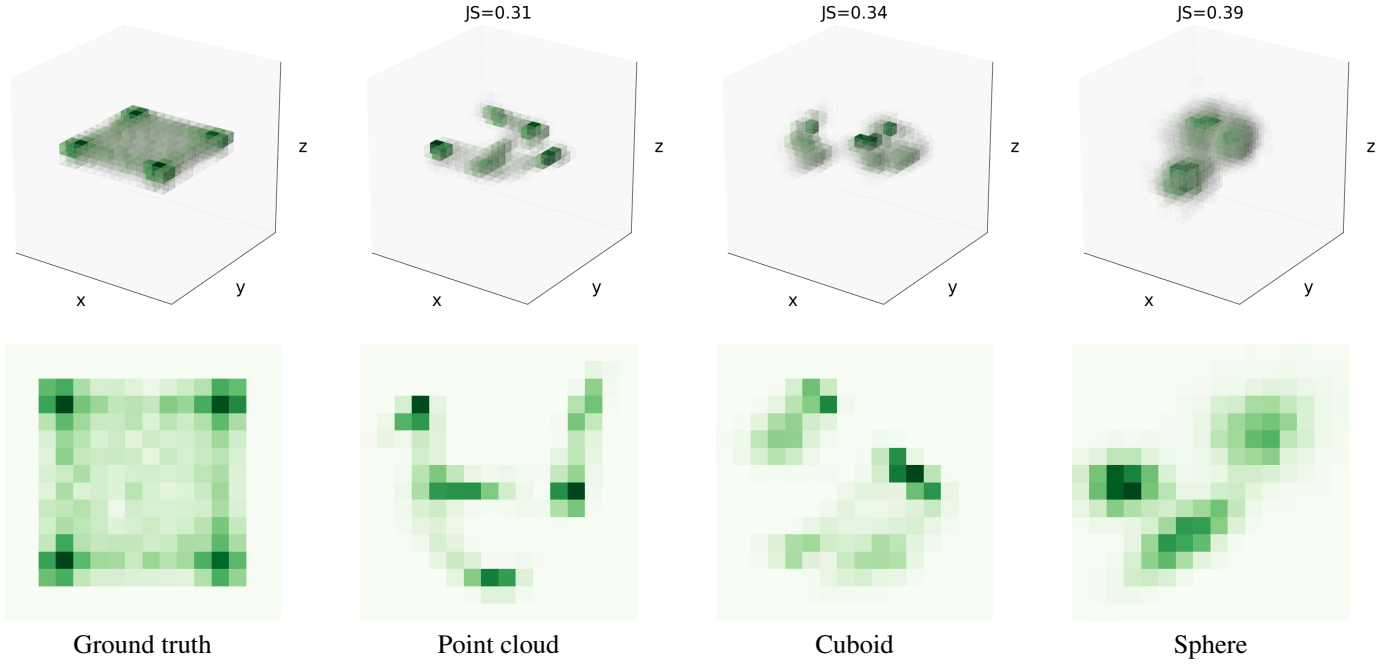

Figure 5: **Distribution of 3D translations.** From left to right, we show the underlying distribution of the 3D object translations of the real images $P(\mathbf{t})$, the distribution $Q(\mathbf{t})$ of our generated samples based on point cloud, cuboid and sphere respectively. The top row illustrates the 3D distribution and the bottom row shows the distribution in bird-eye view (marginalized out z-axis). Darker color denotes a higher probability that the translation $\mathbf{t}$ falls into the corresponding 3D voxel/2D grid.

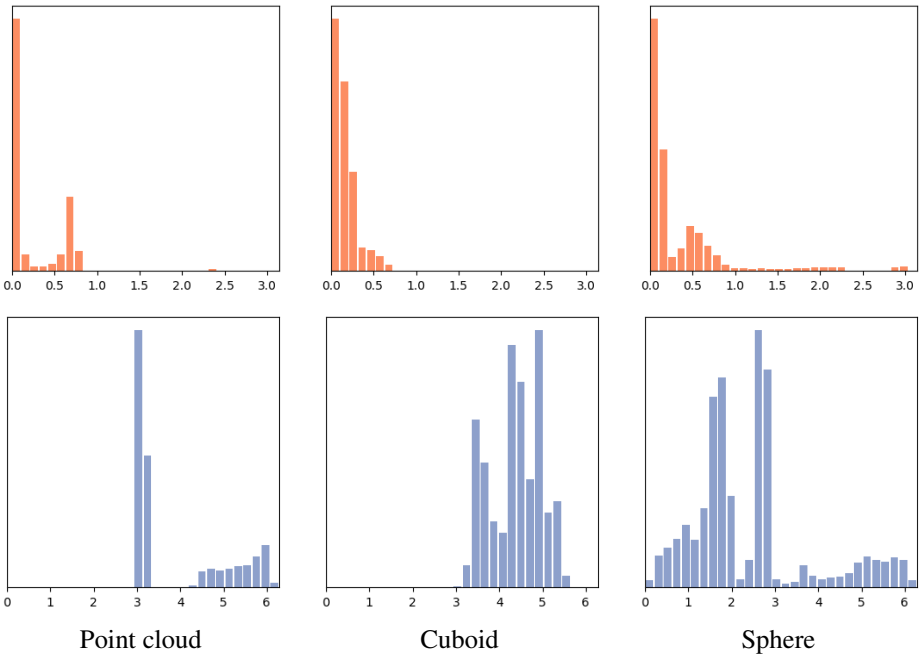

Figure 6: **Distribution of 3D rotations.** In the first row, we show the angle between the rotation axis $\mathbf{e}_i$ and the mean rotation axis $\bar{\mathbf{e}}_i$. Results show that most of the angles are close to 0, indicating that the rotation axis of each primitive is relatively stable across all samples. In the second row, we show the distribution of the rotation angle ω about the rotation axis $\mathbf{e}$. We omit the ground truth distribution here as it is a uniform distribution.