

# Neural Scene Graphs for Dynamic Scenes

## Supplementary Material

Julian Ost<sup>1,3</sup>   Fahim Mannan<sup>1</sup>   Nils Thuerey<sup>3</sup>   Julian Knodt<sup>2</sup>   Felix Heide<sup>1,2</sup>  
<sup>1</sup>Algolux   <sup>2</sup>Princeton University   <sup>3</sup>Technical University of Munich

In this supplemental document, we present additional details and results for the methods presented in the main text. Specifically, we present

- Neural Radiance Fields (NeRF) [4], which our representation nodes are based on 1.
- Point sampling from efficient ray-plane and ray-box intersections 2.
- Implementation details on ray sampling and the model architecture 3.
- Additional experiments, results and ablation study 4.
- A study and comparison on the complexity of neural scene graphs 5.

### 1. Background on Neural Radiance Fields

Several scene graph node representations, as described in the main paper, are based on Neural Radiance Fields (NeRF) by Mildenhall et al. [4]. We review NeRF as a successful approach for learning static scene representations from a collection of images, under the assumption of consistency between view points. In NeRF, scenes are modeled by a continuous, volumetric scene function  $F_\theta : (\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma)$ , approximated using a multilayer perceptron (MLP), which encodes the scene in its weights. Inputs to the volumetric scene approximation function are a position in Euclidean space  $\mathbf{x} = (x, y, z)$  and a unit vector viewing direction  $\mathbf{d} = (d_x, d_y, d_z)$  which map to a volumetric density  $\sigma$  and RGB color  $\mathbf{c} = (r, g, b)$ . The architecture for the MLP can be decomposed into two sets of linear layers, the first of which is described in Eq. (1) and takes the position  $\mathbf{x}$  and maps it to a density,  $\sigma$ . The model’s second set of linear layers maps the viewing direction  $\mathbf{d}$  concatenated with the output feature vector of the first component to a color  $\mathbf{c}$ , as in Eq. (2). This MLP’s architecture ensures consistency of the encoded scene across camera views by having the volumetric density depend only on the 3D position, and varying color with viewing direction.

$$[\mathbf{y}(t), \sigma(t)] = F_{\theta_1}(\gamma_{\mathbf{x}}(\mathbf{r}(t))) \quad (1)$$

$$\mathbf{c}(t, \mathbf{d}) = F_{\theta_2}(\gamma_{\mathbf{d}}(\mathbf{d}), \mathbf{y}(t)) \quad (2)$$

Experiments [4, 6] have shown that learning functions which map from low dimensional problem spaces to high-frequency features can be improved significantly by adding Fourier feature mappings. Therefore, it is beneficial to apply this feature encoding to all low dimensional inputs like 3D positions and directions. The original NeRF methodology used positional encodings, used in many natural language processing tasks, but has worse performance. Fourier encodings reduce the training time as compared to positional encoding and are used liberally for inputs to representation models.

The synthesized views are rendered by tracing rays  $\mathbf{r}(t)$  of a pinhole camera model through the described volumetric scene representation. With NeRF, the pixel value is approximated using volumetric integration between a near and far distance along each ray,  $t_n$  and  $t_f$ , using the equation  $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$  to sample at equal distances along the ray. The following integral is then used to compute transmitted color from the scene along each ray

$$\mathbf{C}(\mathbf{r}) = \int_{t_n}^{t_f} T(t) \sigma(\mathbf{r}(t)) \mathbf{c}(\mathbf{r}(t), \mathbf{d}) dt, \quad (3)$$

$$\text{where } T(t) = \exp\left(-\int_{t_n}^t \sigma(\mathbf{r}(s))ds\right), \quad (4)$$

with  $T(t)$  in Eq. (4) as the accumulated transmittance along a ray from the near bound  $t_n$  to  $t$ . Here,  $\sigma(t)$  and  $\mathbf{c}(t)$  are the density and color at the 3D coordinate  $\mathbf{r}(t)$ , respectively.

$$\begin{aligned} \hat{\mathbf{C}}(\mathbf{r}) &= \sum_{i=1}^N T_i(\alpha_i) \mathbf{c}_i, \text{ where} \\ T_i &= \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right) \text{ and } \alpha_i = 1 - \exp(-\sigma_i \delta_i). \end{aligned} \quad (5)$$

The continuous rendering functions (3) and (4) are approximated with numerical quadrature as in Eq. (5). To make the MLP more generalizable and prevent overfitting to one set of discrete points, samples are generated randomly between  $t_n$  and  $t$ . This ensures a smoother, continuous representation despite it being evaluated at a fixed set of discrete points at each training step.

The function is optimized given some distance between two sampling points,  $\delta_i = t_{i+1} - t_i$ , over  $\alpha$ , denoting the value used for alpha compositing. In addition to  $\alpha$ , the equation for  $\hat{\mathbf{C}}$  is differentiable with respect to the outputs  $(\mathbf{c}, \sigma)$ , therefore it can be optimized with respect to a ground truth pixel color  $\mathbf{C}$ . The rendering loss can be expressed as

$$\mathcal{L} = \sum_{\mathbf{r} \in \mathcal{R}} \left[ \left\| \hat{\mathbf{C}}(\mathbf{r}) - \mathbf{C}(\mathbf{r}) \right\|_2^2 \right]. \quad (6)$$

## 2. Sampling Points

The rendering pipeline introduced in the main paper uses discrete quadrature points along a ray, evaluated at all scene graph nodes a ray intersects. Departing from previous neural rendering methods with implicit representation models that rely on efficient sampling through iterative steps, such as ray-marching [5] or importance sampling [4], we rely on the scene graph structure to sample points efficiently in a single step. It is necessary to deviate from the proposed methodology, since the scene is sparsely covered with dynamic nodes, while the static node fully covers the rendered image. In the following, we describe the computational differences in sampling strategies in ray coordinates at the static and the dynamic nodes.

### 2.1. Ray-Plane Intersection

The background model, which represents all non-moving, static parts of a scene, is stored on sparse planes to efficiently model static components. To get a consistent view, these fixed planes are defined from a non-moving reference pose. That pose is chosen as one of the initial camera poses  $\mathbf{T}_{c,0}^{\mathcal{V}}$  from the training dataset. We use OpenGL [8] coordinate conventions and select the initial camera as the only pose with no other camera located on the side of its negative z-axis.

The first background plane is defined by a point  $\mathbf{p}$  located at  $d_n$  on the negative z-axis and its normal  $\mathbf{n}$  is the negative z-axis of  $\mathbf{T}_{c,0}^{\mathcal{V}}$ . All further planes are uniformly distributed between  $d_n$  and  $d_f$  along the normal of the first plane.

Sampling points are generated for each ray  $\mathbf{r} = \mathbf{o} + t\mathbf{d}$ , by calculating an intersection of the ray and each plane in ray coordinates  $t$  with

$$t = \frac{(\mathbf{o} - \mathbf{p}) \cdot \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}}. \quad (7)$$

In our experiments, we use  $N_s = 6$  static planes to represent the background in scenes with little to no view movement. For scenes with significant movement in the camera pose, especially in the plane’s normal direction, we use 10 planes to account for large perspective shifts in planes near the camera. The distance  $d_n$  is set to 0.5 and  $d_f$  is set to a value between 100 meters and 150 meters, depending on a scene’s depth and object distribution, such that occlusion and view interpolation can be handled accordingly.

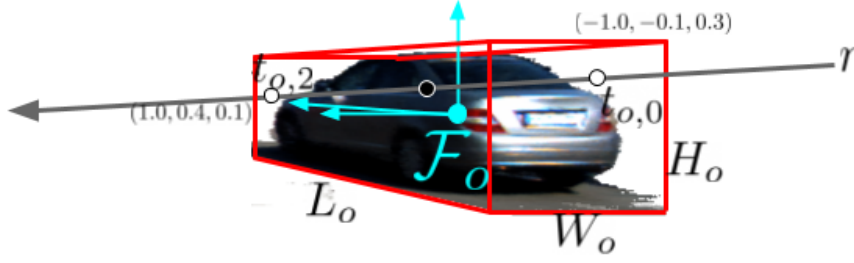


Figure 1: Valid sampling points along a ray between the entrance and exit points  $t_{o,0}$  and  $t_{o,N_d}$  of a ray and the AABB of object  $o$  for  $N_d = 2$ .

## 2.2. Ray-Box Intersection

As described in the main paper, all valid sampling points from the dynamic nodes are given in the local frame of the object hit,  $\mathcal{F}_o$  and are scaled by  $S_o$ , the inverse dimension of the bounding box enclosing the model. The boxes are now defined in this local frame as an axis-aligned bounding box (AABB) with a minimum bound  $[-1, -1, -1]$  and a maximum bound  $[1, 1, 1]$ . For each ray  $r$ , we transform its origin  $o$  and direction  $d$  into each local frame  $\{\mathcal{F}_o\}_{o=1}^{N_{obj}}$  with

$$p_o = S_o T_o^w p, \quad (8)$$

following the edges of the scene graph from  $\mathcal{W}$  to the scaled dynamic frames  $\mathcal{F}_o$ . After the rays are defined in the same reference frame as the unit scaled AABB, we can compute all ray-box intersections and entry and exit points  $t_{o,0}$  and  $t_{o,N_d}$  locally. For each intersected node, we sample equidistant quadrature points  $t$  between  $t_{o,0}$  and  $t_{o,N_d}$ , such that  $x = r(t)$  with  $x_o \in [1, -1]^3$ .

We found that  $N_d = 7$  equidistant points  $t$  per ray and object node are enough to represent dynamic objects accurately while maintaining short rendering times, see ablation study in this document. As mentioned in the main paper, this set of points is defined as

$$\{\{t_i\}_{i=1}^{N_s+m_j N_d}\}_j = \{\{t_p\}_{p=1}^{N_s}\} \cup \{\{t_{i,k}\}_{i=1}^{N_d}\}_{k=1}^{m_j} \}_j \quad (9)$$

for each ray with  $m_j$  intersecting boxes.

## 3. Implementation Details

**Latent Ray Balancing** The optimizer back-propagates through the rendering pipeline to each trainable node in  $L$  and  $F$  intersecting with the rays in a batch  $\mathcal{R}$ , and, without compensation naturally leads to an imbalance in learning rates between objects intersected more frequently. In a ray batch, each latent object descriptor is represented once per bounding box intersection with a ray in the set. Similarly, the representation networks of object types are considered once per each intersection with a node of that type. We found that a training set with an unbalanced number of intersection points with each node is not sufficient to allow optimizing over all representations jointly.

To mitigate the imbalance, we introduce a pre-processing strategy for the training set. Given all rays from all scene captures, we estimate the number of hits for each object by a ray using the ray-box intersection algorithm from the rendering pipeline. For classes less represented in this process by the maximum number of hits on a single representation node  $F_{\theta_c}$ , we replicate the rays hitting this node, such that it matches the maximum number of hits on any single class. We repeat this step for all object nodes in each class. With this procedure, we ensure that each latent object descriptor and representation model is almost equally represented in the training set.

**Model Architecture Details** For the background model and all dynamic models, our first stage uses 8 fully-connected layers and the second stage uses 4 fully-connected layers, departing from the original implementation, which uses a single fully connected layer in the second stage. All layers have a hidden size of 256.

The inputs to the static model are exactly the same as for NeRF [4], which is reviewed in Sec. 1. The inputs consist of a 3D point given in Cartesian coordinates  $x = x, y, z$  to the first stage and a 3D direction with  $d = d_x, d_y, d_z$  as input to the second. We use a Fourier feature mapping, which increases training speed for low dimensional features such as 3D coordinates learning high dimensional feature spaces as is described by Tancik et al. [6]. The 3D location  $x$  is mapped with a Fourier encoding at  $k = 10$  frequencies for

$$\gamma(p) = [..., \sin(2^k \pi p), \cos(2^k \pi p), ...] \text{ for } k = 0, ..., K - 1, \quad (10)$$

| Representation Model First Stage $F_{\text{bckg},1}$ |                             |            | Representation Model Second Stage $F_{\text{bckg},2}$ |                             |              |
|--|-----------------------------|------------|---|-----------------------------|--------------|
| Layer Name   | Note                        | Input Size | Layer Name  | Note                        | Input Size   |
| Input  | Local Position + Fourier    | 3 + 60     | Input   | Direction + Fourier+PrevOut | 3 + 24 + 256 |
| layer1   | Fully Connected ReLU        | 256        | second1   | Fully Connected ReLU        | 256          |
| layer2   | Fully Connected ReLU        | 256        | second2   | Fully Connected ReLU        | 256          |
| layer3   | Fully Connected ReLU        | 256        | second3   | Fully Connected ReLU        | 256          |
| layer4   | Fully Connected ReLU + Skip | 256        | RGB   | Fully Connected ReLU        | 256          |
| layer5   | Fully Connected ReLU        | 256        |   |                             |              |
| layer6   | Fully Connected ReLU        | 256        |   |                             |              |
| layer7   | Fully Connected ReLU        | 256        |   |                             |              |
| PrevOut, Density                                     | Fully Connected ReLU        | 256        |   |                             |              |

Figure 2: Static Model Architecture. The static representation model is implemented as a two staged multilayer perceptron. A positional encoded spatial point is the only input to the first stage. 8 fully-connected layers form the first stage, that outputs a density value and an intermediate feature vector of size 256. The 4 layer deep second stage outputs a color value.

| Representation Model First Stage $F_{c,1}$ |                                   |              | Representation Model Second Stage $F_{c,2}$ |  |              |
|--|-----------------------------------|--------------|---|--|--------------|
| Layer Name                                 | Note                              | Input Size   | Layer Name                                  | Note   | Input Size   |
| Input                                      | Local Position + Fourier + Latent | 3 + 60 + 256 | Input                                       | Direction + Fourier + Position + Fourier + PrevOut | 3 + 24 + 256 |
| layer1                                     | Fully Connected ReLU              | 256          | second1                                     | Fully Connected ReLU                               | 256          |
| layer2                                     | Fully Connected ReLU              | 256          | second2                                     | Fully Connected ReLU                               | 256          |
| layer3                                     | Fully Connected ReLU              | 256          | second3                                     | Fully Connected ReLU                               | 256          |
| layer4                                     | Fully Connected ReLU + Skip       | 256          | RGB   | Fully Connected ReLU                               | 256          |
| layer5                                     | Fully Connected ReLU              | 256          |   |  |              |
| layer6                                     | Fully Connected ReLU              | 256          |   |  |              |
| layer7                                     | Fully Connected ReLU              | 256          |   |  |              |
| PrevOut, Density                           | Fully Connected ReLU              | 256          |   |  |              |

Figure 3: Dynamic Model Architecture. The dynamic representation models are implemented as two staged multilayer perceptrons just like the static model. A positional encoded spatial point and a latent object code are input to the first stage and a positional encoded ray direction and a positional encoded object position are concatenated with the intermediate feature vector, which form the input to the second stage.

resulting in a length 63 input vector to the first stage. The Fourier encoding for the direction uses 4 frequencies and the encoded vector is concatenated with the output feature vector  $\mathbf{y}(x)$  of size 256, resulting in a input vector of length 283 to the second stage.

The dynamic model departs from the static model in the inputs passed to both stages. The latent object code  $l_o \in \mathcal{R}_{256}$  is concatenated with the encoded location and the input to the first stage. For the second stage, the object pose  $\mathbf{p}$  is Fourier encoded at 4 frequencies, and concatenated with  $\mathbf{y}(x)$  and  $\gamma_{\mathbf{y}}(\mathbf{y})$ .

## 4. Additional Experiments

We use  $N_s = 6$  planes and  $N_d = 7$  object sampling points, and scene-specific far planes at  $d_f$ , for all our experiments. Only  $N_s$  is changed for scenes to evenly learn representations from sequences with bigger movement of the ego camera, as described in subsection 2.1. The bounding box dimensions  $\mathbf{s}_o$  are scaled to include the shadows of the object. Each network  $F_{\theta_{\text{bckg}}}$  and  $F_{\theta_c}$  follows the previously detailed architecture and consists of eight fully connected layers with a skip connection from the input to the fourth layer to compute density,  $\sigma$ , and four fully connected layers to compute color. We assign a latent vector of size 256 to each node to represent the specific object class. We add an additional Fourier encoding of size 10 for  $\gamma_{\mathbf{x}}$  and 4 for the viewing direction and object pose. Finally, we train the scene graph using the Adam optimizer [3]



with a linear learning rate decay.

In the following sections, we present additional results that further validate the proposed method and an ablation study on our design and sampling choices. In addition, we encourage the reader to view our supplementary video, which shows additional results and better illustrates dynamic context.

#### 4.1. Additional Results

**Virtual KITTI Experiments** Besides the neural scene graphs that we present in the main paper, we applied our method to the Virtual KITTI 2 data set [1], a synthetic automotive data set mimicking the captured Data of the KITTI data set [2]. The renderings in Fig. 4 show that the method is also applicable to synthetic video data. There is no difference between synthetic and real data in how we apply the proposed method and we present additional decomposition and reconstruction results on the same scene in the supplementary video.



Figure 4: Renderings from Novel Scene Graphs. Randomly sampled scene graphs using the nodes of a scene from Virtual KITTI2 [1] demonstrates the neural scene graph method applied to synthetic data.

#### Novel Scene Graphs



Figure 5: Randomly sampled scene graphs, in addition to the renderings presented in the main paper.

**View Interpolation** In the main paper, we presented alternating views for a camera movement perpendicular to the background planes. In Fig. 6 we demonstrate that our method is also capable of rendering novel views in proximity to the training camera poses, similar to the majority of state-of-the-art methods. We note that only two views from a narrow-baseline stereo-pair were used for training.



Figure 6: Demonstration of view interpolation of the camera position. We translate the camera pose from the left camera to the right camera of the stereo camera setup from a KITTI sequence [2].

## 4.2. Ablation Studies

We perform an ablation study on the sampling parameters  $N_d$  and  $N_s$ , the addition of location input to the second layer and latent code on a scene from KITTI [2].

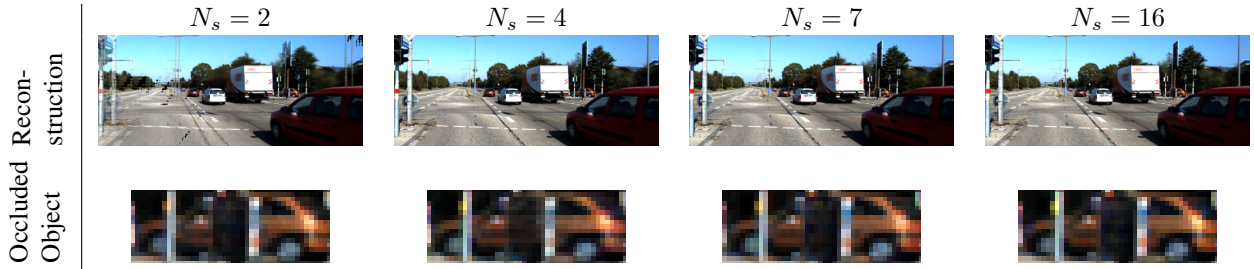


Figure 7: We vary the number of static background planes  $N_s$  and compare qualitative results. In the bottom row, we focus on an object occluded by a static part of the scene, which should be represented by the background.

| $N_s$              | 2     | 4            | 7            | 16    | 32    |
|--------------------|-------|--------------|--------------|-------|-------|
| PSNR $\uparrow$    | 18.51 | <b>26.84</b> | <b>26.78</b> | 26.36 | 25.30 |
| SSIM $\uparrow$    | 0.633 | <b>0.812</b> | 0.806        | 0.798 | 0.771 |
| LPIPS $\downarrow$ | 0.389 | <b>0.172</b> | 0.186        | 0.189 | 0.203 |

Table 1: We evaluate different plane sampling settings  $N_s$  for reconstructing a scene with the image quality metrics PSNR, SSIM [7] and LPIPS [9].

**Background Sampling** For quantitative results on the effect of the sampling  $N_s$  see Tab. 1 and for a qualitative comparison see Fig. 7. The best reconstruction quality can be achieved for  $N_s = 4$  and  $N_s = 6$ . Despite  $N_s = 4$  giving slightly better results, it fails to properly recover complex occlusion, such as the orange car behind the traffic sign in Fig. 7. The image

quality metrics also show that more planes do not favor dynamic scenes, instead they add more noise to the background representation from the dynamic objects in the scene. When only reconstructing static parts of the scene on a near and a far plane,  $N_s = 2$ , the model fails to recover some parts of the scene completely. The qualitative results in the figure show black spots and overlapping traffic lights. The occlusions that could not be handled by more planes seem to be modeled by the two planes but looking at the decomposed scene in Fig. 8 we can clearly see that the traffic sign and parts of the background are projected to the dynamic node.



(a) Rendering of all objects with  $N_s = 2$ . The occluding traffic signs and other parts of the background are projected to the dynamic node.



(b) Zoomed in rendering of objects with  $N_s = 7$ . The occluding traffic sign is handled by the static radiance fields. Distant objects present another challenge to our method.

Figure 8: Detailed comparison for sample density  $N_s = 2$  and  $N_s = 7$  on the same frame.

**Object Sampling** We present quantitative results for different values of the object sampling  $N_d$  in Tab. 2. The results only show marginal differences in all image quality metrics. The reconstruction task for the moving object gives no strong cue as to which setting is best for  $N_d$ . We assume that the scene graph structure and our rendering pipeline that reconstructs an object through ray-casting lets the models project on fewer samples if forced, and the sampling density does not have a striking effect.

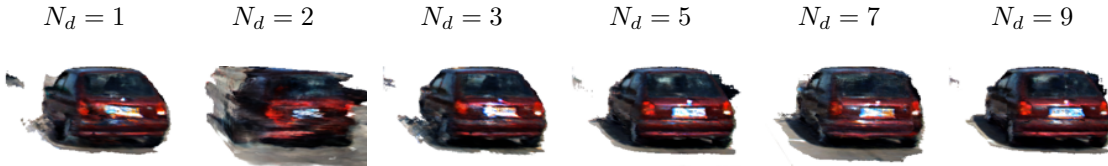


Figure 9: 7.5 deg rotation of objects around their own axes for different sampling parameters  $N_d$ .

| $N_d$              | 1     | 2     | 3     | 5     | 7     | 9     |
|--------------------|-------|-------|-------|-------|-------|-------|
| PSNR $\uparrow$    | 26.84 | 26.53 | 26.82 | 26.89 | 26.66 | 26.85 |
| SSIM $\uparrow$    | 0.815 | 0.806 | 0.813 | 0.814 | 0.806 | 0.813 |
| LPIPS $\downarrow$ | 0.171 | 0.189 | 0.173 | 0.169 | 0.186 | 0.172 |

Table 2: We quantitatively evaluate different object sampling point settings  $N_d$  on the reconstruction task.

Analyzing the metrics for reconstructed frames with dynamic objects, the model does clearly not prefer any specific sampling parameter setting. Therefore, we also show further qualitative results in Fig. 9 for novel views of objects, in which we rotate each object 7.5 deg around its own axes. For  $N_d = 1$ , we see a distorted image of the original object at that location, as would be expected from rotating a single plane inside a box. Setting  $N_d = 2$ , we see that two samples are not enough to encode any volumetric information within the representation network. Only minor differences, such as noise, artifacts from the pavement and different color shifts can be observed for more than three sampling points. We settled with  $N_d = 7$ , which has been reliable across scenes and objects, without adding significant sampling time.

The parameter  $N_d$  shows a trade-off between efficiency and quality. Pairing our method with advanced sampling techniques could allow for larger transformations without adding a vast amount of sampling points. Across all experiments shown in the paper, the chosen set of parameters delivered the most robust results without adding significant cost.

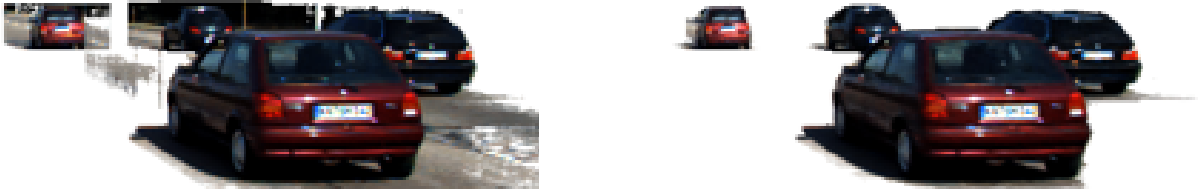


**Pose Input** We compare results from the model that we present in the main paper and a variation without an additional location input  $p_o$  to the second stage. The location input is necessary to represent the color of an object at each location in the scene and untangle the influence of global, environmental lighting from an object’s material properties. In Fig. 10 we present objects, that we rotate and translate randomly from their training location. We can see that without a global location input the color is darker and less detailed. We note that it does not completely fail and the differences are marginal for small deviations.



Figure 10: Comparison of different objects with and without the object’s location  $p_o$  as an input to the second stage of the representation model.

**Latent Code** As explained in the main paper, we introduce latent codes  $l_o$  for each object  $o$  to benefit from similarities of objects from the same class  $c$  in their shared representation function  $F_{\theta_c}$ . In Fig. 11 we show reconstruction results for the same scene that we trained with a unique representation model  $F_{\theta_o}$  for each object and with a shared representation  $F_{\theta_c}$ . We can observe that the shared model benefits from similarities in the shape and is able to recover each object without substantial artefacts, as in the neural scene graph without latent codes. Moreover, the models that do not share their weights across a class, are hardly able to represent distant object’s shapes, when compared to the approach presented in the main paper. The shared model helps representing similar shapes and model discontinuities better, and reducing artifacts.



(a) No latent code and an unique radiance field  $F_{\theta_o}$  for each object. (b) Shared representations  $F_{\theta_c}$  with latent codes  $l_o$  for each object.

Figure 11: Comparison of object renderings with and without latent codes and shared neural radiance fields.

## 5. Complexity

The efficiency of the proposed method is evaluated by comparing the number of evaluations needed to render a scene (and ray-space gradients) at training and test time.

| Method   | Points Per Ray | Ray Passes | Run Time per Pixel [sec]                  |
|----------|----------------|------------|---|
| SRN [5]  | 1              | 10         | $6.2 \times 10^{-6}$                      |
| NeRF [4] | 64 and 128     | 2          | $9.9 \times 10^{-5}$                      |
| Ours     | $6 + n_{obj}$  | 1          | $5.9 \times 10^{-6} - 1.5 \times 10^{-5}$ |

Table 3: A complexity comparison among different methodologies. Run time per pixel is calculated as the total rendering time for a single frame divided by the amount of rays traced. The run time for our method is given as a range, for rays which intersect 0 to 5 objects. For this comparison, we have chosen the same parameters as in the above qualitative comparison.

While SRN [5] and NeRF [4] have a fixed complexity, defined by the number of rays and desired quality, i.e, the number of ray marching steps or sampling points, the complexity of the proposed method depends on the number of objects a ray intersects with. In Tab. 3, we show that our method achieves comparable rendering times to state-of-the-art methods on less



complex scenes and improves substantially on complex scenes over NeRF [4], as the proposed scene graph structure lifts the requirement on importance sampling. Specifically, by combining the proposed multi-plane sampling approach for the background node, using the intersection points with dynamic nodes, and by sharing object models across multiple scene graph leaf nodes, we effectively reduce the number of required sampling points per ray.

## 6. 3D Object Detection

In the main document we briefly described the novel application of 3D object detection via inverse rendering using the proposed neural scene graphs. Here, we expand on those details and describe specifics of this approach. For all scenes, we first train a neural scene graph using our method that accurately reconstructs the graph.



Figure 12: 3D Object Detection via Inverse Rendering. For the observed image on the left, the 3D object detection results in the rendering presented in the middle after a fixed number of iteration steps. The poses and dimensions for all objects from the corresponding scene graph are drawn as 3D bounding boxes on the right.

Given an observed image  $\mathcal{I}$ , which was *not trained on*, we detect objects by fitting the scene graph  $\mathcal{S}$  by minimizing the  $\ell_1$  image loss between the synthesized image and the sample.

$$\min_{\mathcal{S}} (\|\mathcal{I} - \hat{\mathcal{I}}(\mathcal{S})\|) \quad (11)$$

We first render the static node and calculate the pixel difference  $d$  to the observed image. This is done for subsampled images at a lower resolution, to avoid local minima due to noisy deviations in the background. We then narrow the 3D search space for poses, using the image difference, restricting the search space to the volume hit by rays from pixels with a difference  $|d| > |\mu| + \alpha\sigma$ , with  $\alpha = 0.95$  as a manually chosen significance threshold. In Fig. 13 we present an example where all pixels with a difference above this threshold are gray. We further restrict positions to be within the near and far planes. Finally, we assume a 2D drivable plane for our specific automotive scenarios, further reducing the search space. In many datasets, such an assumption is likely since the positions of objects are constrained to be on some 2D plane.



Figure 13: Initialization of the search space for object detection. The poses in the first optimization step are initialized to a derived search space, calculated from the difference of the static node and the observed image.

In this search space, we sample  $n$  anchors points uniformly. From all anchor points, edges for scene graphs are sampled. The latent nodes of the graphs are randomly generated from the latent codes training distribution and assigned to the object models. We then select the best 10 graphs, using the  $\ell_1$  image loss. Given the top graphs, we iteratively optimize those by applying stochastic gradient descent for 250 iteration. The scene graphs in each step are calculated by modifying the pose and dimensions from the previously selected graphs inside the search space. Additionally, the latent code is optimized using [3], propagating the fixed scene graph to the latent nodes. After the last iteration, we select the top set of graphs again and use the edges of the scene graph that minimize the loss.

The presented approach is currently not competitive with traditional learning-based detection approaches but highlights how a learned and structured scene representation may be beneficial in other applications in the future.

## References

- [1] Yohann Cabon, Naila Murray, and Martin Humenberger. Virtual kitti 2, 2020. 5
- [2] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 5, 6
- [3] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015. 4, 9
- [4] 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. 1, 2, 3, 8, 9
- [5] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems*, 2019. 2, 8
- [6] Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains, 2020. 1, 3
- [7] Z. Wang, Eero Simoncelli, and Alan Bovik. Multiscale structural similarity for image quality assessment. volume 2, pages 1398 – 1402 Vol.2, 12 2003. 6
- [8] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL programming guide: the official guide to learning OpenGL, version 1.2*. Addison-Wesley Longman Publishing Co., Inc., 1999. 2
- [9] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 6