Inverse Neural Rendering for Explainable Multi-Object Tracking (Supplementary Information)

Julian Ost* Tanushree Banerjee* Yuval Bahat Mario Bijelic Felix Heide

Princeton University

In this supplementary document, we present information and experiments in support of the main manuscript. We provide implementation and architecture details, additional details on the object prior, analysis of failure cases, and additional results.

Contents

1. Additional Implementation Details

In the following, we provide a description of the implementation and of all design choices including the composition of the loss term, the proposed optimization schedule, heuristics applied in the matching stage of the multi-object tracker, and details on the generative object model.

1.1. Loss Terms

The test-time optimized inverse rendering of all objects in every scene and across datasets minimizes the loss term

$$\begin{aligned}\mathcal{L}_{IR+embedd} &= \mathcal{L}_{RGB} + \lambda \mathcal{L}_{perceptual} + \mathcal{L}_{embed} \\ &= \| \left(I_c - \hat{I}_c \right) \circ \hat{M}_{I_c} \|_2 + \lambda_1 \text{LPIPS}_{patch} \left(I_c, \hat{I}_{c,p} \right) + \lambda_2 \left(\alpha_S \mathbf{z}_S + (1-\alpha_S) \mathbf{z}_S^{avg} \right) + \lambda_3 \left(\alpha_T \mathbf{z}_T + (1-\alpha_T) \mathbf{z}_T^{avg} \right).\end{aligned} \tag{1}$$

This combines RGB-MSE and a learned perceptual loss term with a regularization term, for which we describe implementation details in the following.

RGB Loss. The RGB loss is computed as the pixel-wise ℓ_2-norm. Only pixels inside a mask are considered for which each pixel in the composed image at least one of the objects in the respective frame c projects too. Only pixel values inside this mask $\hat{M}_{I_c}$ are considered in the loss function. Please note, that in scenes with occluded objects only the object closest to the camera contributes to the rendered pixel in this non-volumetric rendering pipeline. We also make this assumption for the input images, given that considered objects are solid and mostly non-transparent.

*equal contribution

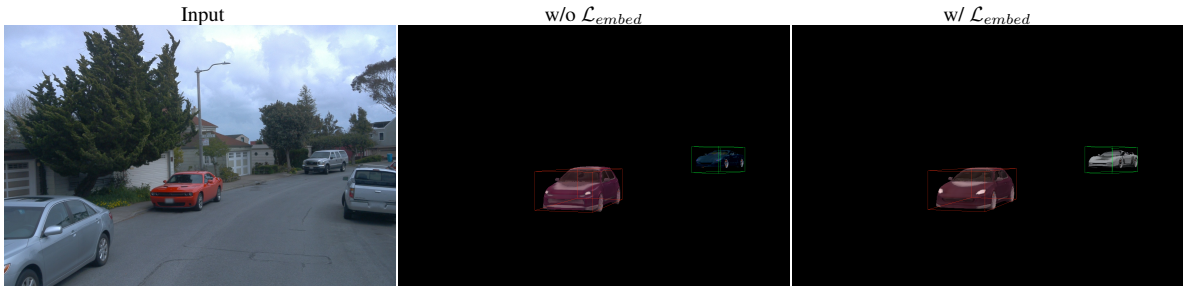

Figure 1. **Truncation Trick.** From left to right, (i) The input observed image, (ii) results without truncation regularize applied and (iii) results with a truncation regularize applied. For each of images (ii) and (iii), we also show bounding boxes for each object color-coded by the respective predicted object IDs. As shown here, applying the truncation regularizer helps us achieve more accurate textures and better shapes and colors for the predicted car surface by forcing the optimized latent codes to be "well behaved", i.e., close to the distribution of latent codes seen during the training by our representation model.

Learned Perceptual Loss. The goal of the perceptual loss is to guide the inverse rendering to match the abstracted feature-level appearance of individual objects. We use the pre-trained LPIPS [13] loss with VGG16 [10] backbone for this, which operates on rectangular images with a minimum side length of 16 pixels. To consider objects individually we crop and resize patches from $\hat{I}_c$ for each object p respectively. Mask $M_{c,p}(i,j)$ describes all pixels rendered from each object with their respective index i, j. $M_{c,p}(i,j) = 1$ if object p is projected into the respective pixel i, j from camera c. We then can describe an image patch by its top and bottom corner. The patch top corner is defined as $(u_{top}, v_{top}) = \left(i_{min,M_{c,p}}, j_{min,M_{c,p}}\right)$, the upper and left corner of a tight axis-aligned rectangle around all rendered pixels. The patch bottom corner is defined as $(u_{bot}, v_{bot}) = \left(i_{max,M_{c,p}}, j_{max,M_{c,p}}\right)$, the lower and right corner of the same axis-aligned rectangle.

Latent Embedding Regularization. Modern generative-adversarial (GAN) [3–6, 8] methods, such as the used 3D object generator [3] first map an embedding sample from a multivariate Gaussian, called z-space or distribution, into a learned embedding space, called w-space, following a different distribution. The intuition behind this is, that there are more optimal embedding distributions, which can easier be mapped to the data distribution, the GAN is generating. High-quality samples are only generated from embeddings inside the high-dimensional embedding distribution. Therefore, we regularize the optimized embedding code through inverse rendering, with

$$\mathcal{L}_{embed} = \alpha_T \mathbf{z}_T + (1 - \alpha_T)\mathbf{z}_T^{avg} + \alpha_S \mathbf{z}_S + (1 - \alpha_S)\mathbf{z}_S^{avg}, \tag{2}$$

that is Eq.(14) in the main paper. Here, α_T and α_S are set to 0.7.

We employ the *truncation trick* widely used in GAN-based generators and first presented in StyleGAN [5], where z^{avg} represents an exponential-moving average of embedding codes generated from the Gaussian training during the training of the generator. This stabilizes the optimization through inverse rendering as Fig. 1 shows.

Weighting With empirical analysis of the validation set of both datasets, we find the weighting of loss terms $\lambda_1 = 0.4$, $\lambda_2 = 3$, and $\lambda_3 = 10$ for stable and truthfully generated objects via inverse rendering.

1.2. Optimization Schedule

For the loss function presented in Eq. (5) of the main paper, we found that the schedule presented in Tab. 1 solves this optimization problem effectively, while being stable across a variety of scenes and datasets.

During the test-time optimization of all object parameters, we first fit the texture embedding in only two steps. The remaining three steps, first solve for pose, scale, and shape jointly, followed by two more steps on shape. Details on the learning rate for the respective parameters are reported in Tab. 1.

Due to the large number of hyper-parameters, when including different loss functions and terms, an exhaustive search is not possible. We therefore performed empirical investigation on small, diverse subsets of scenes to find the parameter set used. The same setting works well on all datasets and *have not been changed* for the Waymo Open Dataset [11] and the nuScenes dataset [1].

Step \ Parameter (learning rate)	$\mathbf{z}_S$	$\mathbf{z}_T$	$\mathbf{t}$	Φ	s
1	-	3×10^{-1}	-	-	-
2	-	3×10^{-1}	-	-	-
3	6×10^{-3}	-	3×10^{-3}	3×10^{-3}	1×10^{-5}
4	6×10^{-3}	-	-	-	-
5	6×10^{-3}	-	-	-	-

Table 1. **Optimization Schedule.** Test time optimization of all object parameters, the shape and texture embeddings $\mathbf{z}_S, \mathbf{z}_T$, their location $\mathbf{t}$, rotation Φ in $\mathfrak{so}(3)$ and scale s follows this schedule. First, the texture is fitted in for two steps, followed by a pose adjustment in one step and inverse rendering of the shape, defined by the respective embedding code. Green denotes the optimization of the parameter in the respective step. The learning rates for all optimized parameters are noted in each field.

1.3. Tracking Heuristics

The main paper describes the integration of test-time optimized object representations through inverse rendering into the tracking pipeline. Specifically, Eq. (6) in the main paper defines the following affinity

$$A = w_{IoU} A_{IoU} + w_z A_z + w_c D_{centroid}, \tag{3}$$

that is used to describe the similarity of tracked and detected objects in the matching step. We follow [1, 12] and assign 0 to all object pairs for which their center is more than $10m$ apart. In all other cases, we apply the weights $w_{IoU} = 0.7$, $w_z = 0.4$, and $w_c = 0.5$ between different affinity parameters. Here, A_{IoU} is the true 3D IoU of the bounding boxes, which is computed with the efficient pytorch3D [7] implementation. The affinity A_z is computed as the pair-wise cosine distance between all tracklet-detection pairs. Centroid distances between pairs are computed in addition to the IoU to compensate for larger errors in line with the camera axis, common in vision-based object detectors. In such cases, IoU might be low, but object distances are still in a reasonable range which we empirically found at $5m$ for the object detector used. Finally, we define the distance-based affinity score as

$$D_{centroid} = \text{maximum}\left(\frac{-\|\mathbf{t}_{tracklet} - t_{detection}\|_2}{d_{max}}, 0\right) \text{, with } d_{max} = 5m. \tag{4}$$

Matches that are below the threshold of 0.48 for their affinity score are not considered matched and the respective tracklet is set to lost and the detection is added as a new tracklet in the next step. Tracklets are considered "dead" and removed after a maximum of 4 consecutive lost steps.

1.4. Details on Generative Object Model

The object generator, used as the prior for car representation, follows the architecture from GET3D [3]. The generator maps two sample codes z_{tex} and z_{geo}, drawn from a Gaussian distribution for the texture and shape respectively, to samples of a 3D SDF and a texture field. Details of the architecture, training, and usage of this object model are given below.

Generator Architecture. In this work, we employ a variant described in the appendix of [3]. Following [5, 6], two Gaussian variables are mapped independently to intermediate style embedding w_S and w_T in a learned $\boldsymbol{W}$-space. Style embeddings are then used as an input to the CNN-based StyleGAN2 [6] generator. Both style embeddings for shape and texture jointly condition the generator in each block, allowing texture and shape to influence the other modality. Each intermediate feature map of the generator backbone is mapped to its respective modality through a mapping layer that is only conditioned on the respective style embedding. In the last generator layer, all feature maps are accumulated and reshaped into three feature planes. These planes represent textures as texture fields and object shapes as Signed Distance Fields (SDFs) and vertex offsets of a corresponding mesh. This forms a feature volume representation of the textured 3D object on two tri-planes.

Rendering. We employ the differentiable marching tetrahedra [9] method and extract a mesh representation from the SDF and vertex offsets, allowing more efficient rendering compared to sampling-based volumetric SDF rendering. Differentiable rasterization for meshes efficiently renders a 2D image of the respective mesh. By querying the texture field only at visible surface points, the respective vertex color can be efficiently retrieved to render the RGB image output.

Training. The model is trained using adversarial losses defined on the 2D renderings of 3D objects from the ShapeNet version 1 dataset [2]. Specifically, we use a differentiable rasterizer to render RGB images together with the silhouette masks of the objects as in [3] with a training configuration that largely follows StyleGAN2 [6] including using a minibatch standard deviation in the discriminator, exponential moving average for the generator, non-saturating logistic loss, and R1 regularization.

2. Interpretability of Failure Cases

Being able to visualize the reconstructed objects allows us to reason about failure cases. For example, in scene **(e)** in Fig. 2, we see that the initial object significantly overlaps with the background asphalt in the shadow region, causing the reconstructed car to erroneously generate a darker gray color. Thus, not only does our method allow us to reason about failure cases, but it also identifies ways in which our representation model can be modified in order to rectify such failures. For example, a generative object model with an additional component that can model different lighting conditions to account for shadows might allow us to identify and reconstruct cars in varied lighting conditions, including shadows. This can guide future work for perception tasks through inverse rendering.

Next, we analyze common failure cases using the pre-trained generator as an object prior to the presented tracking method. The visualizations allow us to assess the types of cases where this pipeline fails to track objects. Common failure cases we observed are listed below, with visualizations of such failure cases shown in Figure 2. We describe the cases corresponding to the rows (a-d, see figure lables) as follows:

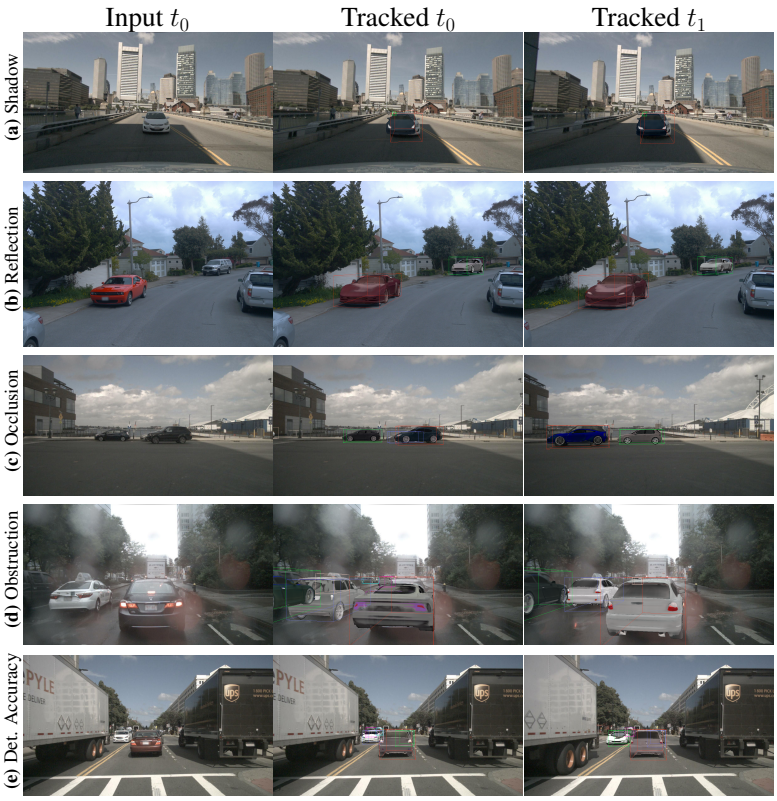

Figure 2. Examples of failure cases, such as lighting (shadows and reflections) or occluded objects, where the reconstructed object differs significantly from the observed object. These visualizations allow us to understand exactly why our model fails at reconstructing and tracking objects. This also allows us to identify ways the representation model and perception pipeline can be improved to incorporate effects that cause the method to fail.

(a) The apparent darker color of the car due to **shadows** often causes the predicted object color to be darker than the color a human would perceive the car as. In the presented case on the right, the white car is completely occupied by the shadow of the neighboring truck. While the human visual system perceives the color of the car as white, the numerical RGB value in the image is closer to grey/black. This causes the predicted embedding corresponding to the texture of the object to represent the darker color. This might cause the tracking to fail due to incorrect matching of corresponding objects in consecutive frames with and without shadows.

(b) Extreme **reflections** on the car due to the lighting conditions cause the model to try to model the RGB color of the reflection by erroneously modifying the texture of the generated object. Here, clouds in the sky are reflected as white spots on the hood and windshield of the red car, causing reflection and changes of these, influencing the generated texture as white spots. Future work that includes explicit models BRDF would be beneficial in mitigating this class of failure modes.

(c) In addition to visualization and interpretation of the object prior more traditional aspects of the perception pipeline, such as ID-switches through **occlusion** in multi-object tracking scenarios can be observed. Here, the first object in the background (green box, t_0) is misidentified as the second object (green box, t_1). Such visualization allows further reasoning about the full pipeline.

(d) Camera **obstructions** and extreme local **lighting**, such as in raindrops, lens flares, and bright lights, cause our method to erroneously predict the shape and texture of many cars to match the perceived shape of the car, causing matching to fail.

(e) Inaccuracies in the predicted pose cause the predicted car in front to not perfectly align with the observed car patch, causing there to be an overlap between the predicted car and the immediate surroundings in the observed image, here the asphalt. Since only information on the detection is available, the color of the optimized object tends to be predicted gray (since the road is of gray color, and so the optimized texture embedding is closer to gray).

3. Additional Results

In the following, we present additional tracking results on the real-world datasets on which we test our method on (see evaluation section in main paper).

Additional Results on nuScenes [1]. Although the nuScenes dataset consists of sensor data from 6 cameras, 5 radars, and 1 lidar, we tackle monocular camera-based 3D object tracking in this work, so we only use the data collected from the 6 cameras. The dataset comprises 1000 scenes, with each scene being 20s long. The test set contains 150 scenes. Each of these scenes is selected to be *interesting*, which include scenes with high traffic density (e.g., intersections, construction sites), rare classes (e.g. ambulances, animals), potentially dangerous traffic situations (e.g., jaywalkers, incorrect behavior), maneuvers (e.g., lane change, turning, stopping) and situations that may be difficult for an Autonomous Vehicle. Additional results of our method on the nuScenes dataset are listed in Fig. 3. We note that the colors of the cars are matched quite accurately. Moreover, the shapes of the cars get reconstructed as well. In turn, we can see that the tracking quality is high as visualized by bounding boxes. Note that the color of bounding boxes marks the same instance in consecutive frames.

Additional Results on Waymo Open Dataset [11]. The Waymo Open Dataset consists of 1150 scenes that each span 20s. Again, since we tackle monocular tracking, we only use the data from the 5 camera sensors. The dataset was collected by driving in Phoenix AZ, Mountain View CA, and San Francisco CA across daytime, nighttime, and dawn lighting conditions. Additional results of our method on the Waymo Open dataset are given in Figure 4. We find that our method generalizes effectively to this dataset. The colors of the cars are matched quite accurately, as shown in Figure 4. Moreover, the shapes of the cars get reconstructed as well. As such, again, tracking quality is high as visualized by bounding boxes. Note that the color of bounding boxes marks the same instance in consecutive frames.

Figure 3. Additional visualizations on nuScenes [1]. From left to right, we show (i) observed images from diverse scenes at timestep $k = 0$; (ii) an overlay of the optimized generated object and its 3D bounding boxes at timestep $k = 0, 1, 2$ and 3. The color of the bounding boxes for each object corresponds to the predicted tracklet ID. We see that our method is able to accurately reconstruct objects in diverse scenarios.

Figure 4. Additional visualizations on Waymo [11]. From left to right, we show (i) observed images from diverse scenes at timestep $k = 0$; (ii) an overlay of the optimized generated object and its 3D bounding boxes at timestep $k = 0, 1, 2$ and 3. The color of the bounding boxes for each object corresponds to the predicted tracklet ID. We see that our method is able to accurately match and track tracklets in diverse scenarios in the Waymo dataset as well, confirming that the method is dataset-agnostic.

References

[1] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020.

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

[3] Jun Gao, Tianchang Shen, Zian Wang, Wenzheng Chen, Kangxue Yin, Daiqing Li, Or Litany, Zan Gojcic, and Sanja Fidler. Get3d: A generative model of high quality 3d textured shapes learned from images. In *Advances In Neural Information Processing Systems*, 2022.

[4] Minguk Kang, Jun-Yan Zhu, Richard Zhang, Jaesik Park, Eli Shechtman, Sylvain Paris, and Taesung Park. Scaling up gans for text-to-image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10124–10134, 2023.

[5] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

[6] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8110–8119, 2020.

[7] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501*, 2020.

[8] Axel Sauer, Katja Schwarz, and Andreas Geiger. Stylegan-xl: Scaling stylegan to large diverse datasets. In *ACM SIGGRAPH 2022 conference proceedings*, pages 1–10, 2022.

[9] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. *Advances in Neural Information Processing Systems*, 34:6087–6101, 2021.

[10] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[11] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2446–2454, 2020.

[12] Xinshuo Weng, Jianren Wang, David Held, and Kris Kitani. 3d multi-object tracking: A baseline and new evaluation metrics. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10359–10366. IEEE, 2020.

[13] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 586–595, 2018.