GINA-3D: Learning to Generate Implicit Neural Assets in the Wild Supplementary Materials

Abstract

In this supplementary document, we first describe in details our proposed dataset and the processing behind it in Sec. 1. Then, we discuss various implementation details including network architectures, evaluation metrics and conditional synthesis details in Sec. 2. Next, we examine ablation of different loss terms, and evaluate stage 1 model's performance. Finally, we discuss baselines in more details in Sec. 5 and showcase mesh extraction visualization results in Sec. 6.

1. Dataset

We build the object-centric benchmark on top of the Waymo Open Dataset (WOD) [1] and our Longtail dataset. The Waymo Open Dataset (WOD) is one of the largest and most diverse autonomous driving datasets among others [2–4], containing rich geometric and semantic labels such as 3D bounding boxes and per-pixel instance masks. Specifically, the dataset includes 1,150 driving scenes captured mostly in downtown San Francisco and Phoenix, each consisting of 200 frames of multi-sensor data. Each data frame includes 3D point clouds from LiDAR sensors and high-resolution images from five cameras (positioned at Front, Front-Left, Front-Right, Side-Left, and Side-Right). The objects were captured in the wild and their images exhibit large variations due to object interactions (e.g., heavy occlusion and distance to the robotic platform), sensor artifacts (e.g., motion blur and rolling shutter) and environmental factors (e.g., lighting and weather conditions).

To construct a benchmark for object-centric modeling, we propose a *coarse-to-fine* procedure to extract collections of single-view 2D photographs by leveraging 3D object boxes, camera-LiDAR synchronization, and fine-grained 2D panoptic labels. First, we leverage the 3D box annotations to exclude objects beyond certain distances to the surveying vehicle in each data frame (e.g., $40m$ for pedestrians and $60m$ for vehicles, respectively). At a given frame, we project 3D point clouds within each 3D bounding box to the most visible camera and extract the centering patch to build our single-view 2D image collections. Furthermore, we train a Vip-Deeplab model [5] using the 2D panoptic segmentations on the labeled subset [6] and create per-pixel *pseudo-labels* for each camera image on the entire WOD. This allows us to differentiate pixels belonging to the object of interest, background, and occluder (e.g., standing pole in front of a person). We further exclude certain patches where objects are heavily occluded using the 2D panoptic predictions. Even with the filtering criterion applied, we believe that the resulting benchmark is still very challenging due to occlusions, intra-class variations (e.g., truck and sedan), partial observations (e.g., we do not have full 360 degree observations of a single vehicle), and imperfect segmentation. Finally, we use the sensor calibration information to compute ray directions for each 2D pixel, taking into account the camera rolling shutter.

Our Longtail dataset contains LiDAR point clouds and camera images, along with 3D bounding box annotations. We obtain the pseudo-labeled segmentations using the same 2D panoptic model pretrained on WOD. We apply the same *coarse-to-fine* procedure to obtain the Longtail-Vehicle benchmark.

2. More Implementation Details

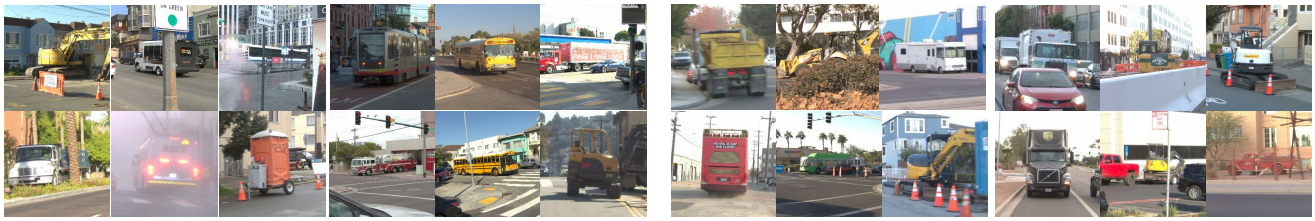
2.1. Network Architecture

All models use exponential moving average of weights.

Encoder E_ϕ. Our encoder contains three vision transformer blocks and three cross-attention blocks. The vision transformer takes input images of resolution of 256^2, and first map each patch into a 512 dimensional token. A `CLS` token is appended to the list of image patch tokens. Then, the transformer blocks are used to process the image patch tokens. Each transformer

(a) WOD-Vehicle (Box, Mask, LiDAR)

(b) WOD-Pedestrian (Box, Mask, LiDAR)



(c) Longtail-Vehicle

Figure 1. Our object-centric benchmark.

block has 8 heads, an embedding dimension of 512 and a hidden dimension of 2048. For cross-attention blocks, we first initialize tri-plane positional embedding of shape 16×16, each embedding is of 512 dimension. The tri-plane positional embedding is passed through a fully-connected layer of 512 dimension. The processed tri-plane positional embedding is then used a query input to the cross-attention transformer blocks, while the image patch tokens serve as key and value. Each cross-attention transformer block has 8 heads, an embedding dimension of 512 and a hidden dimension of 2048. Finally, the output of the cross-attention transformer blocks are passed through a fully-connected layer with Layer Normalization [7] and `tanh` activation into 16×16 tokens of 32 dimension, which is the dimension of each entry in the codebook $\mathbb{K}$.

Codebook $\mathbb{K}$. Our discrete codebook contains 2048 entries with lookup dimension of 32, which means each entry is of 32-dimensional. Codebook are initialized using fan-in variance scaling, scale equals 1 and uniform distribution. Following Yu *et al*. [8], we use l_2-normalized codes, which means applying l_2 normalization on the encoded tri-plane latents $\mathbf{e}^{3D}$ and codebook entries in $\mathbb{K}$.

Decoder G_θ - Token Transformer. The token transformer contains 3 self-attention transformers blocks. A `CLS` token is appended to the tri-plane latents. Positional encoding is used to represent 3D spatial locations. Each transformer block has 8 heads, an embedding dimension of 512 and a hidden dimension of 2048. Finally, the output of the transformer blocks are passed through a fully-connected layer with Layer Normalization [7] and `tanh` activation into 16×16 tokens of 256 dimension (and an additional `CLS` token).

Decoder G_θ - Style-based Generator. We first use a mapping network [9] to map the aforementioned `CLS` token into intermediate latent space $\mathbf{W}$. The mapping network contains 8 fully-connected layers of hidden dimension 512. The mapping network outputs a vector w of 512 dimensional. Following Karras *et al*. [9], we use w for a style-based generator. For each plane in our tri-plane representation (xy, xz, yz planes), we use a generator contains three up-sampling blocks with hidden dimensions of 512, 256 and 128 respectively. Finally, the style-based generators output tri-plane feature maps with 32 feature channels.

Decoder G_θ - Volume Rendering. Our volume renderer is implemented as 2 fully-connected layers, similar to Chan *et al*. [10]. The decoder takes as input the 32-dimensional aggregated feature vector from the style-based generator. For each pixel, we query 40 points, with 24 uniformly sampled and 16 importance-sampled. We use MipNeRF [11] as our volume rendering module. Volume rendering is performed at a resolution of 128×128.

Discriminator. We use a StyleGAN2 [9] discriminator with hidden dimensions $16, 32, 64, 128, 256$. We use R1 regularization with $\gamma = 1$.

Stage-2 Modeling M_ψ. We follow a shallower verions of the network architecture and training set up introduced in [12]. We use 12 layers, 8 attention heads, 768 embedding dimensions and 3072 hidden dimensions. The model uses learnable positional embedding, Layer Normalization, and truncated normal initialization (stddev$= 0.02$). We use the following training hyperparameters: label smoothing=0.1, dropout rate=0.1, Adam optimizer [13] with $\beta_1 = 0.9$ and $\beta_2 = 0.96$. We use a cosine masking schedule. During inference, token synthesis are performed in 10 steps.

2.2. Aligning Tri-plane to Object Scale

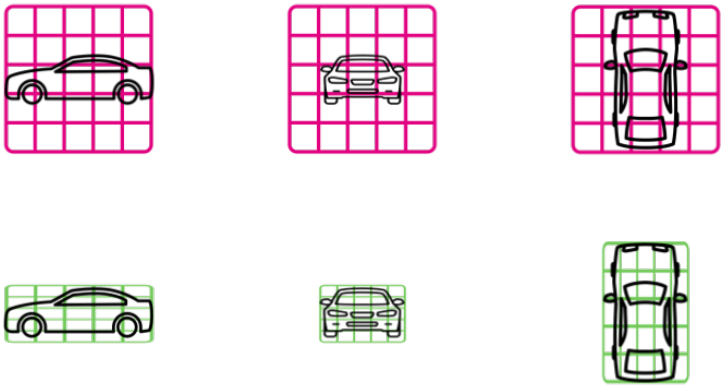


Figure 2. Illustration of using uniform tri-plane versus using scale-aligned triplane.

Since vehicles can have drastically different scales in its x, y, z directions, using a naive uniform scale tri-plane to cover the object leaves a lot of computation capacity under-utilized. As illustrated in the top row of Fig. 2, if we cover a normal sedan using uniform size tri-plane, most of the entries in the tri-plane features correspond to empty space. The problem becomes more severe for longer-tail instances of truck, bus *etc*., where the scale ratio among x, y, z become even more extreme.

To encourage a more efficient tri-plane features usage, we make tri-plane latents aligned to object scales during the coordinate feature orthographic projection step. As illustrated in the bottom row of Fig. 2, when querying feature of coordinate $p \in [0, 1]^3 \subset \mathbb{R}^3$, if we have object scale s_x, s_y, s_z, we simply scale p as $\hat{p} := \frac{p}{[s_x, s_y, s_z]} \in [0, \frac{1}{s_x}] \times [0, \frac{1}{s_y}] \times [0, \frac{1}{s_z}]$, and

query tri-plane features using $\hat{p}$. The orthographic projection follows the same tri-plane grid-sampling and aggregation as in prior works [10, 14, 15].

In the basic GINA-3D pipeline without using scaled tri-plane features, the model learns to handle object scale implicitly. In our scaled box model variations, the model leverages the object scale only in tri-plane feature orthographic projection step. The model implicitly learns to produce feature maps that align with object scale. As illustrated in the main paper, such design greatly improve model performance. We leave feeding object scale information explicit to the model as a future direction to explore.

2.3. Evaluation Metrics

We discuss in details the metrics we have used for quantitative evaluations.

Image Quality. To evaluate the image quality, we employ two metrics Fréchet Inception Distance (FID) [16] and Mask Floater-Over Union (Mask FOU) over 50K generated images. Fréchet Inception Distance (FID) [16] is commonly used to evaluate the quality of 2D images. The generated images are encoded using a pretrained Inception v3 [17] model, and the last pooling layer's output was stored as the final encoding. The FID metric is computed as:

$$\text{FID}(I_g, I_v) = ||\mu_g - \mu_v||_2^2 + \text{Tr}[\Sigma_g + \Sigma_v - 2\sqrt{\Sigma_g \cdot \Sigma_v}] \tag{1}$$

where Tr denotes the trace operation, μ_g, Σ_g are the mean and covariance matrix of the generated images encodings, and μ_v, Σ_v are the mean and covariance matrix of the validation images encodings.

We additionally measure if the generated texture forms a single full object, which is implemented by checking if the generated pixels span a connected region. We measure this by calculating percentage of pixels that are not connected. Since all images from baselines and GINA-3D are generated using a white background, we measure pixels connected components using the `findContours` function from OpenCV [18] to find connected components, and use `contourArea` to find the largest connected component, which we denote C_l. We then use the aggregated density (alpha) value to find the entire shape's projection on the image, which we denote S. Mask FOU is simply calculated mean over entire generated image set (as percentage):

$$\text{Mask FOU}(I_g) = \frac{1}{|I_g|} \sum_{i \in I_g} (1 - \frac{\text{Area}(C_{l,i})}{\text{Area}(S_i)}) \tag{2}$$

Image Diversity. We want to evaluate the semantic diversity of the generated image, which we measure with Coverage (COV) score and Minimum Matching Distance (MMD) [19] using pretrained CLIP [20] embeddings. Specifically, Coverage (COV) score measures the fraction of images in the validation set that are matched to at least one of the images in the generated set. Formally, it's defined as:

$$\text{COV}(I_g, I_v) = \frac{|\{\text{argmin}_{i \in I_v} ||\text{CLIP}(i) - \text{CLIP}(j)||_2^2 | j \in I_g\}|}{|I_v|} \tag{3}$$

Intuitively, COV uses CLIP embedding distance to perform nearest-neighbor matching for each generate image towards validation set. It measures diversity by checking what percentage of validation set is being matched as a nearest neighbor. However, COV is only one side of the story. A set of generated image can have a high COV score by having purely random generated images that are randomly matched to validation set. This issue is alleviated by the incorporation of Minimum Matching Distance (MMD), which measures if the nearest-neighbor matching yields high-quality matching pairs:

$$\text{MMD}(I_g, I_v) = \frac{1}{|I_v|} \sum_{i \in I_v} \min_{j \in I_g} ||\text{CLIP}(i) - \text{CLIP}(j)||_2^2 \tag{4}$$

Intuitively, MMD measures the average closest distance between images in the validation set and their corresponding nearest neighbor in the training set. MMD correlates well with how faithful (with respect to the validation set) elements of generated set are [19].

Geometry Quality. Due to a lack of 3D geometry ground-truth for in-the-wild data, we measure geometry quality using an existing metric Consistency score from Or-El *et al*. [21], and a Mesh Floater-Over Union (Mesh FOU) which measures if the geometry forms a single connected object. Consistency score measures if the implicit fields are evaluated at consistent

3D locations, which is an important characteristic for view-consistent renderings [21]. In practice, it measures depth map consistency across viewpoints by back-projecting depth map to the 3D space. For each model, we normalize the object longest edge to length of 10 for numeric clarity, and compare two depth maps at an angle difference of 45 degrees along the z-axis (yaw). We calculate consistency across depth maps for all images in the generated set, denote as D_g:

$$\text{Consistency}(D_g) = \frac{1}{|D_g|} \sum_{i \in D_g} \text{CD}(i, i_{rot}) \tag{5}$$

where i_{rot} represents the depth map after rotating the view point by 45 degree along z-axis.

We additionally measure if each generated shape forms a single full object, which is measured by checking if the generated mesh forms a single mesh. We measure this by calculating percentage of mesh surface area that is not connected. We use surface area over volume because we observe that volume calculation is unstable with non-watertight meshes. For each generated mesh S, we use `split` function from Trimesh [22] to find the largest connected component, which we denote C_l. Mesh FOU is simply calculated mean over entire generated mesh set M_g (as percentages):

$$\text{Mesh FOU}(M_g) = \frac{1}{|M_g|} \sum_{i \in M_g} (1 - \frac{\text{Area}(C_{l,i})}{\text{Area}(S_i)}) \tag{6}$$

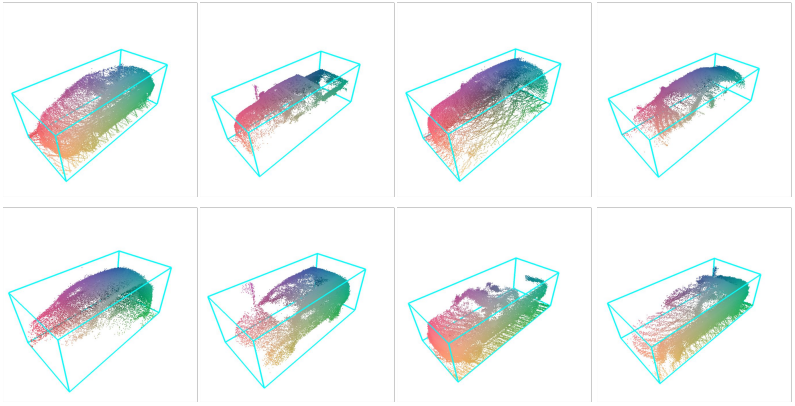


Figure 3. Illustrations of aggregated point clouds.

Geometry Diversity. We use Coverage (COV) and Minimum Matching Distance (MMD) again for measuring diversity. However, due to the lack of ground truth full 3D shape from in-the-wild data, our metric needs to be more carefully designed. A source for accurate but partial geometry that we can obtain is by aggregating LiDAR point-cloud scans for a given instance from different observations. We then uniformly subsample 2048 points from the aggregated point cloud. We show examples of aggregated point clouds in Fig. 3. As shown in the figure, the aggregated point clouds are indicative of the underlying shapes, but are incomplete. Chamfer distance, a common metric for shape similarity, calculates bi-directional nearest neighbors. However, due to incompleteness, finding the nearest neighbors of the generated points in the partial point will only result in noisy matches. Therefore, we do not measure the two-sided Chamfer distance, but measure only the distance of nearest neighbors of validation point clouds in the generated mesh. Formally, we have:

$$\text{COV}(M_g, P_v) = \frac{|\{\text{argmin}_{i \in P_v} D(i,j) | j \in M_g\}|}{|P_v|} \tag{7}$$

$$\text{MMD}(M_g, P_v) = \frac{1}{|P_v|} \sum_{i \in P_v} \min_{j \in M_g} D(i,j) \tag{8}$$

$$D(i,j | i \in P_v, j \in M_g) = \frac{1}{|i|} \sum_{x \in i} \min_{y \in j} ||x - y||_2^2 \tag{9}$$

2.4. Conditional Synthesis

We showcased in the main paper various conditional synthesis tasks, for which we provide more details here.

1st, 2nd, 3rd	~~$\mathcal{L}_{\text{GAN}}$~~	~~LPIPS~~	~~$\mathcal{L}_{\alpha}$~~	~~$\mathcal{L}_{\text{VQ}}$~~	No VQ	$\lvert\mathbb{K}\rvert = 2^{10}$	$\lvert\mathbb{K}\rvert = 2^{12}$	Full
Generative Metric (FID)	65.1	83.0	80.3	64.7	-	66.2	58.9	59.5
Recon. Metric (ℓ_2 input view)	1.78	1.92	1.44	1.62	1.01	2.21	1.81	1.55
Recon. Metric (ℓ_2 cross view)	2.42	2.28	1.55	2.14	1.71	2.28	2.30	1.83

Table 1. We perform various ablation studies on 1) removing each term in out overall loss function; 2) Removing vector quantization entirely; 3) Different codebook sizes $\mathbb{K}$. We further report stage 1 model's reconstruction quality using ℓ_2 losses for input views as well as novel views.

Discrete Conditions. We feed discrete conditions (object class, time-of-day) as additional tokens to MaskGIT. Specifically, we increase the vocabulary size by the number of classes in the discrete conditions. Object class contains 4 options: cars, truck, bus and others. Time-of-day is a binary variable of day versus night. The vocabulary thus becomes $2048 + 4$ for object class, and $2048 + 2$ for time-of-day. We feed the conditional input as an additional token to the 768 tri-plane latents by concatenating the two, resulting in an input of sequence length 769. The sequence is then fed into MaskGIT for masked token prediction as in unconditional case.

Continuous Conditions. Alternatively, we feed continuous conditions to MaskGIT by concatenating conditional input with MaskGIT intermediate layer's output. Specifically, MaskGIT first generates word embedding for each token in the sequence. We pass the continuous condition through a fully-connected layer and concatenate the output with each token's word embedding. The concatenated embedding is then passed through the rest of the network. To synthesize samples conditioned on object semantics, we feed semantic embedding from a pre-trained DINO model [23]. To condition on object scale, we pass in positional embedding of object scale. We use standard cosine and sine positional embedding of degree 6.

Image-conditioned Assets Variations. Given our mask-based iterative sampling stage, we can generate image-conditioned asset with variations. We first use stage-1 model to perform reconstruction, retrieving a full-set of predicted tri-plane latents. We then generate variations of the reconstructed instance by randomly masking out tri-plane latents. The degree of variations can be controlled by masking out different number of tokens. By masking 90% of tokens, we observe the variations are mostly reflected in generated assets under different textures. By masking out 99% of tokens, we see changes in object shapes more significantly, while the general object class remain the same. We believe how to better control the variation process is an interesting direction to explore in the future.

3. Additional GINA Visualizations

We present additional visualizations of GINA-3D model in Fig. 4.

4. Ablation on Loss Terms and Stage 1 Evaluation

Ablation study. In this experiment, we use our scaled box model, trained with LiDAR supervision as our base model. We conduct ablation studies by removing each loss, removing quantization entirely and training with different codebook sizes. As shown in Table 1, the ablaion results justify each loss term we introduced in the paper, as removing each one of them leads to higher FID compared to the full model. This finding is consistent with Esser *et al*. [24], which suggests LPIPS is important for visual fidelity. In addition, larger codebook $\mathbb{K}$ (2^{12}) has marginal impact in our setting.

Evaluating stage 1 model. We report ℓ_2 reconstruction loss (in 10^{-2}) on the input and novel views of unseen instances. The model is able to obtain better reconstruction performance by removing quantization entirely (*No VQ*), but it deprives the discrete codebook for stage 2 generative training. While generation and reconstruction correlates to some extent, performance rankings (color-coded) differ between them.

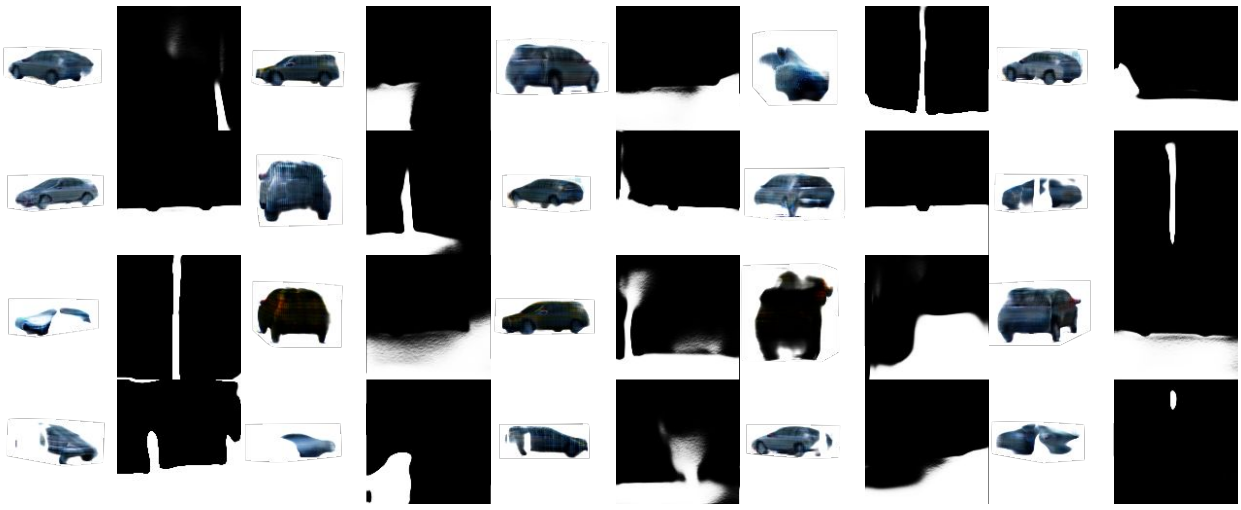
5. Discussions on Baselines

Generating the full images. Directly modeling full images of data-in-the-wild yields significant challenges. In the early stage of the project, we experimented with directly using GAN-based approaches on full images. As illustrated in Fig. 5-a,b, feeding full images without explicit modeling of occlusion makes learning challenging on our benchmark. For EG3D, we observed that training EG3D with unmasked image leads to training collapse, due to the absence of foreground and background modeling. For example, the generated image samples in Fig. 6-b lack diversity in shape and appearance (*e.g*. color).

Figure 4. Additional qualitative results of GINA-3D.

(a) Generated image samples from GIRAFFE trained on full images (view, view+45°)

(b) Generated image samples from EG3D trained on full images (random views)



(c) Generated image samples from jointly generating object RGB and occlusion masks (random views)

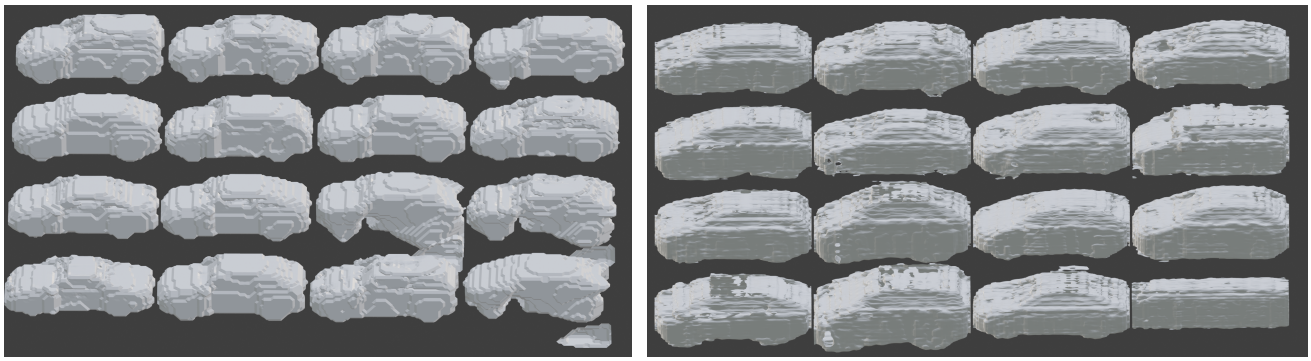
Figure 5. Additional results on generation results trained on full images. a) We trained GIRAFFE on full images; b) We trained EG3D models on full images; c) We augmented the EG3D model by jointly generating object RGB, background RGB and occlusion masks. We visualizes object RGB and its corresponding occlusion mask in alternate columns. Results suggest that it's difficult for the model to disentangle object shape and occlusion.

We clarify a key difference to pure GAN-based approaches (GIRAFFE and EG3D) is that our approach has two training stages and the masked loss is only applied in the first stage to *reconstruct* the input. In other words, masked loss cannot be

(a) Generated image samples from GIRAFFE trained on masked images (fixed views)

(b) Generated image samples from EG3D trained on masked images (fixed views)

Figure 6. Additional results on generation results trained on masked images. a) Additional visualizations of GIRAFFE baseline reported in the main paper; b) Additional visualizations of EG3D baseline reported in the main paper. Results suggest that it's difficult for GIRAFFE to disentangle rotation. Both baselines show significant occlusion artifacts.



(a) A random batch of 16 EG3D extracted meshes.

(b) A random batch of 16 GINA-3D extracted meshes.

Figure 7. Example mesh extractions from EG3D and GINA-3D.

directly applied to existing GAN-based approaches as the corresponding object mask for each generated RGB is not observed in the adversarial (encoder-free) training. Alternatively, one can still apply the masked loss by factorizing RGB, object silhouette and occlusion. We have tried many variations of this idea in the early stage without avail, as learning disentangled factors was challenging for adversarial training. We provide such examples in Fig. 5-c. In this experiment, we tried to extend EG3D by generating occlusion masks with a separate branch. However, the training became very unstable and we were not able to produce improved results beyond the original EG3D on our benchmark. As we can see, the model fails to disentangle object silhouette and occlusion. It still generates partial shape, while generating some plausible foreground occlusion. In fact, occlusion is even more challenging to generate explicitly on our data where object silhouettes and occlusion masks are entangled, as the outcome depends on the view and layout.

Generating the object images. Whitening out non-object regions has been used by EG3D (see ShapeNet-Cars in its supp.) and GET3D. It combines white color to pixels with $\alpha < 1$ during neural rendering, which implicitly supervise α. Such set up separates object pixels from the surroundings, and makes generation focused on object modeling. We follow this design and have found in our experiments that baselines fail to generate separated target object without whitening-out.

We provide additional details about the baseline methods GIRAFFE and EG3D in Fig. 6. We noticed that the learned GIRAFFE models are capable of generating vehicle-like patches but with viewpoints, occlusions and identities entangled in the latent space. For example, we generate a pair of images (in Fig. 6(a)) by varying the viewpoint variable while keeping the

identity latent variable fixed. It turns out that the generations are not easily controllable by the viewpoint variables, while the vehicle identities often change across views. The entangled representation makes the extracted meshes not very meaningful for the GIRAFFE baseline on our benchmark. Additionally, the geometry extraction becomes even harder as the rendering mask is defined at a low dimensional resolution 16^2.

6. Extracted Meshes

As mentioned in the main text, we use marching cubes [25] with density threshold of 10 to extract meshes for geometry evaluation. We showcase here random samples of extracted meshes from EG3D and GINA-3D. We show 16 examples each in Fig. 7a-7b. As we see, EG3D meshes can contain artifacts like missing parts of shape (row 3 right two). Furthermore, it shows relatively little diversity. GINA-3D not only preserves complete shapes, but also demonstrate a greater diversity, including more shape variation and semantic variation (mini-van row 2 column 4; bus row 4 column 4). Such observation is consistent with our quantitative evaluations.

However, we do observe that GINA-3D meshes can be non-watertight and contain holes. We hope to address such problems in future works. We believe that by incorporating other representations like Signed Distance Fields (SDF), the mesh quality can be further improved.

References

[1] 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 *CVPR*, 2020. 1

[2] Will Maddern, Geoffrey Pascoe, Chris Linegar, and Paul Newman. 1 year, 1000 km: The oxford robotcar dataset. *The International Journal of Robotics Research*, 36(1):3–15, 2017. 1

[3] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, et al. Argoverse: 3d tracking and forecasting with rich maps. In *CVPR*, 2019. 1

[4] 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 *CVPR*, 2020. 1

[5] Siyuan Qiao, Yukun Zhu, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. Vip-deeplab: Learning visual perception with depth-aware video panoptic segmentation. In *CVPR*, 2021. 1

[6] Jieru Mei, Alex Zihao Zhu, Xinchen Yan, Hang Yan, Siyuan Qiao, Yukun Zhu, Liang-Chieh Chen, Henrik Kretzschmar, and Dragomir Anguelov. Waymo open dataset: Panoramic video panoptic segmentation. In *ECCV*, 2022. 1

[7] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 2, 3

[8] Jiahui Yu, Xin Li, Jing Yu Koh, Han Zhang, Ruoming Pang, James Qin, Alexander Ku, Yuanzhong Xu, Jason Baldridge, and Yonghui Wu. Vector-quantized image modeling with improved vqgan. In *ICLR*, 2021. 3

[9] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *CVPR*, 2020. 3

[10] Eric R Chan, Connor Z Lin, Matthew A Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas J Guibas, Jonathan Tremblay, Sameh Khamis, et al. Efficient geometry-aware 3d generative adversarial networks. In *CVPR*, 2022. 3, 4

[11] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *ICCV*, 2021. 3

[12] Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. Maskgit: Masked generative image transformer. In *CVPR*, 2022. 3

[13] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 3

[14] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *ECCV*. Springer, 2020. 4

[15] Bokui Shen, Zhenyu Jiang, Christopher Choy, Leonidas J Guibas, Silvio Savarese, Anima Anandkumar, and Yuke Zhu. Acid: Action-conditional implicit visual dynamics for deformable object manipulation. 2022. 4

[16] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *NIPS*, 30, 2017. 4

[17] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016. 4

[18] Itseez. Open source computer vision library. https://github.com/itseez/opencv, 2015. 4

[19] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *ICML*. PMLR, 2018. 4

[20] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*. PMLR, 2021. 4

[21] Roy Or-El, Xuan Luo, Mengyi Shan, Eli Shechtman, Jeong Joon Park, and Ira Kemelmacher-Shlizerman. Stylesdf: High-resolution 3d-consistent image and geometry generation. In *CVPR*, 2022. 4, 5

[22] Dawson-Haggerty et al. trimesh. 5

[23] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *ICCV*, 2021. 6

[24] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *CVPR*, 2021. 6

[25] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987. 10