Learning 3D Scene Priors with 2D Supervision
Supplementary Material

Yinyu Nie[1] Angela Dai[1] Xiaoguang Han[2] Matthias Nießner[1]
[1]Technical University of Munich [2]The Chinese University of Hong Kong (Shenzhen)

In this supplementary material, we provide details of our network architecture in Sec. A, additional details of loss function design in Sec. B, data collection setup for 3D-Front in Sec. C, latent vector optimization setup for single-view reconstruction in Sec. D, additional details of baselines in Sec. E, additional shape retrieval details in Sec. F, additional ablation studies in Sec. G, and more qualitative results on scene synthesis and single-view reconstruction in Sec. H. *Our code and data will be publicly released.*

A. Network Specifications

We detail the full list of layer parameters in this section. We denote the MLP layers in our network uniformly by MLP$[l_1, l_2, ..., l_d]$, where l_i is the number of neurons in the i-th layer. Each fully connected layer is followed by a ReLU layer except the final one. We set the latent hypersphere dimension D_z=512 in Sec. 3.1. Every generated object feature $\boldsymbol{x}_1, ..., \boldsymbol{x}_N$ and the start token $\boldsymbol{x}_0$ are with the dimension of D_z=512 as well.

A.1. Permutation-invariant Transformer

From the input latent vector $\boldsymbol{z} \in \mathbb{R}^{D_z}$, we use a transformer to generate N object features autoregressively. N is the maximal number of objects in a scene. For ScanNet scenes [4], we use $N = 53$. For 3D-Front [6] bedrooms and living rooms, we use $N = 13, 28$, respectively. We use the transformer library from [1] to build the architecture. Since our method does not require 3D supervision, we train the transformer without using teacher-forcing strategy.

Transformer Encoder With the previous object features $\boldsymbol{x}_0, \boldsymbol{x}_1, ..., \boldsymbol{x}_{k-1}$ as input, we adopt a transformer encoder to produce the scene context feature $\boldsymbol{F}_k \in \mathbb{R}^{k\times 512}$. The encoder consists of a single layer of multi-head self attention without positional encoding, followed by a feed forward network. We use four heads in our multi-head attention, with the input and output dimension d_{model}=512. The feed forward network is a two-layer MLP$[1024, 512]$ but with GeLU activation. For other parameters we keep the default setting as in [1].

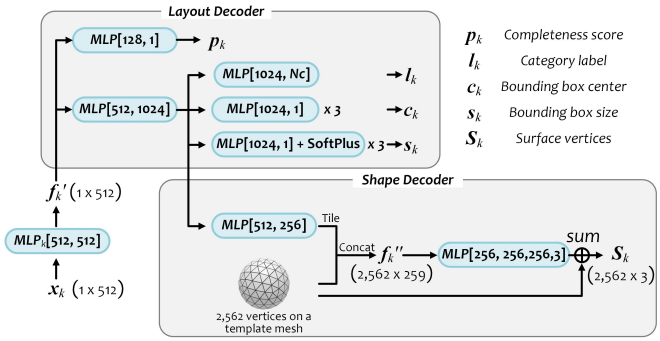

Figure 1. Network architecture details in layout and shape decoders.

Transformer Decoder The transformer decoder generates the next object feature $\boldsymbol{x}_k$ from $\boldsymbol{F}_k \in \mathbb{R}^{k\times 512}$. It is a single layer of multi-head cross attention without positional encoding, followed by a feed forward network. Similar to the encoder, we use four heads in the cross attention module with d_{model}=512. For the cross attention module, we use the scene context $\boldsymbol{F}_k$ as keys and values, and the latent vector $\boldsymbol{z}$ as the query to infer the next object $\boldsymbol{x}_k$. Similar to the encoder, the feed forward network is a two-layer MLP$[1024, 512]$ with GeLU activation. We keep the default setting for other parameters.

This generates an object feature sequence $\{\boldsymbol{x}_k\}$, $\boldsymbol{x}_k \in \mathbb{R}^{512}, k = 1, ..., N$, from our transformer.

A.2. Layout and Shape Decoder

For each object feature $\boldsymbol{x}_k$, we decode its class label $\boldsymbol{l}_k \in \mathbb{L}$, axis-aligned 3D bounding box with size $\boldsymbol{s}_k \in \mathbb{R}^3$ and center $\boldsymbol{c}_k \in \mathbb{R}^3$, as well as completeness score $\boldsymbol{p}_k$, via our layout decoder, and decode its mesh surface $\boldsymbol{S}_k$ via our shape decoder. We detail the layer specifications with parameters in Fig. 1. For $\boldsymbol{s}_k$, we process it with a SoftPlus layer to make sure object sizes are positive values. N_c denotes the number of object class categories. For ScanNet, we use the official 'nyu40class' object category split and use 19 common indoor furniture categories. For 3D-Front, we follow [**?**] and use 22 object categories for bedrooms, and 15 object categories for living rooms. The category names

are listed below. All baselines are trained and tested with a unified category split.

```
# 19 ScanNet categories
['void', 'bathtub', 'bed', 'bookshelf',
'cabinet', 'chair', 'counter', 'desk',
'dresser', 'lamp', 'night stand',
'refridgerator', 'shelves', 'sink', 'sofa',
'table', 'television', 'toilet', 'whiteboard']

# 22 3D-Front bedroom categories
['void', 'armchair', 'bookshelf', 'cabinet',
'ceiling_lamp', 'chair', 'children_cabinet',
'coffee_table', 'desk', 'double_bed',
'dressing_chair', 'dressing_table', 'kids_bed',
'nightstand', 'pendant_lamp', 'shelf',
'single_bed', 'sofa', 'stool', 'table',
'tv_stand', 'wardrobe']

# 15 3D-Front living room categories
['void', 'armchair', 'bookshelf', 'cabinet',
'ceiling_lamp', 'chair', 'coffee_table', 'desk',
'pendant_lamp', 'shelf', 'sofa', 'stool',
'table', 'tv_stand', 'wardrobe']
```

In our layout decoder, all objects are constrained to be located above the floor, which can be achieved by

$$\boldsymbol{c}'_y = \boldsymbol{c}_y + \boldsymbol{s}_y/2; \quad \boldsymbol{c}_y > 0, s_y > 0, \tag{1}$$

where c_y is the vertical coordinate of an object center. $c_y \geq 0$ as it is outputted after a ReLU layer; $\boldsymbol{s}_y$ is the height of an object bounding box. In the following calculations, we use $\boldsymbol{c}'_y$ as the vertical coordinate of the object center.

A.3. Differentiable Rendering Setup

Given a scene with generated objects, we render it back to input views with differentiable rendering. Thus in each input view, there is a rendered instance map with object silhouettes. We use PyTorch3D [8] to implement our rendering process.

During mini-batch training, we randomly select 20 views from all frames in each scene. We render the generated scene back to the 20 views with a lower resolution to the original input image (1/4 of the original[1]) due to the high GPU memory consumption of differentiable rendering. We render 50 faces per pixel in mesh rasterization [8]. Blur radius and blend sigma are 1e-4. A soft silhouette shader is applied to render instance silhouettes. Thus, in each view we obtain two maps: one differentiable silhouette map and one instance ID map, from which we can obtain the instance silhouette $\boldsymbol{r}_k^p$ of each object under 20 views.

B. Additional Loss Details

B.1. Hungarian Matching

For each object prediction $\boldsymbol{o}_k$ in a scene (k=1,...,N), we have its class label $\boldsymbol{l}_k$ and a set of 2D bounding boxes $\boldsymbol{B}_k = \{\boldsymbol{b}^1, ..., \boldsymbol{b}^T\}_k$ in all input views T, where T indicates 20 random views in a mini-batch. Additionally, we also have ground-truth objects $\{\boldsymbol{o}_j^{gt}\}$ in this scene (j=1,...,n). For each $\boldsymbol{o}_j^{gt}$, it has a class label $\boldsymbol{l}_j^{gt}$ and a set of 2D bounding boxes $\boldsymbol{B}_j^{gt} = \{\boldsymbol{b}^1, ..., \boldsymbol{b}^{T_j}\}_j^{gt}$ in T_j views, where T_j denotes all the observable views of $\boldsymbol{o}_j^{gt}$ ($T_j \subseteq T$). N is the maximal object number among all scenes. n is the object number of this target scene in a mini-batch.

For each object prediction $\boldsymbol{o}_k$, we use the Hungarian algorithm to find its optimal bipartite matching $\boldsymbol{o}_{\sigma(k)}^{gt}$ for loss calculation in Sec. 3.4, $\sigma(k) = 1, ..., n$ or $\varnothing$. We use the implementation of Hungarian algorithm from [3].

In our case, a prediction $\boldsymbol{o}_k$ and a ground-truth object $\boldsymbol{o}_{\sigma(k)}^{gt}$ are characterized by $(\boldsymbol{B}_k, \boldsymbol{l}_k)$ and $(\boldsymbol{B}_{\sigma(k)}^{gt}, \boldsymbol{l}_{\sigma(k)}^{gt})$, respectively. As in [3], our matching cost takes into account the similarity between both class labels $<\boldsymbol{l}_k, \boldsymbol{l}_{\sigma(k)}^{gt}>$and 2D bounding boxes $<\boldsymbol{B}_k, \boldsymbol{B}_{\sigma(k)}^{gt}>$. Thus the matching problem can be formulated as

$$\hat{\sigma} = \underset{\sigma}{\operatorname{argmin}} \sum_{k=1}^{n} \left[\mathcal{L}_{\text{match}}^{l}(\boldsymbol{l}_k, \boldsymbol{l}_{\sigma(k)}^{gt}) + \lambda_B \mathcal{L}_{\text{match}}^{B}(\boldsymbol{B}_k, \boldsymbol{B}_{\sigma(k)}^{gt}) \right], \tag{2}$$

where $\mathcal{L}_{\text{match}}^{l}$ and $\mathcal{L}_{\text{match}}^{B}$ are pair-wise matching costs. Note that we find the bipartite matching for the first n predictions only, i.e., $\{\boldsymbol{o}_k\}$, k=1, ..., n, $n \leq N$, and predict their completeness scores as ones. For the additional predictions $\{\boldsymbol{o}_k\}$, k=n+1, ..., N, we predict their completeness scores as zeros. The completeness score here indicates whether the generated scene is complete or not. We set λ_B=5 to balance the importance of the two costs.

We keep the definition of $\mathcal{L}_{\text{match}}^{l}$ as in [3], and formulate our $\mathcal{L}_{\text{match}}^{B}$ as

$$\mathcal{L}_{\text{match}}^{B}(\boldsymbol{B}_k, \boldsymbol{B}_{\sigma(k)}^{gt}) = \frac{1}{T_{\sigma(k)}} \sum_{p \in T_{\sigma(k)}} L_1(\boldsymbol{b}_k^p, \boldsymbol{b}_{\sigma(k)}^{p,gt}), \tag{3}$$

where $T_{\sigma(k)}$ denotes all visible views of $\boldsymbol{o}_{\sigma(k)}^{gt}$; $\boldsymbol{b}_k^p$ and $\boldsymbol{b}_{\sigma(k)}^{p,gt}$ correspond to the 2D bounding box of $\boldsymbol{o}_k$ and $\boldsymbol{o}_{\sigma(k)}^{gt}$ in view p, respectively. Each 2D bounding box is parameterized with a vector of $(x_1, y_1, x_2, y_2) \in [0, 1]^4$, which contains the 2D coordinates of upper-left and bottom-right box corners relative to the image size.

By solving Eq. 2, we can assign a ground-truth object $\boldsymbol{o}_{\sigma(k)}^{gt}$ to each prediction $\boldsymbol{o}_k$, which facilitates our view loss calculation in Sec.3.4.

B.2. Frustum Loss

As a component in layout loss $\mathcal{L}_L$ (see Sec.3.4), the frustum loss is designed to optimize the 3D center $\boldsymbol{c}_k$ of a prediction $\boldsymbol{o}_k$ if it is not located inside a view frustum $q \in T_{\sigma(k)}$ from its matched ground-truth object $\boldsymbol{o}_{\sigma(k)}^{gt}$. An illustration of our frustum loss is shown in Fig. 2.

[1]Images in ScanNet and 3D-Front have resolution 1296×968 and 480×360 respectively.

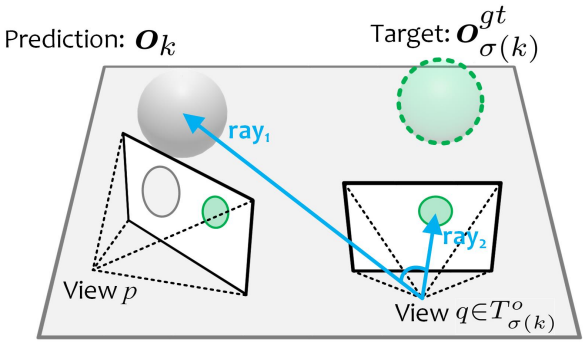

Figure 2. Illustration of frustum loss. p and q are two views where the ground-truth object $\boldsymbol{o}^{gt}_{\sigma(k)}$ is visible, while our prediction $\boldsymbol{o}_k$ is visible in view p only. We design a frustum loss, by maximizing the cosine similarity between ray$_1$ and ray$_2$, to optimize $\boldsymbol{o}_k$'s location and make it visible in view q as well.

As in our paper, we denote $T_{\sigma(k)}$ as all the visible views of the ground-truth object $\boldsymbol{o}^{gt}_{\sigma(k)}$. We calculate the average view loss between $\boldsymbol{o}_k$ and $\boldsymbol{o}^{gt}_{\sigma(k)}$ over all views in $T_{\sigma(k)}$, where the view loss includes object classification loss $\mathcal{L}_l$, 2D bounding box loss $\mathcal{L}_{box}$, completeness loss $\mathcal{L}_p$, frustum loss $\mathcal{L}_f$, and shape loss $\mathcal{L}_S$ (see Sec.3.4). However, when $\boldsymbol{o}_k$ is out of some view frustum $q \in T_{\sigma(k)}$, the calculation of $\mathcal{L}_{box}$ and $\mathcal{L}_S$ are meaningless, making convergence difficult. In this case, we design a frustum loss to force $\boldsymbol{o}_k$ to move towards the view frustum of q.

Assume that $\boldsymbol{o}_k$ is outside of $T^o_{\sigma(k)}$ views, where $T^o_{\sigma(k)} \subseteq T_{\sigma(k)}$. Then, we formulate the frustum loss $\mathcal{L}_f$ of $\boldsymbol{o}_k$ as

$$\mathcal{L}_f = \frac{1}{|T^o_{\sigma(k)}|} \sum_{q \in T^o_{\sigma(k)}} 1 - \text{Cosine}(\boldsymbol{c}_k - \boldsymbol{c}^q_{cam}, \boldsymbol{c}^{q,gt}_{\sigma(k)} - \boldsymbol{c}^q_{cam}), \tag{4}$$

where $|T^o_{\sigma(k)}|$ is the number of views in $T^o_{\sigma(k)}$; $\boldsymbol{c}_k$ is the 3D center of $\boldsymbol{o}_k$; $\boldsymbol{c}^q_{cam}$ is the 3D camera position at view q. $\boldsymbol{c}^{q,gt}_{\sigma(k)}$ is the center of $\boldsymbol{b}^{q,gt}_{\sigma(k)}$ in 3D space, where $\boldsymbol{b}^{q,gt}_{\sigma(k)}$ is the 2D bounding box of $\boldsymbol{o}^{gt}_{\sigma(k)}$ in view q. Therefore, $\boldsymbol{c}_k - \boldsymbol{c}^q_{cam}$ denotes a ray in world system from the camera center $\boldsymbol{c}^q_{cam}$ to the 3D object center $\boldsymbol{c}_k$, and $\boldsymbol{c}^{q,gt}_{\sigma(k)} - \boldsymbol{c}^q_{cam}$ is the ray from $\boldsymbol{c}^q_{cam}$ to the 2D ground-truth center on the image plane.

For an object $\boldsymbol{o}_k$ invisible to views $T^o_{\sigma(k)}$, we minimize the frustum loss while switching off the box loss $\mathcal{L}_{box}$ and shape loss $\mathcal{L}_S$ under those views. For visible views, we consider all view losses in Sec. 3.4 while switching off the frustum loss $\mathcal{L}_f$, because $\boldsymbol{o}_k$ in visible views have valid 2D bounding boxes and instance masks.

C. Additional Rendering Details for 3D-Front

We use BlenderProc [5] to sample cameras and render 2D images in 3D-Front scenes. Each scene in 3D-Front is an apartment which has several room types (bedroom, living room, library, etc.). In each scene, we uniformly sample at most 100 view points, with each view rendered into a 360×480 image with field of view of 90 degrees. The view number in each room is proportional to its floor area. The average object number captured in each view is 3.89 and 5.79, for bedroom and living room respectively, while the average object number contained in each room is 5.53 and 9.82 respectively. For each view, we export camera intrinsic and extrinsic parameters, instance masks, IDs, and category labels for our training.

D. Optimization Setup for Single View Reconstruction

In single-view scene reconstruction, we have an image with instance masks as the input. Our network and latent vectors are trained under multiple views. In this task, we freeze our pretrained network while only optimizing the latent vector for each single image, where instance masks are used for supervision. We use our view loss for this optimization but do not consider completeness loss in our final loss, because a single image is only a partial observation of an entire scene and we do not know how many objects this scene contains. We use RMSProp from PyTorch as the optimizer and train 1000 epochs with the initial learning rate at 0.01, which drops by 0.1x after 500 epochs.

After training, we input the optimized latent vector to our network and generate a set of objects. We output the first n objects for our qualitative and quantitative evaluation, where n is the object number in each input image.

E. Baselines

ATISS-2D We replace our transformer with the transformer encoder from ATISS [7] to generate object features, while keeping other settings unchanged. Note that the transformer encoder in ATISS is also permutation-invariant. We use the latent vector (sampled from the hypersphere surface) as the empty embedding in ATISS transformer to autoregressively generate a set of object features, which are input to our layout decoder and shape decoder for scene generation.

GAN We use our transformer as the generator to generate scenes and use the discriminator from [9] to distinguish the generated scene to real scenes. We apply a discriminator loss here to replace our view loss. The generator predicts a scene from a random vector sampled from our latent space. Then we render the semantic maps of this scene back to input views, and use the discriminator from [9] to classify if they are real or fake. Note that we do not consider the depth channel in the discriminator loss calculation, i.e., we only use the 2D semantic maps.

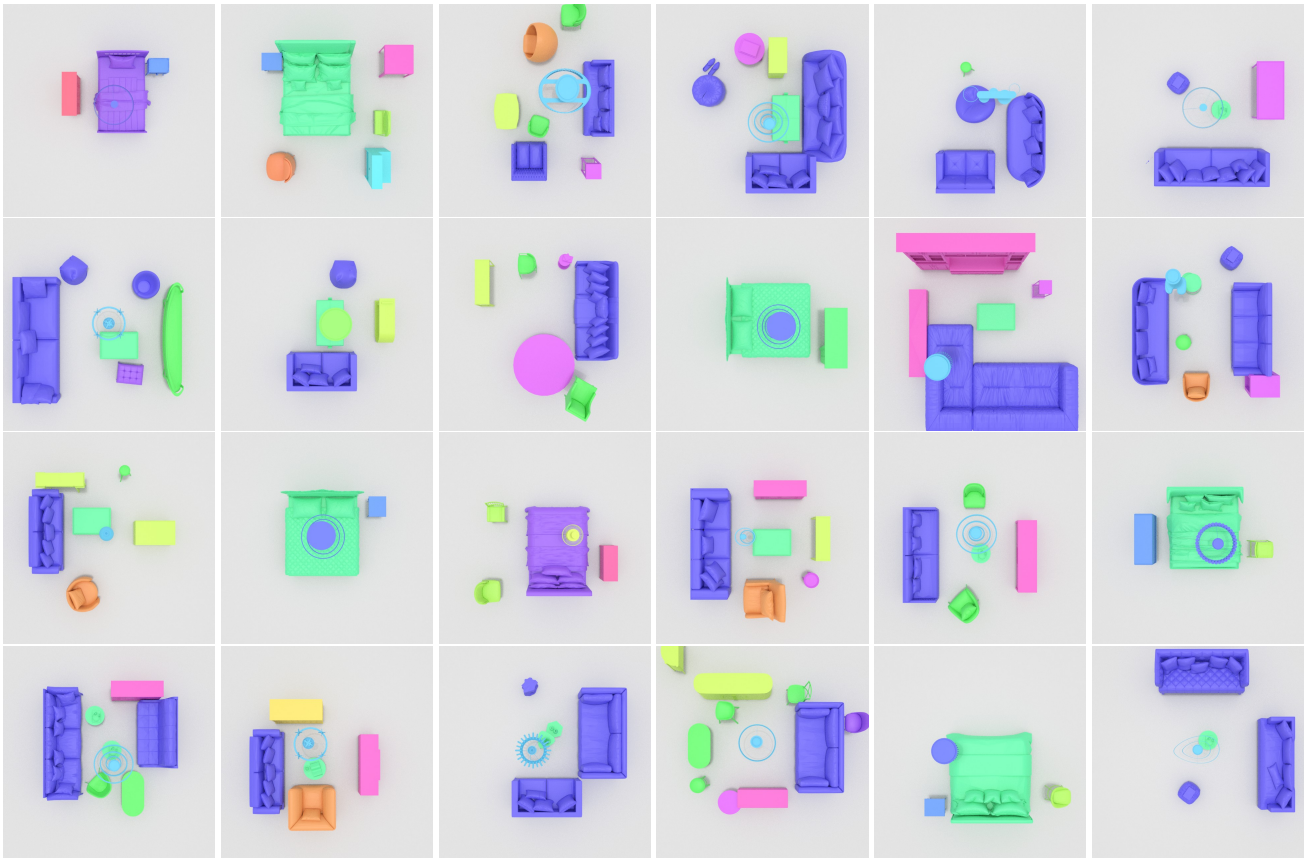

Figure 3. Additional qualitative results on 3D scene synthesis.

LSTM We replace our transformer with a traditional LSTM RNN [2] and keep other settings unchanged. It takes a latent vector z as input and recursively generates a sequence of features.

F. Additional Details of Shape Retrieval

In Sec. 5, we apply a shape retrieval post-processing to search for a CAD model for our reconstructed mesh. Since we already predict an object mesh, shape retrieval from this prediction is much easier. For each predicted object $\boldsymbol{o}$, we have its category label $\boldsymbol{l}$, 3D bounding box $\boldsymbol{b}$ and mesh $\boldsymbol{m}$. For those objects whose center height $\leq$ 1 meter, we extrude the bottom face of their 3D bounding box onto the floor. We search through the CAD models under the same category $\boldsymbol{l}$, and transform (move and scale) them to the 3D bounding box $\boldsymbol{b}$. All CAD models are augmented by rotating with 0, 90, 180, 270 degrees in the bounding box, then we calculate the Chamfer distance between the surface points from all augmented CAD models and the surface points from our mesh $\boldsymbol{m}$. The CAD model with the lowest Chamfer distance is retrieved. We repeat this process for all objects to model a 3D CAD scene.

	Bedroom			Living room		
	FID (↓)	SCA	KL (↓)	FID (↓)	SCA	KL (↓)
25%	29.48	0.96	0.06	105.33	0.98	0.35
50%	24.91	0.91	0.04	44.85	0.98	0.04
Full	**21.59**	**0.85**	**0.03**	**40.47**	**0.96**	**0.02**

Table 1. Ablation analysis on the number of views for training, evaluating scene generation.

G. Additional Ablation Experiments

In our training data, each scene contains at most 100 images to train our model (see Sec. C), where each scene has multiple rooms (bedroom, living room, etc.). The view number of each room is proportional to its floor area. More views sampled in a room indicate better coverage to capture more indoor objects.

In Tab. 1, we investigate the influence of different view numbers on our scene generation performance. We observe that better view coverage brings notable gains to all metrics, which is particularly significant for large-scale rooms (e.g., living rooms). This indicates that, without enough view coverage, our method cannot observe objects in different views, which would lead to ambiguities in object localization and deformation. However, more camera views means

Figure 4. Additional qualitative results on single-view scene reconstruction.

more training time, and extra human labor on data collection. In our experiments, we observe that using 100 views for both 3D-Front and ScanNet scenes presents enough scene coverage to learn general 3D scene priors.

H. Additional Qualitative Results

In Fig. 3 and Fig. 4, we list additional qualitative results on 3D scene synthesis and single-view reconstruction from the test set.

References

[1] Fast transformers. https://fast-transformers.github.io/. Accessed: 2022-11-12. 1

[2] Sequence models and long short-term memory networks. https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html. Accessed: 2022-11-14. 4

[3] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020. 2

[4] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5828–5839, 2017. 1

[5] Maximilian Denninger, Martin Sundermeyer, Dominik Winkelbauer, Youssef Zidan, Dmitry Olefir, Mohamad Elbadrawy, Ahsan Lodhi, and Harinandan Katam. Blenderproc. *arXiv preprint arXiv:1911.01911*, 2019. 3

[6] Huan Fu, Bowen Cai, Lin Gao, Ling-Xiao Zhang, Jiaming Wang, Cao Li, Qixun Zeng, Chengyue Sun, Rongfei Jia, Binqiang Zhao, et al. 3d-front: 3d furnished rooms with layouts and semantics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10933–10942, 2021. 1

[7] Despoina Paschalidou, Amlan Kar, Maria Shugrina, Karsten Kreis, Andreas Geiger, and Sanja Fidler. Atiss: Autoregressive transformers for indoor scene synthesis. *Advances in Neural Information Processing Systems*, 34:12013–12026, 2021. 3

[8] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv preprint arXiv:2007.08501*, 2020. 2

[9] Ming-Jia Yang, Yu-Xiao Guo, Bin Zhou, and Xin Tong. Indoor scene generation from a collection of semantic-segmented depth images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15203–15212, 2021. 3