Supplementary Material: Learning Occupancy for Monocular 3D Object Detection

In this supplementary material, we provide more details and experimental results including more ablations and qualitative results and associated discussions. We organize it as follows:

- Section A: detailed network architecture.
- Section B: more ablations and analysis.
  - Section B.1: latency.
  - Section B.2: occupancy learning as auxiliary tasks.
  - Section B.3: ablation on depth estimation.
  - Section B.4: ablation on voxel sizes for occupancy learning.
  - Section B.5: ablation on sub-networks for occupancy learning.
  - Section B.6: ablation on tricks.
- Section C: more qualitative results.
  - Section C.1: KITTI and WaymoOD results.
  - Section C.2: limitations and failure cases.

A. Detailed Network Architecture

In this section, we describe the detailed network architecture in our method.

2D backbone: Assume the input RGB image is $\mathbf{I} \in \mathbb{R}^{W_I \times H_I \times 3}$, where W_I, H_I are the input image resolution. First, we employ DLA34 [7] to extract features, achieving different stage features, which have different downsampling factors. Then we use its upsampling neck to obtain final backbone features $\mathbf{F_{img}} \in \mathbb{R}^{W_F \times H_F \times C_b}$, which are used for depth estimation, where W_F, H_F, C_b are feature resolution and channel, respectively. This backbone feature has a downsampling factor of 4, thus $W_F = \frac{W_I}{4}, H_F = \frac{H_I}{4}$. The feature channel C_b is 64. Following CaDDN [5], we use LID discretization method and 80 bins for depth estimetation. Thus we use a convolution layer $(3\times3, 1, 64, 81$, where 3×3 is the kernel size, 1 is the stride, 64 is the input feature channel and 81 is the output feature channel)

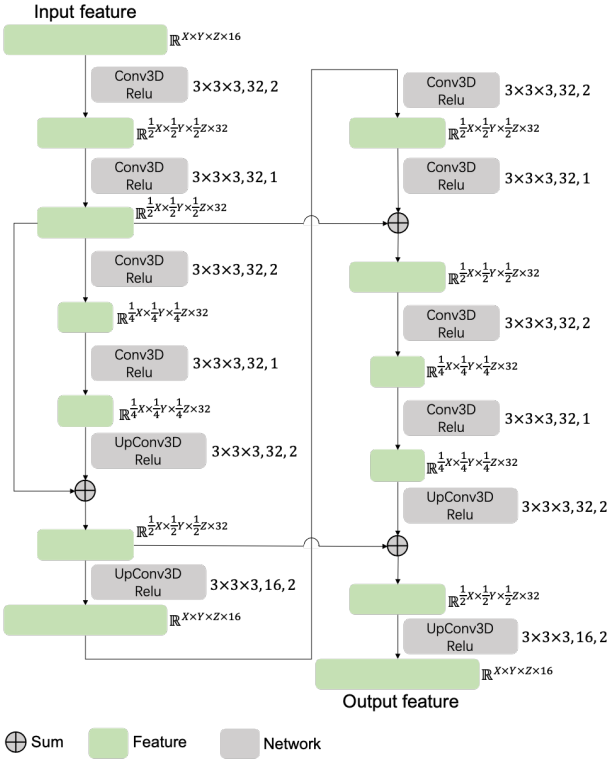

Figure 1. 3D hourglass like sub-network. $(3 \times 3 \times 3, 32, 2)$ in the figure denote the convolution kernel size, output channel, and stride, respectively. The "UpConv3D" in the figure denotes `torch.nn.ConvTranspose3d`.

for depth estimation. Please note that the last depth bin is responsible for predicting the out-of-range depth, thus the valid depth bin is 80. We obtain category depth distribution $\mathbf{D} \in \mathbb{R}^{W_F \times H_F \times D}$, where $D = 80$.

Frustum space: We generate the initial frustum feature using the same way in CaDDN. To save GPU memory, we use a convolution layer $(3 \times 3, 1, 64, 16)$ to reduce the backbone feature channel from 64 to 16. By enforcing outer product between category depth distribution and the reduced backbone feature, we have $\mathbf{Fru_1} \in \mathbb{R}^{W_F \times H_F \times D \times C}$, where C is the reduced feature channel and $W_F = \frac{W_I}{4}, H_F = \frac{H_I}{4}, D = 80, C = 16$. We use two 3D convolution layers $([3 \times 3 \times 3, 1, 16, 16] * 2, \text{ReLU})$

to obtain $\mathbf{Fru_2} \in \mathbb{R}^{W_F \times H_F \times D \times C}$. Then we employ a 3D convolution layer ($[3 \times 3 \times 3, 1, 16, 1]$, Sigmoid) to predict frustum occupancy $\mathbf{O_{fru}} \in \mathbb{R}^{W_F \times H_F \times D \times 1}$. Therefore, we have the enhanced frustum feature $\mathbf{Fru_3} \in \mathbb{R}^{W_F \times H_F \times D \times C}$ by combing frustum feature $\mathbf{Fru_2}$ and frustum occupancy $\mathbf{O_{fru}}$.

3D space: We employ grid sampling operation to transform frustum feature $\mathbf{Fru_3}$ into the regular 3D feature $\mathbf{V_1} \in \mathbb{R}^{X \times Y \times Z \times C}$, where X, Y, Z are the 3D voxel grid shape and are decided by pre-defined detection range and voxel size. Then we use a 3D hourglass like design to extract 3D features for 3D occupancy prediction. A 3D convolution ($[3 \times 3 \times 3, 1, 16, 16]$) is employed, followed by a 3D hourglass like sub-network, as shown in Figure 1. We have $\mathbf{V_2} \in \mathbb{R}^{X \times Y \times Z \times C}$ and then conduct 3D occupancy $\mathbf{O_{3d}} \in \mathbb{R}^{X \times Y \times Z \times 1}$ via a 3D convolution layer ($[3 \times 3 \times 3, 1, 16, 1]$, Sigmoid) . Finally, we can perform Hadamard product on 3D occupancy $\mathbf{O_{3d}}$ and 3D features $\mathbf{V_2}$ to obtain enhanced 3D feature $\mathbf{V_3} \in \mathbb{R}^{X \times Y \times Z \times C}$. We empirically find that the 3D hourglass like sub-network in frustum occupancy learning is unnecessary. We do not use 3D batch normalization in occupancy learning in both frustum and 3D space, as the features are not fully dense within the space. Please note that our goal is not to explore 3D network design, but to employ a typical network for occupancy learning.

Detector module: We follow the same detector module design in CaDDN. It consists of Voxel collapse (to BEV) and BEV 3D detection components. We refer the reader to [5, 2] for more details.

B. More Ablations and Analysis

B.1. Latency

We provide the overall latency of our method in Table 1, which is tested on a NVIDIA 3080Ti GPU. We can see that the occupancy learning modules cost most time due to the 3D convolutions. The 2D backbone and detector cost 32 ms while occupancy learning costs 80 ms. Compared to other real-time methods (*e.g.*, DEVIANT [1]:40ms, DID-M3D [4]:40ms), our method is slower. It is a disadvantage of the proposed method. On the other hand, our method is faster than many other methods (*e.g.*, DfM [6]:320ms, CaDDN [5]:630ms). It is worthy noting that the proposed method performs the best on the 3D detection accuracy among the above methods. This work currently does not focus on the runtime optimization, which leaves it to future works.

B.2. Occupancy Learning as Auxiliary Tasks

We also conduct experiments that make occupancy learning as auxiliary tasks. The results are reported in Table 2. For the auxiliary task setting, we still perform occupancy learning, but do not use the occupancy prediction to weight features. Occupancy learning also can make the network learn informative features under this setting, thus largely boosting the performance. When using the full setting, the method performs the best.

2D Backbone	OL-FS	OL-3DS	Detector	All
22ms	28ms	52ms	10ms	112ms

Table 1. Latency of each component. "OL-FS" in the table refers to occupancy learning in frustum space and "OL-3DS" in the table denotes occupancy learning in 3D space.

O-L	As auxiliary?	AP_{BEV}/AP_{3D} (IoU=0.7)$\|_{R_{40}}$ Easy	Moderate	Hard
	N/A	30.32/21.04	24.58/17.05	22.02/15.01
✓	Yes	35.69/26.31	26.47/19.57	23.37/16.93
✓	No	**35.72/26.87**	**26.60/19.96**	**23.68/17.15**

Table 2. Occupancy learning as auxiliary tasks. "O-L" in the table denotes occupancy learning.

B.3. Ablation on Depth Estimation

In the main text, we describe the relationship between occupancy and depth. We argue that occupancy bases on depth and beyond it. Good 2D dense depth estimates provide a good start point for occupancy because directly learning occupancy in large 3D space is difficult. We perform an experiment to prove this and report the results in Table 3. When removing depth estimation, the performance is heavily downgraded. Therefore, we believe that it is a good practice of combing depth estimation and occupancy learning.

Depth estimation	AP_{BEV}/AP_{3D} (IoU=0.7)$\|_{R_{40}}$ Easy	Moderate	Hard
	32.13/21.58	23.58/16.13	20.69/13.90
✓	**35.72/26.87**	**26.60/19.96**	**23.68/17.15**

Table 3. Ablation on depth estimation.

B.4. Ablation on Voxel Sizes for Occupancy Learning

As discussed in the main text, large voxel sizes reduce the GPU memory usage and computational overhead but introduce large quantization errors. We illustrate an example in Figure 2. Concerning irregular frustum voxels, their sizes are determined by feature resolutions and associated depth bin ranges, which usually follow conventional settings. Therefore, we mainly investigate the impact of regular 3D voxel size in 3D space for occupancy learning. We change the 3D voxel size in occupancy learning while keeping the voxel size the same for the later BEV detection module to make fair comparisons. As shown in Table 4, the performance decreases as voxel size increases, which is expected. Note that we can only use a minimum voxel size

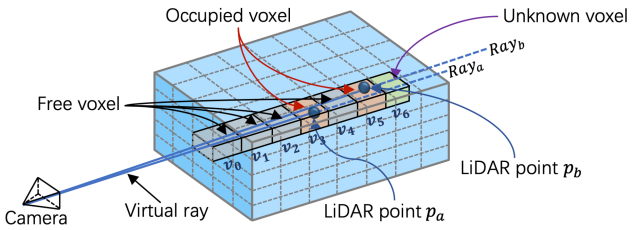

Figure 2. Quantization errors. Voxelized 3D occupancy from LiDAR points can lead to imperfect representations of the scene, as well as adverse effects when using ray-tracing to generate occupancy labels. For instance, given LiDAR points p_a and p_b, with p_a identifying free voxels v_0, v_1, v_2, occupied voxel v_3, and unknown voxels v_4, v_5, v_6. However, p_b regards occupied voxel v_3 as free voxel due to the intersection of Ray_b and v_3. This ambiguity is aggravated by larger voxel sizes. To address the ambiguity, we repeat the step of assigning occupied voxels to ensure that all voxels containing LiDAR points are marked as occupied.

of $0.16(meter)$ size because of the GPU (NVIDIA 3080Ti, 12G) memory limitation.

Voxel size	AP_{BEV}/AP_{3D} (IoU=0.7)$\|_{R_{40}}$ Easy	Moderate	Hard
0.64^3	32.68/23.09	24.14/17.07	21.74/14.98
0.32^3	35.45/26.01	26.12/18.77	23.14/16.26
0.16^3	**35.72/26.87**	**26.60/19.96**	**23.68/17.15**

Table 4. Ablation on regular 3D voxel sizes for occupancy learning. "0.64^3" in table denotes the voxel size of $0.64 \times 0.64 \times 0.64(meter)$.

B.5. Ablation on Sub-networks for Occupancy Learning

This section investigates the impact of different sub-networks used for occupancy learning. Specifically, we use two types of sub-network: 1: two 3D convolution layers; 2: 3D hourglass like architectures, where the latter is more complicated (Figure 1). We employ the two sub-networks in the occupancy learning module in frustum and 3D space, respectively. The results are shown in Table 5. We can see that more deeper and complicated networks benefit the performance. Interestingly, when employing the 3D hourglass like architecture in both frustum and 3D space, the results are not the best. Therefore, we empirically use two 3D convolution layers for frustum occupancy learning and the 3D hourglass like architecture for 3D occupancy learning. Given that this work does not aim to explore the network design, we encourage future works to introduce more delicate networks for this task.

B.6. Ablation on Tricks

We perform experiments to see the influence of tricks. More specifically, we adopt two tricks: pre-trained backbone on depth estimation from DDAD15M [3] and data augmentation. The results are provided in the Table 7. Interestingly, we find that the data augmentation is very important for this task. The performance drastically decreases when removing data augmentation. We think this gap is caused by the limited data scale in KITTI and the data diversity favors our method, and the results in WaymoOD support this viewpoint. Moreover, the pre-trained depth backbone further boosts the performance.

OL-FS	OL-3DS	AP_{BEV}/AP_{3D} (IoU=0.7)$\|_{R_{40}}$ Easy	Moderate	Hard
T-3D	T-3D	32.99/23.64	24.52/17.70	22.27/15.62
3D-Hour	T-3D	35.95/26.09	25.73/18.97	23.41/16.45
T-3D	3D-Hour	**35.72/26.87**	**26.60/19.96**	**23.68/17.15**
3D-Hour	3D-Hour	35.00/26.28	25.85/19.38	23.37/16.81

Table 5. Ablation on sub-networks for occupancy learning. "OL-FS" in the table denotes occupancy learning in frustum space; "OL-3DS" refers to occupancy learning in 3D space. "T-3D" refers to two 3D convolution layers; "3D-Hour" is the 3D hourglass like architecture.

λ	AP_{BEV}/AP_{3D} (IoU=0.7)$\|_{R_{40}}$ Easy	Moderate	Hard
0.1	34.31/25.52	25.85/19.10	23.39/16.95
0.5	34.34/26.42	26.17/19.40	23.30/17.46
1.0	35.72/26.87	26.60/19.96	23.68/17.15
5	35.95/27.57	26.39/19.63	23.34/16.96
10	35.64/27.51	26.20/19.64	23.09/16.95

Table 6. Ablation on λ. λ (in Equation 10 in the main text) is the occupancy loss weighting factor in the final loss.

Pre-trained b.	Data aug.	AP_{BEV}/AP_{3D} (IoU=0.7)$\|_{R_{40}}$ Easy	Moderate	Hard
		24.81/16.83	19.16/13.42	17.77/11.86
✓		28.01/19.72	21.20/14.86	18.83/12.72
	✓	35.20/26.35	25.97/19.17	23.14/16.57
✓	✓	**35.72/26.87**	**26.60/19.96**	**23.68/17.15**

Table 7. Ablation on tricks. "Pre-trained b." in the table denotes pre-trained backbone on depth estimation from DDAD15M [3]; "Data aug." is the data augmentation.

C. More Qualitative Results

We visualize occupancy predictions and 3D detections to better understanding the effectiveness and limitation of our method.

C.1. KITTI and Waymo Results

We provide some qualitative results on KITTI *val* set in Figure 3 and results on WaymoOD (waymo open dataset) *val* set in Figure 4. Our method generates good results for most cases. Considering faraway and some occluded objects, the proposed method still provides dense and reasonable 3D occupancy. This characteristic benefits the downstream monocular 3D detection task.

C.2. Limitations and Failure Cases

Our method also has some limitations and can fail to deal with some objects. First, for heavily occluded objects, the method cannot generate discriminative occupancy for such objects (please see orange arrows in Figure 3 and Figure 4). Second, our method is hard to precisely predict the occupancy behind thin objects (*e.g.*, pole, please see cyan arrows in Figure 4) under the camera perspective. It is because no LiDAR points are provided in these areas, and the relative background LiDAR points tend to make the network regard such invisible regions as occupied. Therefore, we can see tails behind thin objects (please see cyan arrows in Figure 4).

References

[1] Abhinav Kumar, Garrick Brazil, Enrique Corona, Armin Parchami, and Xiaoming Liu. Deviant: Depth equivariant network for monocular 3d object detection. In *European Conference on Computer Vision (ECCV)*, 2022. 2

[2] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. PointPillars: Fast encoders for object detection from point clouds. In *CVPR*, pages 12697–12705, 2019. 2

[3] Dennis Park, Rares Ambrus, Vitor Guizilini, Jie Li, and Adrien Gaidon. Is pseudo-lidar needed for monocular 3d object detection? In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3142–3152, 2021. 3

[4] Liang Peng, Xiaopei Wu, Zheng Yang, Haifeng Liu, and Deng Cai. Did-m3d: Decoupling instance depth for monocular 3d object detection. In *European Conference on Computer Vision*, 2022. 2

[5] Cody Reading, Ali Harakeh, Julia Chae, and Steven L Waslander. Categorical depth distribution network for monocular 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8555–8564, 2021. 1, 2

[6] Tai Wang, Jiangmiao Pang, and Dahua Lin. Monocular 3d object detection with depth from motion. *arXiv preprint arXiv:2207.12988*, 2022. 2

[7] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2403–2412, 2018. 1

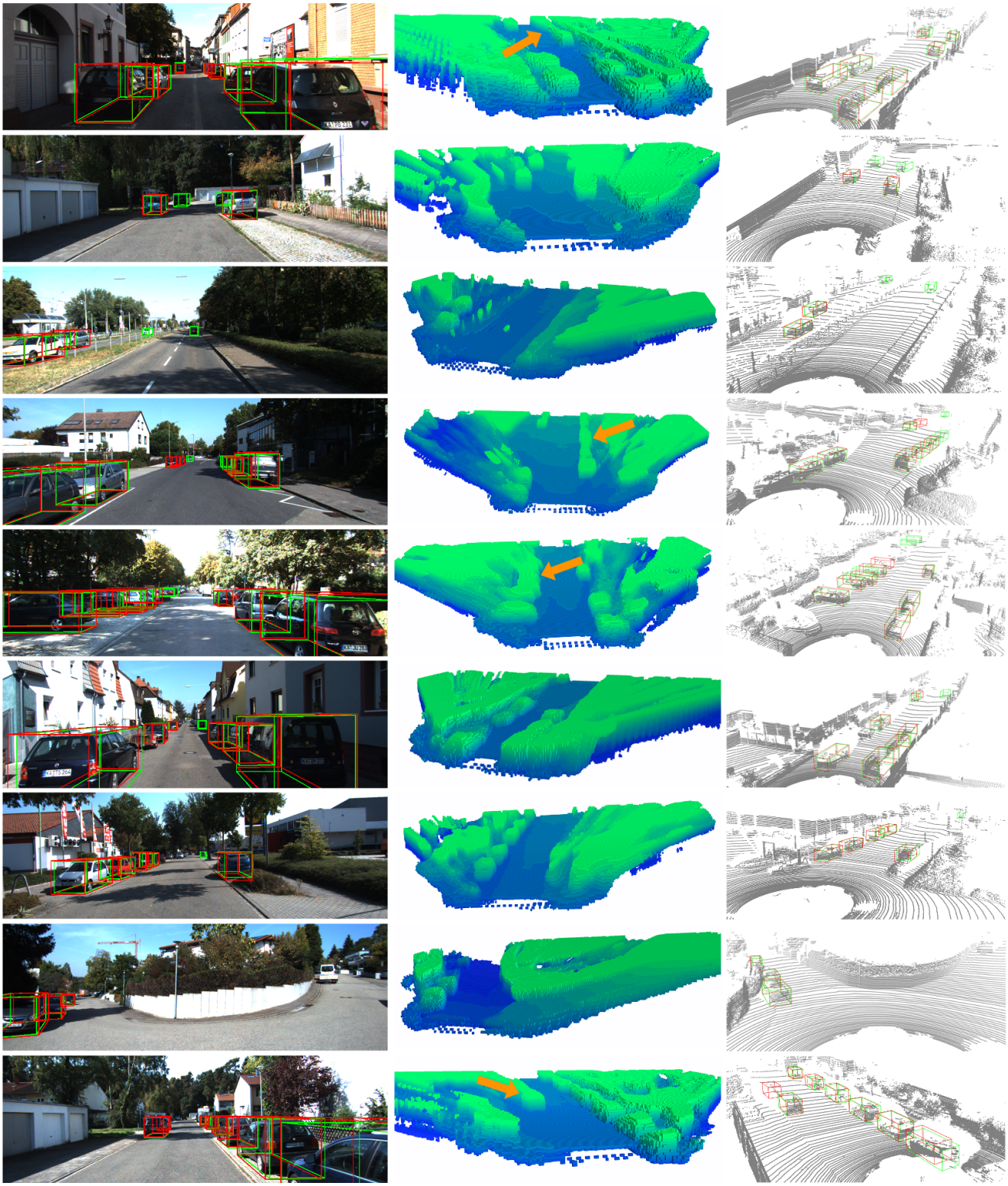

Figure 3. KITTI *val* results. From left to right: ***input RGB images; occupancy predictions; 3D box predictions***. Red boxes are our results and Green boxes denote ground-truths. The LiDAR point clouds in 3D detections are used only for visualization. Orange arrows refer to failure cases caused by heavy occlusions. Best viewed in color with zoom in.

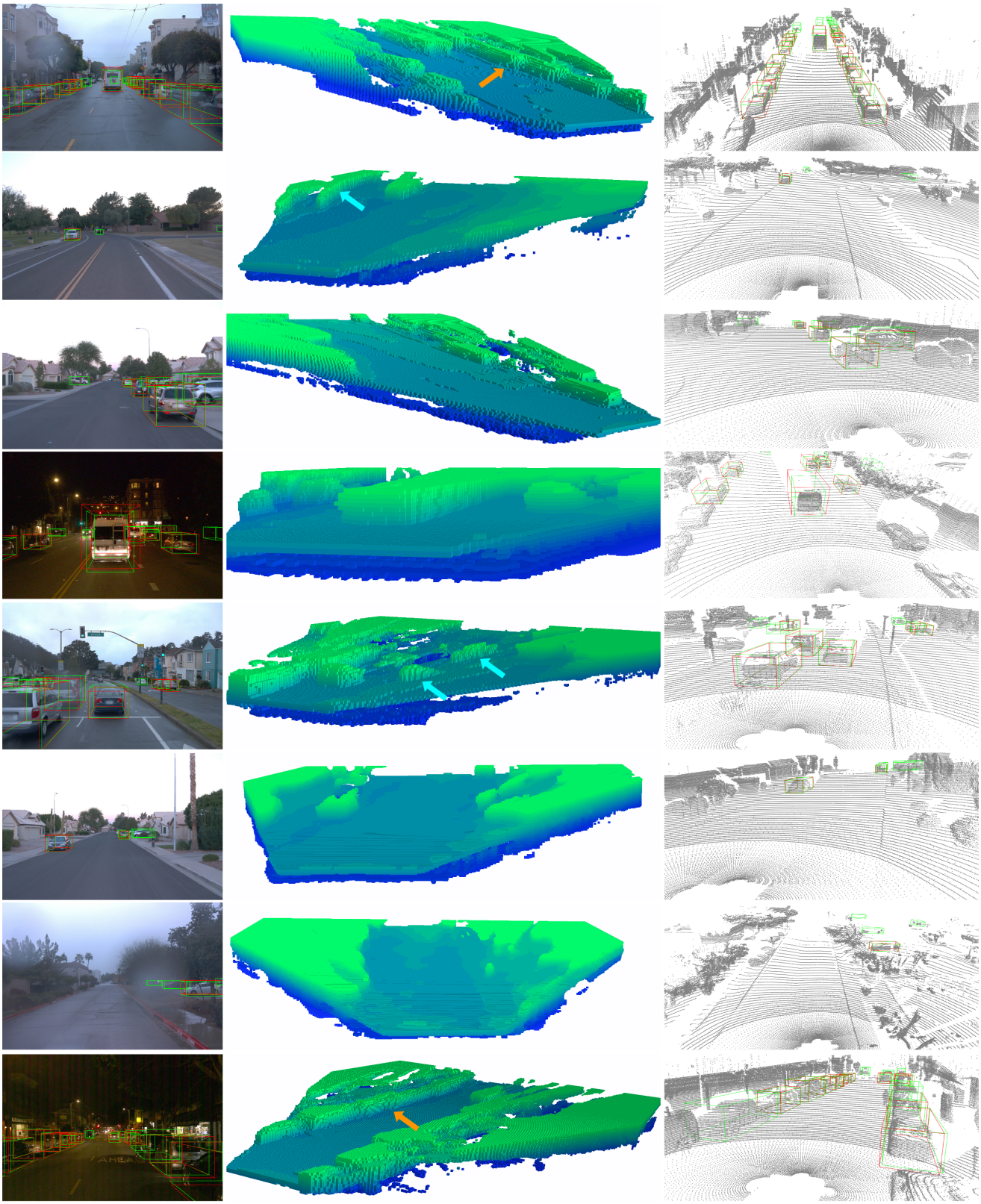

Figure 4. WaymoOD *val* results. From left to right: ***input RGB images; occupancy predictions; 3D box predictions***. Red boxes are our results and Green boxes denote ground-truths. The LiDAR point clouds in 3D detections are used only for visualization. Orange arrows refer to failure cases caused by heavy occlusions and cyan arrows denote failure cases caused by thin objects. Best viewed in color.