Roadside Monocular 3D Detection Prompted by 2D Detection

Supplementary Material

Outline

This document supplements the main paper with further details of Pro3D and more visualizations. Below is the outline of this document.

- **Section A.** We provide implementation details of the prompt encoder in Pro3D.
- **Section B.** We discuss our loss design in Pro3D.
- **Section C.** We evaluate models' performance under occlusion and truncation scenarios.
- **Section D.** We study the robustness of Pro3D to camera noises.
- **Section E.** We report 2D metrics by roadside monocular 3D detectors.
- **Section F.** We conduct cross-dataset evaluation.
- **Section G.** We compare the inference speed of Pro3D using different 3D detectors.
- **Section H.** We provide our code.
- **Section I.** We provide more visual results, including demo videos, and a Jupyter Notebook of code.

A. Details of Prompt Encoder

The prompt encoder transforms the 2D detection outputs to features through three steps: (1) normalize the top-left and bottom-right coordinates of a 2D detection box, denoted as $A \in \mathbb{R}^{2\times2}$; (2) multiply A by a random matrix $B \in \mathbb{R}^{2\times512}$ whose entries are initialized using a Gaussian distribution, and then add a learnable matrix $C \in \mathbb{R}^{2\times512}$ to get $D = A \times B + C$; (3) repeat the predicted class label (represented as a unique class ID) 512 times to get a 512-dimensional vector, and concatenate it with D to obtain the final feature prompts $E \in \mathbb{R}^{3\times512}$.

B. Loss Design

Class grouping is a common strategy in 3D object detector training. It merges classes into some superclasses, allowing trained features to be shared within superclasses [69]. Moreover, multi-head design follows along with groups of classes for discriminative predictions across fine-grained classes [64]. BEVHeight [60] and BEVSpread [51] adopt class grouping and multi-head design by considering the similarities among object appearance, e.g., size and shape in the 3D world. For example, on DAIR-V2X-I, they create three superclasses that merge {*truck*, *bus*}, {*car*, *van*}, and {*bicyclist*, *tricyclist*, *motorcyclist*, *barrowlist*}, respectively, along with the origin *pedestrian* class. The rationale of such grouping is based on object appearance, e.g., both *truck* and *bus* have large size. We call this grouping strategy *appearance-based grouping*. With such a group strategy, they learn a four-head detector with each making predictions for the corresponding superclass. Differently, we adopt *functionality-based grouping*, which merges {*car*, *van*, *truck*, *bus*} into *vehicle*, and {*bicyclist*, *tricyclist*, *motorcyclist*, *barrowlist*} into *cyclist*. This is motivated by the fact that vehicles, cyclists and pedestrians appear at relatively fixed regions on the roadside images. We expect this grouping strategy, or using the location prior of different classes, to facilitate training 3D object detectors on roadside images. Fig. 8 compares the two grouping methods. Perhaps from the same functionality perspective, the DAIR-V2X-I benchmark also merges classes into such superclasses for evaluation. For fair comparison, we use our functionality-based grouping loss to train the 3D detector (e.g., BEVHeight or BEVSpread) within our Pro3D framework to compare with the original 3D detectors. Our extensive study shows that trainig with the functionality-based grouping loss remarkably improves detection performance (Table 11).

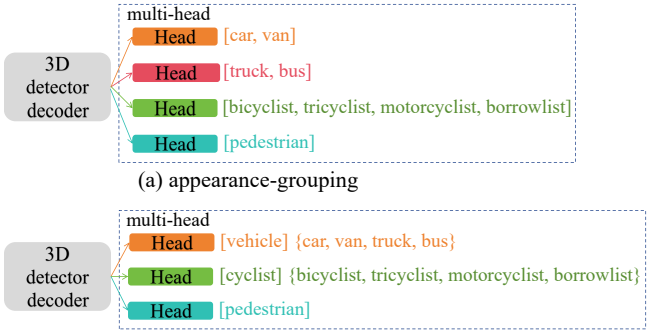

Figure 8. Comparison of two grouping methods in the loss design: (a) appearance-based grouping, and (b) functionality-based grouping. The former is commonly adopted in existing roadside 3D detectors that group classes according to their appearance similarity and use multi-head for prediction (each head is responsible for K-way fine-grained prediction. The latter is our proposed method that groups classes according to their functionalities and uses multi-head for prediction – each head is a single super-class predictor, which merges {*car*, *van*, *truck*, *bus*} into *vehicle*, and {*bicyclist*, *tricyclist*, *motorcyclist*, *barrowlist*} into *cyclist*.

C. Analysis of Occlusion and Truncation

Objects might be occluded by others or truncated around the image borders. We carry out breakdown analysis how detectors perform in these scenarios. The DAIR-V2X-I

Table 10. **Results of occluded and truncated obstacles**. We split occlusions and truncations with different ranges (0%∼50% and 50%∼100%), respectively. Our approach Pro3D resoundingly outperforms BEVHeight [60] and BEVSpread [51] on all the three superclasses in each range.

(a) Occlusion of 0%∼50%

Methods	Vehicle	Cyclist	Pedestrian
BEVHeight	75.6	66.8	48.9
Pro3D w/ BEVHeight	80.8	76.6	58.0
BEVSpread	77.5	67.7	51.2
Pro3D w/ BEVSpread	**83.2**	**76.8**	**60.1**

(b) Occlusion of 50%∼100%

Methods	Vehicle	Cyclist	Pedestrian
BEVHeight	51.2	36.9	19.2
Pro3D w/ BEVHeight	58.0	42.3	28.1
BEVSpread	53.3	38.6	19.9
Pro3D w/ BEVSpread	**60.1**	**44.6**	**28.5**

(c) Truncation of 0%∼50%

Methods	Vehicle	Cyclist	Pedestrian
BEVHeight	76.2	68.0	50.3
Pro3D w/ BEVHeight	83.9	76.8	60.0
BEVSpread	78.8	69.3	52.6
Pro3D w/ BEVSpread	**85.0**	**77.9**	**61.6**

(d) Truncation of 50%∼100%

Methods	Vehicle	Cyclist	Pedestrian
BEVHeight	38.7	18.6	10.8
Pro3D w/ BEVHeight	45.5	27.3	20.5
BEVSpread	38.9	19.1	11.2
Pro3D w/ BEVSpread	**45.6**	**27.6**	**20.6**

Table 11. **Ablation on class-grouping strategies** (w.r.t metric AP in BEV). Grouping classes differently with multi-head design (each detector head makes predictions within a specific superclass) makes a difference in training and yields detectors performing differently. Briefly, instead of appearance-based grouping (AG) which groups classes w.r.t object appearance as done by BEVHeight, we do functionality-based grouping (FG), e.g., merging {*car*, *van*, *truck*, *bus*} into the superclass *vehicle*. To justify the superiority of FG over AG, we replace the original AG of BEVHeight [60] and BEVSpread [51] with FG, particularly improving on the vehicle class (cf. "BEVHeight w/ FG" vs. "BEVHeight w/ AG", "BEVSpread w/ FG" vs. "BEVSpread w/ AG").

Method	Vehicle			Cyclist			Pedestrian		
	Easy	Mid	Hard	Easy	Mid	Hard	Easy	Mid	Hard
BEVHeight w/ AG	77.8	65.8	65.9	60.2	60.1	60.5	41.2	39.3	39.5
BEVHeight w/ FG	80.0	67.9	68.1	60.5	60.3	60.7	41.4	39.4	39.7
performance gain	+2.2	+2.1	+2.2	+0.3	+0.2	+0.2	+0.2	+0.1	+0.2
BEVSpread w/ AG	79.1	66.8	66.9	62.6	63.5	63.8	46.5	44.5	44.7
BEVSpread w/ FG	81.1	68.8	69.0	62.7	63.7	63.9	46.6	44.7	44.8
performance gain	+2.0	+2.0	+2.1	+0.1	+0.2	+0.1	+0.1	+0.2	+0.1

dataset provides occlusion and truncation tags. We split occlusions and truncations with 0%∼50% and 50%∼100% for different superclasses on [65], respectively. As shown in Table 10, our method gets better APs on the all superclasses in different ranges of occlusions and truncations. This shows that our method Pro3D is more robust than BEVHeight [60] and BEVSpread [51].

D. Robustness to Noises of Camera Pose

We consider the robustness of Pro3D to noises in camera calibration parameters. We follow BEVHeight [60] to add noises on camera pitch and roll. Table 12 compares the robustness of BEVHeight [60], BEVSpread [51] and our Pro3D. Clearly, Pro3D is more robust than prior works to camera noises.

Table 12. **Robustness to camera noise**. We follow BEVHeight [60] to simulate calibration errors by adding noises to camera pitch and roll. Results show that Pro3D is more robust than BEVHeight and BEVSPread [51] to these camera pose noises.

Method	Vehicle			Cyclist			Pedestrian		
	Easy	Mid	Hard	Easy	Mid	Hard	Easy	Mid	Hard
BEVHeight	77.8	65.8	65.9	60.2	60.1	60.5	41.2	39.3	39.5
+ noise	62.5	49.9	50.7	45.4	43.4	43.8	34.5	32.6	32.7
performance degradation	-15.3	-15.9	-15.2	-14.8	-16.7	-16.7	-6.7	-6.7	-6.8
Pro3D w/ BEVHeight	83.7	71.8	72.0	71.3	71.2	71.4	53.9	52.9	52.6
+ noise	72.5	59.9	60.3	59.0	60.0	61.9	47.9	46.8	46.5
performance degradation	-11.2	-11.9	-11.7	-12.3	-11.2	-9.5	-6.0	-6.1	-6.1
BEVSpread	79.1	66.8	66.9	62.6	63.5	63.8	46.5	44.5	44.7
+ noise	64.2	51.6	51.4	48.4	48.8	49.4	40.3	37.6	38.6
performance degradation	-14.9	-15.2	-15.5	-14.2	-14.7	-14.4	-6.2	-6.9	-6.1
Pro3D w/ BEVSpread	85.2	73.1	73.6	73.1	73.0	73.1	55.1	54.3	54.1
+ noise	73.1	60.3	61.7	60.4	61.8	63.1	48.9	48.0	47.8
performance degradation	-12.1	-12.8	-11.9	-12.7	-11.2	-10.0	-6.2	-6.3	-6.3

Table 13. **Comparison of inference time (*ms*) between different methods.** BEVHeight [60] has the fastest inference time comparing with other three methods. Here, we build our Pro3D using it as the 3D detector. It is worth noting that we adopt the functionality-based grouping loss, which induces less parameters compared to the wide-used appearance-based grouping loss (Section B). This makes our Pro3D obtain the fastest inference speed among these methods.

Method	BEVDepth	BEVHeight	BEVHeight++	BEVSpread	Pro3D
time(ms)	82	77	92	90	**65**

E. 2D Detection Performance of 3D Detectors

Table 14 reports 2D metrics of both the 2D detector DINO and roadside 3D detection methods including BEVDepth, BEVHeight, BEVSpread, and their improved versions by our Pro3D. Results demonstrate that (1) existing 3D detectors (BEVDepth, BEVHeight, and BEVSpread)

Table 14. **Comparison w.r.t 2D detection metric (mAP) on DAIR-V2X-I.** For both 2D detector (DINO) and 3D detectors BEVDepth and BEVHeight, we train them on the training set of DARI-V2X-I. To evaluate 3D detectors w.r.t 2D metrics, we project their 3D detections onto the 2D image plane to derive their predicted 2D boxes. Compared to BEVDepth and BEVHeight, DINO achieves >10 mAP higher on all classes and ∼20 mAP higher on Pedestrian! Owing to the effective leverage of the 2D detector DINO, our Pro3D significantly boosts performance of the compared methods.

Method	Vehicle			Cyclist			Pedestrian		
	Easy	Mid	Hard	Easy	Mid	Hard	Easy	Mid	Hard
2D detector (DINO) [66]	74.3	75.6	75.6	57.3	54.7	54.9	50.6	51.6	51.6
BEVDepth [18]	62.9	64.8	64.2	41.5	44.8	44.9	32.9	33.1	33.0
Pro3D w/ BEVDepth	68.0	68.3	68.9	45.9	47.7	47.6	42.1	42.6	42.6
BEVHeight [60]	63.5	64.7	64.5	41.4	44.6	44.7	33.6	33.7	33.9
Pro3D w/ BEVHeight	68.1	68.8	68.8	47.9	49.5	49.4	42.8	43.5	43.5
BEVSpread [51]	64.1	65.2	64.6	41.7	45.1	44.9	33.9	34.1	34.3
Pro3D w/ BEVSpread	68.7	69.4	69.2	48.7	50.3	50.3	44.1	44.9	44.9

produce significantly lower 2D metrics than the 2D detector DINO; (2) by exploiting DINO's 2D detections as prompt, our Pro3D greatly enhances 3D detection (see results in Table 1 and 2 in the main paper).

F. Cross-Dataset Evaluation

One may wonder the results of cross-dataset evaluation. The datasets (DAIR-V2X-I and Rope3D) used in our work do not have the same vocabulary, but both have the `vehicle` class. Therefore, focusing on the `vehicle` class, we use the models trained on Rope3D and test on the DAIR-V2X-I benchmark. In this study, we compare our Pro3D that uses the DINO 2D-detector. The table below compares BEVHeight, BEVSpread, and Pro3D. Performance of all methods degrade (marked in parentheses) but our Pro3D yields less degradation compared with the other methods.

Method on `vehicle`	`Easy`	`Mid`	`Hard`
BEVHeight [60]	62.6 (-15.2)	50.1 (-15.7)	50.3 (-15.6)
Pro3D w/ BEVHeight	70.0 (-10.7)	60.7 (-11.1)	61.1 (-10.9)
BEVSpread [51]	65.9 (-13.2)	53.0 (-13.8)	53.3 (-13.6)
Pro3D w/ BEVSpread	77.6 (-7.6)	65.1 (-8.0)	65.3 (-8.3)

G. Comparison on the Inference Speed

We compare the inference time of different methods in Table 13. BEVHeight [60] gets the fastest inference time in previous work, so we use it in our ablation studies. Moreover, our method Pro3D outperforms other works in inference speed, where we use YOLOV7 as the 2D detector in Pro3D.

H. Open-Source Code

Code. We include our self-contained codebase (refer to the zip file `Pro3D-main`) as a part of the supplementary material. Please refer to `README.md` for instructions how to use the code. We do not include model weights in the supplementary material as they are too larger than the space limit (100MB). We will open source our code and release our trained models to foster research.

License. We release open-source code under the MIT License to foster future research in this field.

Requirement. Running our code requires some common packages. We installed Python and most packages through Anaconda. A few other packages might not be installed automatically, such as Pandas, torchvision, and PyTorch, which are required to run our code. Below are the versions of Python and PyTorch used in our work:

- Python version: 3.8.0 [GCC 7.5.0]
- PyTorch version: 1.8.1

I. More Visualizations, Demo Videos, and Jupyter Notebook of Code

Figure 9 visualizes more results on the DAIR-V2X-I dataset [65]. Pro3D not only has better orientation predictions for vehicles but also can detect hard objects such as infrequently-seen ambulances, occluded pedestrians and cyclists.

Demo Videos. We include two demo videos as a part of the supplementary material, named `demo-video-vs-BEVHeight.mp4` and `demo-video-vs-BEVSpread.mp4`. These videos compare our Pro3D with BEVHeight [60] and BEVSpread [51], respectively. They clearly show that our Pro3D performs much better than both methods in roadside monocular 3D detection.

Jupyter Notebook of Code. In the supplement, we provide not only our code but also a Jupyter Notebook file named `demo-pro3d-infer-vis.ipynb` which can readily run our Pro3D and display its results on video frame examples. The reader is referred to this file for a quick visualization and guideline on how to run our code.

Figure 9. More visualizations between the state-of-the-art method BEVHeight [60] and our Pro3D.