Frustum-PointPillars: A Multi-Stage Approach for 3D Object Detection using RGB Camera and LiDAR

Anshul Paigwar, David Sierra-Gonzalez, Özgür Erkent, Christian Laugier

***Abstract*— Accurate 3D object detection is a key part of the perception module for autonomous vehicles. A better understanding of the objects in 3D facilitates better decision-making and path planning. RGB Cameras and LiDAR are the most commonly used sensors in autonomous vehicles for environment perception. Many approaches have shown promising results for 2D detection with RGB Images, but efficiently localizing small objects like pedestrians in the 3D point cloud of large scenes has remained a challenging area of research. We propose a novel method, Frustum-PointPillars, for 3D object detection using LiDAR data. Instead of solely relying on point cloud features, we leverage the mature field of 2D object detection to reduce the search space in the 3D space. Then, we use the Pillar Feature Encoding network for object localization in the reduced point cloud. We also propose a novel approach for masking point clouds to further improve the localization of objects. We train our network on the KITTI dataset and perform experiments to show the effectiveness of our network. On the KITTI test set our method outperforms other multi-sensor SOTA approaches for 3D pedestrian localization (Bird's Eye View) while achieving a significantly faster runtime of 14 Hz.**

I. Introduction

Fully autonomous driving is an important but challenging goal, for which a reliable perception of the local environment is critical [1]. RGB cameras and 3D-LiDARs are widely used sensors in robotics and autonomous vehicles, providing complementary information about the environment. Images from RGB cameras provide feature-rich and dense information of the environment and thus facilitate many perception tasks like 2D detection, semantic segmentation, and action recognition. However, beyond 2D perception, 3D understanding of the environment is vital for many applications such as path planning and decision-making for autonomous driving. LiDARs are commonly used to capture 3D data but, unlike images, the point cloud generated from LiDARs is sparse and unstructured. Designing network architectures for point cloud processing is an ongoing challenge that differs from the techniques used for 2D object detection in RGB images [2].

Early approaches for 3D object detection converted point clouds either into 2D images by view projection [2], or structured volumes of voxel grids through quantization [3]. Then, established techniques for 2D object detection could be directly applied to the converted images/volumes. A common drawback of these methods is that they suffer from the loss of crucial 3D information in the view projection or quantization process.

[1] Univ. Grenoble Alpes, Inria, 38000, Grenoble, France; e-mail: firstname.lastname@inria.fr

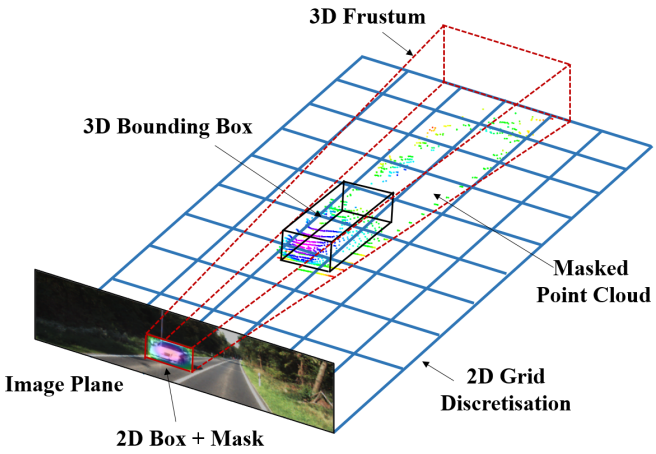

Fig. 1: **Overview of Frustum-PointPillars architecture:** 2D detections of objects are extruded to 3D bounding frustum. Points inside the frustum are masked based on the likelihood of belonging to the object. The 3D frustum area is then divided into a 2D grid, and PointNet is used within each cell of the grid to extract pillar features. Finally, a 3D bounding box is predicted for the object from the points in the 3D frustum.

Significant progress has been made in learning features directly from raw point clouds without converting them into any other forms of representation [4], [5]. The recently proposed PointNet directly works with 3D points and has shown superior performance and efficiency in several 3D understanding tasks such as object classification and semantic segmentation. While PointNet is capable of extracting features from a point cloud with a small number of points, its scalability to larger point clouds is unclear. Researchers have been exploring different pipelines to use PointNet in their deep architecture for instance-level 3D object detection.

A plethora of LiDAR-only approaches subdivide the large 3D space into smaller subspaces or voxels and then use PointNets to extract voxel-wise features. These voxel-wise features are used with different settings and pipelines for 3D object detection [6], [7], [8], [9], [10].

Although LiDAR-only approaches have shown good performance in the detection of large objects like cars and vans, they suffer at localizing smaller objects like pedestrians in 3D point clouds. Unlike cars, pedestrians are not rigid bodies; the point perturbations of a pedestrian do not have a distinctive geometric structure and have fewer data points, making them difficult to distinguish from other objects in the environment.

To deal with these challenges, leveraging multi-sensor data has emerged as a promising option. As RGB cameras are commonly used in autonomous vehicles, many approaches are exploring different ways to leverage the rich features from RGB images to improve LiDAR-based 3D detection.

In this work, we also follow a multi-sensor approach that uses RGB and LiDAR for 3D object detection. We present Frustum-PointPillars (F-PointPillars), a deep architecture for 3D object detection in point clouds. F-PointPillars leverages 2D detections in RGB images, to reduce search space in 3D. 2D detections can be obtained from any of the available state-of-the-art object detectors. The 2D detections of the objects are extruded into 3D space creating a 3D bounding frustum of the object as shown in Fig. 1. We discretize the 3D frustums into a bird-eye-view (BEV) 2D grid to extract features at fine resolution. This enables us to accurately localize smaller objects within the 3D frustum.

The key contributions of this paper can be summarized as follows:

- We present a new method called Frustum-PointPillars for real-time 3D object detection in point clouds. We extend and improve PointPillars architecture by the addition of a new sensor modality of RGB camera and the use of a multi-stage approach. We also extend the data augmentation for training to work with multi-stage network architecture.
- We propose Gaussian-based masking of 3D points to distinguish foreground from background clutter, thus improving localization of objects in 3D. Experiments on the KITTI dataset and quantitative evaluations validate our design choices.
- F-PointPillars outperforms PointPillars and other multi-stage SOTA approaches for localization of pedestrians (BEV detection) on the KITTI benchmark. Our approach significantly outperforms other multi-stage SOTA approaches in terms of runtime.

II. Related Work

In this section, we first categorize methods related to using multiple sensors for 3D object detection into three categories: 1) Early fusion 2) Late fusion 3) Multi-stage methods. Next, we discuss the pros and cons of each category of methods. Finally, we focus in detail on the methods using a multi-stage network architecture for 3D object detection in point clouds.

A. Early fusion methods:

A single network takes input from two or more sensor modalities. The data from different sensors is converted into a common feature space or dimension. The features are fused either using concatenation, projection, or by a small network. The features are fused early in the network pipeline. Early sensor fusion can lead to an increase in the accuracy of the detections [11], [12]; however, they suffer from the fact that a failure of one sensor can lead to the total failure of the network.

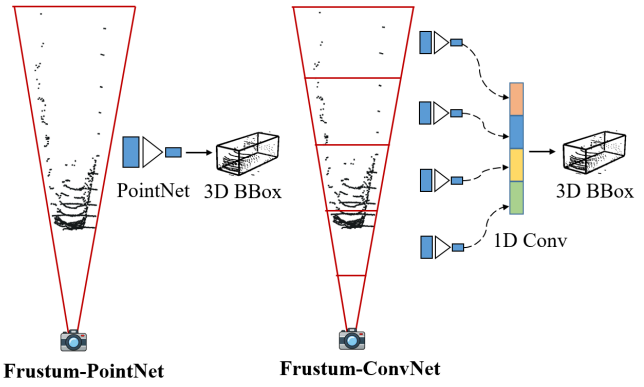

Fig. 2: The figures illustrate simplified pipelines of Multi-Stage approaches for 3D object detection. Left: F-PointNet uses all points within 3D bounding frustum for feature extraction. Right: F-ConvNet sub-divided 3D bounding frustum into smaller frustum and extract multiple features.

B. Late fusion methods:

Independent networks, one for each sensor modality, work in parallel to extract features and output detections in a redundant manner. The detections from the independent networks are then fused using statistical methods or a smaller network. The merit of late fusion approaches is that in case of the failure of one sensor, the detections from another sensor can still be used, bearing a reduction in the accuracy [13]. Nevertheless, sometimes it can be difficult to perform a certain task efficiently in a redundant way due to the limitations of the sensor modalities, for example, 3D object detection using a monocular camera.

C. Multi-stage methods:

In multi-stage methods, two independent networks, one for each modality, are stacked together. The output of the first network (Stage-I) constitutes the input to the second network (Stage-II). The key idea is to use different sensor modalities according to their strength to perform a particular task, such as RGB cameras for 2D detections and LiDAR for 3D localization. Methods using the Multi-Stage approach for sensor fusion have shown high accuracy for pedestrian and cyclist detection in 3D pointclouds [14], [15]. Following this, we choose to use the multi-stage method of sensor fusion to develop our 3D object detection system.

Frustum PointNet [14] and Frustum ConvNet [15] are two methods that use the multi-stage design approach. Given 2D region proposals in RGB images, these methods first find local points corresponding to pixels inside the 2D regions. F-PointNet then uses the PointNet architecture to segment these local points into foreground and background. An amodal 3D box is then estimated using the foreground points as shown in Fig. 2. F-PointNet not an end-to-end learning method and the estimation of 3D bounding boxes relies on fewer foreground points which could possibly be segmented inaccurately.

Different from F-PointNet, F-ConvNet subdivides each region proposal into a sequence of smaller frustums as shown

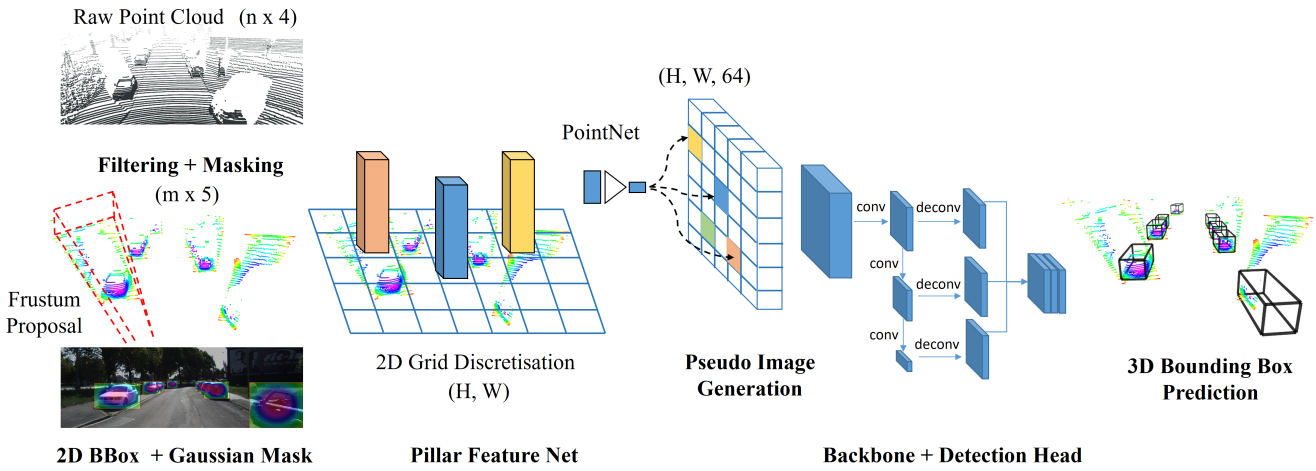

Fig. 3: **Frustum-PointPillars architecture:** Given the 2D detections of the objects in a scene, we extrude 2D detections into 3D bounding frustums and the points outside the frustums are removed. For each 2D detection, a mask is created using a Gaussian function representing the likelihood that the pixels belong to the object. Likelihood values are projected on the point cloud and the entire 3D space is discretized in a 2D grid, forming a set of pillars. A PointNet is used within each non-empty pillar to extract pillar-wise features that are scattered back to a 2D pseudo image. A set of convolution and deconvolutions are used to extract spatial features at multiple resolutions. Finally, an SSD style detection head is used for the regression of the 3D bounding box parameters.

in Fig. 2. These smaller frustums define groups of local points and PointNet is used on the frustums to generate 2D feature maps. Then, a fully convolutional network (FCN) is used to down-sample and up-sample the feature maps so that their features are fully fused across the frustum axis at a higher frustum resolution. F-ConvNet supports an end-to-end and continuous estimation of oriented 3D boxes. Both of these approaches processes each 2D detection in a scene individually which significantly increases computation time.

Motivated to address the limitations in [14], [15], we propose a novel method for 3D object detection termed Frustum PointPillars (F-PointPillars).

III. Frustum PointPillars Architecture

Given the 2D detections of the objects in the scene and their corresponding frustum volume, F-PointPillars subdivides the 3D space within the frustum volume into a top-view, finer 2D grid as shown in Fig. 1. Unlike F-ConvNet, which subdivides the 3D bounding frustum into a sequence of smaller frustums (Fig. 2), we have a much higher resolution for feature extraction. We use PointNet on each cell of the grid to extract pillar features and use 2D convolutions to extract spatial features. We then use a single-stage, anchor-based method to predict the 3D bounding box for all the objects in the scene simultaneously [16]. Provided the 2D detections, F-PointPillars is end-to-end trainable.

F-PointPillars consists of 4 main stages as shown in Fig. 3: (1) Frustum proposal; (2) PointCloud masking; (3) Pillar feature encoding network that converts a point cloud to a sparse pseudo image; and (4) Backbone to process the pseudo-image and produce a high-level representation, and a detection head to regress position, orientation and size of 3D bounding boxes.

A. Frustum Proposal

With a known camera projection matrix, a 2D bounding box can be extruded to a frustum (with near and far planes specified by depth sensor range) that defines a 3D search space for the object. Another way of viewing this is that, given the camera and LiDAR calibration parameters, 3D LiDAR points can be projected onto the image plane. We then filter out all the points outside of the 2D bounding boxes, significantly reducing the number of points to be processed within a scene. Let $P \in \mathbb{R}^3$, a point in the point cloud, transformation matrix $T : \mathbb{R}^3 \rightarrow \mathbb{R}^2$ transforms P to its projection $\bar{P} \in \mathbb{R}^2$ on the image plane.

B. Point Cloud Masking

The 2D bounding boxes predicted by the 2D detector are designed to enclose the object. The points inside 2D boxes can belong to the object itself with some foreground and background clutter. Pedestrians are generally on the sidewalks, in groups, and close to other buildings and objects. In the case of pedestrians, foreground and background clutter are not well Isolated as shown in Fig. 4. One alternative would be to use the object mask such as the one generated by mask-RCNN instead of 2D bounding boxes to reduce the clutter, however, it can worsen the overall performance of 3D object detection due to the accumulated error from inaccurate object mask predictions as discussed in [14]. Moreover, SOTA object masking approaches like Mask-RCNN are significantly slower than SOTA 2D detection approaches [17].

Considering real-time constraints, and issues with 2D mask [14], we propose a simpler and faster approach to use a probability mask. The region near the center of a 2d bounding box is more likely to be occupied by the object, the projected 3d points near the center region are also more likely to belong to the object instead of the background clutter. We define the likelihood of the points belonging to the object as a Gaussian function:

$$\mathcal{L}(\bar{x}, \bar{y}) = \exp\left(-\frac{(\bar{x}-\bar{x}_0)^2}{2w^2} - \frac{(\bar{y}-\bar{y}_0)^2}{2h^2}\right) \quad (1)$$

where $\bar{x}, \bar{y}$ are point cloud projection on image plane, $\bar{x}_0, \bar{y}_0$ are the center coordinates and w, h are the width and height of the 2D bounding box. $\alpha = w^2$, $\beta = h^2$ define the curvature of the likelihood function. We add the likelihood value to the point P as an additional feature vector. Input point feature C_{in} is now $D = 5$ dimensional $(x, y, z, intensity, \mathcal{L})$. If a point is shared by multiple 2D bounding boxes, then the highest likelihood value is chosen.

C. Pillar Feature Encoding and Pseudo Image

After filtering out the points outside of frustum areas and masking the remaining points, we divide the 3D space into 2D grid of shape (H, W) and the cell resolution of r. The points in the non-empty cells are re-sampled to a fixed number N. The points in each pillar are further augmented by adding extra features: x_c, y_c, z_c, x_p and y_p where the c subscript denotes distance to the arithmetic mean of all points in the pillar and the p subscript denotes the offset from the pillar center. Input point feature C_{in} is now D = 10 dimensional. A simplified version of PointNet with (N, C_{in}) as input and $(1, C_{out})$ as output is used to extract features per non-empty cell of the grid. These cell wise features or pillar features are tensors of size C_{out} creating a pseudo-image of size (H, W, C_{out}). It can be noted that the 3D space can also be divided into voxels instead of a 2D grid as in [6] but it exponentially increases computational overhead without a significant increase in the performance [8].

D. Backbone and Detection Head

To extract spatial features, we use a similar backbone as [8]. The backbone consist of two sub-network: one top-down network that uses sequence of 2D convolution blocks to produce features at increasingly small spatial resolution. The second network performs upsampling using transposed 2D convolutions and concatenation of the top-down features as shown in Fig. 3.

For the detection head we use a framework similar to the Single Shot Detector (SSD) setup [16]. We match the prior anchor boxes to the ground truth using 2D Intersection over Union (IoU). Given a positive 2D match, bounding box parameters are regressed, the height and elevation become additional regression targets.

IV. Training and Experiments

Given the 2D detections, F-PointPillars is trained end-to-end. We initialized all the weights randomly using a uniform

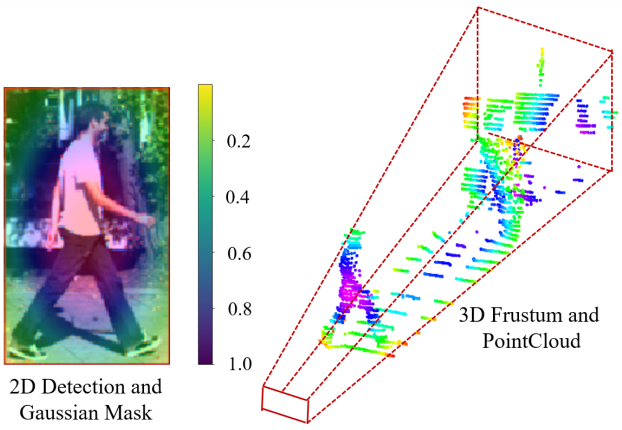

Fig. 4: Left: Illustration of the Gaussian Mask representing likelihood of the pixels belonging to the detected object. Right: Corresponding 3D frustum and masked point cloud.

distribution. Similar to SECOND [7], we used fixed-size anchors with rotations of 0 and 90 degrees. The anchor sizes are determined based on the means of the sizes and center coordinates of all ground truths in the KITTI dataset. Ground truth boxes and anchors are defined by $(x, y, z, w, l, h, \theta)$. Each anchor is assigned a class either to ground-truth objects (positive) or to the background (negative), based on IoU matching. Anchors are also assigned a 7-vector whose elements consist of regression targets:

$$\Delta x = \frac{x_{gt} - x_a}{d_a} \Delta y = \frac{y_{gt} - y_a}{d_a} \Delta z = \frac{z_{gt} - z_a}{h_a}$$

$$\Delta w = \log\frac{w_{gt}}{w_a} \Delta l = \log\frac{l_{gt}}{l_a} \Delta h = \log\frac{h_{gt}}{h_a} \quad (2)$$

$$\Delta\theta = \sin(\theta_{gt} - \theta_a)$$

where subscript gt denote ground truth and subscript a denote anchor boxes and $d_a = \sqrt{(w_a)^2 + (l_a)^2}$.

A. Loss Function

Similar to [8] we use combination of losses for the training of Frustum-PointPillars. 1) L_{loc}: SmoothL1 loss is used for regression of residuals between ground truth and anchors, 2) L_{dir}: a Softmax classification loss is used on the discretized directions to distinguished between flipped boxes and 3)L_{cls}: a Focal loss for object classification. The total loss is:

$$L_{total} = (\beta_{loc} L_{loc} + \beta_{cls} L_{cls} + \beta_{dir} L_{dir}) \quad (3)$$

where, β_{loc}, β_{cls} and β_{dir} are the hyper parameters to give weightage for different losses.

B. Dataset

We use the KITTI 3D object detection dataset for all our experiments and evaluation [18]. The KITTI dataset consists of samples with 3D point clouds, images and corresponding camera-LiDAR calibration data. The dataset is divided into 7481 training and 7518 testing samples. Cars, pedestrians and cyclists are the dominant object categories in KITTI

dataset and only these categories are considered for the benchmark. Following the work of MV3D, we split the official training dataset into 3712 training samples and 3769 validation samples [2]. We perform all our experiments on this train/val split. We also provide our results on KITTI test set obtained after submission on the KITTI server. We use the images to get the 2D region proposals, beyond which we only use the LiDAR point clouds to train the network.

C. Dataset Augmentation

The KITTI dataset contains fewer 3D objects (positive classes) per sample compared to the background (negative classes). Data augmentation becomes essential for high performance on the KITTI benchmark [6], [7]. The SECOND detector introduced a data augmentation process for 3D point clouds, where the authors first create a database of ground-truth 3D boxes and their associated point clouds. Then, during training, 3D boxes for cars, pedestrians, and cyclists are randomly selected from this database and placed in the current pointcloud to increase the number of positive classes. All 3D boxes are further augmented individually by applying rotations and translations.

When using RGB features with point cloud for sensor fusion such complex data augmentations are not possible [11]. F-PointPillar uses RGB images but only to generate 2D detections for creating frustums and masking the point cloud. We extend the data augmentation of SECOND to work with our multi-stage network architecture. We project all ground truth 3D boxes in the augmented point cloud on the image plane to find corresponding 2D bounding boxes. To simulate inconsistencies in 2D detection by a 2D object detector we apply random shift and add noise to the dimensions (uniformly drawn from [-0.1, 0.1]). Augmented 2D detections are then used to create frustums and mask point clouds. It is to be noted that in the case of using an object mask instead of the proposed Gaussian mask such data augmentations would not be feasible resulting in the poorer performance of the network. Also for evaluation, the KITTI benchmark only provides ground truth 2D boxes and object masks are not yet available.

D. Implementation Details

For the training of F-PointPillars, we use ground truth 2D detections provided by the KITTI dataset. We show in the result section how the accuracy of 2D detections affects the performance of our network. We train two networks one for car detection and another for pedestrian and cyclist detection. For car detection, we discretize the environment into a 2D grid of size $(70, 80)$ meters, and for pedestrian detection, we use a grid size of $(40, 50)$ meters. We remove all the points outside the field of view of the camera. The maximum number of points per pillar (N) is kept at 100. We keep the cell resolution r as 0.16m x 0.16m for both the networks. The hyper-parameters β_{cls}, β_{loc} and β_{dir} were set to 1.0, 2.0 and 0.2 respectively. We use Adam optimizer with an initial learning rate of $2{\cdot}10^{-4}$ and decay the learning rate by a factor of 0.8 every 15 epochs. We use a batch size of 2 to train the model. In our ablation study, we observed the network converges differently for different settings on average it takes roughly 120 epochs. Training using an Nvidia GTX 1080 GPU takes around 18 to 20 hours. We use the PyTorch machine learning framework for the development of our model [20].

V. Results

A. Quantitative Evaluation

We evaluate F-PointPillars on KITTI 3D detection benchmarks [18]. In this work, we primarily focus on pedestrian detection. Table I compares the performance of F-PointPillars with state-of-the-art methods on the KITTI test. At the time of submission, F-PointPillars ranked among the top 20 approaches for 3D detection in the pedestrian category. For the final submission on the KITTI server, we used 2D region proposals obtained by Faster R-CNN trained on EuroCityPersons dataset [21]. We use off-the-shelf code from the Pedestron repository which is based on the MMdetection toolbox. We compare our results with approaches using different modalities: STD [19] and PointPillars [8] uses only LiDAR data; AVOD-FPN [11] uses RGB and LiDAR with early fusion method; F-PointNet [14] and F-ConvNet [15] similar to our method use RGB and LiDAR with multi-stage design architecture. Our method outperforms the mentioned approaches for 3D detection at the hard difficulty level. For Bird-Eye-View (BEV) detection our method outperforms other approaches at all difficulty levels indicating the strength of our network for localizing objects in 3D space. This is in part due to the masking of the point cloud as it provides the network more information about the location of objects in 3D space.

Ablation Study: To understand the effectiveness of masking the point cloud we perform ablation experiments using our KITTI val set and provide results for Car, pedestrian, and Cyclist object categories. Table II and Table III shows a comparison of F-PointPillars with and without masking point cloud. To quantify the improvement in performance by using a multi-stage design approach we also compare with PointPillars [8]. Table II and Table III shows masking the point cloud improves 3D detection and BEV detection accuracy across all the object categories and difficulty levels. In another set of experiments, we slightly alter the values of α and β to test different Gaussian functions for the masking of the point cloud but it does not result in a significant difference in the performance of the network.

Table II and Table III also compares our results with F-PointNet and F-ConvNet on val set. For the comparison, we use ground truth 2D region proposals with the validation dataset. It can be seen that F-PointPillars outperforms these two approaches in nearly all object categories and difficulty levels.

Effect of 2D region proposals: Our method relies on 2D region proposals to reduce our search space in 3D. To investigate how much the accuracy of 2D region proposals

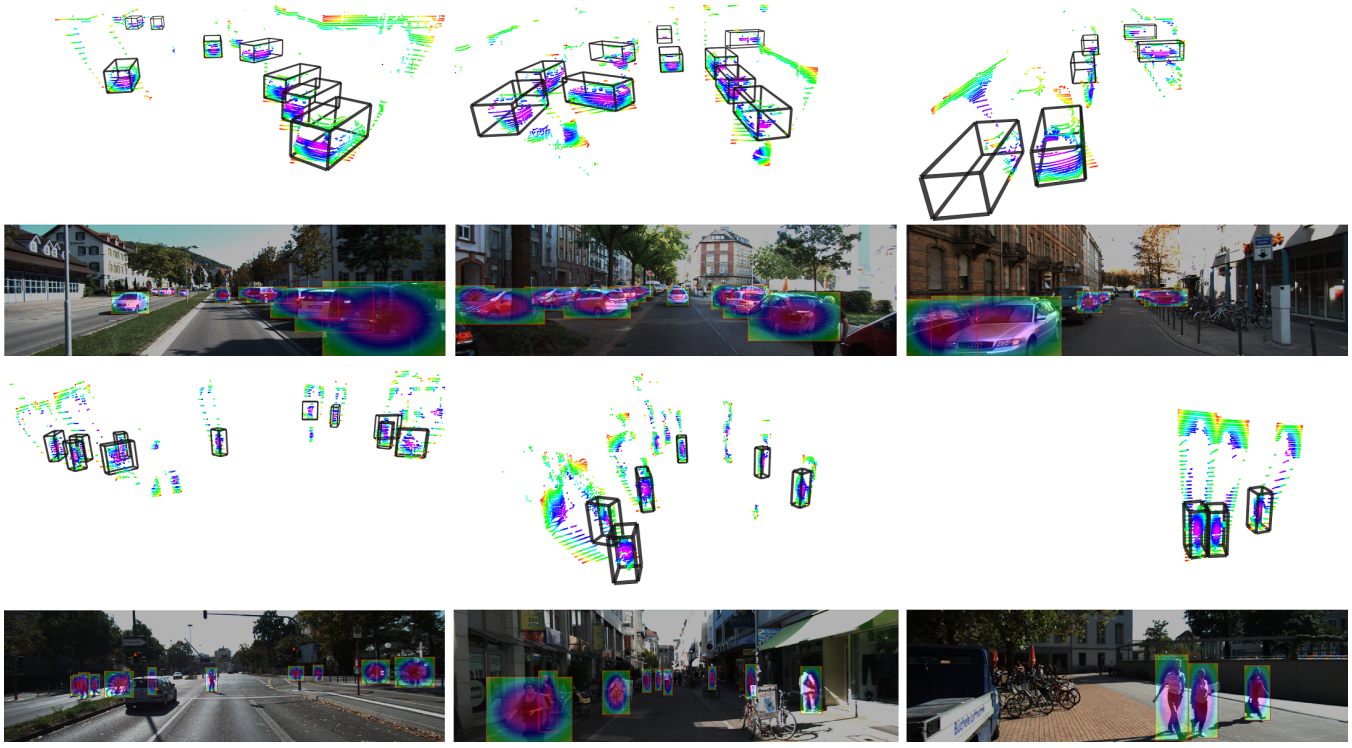

Fig. 5: **3D bounding box prediction by F-PointPillars**: Illustrations show diverse scenes from KITTI dataset. Images show 2D region proposals from an off-the-shelf 2D detector and respective Gaussian mask. Corresponding point cloud after filtering and the masking is shown on top of each image.

Method	**3D detection**			**BEV detection**			**Runtime**
	Easy	**Mod.**	**Hard**	**Easy**	**Mod.**	**Hard**	
STD [19]	53.29	42.47	38.35	60.02	48.72	44.55	0.08 s
PointPillars [8]	51.45	41.92	38.89	57.60	48.64	45.78	**0.07 s**
AVOD-FPN [11]	50.46	42.27	39.04	58.49	50.32	46.98	0.1 s
F-PointNet [14]	50.53	42.15	38.08	57.13	49.57	45.48	0.17 s
F-ConvNet [15]	**52.16**	**43.38**	38.80	57.04	48.96	44.33	0.47 s
F-PointPillars (Ours)	51.22	42.89	**39.28**	**60.98**	**52.23**	**48.30**	**0.07 s**

TABLE I: AP (%) on KITTI test set for pedestrian detection.

Method	**Car**			**Pedestrian**			**Cyclist**		
	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
F-PointNet [14]	83.76	70.92	63.65	**70.00**	61.32	53.59	77.15	56.49	53.37
F-ConvNet [15]	**89.31**	79.08	77.17	-	-	-	-	-	-
PointPillars	84.06	75.13	69.43	62.57	57.52	51.17	81.96	62.11	57.39
F-PointPillars (no mask)	88.02	77.87	76.15	67.24	60.69	54.71	82.12	63.06	60.54
F-PointPillars	88.90	**79.28**	**78.07**	66.11	**61.89**	**56.91**	**87.54**	**72.78**	**66.07**

TABLE II: AP (%) on KITTI val set for 3D object detection.

Method	**Car**			**Pedestrian**			**Cyclist**		
	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
F-PointNet [14]	88.16	84.02	76.44	**72.38**	66.39	59.57	81.82	60.03	56.32
F-ConvNet [15]	**90.42**	88.99	86.88	-	-	-	-	-	-
PointPillars	89.99	87.13	85.15	70.54	65.70	60.18	85.07	65.06	61.72
F-PointPillars (no mask)	89.95	88.38	87.19	71.69	66.20	60.95	83.17	65.76	63.03
F-PointPillars	90.20	**89.43**	**88.77**	72.17	**67.89**	**63.46**	**88.58**	**76.79**	**74.80**

TABLE III: AP (%) on KITTI val set for BEV detection.

2D detections (Mod.)			3D detections (Mod.)		
Car	**Ped.**	**Cyclist**	**Car**	**Ped.**	**Cyclist**
89.47	62.47	72.70	76.79	58.76	64.75
90.30	76.39	81.35	77.39	59.27	65.36
100	100	100	79.28	61.89	72.78

TABLE IV: Influence of 2D region proposal on F-Pointpillars. Left: Each row represent results from a different 2D detector on KITTI val set (Mod. difficulty). Right: Corresponding output of F-Pointpillar 3D detection AP (%).

influence the performance, we perform experiments with 2D object detectors of different practical performance namely F-PointNet [14], PV-RCNN [22] and ground-truth 2D boxes in the KITTI dataset. Table IV confirm that the accuracy of 2D region proposals positively affects our method. While from Table IV and Table II together it can be inferred that even after using predicted 2D bounding boxes our approach still performs better than PointPillars for all object categories .

Inference time: Real-time 3D detection is critical for autonomous navigation. F-PointPillars uses the backbone from PointPillars and predicts 3D bounding boxes for all the objects in the scene in a single forward pass. In contrast, F-PointNet and F-ConvNet process each 2D detection in the scene individually and require multiple forward passes. Table V shows detailed analysis of the computation time for F-PointPillars. Our method achieves a runtime performance of 14 Hz, which is significantly better compared to other SOTA multi-stage approaches like F-PointNet (5 Hz) and F-ConvNet (2 Hz) as shown in Table I. Note that the pre-processing, frustum generation, filtering, and masking of the point cloud is performed on a CPU that could be further accelerated using GPU. In comparison to PointPillars, our network has to process a significantly less number of points, though the gain in computation time for model forward pass is negated by an extra pre-processing task to be performed by our network.

Task	**Time**	**Device**
pre-processing	10.09 ms	CPU
Frustum + Masking	18.29 ms	CPU
Model forward pass	12.57 ms	GPU
Post Processing	28.88 ms	GPU
Total	69.83 ms (14.61 Hz)	

TABLE V: Analysis of computation time required by F-Pointpillars

B. Qualitative Evaluation

The illustrations in Fig. 5 show the output of our F-PointPillars architecture on the KITTI dataset. We see that our model outputs accurate 3D bounding boxes even in diverse and challenging scenarios. We also observe that even in cases of multiple 2D region proposals overlapping (e.g. parallel parked cars) and with few 3D points, our model still can predict the pose of 3D boxes correctly. We also see some failure cases especially for detecting far away pedestrians.

We also observe that the projection of the Gaussian mask on the point cloud, in general accurately indicates the likelihood of the points belonging to the object of interest. In certain cases, as in Fig. 4, where a pedestrian is seen extending its arm, the center of the 2D bounding box is slightly shifted and some of the background points are wrongly masked as more likely of belonging to the object.

VI. Conclusions

In this paper, we tackled the challenging problem of 3D object detection in point clouds. We proposed Frustum-PointPillars, a multi-stage design approach that uses both RBG and LiDAR data for 3D detection. Given 2D region proposals, we extrude 2D detections into 3D bounding frustums. Points outside the frustums are removed and the entire 3D space is discretized into a 2D grid. We use Pillar Feature encoding networks to extract features and predict 3D bounding boxes. We also proposed a novel approach for masking 3D point clouds with likelihood values of the points belonging to the object. We provide a qualitative and quantitative evaluation of our approach on the KITTI dataset. We compare our results with other state-of-the-art approaches for pedestrian detection. F-PointPillars outperforms other multi-stage approaches for 3D pedestrian detection in hard difficulty level and BEV detection in all difficulty levels. Our method achieves a run-time of 14 Hz and is significantly faster than other multi-stage approaches.

Filtering of point cloud with the help of 2D region proposals significantly reduces the number of non-empty cells in the pseudo-image. We plan to further improve the runtime of our method by using sparse convolutions in the backbone of our network. Currently, we train two different networks: one for car and another for pedestrian/cyclist detection as multi-class 3D detection performance is poor. In the future, we plan to leverage the object class information provided by 2D detectors to improve multi-class 3D detection. Another line of future work could be improving 2D detection in low light conditions using data from RGB and Lidar.

Acknowledgment

This work was conducted at Inria, team Chroma. The authors would like to thank all the team members for their constant support on this research work. This work has been conducted within the scope of ES3CAP (Embedded Smart Safe Secure Computing Autonomous Platform) project.

References

[1] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, *et al.*, "A perception-driven autonomous urban vehicle," *Journal of Field Robotics*, vol. 25, no. 10, pp. 727–774, 2008.

[2] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1907–1915.

[3] C. R. Qi, H. Su, M. Nießner, A. Dai, M. Yan, and L. J. Guibas, "Volumetric and multi-view cnns for object classification on 3d data," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 5648–5656.

[4] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, vol. 1, no. 2, p. 4, 2017.

[5] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems*, 2017, pp. 5099–5108.

[6] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4490–4499.

[7] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, p. 3337, 2018.

[8] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 12 697–12 705.

[9] A. Paigwar, Ö. Erkent, C. Wolf, and C. Laugier, "Attentional PointNet for 3D-Object Detection in Point Clouds," in *CVPR 2019 - Workshop on Autonomous driving*, Long Beach, California, United States, June 2019, pp. 1–10. [Online]. Available: https://hal.inria.fr/hal-02156555

[10] D. Sierra-González, A. Paigwar, O. Erkent, J. Dibangoye, and C. Laugier, "Leveraging dynamic occupancy grids for 3d object detection in point clouds," in *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 2020, pp. 1188–1193.

[11] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander, "Joint 3d proposal generation and object detection from view aggregation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–8.

[12] M. Liang, B. Yang, Y. Chen, R. Hu, and R. Urtasun, "Multi-task multi-sensor fusion for 3d object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7345–7353.

[13] L. Guan, Y. Chen, G. Wang, and X. Lei, "Real-time vehicle detection framework based on the fusion of lidar and camera," *Electronics*, vol. 9, no. 3, p. 451, 2020.

[14] C. R. Qi, L. Wei, W. Chenxia, *et al.*, "Frustum pointnets for 3d object detection from rgb-d data [c/ol]," *Computer Vision Pattern Recog*, pp. 11–22, 2017.

[15] Z. Wang and K. Jia, "Frustum convnet: Sliding frustums to aggregate local point-wise features for amodal 3d object detection," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 1742–1749.

[16] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[17] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[18] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 3354–3361.

[19] Z. Yang, Y. Sun, S. Liu, X. Shen, and J. Jia, "Std: Sparse-to-dense 3d object detector for point cloud," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1951–1960.

[20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[21] I. Hasan, S. Liao, J. Li, S. U. Akram, and L. Shao, "Pedestrian detection: The elephant in the room," *arXiv preprint arXiv:2003.08799*, 2020.

[22] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. L. Pv-rcnn, "Point-voxel feature set abstraction for 3d object detection," *arXiv preprint arXiv:1912.13192*, 2019.