Reconfigurable Voxels: A New Representation for LiDAR-Based Point Clouds

Tai Wang, Xinge Zhu, Dahua Lin
The Chinese University of Hong Kong
{wt019, zx018, dhlin}@ie.cuhk.edu.hk

Abstract:
LiDAR is an important method for autonomous driving systems to sense the environment. The point clouds obtained by LiDAR typically exhibit sparse and irregular distribution, thus posing great challenges to the detection of 3D objects, especially those that are small and distant. To tackle this difficulty, we propose Reconfigurable Voxels, a new approach to constructing representations from 3D point clouds. Specifically, we devise a biased random walk scheme, which adaptively covers each neighborhood with a fixed number of voxels based on the local spatial distribution and produces a representation by integrating the points in the chosen neighbors. We found empirically that this approach effectively improves the stability of voxel features, especially for sparse regions. Experimental results on multiple benchmarks, including nuScenes, Lyft, and KITTI, show that this new representation can remarkably improve the detection performance for small and distant objects, without incurring noticeable overhead costs.

Keywords: LiDAR-based Point Clouds, 3D Detection, Reconfigurable Voxels

1 Introduction

LiDAR has been widely used in driver assistance or autonomous driving systems [1], which senses the environment via reflected laser light and produces 3D point clouds as the output. Compared to conventional 3D data, *e.g.* those obtained by 3D scanner for object modeling [2], the 3D point clouds derived by LiDAR are usually much more sparse and irregular. Therefore, effective handling of such data requires new methods – in particular new representations tailored to LiDAR's special characteristics.

Existing approaches to 3D point cloud representation mainly follow two streams: *point-based* and *voxel-based*. Point-based methods [3, 4, 5], among which PointNet [3] is a representative, focus on the processing of individual points and integrate the information on top. Due to the narrow focus in the initial processing stage, point-based methods often lack the capability of capturing large spatial structures. Voxel-based methods [2, 6, 7, 8], instead, begin with the space. Specifically, they quantize a 3D space into cells and process the information based on the cells instead of individual points. While this allows spatial distributions of greater scale to be captured, the tradeoff between representation precision and computational complexity remains an open problem. This problem is especially crucial for sparsely distributed point clouds.

In this work, we choose to follow the voxel-based approach, due to its inherent strength in modeling spatial distributions, while aiming to tackle the difficulties caused by the sparsity and irregularity in LiDAR data. Specifically, we propose *Reconfigurable Voxels*, a generic voxel-based representation. As shown in Fig. 1, for each voxel, it adaptively *reconfigures* its neighborhood through a biased random walk so as to cover its surrounding regions more effectively, and then derives an embedding thereon.

The proposed method has several appealing properties: (1) *More stable representation.* By constructing features upon an adaptive neighborhood, it effectively mitigates the difficulties caused by sparsity and irregularity, *e.g.* voxels with few or even no points, thus resulting in more stable features. (2) *Strong locality.* While allowed to be stretched, the reconfigured neighborhood remains within a surrounding region of the target location, and therefore still preserves strong locality. This is important for capturing local structures. (3) *High efficiency.* It is noteworthy that the construction of the voxel neighborhoods can be done in one traversal of the dataset and then fixed. Compared to the overall computing cost, this additional overhead is insignificant.

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

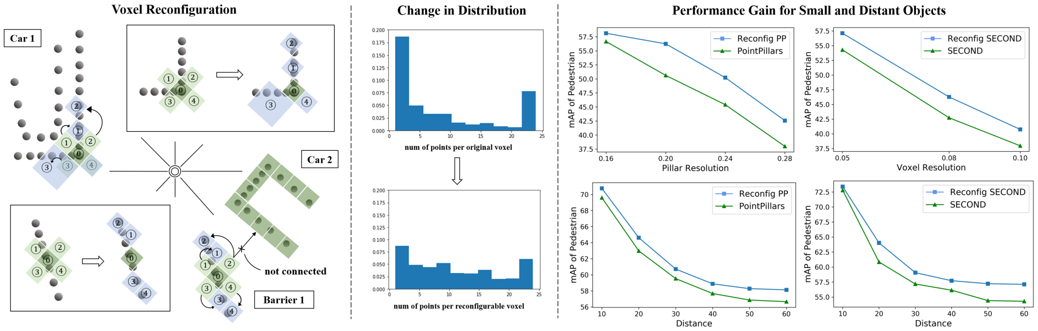

Figure 1: **Left:** we take the original voxel together with its 4 neighbor voxels as a whole to encode features. Light green and blue voxels represent neighbors before and after reconfiguration respectively. Note that the transition of neighbors is only carried out on the same connected component composed of non-empty voxels. Without loss of generality, reconfiguration here operates on X-Y plane. Extension to 3D is straightforward. **Middle:** The reconfigurable voxels greatly improve the imbalance of sampling points in different voxels, thus encoding more robust features in sparse regions. **Right:** Our method can consistently improve the detection performance for small and distant objects under multiple settings and frameworks on KITTI.

We evaluated the proposed representation method on multiple benchmarks of 3D detection, including nuScenes [9], Lyft [10] and KITTI [11]. On these datasets, it consistently achieved significant performance gains. Moreover, our study also shows that *Reconfigurable Voxels* can effectively handle the sparse point clouds, thus substantially improving the capability of detecting small and distant objects: it can boost the performance of most small objects by over 2% mAP on all datasets and objects over 20 meters away by 4.4% NDS on nuScenes.

2 Related Work

3D Object Detection The problem of 3D object detection has been widely explored before deep learning approaches emerged. Firstly, work focusing on indoor scenes includes: [12] modeled contextual relationship to guide object detection; [13] designed sliding-shapes to realize detection in RGB-D images; and VoteNet [14] utilized a reformulation of Hough voting in 3D case, *etc.*

Among work for autonomous vehicles, although image-based methods [15, 16] have made great progress, their performance is still far behind LiDAR-based methods. The methods using LiDAR data can be divided into two categories according to the data types used: the methods using multimodal data and the methods using only LiDAR data. The first batch of methods [17, 18] resolved this problem by fusing features extracted from images and projections of point clouds, which reduced 3D problems to 2D cases. Then with PointNet [3] proposed, it became possible to extract features directly from point cloud data. Earlier works deploying this backbone like [19] used 2D detection guided frustum to reduce search space. The other methods follow two streams: voxel-based and point-based. Among voxel-based methods, [20] utilized hand-crafted features to detect objects in bird view map, while VoxelNet and SECOND [7, 21] directly processed 3D partitioned voxels, used PointNet to encode, and trained them as a module in the end-to-end framework. PointPillars [22] simplified the representation to pillar, thus obtaining a bird view pseudo image after encoding, and further improved the efficiency with 2D convolution. Point-based methods [5, 23], instead, designed frameworks to extract proposals and detected objects in point level based on scene segmentation module, but the number of points needed to process is always a limitation to these methods. Therefore, our work carries on the exploration in voxel-based methods. Although recent work [24, 25] began to explore to encode features more effectively from better representation, it is not divorced from the original voxel layout or simply fuses multi-scale information. In comparison, our reconfigurable voxels is a kind of deformable voxel representation which is constructed in reasonable local space according to the spatial distribution of points, so as to depict the shape of objects implicitly.

Voxel-based Learning on Point Cloud Utilizing voxel as the basic representation is an intuitive way to migrate 2D methods to 3D problems. To mention only a few, [2] proposed 3D ShapeNets to achieve object recognition and shape completion. [26] improved it with fewer input parameters. [6, 27] used octree structure to improve the efficiency problem caused by 3D convolution. Nevertheless, these works only focus on the case of a single object. Given the difference between the point cloud of CAD models and LiDAR-based data, how to transfer these ideas is still an open question.

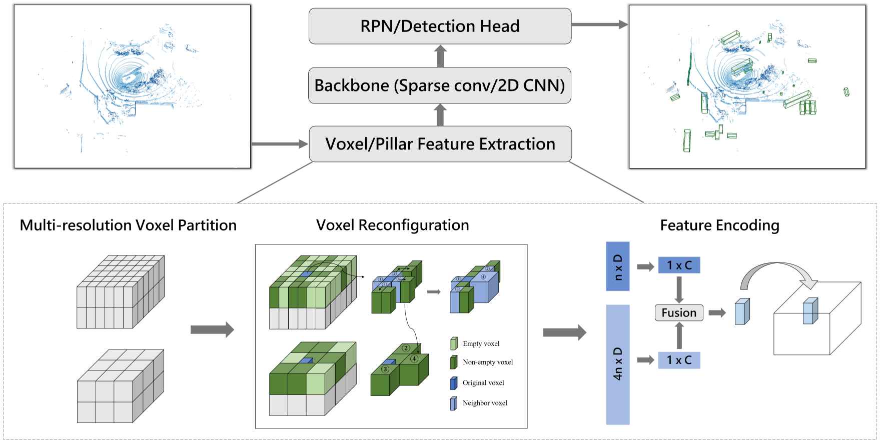

Figure 2: An overview of our pipeline. Reconfigurable voxels is a generic representation which can be exploited when extracting voxel features. With feature fusion of the original voxel and its 4 reconfigured neighbors, we can adaptively encode local shape without modifying the data structure of the following operations. Here we show voxel reconfiguration module in the multi-resolution case. Arrows between voxels indicate transitions of neighbors. See more details in Sec. 3.4

Deformable Convolutional Networks The traditional convolution can be regarded as a fixed kernel executing point-wise inner-product with the corresponding content of the image at a specific location. In the 2D case, deformation modeling is a common and principled problem, and there are many works like [28] targeting it and designing variant convolutions to extract features flexibly. In addition, some other works considered to sample in the kernel space without changing the theoretical receptive fields, such as [29] in the 2D case and [30] in 3D point clouds. In comparison, while our devised reconfiguration is similar with deformation, the motivation is not the same: the deformation and scale problems in 2D do not exist in 3D cases. The problem we try to tackle is the difficulty of detecting small and distant objects caused by the irregular spatial distribution of LiDAR-based point clouds. It is intuitively more straightforward to introduce deformation into the point-to-voxel process instead of modifying the convolution operation on the voxel feature maps.

3 Approach

Overview How to construct an efficient representation from sparsely and irregularly distributed point cloud is a key problem for scene understanding tasks, like 3D detection in autonomous driving. In general, a voxel-based 3D detection framework groups raw point cloud into voxels, applies voxel feature encoding layers and scatters them back for the subsequent convolutional backbone and prediction of 3D bounding boxes. Our reconfigurable voxels is a generic representation when partitioning the space, which encodes local information more effectively by covering voxel neighbors based on the spatial distribution. Next in this section, we will elaborate the construction method and technical details of reconfigurable voxels in turn, and finally extend the single-resolution case to multi-resolution, making the whole design more flexible and robust.

3.1 Construction of Reconfigurable Voxels

To address the problem caused by sparsity and irregularity, a simple idea is to allow the existence of voxels in different sizes, but this easily destroys the data structure of subsequent computations, and thus is not conducive to maintaining the real-time performance of the algorithm. Therefore, we propose that on the basis of primitive voxel partition, the original voxel can cover its surrounding regions more effectively by reconfiguring its neighborhood based on the local spatial distribution.

Specifically, the process of constructing reconfigurable voxels is as follows: Firstly, the whole scene is divided into voxels of the same size, and the index of each neighbor is recorded in the process of partitioning. Thus, the construction of graphs can be completed in a one-time traversal process. Subsequently, we make every neighbor of each voxel carry out a biased random walk. A mechanism is designed to make the neighbors walk to voxels with denser point clouds. Finally, we compose these reconfigured neighbors with the original voxel, extract features, fuse them, and scatter the final features back to the original location. See the process in Fig. 2.

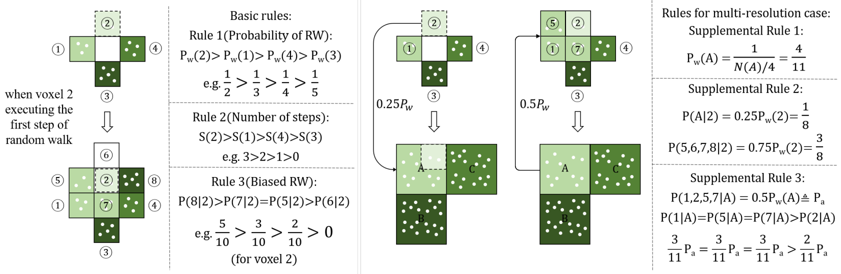
②
①
④
③
when voxel 2 executing the first step of random walk
⑥
⑤
②
⑧
①
⑦
④
③
Basic rules:
Rule 1(Probability of RW):
$P_w(2) > P_w(1) > P_w(4) > P_w(3)$
e.g. $\frac{1}{2} > \frac{1}{3} > \frac{1}{4} > \frac{1}{5}$
Rule 2(Number of steps):
S(2)>S(1)>S(4)>S(3)
e.g. 3>2>1>0
Rule 3(Biased RW):
P(8|2)>P(7|2)=P(5|2)>P(6|2)
e.g. $\frac{5}{10} > \frac{3}{10} > \frac{2}{10} > 0$
(for voxel 2)
②
①
④
③
$0.25P_w$
A
C
B
⑤
②
①
⑦
④
③
$0.5P_w$
A
C
B
Rules for multi-resolution case:
Supplemental Rule 1:
$P_w(A) = \frac{1}{N(A)/4} = \frac{4}{11}$
Supplemental Rule 2:
$P(A|2) = 0.25P_w(2) = \frac{1}{8}$
$P(5,6,7,8|2) = 0.75P_w(2) = \frac{3}{8}$
Supplemental Rule 3:
$P(1,2,5,7|A) = 0.5P_w(A) \triangleq P_a$
P(1|A)=P(5|A)=P(7|A)>P(2|A)
$\frac{3}{11}P_a = \frac{3}{11}P_a = \frac{3}{11}P_a > \frac{2}{11}P_a$

Figure 3: An example of 3 basic rules and 3 supplemental rules for biased random walk is shown above. Note that the basic rule 2 is only needed before starting random walk while all the other basic and supplemental rules are followed when executing every step of random walk

It can be seen that this voxel partition process does not affect the operations of the subsequent backbone. Meanwhile, through the reconfiguration of neighbors, the vulnerability of voxel features in sparse regions is improved. Note that this process is free of learning parameters, which thus avoids possible indifferentiable problems when how to carry out random walk between voxels needs to be learned and maintains our end-to-end training. In addition, because these neighbors are only allowed to walk on the same connected component, it basically ensures that they will be in the adjacent area instead of freely running across the open area to other irrelevant objects, which leverages the sparsity of LiDAR data and depicts local shape of objects implicitly.

3.2 Biased Random Walking Neighbors

As mentioned previously, we hope that by designing a biased random walk scheme, neighbor voxels will tend to move to areas with dense points. An intuitive idea is when a voxel contains fewer points, it should be more likely to execute random walk and take more steps on the same connected component. In addition, voxels should have a greater probability of transitioning to those with denser points. We formulate this idea as follows.

Suppose the j-th voxel contains $N(j)$ points, the maximum number of points is n, the probability of executing random walk is $P_w(j)$, the number of steps it takes is $S(j)$, the voxel index of the i-th step is $w_j(i)$, the set of four neighbor voxels of $w_j(i)$ is $V(w_j(i))$, and the transition probability from i-th step voxel to the next step voxel is $P(w_j(i+1)|w_j(i))$, our mechanism is given by the following 3 basic rules:

$$P_w(j) = \frac{1}{N(j)} \tag{1}$$

$$S(j) = n - N(j) \tag{2}$$

$$P(w_j(i+1)|w_j(i)) = \frac{N(w_j(i+1))}{\sum_{v \in V(w_j(i))} N(v)} \tag{3}$$

where $P(w_j(i+1)|w_j(i))$ is not zero if and only if $w_j(i+1)$ and $w_j(i)$ are non-empty neighbor voxels to each other. From the first 2 rules, the more points a voxel has, the lower its random walk probability is and the fewer steps it takes. It should be noted that the number of steps are decided at the beginning for every neighbor voxel, which is different from the transition probability. In particular, when the number of points reaches the maximum, the step number is 0, meaning that once the random walking neighbor reaches the voxel with the largest number of points, it will not leave. Voxels with only one point take the most $n-1$ steps among all cases, and according to the statistics of random walk in 2D case, the distance traveled from starting point is approximately on the order of $\sqrt{n-1}$ on average. Finally, the third rule says when walking between voxels, the probability of transferring to voxels with dense points is higher, and the sum of probabilities is 1.

Up to now, we have preliminarily devised a scheme of biased random walk to achieve the transition between voxels that meet our requirements. It should be mentioned that this particular design sometimes needs to be adjusted according to the specific implementation and hyper parameters to ensure that voxel does not go too far. Specific adjustments are described in supplementary materials. See an example of this scheme in Fig. 3 and voxel reconfiguration results in Fig. 4.

3.3 Reconfigurable Voxels Encoder

With n point features of the center voxel and $4n$ point features of neighbor voxels, we utilize a function, denoted as ψ, to extract voxel features. If the i-th input center voxel features and neighbor

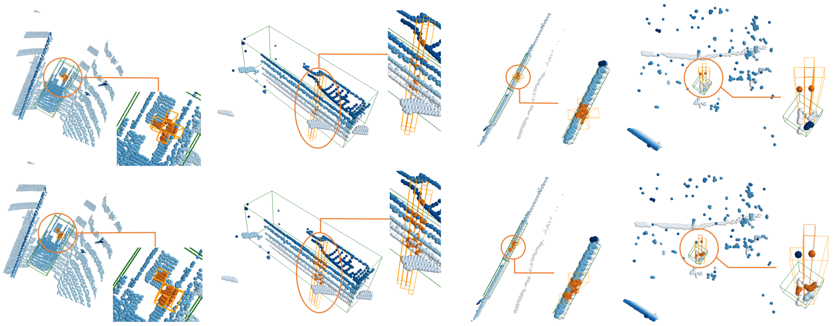

Figure 4: Visualization of reconfigurable voxels. We show dilated (top) and reconfigurable voxels (bottom) in orange boxes. Points in the partitioned voxels are also marked in orange. For other points, the darker the blue, the higher the points on z axis. It demonstrates that reconfigurable voxels quantize the space adaptively, in which biased random walk helps neighbors cover more meaningful regions

voxel features are denoted as f_{v_i} and $f_{V(v_i)}$, the derived i-th voxel features is denoted as $F(v_i)$, then:

$$F(v_i) = \psi(f_{v_i}, f_{V(v_i)}) \tag{4}$$

where we take the original center as the center of reconfigurable voxels to obtain relative locations in $f_{V(v_i)}$. For SECOND and PointPillars, considering their different partition methods in z axis, ψ has different implementations. SECOND partitions the space more carefully, so it is more difficult to form a connected component. For instance, suppose a voxel at a certain height does not have non-empty neighbors in SECOND, the case can be not true if the neighbor pillar contains points at different heights in PointPillars. So it should be careful when we leverage neighbor voxel features in PointPillars: our encoder needs to ensure that neighbor pillar features will not overwhelm the original pillar information. To this end, we adopt ψ as follows:

$$\psi(f_{v_i}, f_{V(v_i)}) = \phi_1([f_{v_i}, f_{V(v_i)}]_p) \tag{5}$$

$$\psi(f_{v_i}, f_{V(v_i)}) = [\phi_2(f_{v_i}), \phi_2(\sum_{j=1}^{4} W_j(f_{v_i}) f_{V_j(v_i)})]_f \tag{6}$$

where ϕ_1 is a low-level operation, average pooling, for SECOND, while ϕ_2 is a high-level operation, shared MLP and maxpooling, for PointPillars. $W_j(f_{v_i})$ is the weight corresponding to the j-th neighbor of v_i, which is derived from f_{v_i}. In a word, we just encode the concatenated features (of different points) in SECOND, while concatenate the encoded features in PointPillars. From this perspective, our approach basically aggregates more meaningful point features locally for a better input representation, and thus eases the burden of learning ψ as well as the following networks.

3.4 Multi-resolution Reconfigurable Voxels

So far, we have designed a method to construct reconfigurable voxels in the single-resolution case, in which we devise a scheme so-called *intra-resolution* random walk. In order to make it more flexible and robust, we extend it to the case of multi-resolution random walk, namely, *inter-resolution* random walk. Here we give a detailed implementation of two-resolution scenarios.

Firstly, suppose that under the initial resolution partition, the voxel size on the X-Y plane is $[l, w]$, and each voxel contains at most n points. To preserve the resolution of the original voxel, we consider the second resolution with a larger-voxel partition: the voxel size on the X-Y plane is $[2l, 2w]$. Then a large voxel will contain up to 4 small voxels. In order to ensure the consistency of data format, we record the indices of 4 children voxels for the large ones when implementing voxel partition. After completing the partition, we randomly sample the points in the large voxel to make it contain up to n points. As a result, the voxels with dense points will not contain more points with the change of spatial quantization, whereas the voxels with less than n points have chance containing enough data. Besides, this design also facilitates the convenience of subsequent voxel feature extraction.

Problems mentioned in Sec. 3.2 also exist when it comes to random walk operations between different resolutions. As Fig. 3 shows, we put forward 3 supplemental rules for multi-resolution case. Firstly, when computing P_w, we need to divide the number of points by 4 to make it consisent with the single-resolution case. For supplemental rule 2 and 3, we assume that the transition probability from smaller voxel to larger one is $0.25P_w$, and from larger voxel to smaller one is $0.5P_w$, which ensures that all voxels will remain in the original resolution at a higher probability. Note that it follows similar rules as the basic rule 3 when choosing which small voxel to transition. Finally, we will also record the neighbors of large voxels, and it satisfies Eqn. 3 when they execute random walk in the graph composed of large voxels.

Thus, we complete the generalization to the multi-resolution case. Specification of the algorithm is included in supplementary materials. In conclusion, this extension makes reconfiguration more flexible. In particular when the points in a voxel are very sparse, the higher probability to be a larger voxel will make it easier to contain more points, so as to ease the difficulty caused by sparsity. It should be noted that our purpose is to construct the new representation with voxels in different resolutions given the local spatial distribution of point clouds, which is different from general multi-scale tricks.

4 Experimental Setup

4.1 Datasets & Evaluation Metrics

We evaluated our approach on three commonly used benchmarks: nuScenes [9], Lyft [10] and KITTI [11]. NuScenes dataset is split in 700/150/150 scenes for training/validation/testing respectively. There are overall 1.4M annotated 3D boxes, far more than KITTI's 200K 3D boxes in 22 scenes. Lyft dataset has 180 and 218 scenes for training and testing respectively. It can be seen that nuScenes and Lyft have more data, more object categories and richer scenes than KITTI. Therefore, at first, we conducted toy experiments on KITTI to analyze the computational complexity and the efficacy of our method under different settings. Then we designed experiments on nuScenes and Lyft to test it on large-scale datasets. Finally, more detailed ablation studies on KITTI are given. It should be noted that nuScenes and Lyft have the same data format, and need to predict one key frame detection result every ten frames. Therefore, in those experiments, we transformed the point clouds of ten consecutive frames into the coordinate system of key frames and input them to the network for detection. As for metrics, distance-based mAP and nuScenes detection score (NDS[1]) were used as the main metrics on nuScenes, while mAP of all categories was compared under 0.5-0.95 IOU on Lyft. Here we name the much more strict metric in Lyft as mAP-3D for clarification. We follow the official evaluation protocol in KITTI experiments as well, *i.e.*, mAP was compared for different categories with 0.7 IOU threshold for car and 0.5 IOU for pedestrian and cyclist.

4.2 Implementation Details

Network Architectures Our whole framework follows the ideas of PointPillars and SECOND with the following adjustments in specific details.

First, when extracting features from voxels, we use different point features and different settings of X-Y resolution, max number of voxels and max number of points per voxel for different experiments. Another change on the PointPillars is that we implement multi-group head for the experiments on nuScenes and Lyft given the category diversity. See more details in supplementary materials.

Loss We use a loss function similar to that described in [22, 21]. It should be noted that we need to predict the object's velocity and attribute in the nuScenes experiment, so we add the velocity into the regression target and add attribute classification loss into the overall loss.

$$L_{loc} = \sum_{b \in (x,y,z,w,l,h,\theta,v_x,v_y)} \text{SmoothL1}(\Delta b) \tag{7}$$

where the weight of x, y, z, w, l, h, θ error is 1 and the weight for v_x, v_y is 0.5. The total loss is:

$$L = \frac{1}{N_{pos}}(\beta_{loc}L_{loc} + \beta_{cls}L_{cls} + \beta_{attr}L_{attr} + \beta_{dir}L_{dir}) \tag{8}$$

where N_{pos} is the number of positive anchors and $\beta_{loc} = 2$, $\beta_{cls} = 1$, $\beta_{attr} = 1$ and $\beta_{dir} = 0.2$.

Training Parameters For all the experiments, we trained randomly initialized networks end-to-end. Models were trained with ADAM optimizer, in which we adopted one-cycle policy.

Data Augmentation Data augmentation is particularly important for 3D detection. First, we establish the ground truth database of all objects as mentioned in [21]. During training, we sample a few objects which have fewer instances, and place them into different point clouds. Because this kind of augmentation may be unreasonable due to the characteristic of LiDAR sampling, we also analyze the number of different categories of objects in all samples, select specific samples, copy them, and alleviate the imbalance of the number of objects in all categories as [31] proposed. Finally, we randomly flip the LiDAR sweep along the x-axis or y-axis to realize global augmentation.

[1] NDS is a more comprehensive metric with consideration of attribute and velocity prediction in [9].

Table 1: Inference speed of models with and without reconfigurable voxels.

Method	**Speed(Hz)**
SECOND	23
Reconfig SECOND	21
PointPillars	53
Reconfig PP	47

Table 2: Results in different distance ranges on the nuScenes val benchmark, where the object distance from ego vehicle is denoted as d and *nuScenes range* refers to the official evaluation range

Method	d $<$ 20m		d $\geq$ 20m		nuScenes range	
	mAP	NDS	mAP	NDS	mAP	NDS
PointPillars	45.3	58.1	11.8	33.8	30.3	48.6
Reconfig PP (sing-res)	**48.8**	**60.3**	12.4	**38.2**	32.8	50.3
Reconfig PP (multi-res)	48.4	59.7	**12.6**	**38.2**	**32.9**	**50.5**

Table 3: Distance-based mAP by categories compared to PointPillars on the nuScenes test 3D detection benchmark. Here according to the average size of all the bounding boxes, we consider the first 5 categories (car, bus, truck, trailer and construction vehicle) as large objects while the last 5 categories (pedestrian, barrier, traffic cone, motorcycle and bicycle) as small objects. We compute the mAP of all the small objects and record it as mSAP in the table

Method	Car	Bus	Truck	Trail	CV	Ped	Bar	TC	Moto	Bicy	**mAP**	**mSAP**
PointPillars	74.4	38.5	23.4	36.1	4.8	60.1	30.5	19.8	12.9	0.1	30.1	24.7
Reconfig PP (sing-res)	75.6	38.5	26.5	**38.9**	**7.5**	**63.1**	34.4	23.8	**15.2**	0.1	32.4	27.3
Reconfig PP (multi-res)	**75.8**	**39.5**	**27.2**	38.0	6.5	62.5	**34.9**	**25.7**	**15.2**	**0.2**	**32.5**	**27.7**

5 Results

In this section, we first present the complexity analysis of our reconfiguration algorithm along with relevant experimental results. Then quantitative and qualitative results are given to show the performance improvement, especially the performance for small and distant objects. For fairness, all of the following experiments are conducted without further tuning network architecture and parameters or introducing more tricks.

5.1 Complexity Analysis

Firstly, let us briefly compare the complexity of vanilla voxelization and our improved version. Suppose there are N points and M voxels, the reconfiguration process only adds constant operations when traversing all points, as well as one-time traversal of voxels when performing random walk. So the complexity changes from $O(N)$ to $O(N+M)$. The more points each voxel contains, the greater the ratio $\frac{N}{M}$ is, then the effect on the efficiency of voxelization is more limited.

To indicate the influence of this representation on the inference speed more empirically, we validate it in KITTI experiments (Tab. 1). Our method hardly affects the algorithm efficiency and the inference speed is still much faster than point-based methods (about 10Hz of [5, 23, 32]) and can achieve real-time detection.

5.2 Quantitative Analysis

Toy experiments on KITTI First, we did a series of preliminary experiments on the KITTI dataset to investigate the effectiveness of our method under different settings. As shown in Fig. 1, taking the representative small object, pedestrian, as an example, we find that our method can consistently improve the detection performance when using different pillar or voxel resolutions. In addition, we also compare their performance at different distances in the experiments where minimum pillar or voxel resolution is adopted. As we expected, performance improvements become more evident as distance increases. See more detailed results in the supplementary materials.

Experiments on large-scale datasets Then we test our methods on large-scale datasets. Considering the large amount of data and the difficulty of training networks including SECOND, we only give the experimental results on PointPillars here. Due to higher ranked models on these two benchmarks typically adopt heavy heads, we validate the efficacy of our methods both on lightweight, real-time baselines (Fast PP) and those with higher performance (Heavy PP).

Firstly, in order to study the improvement details, we evaluate the detection performance of objects from different categories and distance ranges, where the latter is conducted on the validation set. Taking the Fast PP experiments as the example, from Tab. 3, it can be seen that mAPs of smaller objects are greatly improved, among which the multi-resolution version increases mAPs of pedestrian, barrier, traffic cone and motorcycle by 2.4%, 4.4%, 5.9% and 2.3% respectively. In addition, from Tab. 2, we can observe that in terms of distant object detection in the distance range over 20m, NDS is increased by up to 4.4%. Meanwhile, in the above two experiments, the detection performance of large and close objects is not affected, but most aspects are also improved. Finally, compared with

Table 4: Results on the nuScenes dataset

Method	Modality	mAP	NDS
MAIR [33]	RGB	30.4	38.4
Freespace [34]	LiDAR	35.0	41.9
PP [22]	LiDAR	30.5	45.3
SECOND [21]	LiDAR	31.6	46.8
SHAPNET [35]	LiDAR	32.4	48.4
3DSSD [36]	LiDAR	42.6	56.4
Painting [37]	LiDAR+RGB	46.4	58.1
CBGS [31]	LiDAR	**52.8**	**63.3**
Fast PP [22]	LiDAR	30.1	48.5
+Reconfig	LiDAR	32.4	50.2
+Multi-res	LiDAR	**32.5**	**50.6**
Heavy PP	LiDAR	43.4	54.1
+Reconfig	LiDAR	45.4	56.1
+Multi-res	LiDAR	**45.7**	**56.3**
Ours (Final)	LiDAR	48.5	59.0

Table 5: Results on the Lyft dataset

Team/Method	Reference	Modality	mAP-3D
STL-IV Lab	11st place	-	14.2
MIT HAN Lab	10th place	-	14.4
...	...	-	-
Wenjing (single model)	1st place	LiDAR	**17.9**
VoxelNet [7]	CVPR 2018	LiDAR	10.1
SECOND [21]	Sensors 2018	LiDAR	13.0
Fast PP [22]	CVPR 2019	LiDAR	10.4
+Reconfig	-	LiDAR	11.3
+Multi-res	-	LiDAR	**11.4**
Heavy PP	CVPR 2019	LiDAR	11.9
+Reconfig	-	LiDAR	12.7
+Multi-res	-	LiDAR	**12.9**
Larger range	CVPR 2019	LiDAR	16.0
+Reconfig	-	LiDAR	16.7
+Multi-res	-	LiDAR	**16.9**

Table 6: Ablation studies on the KITTI val 3D detection benchmark

DL	Sparse Reconfig	Dense Reconfig	Multi res	**mAP**	Car			Cyclist			Pedestrian		
					Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
×	×	×	×	66.76	88.31	77.79	75.91	77.07	59.95	58.96	59.78	53.22	49.88
√	×	×	×	67.07	88.39	77.90	75.92	75.17	58.94	57.49	61.39	57.15	51.31
√	√	×	×	67.08	88.17	77.38	75.56	77.48	59.94	58.02	60.64	56.33	50.22
√	×	√	×	67.40	88.14	77.75	76.03	76.36	60.03	57.71	61.70	57.71	51.13
√	√	√	×	68.36	**88.88**	78.09	76.13	79.63	61.87	59.26	**61.97**	**57.77**	**51.63**
√	√	√	√	**68.41**	88.65	**78.22**	**76.21**	**80.50**	**65.82**	**60.24**	61.63	54.08	50.33

baseline models, our method can respectively improve 2.4% mAP, 2.1% NDS and 2.3% mAP, 2.2% NDS on top of Fast PP and Heavy PP. With further training steps and adding more data augmentation (without model ensemble), we achieve 48.5% mAP and 59.0% NDS in our final model, which is comparable with ensembled top entries [31] and outperforms all the published methods.

In addition to nuScenes, we also tested on Lyft benchmark as Tab. 5 shows, where the *Larger range* refers to the change of x,y range both from [-49.6, 49.6] to [-89.6, 89.6] on the basis of Heavy PP. Our final model can consistently achieve about 1.0% mAP increase for all 3 baselines under the more difficult mAP-3D metric. Furthermore, this improvement is mainly achieved by the enhanced detection performance of small and distant objects, which are only a minority of all the objects. Detailed analysis of Lyft results can be referred to supplementary materials.

Ablation studies Finally, we take SECOND experiments on KITTI as the example to give more detailed ablation studies. In the experiments, we controlled whether to add 4 neighbor voxels (Dilated, abbrev. DL in Tab. 6), whether to reconfigure sparse voxels, whether to reconfigure dense voxels, whether in different resolutions, and carried out the corresponding experiments. Here *dense voxels* means that they contain the maximum number of points while *sparse* indicates otherwise, and *DL* corresponds to the case with the same framework but without neighbor voxels reconfiguration (see the comparison in Fig. 1). It turned out that the improvement of detecting larger objects like car is slight but stable. On cyclist and pedestrian, almost all of our models are better than the baseline model, which shows the necessity of improving the representation. Especially for cyclist, our best model can achieve better mAPs on the easy, moderate and hard sets by 3.43%, 5.87% and 1.28% increase respectively. Most importantly, comparison with the dilated voxels (DL) based on the original voxel neighbor layout shows the effectiveness of our reconfiguration mechanism.

5.3 Qualitative Analysis

We visualize some samples to show the results of voxel reconfiguration (Fig. 4). It can be seen that with the help of our mechanism, neighbor voxels move to regions with more points and implicitly follow surface and shape of objects as well. We thus believe that voxel encoder can benefit a lot from this more reasonable spatial quantization. See the supplemental materials for qualitative analysis of detection results on nuScenes.

6 Conclusion

In this paper, we propose *Reconfigurable Voxels*, a novel representation that can significantly improve the imbalance of sampling points in different voxels caused by sparsity and irregularity of LiDAR point cloud. We demonstrate that on various 3D detection benchmarks, incorporating this lightweight representation into the state-of-the-art voxel-based frameworks can greatly enhance the performance in terms of small and distant objects without much computation overhead. Future work includes designing this mechanism more carefully and figuring out this problem in point-based and multi-sensor fusion methods.

Acknowledgments

This work is partially supported by the SenseTime Collaborative Grant on Large-scale Multi-modality Analysis (CUHK Agreement No. TS1610626 & No. TS1712093), the General Research Fund (GRF) of Hong Kong (No. 14236516 & No. 14203518).

References

[1] M. Himmelsbach, A. Müller, T. Lüttel, and H.-J. Wünsche. Lidar-based 3d object perception. In *Proceedings of 1st international workshop on cognition for technical systems*, 2008.

[2] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[3] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[4] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. Pointcnn: Convolution on x-transformed points. In *Conference on Neural Information Processing Systems*, 2018.

[5] S. Shi, X. Wang, and H. Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[6] G. Riegler, A. O. Ulusoy, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[7] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[8] X. Zhu, Y. Ma, T. Wang, Y. Xu, J. Shi, and D. Lin. Ssn: Shape signature networks for multi-classobject detection from point clouds. In *Proceedings of the European Conference on Computer Vision*, 2020.

[9] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuscenes: A multimodal dataset for autonomous driving. *CoRR*, abs/1903.11027, 2019. URL `http://arxiv.org/abs/1903.11027`.

[10] R. Kesten, M. Usman, J. Houston, T. Pandya, K. Nadhamuni, A. Ferreira, M. Yuan, B. Low, A. Jain, P. Ondruska, S. Omari, S. Shah, A. Kulkarni, A. Kazakova, C. Tao, L. Platinsky, W. Jiang, and V. Shet. Lyft level 5 av dataset 2019. `https://level5.lyft.com/dataset/`, 2019.

[11] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[12] D. Lin, S. Fidler, and R. Urtasun. Holistic scene understanding for 3d object detection with rgbd cameras. In *IEEE International Conference on Computer Vision*, pages 1417–1424, Dec 2013. doi:10.1109/ICCV.2013.179.

[13] S. Song and J. Xiao. Sliding shapes for 3d object detection in depth images. In *European Conference on Computer Vision*, pages 634–651, 2014.

[14] C. R. Qi, O. Litany, K. He, and L. J. Guibas. Deep hough voting for 3d object detection in point clouds. In *IEEE International Conference on Computer Vision*, 2019.

[15] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell, and K. Q. Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[16] P. Li, X. Chen, and S. Shen. Stereo r-cnn based 3d object detection for autonomous driving. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[17] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[18] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. Waslander. Joint 3d proposal generation and object detection from view aggregation. In *International Conference on Intelligent Robots and Systems*, 2018.

[19] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[20] B. Yang, W. Luo, and R. Urtasun. Pixor: Real-time 3d object detection from point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[21] Y. Yan, Y. Mao, and B. Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18 (10), 2018.

[22] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[23] Z. Yang, Y. Sun, S. Liu, X. Shen, and J. Jia. Std: Sparse-to-dense 3d object detector for point cloud. In *IEEE International Conference on Computer Vision*, 2019.

[24] Y. Zhou, P. Sun, Y. Zhang, D. Anguelov, J. Gao, T. Ouyang, J. Guo, J. Ngiam, and V. Vasudevan. End-to-end multi-view fusion for 3d object detection in lidar point clouds. In *Conference on Robot Learning*, 2019.

[25] M. Ye, S. Xu, and T. Cao. Hvnet: Hybrid voxel network for lidar based 3d object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[26] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *International Conference on Intelligent Robots and Systems*, 2015.

[27] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong. O-cnn: Octree-based convolutional neural networks. *ACM Transactions on Graphics (SIGGRAPH)*, 36(4), 2017.

[28] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. Deformable convolutional networks. In *IEEE International Conference on Computer Vision*, 2017.

[29] H. Gao, X. Zhu, S. Lin, and J. Dai. Deformable kernels: Adapting effective receptive fields for object deformation. *CoRR*, abs/1910.02940, 2019. URL `http://arxiv.org/abs/1910.02940`.

[30] Y. Xiong, M. Ren, R. Liao, K. Wong, and R. Urtasun. Deformable filter convolution for point cloud reasoning. *CoRR*, abs/1907.13079, 2019. URL `http://arxiv.org/abs/1907.13079`.

[31] B. Zhu, Z. Jiang, X. Zhou, Z. Li, and G. Yu. Class-balanced grouping and sampling for point cloud 3d object detection. *CoRR*, abs/1908.09492, 2019. URL `http://arxiv.org/abs/1908.09492`.

[32] Y. Chen, S. Liu, X. Shen, and J. Jia. Fast point r-cnn. In *IEEE International Conference on Computer Vision*, 2019.

[33] A. Simonelli, S. R. Bulò, L. Porzi, M. López-Antequera, and P. Kontschieder. Disentangling monocular 3d object detection. In *IEEE International Conference on Computer Vision*, 2019.

[34] P. Hu, J. Ziglar, D. Held, and D. Ramanan. What you see is what you get: Exploiting visibility for 3d object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[35] Y. Ye, H. Chen, C. Zhang, X. Hao, and Z. Zhang. Sarpnet: Shape attention regional proposal network for lidar-based 3d object detection. *Neurocomputing*, 379:53 – 63, 2020. ISSN 0925-2312.

[36] Z. Yang, Y. Sun, S. Liu, and Jia. 3dssd: Point-based 3d single stage object detector. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[37] S. Vora, A. H. Lang, B. Helou, and O. Beijbom. Pointpainting: Sequential fusion for 3d object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2020.