Supplementary Material for "CurveCloudNet: Processing Point Clouds with 1D Structure"

Colton Stearns
Stanford University

Alex Fu
Stanford University

Jiateng Liu
UIUC

Jeong Joon Park
University of Michigan

Davis Rempe
NVIDIA

Despoina Paschalidou
Stanford University

Leonidas J. Guibas
Stanford University

1. Overview

In this document, we provide additional method details, dataset details, implementation details, experimental analysis, and qualitative results. In Sec. 2, we concretely outline how we convert a point cloud into a curve cloud, analyze our 1D farthest point sampling, and discuss our modifications to point-based operations. In Sec. 3, we provide a detailed overview of the Kortx software system and dataset, our ShapeNet simulator, and the A2D2 dataset. In Sec. 4, we discuss implementation details of CurveCloudNet and baselines that are not covered in the main paper. In Sec. 5, we report results on a translated A2D2 experiment, object classification, and the nuScenes test split. Finally, in Sec. 6, we conclude with additional limitations.

2. Additional Method Details

2.1. Constructing Curve Clouds

Curve Cloud Conversion.

Constructing Curve Cloud. We refer the reader to Sec. 3.1 of the main paper for an overview of constructing curve clouds. As input, we assume that a laser-based 3D sensor outputs a point cloud $P = \{p_1, ..., p_N\}$ where $p_i = [x_i, y_i, z_i] \in \mathbb{R}^3$, an acquisition timestamp $t_i \in \mathbb{R}$ for each point, and an integer laser-beam ID $b_i \in [1, B]$ for each point. We wish to convert the input into a curve cloud $C = \{c_1, ..., c_M\}$, where a curve $c = [p_i, ..., p_{i+K}]$ is defined as a sequence of K points where consecutive point pairs are connected by a line segment, i.e., a *polyline*. As outlined in Fig. 1, we first group points by their laser beam ID and sort points based on their acquisition timesteps, resulting in an ordered sequence of points that reflects a single beam's traversal through the scene. Next, for each sequence, we compute the distances between pairs of consecutive points (denoted as polyline "edge lengths"). Finally, we split the sequence whenever an edge length is greater than a threshold δ, resulting in many variable-length polyline "curves". In practice, we parallelize the conversion across all points, and on the large-scale nuScenes dataset, the algorithm runs at 1500Hz.

We select a threshold δ that reflects the sensor specifications and scanning environment. In particular, the threshold is conservatively set to approximately $10\times$ the *median distance* between consecutively scanned points one meter away from the sensor. On the A2D2 dataset [11], we set $\delta = [0.1, 0.17, 0.1, 0.12, 0.1]$ for the five LiDARs, and on

```
'''
Inputs: P, T, B, delta
    P: array of size (N, 3) with xyz coordinates
    T: array of size (N,) with timesteps
    B: array of size (N,) with beam IDs
Outputs: curves
    curves: list of arrays, array j is size (N_j, 3)
'''
curves = []
for b in unique(B):
    # filter to a single laser beam's measurments
    beam_P, beam_T = P[beams==b], T[beams==b]

    # order points by laser's traversal
    sequential_ordering = argsort(beam_T)
    beam_P = beam_P[sequential_ordering]

    # split laser's traversal into cont. curves
    edge_lens = norm(beam_P[1:] - beam_P[:-1])
    split_locations = edge_lens > delta

    # convert into polylines
    beam_C = split_seq(beam_P, split_locations)
    curves += beam_C
```

Figure 1. *Point to Curve Cloud Conversion.* Algorithm (in Python) to convert an input point cloud into a set of polylines.

the nuScenes dataset [5] and KITTI dataset [2, 10] we set $\delta = 0.08$. Additionally, on the A2D2, nuScenes, and KITTI datasets, we scale δ proportional to the square root of the distance from the sensor, since point samples becomes sparser at greater distance. On the object-level ShapeNet dataset [6], we set $\delta = 0.01$. Experimentally, we observed that CurveCloudNet is flexible across different δ values.

Kortx Curve Representation. The Kortx vision system directly generates and operates on 3D curves sampled from a triangulated system of event-based sensors and laser scanners. As the detected laser reflection traverses the scene, it produces a frameless 4D data stream that enables low latency, low processing requirements, and high angular resolution. 3D curves are an intrinsic component of the Kortx perception system, and the system directly outputs a curve cloud without the need for additional data processing.

2.2. Additional Details on Curve Operations

1D Farthest Point Sampling. We refer the reader to Sec. 3.2.2 of the main paper for an overview of our 1D farthest point sampling (FPS) algorithm. The goal of this algorithm is to *efficiently* sample a subset of points on each curve such that consecutive points will be approximately ϵ apart along the downsampled curve. Concretely, for a curve c with K points, $c = [p_i, ..., p_{i+K}]$, we first compute the $K-1$ "edge lengths", $e = [d_i, ..., d_{i+K-1}]$, where d_i is the distance between consecutive points (p_i, p_{i+1}). Next, we estimate the geodesic distance along the curve via a cumulative sum operation on the edge lengths: $g = \texttt{CUMSUM}(e)$. Then, we divide the geodesic distances by our desired spacing, ϵ, and take the floor, resulting in ϵ-spaced intervals $I = \texttt{FLOOR}(g/\epsilon)$. We output the first point in each interval, resulting in L ϵ-spaced subsampled points $\{q_1, ..., q_L\}$.

For each curve, the computational complexity of the 1D FPS algorithm is $O(K)$. Extending this to all curves, the computational complexity is $O(N)$. When parallelized on a GPU, for each curve, the 1D FPS algorithm has a has *parallel* complexity of $O(\log K)$, where the CUMSUM operation is the parallelization bottleneck (see **prefix sum** algorithms for more information [4]). The algorithm trivially parallelizes across curves, leading to a total parallel complexity of $O(\log K)$. In contrast, Euclidean farthest point sampling has a comptational complexity of $O(N^2)$ or $O(Nlog^2N)$ (depending on whether a KD-tree is used), and a parallel complexity of $O(L)$. When L is large (i.e. we are subsampling a large number of points), Euclidean FPS is a significant performance bottleneck.

2.3. Additional Details on Point Operations

Set Abstraction (SA). CurveCloudNet uses a series of set abstraction layers from PointNet++ [17], and we follow previous works to improve the set abstraction layer. First, we perform relative position normalization [18] – given a centroid point p_i with local neighborhood points $\mathcal{N}_i$, we center the neighborhood about p_i **and** we divide by r to normalize the relative positions. Additionally, we opt for the attentive pooling from RandLANet [12] instead of max pooling.

Graph Convolution. CurveCloudNet uses a series of graph convolutions, which are modeled after the edge convolution from DGCNN [20]. Unlike DGCNN however, we construct the K-Nearest-Neighbor graph based on 3D point distances instead of feature distances – this permits more efficient neighborhood construction, irrespective of feature size. Furthermore, we use attentive pooling from RandLANet [12] instead of max pooling.

Figure 2. *Kortx Dataset Objects.* Our Kortx dataset contains scans of 7 real-world objects. We visualize one aggregated "scan" per object from a single viewpoint.

3. Additional Dataset Details

3.1. Kortx Perception System and Dataset

Kortx Perception System. Kortx is a perception software system developed by Summer Robotics [1]. It is an active-light, multi-view stereoscopic system using one or more scanning lasers. It can be configured to use two or more event-based vision sensors to build up arbitrary capture volumes. Event-based vision sensors are used to detect the scanning laser reflection from target surfaces. Event-based sensors are well suited to this setup as their readout electronics are event triggered instead of time triggered. Furthermore, the Kortx System supports arbitrary continuous scan patterns, allowing a user to create their own patterns and use their own scan hardware. For more information, please visit the Summer Robotics Website.

Kortx Dataset. Using Kortx, we scanned 7 real-world objects: *cap, chair, earphone, knife, mug-1, mug-2* and *rocket* (see Fig. 2). Each object was scanned multiple times in different poses, resulting in 39 total scans (summarized in Tab. 1). Because the Kortx platform provides a continuous

Object	Total	Cap	Chair	Earphone	Knife	Mug	Rocket
Instances	7	1	1	1	1	2	1
Scans	39	6	6	4	6	12	5
Frames	195	30	30	20	30	60	25

Table 1. *Kortx Dataset Statistics.* "Instance" is a unique 3D object. "Scan" is a dense object scan from a single viewpoint. "Frame" is a single frame within the 20Hz stream of the dense scan.

event-based 3D scan output (points are sampled every 5µs), we defined a "frame" as a batch of 2048 consecutive point measurements, corresponding to roughly a 20Hz frame rate. Because each frame differs in its dynamic scanning pattern, we evaluate on 5 consecutive frames per scan in our Kortx dataset, hence resulting in 195 point clouds in total. We manually labeled scanned points with the semantic part categories defined in the ShapeNet Part Segmentation Benchmark [6]. Each Kortx scan is mean-centered, however it is *not* aligned into a canonical pose, resulting in an object's orientation depending on the sensor's reference frame.

3.2. ShapeNet Simulator

We simulate laser-based 3D capture on the ShapeNet Dataset [6]. For each mesh, we randomly sample a sensor pose on the unit sphere and render the mesh's depth values into a 2048×2048 image. Next, we sample 2D lines on the depth image that correspond to a laser's traversal. For the *random* sampling pattern used in the Kortx evaluation (see Sec 4.1 of the main paper) and the ShapeNet Classification evaluation (see Sec. 5.2), we select random linear traversals in the image plane, with each traversal parameterized by a pixel coordinate (i, j) and direction $\theta \in [0, \pi)$. For the *grid* and *parallel* sampling patterns used in our ShapeNet Segmentation evaluation (see Sec 4.1 of the main paper), we sample evenly-spaced vertical and horizontal lines. To reduce descritization artifacts introduced from the rasterization, we query every 6th pixel along each line for the Kortx segmentation task and every 4th pixel for the ShapeNet segmentation and classification tasks. We repeatedly generate synthetic laser traversals until we have sampled 2048 points from the mesh. Fig. 1 of the main paper shows an example of the *random*, *grid*, and *parallel* sampling patterns used in our ShapeNet Segmentation evaluation. Fig. 3 provides an additional qualitative illustration of the three sampling patterns, but showing 4096 points per scan for greater visual clarity.

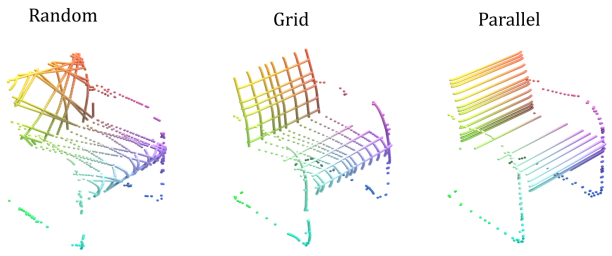

Figure 3. *ShapeNet Simulator.* Our ShapeNet laser-based 3D capture simulator can produce different types of sampling patterns.

3.3. A2D2 LiDAR Segmentation

The Audi Autonomous Driving Dataset (A2D2) [11] contains 41,280 frames of labeled outdoor driving scenes captured in three cities. The vehicle is equipped with five LiDAR sensors, each mounted on a different part of the vehicle and with a different orientation, resulting in a unique grid-like scanning pattern. The A2D2 data was captured in urban, highway, and rural environments as well as in different weather conditions. At the time of writing, the A2D2 dataset only contains semantic labels for the front-facing camera. Thus, we evaluate on LiDAR observations within the front-facing camera's field of view, and we map camera categories to LiDAR categories. We will release the code detailing the exact mapping.

3.4. Discussion on 3D Datasets

As LiDAR and other 3D scanning technologies continue to develop, they are being applied to new and diverse applications, including open-world robotics (i.e. embedded agents), city planning, agriculture, mining, and more. Additionally, there is an increasing variety of sensors and sensor configurations, spanning hardware that scans at different point densities, different ranges, and with unique (or controllable) scanning patterns. The A2D2 and Kortx datasets are two recent examples of such a trend. We believe an important future direction will be to develop a 3D backbone that is performant in *all* these settings. Furthermore, we believe it is important to understand *which* settings "break" previous assumptions such as the range-view projection, the birds-eye-view projection, and spherical attention. While CurveCloudNet is a first step towards this goal, we believe it will be important to capture and compile new 3D datasets, and to evaluate on a greater diversity of environments.

4. Additional Implementation Details

4.1. Baselines

PointNet++ and DGCNN. We train and evaluate PointNet++ [17] and DGCNN [21] using the reproduced implemenations from Pytorch Geometric [9][1]. For PointNet++, we run hyperparameter sweeps to tune the radius and downsampling ratio on each dataset. For DGCNN, we use the authors' reported hyperparameters.

[1] https://github.com/pyg-team/pytorch_geometric/

RandLANet. We train and evaluate RandLANet [12] using the reproduced implementation from Open3D-ML [26][2]. We additionally improve the latency by incorporating GPU-implementations for point grouping and sampling from PyTorch3D [19][3]. We use the authors' reported hyperparameters.

CurveNet. We train and evaluate CurveNet [22] using the authors' official implementation[4]. We use the authors' reported hyperparameters for all datasets.

PointMLP. We train and evaluate PointMLP [16] using the authors' official implementation[5]. We use the reported hyperparameters for all datasets.

PointNext. We train and evaluate PointNext [18] using the authors' official implementation[6]. As outlined by the authors, we use PointNext-Small for the ShapeNet and KortX datasets. On the A2D2, nuScenes, and KITTI datasets, we use the larger PointNext-XL. Because the authors indicate the importance of the network "radius", we additionally performed a hyperparameter sweep to find the best radius of 0.05 for the A2D2, nuScenes, and KITTI datasets.

MinkowskiNet. We train and evaluate MinkowskiNet [8] using the authors' official implementation[7]. We use the larger MinkUNet-34A for all experiments. We use an initial voxel size of 0.05 on outdoor datasets and 0.015 on object-level datasets.

Cylinder3D. We train and evaluate Cylinder3D [25] using the authors' official implementation[8]. We use the reported hyperparameters on the nuScenes and KITTI datasets. On the A2D2 dataset, we set the cylindrical voxel grid to cover a $\pm 31^\circ$ forward-facing azimuth with a maximum radius of 80 meters and a height covering $[-5, 20]$ meters; we define the initial grid to have 360 radial partitions, 120 angular partitions, and 120 height partitions. On the ShapeNet and KortX datasets, we set the voxel grid to cover all 360° with a radius of 1.0 and height of 1.0; to address latency and memory constraints, we define the initial grid to have 96 radial partitions, 96 angular partitions, and 96 height partitions.

SphereFormer. We train and evaluate SphereFormer [14] using the authors' official implementation[9]. We use the reported hyperparameters for the nuScenes and KITTI datasets, and we use the reported nuScenes hyperparameters for the A2D2 dataset. For the KortX and ShapeNet datasets, we also use the reported hyperparameters, and we reduce the voxel size from 0.1 to 0.015 to account for the dataset's smaller 3D scale. On the KortX and ShapeNet datasets, we additionally ran a sweep on different voxel sizes and spherical window sizes, but observed limited differences.

[2] https://github.com/isl-org/Open3D-ML
[3] https://github.com/facebookresearch/pytorch3d
[4] https://github.com/tiangexiang/CurveNet
[5] https://github.com/ma-xu/pointMLP-pytorch
[6] https://github.com/guochengqian/PointNeXt
[7] https://github.com/NVIDIA/MinkowskiEngine
[8] https://github.com/xinge008/Cylinder3D
[9] https://github.com/dvlab-research/SphereFormer

4.2. Training Strategy

We train CurveCloudNet and baselines on segmentation tasks with a standard cross-entropy loss. Following previous works, we also supplement the loss with a Lovasz loss [3, 25] for the nuScenes, A2D2, and KITTI datasets. At training, we apply random scaling and translation augmentations, as well as random flips on the nuScenes, A2D2, and KITTI datasets. Importantly, we use an ***identical*** training strategy for CurveCloudNet and each baseline. We experimentally observe convergence in all models' validation accuracies by the end of training.

Object Part Segmentation. We train CurveCloudNet and all baselines for 60 epochs in the KortX experiment and 120 epochs in the ShapeNet experiment with the Adam optimizer [13], a learning rate of $1e^{-4}$, batch momentum decay of 0.97, and exponential learning rate decay of 0.97. For all models, except for Cylinder3D, we use a batch size of 24. For Cylinder3D, we use a batch size of 12 because 24 exceeds our GPU memory capacity.

A2D2 LiDAR Segmentation. We train CurveCloudNet and all baselines for 140 epochs with the Adam optimizer, a batch size of 7, a learning rate of $1e^{-3}$, and an exponential learning rate decay of 0.97.

nuScenes and KITTI LiDAR Segmentation. We train CurveCloudNet and all baselines for 100 epochs with the Adam optimizer, a batch size of 4 on nuScenes and 2 on KITTI, a learning rate of $1e^{-3}$, and an exponential learning rate decay of 0.97. At test time, we follow previous works [15, 25] and average model predictions over axis-flipping and scaling augmentations.

5. Additional Experiments

5.1. Translated A2D2

Overview. In Sec. 4.2 of the main paper, we reported that SphereFormer outperforms CurveCloudNet by $+1.0\%$ mIOU on the A2D2 dataset. In this section, we show that, on the same data, SphereFormer underperforms CurveCloudNet when the scene does not exhibit consistent and aligned global structure.

We apply a simple translation augmentation to the A2D2 training and validation data – for each scan, we offset all points by a translation sampled from a uniform Gaussian with $\mu = 0$ and $\sigma = 20$. Note that this removes the *global* alignment of point clouds, but completely preserves all *local* structure. In the real world, this setup could occur in topography or mapping, i.e. when a large region is scanned but only one area is of interest (which could be anywhere in the scan). We train and evaluate all models with an identical

Method	mIoU (↑)	Per-Class mIoU (↑)											
		car	bicycle	truck	person	road	sidewalk	obstacle	building	nature	pole	sign	signal
MinkowskiNet [8]	42.7	54.7	3.1	60.0	7.4	87.3	54.8	9.7	71.8	76.7	14.3	31.5	42.4
SphereFormer [14]	40.5	52.7	1.1	62.4	3.8	85.9	53.5	10.9	70.5	76.3	9.6	29.1	29.9
CurveCloudNet	**44.8**	58.4	2.2	59.8	6.7	89.9	58.5	11.1	77.8	83.3	13.1	35.6	42.2

Table 2. *Translated A2D2 Results.* When A2D2 scans are randomly translated, CurveCloudNet significantly outperforms SphereFormer, suggesting that SphereFormer relies on a dataset exhibiting a highly consistent global structure.

setup to the original A2D2 experiment.

Results. We summarize results on the translated A2D2 experiment in Tab. 2. Without a consistent global alignment of the scene layout, all methods perform worse. However, CurveCloudNet is less effected and outperforms SphereFormer by over 4%. This further suggests that SphereFormer's radial window is tailored for outdoor driving scenes and cannot be flexibly applied to environments with weaker global structure. In contrast, CurveCloudNet can successfully leverage local structures to reason in more diverse environments.

5.2. ShapeNet Classification

In addition to semantic segmentation tasks, we also evaluate CurveCloudNet's performance in shape classification.

ShapeNet Classification Dataset. We use the ShapeNet Part Segmentation Benchmark [6, 23] as described in Sec 4.1 of the main paper. In the classification setting, the network is tasked with classifying a point cloud into one of the 16 object categories. Using our ShapeNet laser-scanner simulator (see Sec. 3.2), we generate a single synthetic "scan" for each ShapeNet mesh from a *fixed* sensor viewpoint, resulting in scanned objects sharing a canonical orientation. Following the official training and validation splits [23], this yields 12139 training point clouds and 1872 validation point clouds. For this experiment, we consider the *random* curve sampling pattern (see Sec. 3.2).

Setup. We train CurveCloudNet and several baselines on the simulated ShapeNet training set. Similar to the settings used for the segmentation task, all models are trained for 120 epochs with the Adam optimizer, a batch size of 24, a learning rate of $3e^{-4}$, batch momentum decay of 0.97, and exponential learning rate decay of 0.97. We record the best validation class-averaged accuracy, instance-averaged accuracy, and class-averaged F1 score that is achieved during training. We report means and standard deviations across 3 runs.

Results. Results are summarized in Tab. 3. CurveCloudNet outperforms the baselines on all three metrics. Additionally, CurveCloudNet exhibits improved latency and lower GPU memory compared to PointNet++, DGCNN, and PointMLP.

5.3. Additional nuScenes Results

We provide qualitative results on the nuScenes validation split in Fig. 4. For the corresponding evaluation on the validation split, see Sec. 4.3 of the main paper.

5.3.1 nuScenes Test Split

We evaluate our model on the test split of the nuScenes LiDAR segmentation task, and compare to top-performing baselines from the academic literature. We summarize our results in Tab. 4. CurveCloudNet outperforms many sparse-voxel methods in both class-averaged and frequency-weighted mIOU, such as Cylinder3D [25], SPVNAS [15], and AF^2-S3Net [7].

5.4. Additional KortX Results

We provide additional qualitative results on the KortX dataset in Fig. 5. We observe that CurveCloudNet distinguishes finegrained structures, such as the legs and back of the chair, the handle of the mug, the boundary where the nose of the rocket begins, and the brim of the cap.

5.5. Additional A2D2 Results

We provide additional qualitative results on the A2D2 dataset in Fig. 6. In many examples, CurveCloudNet distinguishes the sidewalk and the road much better than Cylinder3D. Furthermore, in contrast to CurveCloudNet, examples show that Cylinder3D can fail to detect pedestrians, swaps the "truck" and "car" categories, and swaps the "building" and "sign" categories.

Method	Accuracy			Performance		
	Class (↑)	Instance (↑)	F1 (↑)	Time (ms) (↓)	GPU (GB) (↓)	Param (M)
PointNet++ [17]	95.3 ± 0.7	99.0 ± 0.05	95.5 ± 0.6	51	0.91	1.6
DGCNN [20]	93.7 ± 0.5	98.9 ± 0.03	93.6 ± 0.5	73	0.78	0.6
PointMLP [16]	94.8 ± 1.3	99.2 ± 0.05	95.3 ± 1.0	54	0.76	13.2
CurveCloudNet	**96.3** ± **0.4**	**99.3** ± **0.04**	**96.0** ± **0.5**	37	0.66	10.3

Table 3. *Object Classification Results.* Mean class-averaged accuracy (Class), instance-averaged accuracy (Instance), and class-averaged F1 score (F1) are reported for the ShapeNet data. CurveCloudNet outperforms baselines on all metrics. Performance is on an Nvidia RTX 3090 GPU (batch size 16).

Method	Type	mIoU (↑)	fwIoU (↑)	Per-Class mIoU (↑)															
				barrier	*bicycle*	*bus*	*car*	*construction*	*motorcycle*	*pedestrian*	*traffic cone*	*trailer*	*truck*	*driveable*	*other flat*	*sidewalk*	*terrain*	*manmade*	*vegetation*
PolarNet [24]	Voxel	69.4	87.4	72.2	16.8	77.0	86.5	55.1	69.7	64.8	54.1	69.7	63.5	96.6	67.1	77.7	72.1	87.1	84.5
Cylinder3D [25]		77.2	89.9	82.8	29.8	84.3	89.4	63.0	79.3	77.2	73.4	84.6	69.1	97.7	70.2	80.3	75.5	90.4	87.6
SPVNAS [15]		77.4	89.7	80.0	30.0	91.9	90.8	64.7	79.0	75.6	70.9	81.0	74.6	97.4	69.2	80.0	76.1	89.3	87.1
AF^2-S3Net [7]		78.3	88.5	78.9	52.2	89.9	84.2	77.4	74.3	77.3	72.0	83.9	73.8	97.1	66.5	77.5	74.0	87.7	86.8
SphereFormer [14]		**81.9**	**91.7**	83.3	39.2	94.7	92.5	77.5	84.2	84.4	79.1	88.4	78.3	97.9	69.0	81.5	77.2	93.4	90.2
CurveCloudNet	Curve	78.5	90.4	81.7	38.4	93.8	90.3	70.1	77.3	75.1	69.5	84.9	74.1	97.5	67.8	79.7	75.7	91.7	88.7

Table 4. *nuScenes Test Split.* CurveCloudNet demonstrates improved or competitive performance with top-performing baselines.

6. Additional Discussion

Our intuition suggests that when a point cloud exhibits 1D curve structures, it is usually advantageous to make use of the structure. However, as we continue using CurveCloudNet in new environments, limitations arise. For instance, it can be challenging to dynamically identify *when* curve structure exists and *the extent* to which it exists. While the datasets we evaluated on exhibit clear curve structure, curve structure can diminish in settings such as far-away scanning, superimposing large volumes of scans, or gradually acquiring 3D measurements. In such cases, we speculate that curve modules offer little to no improvement, and that CurveCloudNet reduces to a point cloud backbone; however, additional follow-up analysis is needed.

References

[1] Summer robotics. https://www.summerrobotics.ai/. Accessed: 2023-02-15. 2

[2] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019. 2

[3] Maxim Berman, Amal Rannen Triki, and Matthew B. Blaschko. The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018. 4

[4] Guy E. Blelloch. Prefix sums and their applications. 1990. 2

[5] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. 2020. 2

[6] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 2, 3, 5

[7] Ran Cheng, Ryan Razani, Ehsan Moeen Taghavi, Enxu Li, and Bingbing Liu. (af)2-s3net: Attentive feature fusion with adaptive feature selection for sparse semantic segmentation network. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12542–12551, 2021. 5, 6

[8] Christopher B. Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019. 4, 5

[9] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. 3

[10] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012. 2

[11] Jakob Geyer, Yohannes Kassahun, Mentar Mahmudi, Xavier Ricou, Rupesh Durgesh, Andrew S. Chung, Lorenz Hauswald, Viet Hoang Pham, Maximilian Mühlegg, Sebas-

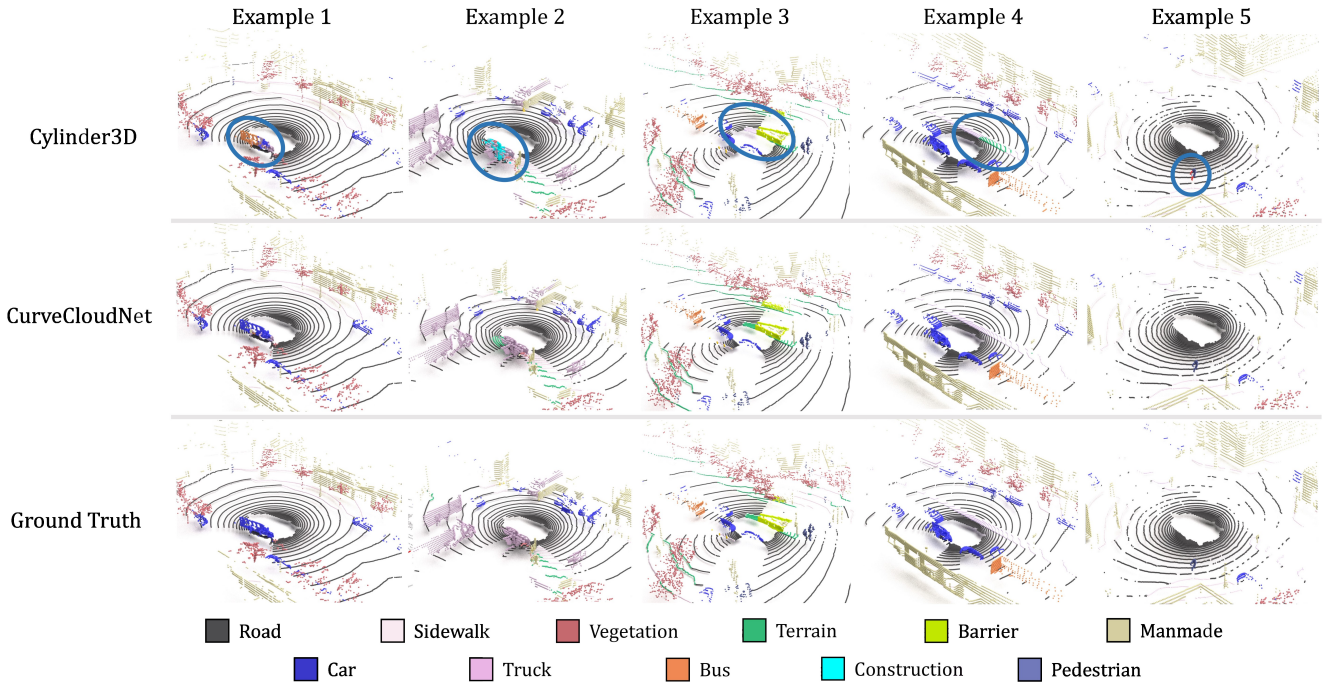

Figure 4. *nuScenes Segmentation.* We visualize semantic segmentation predictions of CurveCloudNet and Cylinder3D on the nuScenes validation split. We highlight regions where Cylinder3D errors and CurveCloudNet predicts correctly.

tian Dorn, Tiffany Fernandez, Martin Jänicke, Sudesh Mirashi, Chiragkumar Savani, Martin Sturm, Oleksandr Vorobiov, Martin Oelker, Sebastian Garreis, and Peter Schuberth. A2D2: audi autonomous driving dataset. *arXiv.org*, 2020. 1, 3

[12] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020. 2, 4

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2015. 4

[14] Xin Lai, Yukang Chen, Fanbin Lu, Jianhui Liu, and Jiaya Jia. Spherical transformer for lidar-based 3d recognition. In *CVPR*, 2023. 4, 5, 6

[15] Zhijian Liu, Haotian Tang, Shengyu Zhao, Kevin Shao, and Song Han. Pvnas: 3d neural architecture search with point-voxel convolution. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44:8552–8568, 2021. 4, 5, 6

[16] Xu Ma, Can Qin, Haoxuan You, Haoxi Ran, and Yun Fu. Rethinking network design and local geometry in point cloud: A simple residual mlp framework. In *Proc. of the International Conf. on Learning Representations (ICLR)*, 2022. 4, 6

[17] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems (NIPS)*, 2017. 2, 3, 6

[18] Gordon Guocheng Qian, Yuchen Li, Houwen Peng, Jinjie Mai, Hasan Abed Al Kader Hammoud, Mohamed Elhoseiny, and Bernard Ghanem. Pointnext: Revisiting pointnet++ with improved training and scaling strategies. *ArXiv*, abs/2206.04670, 2022. 2, 4

[19] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv:2007.08501*, 2020. 4

[20] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph CNN for learning on point clouds. *ACM Trans. on Graphics*, 2019. 2, 6

[21] Ziyan Wang, Buyu Liu, Samuel Schulter, and Manmohan Chandraker. A parametric top-view representation of complex road scenes. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019. 3

[22] Tiange Xiang, Chaoyi Zhang, Yang Song, Jianhui Yu, and Weidong Cai. Walk in the cloud: Learning curves for point clouds shape analysis. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2021. 4

[23] Li Yi, Vladimir G Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics (ToG)*, 2016. 5

[24] Yang Zhang, Zixiang Zhou, Philip David, Xiangyu Yue, Zerong Xi, Boqing Gong, and Hassan Foroosh. Polarnet: An improved grid representation for online lidar point clouds se-

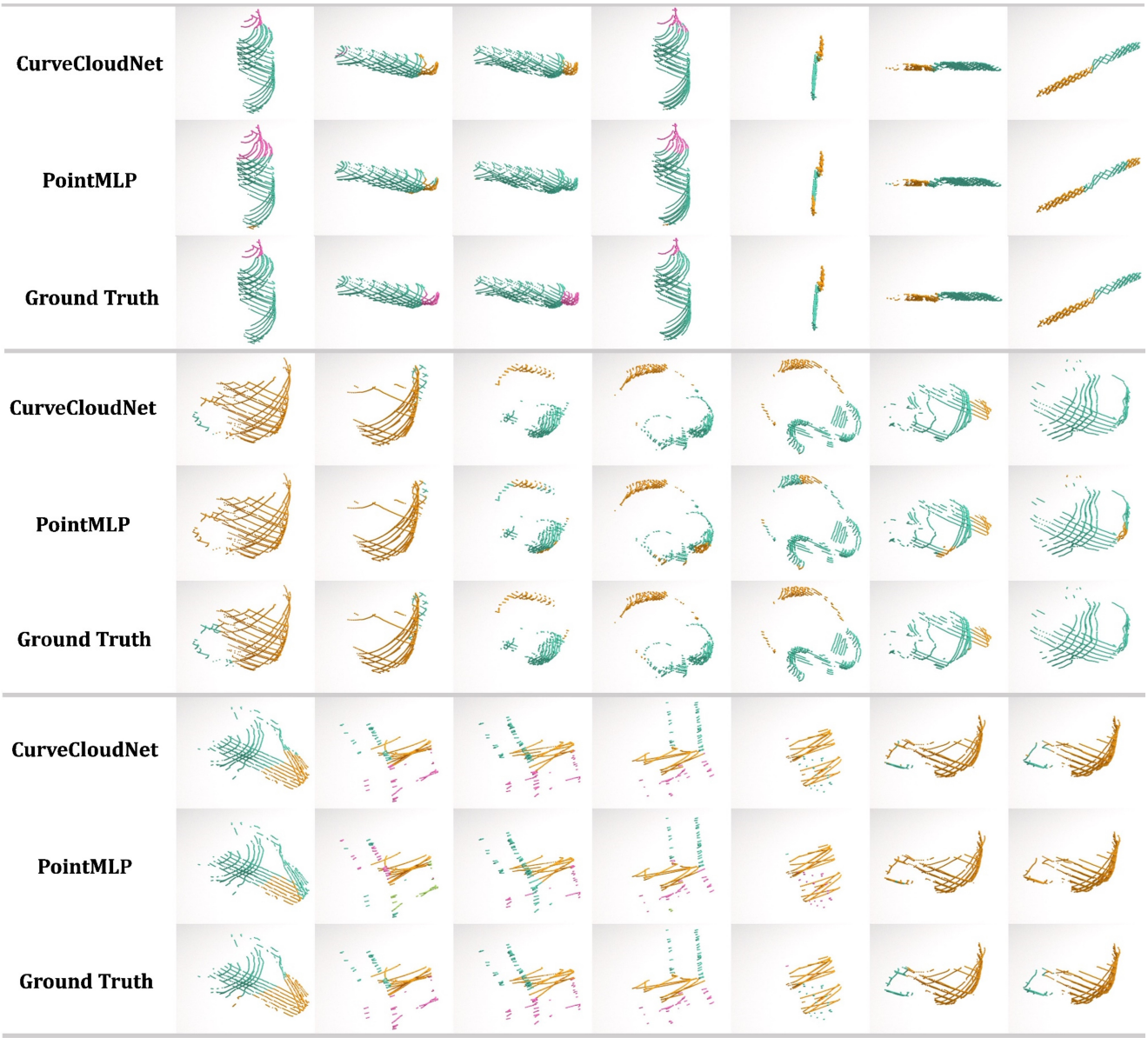

Figure 5. *Kortx Dataset Part Segmentation.* We visualize segmentation predictions of CurveCloudNet and PointMLP on the KortX dataset.

mantic segmentation. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020. 6

[25] Hui Zhou, Xinge Zhu, Xiao Song, Yuexin Ma, Zhe Wang, Hongsheng Li, and Dahua Lin. Cylinder3d: An effective 3d framework for driving-scene lidar semantic segmentation. *arXiv.org*, 2020. 4, 5, 6

[26] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018. 4

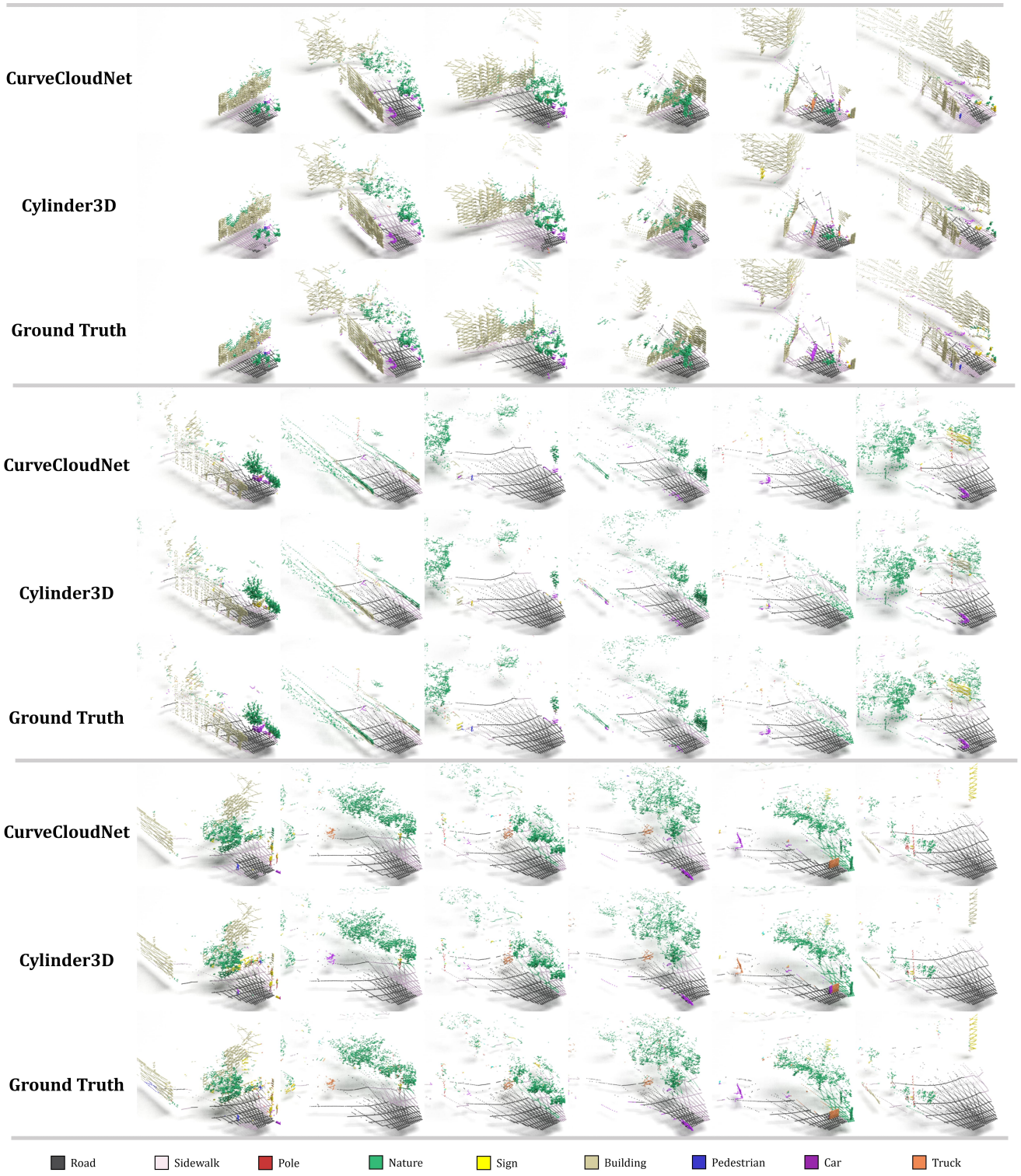

Figure 6. *A2D2 Dataset Segmentation.* We visualize segmentation predictions of CurveCloudNet and Cylinder3D on the A2D2 dataset.