Supplementary Material: Grid-GCN for Fast and Scalable Point Cloud Learning

Qiangeng Xu[1] Xudong Sun[2] Cho-Ying Wu[1] Panqu Wang[2] Ulrich Neumann[1]
[1]University of Southern California [2]Tusimple, Inc
{qiangenx,choyingw,uneumann}@usc.edu {xudong.sun,panqu.wang}@tusimple.ai

Appendices

A. Time complexity deductions of center sampling/node querying methods

We treat the number of voxel neighbors λ as a constant. In addition, the center sampling methods are only used during downsampling GridConv.

1. The time complexity of center sampling methods:

- RPS: Random sampling methods such as [1] provide RPS a complexity of $O(min(N, MlogM))$. In practice, $MlogM$ has a same or smaller magnitude of N, therefore report $O(N)$ in the time complexity table.
- FPS: FPS on a finite point set has $O(N^2)$, when M is not extremely small. However, [4] uses a Voronoi diagram to find the area that the point should exist, then find the nearest point in the calculated area. As an approximate algorithm, it has $O(NlogN)$.
- RVS: To sample point groups, CAGQ first scans over all points and build indices, which takes $O(N)$, RVS then randomly samples M center voxels from at most N occupied voxels (the num. of occupied voxels $\leq$ the number of raw points), which takes $O(min(N, MlogM))$. Under the same assumption of RPS, we report $O(N)$ in the time complexity table.
- CAS: If choosing CAS, we need to check all the unpicked occupied voxels and challenge the incumbents. To calculate a pair of H_{add} and H_{rmv}, CAGQ checks λ voxel neighbors of a challenger and an incumbent, result in a $O(\lambda N) = O(N)$ for all extra operations. Therefore CAS still has a complexity of $O(N)$.

2. The time complexity of node querying methods:

- Ball Query: For each center, Ball Query needs to run over N points to collect in-range points, then sample$K$points. Therefore it needs $O(MN)$
- k-NN: For each center, vanilla k-NN picks$K$nearest points from N points. The partition-based topK method takes $O(N)$ computation. Therefore each center has $O(N)$. k-NN can also first check if a point is within a range, then query top$K$candidate points. These two methods have the same worst-case complexity. The overall complexity is $O(MN)$.
- Cube Query: CAGQ's Cube Query randomly picks$K$points from λn_v context points. Since the order of points in each neighborhood is already randomized during GPU's multithreading collection, the overall complexity is $O(MK)$.
- CAGQ's k-NN: CAGQ picks$K$nearest points among points in the neighborhood. The partition-based topK algorithm provides a $O(\lambda n_v)$ solution for each point group. If λ is treated as a constant, the overall complexity is $O(Mn_v)$.

B. Training Details

For all experiments on ModelNet10, ModelNet40[10], ScanNet[3] and S3DIS[2], we use Adam[6] optimizer with beta1 = 0.9 and beta2 = 0.999. All models use batch normalization[5] with no momentum decay and trained on a single RTX 2080 GPU.

For ModelNet10 and ModelNet40[10], we start with a learning rate of 0.001 and reduce the learning rate by a factor of 0.7 every 60 epochs and stop at 330 epochs. We don't apply weight decay. The network has 2 downsampling GridConv layers each has 1024 and 128 point groups and a final global GridConv layer to group all points as one graph.

For ScanNet[3], we start with a learning rate of 0.001 and reduce the learning rate by a factor of 0.7 every 150 epochs and stop at 1500 epochs. Please note during each epoch, we only sample one block on the fly in each region. Therefore the training of one epoch is very quick. The network has 3 downsampling GridConv layers each has 1024, 256 and 24 point groups and 3 upsampling GridConv layers. We set our weight decay as 10^{-5} during training.

For S3DIS[2], we start with a learning rate of 0.001 and reduce the learning rate by a factor of 0.8 every 40 epochs and stop at 200 epochs. The network has 3 downsampling GridConv layers each has 512, 256 and 24 point groups and 3 upsampling GridConv layers. We set our weight decay as 10^{-8} during training.

C. Comparison between CAGQ and naive Grid Query

In 2D image learning, a convolution kernel usually traverses with a stride size that is smaller than the kernel size, leading to overlapping receptive fields. Since naive Grid Query first voxelizes the space and randomly picks M voxels, and then samples K points only within the voxel, the queried point groups have no space overlaps. On the other hand, CAGQ queries points inside voxel neighbors and utilizes Coverage-aware sampling to make the center voxels more evenly distributed.

To show the advantage of CAGQ over naive Grid Query, we compare 3 models on ScanNet[3] and report the results in Table 1. The two models using CAGQ are also the models we report in section 5.2 of the paper. We also train a model using naive Grid Query. As a result of the non-overlapping coverage by its point groups, the overall accuracy of the model with naive Grid Query can hardly reach 80%.

	OA	Latency (ms)
Grid-GCN(Grid Query + $1 \times K$)	79.9%	15.9
Grid-GCN(CAGQ + $0.5 \times K$)	83.9%	16.6
Grid-GCN(CAGQ + $1 \times K$)	85.4%	20.8

Table 1: The overall accuracy and latency of three Grid-GCN models on ScanNet[3]. Our full model uses CAGQ with $1 \times K$ node points in each group. A compact model with $0.5 \times K$ is also reported. Another model uses naive Grid Query with $1 \times K$ node points.

D. Algorithms of CAGQ

The general procedure of CAGQ is listed as Algorithm 1. The CAGQ k-NN algorithm mentioned in the paper is listed as Algorithm 2. To use RVS or CAS, we can embed the chosen algorithm in to step 2 of Algorithm 1. Cube Query and k-NN can be embedded in to step 3.

The k-NN methods in Algorithm 2 is efficient in three aspects:

- instead of all points, the candidates of k-NN are only the points in the neighborhood.
- We collect K nearest neighbor points first from inner voxel layers, then the outer voxel layers. We can stop at a layer if we have already got K points, since all the points in outer voxel layers are farther away than the point collected so far.
- In each layer of voxel neighbors, if the number of points so far collected plus the points in this layer is less than K, we do not need to sort them but can directly include all points in this layer.

Algorithm 1: CAGQ general procedure

Input N points: $p_i(\chi_i, w_i), i \in (1, ..., N)$
Parameters N, M, K, λ, n_v
1 Build voxel-point index Vid, collect O_v:
  For each p_i :
    $(u, v, w) \leftarrow$ quantize $p_i(x, y, z)$ into voxel index
    If voxel(u, v, w) is first visited
      Add (u, v, w) into O_v
    If $Vid_{(u,v,w)}$ stores fewer than n_v points
      Push point index into $Vid_{(u,v,w)}$
2 Center voxel sampling:
  $O_c \leftarrow$ select M voxels from O_v, by RVS or CAS.
3 Query node points and calculate group centers:
  For each center voxel v_j in O_c
    Retrieve points in $\pi(v_j)$ by using indices.
    Pick node points $\{p_{j1}, p_{j2}, ..p_{jK}\}$ in the neighborhood by using Cube Query or k-NN
    $w_{c_j} \leftarrow \sum_{k=1}^{K} w_{jk}$
    $\chi_{c_j} \leftarrow$ weighted mean of $\{\chi_{j1}, ..., \chi_{jK}\}$
Return M point groups: $group_j$:
  $(c_j(\chi_{c_j}, w_{c_j}), \{p_{j1}, p_{j2}, ..p_{jK}\}), j \in (1, ..., M)$

Algorithm 2: CAGQ k-NN for one point group in step 3 of Algorithm 1

Input A center voxel v_j, voxel-point index Vid, O_v
Parameters K
Counter = 0; node points ={}
For each level$_i$ of $\pi(v_j)$ (level$_0$ is the center voxel v_j itself, level$_1$ is the surrounding 26 voxels, etc.):
  $kl = 0$
  LayerPoints = {}
  For each voxel v_l in level$_i$:
    If $v_l \in O_v$:
      stored points $\leftarrow$ Retrieve points from $Vid(v_l)$:
      LayerPoints $\leftarrow$ add stored points
      $kl +=$ |stored points|
  $topkl = min(K$-Counter, $kl)$
  If $topkl = kl$:
    node points $\leftarrow$ LayerPoints
  Else:
    node points $\leftarrow$ topK(LayerPoints, $topkl$)
  Counter += $topkl$
  If Counter $\geq K$:
    break;
Return node points

E. Calculation of center

In a point group, we calculate w_c as the sum of its node points' coverage weight. χ_c is calculated as the barycenter of these nodes, weighted by the coverage weight.

$$w_c = \sum_{j=1}^{K} w_j \tag{1}$$

$$\chi_c(x, y, z) = \frac{\sum_{j=1}^{K} w_j \cdot \chi_j(x, y, z)}{\sum_{j=1}^{K} w_j} \tag{2}$$

F. Performance comparisons of data structuring methods (more conditions)

We list the full experiments of different data structuring methods' coverage and latency under more conditions in Table 2. The first section shows the coverage of occupied voxels. We only report the space coverage of center sampling methods+Ball Query or Cube Query, because the purpose of k-NN's is not to query node points that is evenly spread, but to query the nearest neighbor node points. The second and third section report the latency.

Center sampling			RPS	FPS	RVS*	CAS*	RPS	FPS	RVS*	CAS*	RPS	FPS	RVS*	CAS*
Neighbor querying			Ball	Ball	Cube	Cube	Ball	Ball	Cube	Cube	k-NN	k-NN	k-NN	k-NN
N	K	M	Occupied space coverage(%)				Latency (ms) with batch size = 1							
1024	8	8	12.3	12.9	13.1	**14.9**	**0.29**	0.50	0.51	0.64	0.84	0.85	**0.51**	0.65
	32	8	22.9	21.4	22.4	**31.7**	**0.34**	0.51	0.51	0.69	2.12	1.96	**0.63**	0.71
	128	8	22.3	22.6	23.5	**34.2**	**0.34**	0.51	0.94	1.04	8.26	6.70	**1.41**	1.63
	8	32	34.4	43.7	40.0	**45.6**	**0.31**	0.53	0.51	0.65	1.31	1.36	**0.57**	0.69
	32	32	58.2	69.48	60.1	**73.0**	**0.36**	0.55	0.53	0.57	4.68	4.72	**0.93**	0.68
	128	32	60.0	70.1	61.3	**74.7**	**0.37**	0.53	0.96	1.08	22.23	21.08	**2.24**	2.58
	8	128	64.0	72.5	82.3	**85.6**	**0.32**	0.78	0.44	0.58	1.47	1.74	**0.52**	0.61
	32	128	92.7	98.9	95.3	**99.6**	**0.38**	0.81	0.50	0.62	5.34	5.66	**1.18**	1.39
	128	128	93.6	99.5	95.8	**99.7**	**0.38**	0.69	0.97	0.97	32.48	32.54	**6.85**	6.94
8192	8	64	19.2	22.9	22.1	**25.1**	**0.64**	1.16	0.66	0.82	1.58	1.80	**0.65**	0.76
	32	64	42.7	42.7	35.8	**46.3**	1.47	1.47	**1.15**	1.39	2.73	2.73	**1.72**	2.16
	128	64	40.6	47.1	38.6	**51.3**	**1.14**	1.55	1.18	1.38	13.70	12.72	**9.42**	11.71
	8	256	60.1	64.1	73.3	**94.3**	**0.40**	1.51	0.53	0.61	4.53	5.54	**0.54**	0.68
	32	256	75.4	88.4	77.6	**90.7**	1.11	2.19	**1.04**	1.29	5.13	5.94	**3.06**	3.52
	128	256	79.9	90.7	80.0	**93.5**	1.19	1.19	**1.17**	1.31	21.5	21.5	**15.19**	17.38
	8	1024	82.9	**96.8**	92.4	94.4	0.81	4.90	**0.54**	0.77	1.53	5.36	**0.93**	0.97
	32	1024	96.3	97.8	99.3	**99.9**	1.15	5.09	**1.10**	1.54	5.18	8.99	**4.92**	6.32
	128	1024	98.8	99.9	99.5	**100.0**	**1.22**	5.25	1.40	1.76	111.42	111.74	**24.18**	26.45
81920	8	256	21.7	25.7	26.2	**31.2**	3.46	10.54	**2.20**	2.87	9.77	15.97	**1.85**	2.15
	32	256	34.2	40.1	36.0	**48.5**	7.59	14.51	**3.15**	4.34	20.43	26.14	**5.95**	6.17
	128	256	36.6	42.6	37.4	**51.1**	9.41	15.91	**3.52**	4.19	77.68	78.34	**34.04**	40.04
	8	1024	50.7	63.8	67.4	**76.0**	4.73	30.79	**2.13**	2.34	10.01	35.18	**1.84**	2.02
	32	1024	70.6	86.3	78.3	**91.6**	8.30	33.52	**3.34**	3.88	19.49	43.69	**8.76**	10.05
	128	1024	72.7	88.2	79.1	**92.6**	9.68	34.72	**4.32**	4.71	71.99	93.02	**50.70**	51.94
	8	10240	98.8	99.2	**100.0**	**100.0**	8.82	255.9	**4.11**	8.23	19.96	268.22	**9.54**	14.88
	32	10240	98.8	99.2	**100.0**	**100.0**	8.93	260.48	**4.22**	9.35	20.38	272.48	**9.65**	17.44
	128	10240	99.7	**100.0**	**100.0**	**100.0**	10.73	258.49	**5.83**	11.72	234.19	442.87	**69.02**	83.32

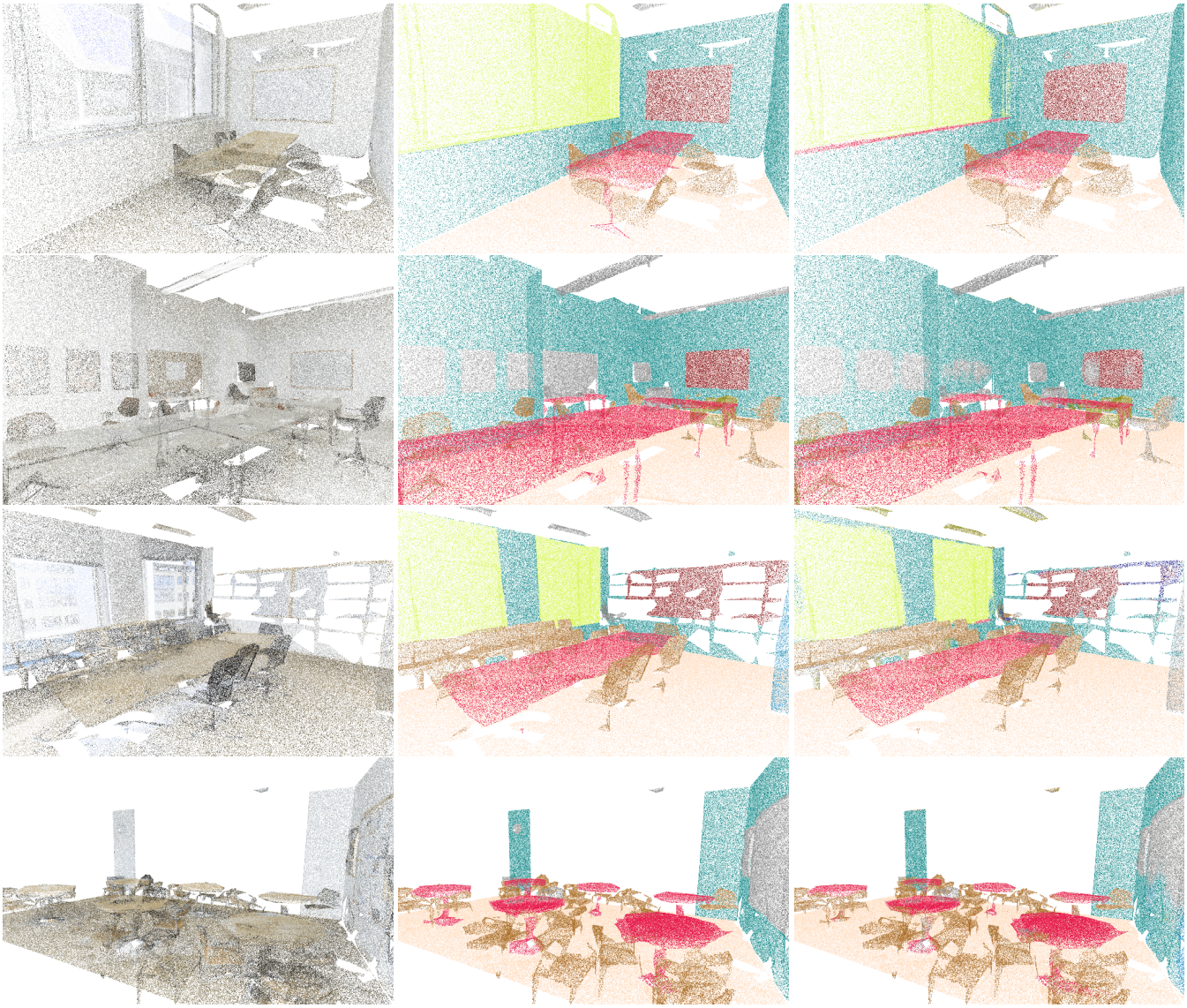
Table 2: Performance comparisons of data structuring methods, run on ModelNet40 [10]. Center sampling methods include RPS, FPS, CAGQ's RVS and CAS. Neighbor querying methods include Ball Query, Cube Query and k-Nearest Neighbors. Condition variables include N points, M groups, and K node points per group. Occupied space coverage = num. of occupied voxels of queried points / num. of occupied voxels of the original N points.

G. Performance on each object class and more visual results of S3DIS

We report the IOU of each object class in Table 3 and visualize more results of S3DIS[2] in Figure 1 and 2. The segmentation results are generated by our full model. In the visual results, Grid-GCN can predict objects such as chairs and tables very accurately, but sometimes mislabels the points on the border of two planar objects such as a board and a wall.

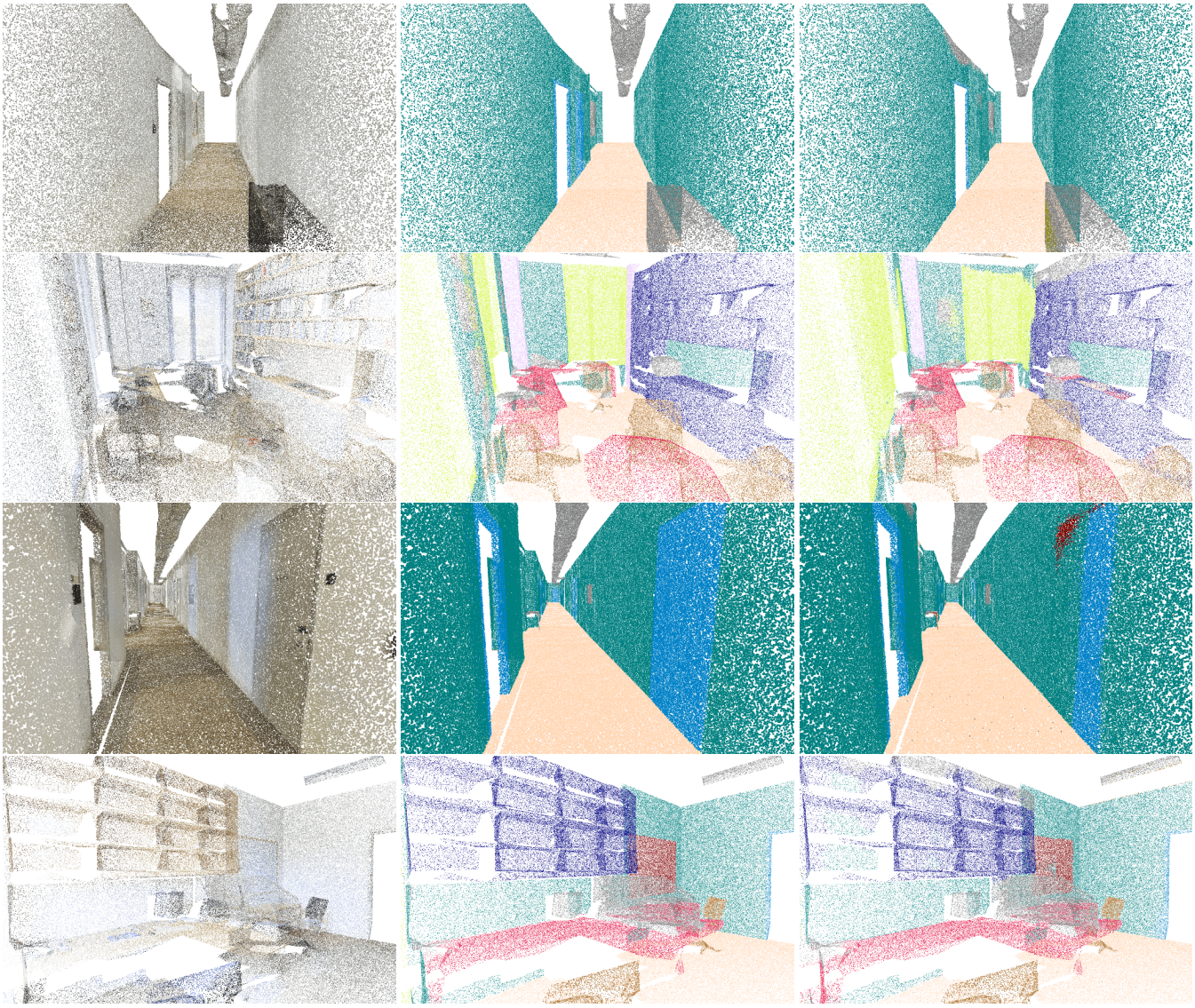
Method	OA	mIoU	ceiling	floor	wall	beam	column	window	door	table	chair	sofa	bookcase	board	clutter
PointNet[8]	-	41.09	88.80	97.33	69.80	**0.05**	3.92	**46.26**	10.76	58.93	52.61	5.85	40.28	26.38	33.22
SegCloud[9]		48.92	90.06	96.05	69.86	0.00	**18.37**	38.35	23.12	70.40	75.89	**40.88**	58.42	12.96	41.60
PointCNN[7]	85.91	57.26	92.31	**98.24**	**79.41**	0.00	17.60	22.77	**62.09**	**74.39**	80.59	31.67	66.67	62.05	**56.74**
Grid-GCN	**86.94**	**57.75**	**94.12**	97.28	77.66	0.00	16.61	32.91	58.53	72.15	**81.32**	36.46	**68.74**	**64.54**	50.46

Table 3: Segmentation result on S3DIS[2] area 5. We report overall accuracy (OA, %), mean class IoU (mIoU, %) and per-class IoU (%). Grid-GCN achieves the highest overall accuracy and mIoU among 4 models.



(a) Input XYZ+RGB (b) Ground truth (c) Ours

Figure 1: More visual result of S3DIS[2] area 5



(a) Input XYZ+RGB (b) Ground truth (c) Ours

Figure 2: More visual result of S3DIS[2] area 5

H. Summary of notations, acronyms and concepts

In the paper, we introduce the following concepts as well as their definitions. Here we list them again for clarity.

Notations and Concepts	Descriptions
GridConv	A network layer of our model, which includes a data structuring stage by using Coverage-Aware Grid Query and a data aggregation stage by using Grid Context Aggregation.
CAGQ	A data structuring module, named Coverage-Aware Grid Query.
GCA	A graph convolution module, named Grid Context Aggregation.
RPS	Random Point Sampling: A method that randomly samples M group centers from N points.
FPS	Farthest Point Sampling: A method that samples one point a time, each time picks the point that maximizes the distance to the selected points.
RVS	Random Voxel Sampling (CAGQ)
CAS	Coverage-Aware Sampling (CAGQ)
N	The number of input points of a GridConv layer.
M	The number of point groups sampled from N input points, the Grid Context Aggregation module aggregates node points' information to each group center, then output M representative points.
K	The number of node points in each point groups.
center voxel	For each point group, we select an occupied voxel and query node points in its neighborhood.
O_v	The set of all occupied (non-empty) voxels in the space.
O_c	The set of M center voxels that's sampled from O_v.
node points	For each point group, we query K points from context points in a neighborhood.
group center	The barycenter of K node points in each point group.
$\pi(v_i)$	The occupied voxel neighbors of an occupied voxel v_i.
λ	The number of occupied voxel neighbors of an occupied voxel v_i.
n_v	The max number of points CAGQ stores in each occupied voxel.
context points	All stored points in $\pi(v_i)$, these points are the context points of the center voxel v_i and the point group sampled in this neighborhood afterwards.
χ_i	The x,y,z vector of a node point p_i or a group center c.
w_i	The coverage weight of a node point p_i, which is the number of points that have been aggregated to that point in previous layers. We initialize w_i to 1 for each raw points.
f_i	The semantic features carried by a node point p_i.
$\widetilde{f}_i$	The semantic features calculated from f_i.
e_i	The edge attention between node point p_i and the center, calculated by the edge attention function.
$\widetilde{f}_{c,i}$	The contribution from $\widetilde{f}_i$ to the center, determined by e_i and $\widetilde{f}_i$.
$\widetilde{f}_c$	The features of the group center, aggregated from all node points' contribution $\widetilde{f}_{c,i}$.

Table 4: Notations, acronyms and concepts.

References

[1] Abejide Olu Ade-Ibijola. A simulated enhancement of fisher-yates algorithm for shuffling in virtual card games using domain-specific data structures. *International Journal of Computer Applications*, 54(11), 2012. 1

[2] I. Armeni, A. Sax, A. R. Zamir, and S. Savarese. Joint 2D-3D-Semantic Data for Indoor Scene Understanding. *ArXiv e-prints*, Feb. 2017. 1, 4, 5

[3] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017. 1, 2

[4] Y Eldar. *Irregular image sampling using the voronoi diagram*. PhD thesis, M. Sc. thesis, Technion-IIT, Israel, 1992. 1

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 1

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1

[7] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. In *Advances in Neural Information Processing Systems*, pages 820–830, 2018. 4

[8] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017. 4

[9] Lyne Tchapmi, Christopher Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. Segcloud: Semantic segmentation of 3d point clouds. In *2017 International Conference on 3D Vision (3DV)*, pages 537–547. IEEE, 2017. 4

[10] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015. 1, 3