Supplementary Material for SparsePCGC

Jianqiang Wang, Dandan Ding, Zhu Li, Xiaoxing Feng, Chuntong Cao, and Zhan Ma

Abstract—This document provides additional experimental details for comparative studies.

1 n-Stage SOPA Model

In the main text of this paper, we show that multi-stage SOPA can not only improves the compression efficiency but also enables parallel processing by properly grouping MP-POVs.

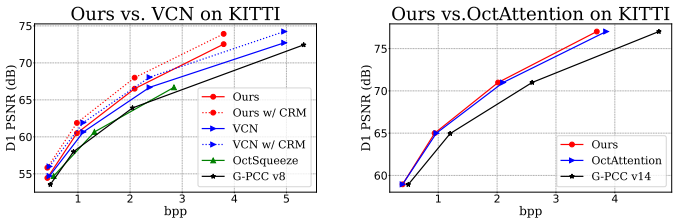
Here we offer one special case with which we sequantially process the element in upscaled MP-POV tensor to best exploit correlations from autoregressive neighbors for entropy context modeling. Similar method has been extensively used in learning-based 2D image/video coding [4], [9] with superior efficiency. Often times, a mask CNN is an efficient approach to facilitate such sequential computation where first upscale all decoded POVs, and then derive $p_{\text{MP-POV}}$ one by one from the top to bottom, from the left to right, and from the front to back. As all elements are completely available in encoder, we can apply full parallel processing; On the contrary, due to the causal dependency in context modeling, we have to process the element one by one.

Such "n-Stage SOPA" offers the best compression performance as listed in Table 3, with 50.8% and 39.4% compression ratio gains over the G-PCC anchor averaged on 8iVFB and MVUB samples, respectively. As seen, it significantly outperforms other methods that also use the autoregressive neighbors for context modeling, such as the VoxelDNN, NNOC, and OctAttention. Since it allows to process all MP-POV elements in parallel at encoder, the encoding time of n-Stage SOPA model only requires less than 1s. Note that these numbers are only served as the reference to have a general idea on the computational complexity because all these learning-based methods are implemented differently without a common software base. Unfortunately, such "n-Stage SOPA" suffers from an extremely-long decoding time due to the sequential computation, e.g., ≈2 hours averaged for 8iVFB and MVUB test samples, making it unbearable for applications. Impractical long decoding time is reported for VoxelDNN, NNOC and OctAttention as well.

2 Performance Comparisons

The proposed SparsePCGC unifies the compression of dense object and sparse LiDAR point clouds in both lossy and lossless modes. In addition to the performance summary in the main text, this section is devoted to offer more details used for experimental comparisons.

We have tried our best to faithfully reproduce the best results for the methods used in comparative studies, by enforcing the same test conditions (training/testing samples, bitrate points, etc) for performance evaluation, the same hardware platform (e.g., RTX 2080 GPU) for runtime collection, etc. Our sincere gratitude is directed the authors of [6], [8], [10]–[13] for their publicly-accessible materials and generous help.



(a) (b)

Fig. 1: Performance comparison using rate-distortion curves (a) Ours versus the VCN under the same test conditions used by VCN [13]; (b) Ours versus the OctAttention under the same test conditions used in OctAttention [6].

2.1 Sparse LiDAR PCG

Two approaches that offer the leading performance are selected for comparison, including the VoxelContext-Net (VCN) [13] and OctAttention [6]. For the comparison with OctSqueeze [7], we cite the results reproduced in VCN paper [13].

2.1.1 VoxelContext-Net (VCN)

To have the fair comparison with the VCN [13], we replicate its experimental setup. Since the source code of VCN is not publicly accessible, we *have discussed with the authors offline and obtained their test conditions and the latest results directly*. We greatly appreciate the generous help offered by them. The test conditions are shown below, which are strictly followed by us for conducting the comparison.

- **Training/Testing sequences**: Eleven point cloud sequences (#0∼#10) of KITTI (SemanticKITTI) dataset are used for training and the other 11 sequences (#11∼#21) are used for testing [2]. As for all testing sequences, a subset which is composed of 10 scans sampled from each testing sequence (the frame numbers of sampled scans are #0, #50, #100,. . ., #450) are selected to obtain the results presented in VCN [13].

TABLE 1: Comparison with VCN under VCN's test conditions. G-PCC v8 [1] is the anchor for BD-Rate computation. Encoding/decoding time is measured using second(s).

KITTI	G-PCCv8	VCN	OctSqueeze	Ours
bpp	0.469,0.914,2.045,5.330	0.453,1.103,2.373,4.957	0.195,0.534,1.316,2.842	0.412,0.979,2.087,3.795
PSNR	53.56,57.99,63.91,72.43	54.70,60.67,66.67,72.71	48.60,54.70,60.66,66.69	54.45,60.50,66.51,72.53
BD-Rate Gain	-	-16.72%	-2.08%	**-25.62%**
PSNR w/ CRM	-	56.00,61.93,68.06,74.24	-	55.83,61.88,68.00,73.92
BD-Rate Gain	-	-29.97%	-	**-37.74%**
EncTime (s)	0.539,0.620,0.817,1.281	-	-	0.744,0.927,1.154,1.438
DecTime (s)	0.090,0.143,0.261,0.533	0.052,0.058,0.078,0.090	0.006,0.007,0.007,0.008	0.634,0.828,1.061,1.319

TABLE 2: Comparison with OctAttention under OctAttention's test conditions. Anchor is the G-PCC v14 [1]. Encoding/decoding time is measured using second(s).

KITTI	G-PCC v14	OctAttention	Ours
bpp	0.491,1.202,2.595,4.748	0.394,0.968,2.101,3.852	0.395,0.939,2.016,3.694
PSNR	58.93,64.95,71.00,77.02	58.95,65.00,70.99,77.02	58.95,65.00,70.99,77.02
BD-Rate Gain	-	-19.48%	**-22.00%**
EncTime (s)	0.322,0.443,0.643,0.898	0.064,0.138,0.267,0.445	0.768,0.941,1.166,1.437
DecTime (s)	0.128,0.226,0.389,0.601	45,115,267,530	0.551,0.712,0.923,1.185

- **Training/Testing Strategy**: In training, the maximum depth level of KITTI sample is set to 12. A single entropy model of VCN is optimized for all 12 octree levels in general, but the Coordinate Refinement Model (CRM) is trained specifically for each depth, e.g., four CRM models for octree depths at 12, 11, 10 and 9 are used in [13]. In testing stage, different bitrates is obtained by truncating octree at different depth levels.
- **PSNR calculation**: D1 (Point-to-point) PSNR is used to measure the quality of the reconstructed point cloud, where all points are normalized to (-1,1) and the peak value p is set to 1.
- **Anchor**: MPEG G-PCC reference software version 8, a.k.a, G-PCC v8 (or more specifically v8.1 implementation) [1] is used to generate the anchor, where the raw floating-point coordinates are first multiplied by 100, and the quantization parameter ("positionQuantizaionScale") of G-PCC is set to 0.1/0.06/0.03/0.01 to obtain 4 bitrate points from around 0.5 bpp to around 5 bpp approximately.

Testing results are detailed in Table 1 and also illustrated in Fig. 1a. The numbers (bpp, PSNR) of VCN are provided by the authors directly, which corresponds to the Figure 6 in VCN paper [13]. The decoding time of VCN are directly quoted from the Table 4 in [13]. Note that VCN does not include the entropy decoding time into the measurement, and the encoding time is currently not available. The results of G-PCC v8 anchor are also quoted from [13] which are further confirmed by re-running the G-PCC codec.

As seen, our method provides better compression performance than the VCN regardless of the use of CRM. Currently, the VCN trains individual CRM model for each rate point, while our method reuses a single model for CRM[1] at all rate points. Currently, VCN shows the fastest decoding speed. Our method runs at second scale for both encoding and decoding, presenting the complexity at the same order of magnitude as the G-PCC v8.

1. In our implementation, the CRM module is referred to as the Position Offset Adjustment.

2.1.2 OctAttention

We also compare our method with the latest OctAttention [6] - a recently-published work in AAAI 2022 that reportedly provides state-of-the-art compression performance on large-scale point clouds. OctAttention reuses the same test conditions as the VCN, only having slightly differences on PSNR calculation (e.g., the peak value in VCN is derived using p^2 having $p = 1$, while in OctAttention, it is $3p^2$) and anchor selection (e.g., G-PCC v14 but not G-PCC v8 used in VCN). Given that source codes, pre-trained models and results of the OctAttention are made publicly accessible at https://github.com/zb12138/OctAttention, we faithfully reproduce their results on the same GPU platform, without any modification.

The results are shown in Table 2 and Fig. 1b. As seen, our method still offers better compression efficiency than OctAttention (e.g., -22.00% vs. -19.48%), when having the G-PCC v14 as the anchor. Although our encoding time is $\approx 2\times$ times longer than OctAttention, it is still very fast. More importantly, the decoding speed of our method shares the same order of magnitude as the encoding, which accelerates the decoding of OctAttention by at least two orders of magnitude.

2.2 Dense Object PCG

Three popular codecs, e.g., VoxelDNN [10], NNOC [8], and OctAttention [6], are used for comparison to understand the efficiency and complexity tradeoff when compressing the dense object PCG in lossless mode.

2.2.1 VoxelDNN

The VoxelDNN and its improvements [10], [11] are representative learning-based lossless PCC methods, which mainly apply mask 3D CNN for voxel occupancy probability approximation under a uniform voxel representation model. Here we follow their latest work [10] to replicate their training and testing conditions for our method. Unfortunately,

TABLE 3: Performance comparison of losslessly coded dense object point clouds. The numbers of VoxelDNN are quoted from the original papers while we faithfully reproduce the numbers of NNOC and OctAttention using their sources on the same platform. Please note the runtime are for reference only, since the testing dataset and implementation details of all methods are not completely the same. "seqs" and "one" are used for indicating whether the testing and runtime collection using all frames or just one frame, in each sequence.

Dense PCGs	**G-PCC (seqs/one)**	**Ours**		**VoxelDNN**		**NNOC**		**OctAttention**
		n**-Stage**	**8-Stage**	**VDNN**	**MsVDNN**	**NNOC**	**fNNOC**	
Test Conditions								
Training Set	-	8iVFB/MVUB		8iVFB/MVUB/ModelNet		8iVFB/MVUB		8iVFB/MVUB
Testing Set	seqs/one	seqs		one		seqs		seqs
Runtime Collection	seqs/one	seqs		one		seqs		seqs
8iVFB								
Red&black×300	1.088/1.083	0.574	0.635	0.665	0.87	0.723	0.866	0.73
Loot×300	0.961/0.947	0.469	0.533	0.577	0.73	0.59	0.743	0.62
Thaidancer×1	0.986	0.492	0.557	0.677	0.85	0.684	0.807	0.65
Boxer×1	0.943	0.421	0.493	0.550	0.70	0.551	0.682	0.59
Average Bpp	0.994/0.990	**0.489**	0.555	0.617	0.788	0.637	0.775	0.648
Gain	-	**-50.8%**	-44.2%	-37.7%	-20.5%	-35.9%	-22.1%	-34.9%
EncTime (s)	5.56/5.49	**0.75**	2.09	(885)	54)	91	64	1.06
DecTime (s)	3.26/3.21	2 hrs	**1.91**	(640)	(58)	1079	66	1229
MVUB								
Phil×245	1.135/1.142	0.711	0.714	0.76	1.02	0.782	1.021	0.79
Ricardo×216	1.061/1.103	0.623	0.647	0.687	0.95	0.701	0.941	0.72
Average Bpp	1.098/1.123	**0.667**	0.681	0.724	0.985	0.742	0.981	0.755
Gain	-	**-39.4%**	-37.7%	-35.5%	-12.5%	-32.1%	-10.2%	-30.9%
EncTime (s)	7.56/9.78	**0.96**	2.81	(885)	(85)	101	66	1.39
DecTime (s)	4.53/5.47	2 hrs	**2.64**	(640)	(92)	1318	69	1520

we are not able to reproduce their results by ourselves due to the lack of source codes and relevant materials. We therefore choose to cite the numbers from their paper.

- **Training sequences**: VoxelDNN and MsVoxelDNN used samples from the ModelNet [14], MVUB [3], and 8iVFB [5] for training. In ModelNet, they selected the largest 200 mesh models to generate 200 9-bit point clouds accordingly. In MVUB, they selected 17 frames from "Andrew", "David", and "Sarah" sequences. In 8iVFB, they selected 18 frames from "Soldier" and "Longdress" sequences.
- **Testing sequences**: Four other frames were chosen from MVUB and 8iVFB, e.g., "Phil_frame10", "Ricardo_frame76", "Loot_1000", and "Redandblack_1510"; In the mean time, another 2 frames from "Boxer" and "Thaidancer" from 8iVFB were used as well. All of these point clouds were quantized to 10-bit format for testing.
- **Runtime**: Both VoxelDNN and MsVoxelDNN were tested on a GeForce RTX 2080 GPU. The encoding and decoding time of VoxelDNN were 885s and 640s, respectively, directly quoted from the Table VII in [10]. However, the authors did not specify the test set for these numbers, we therefore marked them for both 8iVFB and MVUB test samples as in Table 3. The encoding/decoding time of MsVoxelDNN was 52s/58s on average for 8iVFB, 85s/92s for MVUB, respectively. Similarly, these numbers were presented in Table III of [11].

2.2.2 NNOC

The NNOC [8] is a learning-based octree codec, in which the occupancy probability of the octree node is estimated by available nodes surrounding it using Multi-Layer Perception (MLP). Its test condition is similar to the VoxelDNN, but it does not include ModelNet samples in training. For testing, all frames are used (and averaged) instead of a single frame in VoxelDNN [10]. For clarification, we use "seqs" and "one" in Table 3 to differentiate the testing cases using all frames in each sequence, or only one frame, respectively.

- **Training/Testing sequences**: NNOC method selected 18 frames from "Andrew", "David", and "Sarah" from MVUB (6 from each), and 18 frames from "Longdress" and "Soldier" from 8iVFB (9 from each) for training. For the testing dataset, NNOC used all frames of "Phil", "Ricardo", "Redandblack", and "Loot" for testing. The training set and the main testing set are all presetned at 10 bits.
- **Runtime**: NNOC only reported the runtime for the processing of three individual point clouds in the paper [8]. Given that its source codes, pre-trained models are made available to the public at https://github.com/marmus12/NNCTX. We faithfully reproduced their results, following the author's instructions without any modifications. The results are detailed in Table 3, revealing that our method using 8-Stage SOPA model is at least 20x faster than its fast version - fNNOC.

2.2.3 OctAttention

OctAttention [6] applied the similar test conditions of VoxelDNN, but had the differences below:

- **Training/Testing sequences**: OctAttention used all frames of "Soldier" and "Longdress" sequences in 8iVFB, and all frames of "Andrew", "David", and

"Sarah" sequences in MVUB for training. For testing, it used all frames of "Phil", "Ricardo", "Redandblack", and "Loot".

- **Runtime** : OctAttention reported the runtime on NVIDIA TITAN Xp GPU in its paper. Here we retested it on our RTX 2080 GPU - an inferior model to the TITAN Xp, for fair comparison. Thus a bit longer time was collected here than that in the original paper. In our test, it presented fast encoding by enforcing the massive parallelism, i.e., 1.06s and 1.39s encoding time averaged for 8iVFB and MVUB sequences, respectively; however, its decoding was super slow, requiring 1229s and 1520s on average to decode a frame, which was due to the causal dependency in context modeling as extensively discussed.

TABLE 4: **Runtime on different devices.** Averaged runtime of encoder and decoder using "longdress_vox10_1300", "loot_vox10_1200", "redandblack_vox10_1550", and "soldier_vox10_0690" in 8iVFB.

Device	EncTime (s)	DecTime (s)
Xeon Silver 4210 CPU	115	110
GeForce RTX 2080	1.86	1.78
GeForce RTX 3090	1.58	1.45

2.2.4 Discussion

Table 3 lists the efficiency and complexity tradeoff for VoxelDNN, NNOC, OctAttention, G-PCC v14 and ours. We have tried our best to reproduce the numbers of our method under the same conditions as these three methods. Given that these methods have used slightly different datasets for training, we propose to utilize the common sets, e.g., MVUB and 8iVFB, to train our model. We also choose the RTX 2080 GPU used by VoxelDNN and NNOC, instead of recently-released RTX 3090 or TITAN Xp, for testing and runtime collection. For those methods that are not publicly accessible, we choose to quote the best results from their paper instead. We particularly want to thank authors of NNOC and VCN for their offline discussions and confirmation of the numbers reported here.

As we can clearly observe, our method consistently demonstrates the better performance with higher compression ratio. More importantly, our method also provides the attractive complexity-performance tradeoff, exhibiting lightweight complexity to encode/decode large-scale dense object and sparse LiDAR point clouds.

Additionally, we collect runtimes in Table 4 by executing our model to encode/decode 8iVFB samples at different hardware platforms. This helps us to have a rough idea about the computational capacity of these devices. As seen, the use of 2080/3080 GPU can massively accelerate the processing of our SparsePCGC by $> 60\times/70\times$ to the case running our model on Xeon CPU. Whereas, the use of 3080 GPU only offers 15%$\sim$20% speedup to the use of 2080 GPU for executing the proposed SparsePCGC.

3 Open Source

Pretrained models, testing conditions and other relevant materials are made publicly accessible at https://github.com/NJUVISION/SparsePCGC. Source codes will be released soon to the public after the approval from the funding agency.

References

[1] Mpeg-pcc-tmc13. https://github.com/MPEGGroup/mpeg-pcc-tmc13. Accessed: 2021.

[2] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, C. Stachniss, and Juergen Gall. Semantickitti: A dataset for semantic scene understanding of lidar sequences. 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pages 9296–9306, 2019.

[3] Loop Charles, Cai Qin, O.Escolano Sergio, and A. Chou Philip. Microsoft voxelized upper bodies - a voxelized point cloud dataset. ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) m38673/M72012, May 2016.

[4] Tong Chen, Haojie Liu, Zhan Ma, Qiu Shen, Xun Cao, and Yao Wang. End-to-end learnt image compression via nonlocal attention optimization and improved contenxt modeling. IEEE Trans. Image Processing, 30:3179–3191, 2021.

[5] Eugene d'Eon, Bob Harrison, Taos Myers, and Philip A. Chou. 8i voxelized full bodies - a voxelized point cloud dataset. ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) m38673/M72012, May 2016.

[6] Chunyang Fu, Ge Li, Rui Song, Wei Gao, and Shan Liu. Octattention: Octree-based large-scale contexts model for point cloud compression. In AAAI, 2022.

[7] Lila Huang, Shenlong Wang, K. Wong, Jerry Liu, and R. Urtasun. Octsqueeze: Octree-structured entropy model for lidar compression. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 1310–1320, 2020.

[8] Emre Can Kaya and I. Tabus. Neural network modeling of probabilities for coding the octree representation of point clouds. IEEE MMSP, 2021.

[9] David Minnen, Johannes Ballé, and George D Toderici. Joint autoregressive and hierarchical priors for learned image compression. In IEEE NeurIPS, pages 10771–10780, 2018.

[10] D. Nguyen, Maurice Quach, G. Valenzise, and P. Duhamel. Lossless coding of point cloud geometry using a deep generative model. IEEE Transactions on Circuits and Systems for Video Technology, 2021.

[11] D. Nguyen, Maurice Quach, G. Valenzise, and P. Duhamel. Multiscale deep context modeling for lossless point cloud geometry compression. 2021 IEEE International Conference on Multimedia Expo Workshops (ICMEW), 2021.

[12] Maurice Quach, Giuseppe Valenzise, and F. Dufaux. Improved deep point cloud geometry compression. 2020 IEEE MMSP Workshop, 2020.

[13] Zizheng Que, Guo Lu, and Dong Xu. Voxelcontext-net: An octree based framework for point cloud compression. 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021.

[14] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1912–1920, 2015.