Multiscale deep context modeling for lossless point cloud geometry compression

Dat Thanh Nguyen, Maurice Quach, Giuseppe Valenzise, Pierre Duhamel
Université Paris-Saclay, CNRS, CentraleSupelec, Laboratoire des Signaux et Systèmes
91190 Gif-sur-Yvette, France
{thanh-dat.nguyen, maurice.quach, giuseppe.valenzise, pierre.duhamel}@l2s.centralesupelec.fr

2021 IEEE International Conference on Multimedia & Expo Workshops (ICMEW) | 978-1-6654-4989-2/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICMEW53276.2021.9455990

***Abstract*—We propose a practical deep generative approach for lossless point cloud geometry compression, called MSVoxelDNN, and show that it significantly reduces the rate compared to the MPEG G-PCC codec. Our previous work based on autoregressive models (VoxelDNN [1]) has a fast training phase, however, inference is slow as the occupancy probabilities are predicted sequentially, voxel by voxel. In this work, we employ a multiscale architecture which models voxel occupancy in coarse-to-fine order. At each scale, MSVoxelDNN divides voxels into eight conditionally independent groups, thus requiring a single network evaluation per group instead of one per voxel. We evaluate the performance of MSVoxelDNN on a set of point clouds from Microsoft Voxelized Upper Bodies (MVUB) and MPEG, showing that the current method speeds up encoding/decoding times significantly compared to the previous VoxelDNN, while having average rate saving over G-PCC of 17.5%. The implementation is available at https://github.com/Weafre/MSVoxelDNN.**

***Index Terms*—Point Cloud Compression, context model, Deep Generative Models, G-PCC, VoxelDNN**

I. Introduction

A point cloud is a set of 3D points, where each point is associated to spatial coordinates x, y, z (geometry) and attributes (color, reflectance, etc.). Unlike 2D images, the irregular spatial sampling of point clouds makes the coding task more challenging than for traditional video. As point clouds are a commonly used data structure in many applications such as immersive communication, autonomous vehicles, cultural heritage, etc., efficient compression methods are required for enabling effective point cloud transmission/storage.

Two Point Cloud Compression (PCC) standards have been developed by the Moving Picture Expert Group (MPEG) [2]: Video-base PCC (V-PCC) and Geometry-based PCC (G-PCC). V-PCC is based on 3D-to-2D projections and the 2D image/video compression standards are utilized to encode the projected data. On the other hand, G-PCC processes point clouds directly in the 3D space. The geometry and attribute information of a point cloud are first separated and G-PCC encodes them independently. Prior to actual geometry coding, the spatial coordinates of the points are first quantized to integer precision (voxelization). Once the PC is voxelized, its geometry can be represented in the voxel domain or octree domain. In particular, a binary occupancy map is defined over the voxel grid to indicate whether a voxel contains at least one point. This is the signal that we aim at coding.

In order to efficiently code the PC geometry losslessly, it is necessary to accurately estimate the occupancy probabilities

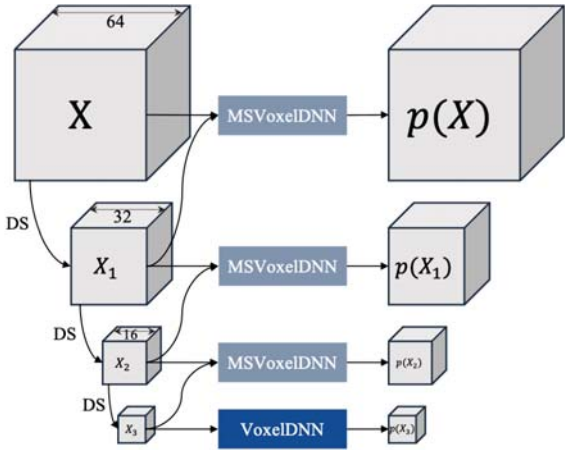

Fig. 1: Overview of the MSVoxelDNN architecture with input block of size 64 and 3 scales. DS is the downsampling operation (max-pooling). The base resolution of size 8 is encoded using a VoxelDNN context model. The higher resolutions are predicted from lower resolution as well as encoded groups at the same scale. The predicted block probabilities on the right side are passed to an arithmetic coder for encoding voxel occupancies. The final bitstream is the concatenation of all bits at all scales.

to be employed into a context-adaptive arithmetic codec. In our previous work, we have modeled the voxel occupancy distributions using a likelihood-based deep autoregressive network called VoxelDNN [1], inspired by the popular PixelCNN model [3]. VoxelDNN achieves state-of-the-art gains (up to 34%) over the MPEG G-PCC reference codec.

Autoregressive models can accurately predict probability distributions. However, the decoding process using this approach is equivalent to sampling from the high-dimensional conditional distribution of voxel occupancies, which is computationally complex as it demands one network evaluation per voxel. In this paper, taking inspiration from previous work in 2D image generation [4], we propose a multiscale method (named MSVoxelDNN) for lossless geometry compression of static dense point clouds which addresses the complexity problem of VoxelDNN. Our main contributions are:

- We introduce for the first time a multiscale deep context model in the voxel domain to estimate occupancy probabilities, in which higher-resolution scales are modeled

conditioned on the lower-resolution ones.

- We accelerate the inference by parallelizing voxel prediction. At each scale, voxels are partitioned into groups (see Figure 2). Voxels belonging to the same group are assumed to be conditionally independent from each other. In this way, we can predict all the voxels of the same group *simultaneously*, reducing the computation time. Instead, each group of voxels is assumed to depend on the previously decoded ones, and thus out context model can leverage dependencies between groups.

Compared to VoxelDNN, we make an approximation in that we do not utilize the statistical dependencies of voxels inside groups (due to the conditional independence assumption). We demonstrate experimentally that this approximation entails only a small loss of performance compared to the original VoxelDNN, and still outperforms significantly MPEG G-PCC in terms of bits per occupied voxel. However, in terms of complexity, MSVoxelDNN is on average 35 and 109 times faster compared to VoxelDNN for encoding and decoding, respectively. The rest of the paper is structured as follows: Section II reviews related work; the proposed MSVoxelDNN method is described in Section III-D; Section IV-B presents the experimental results; and conclusions are drawn in Section V.

II. Related Work

To deal with the irregular distribution of points in 3D space, many PCC methods employ octree representations [5]–[11] or local approximations [12]. The octree based method P(PNI) proposed in [10] builds a reference octree using an intra prediction mode. Each octant is then encoded with 255 contexts and a 255×255 frequency table must be transmitted to the decoder. In the MPEG G-PCC codec, geometry can be represented by a pruned octree plus a surface model (trisoup coder) or a full octree (octree coder). To exploit local geometry information within the octree and obtain an accurate context for arithmetic coding, the G-PCC octree coder introduces many techniques such as Neighbour-Dependent Entropy Context [13], intra prediction [14], planar/angular coding mode [15], [16], etc. Instead, in this paper, we represent the PC geometry in a *hybrid* mode, mixing the octree and voxel domains. On the one hand, the octree can adapt to the sparsity of the point cloud, as partitioning stops at the empty node; on the other hand, geometric information are kept and can be naturally processed by a neural network.

Recently, deep learning has been widely applied in point cloud coding in both the octree domain [11], [17] and especially voxel domain [1], [18]–[20]. A coding method for static LiDAR point cloud is proposed in [11], which learns the probability distributions of the octree based on contextual information and uses an arithmetic coder for lossless coding. In this work we focus instead on *dense* point clouds, where voxel-based approaches have shown interesting results. In particular, our recent work, VoxelDNN [1], is an auto-regressive based model which predicts the distribution of each voxel conditioned on the previously decoded voxels. VoxelDNN obtains an average rate saving of 30% over G-PCC. The auto-regressive approach of VoxelDNN is similar to PixelCNN [3] which provides accurate 2D data likelihood estimations. However, the common problem of auto-regressive models is the complexity, as these models require a network evaluation per voxel/sub-pixel. In 2D, several methods have been proposed to overcome this limitation. PixelCNN++ [21] models the joint distribution of three color channels simultaneously and proposes several optimizations to PixelCNN. Multiscale PixelCNN [4] generate pixels in certain groups in parallel. The L3C method [22] employs a latent space to facilitate the learning of conditional probability estimates at several scales. While this technique solves the *complexity* issue, the estimated probabilities are not accurate enough and the coding gains are significantly less interesting than using Pixel CNN. In this paper, we aim at finding a good trade-off between complexity and compression performance. Thus, we follow the principle of [4] introducing parallelization and multiscale prediction.

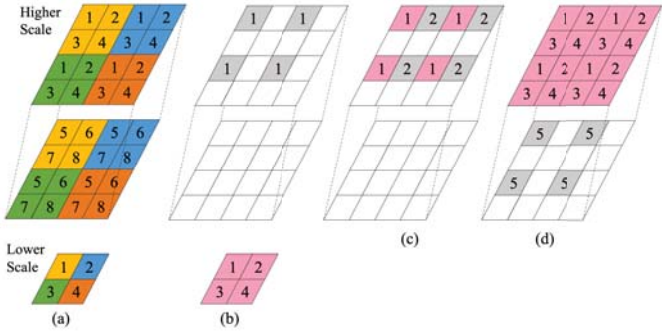

Fig. 2: Prediction parallelization in MSVoxelDNN. (a) partitioning of a block into groups of conditionally independent voxels. For the sake of clarity and without loss of generality, we show the context modeling for a block of size $2 \times 4 \times 4$. Downsampling is achieved by applying a $2 \times 2 \times 2$ max pooling operator, i.e., the voxels in the lower scale are the maximum of all voxels having the same color in the higher scale (MaxPooling operation). (b), (c) and (d) illustrate some steps of the groups prediction. The target groups are in gray, while the input (context) groups are in pink. (b) the 1^{st} group is predicted from all the groups at the lower resolution. (c) the 2^{nd} group is predicted from group 1 at the same resolution. (d) the 5^{th} group is predicted from group 1,2,3 and 4 (at the same scale).

III. Proposed Method

As mentioned before, in this paper we focus on voxelized point clouds. Without loss of generality, we assume the point cloud contains $2^n \times 2^n \times 2^n$ voxels. An octree is obtained by recursively spliting the voxel volume into eight sub-cubes until the desired precision is achieved. An occupied cube is marked by bit 1 and an empty cube is marked by bit 0. As a result, in each octree node, the generated 8 bits represent the occupancy of the 8 child nodes. A point cloud of size $2^n \times 2^n \times 2^n$ can be represented by an n level octree. In this

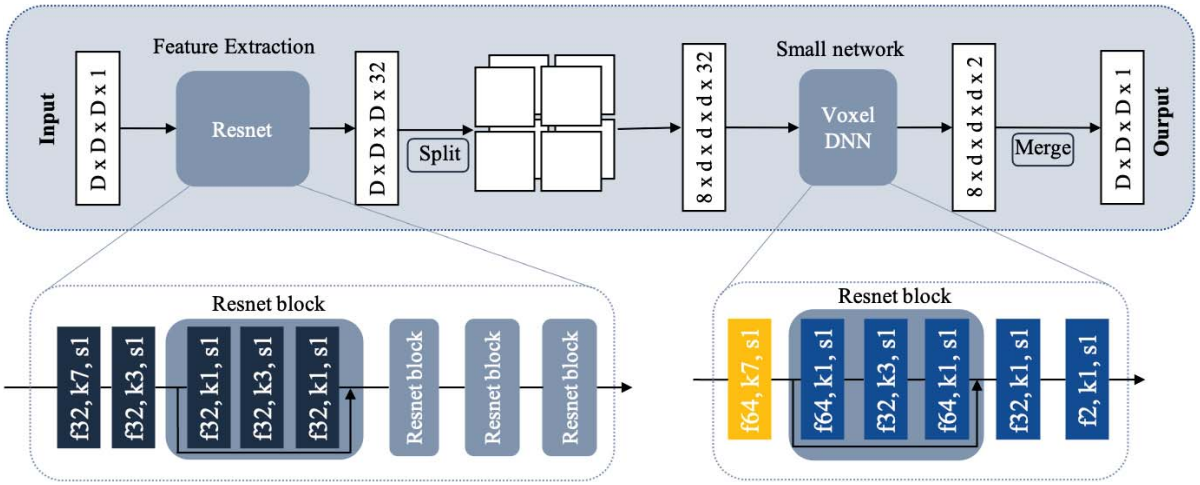

Fig. 3: Group prediction network architecture. This network predicts group 2 from group 1, corresponding to (c) in the example in Figure 2. The only learnable modules are Resnet and VoxelDNN. Merge and split operations only reshape data. We use a sequence of Resnet blocks to extract features from input. The features are then spatially split into smaller blocks before parallel processing by a small VoxelDNN. Black rectangular blocks are normal 3D convolution where 'f32,k7,s1' stands for 32 filters, kernel size 7 and stride 1. All convolutional blocks of VoxelDNN are masked convolutions [1], type A mask is in the first layer (in yellow), followed by type B masks.

work, and similar to [1], we partition an n-depth point cloud up to level $n-6$, and thus obtain a $n-6$ high level octree and a number of non-empty binary blocks v of size $2^6 \times 2^6 \times 2^6$, which we refer to as resolution $d = 64$. The high-level octree allows to coarsely remove most of the empty space in the point cloud, which does not contain any useful context information to predict occupancies. All the non-empty voxel blocks are further processed with our multiscale scheme. We first define a raster scan order in the 3D space that scans one voxel at a time in depth, height and width order. We index all voxels in block v at resolution d from 1 to d^3 in 3D raster scan order with:

$$v_i = \begin{cases} 1, & \text{if } i^{th} \text{ voxel is occupied} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

A. VoxelDNN context model

VoxelDNN [1] factorizes the joint distribution of a voxel block into a product of conditional distributions:

$$p(v) = \prod_{i=1}^{d^3} p(v_i | v_{i-1}, v_{i-2}, \ldots, v_1). \tag{2}$$

Each term $p(v_i|v_{i-1}, \ldots, v_1)$ above is the occupancy probability of voxel v_i given only the occupancy of previous voxels, referred to as *causality constraint*. All factors in equation (3) are estimated by a neural network with masked filters to enforce causality [1]. Therefore, the inference must also proceed sequentially voxel-by-voxel and VoxelDNN performs one network evaluation per voxel.

B. MSVoxelDNN context model

In this paper, we predict multiple voxels in parallel. As mentioned above, this calls for relaxing some dependencies between voxels.

First, we partition a voxel block into G separate groups and use v^g to represent all voxels in group g, $g = 1, \ldots, G$. We factorize the joint distribution $p(v)$ as a product of G conditional distributions $p(v^g|v^{g-1}, v^{g-2}, \ldots, v^1)$:

$$p(v) = \prod_{g=2}^{G} p(v^g | v^{g-1}, v^{g-2}, \ldots, v^1) \times p(v^1 | v_{LS}). \tag{3}$$

Each term $p(v^g|v^{g-1}, v^{g-2}, \ldots, v^1)$ is the joint probability of all voxels in v^g being occupied given all previous groups. Compared to VoxelDNN, we have removed the dependencies of voxels within each group. In return, we are able to predict all voxels in group g in parallel. In addition, given the first group, all other groups can be autoregressively predicted. We model voxels in the first group v^1 as conditionally independent given the lower resolution v_{LS}. This procedure is applied recursively to lower resolutions until the lowest scale, which is encoded using VoxelDNN. Figure 1 shows the general scheme of our Multiscale VoxelDNN encoder (MSVoxelDNN). At each step of the pyramid, downsampling is obtained by applying a maxpooling operation of size $2 \times 2 \times 2$ to the high resolution block, i.e., the resulting lower resolution voxel occupancy is one if at least one of the 8 higher resolution voxels is occupied. Therefore, by training the context model to predict the first group from v_{LS}, we somehow learn an inverse max pooling mapping for occupancy probabilities.

At a given scale, voxel groups are obtained by dividing the voxel block into non-overlapping $2 \times 2 \times 2$ blocks. We then select one of the 8 corners for each of $2 \times 2 \times 2$ blocks to get 8 groups. We build different models for different group predictions. Figure 2 shows a grouping example and prediction

TABLE I: Number of blocks in training sets for each block size.

Block size	MVUB	8i	CAT1	ModelNet40	Total
64	5,777	4,797	2,777	1,1147	24,498
32	22,082	20,436	15,243	50,611	108,372
16	87,578	86,106	45,626	224,951	444,261
8	354,617	349,760	180,037	986,253	1,870,667

scheme for group 1, 2 and 5. The other groups are modeled from previous groups in a similar manner as group 2, 5.

C. Network architecture

We employ a network structure similar to [4]. The network is composed of two stages: first, for each group prediction, we extract features using ResNet blocks. Compared to [4], we reduce the complexity of the feature extraction layer by just using 4 Resnet blocks instead of 12. The features enable to smooth out the discontinuities in the input voxel data, due to the sampling introduced with the grouping. The so-obtained spatial feature map is then partitioned into contiguous patches, such that there are P patches for each dimension, and thus $P \times P \times P$ in total (we omit here for simplicity the number of channels in the feature space). In the second stage, each patch is inputted to a shallow auto-regressive model (in this case, a VoxelCNN). We can accelerate training/inference with parallel patches prediction instead of prediction on the whole spatial feature map (i.e., using $P > 1$). However, too small patches can lead to inaccurate probabilities due to limited contexts. Therefore, we use $P = 2$ which require 2^3 small network evaluations in each forward pass ($P = 4$ in [4]).

Figure 3 shows the network architecture to predict group 2. Given group 1 of size $D \times D \times D$, the network outputs the predicted occupancy probabilities of all voxels in group 2. First, we use 4 ResNet blocks to extract a feature map from input, in each ResNet block, a 3D convolution with $3 \times 3 \times 3$ filter size is placed between two $1 \times 1 \times 1$ convolution layers. Next, the feature map are spatially divided into 8 patches of the same size and parallelly processed by a shallow VoxelCNN to produce occupancy probabilities. The probabilities are then merged back to a block of size $D \times D \times D \times 1$ which is the size of group 2. The shallow VoxelCNN is composed by one 3D convolutional layer with type A mask, a Resnet block followed by a 3D convolution layer. In each scale, we performs one network evaluation per group and then merge all 8 group probabilities into their spatial position in the output block.

Our predicted probabilities are fed as input to an arithmetic coder for lossless coding. Therefore, to minimize the output bitrate, we train MSVoxelDNN using cross-entropy loss, which is a measurement of the bitrate cost to be paid when the approximate symbol distribution $\hat{p}$ is used instead of the true symbol distribution p.

D. Complexity analysis

The bottleneck of VoxelDNN comes from the fact that it is necessary to apply the network on each new voxel to encode/decode. If there are d^3 voxels in a block, VoxelDNN requires $O(d^3)$ network evaluations to estimate probabilities during decoding. For VoxelDNN encoding, it is possible to partially parallelize the process by evaluating several contexts in parallel (since they are known at the encoder side), and thus divide the computational time by a constant factor. However, this does not influence the computational complexity.

Instead, MSVoxelDNN enables to reduce substantially the computational complexity compared to VoxelDNN. At the lowest resolution the block is coded using VoxelDNN with a small number of voxels ($d = 8$). Then, for each resolution level, the network is evaluated only G times, where G (the number of groups) is constant across scales. As we use a $2 \times 2 \times 2$ max pooling as downsampling operator in our work, the total number of levels is $\lceil \log_8 d^3 \rceil$, and thus the complexity is $O(\log d)$.

IV. Experimental results

A. Experimental Setup

Training dataset: We consider point clouds from different and varied datasets, including ModelNet40 [26] which contains 12,311 models from 40 categories and three smaller datasets: MVUB [23], MPEG CAT1 [24] and 8i [25]. We uniformly sample points from the mesh models from ModelNet40 and then scale them to voxelized point clouds with 9 bit precision. To enforce the fairness between the smaller datasets in which we select point clouds for testing, point clouds from MPEG CAT1 are sampled to 10 bit precision as in MVUB and 8i.

To train a MSVoxelDNN model at scale d we divide all selected PCs into occupied blocks of size $d \times d \times d$. Table I reports the number of blocks from each dataset for training, with the majority coming from the ModelNet40 dataset. Block 8 dataset is also used to train VoxelDNN model.

Training: We have 3 scales and at each scale we have 8 models for 8 groups and thus, in total we train 24 MSVoxelDNN models. The mini-batch sizes are 32 at scale 64 and 64 at other scales. Our models are implemented in PyTorch and trained with Adam optimizer, with a learning rate of $1e-5$ for 100 epochs on a GeForce RTX 2080 GPU.[1]

Experiments: We evaluate the performance of MSVoxelDNN on a set of dense point clouds from MPEG and Microsoft datasets. These PCs were not used during training. The final bitstream is composed of the bits at all scales and the bits for the high-level octree. The average bits per occupied voxel ($bpov$) are then measured by dividing the total bits by the number of occupied voxels. We compare the performance of MSVoxelDNN, VoxelDNN and G-PCC (version 12). Note that in the VoxelDNN paper [1], we use a single model for all block sizes. However, in this paper, we train separate VoxelDNN models for each block sizes on the same dataset as MSVoxelDNN to have a fair comparison.

[1]The source code, as well as the trained models, are available at https://github.com/Weafre/MSVoxelDNN.

TABLE II: Average rate in bpov of MSVoxelDNN compared with MPEG G-PCC v12 and VoxelDNN. The last column shows the gain reduction of MSVoxelDNN from VoxelDNN over G-PCC.

	G-PCC	VoxelDNN		MSVoxelDNN		
Point Cloud	bpov	bpov	Gain over G-PCC	bpov	Gain over G-PCC	Rate increase over VoxelDNN
Microsoft [23]						
Phil10	1.15	0.82	-29.37%	1.02	-11.13%	+18.25%
Ricardo10	1.07	0.74	-30.28%	0.95	-11.21%	+19.07%
Average	**1.11**	**0.78**	**-28.90%**	**0.99**	**-11.17%**	**+17.73%**
MPEG [24], [25]						
Redandblack10	1.09	0.71	-34.31%	0.87	-20.18%	+14.13%
Loot10	0.95	0.62	-34.16%	0.63	-21.05%	+13.11%
Thaidancer 10	1.00	0.73	-27.00%	0.85	-15.00%	+12.00%
Boxer 10	0.90	0.59	-34.44%	0.70	-26.32%	+8.12%
Average	**1.00**	**0.67**	**-31.79%**	**0.79**	**-20.55%**	**+11.24%**

B. Experimental results

In all experiments, the high-level octree are directly converted to bytes without any compression, as this part only accounts for less than 1% of the bitstream.

Rate comparison: Table II reports the rate in *bpov* of the proposed method, MSVoxelDNN, compared with G-PCC and VoxelDNN. We observe that MSVoxelDNN outperforms G-PCC on all test point clouds with rate savings from 11.13% to 26.32%. Compared to VoxelDNN, MSVoxelDNN has smaller gains over G-PCC, with a bitrate increase of 8.12% to 18.21%. This is due to the fact that MSVoxelDNN breaks some dependencies between voxels to model voxel probabilities in parallel, resulting in a less accurate context model.

Complexity comparison: Table III shows the encoding/decoding run-time for G-PCC, VoxelDNN and MSVoxelDNN. It can be seen that both VoxelDNN and MSVoxelDNN are slower than G-PCC, however there is a very large speedup of MSVoxelDNN compared to VoxelDNN. Specifically, MSVoxelDNN is 35 and 109 times faster for encoding and decoding, respectively, compared to VoxelDNN. The asymmetry of this time saving is due to the possibility to partially parallelize VoxelDNN at the encoder, as mentioned in Section III-D.

TABLE III: Encoding/decoding time comparison per dataset (in seconds).

	G-PCC	VoxelDNN	MSVoxelDNN
Encoding			
Microsoft	7	4,124	85
MPEG	3	2,459	54
Decoding			
Microsoft	5	10,332	92
MPEG	3	6,274	58

V. Conclusions

In this paper, we propose a Multiscale VoxelDNN method to lossless code the geometry of dense point clouds. On this kind of content, MSVoxelDNN reduces the bitrate compared to G-PCC by up to 17% on average, while reducing by over two orders of magnitudes the decoding complexity of the state-of-the-art VoxelDNN lossless codec. This is obtained by removing some dependencies between voxels in the same group in order to process them in parallel.

The performance of MSVoxelDNN could be further improved by optimizing the grouping of voxels, in such a way to remove only those dependencies that do not contribute significantly to the estimation of conditional occupancy probabilities. Also, the MSVoxelDNN scheme (but this is a common issue of VoxelDNN as well) yield poor performance on sparse point clouds – in general MSVoxelDNN has higher bitrates than G-PCC on point clouds which are not sufficiently dense. This is due to some basic hypotheses behind voxelization and convolutional neural networks, which require some substantial change of network architectures and PC representation. We are currently working towards an efficient learning-based lossless coding scheme for sparser point clouds to overcome these limitations.

References

[1] D. T. Nguyen, M. Quach, G. Valenzise, and P. Duhamel, "Learning-based lossless compression of 3d point cloud geometry," *arXiv preprint arXiv:2011.14700*, 2020.

[2] D. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, and A. Tabatabai, "An overview of ongoing point cloud compression standardization activities: video-based (V-PCC) and geometry-based (G-PCC)," *APSIPA Trans. on Signal and Information Process.*, vol. 9, 2020.

[3] A. Van Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," in *Intl. Conf. on Mach. Learn.* PMLR, 2016, pp. 1747–1756.

[4] S. Reed, A. Oord, N. Kalchbrenner, S. G. Colmenarejo, Z. Wang, Y. Chen, D. Belov, and N. Freitas, "Parallel multiscale autoregressive density estimation," in *Intl. Conf. on Mach. Learn.* PMLR, 2017, pp. 2912–2921.

[5] R. Schnabel and R. Klein, "Octree-based point-cloud compression." *Spbg*, vol. 6, pp. 111–120, 2006.
[6] R. Mekuria, K. Blom, and P. Cesar, "Design, implementation, and evaluation of a point cloud codec for tele-immersive video," *IEEE Trans. on Circuits and Syst. for Video Technol.*, vol. 27, no. 4, pp. 828–842, 2017.
[7] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach, "Real-time compression of point cloud streams," in *2012 IEEE Intl. Conf. on Robotics and Automation*, 2012, pp. 778–785.
[8] D. C. Garcia and R. L. de Queiroz, "Context-based octree coding for point-cloud video," in *2017 IEEE Intl. Conf. on Image Process. (ICIP)*, Sep. 2017, pp. 1412–1416, iSSN: 2381-8549.
[9] ——, "Intra-Frame Context-Based Octree Coding for Point-Cloud Geometry," in *2018 25th IEEE Intl. Conf. on Image Process. (ICIP)*, Oct. 2018, pp. 1807–1811.
[10] D. C. Garcia, T. A. Fonseca, R. U. Ferreira, and R. L. de Queiroz, "Geometry Coding for Dynamic Voxelized Point Clouds Using Octrees and Multiple Contexts," *IEEE Trans. on Image Process.*, vol. 29, pp. 313–322, 2019.
[11] L. Huang, S. Wang, K. Wong, J. Liu, and R. Urtasun, "OctSqueeze: Octree-Structured Entropy Model for LiDAR Compression," *arXiv:2005.07178 [cs, eess]*, May 2020.
[12] A. Dricot and J. Ascenso, "Adaptive multi-level triangle soup for geometry-based point cloud coding," in *2019 IEEE 21st Intl. Workshop on Multimedia Signal Process. (MMSP)*. IEEE, 2019, pp. 1–6.
[13] "Neighbour-dependent entropy coding of occupancy patterns," in *TMC3, ISO/IEC JTC1/SC29/WG11 input document m42238, Gwangju, Korea, January 2018.*
[14] "Intra mode for geometry coding," in *TMC3, ISO/IEC JTC1/SC29/WG11 input document m43600, Ljubljana, Slovenia, July 2018.*
[15] "Planar mode in octree-based geometry coding," in *TISO/IEC JTC1/SC29/WG11 input document m48906, Gothenburg, Sweden, July 2019.*
[16] "An improvement of the planar coding mode," in *ISO/IEC JTC1/SC29/WG11 input document m50642, Geneva, CH, Oct 2019.*
[17] S. Biswas, J. Liu, K. Wong, S. Wang, and R. Urtasun, "Muscle: Multi sweep compression of lidar using deep entropy models," *Advances in Neural Information Process. Syst.*, vol. 33, 2020.
[18] A. F. R. Guarda, N. M. M. Rodrigues, and F. Pereira, "Point cloud coding: Adopting a deep learning-based approach," in *2019 Picture Coding Symposium (PCS)*, 2019, pp. 1–5.
[19] M. Quach, G. Valenzise, and F. Dufaux, "Learning Convolutional Transforms for Lossy Point Cloud Geometry Compression," in *2019 IEEE Intl. Conf. on Image Process. (ICIP)*, Sep. 2019, pp. 4320–4324, iSSN: 1522-4880.
[20] J. Wang, H. Zhu, Z. Ma, T. Chen, H. Liu, and Q. Shen, "Learned point cloud geometry compression," *arXiv preprint arXiv:1909.12037*, 2019.
[21] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications," *arXiv preprint arXiv:1701.05517*, 2017.
[22] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. V. Gool, "Practical full resolution learned lossless image compression," in *Proceedings of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition*, 2019, pp. 10 629–10 638.
[23] C. Loop, Q. Cai, S. O. Escolano, and P. A. Chou, "Microsoft voxelized upper bodies - a voxelized point cloud dataset," in *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document m38673/M72012*, May 2016.
[24] "Common test conditions for PCC," in *ISO/IEC JTC1/SC29/WG11 MPEG output document N19324*.
[25] E. d'Eon, B. Harrison, T. Myers, and P. A. Chou, "8i Voxelized Full Bodies - A Voxelized Point Cloud Dataset," in *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document WG11M40059/WG1M74006*, Geneva, Jan. 2017.
[26] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D ShapeNets: A deep representation for volumetric shapes," in *2015 IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015, pp. 1912–1920.