ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) | 978-1-7281-7605-5/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICASSP39728.2021.9414763

LEARNING-BASED LOSSLESS COMPRESSION OF 3D POINT CLOUD GEOMETRY

Dat Thanh Nguyen, Maurice Quach, Giuseppe Valenzise, Pierre Duhamel

Université Paris-Saclay, CNRS, CentraleSupélec, Laboratoire des Signaux et Systèmes
91190 Gif-sur-Yvette, France

ABSTRACT

This paper presents a learning-based, lossless compression method for static point cloud geometry, based on context-adaptive arithmetic coding. Unlike most existing methods working in the octree domain, our encoder operates in a *hybrid* mode, mixing octree and voxel-based coding. We adaptively partition the point cloud into multi-resolution voxel blocks according to the point cloud structure, and use octree to signal the partitioning. On the one hand, octree representation can eliminate the sparsity in the point cloud. On the other hand, in the voxel domain, convolutions can be naturally expressed, and geometric information (i.e., planes, surfaces, etc.) is explicitly processed by a neural network. Our context model benefits from these properties and learns a probability distribution of the voxels using a deep convolutional neural network with masked filters, called VoxelDNN. Experiments show that our method outperforms the state-of-the-art MPEG G-PCC standard with average rate savings of 28% on a diverse set of point clouds from the Microsoft Voxelized Upper Bodies (MVUB) and MPEG. The implementation is available at `https://github.com/Weafre/VoxelDNN`.

Index Terms— Point Cloud Compression, Deep Learning, G-PCC, context model.

1. INTRODUCTION

Recent visual capturing technology has enabled 3D scenes to be captured and stored in the form of Point Clouds (PCs). PCs are now becoming the preferred data structure in many applications (e.g., VR/AR, autonomous vehicle, cultural heritage), resulting in a massive demand for efficient Point Cloud Compression (PCC) methods.

The Moving Picture Expert Group has been developing two PCC standards [1–3]: Video-based PCC (V-PCC) and Geometry-based PCC (G-PCC). V-PCC focuses on dynamic point clouds and is based on 3D-to-2D projections. On the other hand, G-PCC targets static content and encodes the point clouds directly in 3D space. In G-PCC, the geometry and attribute information are independently encoded. However, the geometry must be available before filling point clouds with attributes. Therefore, having an efficient lossless geometry coding is fundamental for efficient PCC. A typical point cloud compression scheme consists in pre-quantizing the geometric coordinates using voxelization. In this paper, we also adopt this approach, which is particularly suited for dense point clouds. After voxelization, the point cloud geometry can be represented either directly in the voxel domain, or using an octree spatial decomposition.

In this work, we propose a deep-learning-based method for lossless compression of voxelized point cloud geometry. Using a masked 3D convolutional network, our approach (named VoxelDNN) first learns the distribution of a voxel given all previously decoded ones. This conditional distribution is then used to model the context of a context-based arithmetic coder. In addition, we reduce point cloud sparsity by adaptively partitioning the PC. We demonstrate experimentally that the proposed solution outperforms the MPEG G-PCC solution in terms of bits per occupied voxel with average rate savings of 28% on all test datasets.

The rest of the paper is structured as follows: Section 2 reviews the related work; the proposed method is described in Section 3; Section 4 presents the experimental results; and finally Section 5 concludes the paper.

2. RELATED WORK

Most existing point cloud geometry compression methods, including MPEG G-PCC test models, represent and encode occupied voxels using octrees [4–6] or local approximations called "triangle soups" [7]. Recently, the authors of [6] proposed an intra-frame method called P(PNI), which builds a reference octree by propagating the parent octet to all children nodes, thus providing 255 contexts to encode the current octant. A frequency table of size 255×255 is built to encode the octree and needs to be transmitted to decoder. A drawback of this octree representation is that, at the first levels of the tree, it produces "blocky" scenes, and geometry information of point clouds (i.e., curve, plane) is lost. Instead, in this paper we work in the *hybrid domain* to exploit the geometry information. In addition, our method predicts voxel distributions in a sequential manner at the decoder side, thus avoiding the extra cost of transmitting large frequency tables.

Our work draws inspiration from the recent advances in deep generative models for 2D images. The goal of generative models is to learn the data distribution, which can be used for a variety of tasks, with image generation being probably the most popular [8]. Among the various classes of generative models, we consider methods able to explicitly estimate data likelihood, such as the PixelCNN model [9, 10]. Specifically, these approaches factorize the likelihood of a picture by modeling the conditional distribution of a given pixel's color given all previously generated pixels. PixelCNN models the distribution using a neural network. The causality constraint is enforced using masked filters in each convolutional layer. Recently, this approach has been employed in image compression to yield accurate and learnable entropy models [11]. This paper extends the generative model and masking filters to 3D point cloud geometry compression.

Inspired by the success in learning-based image compression, deep learning has been recently adopted in point cloud coding methods [12–16]. The proposed methods in [15, 16] encode each $64 \times 64 \times 64$ sub-block of PC using a 3D convolutional auto-encoder. In contrast, in this paper we losslessly encode the voxels by directly learning the distribution of each voxel from its 3D context. We also offer block-based coding, which was successful in traditional image and video coding.

3. PROPOSED METHOD

3.1. Definitions

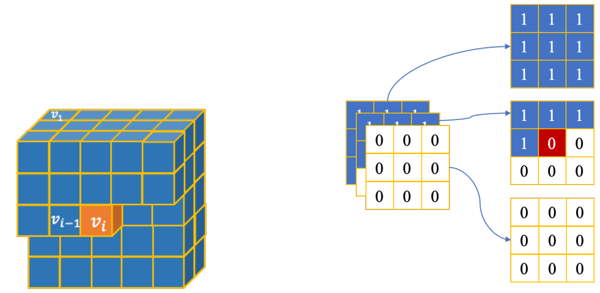
A point cloud voxelized over a $2^n \times 2^n \times 2^n$ grid is known as an n-bit depth PC, which can be represented by an n level octree. In this work, we represent point cloud geometry in a hybrid manner: both in octree and voxel domain. We coarsely partition an n-depth point cloud up to level $n-6$. As a result, we obtain a $n-6$ level octree and a number of non-empty binary blocks v of size $2^6 \times 2^6 \times 2^6$ voxels, which we refer to as resolution $d = 64$ or simply block 64 in the following. Blocks 64 can be further partitioned at resolution $d = \{32, 16, 8, 4\}$ as detailed in Section 3.3. At this point, we encode the blocks using our proposed encoder in the voxel domain (Section 3.2). The high-level octree, as well as the depth of each block, are converted to bytes and signaled to the decoder as side information. We index all voxels in block v at resolution d from 1 to d^3 in raster scan order with:

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

3.2. VoxelDNN

Our method encodes the voxelized point cloud losslessly using context-adaptive binary arithmetic coding. Specifically, we focus on estimating accurately a probability model $p(v)$ for the occupancy of a block v composed by $d \times d \times d$ voxels. We factorize the joint distribution $p(v)$ as a product of conditional distributions $p(v_i|v_{i-1}, \dots, v_1)$ over the voxel volume:

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



(a) 3D voxel context (b) 3D type A mask

Fig. 1 (a): Example 3D context in a $5 \times 5 \times 5$ block. Previously scanned elements are in blue. (b): $3 \times 3 \times 3$ 3D type A mask. Type B mask is obtained by changing center position (marked red) to 1.

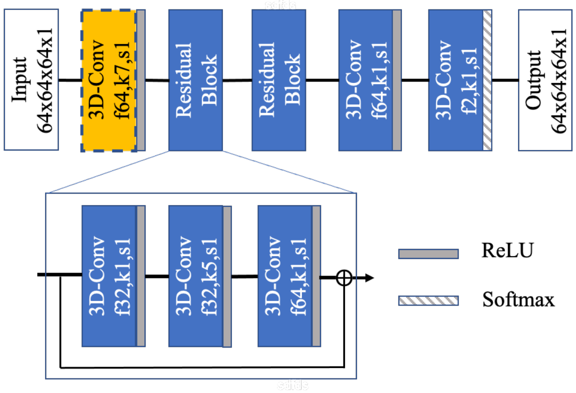


Fig. 2 VoxelDNN architecture. A type A mask is applied in the first layer (dashed borders) and type B masks afterwards. 'f64,k7,s1' stands for 64 filters, kernel size 7 and stride 1.

Each term $p(v_i|v_{i-1}, \dots, v_1)$ above is the probability of the voxel v_i being occupied given all previous voxels, referred to as a context. Figure 1(a) illustrates an example 3D context. We estimate $p(v_i|v_{i-1}, \dots, v_1)$ using a neural network which we dub **VoxelDNN**.

The conditional distributions in (2) depend on previously decoded voxels. This requires a *causality* constraint on the VoxelDNN network. To enforce causality, we extend to 3D the idea of masked convolutional filters, initially proposed in PixelCNN [9]. Specifically, two kinds of masks (A or B) can be employed. Type A mask is filled by zeros from the center position to the last position in raster scan order as shown in Figure 1(b). Type B mask differs from type A in that the value in the center location is 1. We apply type A mask to the first convolutional layer to restrict both the connections from all future voxels and the voxel currently being predicted. In contrast, from the second convolutional layer, type B masks are applied which relaxes the restrictions of mask A by allowing the connection from the current spatial location to itself.

In order to learn good estimates $\hat{p}(v_i|v_{i-1}, \dots, v_1)$ of the underlying voxel occupancy distribution $p(v_i|v_{i-1}, \dots, v_1)$, and thus minimize the coding bitrate, we train VoxelDNN using cross-entropy loss. That is, for a block v of resolution d,

Algorithm 1: Block partitioning selection

```
Input: block: $B$, current level: $curLv$, max level: $maxLv$
Output: partitioning flags: $fl$, output bitstream: $bits$
Function partitioner ($B, curLv, maxLv$):
    $fl2 \leftarrow 2$; // encode as 8 child blocks
    for block $b$ in child blocks of $B$ do
        if $b$ is empty then
            $child\_flag \leftarrow 0$;
            $child\_bit \leftarrow empty$;
        else
            if $curLv == maxLv$ then
                $child\_flag \leftarrow 1$;
                $child\_bit \leftarrow$ singleBlockCoder($b$);
            else
                $child\_flag, child\_bit \leftarrow$ partitioner($b$,
                  $curLv + 1, maxLv$);
            end
        end
        $fl2 \leftarrow [fl2, child\_flag]$;
        $bit2 \leftarrow [bit2, child\_bit]$;
    end
    $total\_bit2 = sizeOf(bit2) + len(fl2) \times 2$;
    $fl1 \leftarrow 1$; // encode as a single block
    $bit1 \leftarrow$ singleBlockCoder($B$);
    $total\_bit1 = sizeOf(bit1) + len(fl1) \times 2$;
    /* partitioning selection */
    if $total\_bit2 \geq total\_bit1$ then
        return $fl1, bit1$;
    else
        return $fl2, bit2$;
    end
```

we minimize :

$$H(p, \hat{p}) = \mathbb{E}_{v \sim p(v)} \left[\sum_{i=1}^{d^3} -\log \hat{p}(v_i) \right]. \tag{3}$$

It is well known that cross entropy represents the extra bitrate cost to pay when the approximate distribution $\hat{p}$ is used instead of the true p. More precisely, $H(p, \hat{p}) = H(p) + D_{KL}(p \| \hat{p})$, where D_{KL} denotes the Kullback-Leibler divergence and $H(p)$ is Shannon entropy. Hence, by minimizing (3), we indirectly minimize the distance between the estimated conditional distributions and the real data distribution, yielding accurate contexts for arithmetic coding. Note that this is different from what is typically done in learning-based *lossy* PC geometry compression, where the focal loss is used [15, 16]. The motivation behind using focal loss is to cope with the high spatial unbalance between occupied and non-occupied voxels. The reconstructed PC is then obtained by hard thresholding $\hat{p}(v)$, and the target is thus the final classification accuracy. Conversely, here we aim at estimating accurate soft probabilities to be fed into an arithmetic coder.

Figure 2 shows our VoxelDNN network architecture. Given the $64 \times 64 \times 64$ input block, VoxelDNN outputs the predicted occupancy probability of all input voxels. Our first 3D convolutional layer uses $7 \times 7 \times 7$ kernels with a type A mask. Type B masks are used in the subsequent layers. To avoid vanishing gradients and speed up the convergence, we implement two residual connections with $5 \times 5 \times 5$ kernels. Throughout VoxelDNN, the ReLu activation function is applied after each convolutional layer, except in the last layer where we use softmax activation. In total, our model contains 290,754 parameters and requires less then 4MB of disk storage space.

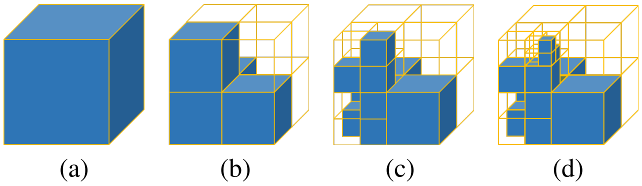

(a) (b) (c) (d)

Fig. 3 Partitioning a block of size 64 into: (a) a single block of size 64, (b): blocks of size 32, (c): 32 and 16, (d): 32, 16 and 8. Non-empty blocks are indicated by blue cubes.

3.3. Multi-resolution encoder and adaptive partitioning

We use an arithmetic coder to encode the voxels sequentially from the first voxel to the last voxel of each block in a generative manner. Specifically, every time a voxel is encoded, it is fed back into VoxelDNN to predict the probability of the next voxel. Then, we pass the probability to the arithmetic coder to encode the next symbol.

However, applying this coding process at a fixed resolution d (in particular, on blocks 64), can be inefficient when blocks are sparse, i.e., they contain only few occupied voxels. This is due to the fact that in this case, there is little or no information available in the receptive fields of the convolutional filters. To overcome this problem, we propose a rate-optimized multi-resolution splitting algorithm as follows. We partition a block into 8 sub-blocks recursively and signal the occupancy of sub-blocks as well as the partitioning decision (0: empty, 1: encode as single block, 2: further partition). The partitioning decision depends on the output bits after arithmetic coding. If the total bitstream of partitioning flags and occupied sub-blocks is larger than encoding parent block as a single block, we do not perform partitioning. The details of this process are shown in Algorithm 1. The maximum partitioning level is controlled by $maxLv$ and partitioning is performed up to $maxLv = 5$ corresponding to a smallest block size of 4. Depending on the output bits of each partitioning solution, a block of size 64 can contain a combination of blocks with different sizes. Figure 3 shows 4 partitioning examples for an encoder with $maxLv = 4$. Note that VoxelDNN learns to predict the distribution of the current voxel from previous encoded voxels. As a result, we only need to train a *single* model to predict the probabilities for different input block sizes.

Table 1: Average rate in bpov of VoxelDNN at different partitioning levels compared with MPEG G-PCC v12 and P(PNI).

		P(PNI)	**G-PCC**	**block 64**		**block 64 + 32**		**block 64 + 32 + 16**		**block 64 + 32 + 16 + 8**	
Dataset	Point Cloud	bpov	bpov	bpov	Gain over G-PCC	bpov	Gain over G-PCC	bpov	Gain over G-PCC	bpov	Gain over G-PCC
MVUB	Phil9	1.88	1.2285	0.9819	20.07%	0.9317	24.16%	0.9203	25.09%	0.9201	25.10%
	Ricardo9	1.79	1.0423	0.7910	24.11%	0.7276	30.19%	0.7175	31.16%	0.7173	31.18%
	Phil10	-	1.1599	0.8941	22.92%	0.8381	27.74%	0.8308	28.37%	0.8307	28.38%
	Ricardo10	-	1.0673	0.8108	24.03%	0.7596	28.83%	0.7539	29.36%	0.7533	29.42%
	Average	**1.84**	**1.1245**	**0.8695**	**22.78%**	**0.8143**	**27.73%**	**0.8056**	**28.50%**	**0.8054**	**28.52%**
MPEG 8i	Loot10	1.69	0.9525	0.7016	26.34%	0.6464	32.14%	0.6400	32.81%	0.6387	32.94%
	Redandblack10	1.84	1.0890	0.7921	27.26%	0.7383	32.20%	0.7317	32.81%	0.7317	32.81%
	Boxer9	-	1.0816	0.8034	25.72%	0.7620	29.55%	0.7558	30.12%	0.7560	30.10%
	Thaidancer9	-	1.0679	0.8574	19.71%	0.8145	23.73%	0.8091	24.23%	0.8078	24.36%
	Average	**1.77**	**1.0478**	**0.7886**	**24.76%**	**0.7403**	**29.40%**	**0.7342**	**29.99%**	**0.7336**	**30.05%**

4. EXPERIMENTAL RESULTS

4.1. Experimental Setup

Training dataset: Our models were trained on a combined dataset from ModelNet40 [17], MVUB [18] and 8i [19, 20] datasets. We sample 17 PCs from Andrew10, David10, Sarah10 sequences in MVUB dataset into the training set. In 8i dataset, 18 PCs sampled from Soldier and Longdress sequence are selected. The ModelNet40 dataset, is sampled into depth 9 resolution and the 200 largest PCs are selected. Finally, we divide all selected PCs into occupied blocks of size $64 \times 64 \times 64$. In total, our dataset contains 20,264 blocks including 4,297 blocks from 8i, 4,820 blocks from MVUB and 11,147 blocks from ModelNet40. We split the dataset into 18,291 blocks for training and 1,973 blocks for testing.

Training: VoxelDNN is trained with Adam [21] optimizer, a learning rate of 0.001, and a batch size of 8 for 50 epochs on a GeForce RTX 2080 GPU.

Evaluation procedure: We evaluate our methods on both 9 and 10 bits depth PCs that have the lowest and highest rate performance when testing with G-PCC method from the MVUB and MPEG datasets. These PCs were not used during training. The effectiveness of the partitioning scheme is evaluated by increasing the maximum partitioning level from 1 to 5 corresponding to block sizes 64, 32, 16, 8 and 4.

4.2. Experimental results

Table 1 reports the average rate in bits per occupied voxel (bpov) of the proposed method for 4 partitioning levels, compared with G-PCC v.12 [3]. We also report the results of the recent intra-frame geometry coding method P(PNI) from [6] (the coding results are available only for some of the tested PCs). The results with 5 partitioning levels are identical to 4 partitioning levels and are not shown in the table. In all experiments, the total size of signaling bits for the high-level octree and partitioning accounts for less than 2% of the bitstream.

We observe that our proposed solution at all 4 levels and G-PCC outperform P(PNI) by a large margin. VoxelDNN solutions outperform G-PCC on the MVUB and MPEG 8i datasets with the highest average rate saving of 28.4% and 29.9%, respectively. As partitioning levels increases, the corresponding gain over G-PCC also increases; however, there is only a slight increase with 3 and 4 levels compared to the gain of the lower level. This can be explained with Figure 4, which shows the percentage allocation of the encoded points in each partitioning size, for different PCs, after optimal partitioning. It can be seen that most voxels are encoded using blocks 64 and 32, while very few ones are encoded with blocks of smaller size. While adding more partition levels enables to better adapt to point cloud geometry, smaller partitions entail a higher signalization cost. This is not often compensated by a bitrate reduction of the sub-blocks, since in the smaller partitions the encoder has access to limited contexts, resulting in less accurate probability estimations.

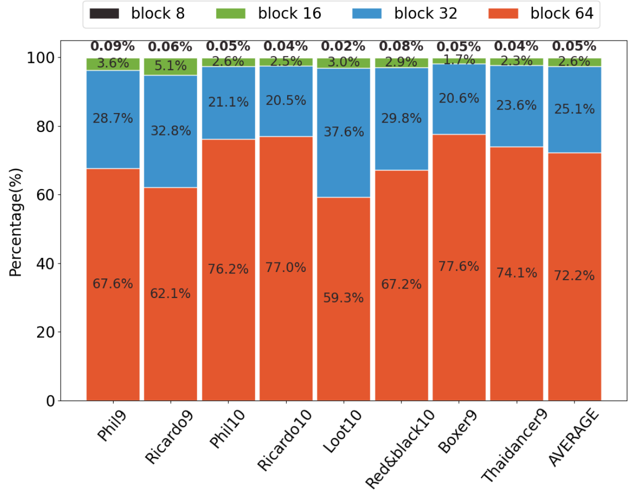


Fig. 4 Percentage of encoded points in each block size. From top to bottom: block 8, 16, 32, 64.

5. CONCLUSIONS

This paper proposed a hybrid octree/voxel-based lossless compression method for point cloud geometry. It employs for the first time a deep generative model in the voxel space to estimate the occupancy probabilities sequentially. Combined with a rate-optimized partitioning strategy, the proposed method outperforms MPEG G-PCC with average 28% rate savings over all tested datasets. We are now working on improving lossless coding of PC geometry by using more powerful generative models, and jointly optimizing octree and voxel coding.

6. REFERENCES

[1] S. Schwarz, M. Preda, V. Baroncini, M. Budagavi, P. Cesar, P. A. Chou, R. A. Cohen, M. Krivokuca, S. Lasserre, Z. Li, J. Llach, K. Mammou, R. Mekuria, O. Nakagami, E. Siahaan, A. Tabatabai, A. M. Tourapis, and V. Zakharchenko, "Emerging MPEG Standards for Point Cloud Compression," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, pp. 1–1, 2018.

[2] E. S. Jang, M. Preda, K. Mammou, A. M. Tourapis, J. Kim, D. B. Graziosi, S. Rhyu, and M. Budagavi, "Video-Based Point-Cloud-Compression Standard in MPEG: From Evidence Collection to Committee Draft [Standards in a Nutshell]," *IEEE Signal Processing Magazine*, vol. 36, no. 3, pp. 118–123, May 2019.

[3] 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 Transactions on Signal and Information Processing*, vol. 9, 2020.

[4] D. C. Garcia and R. L. de Queiroz, "Context-based octree coding for point-cloud video," in *2017 IEEE International Conference on Image Processing (ICIP)*, September 2017, pp. 1412–1416, ISSN: 2381-8549.

[5] D. C. Garcia and R. L. d. Queiroz, "Intra-Frame Context-Based Octree Coding for Point-Cloud Geometry," in *2018 25th IEEE International Conference on Image Processing (ICIP)*, October 2018, pp. 1807–1811.

[6] 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 Transactions on Image Processing*, vol. 29, pp. 313–322, 2019.

[7] A. Dricot and J. Ascenso, "Adaptive multi-level triangle soup for geometry-based point cloud coding," in *2019 IEEE 21st International Workshop on Multimedia Signal Processing (MMSP)*. IEEE, 2019, pp. 1–6.

[8] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1, MIT press Cambridge, 2016.

[9] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel Recurrent Neural Networks," *arXiv:1601.06759 [cs]*, August 2016.

[10] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "PixelCNN++: Improving the PixelCNN with Discretized Logistic Mixture Likelihood and Other Modifications," *arXiv:1701.05517 [cs, stat]*, January 2017.

[11] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. V. Gool, "Conditional Probability Models for Deep Image Compression," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, June 2018, pp. 4394–4402, IEEE.

[12] 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.

[13] 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.

[14] A. F. R. Guarda, N. M. M. Rodrigues, and F. Pereira, "Point cloud geometry scalable coding with a single end-to-end deep learning model," in *2020 IEEE International Conference on Image Processing (ICIP)*, 2020, pp. 3354–3358.

[15] M. Quach, G. Valenzise, and F. Dufaux, "Learning Convolutional Transforms for Lossy Point Cloud Geometry Compression," in *2019 IEEE International Conference on Image Processing (ICIP)*, September 2019, pp. 4320–4324, ISSN: 1522-4880.

[16] M. Quach, G. Valenzise, and F. Dufaux, "Improved Deep Point Cloud Geometry Compression," in *arXiv:2006.09043 [cs, eess, stat]*, June 2020.

[17] 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 Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 1912–1920.

[18] 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.

[19] 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, January 2017.

[20] "Common test conditions for PCC," in *ISO/IEC JTC1/SC29/WG11 MPEG output document N19324*.

[21] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *2015 3rd International Conference on Learning Representations*, December 2014, arXiv: 1412.6980.