Point Cloud Coding: Adopting a Deep Learning-based Approach

André F. R. Guarda
Instituto Superior Técnico
Instituto de Telecomunicações
Lisbon, Portugal
andre.guarda@lx.it.pt

Nuno M. M. Rodrigues
ESTG, Instituto Politécnico de Leiria
Instituto de Telecomunicações
Leiria, Portugal
nuno.rodrigues@co.it.pt

Fernando Pereira
Instituto Superior Técnico
Instituto de Telecomunicações
Lisbon, Portugal
fp@lx.it.pt

***Abstract*—Point clouds have recently become an important visual representation format, especially for virtual and augmented reality applications, thus making point cloud coding a very hot research topic. Deep learning-based coding methods have recently emerged in the field of image coding with increasing success. These coding solutions take advantage of the ability of convolutional neural networks to extract adaptive features from the images to create a latent representation that can be efficiently coded. In this context, this paper extends the deep-learning coding approach to point cloud coding using an autoencoder network design. Performance results are very promising, showing improvements over the Point Cloud Library codec often taken as benchmark, thus suggesting a significant margin of evolution for this new point cloud coding paradigm.**

Keywords—point cloud coding, deep learning, convolutional neural network

I. Introduction

Interactive immersion has become a critical requirement for many visual based applications, where the user is expected not only to feel immersed in a very realistic environment but also to easily interact with it. This applies for a large range of applications from broadcasting and cultural heritage to personal communications and geographical information systems. This type of experiences requires accurate 3D visual data representation models, amongst which light fields and point clouds (PCs) are currently the best candidates; however, point clouds have major advantages in terms of interactivity and for this reason are addressed in this paper.

A point cloud (PC) is an unorganized set of points in 3D space. It is typically represented as a list of 3D coordinates (x, y, z), defining the geometry as points over a surface, which may have associated a list of attributes, such as RGB colors or normal vectors. This makes PCs a powerful 3D visual representation model, allowing easy access to multiple viewpoints of an object/scene in immersive and interactive multimedia applications. In particular, these characteristics make PCs an ideal visual representation format for virtual and augmented reality [1], [2]. PCs can include a significant amount of information, in order to reliably represent an object or scene through points in their surfaces – a high precision PC can easily have tens of millions of points, with potentially multiple attributes per point. Thus, to make PCs practically useful for relevant applications, efficient PC coding solutions are required. This has lead the most relevant standardization groups in this domain, JPEG and MPEG, to the decision to develop PC coding solutions. For example, MPEG issued a Call for Proposals on Point Cloud Compression (PCC) in 2017, for both static and dynamic objects, scenes and real-time environments [3]. Also, JPEG has an early stage PC coding activity within the JPEG Pleno project [4].

In this context, the objective of this paper is to propose a new static, lossy PC geometry coding approach based on deep learning, thus opening a new PC coding research branch. Deep learning has entered many research fields in the recent years, notably for visual data processing [5], achieving incredible performance results. This has also recently been the case for image coding, with very promising results [6], [7]. The proposed new static PC geometry coding solution adopts an autoencoder (AE) neural network design including only convolutional layers. The comparison with the most common benchmark, the Point Cloud Library (PCL) [8] codec, shows very encouraging RD performance results. This is particularly exciting when acknowledging the large margins of improvement available for this coding solution, which will be addressed in the future work section.

The remaining sections of this paper are organized as follows: Section II briefly reviews related work on PC coding, as well as deep learning-based image coding. Section III presents the proposed deep learning-based PC coding solution. Section IV assesses and discusses the performance of the proposed PC coding solution, and Section V concludes the paper and offers some directions for future work.

II. Background Work

This section briefly reviews some relevant background work on static PC geometry coding as well as deep learning-based image coding, since these are the deep-learning based visual coding solutions closer to PC coding. In the literature, PC geometry coding has taken different approaches: the most common are octree-based [9], [10], graph-based [11] and projection-based [12], [13]. Due to the unique characteristics of geometry data, octree-based coding became a popular coding method. To construct an octree, the bounding box is divided into eight cells, with each one being recursively divided, until the desired precision level is achieved [9]. The Point Cloud Library (PCL) codec [8] became the reference solution in PC geometry octree-based coding. In [11], an octree is firstly used to code a coarser version of the PC, and then a graph-based method is used to add additional, relevant detail. Whereas octree and graphs try to fully exploit the 3D geometry characteristics and the relations between close neighboring points, projection-based methods, convert the 3D coding problem into a 2D coding problem, for which the very efficient image and video coding standards developed along the past decades can excel. Following the Call for Proposals launched in 2017, MPEG has developed two PC coding standards. The so-called Geometry-based Point Cloud Compression (G-PCC) standard [14], [2] targets static objects, using an octree-based coding method up to a given precision level, and a surface-based coding method to add detail to each

This work was funded by Fundação para a Ciência e Tecnologia (FCT), Portugal, Ph.D. Grant SFRH/BD/118218/2016, by FCT/MEC through national funds and when applicable co-funded by FEDER – PT2020 partnership agreement under the project UID/EEA/50008/2019.

978-1-7281-4704-8/19/$31.00 ©2019 IEEE

octree voxel. The so-called Video-based Point Cloud Compression (V-PCC) standard [15], [13] is focused on dynamic PCs, and uses a projection-based approach, where the PC geometry is projected into 2D images, using multiple patches from different views, and then resorts to a video codec to code the 2D images, e.g. HEVC [16].

Regarding deep learning-based image coding, several solutions have been recently proposed in the literature. Due to its structure, AE are particularly suited for data coding, and several works on AE-based image coding using convolutional layers have emerged. Earlier solutions [6], [17] are based on recurrent neural networks (RNN), allowing variable bitrate and progressive coding by iteratively coding the residue, i.e. the difference between the original and decoded images. Whereas the previous works allocate beforehand a fixed number of bits for the code, and minimize only the reconstruction loss, the solutions in [18] and [19] introduce rate-distortion (RD) optimization during CNN training, allowing to improve the RD performance. In [7], a variational autoencoder was introduced within the underlying AE architecture, to estimate the probability distributions of the AE latent representation and better capture the dependencies in the latent representation, thus improving entropy coding. Whereas most of these works easily outperform JPEG, only [7] comes close to HEVC Intra, even outperforming it for some perceptual quality metrics. Given these promising performance results, this paper proposes to extend the deep learning-based coding paradigm to static PC geometry coding as described in detail in next section.

III. Proposed Point Cloud Geometry Coding Solution

This section proposes a novel static PC geometry coding solution based on a deep 3D-CNN AE architecture, in the following named as PC Geometry AutoEncoder (PCG-AE). Considering that recent research works on image coding with neural networks have shown very promising results, the coding of PC geometry with deep learning-based methods is certainly an appealing research approach. In this new coding paradigm, the PCG-AE may be understood as performing a transform over the input data in a similar way to the transform-based coding approaches, often used for image and video coding, e.g. JPEG [20], MPEG-X, H.264/AVC [21], HEVC [16]. However, instead of using a predefined transform, such as the DCT for images in the popular JPEG standard, a deep neural network is used to learn an eventually more adapted and efficient transform and its inverse. The proposed deep learning-based PC geometry coding solution (PCG-AE) is described in the following sections.

A. Overall PCG-AE Architecture

The overall architecture for the novel PCG-AE coding solution is presented in Fig. 1. The first important decision in the PCG-AE development was related to the adopted PC representation model. Instead of 3-tuple coordinate, i.e. (x, y, z), the PC is represented as a set of 3D blocks, each structured as a binary 3D voxel-based grid, where a '1' is placed in the filled voxels and a '0' is associated to the empty voxels. Very importantly, this PC representation format offers a uniform, organized structure for the PC, just as images and video in the past, now in the 3D domain. This allows to easily use neural networks equivalent to those used for image processing and coding, namely convolutional neural networks [5]. CNNs have been widely researched for image processing and have proven to be very effective and powerful in extracting useful features from neighboring pixels. In contrast, dealing with an unorganized point set in a neural network can be challenging, especially in a coding context where exploiting redundancies between neighboring points is essential.

By definition, an AE learns an analysis transform to represent the input data as *latents* with a reduced dimensionality (encoder), i.e. the transform coefficients, and a synthesis transform to decode the input data from the (likely quantized) latents, as close as possible to the original data (decoder). The encoder and decoder are trained together as a single network, being typically symmetric, though this is not mandatory. These characteristics make the AE architecture an ideal fit for data coding purposes, notably PC coding.

At the PCG-AE encoder side, the full PC is divided into 3D blocks, and each one is transformed with the AE encoder, obtaining the *latent representation*. Afterwards, the latent representation is quantized and entropy coded to generate the final bitstream. At the decoder side, the bitstream is decoded and the dequantized latent representation is obtained, which is then transformed back into a 3D block via the AE decoder and, finally, all 3D blocks are merged to generate the full reconstructed PC. Naturally, this novel PC coding paradigm assumes the availability of an AE network which has been previously trained to learn an AE model, i.e. the appropriate filter weights, thus defining the CNN-based transforms to be applied in the coding and decoding processes.

B. Division into Blocks & Binarization and Merging

Due to the nature of PCs, most voxels in the full PC bounding box are empty. In this context, representing a PC in a single, full resolution volumetric grid can be impracticable, due to the associated memory and computational requirements. Thus, dividing the PC into smaller 3D blocks is a natural option, akin to coding units as used for image and video coding in the past.

The *Division into 3D Blocks* module receives a voxelized PC, eventually with millions of points, and divides it into 3D coding units with the same size, here 32×32×32, where each unit is a 3D block of binary voxels. Since each 3D block does not explicitly contain information on its original position in the overall PC, an index is created denoting its position; this index is coded and multiplexed as side information in the bitstream. Logically, only non-empty 3D blocks are considered for coding. The *Binarization and Merging* counterpart module at the decoder performs the inverse task. Each decoded 3D block is converted into the coordinates' representation format by firstly binarizing each voxel, i.e. rounding to '0' or to '1' with a threshold of 0.5, to obtain the reconstructed binary 3D block. Then the filled voxels are placed in their corresponding positions, thus effectively merging the points of all 3D blocks to reconstruct the full PC.

C. 3D-CNN AE Encoding & 3D-CNN AE Decoding

The detailed architecture of the AE shaped 3D-CNN in the proposed PCG-AE coding solution is represented in Fig. 2. The transform and its inverse operations are learned with a CNN with a rather basic design. It consists of a symmetric autoencoder in which all layers are convolutional, four at the encoder side and four at the decoder side. In this AE, the convolutional layers are the natural extension of the 2D convolutional layers typically used in 2D-CNN image processing. While other, more sophisticated, CNN-based architectures may be used, this paper intends to propose a rather basic CNN-based architecture as a proof-of-concept for

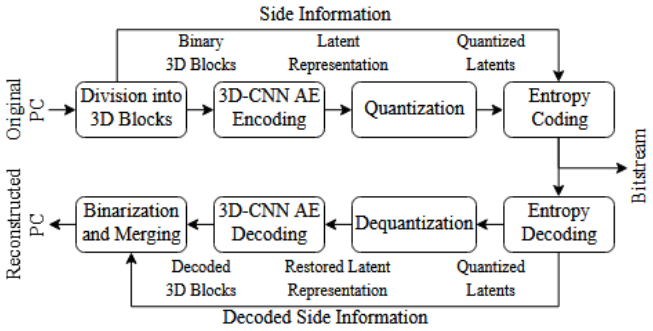

Fig. 1. Overall architecture of the proposed PCG-AE coding solution.The decoder (bottom) mirrors the encoder (top).

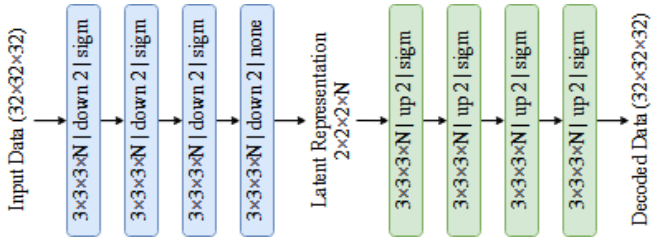

Fig. 2. Architecture of the PCG-AE deep 3D-CNN autoencoder. All layers are convolutional with the following hyper-parameters: filter support width × filter support height × filter support depth × number of filters (N) | down or up-sampling stride | activation function.

PC geometry coding. After showing that this initial solution has a quite promising performance, future developments should enhance the neural network and other modules in the overall coding architecture.

A convolutional layer may be characterized by a set of hyper-parameters, notably the filter support width (W), height (H) and depth (D), the number of filters per layer (N) and the (down or up-sampling) stride (S). Thus, a convolutional layer corresponds to a group of N filters, with each filter associated to a set of learnable parameters. The filter support with shape $W \times H \times D$, also known as *receptive field*, is convolved with the input volume to generate a feature map. In practice, the filter support slides along the input volume, computing the dot product at every position to extract features based on a point's neighborhood. By setting $S > 1$, filtering is applied at only one out of every S positions, thus reducing the feature maps resolution compared to the input resolution. For example, using a stride of two, the filter is only applied at every other position, reducing the resolution to half. This is important to obtain a latent representation with lower dimensionality. The number of filters N parameter plays a critical role as it determines the size of the latent representation; this has a major impact on the reconstruction quality and associated coding rate, and therefore on the final RD performance.

In the proposed PCG-AE coding solution, each layer in the AE encoder down-samples the successively created feature maps by a factor of two along all three directions, via the filter stride; conversely, the layers at the decoder perform the corresponding up-sampling. Thus, after the encoder, the latent representation of an input 3D block of size 32×32×32 consists on N blocks of size 2×2×2. The filter support was defined as 3×3×3 for every layer since this was found a good initial solution. A sigmoid activation function is used after each layer, with the exception of the last encoder layer that generates the latent representation to be coded. In the last decoder layer, the sigmoid activation function fixes the range of the output to be between zero and one. The decoder task is thus to transform the latent representation, which has been quantized, back to the PC coding unit, i.e. the 3D block.

Fig. 3. Test PCs: (a) *Statue Klimt*; (b) *Queen*; (c) *Longdress*; (d) *House without Roof*.

D. Quantization & Entropy Coding

After the AE encoder transforms each input 3D block into a set of real floating-point latent values, quantization is applied to obtain integers. The quantization strength can be set by choosing the desired number of bits per latent value, here designated as the *quantization parameter QP*. The quantization inherently introduces some error which will impact the ability of the AE decoder; in practice, the quantization parameter allows defining the rate-quality trade-off. While more sophisticated quantization solutions may be designed in the future, this basic one is already powerful enough to show the potential of this 3D-CNN AE PC coding paradigm.

The quantized latent representation of all 3D blocks is finally entropy coded, in this case into a single 7z file, i.e. the bitstream, which also includes some side information conveying the 3D blocks positioning in the overall PC. This information is sufficient for the AE decoder to decode each 3D block and for appropriately merging them to reconstruct the full PC. Again, more sophisticated entropy coding solutions are expected to be designed in the future.

IV. Performance Assessment

In this section, the performance of the proposed PCG-AE coding solution is evaluated and discussed. Before presenting the results, the setup, conditions and methodology for the performed coding experiments are described.

A. Datasets

The performance of PCG-AE was evaluated on PCs from the MPEG dataset [22], used for the MPEG PCC standardization process. The dataset contains people, static objects, such as statues and buildings, and also some synthetic content. From this dataset, four PCs were selected for testing (*Statue Klimt, Queen, House without Roof* and *Longdress,* see Fig. 3). The remaining PCs were used to train the AE models. For training, PCs were divided into 3D blocks of 32×32×32, as described in Section III.B. Blocks with a very low point count, i.e. less than 500 points, were discarded, since they contain very little relevant geometry information to be learned. Out of the remaining blocks, 6000 were chosen randomly for training. When appropriate, the PCs were down-sampled to a precision of 9 or 10 bits, as shown in Table I, in order to reduce the sparsity of PCs with lower point density.

B. Training Setup

Prior to using the PCG-AE codec to code the test PCs, the AE model must be trained, so it can learn adequate (analysis

TABLE I. MPEG DATASET PCS [22].

Point Cloud Name	# Points	Precision (bit)	Split
Egyptian Mask	269,723	9	Training
Arco Valentino	606,004	9	
Façade 00009	889,698	10	
Frog	1,580,392	10	
Façade 00015	831,485	10	
Façade 00064	1,084,875	10	
ULB Unicorn	243,839	10	
Loot	805,285	10	
Red and Black	757,691	10	
Soldier	1,089,091	10	
Shiva	900,331	10	
Palazzo Carignano	731,342	9	
Head	3,453,846	10	
Statue Klimt	243,839	9	Testing
Queen	1,000,993	10	
House without Roof	1,724,175	10	
Longdress	857,986	10	

and synthesis) transforms to efficiently reduce the data dimensionality. Besides the choice of training material, made as described above, another critical training component is the loss function. During training, the key objective is to learn the AE model parameters that minimize the error between the original and decoded 3D blocks. Given that the input data is binary, and the AE decoder output is a real value between 0 and 1, the Binary Cross Entropy (BCE) has been adopted as loss function. However, due to the highly unbalanced number of '1' and '0' in the 3D blocks, namely less than 5% of '1's per 3D block, on average, for both the training and test material, a biasing weight is introduced. The weighted BCE (wBCE) loss function is, thus, defined as:

$$wBCE = -\frac{1}{K}\sum_{K}(\beta x \log(y) + (1-x)\log(1-y)), \quad (1)$$

where x is the original binary voxel value ('1' or '0'), y is the respective decoded floating-point voxel value, $\beta > 1$ is the weight giving a higher importance to a false negative ('1' becoming a '0') relative to a false positive ('0' becoming a '1'), and K is the total number of voxels in a 3D block (32,768 voxels). When the original voxel is '0', only the right term in (1) contributes to the loss, notably increasing as the decoded voxel approximates '1'. When the original voxel is '1', only the left term in (1) contributes to the loss, in this case amplified by the chosen weighting factor. If the weight is too small, the loss function minimization might fall into a local minimum and the network would simply learn to decode an all-0 block, due to the unbalanced number of '0' and '1' and the error averaging for all voxels in a 3D block. On the other hand, if the weight is too large, the network may learn to decode an all-1 block. Experiments found that a weight between 30 and 35 was appropriate for the adopted test conditions.

As for the training configuration, since the number of filters N is a key hyper-parameter for the PCG-AE RD performance, four different AE models were trained using different values of N, namely 32, 64, 96 and 128 filters. All models were trained with stochastic gradient descent using the Adam algorithm [23] with learning rate of 10^{-4}, and minibatches of 8 3D blocks at a time.

C. Test Setup

For each of the four trained AE models, the corresponding PCG-AE codec was tested with multiple quantization steps, by varying the quantization parameter QP from 2 to 16 bits per latent value. The reconstruction quality can then be obtained for different bitrates, determined by N and QP, to generate a RD curve for each AE model, thus allowing the easy comparison of different AEs.

D. Performance Metrics and Benchmarking

To compare the RD performance of the various PCG-AE model variations (associated to different N values), and also with the selected PCL benchmark, both the bitrate and geometry quality are measured and represented in a RD chart. The bitrate is evaluated as the total number of bits in the bitstream divided by the total number of points in the original point cloud (this means '1' in the 3D blocks or occupied voxels), as shown in Table I; this measure is designated as bits-per-(occupied)point (bpp). As for the geometry quality, the point-to-point distance metric referred as D1 in MPEG PCC standardization process [22] is used. This metric is based on the mean squared error (MSE) associated to the distances between the original PC points and the closest points in the reconstructed PC, and vice-versa, since this is a symmetric metric. The D1 PSNR is then computed as:

$$PSNR = 10\log(3p^2/MSE), \quad (2)$$

where p is the peak constant value, which depends on the PC precision and is given as $p = 2^{\text{precision}} - 1$, and MSE is the maximum of the two computed MSE (original to reconstructed PCs and vice-versa).

For benchmarking purposes, the proposed PCG-AE RD performance is compared with the PCL (briefly described in Section II) RD performance; this was also the coding benchmark used for the MPEG PCC development. The test PCs were coded with PCL, starting from the original precision shown in Table I and successively reducing it by one level, until four RD points were obtained. No comparisons are yet shown with the MPEG G-PCC standard codec since its reference software is not publicly available.

E. RD Performance

The RD charts for each of the test PCs are shown in Fig. 4. Considering only the PCG-AE codec variations, it can be concluded that, given enough bitrate, using more filters (N) allows reaching higher quality, eventually even qualities that are too high to be perceptually relevant. While the maximum qualities achieved with 128 and 96 filters are quite similar, there is a drop-off for the AE models with 32 and 64 filters. However, for lower bitrates, a lower number of filters improves the quality and obtain better RD points.

When comparing PCG-AE to PCL, for the medium and higher bitrates/qualities, the PCG-AE RD curves consistently outperform the PCL RD curves. However, PCL still outperforms PCG-AE for the very low bitrates, hinting that a better usage of the rate has to be made, which may be achieved by adopting a more sophisticated loss function. In general, results are similar for all PCs, although the RD gains over PCL are more accentuated for *Queen* and *Longdress*.

Since the number of reconstructed points has typically a major impact on the subjective quality, Fig. 5 shows the ratio between the number of reconstructed points and the number of original points for each PC. This ratio may be rather low for PCL since it essentially lowers the rate by down-sampling the PC, thus largely reducing the number of points; this results in reconstructed PCs with a very low point count and a significant quality loss. On the contrary, the proposed PCG-AE tends to reconstruct more points than the original PC, avoiding the subsampling subjective impression; this ratio

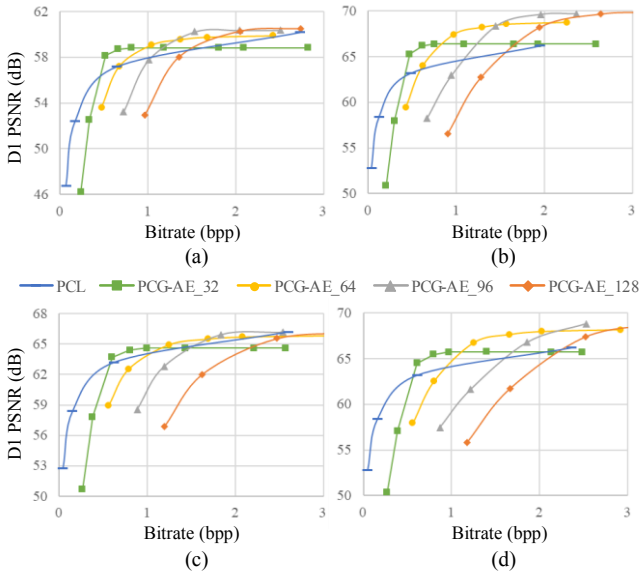

Fig. 4. RD performance comparison between the proposed PCG-AE and the PCL benchmark for different number of filters *N*. (a) *Statue Klimt*; (b) *Queen*; (c) *House without Roof*; (d) *Longdress*.

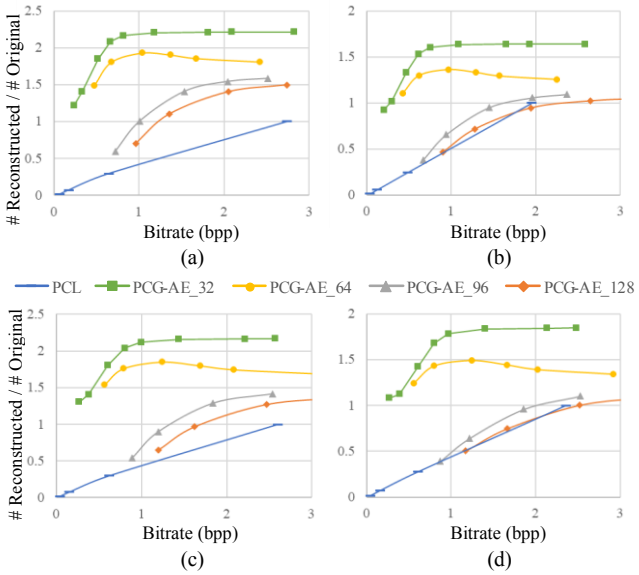

Fig. 5. Ratio between the number of reconstructed and original points for the proposed PCG-AE and the PCL benchmark for different number of filters *N*. (a) *Statue Klimt*; (b) *Queen*; (c) *House without Roof*; (d) *Longdress*.

may be optimized both by controlling the loss function weight and the voxel binarization threshold.

While these results do not fully overcome the state-of-the-art performance for static PC geometry coding for all rates/qualities, they clearly demonstrate the potential of deep learning-based approaches for PC coding, since a relatively basic CNN architecture was able to compete with and even outperform the common PCL benchmark. This is very promising and exciting in terms of future PC coding research.

V. Conclusions and Future Work

This paper proposes a deep learning-based approach to static PC geometry (lossy) coding, thus opening a new branch in this hot research area. Considering the very promising performance results and the relative simplicity of the CNN based solution, it may be expected that significant RD gains may be obtained in the future, thus further raising the competitiveness of this novel PC coding paradigm. Future work has multiple possible research directions. Naturally, the CNN architecture may become more sophisticated. A better loss function, eventually RD performance based, may be adopted. A more efficient, non-uniform, quantization scheme, together with more adapted entropy coding solutions may be developed, for the compression of the latent representation. In summary, a new PC geometry coding paradigm has emerged, which has good chances to seriously compete with the already available octree, patch and graph-based PC coding paradigms.

References

[1] K. Sugimoto et al., "Trends in efficient representation of 3D point clouds," APSIPA Annu. Summit Conf. (APSIPA ASC), Kuala Lumpur, Malaysia, Dec. 2017.

[2] S. Schwarz et al., "Emerging MPEG standards for point cloud compression", IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 9, no. 1, pp. 133–148, Mar. 2019.

[3] MPEG, "Call for proposals for point cloud compression v2," Doc. MPEG N16763, Hobart, Australia, Apr. 2017.

[4] T. Ebrahimi *et al.*, "JPEG Pleno: Toward an efficient representation of visual reality," in IEEE MultiMedia, vol. 23, no. 4, pp. 14-20, Oct.-Dec. 2016.

[5] M. Z. Alom et al., "The history began from AlexNet: a comprehensive survey on deep learning approaches," arXiv:1803.01164, Sept. 2018.

[6] G. Toderici et al., "variable rate image compression with recurrent neural networks," Int. Conf. on Learning Representations, San Juan, Puerto Rico, May 2016.

[7] J. Ballé et al., "Variational image compression with a scale hyperprior," Int.Conf. on Learning Representations, Vancouver, Canada, Apr.2018.

[8] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," IEEE Int. Conf. Robot. Automat., Shanghai, China, 2011.

[9] R. Schnabel, and R. Klein , "Octree-based point-cloud compression," in IEEE VGTC Conf. Point-Based Graph., Boston, USA, July 2006.

[10] R. Mekuria et al., "Design, implementation, and evaluation of a point cloud codec for tele-immersive video," IEEE T-CSVT, vol. 27, no. 4, pp. 828-842, Apr. 2017.

[11] P.Rente *et al.*, "Graph-based static 3D point clouds geometry coding," in IEEE T-MM, vol. 21, no. 2, pp. 284-299, Feb. 2019.

[12] H. Houshiar and A. Nüchter, "3D point cloud compression using conventional image compression for efficient data transmission," Int. Conf. Inf. Commun. Automat. Technol., Sarajevo, B&H, 2015.

[13] E. S. Jang et al., "Video-based point-cloud-compression standard in MPEG: from evidence collection to committee draft [Standards in a nutshell]," IEEE Signal Process. Mag., vol. 36, no. 3, pp. 118-123, May 2019.

[14] Moving Picture Experts Group, "ISO/IEC CD 23090-9 Geometry-based point cloud compression", Geneva Meeting, Mar. 2019.

[15] Moving Picture Experts Group, "ISO/IEC CD 23090-5 Video-based point cloud compression", Macau Meeting, Oct. 2018.

[16] G. J. Sullivan et al., "Overview of the High Efficiency Video Coding (HEVC) Standard," IEEE T-CSVT, vol. 22, no. 12, pp. 1649-1668, Dec. 2012.

[17] G. Toderici et al., "Full resolution image compression with recurrent neural networks," IEEE Conf. Comput Vision and Pattern Recognit. (CVPR), Honolulu, HI, USA, July 2017.

[18] L. Theis et al., "Lossy image compression with compressive autoencoders," Int.Conf. on Learning Representations, Toulon, France, Apr. 2017.

[19] J. Ballé et al., "End-to-end optimized image compression," Int. Conf. on Learning Representations, Toulon, France, Apr. 2017.

[20] G. K. Wallace, "The JPEG still picture compression standard," Commun. ACM, vol. 34, pp. 30-44, Apr. 1991.

[21] T. Wiegand et al., "Overview of the H.264/AVC video coding standard," IEEE T-CSVT, vol. 13, no. 7, pp. 560-576, Jul. 2003.

[22] S. Schwarz et al., "Common test conditions for point cloud compression," Doc. MPEG N17345, Gwangju, Korea, Jan. 2018.

[23] D. P. Kingma and Jimmy Ba, "Adam: a method for stochastic optimization", in Int. Conf. on Learning Representations, San Diego, USA, May 2015.