Adaptive Deep Learning-Based Point Cloud Geometry Coding

André F. R. Guarda, *Student Member, IEEE*, Nuno M. M. Rodrigues, *Member, IEEE*, and Fernando Pereira, *Fellow, IEEE*

***Abstract*—Point clouds are a very rich 3D visual representation model, which has become increasingly appealing for multimedia applications with immersion, interaction and realism requirements. Due to different acquisition and creation conditions as well as target applications, point clouds' characteristics may be very diverse, notably on their density. While geographical information systems or autonomous driving applications may use rather sparse point clouds, cultural heritage or virtual reality applications typically use denser point clouds to more accurately represent objects and people. Naturally, to offer immersion and realism, point clouds need a rather large number of points, thus asking for the development of efficient coding solutions. The use of deep learning models for coding purposes has recently gained relevance, with latest developments in image coding achieving state-of-the-art performance, thus making natural the adoption of this technology also for point cloud coding. This paper presents a novel deep learning-based solution for point cloud geometry coding which is able to efficiently adapt to the content's characteristics. The proposed coding solution divides the point cloud into 3D blocks and selects the most suitable available deep learning coding model to code each block, thus maximizing the compression performance. In comparison to the state-of-the-art MPEG G-PCC Trisoup standard, the proposed coding solution offers average quality gains up to 4.9 and 5.7 dB for PSNR D1 and PSNR D2, respectively.**

***Index Terms*—Point cloud coding, deep learning, adaptive models.**

I. Introduction

POINT clouds (PCs) have become a very appealing 3D visual representation approach in recent years due to their characteristics, notably easy acquisition and processing; with appropriate rendering before visualization, they are very appealing for immersive, interactive and realistic user experiences. In this context, PCs are a competitive alternative to meshes, which include connectivity information, and thus require additional storage and processing, although they may be more rendering friendly. For these reasons, PCs are at the forefront of modern multimedia applications such as virtual and augmented reality. A PC consists of a set of points in the 3D space, represented by their 3D coordinates, which are referred to as the *geometry*. To provide a more realistic representation, each point may have associated multiple *attributes*, notably color, normal vectors and reflectance. Depending on the target application, PCs may or may not vary in time, corresponding to the so-called *dynamic* and *static* PCs, respectively, analogous to video and image in the 2D world. Naturally, the number of points in a PC severely impacts the final user visual experience, eventually even more than the spatial resolution of 2D images, since PCs aim at highly immersive and realistic experiences. This need for PCs with a very high number of points, thus implying a huge amount of raw data, makes PC efficient compression a must in order for storage and transmission-based applications to be largely deployed.

The unique characteristics of PCs have created new coding challenges [1]. For instance, the unstructured nature of PC data differs from the traditional image and video data, where pixels totally fill a uniform 2D grid; this makes it more difficult to find and exploit the spatial and temporal correlation between the irregular 3D points. In addition, the acquisition process may impact the PC characteristics, notably their density, meaning that both very dense and very sparse PCs may have to be coded. This heterogeneity must be acknowledged and accounted by the coding solution since it may otherwise be difficult to reach high compressions for all type of PCs with a single PC codec.

The MPEG and JPEG standardization groups have recognized the appeal and practical relevance of the PC representation model as well as the need to offer efficient coding standards to the industry. MPEG has already developed two PC compression (PCC) standards [2]–[4]: one for dynamic content, referred to as Video-based PCC (V-PCC), and another for static content, referred to as Geometry-based PCC (G-PCC). While G-PCC has adopted an octree-based coding approach complemented with a triangle-based surface representation, V-PCC has adopted a patch-based approach where the 3D data is projected into texture, depth and occupancy maps, which are coded with available 2D coding solutions. As for JPEG, it recently launched (July 2020) a Call for Evidence on PC coding [5], focusing on scalability and random access requirements.

But times are changing also in the coding arena… Deep learning (DL) has become massively popular in recent years, spanning a multitude of applications across many different areas [6], propelled by the large-scale availability of data and the

Manuscript received July 1, 2020; revised October 15, 2020 and December 19, 2020; accepted December 21, 2020. Date of publication December 25, 2020; date of current version February 22, 2021. This work was supported in part by Fundação para a Ciência e Tecnologia, Portugal, Ph.D. Grant SFRH/BD/118218/2016, and in part by FCT/MEC through national funds and co-funded by EU funds under Project UIDB/EEA/50008/2020. *(Corresponding author: André F. R. Guarda.)*

André F. R. Guarda and Fernando Pereira are with the Instituto Superior Técnico, Universidade de Lisboa and Instituto de Telecomunicações, 1049-001 Lisboa, Portugal (e-mail: andre.guarda@lx.it.pt; fp@lx.it.pt).

Nuno M. M. Rodrigues is with the ESTG – Politécnico de Leiria and Instituto de Telecomunicações, 2411-901 Leiria, Portugal (e-mail: nuno.rodrigues@co.it.pt).

Digital Object Identifier 10.1109/JSTSP.2020.3047520

advances on hardware, namely Graphics Processing Units (GPU). In particular, DL has taken over the field of computer vision, with a dominating performance in most tasks [7]. Part of its success may be attributed to the use of convolutional neural networks (CNN), which learn to identify and extract useful features for a specified task, as opposed to simply extracting handcrafted features as before. More recently, DL-based image coding has become a hot research topic, with the latest results already competing with state-of-the-art image codecs. Acknowledging the potential of DL for image coding, JPEG has recently issued (January 2020) a Call for Evidence for learning-based image coding [8] and is planning a Call for Proposals on April 2021, to follow with the design of the so-called JPEG AI standard. Given its performance for image coding, it was only natural for DL-based coding to expand to other data domains, one of which being PC coding.

With this in mind, this paper proposes a deep learning-based static PC geometry coding solution, in the following designated as Adaptive Deep Learning-based Point Cloud Coding (ADL-PCC) solution. This novel coding solution efficiently addresses the challenge of PC content diversity, namely the different density between PCs or even within the same PC by adaptively selecting the DL model, from those available, which best suits a particular region/block of a PC. In this way, the proposed ADL-PCC solution can effectively adapt to any generic PC content, achieving more efficient and competitive compression performance. In fact, ADL-PCC offers significant compression gains over the natural benchmark, this means the G-PCC Trisoup, notably a BD-PSNR up to 4.9dB (5.7dB) for PSNR D1 (PSNR D2) metric, on average, for a representative PC dataset. Moreover, these compression gains do not arise at the cost of encoding and decoding complexity penalties regarding G-PCC Trisoup. These advantages highlight that DL-based PC coding may be a key technology for future PC-based applications.

To achieve its objectives, this paper is organized as follows: Section II reviews the background on DL-based coding and the state-of-the-art in PC coding. Section III describes the proposed ADL-PCC solution, and Section IV describes its associated training process. Section V presents and discusses the experimental results. Finally, Section VI concludes the paper.

II. Background Work Review

This section reviews the background work relevant to the proposed ADL-PCC solution, notably the state-of-the-art in PC coding, followed by the key developments in DL-based coding, both in the image and PC domains.

A. Point Cloud Coding

A natural way to represent 3D geometry is by using an octree structure, where the 3D space is recursively partitioned into octants, and a binary notation is adopted to represent the occupied and empty octants. Currently, most popular PC coding solutions are based on octrees, including the Point Cloud Library (PCL) codec [9]. An extension for dynamic PCs has been proposed in [10], where the octree difference between frames is coded to exploit the temporal redundancy. In [11], the frames of a dynamic PC are divided into blocks, and each block is coded with one of two modes in a choice based on rate-distortion (RD) minimization: intra mode, using an octree, or inter mode, simply using a motion vector to create a prediction without any residual coding. The MPEG G-PCC standard [2], which represents the state-of-the-art in geometry-based PC coding methods, also uses an octree-based coding approach. When the goal is lossless compression, G-PCC builds the octree until reaching the maximum depth of the voxelized PC; for lossy compression, simply pruning the octree at early levels results in a severe loss of points, and thus poor quality. Thus, the pruned octree is complemented with a surface-based approach called *Trisoup* to obtain further detail and improve the quality within each octree leaf block. With Trisoup, the surface formed by the points inside each leaf block is estimated by determining its intersection with each of the block's edges. Then, the surface is reconstructed as a set of triangles defined by those intersection points.

An alternative to the natural 3D geometry coding approach is to convert the 3D coding problem into a 2D coding problem by projection or mapping the 3D data onto a 2D plane [12], [13]. This way, the well-established image and video codecs may be used after to efficiently compress the resulting 2D images. The MPEG V-PCC standard [4] represents the state-of-the-art in projection-based PC coding solutions. V-PCC starts by segmenting the PC into patches, according to the direction of the normal vector associated with each point. The 3D patches are then projected onto the 2D planes associated to the PC bounding box and arranged/packed as 2D images. The geometry information is represented as depth images, while the color information corresponds to a regular texture image. Binary occupancy maps are also generated to signal the 2D map pixels corresponding to actual 3D points, in order to be able to reconstruct the PC. The 2D images may be then coded by any image/video codec/standard, notably the very efficient HEVC video coding standard [14] or even the emerging Versatile Video Coding (VVC) standard [15].

B. Deep Learning-Based Coding

Before PCs, DL-based coding has first targeted images, notably using neural networks based on CNNs; in this context, the autoencoder (AE) design was the natural starting point [16]. An AE consists of a (often symmetric) combination of encoder and decoder DL networks. The encoder DL network transforms the input data into a so-called *latent representation*, which is designed so that the decoder DL network can reconstruct a version of the original input as accurately as possible. Since compression is targeted, the AE imposes a bottleneck to reduce the dimensionality of the latent representation, forcing the network to extract important features and generate an expressive latent representation. The key difference regarding traditional transform-based image coding is that instead of using a transform with fixed basis functions, the AE learns a transform that is more suitable to the target data. Some earlier works have also used recurrent neural networks (RNN) [17], [18]. In these works,

a residual AE successively and progressively encodes the residue of the image at each iteration to allow achieving a target rate and quality depending on the number of iterations. In [19], [20], the DL coding model is trained/optimized using a RD loss function, to allow obtaining different target rates and qualities depending on the selected RD trade-off during training. The solution in [20] was later extended by introducing a variational autoencoder (VAE) to estimate the entropy coding parameters used by the entropy coder [21]; this solution was further improved by adding an autoregressive component [22].

After successfully applied to image coding, notably achieving close or even above state-of-the-art performance as demonstrated in the JPEG AI Call for Evidence [23], DL-based solutions started to be adopted for PC geometry coding. In [24], a point-based AE has been proposed to encode the PC geometry coordinates directly. Since points are unordered and unstructured, this solution only extracts global features, thus implying that it is basically only capable of efficiently coding PCs similar to those used for training, without much generalization capabilities. On the other hand, by adopting a volumetric representation for the PC, e.g. a 3D block, the image coding solutions based on AE CNN [20] could be extended to PC coding [25]– [27]. In [26], [27], the latents are explicitly quantized, using a quantization step to perform RD control, followed by entropy encoding. In [25], RD optimization is performed during training via the loss function, achieving average Bjontegaard-Delta rate (BD-Rate) savings over 50%, compared to a PC coding solution used as anchor for the MPEG G-PCC development [28], derived from PCL [9]. The same authors have improved their solution in [29] by adopting a PC partitioning scheme similar to [26], [27], while also using a deeper AE transform architecture, including the entropy coding approach proposed in [21], and optimizing the binarization threshold at the decoder to better adjust the number of reconstructed points per block for different PC densities. For this improved solution, average Bjontegaard-Delta PSNR (BD-PSNR) gains of 5.5dB (6.5dB) over MPEG G-PCC Trisoup have been obtained for the PSNR D1 (PSNR D2) metric, notably for people PCs. In [30], a solution using a more complex transform is proposed, notably with a Voxception-ResNet structure, combined with an extended entropy coding model [22]; this solution also improves the binarization by selecting the top-k voxels that are more likely to be occupied, to more closely match the number of input points per block. Furthermore, to achieve lower bitrates, [30] proposes to down-sample the PC beforehand using some appropriate scale factor. In [31], explicit quantization is used for RD control, targeting to reduce the number of trained DL models while still achieving significant rate savings over MPEG G-PCC. In [32], the authors study the joint coding of geometry and color attributes using a single DL coding model by stacking the color data to the geometry volumetric representation in separate channels.

The proposed ADL-PCC solution brings significant technical novelty (and performance improvements) regarding the literature, not only on the DL coding model design but mainly on the adaptive coding approach, which together allow achieving competitive RD performance for any type of PCs.

III. Deep Learning-based Static Point Cloud Coding: an Adaptive Solution

This section describes the proposed adaptive DL-based static PC geometry coding solution; this solution is inspired on the successful DL-based coding solution proposed by Ballé *et al.* [21] for (2D) image coding, here extended and adapted to (3D) PCs. After a description of the adopted PC representation data structure, the overall coding architecture and walkthrough are presented, followed by a detailed description of all its modules.

A. Point Cloud Representation Data Structure

Point cloud geometry is usually represented as an unordered set of 3D point coordinates. Some neural networks proposed in the literature, e.g. based on PointNet [33], are directly applied with success to the 3D point coordinates, predominantly to address classification and segmentation problems. These networks typically extract global features, which are mainly useful when targeting such applications. However, when the goal is PC coding, the performance of these methods decays, as local features become of the utmost importance, e.g. to represent details and achieve good fidelity/quality levels.

Convolutional layers are able to take advantage of the regular structure of images' samples/pixels to extract local features [7]. This is not the case for PCs, due to their unordered and unorganized nature, i.e. there are irregularly 'filled' and 'empty' voxels, which makes it more difficult to exploit the spatial correlation between neighboring 3D points. In this context, the proposed ADL-PCC solution adopts a binary voxel-based PC representation data structure, where the 3D space is discretized and voxels containing points are labeled with a '1' value, i.e. classified as 'filled', while the remaining voxels are labeled with a '0' value, i.e. classified as 'empty'. This means that the 'empty space' is also explicitly represented and not only the 'filled' positions. This representation data structure has the advantage of creating 3D blocks with regular size and a binary grid aligned signal, thus allowing to use coding approaches and DL-based architectures that would not be possible otherwise [34].

B. Overall Architecture and Walkthrough

The overall coding architecture of the proposed ADL-PCC solution is presented in Fig. 1a. This solution adopts an end-to-end DL coding model as shown in Fig. 1b. The key coding component is an autoencoder (AE), which learns a signal adaptive transform, followed by discretization and entropy encoding the resulting latents. A variational autoencoder (VAE) is used in addition to estimate the parameters used for entropy coding [21]. All these modules are trained together in an end-to-end fashion to learn the so-called *DL coding model*.

The main novelty of this paper is the extension of a 2D DL-based coding architecture [21] to 3D PCs and its usage in an adaptive way to match the local PC characteristics. In fact, due to massive differences in PCs characteristics, e.g. density, a single DL coding model may not be efficient for every PC (3D) block. By training multiple DL coding models with each one targeting

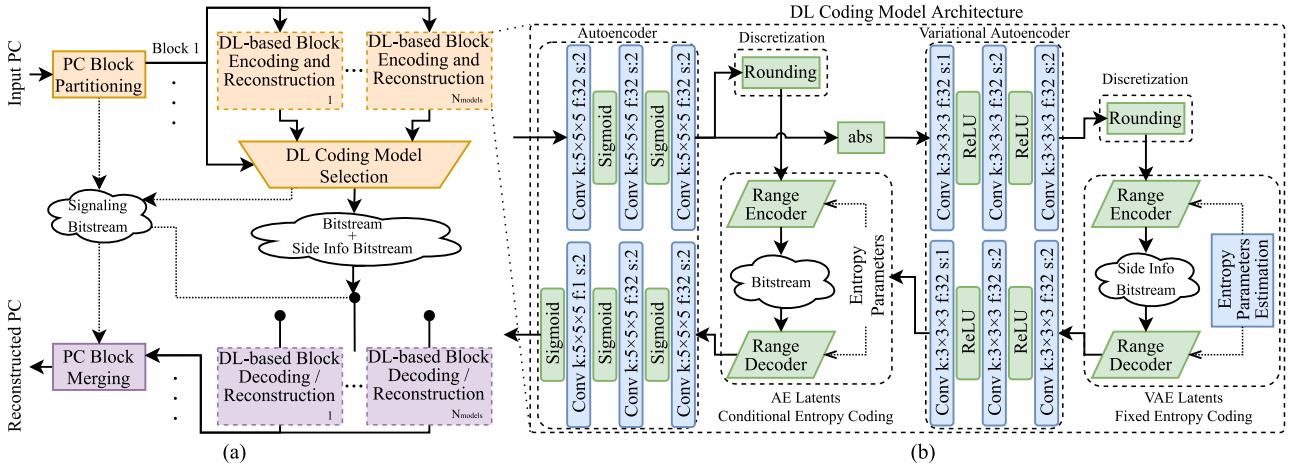


Fig. 1. Adaptive DL-based PC coding (ADL-PCC) solution: (a) overall architecture using N_{models} DL Coding Models; Orange blocks correspond to the encoder, and purple blocks correspond to the decoder; (b) DL coding model architecture; Convolutional layers are defined by the number of kernels (k), filters (f) and stride (s); Blue blocks use trainable parameters.

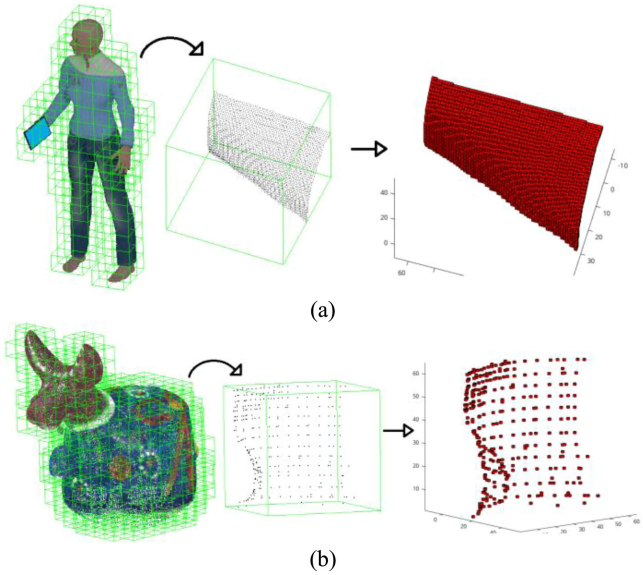


Fig. 2. PC block partitioning: the PC is divided into blocks, which are then represented as $64 \times 64 \times 64$ binary voxels. (a) Example of a dense PC (*Queen*) and a block with 4573 filled points; (b) Example of a sparse PC (*Bumbameuboi*) and a block with 319 filled points.

different PC characteristics, e.g. by adjusting the training loss function parameters, the usage of different DL models for each block type may be optimized. This way, each PC block may be coded using one from several available DL models, which behave like coding modes as in traditional coding, chosen in order for ADL-PCC to maximize the compression performance for PCs with different characteristics, both globally (PC-level) and locally (block-level). The encoding process, shown in Fig. 1, is summarized as follows:

- **PC block partitioning** - The complete PC is first partitioned into regular sized 3D blocks of voxels;
- **DL-based block encoding and reconstruction** - Each block is separately coded and reconstructed with the several available trained DL coding models, whose common architecture is detailed in Fig. 1b; this module includes the following sub-modules: Autoencoder, Discretization, AE Latents Conditional Entropy Coding, Variational Autoencoder, and VAE Latents Fixed Entropy Coding;
- **DL coding model selection** - The reconstructed block for each DL coding model is evaluated to select the model which produces the best compression performance, i.e. adapts better to the specific block characteristics; the selected model is signaled in the bitstream to the decoder.

At the decoder side, the decoding process proceeds as follows:

- **DL-based block decoding/reconstruction** - Blocks are reconstructed using the corresponding/signaled DL coding models, with the decoder counterpart of the sub-modules mentioned above for the DL-based block encoder;
- **PC block merging** - The reconstructed blocks are converted into 3D point coordinates and merged into a full PC represented as a list of points, which is the typical raw format.

The various ADL-PCC modules are described in detail in the next subsections.

C. Point Cloud Block Partitioning

As for image and video blocks and macroblocks, the coding computational complexity grows significantly with the 3D block size, thus, a trade-off between correlation exploitation and complexity is required. To limit the complexity to practical levels while still achieving competitive compression performance, the full PC is partitioned into 3D blocks of constant size, namely $64 \times 64 \times 64$ binary voxels, as shown in Fig. 2. These blocks may then be coded independently, with the additional benefit of allowing spatial random access. Naturally, since only non-empty blocks are coded, the position of each block needs to be signaled in the bitstream (Signaling Bitstream in Fig. 1) to enable full PC reassembling at the decoder.

D. Autoencoder

The first step of the encoding procedure (as represented in Fig. 1b) for each 3D block is to apply an AE that operates as a non-linear transform. Following exhaustive analysis and experiments, the AE hyper-parameters have been defined as follows:

- **Number of layers** – The AE consists of six 3D convolutional layers: the first three layers, corresponding to the encoder side, apply the so-called *direct* or *analysis transform*; likewise, the last three layers, corresponding to the decoder side, apply the so-called *inverse* or *synthesis transform*. This design was chosen by taking into consideration the input block size ($64 \times 64 \times 64$) and the target size of the latent representation ($8 \times 8 \times 8$). While it is desirable to impose a strong enough bottleneck to achieve high compression, very small latent representations cause the loss of too much information, thus limiting the final reconstruction quality. Although seemingly simple, the used three layers AE design has shown to be adequate to create an expressive latent representation. More complex designs and the use of larger blocks have been considered and evaluated, but they have not proven to be worthwhile, since they achieved similar RD performances with significantly higher number of DL model parameters and increase in memory usage and computational requirements.
- **Number of filters** – The number of filters determines the AE model capacity, i.e., the amount of features that the DL model is able to express. Increasing the number of filters will generally improve the RD performance at higher rates at the cost of additional computational complexity and larger number of parameters; however, at lower rates, the DL coding model is prevented from using its full capacity, leading to a similar RD performance. Following extensive experiments, it was concluded that 32 filters provide a good trade-off between these trends.
- **Kernel/Filter support** – A $5 \times 5 \times 5$ kernel was found to provide an appropriate neighborhood region, considering the input block size of $64 \times 64 \times 64$.
- **Stride** – A stride values of 2 has been adopted to down (up)-sample the feature maps at the encoder (decoder), effectively reducing the dimensionality at the bottleneck and achieving higher compression.
- **Activation function** – The Sigmoid function was selected as the non-linear activation function; as it provides values bounded between '0' and '1', it is the typical choice for classification problems. In fact, the decoder process can be viewed as a classification problem, with the goal to classify each voxel as 'empty' or 'filled', making the Sigmoid a natural choice. While other functions, e.g. Rectified Linear Unit (ReLU) or Generalized Divisive Normalization [21], could have been used in intermediate layers, no relevant performance differences were obtained after some experiments; thus only the Sigmoid was selected to avoid mixing different functions.

Given the previously defined hyper-parameters, the latent representation includes a total of $8 \times 8 \times 8 \times 32$ latents while the AE network is defined through 520000 weights plus 129 biases, totaling 520129 trainable parameters.

E. Discretization

After the AE encoding process, the generated latent representation values must be discretized to be entropy coded; in this case, a simple rounding to the nearest integer is used. During training, the differentiable approach from [20] is adopted, in which the discretization operation is replaced by the addition of uniform noise with a value between -0.5 and 0.5 to each latent.

F. AE Latents Conditional Entropy Coding

The discretized AE latent representation is entropy coded to increase compression performance. For this purpose, the entropy coding solution proposed by Ballé *et al.* [21], available in the Tensorflow Compression library, has been adopted. It uses a range coder, and a zero-mean Gaussian scale mixture (GSM) as the entropy model for the latent representation, meaning that the distribution of each latent value is modeled by a Gaussian function with a specific standard deviation. The entropy parameters, notably the standard deviation (i.e. the scale) of each Gaussian, are estimated from the latent representation itself via a variational autoencoder, described in the next subsection. This allows the entropy model to be adapted to each coded block, which increases the compression efficiency.

G. Variational Autoencoder

As in [21], a VAE is used to estimate the scales of the GSM entropy model, thus using only the magnitude (i.e. absolute value – abs) of the AE latent representation. In this case, the VAE can be thought as a secondary – *hyper* – transform, which is applied to the AE latent representation to exploit the remaining redundancy in the structural relations. The adopted VAE design is presented in Fig. 1b, using 3D convolutional layers instead of 2D as in [21]. The VAE hyper-parameters are defined as follows:

- **Number of layers** – A design with three encoder and three decoder layers, as proposed in [21], was found to still be appropriate.
- **Kernel/Filter support** – Unlike in [21], the filter kernel was reduced to $3 \times 3 \times 3$ since the VAE input data (i.e. the AE latent representation) has now a smaller size of $8 \times 8 \times 8$ per filter.
- **Number of filters** – Since the VAE estimates a scale parameter for each AE latent, the number of values reconstructed by the decoder must match the number of AE latents; thus, the number of filters is the same as for the AE, here 32.
- **Stride** – Since the VAE input data has a size of $8 \times 8 \times 8$ per filter, it can only be down-sampled in two of the three layers, to avoid reaching a size of $1 \times 1 \times 1$, as this solution would lose too much information that could not be recovered at the decoder side. While the first layer uses a stride of 1, thus maintaining the resolution, the following two VAE layers use a stride of 2 to down-sample the feature maps.

- **Activation function** – Given the goal of predicting the GSM entropy model scales, the output should be non-negative, making the ReLU the most appropriate activation function [21].

Given the previously defined hyper-parameters, the VAE latent representation has a size of $2 \times 2 \times 2 \times 32$ latents, while the VAE network is defined through 165888 weights plus 160 biases, totaling 166048 trainable parameters.

H. VAE Latents Fixed Entropy Coding

The consequence of having an entropy model adapted to each block's latent representation is that the hyper-latent representation generated by the VAE needs to be coded and transmitted as side information (Side Info Bitstream in Fig. 1), so that the decoder can replicate the entropy model used for the AE latent representation encoding. Nonetheless, since the DL coding model is trained end-to-end, the extra side information is compensated by an increased compression efficiency at the AE latent representation level, thus resulting in an overall RD performance improvement.

Similar to the AE latent representation, the VAE hyper-latent representation is also discretized and entropy coded as proposed by Ballé *et al.* [21]. A fixed entropy model is used in this case. While a more complex, adaptive entropy model could be used for the VAE latents as well, this possibility was excluded since the associated rate accounts already only for a very small part of the total bitstream.

The fixed entropy model is learned and optimized during the DL coding model training process, and used for all blocks to be coded. This fixed entropy coding model is implemented with an Entropy Bottleneck layer available in the Tensorflow Compression library [21], and it includes 1472 trainable parameters; adding to the 520129 and 166048 parameters of the AE and VAE models, respectively, the total number of trainable parameters in the full DL coding model is increased to 687649.

I. DL Coding Model Selection

The characteristics of each PC block are addressed in an adaptive way, by selecting the best of the available N_{models} DL coding models, with the following procedure:

- **Block reconstruction** – The block under consideration is coded with all the available N_{models} DL coding models; the various block reconstructions are then obtained and converted to PC coordinates for quality assessment.
- **Rate and distortion assessment** – The distortion between the original and the reconstructed PC blocks is assessed with a typical PC objective distortion metric, such as the point-to-point distance (D1) or the point-to-plane distance (D2) [35]; as for the rate, the number of bits per input point required by each DL model is determined.
- **RD-based DL coding model selection** – The available DL coding model providing the block reconstruction with the lowest RD cost is selected. This optimization is described by the following equation:

$$Best\ DL\ mode = \\ = \underset{i\ \in\ Models}{\arg\min}\ \left(D_{MSE}\left(inPC,\ recPC_i\right) + \lambda_{mode} \times R_i\right) \tag{1}$$

where $inPC$ is the input block, $recPC_i$ is the corresponding reconstructed block with DL coding model i, D_{MSE} is the mean squared error for the selected distortion metric, R_i is the associated coding rate, and λ_{mode} is a trade-off control parameter. For the selected distortion and rate metrics, it was found after extensive testing that very low λ_{mode} values perform better, meaning that the distortion tends to dominate.

- **Selected DL coding model signaling** – The selected DL coding model for each PC block is signaled to the decoder by using a dedicated symbol in the Signaling Bitstream. This symbol stream is coded with an adaptive arithmetic codec [36], using adaptive probability tables, which are updated as each block is processed. The overall rate used for signaling the selected models is negligible (less than 0.5% of the total PC rate, in the worst case for the selected test PCs).

J. Point Cloud Block Merging

At the decoder, after all blocks have been reconstructed, the voxel-based representation is converted back into PC coordinates, the usual raw format. It is important to mention that the DL coding model does not generate binary voxels, but rather a floating-point probability value between '0' and '1'. Therefore, it is necessary to determine which voxels are 'filled', and thus correspond to points, by first binarizing the voxels. The binarization process using a straightforward rounding threshold of 0.5 was found to be a suitable procedure, after exhaustively testing other binarization thresholds. Following binarization and conversion into point coordinates, all points are merged into a single full PC, considering their original block position, which is also signaled in the bitstream.

IV. Deep Learning Coding Models Training Process

The proposed ADL-PCC solution uses several end-to-end DL coding models. These models are tailored to have different coding capabilities by using specific parameters for the training loss function. This section describes the adopted training loss function, as well as the used training dataset and hyper-parameters.

A. Loss Function

The training loss function has a fundamental impact on the obtained DL coding models and, thus, on the final RD performance. The goal of the loss function is to minimize the distortion between the original and reconstructed blocks, while minimizing the associated block coding rate. For this purpose, a traditional formulation involving a Lagrangian multiplier, λ_{train}, is used for the training loss function, which is given by:

$$Loss\ Function\ =\ Distortion + \lambda_{train} \times Coding\ rate \tag{2}$$

Thus, to obtain multiple RD trade-offs, i.e. RD points offering different rate-quality trade-offs, it is necessary to train multiple DL coding models using different λ_{train} values, to vary the weight of the rate in the trade-off. During training, the coding rate is estimated as the sum of the AE latent representation entropy with the VAE hyper-latent representation entropy.

Given the adopted voxel-based PC representation, where the input data is a block of binary voxels and each reconstructed voxel gets a probability score between '0' and '1', the reconstruction process for each voxel may be understood as a binary classification problem. As such, the block distortion may be measured as the sum of the binary classification error for each individual voxel. In this context, the so-called *Binary Cross Entropy* (BCE) [37] is a popular choice to measure the error for binary classification problems and is defined as:

$$BCE(v,u) = \begin{cases} -\log v, & u=1 \\ -\log(1-v), & u=0 \end{cases} \quad (3)$$

where u is the original voxel binary value and v is the corresponding reconstructed voxel probability score.

Due to the nature of PCs, '0' values, associated to the empty space, typically massively outnumber the '1' valued voxels, associated to the PC points. When summing over an entire block, this disparity causes the losses corresponding to the '0' valued voxels to dominate the total loss, resulting in a failure to properly reconstruct '1' valued voxels. In this context, a BCE variation so-called *Focal Loss (FL)* [37] is used to tackle this class imbalance issue by introducing appropriate weighting factors; this FL loss function is defined as:

$$FL(v,u) = \begin{cases} -\alpha(1-v)^{\gamma}\log v, & u=1 \\ -(1-\alpha)\,v^{\gamma}\log(1-v), & u=0 \end{cases} \quad (4)$$

where γ and α are two tunable parameters which impact the model's behavior and thus also the final RD performance.

Increasing the value of parameter γ gives more relevance to voxels that are hard to classify by downplaying the loss impact of the easy ones. In practice, voxels that already have a correct classification will have a reduced loss value so that the DL coding model focuses on misclassified voxels. Experimental tests have demonstrated that $\gamma = 2$ provides a good trade-off [37].

Parameter $\alpha \in [0,1]$ addresses the key class imbalance issue. Larger values of α increase the importance of the '1' valued voxels, meaning that the DL coding model will tend to reconstruct blocks with more filled voxels; on the other hand, lower α parameter values lead to reconstructed blocks with fewer points. With such behavior, the α parameter has a significant impact on the final RD performance for a given input block or PC. The best α value for a given block largely depends on its class imbalance level ('filled' versus 'empty' voxels ratio), which may vary substantially between PCs or within the various regions of one given PC, since depending on the acquisition method, they may be denser or sparser. As denser blocks typically have more consistently smooth surfaces, filled with many points, they generally benefit from lower α values, which result in less overpopulated reconstructed blocks; on the contrary, sparser blocks typically prefer higher α values so that originally 'filled' voxels are not wrongly discarded.

TABLE I
MPEG AND JPEG DATASET POINT CLOUDS

Training Data	Testing Data				
Point Cloud Name	**Point Cloud Name**	**Coding Depth**	**# Blocks**	**Avg # Points per Block**	**SD # Points per Block**
Egyptian Mask	MPEG				
Arco Valentino	*Statue Klimt*	9	46	5307	3559
PalazzoCarignano	*Queen*	10	208	4813	3678
Façade 00009	*House w/o Roof*	11	2000	1819	1606
Frog	*Longdress*	10	190	4516	3053
Façade 00015	*BasketballPlayer*	11	781	3746	2414
Façade 00064	*Dancer*	11	716	3621	2358
ULB Unicorn	JPEG				
Loot	*Bumbameuboi*	10	637	178	155
Red and Black	*Guanyin*	10	538	4251	2692
Soldier	*Rhetorician*	10	447	3876	2606
Shiva	*Romanoillamp*	10	271	2355	1372
Head	MVUB				
	Andrew	9	59	4740	3781
	David	9	92	3596	2988
	Phil	9	99	3745	3349
	Ricardo	9	51	4209	3343
	Sarah	9	77	3928	3404

Since there is no 'one size fits all' α parameter value, ADL-PCC offers block level adaptation capabilities by allowing the coded block to select one from several trained DL coding models, thus adjusting the α parameter for the specific characteristics of each block, as described in subsection III-I. This process maximizes the final RD performance as the encoder can adapt to the PC specific characteristics.

B. Training Dataset and Hyper-Parameters

The training PC dataset was obtained from the MPEG PC dataset used for the MPEG PCC standardization activities [35]. A subset of 13 PCs was selected, as presented in Table I. These PCs were down-sampled to a precision of 9 or 10 bit, according to the MPEG PCC Common Test Conditions [35]. Training PCs were partitioned into 3D blocks of size $64 \times 64 \times 64$. The blocks with less than 500 'filled' voxels have been removed in order to avoid the blocks with such low point count to negatively affect the training, due to the increased class imbalance. Overall, 6000 blocks were used in the training process.

Five RD trade-offs were considered by setting the λ_{train} value in (2) to 500, 900, 1500, 5000 and 20000. For each RD trade-off, i.e. each λ_{train} value, five models were trained using a different value for the α parameter in (4), namely 0.5, 0.6, 0.7, 0.8 and 0.9. While λ_{train} defines the RD point, α adjusts to the block characteristics.

The DL coding models were implemented and trained in Tensorflow [38] version 1.14, using the Tensorflow Compression library [21] version 1.2. All DL coding models were trained using the Adam algorithm [39] with a learning rate of 10^{-4} and minibatches of 8 blocks during 10^6 steps.

V. PERFORMANCE ASSESSMENT

This section presents the performance assessment of the proposed ADL-PCC solution using a set of meaningful experiments,

metrics and benchmarks. In addition, an ablation study is performed for the ADL-PCC solution without DL model selection, this means for a single α value, which will be called DL-PCC solution since no adaptation capabilities are present for this type of solution.

A. Experimental Setup

A subset of six PCs from the MPEG PC dataset [35] has been selected for the performance assessment experiments, following the MPEG PCC Common Test Conditions (CTC) [35]. In addition, four more PCs from the JPEG Pleno PC Coding dataset were also selected, in accordance with the JPEG Pleno PCC CTC [40], as well as five PCs from the Microsoft Voxelized Upper Bodies (MVUP) PC dataset [41]. The test PCs are detailed in Table I, including the coding depth, i.e. the bit precision at which each PC is coded; the number of (non-empty) 3D blocks of size $64 \times 64 \times 64$ that result from the partitioning; and the average and standard deviation (SD) of the number of points per block. As shown in Table I, test PCs with different characteristics have been used to ensure the representativeness of this performance assessment. The test PCs may be categorized as follows:

- ***Queen*, *Longdress*, *Basketball Player*, *Dancer*** – These four PCs representing people are very dense, with uniformly sampled and smooth surfaces; the latter two PCs have higher precision/coding depth (see Table I), which results in blocks containing simpler surfaces than the other two PCs;
- ***Statue Klimt*** – This PC represents a statue of a person; it is not as dense as the previous PCs, and has a more irregular, noisy surface;
- ***House without Roof*** – This large PC represents a house and a tree; as shown by the high standard deviation in Table I, this PC is more heterogeneous, notably containing both very sparse as well as denser regions/blocks;
- ***Bumbameuboi*** – This PC has a simple shape but is extremely sparse, producing many sparse blocks with low point count;
- ***Guanyin*, *Rhetorician*** – Similar to the PCs representing people, these two PCs, which consist of inanimate objects, are very dense with clean smooth surfaces;
- ***Romanoillamp*** – This PC is similar to the previous two, although with a slightly less dense surface;
- ***Andrew, David, Phil, Ricardo, Sarah*** – These PCs represent the front view of upper bodies, with a generally dense and uniform surface; however, they also contain some holes and noisy regions with some isolated points, mostly in the extremities.

To assess the RD performance, the two PC geometry objective quality metrics adopted by MPEG are used:

- **PSNR D1**: PSNR of the point-to-point (D1) distortion, measured as the symmetric error between every point in the reference PC and its closest neighbor in the reconstructed PC.
- **PSNR D2**: PSNR of the point-to-plane (D2) distortion, which is similar to D1, but now the error vector is first projected onto the normal vector at the reference point.

These objective quality metrics have been measured with the "mpeg-pcc-dmetric" software, version 0.13.4, provided by MPEG [35]. It is worth noting that these quality metrics differ from the distortion loss function used to train the DL coding models, which is not ideal. However, due to the adopted voxel-based PC representation, using D1 or D2 distortion metrics would require a conversion process back into 3D coordinates, which presents differentiability issues and thus prevents their usage. The rate is expressed as the number of bits per input point (filled voxel) of the original PC.

Naturally, the key comparison benchmark is the MPEG G-PCC standard [2], which has been specifically designed for static PCs; here the G-PCC reference software version 10.0 has been used. The coding configurations follow the corresponding MPEG or JPEG Pleno PCC CTC (depending on the test PC) for lossy geometry coding. In this context, MPEG G-PCC uses the octree+trisoup coding mode with the "positionQuantizationScale" parameter set to achieve the coding depth in Table I. Four RD points have been obtained by setting the "trisoup_node_size_log2" parameter to 1, 2, 3 and 4.

B. DL-PCC RD Performance

In a first experiment, the performance of the proposed PC coding solution is assessed when using only one single model. In this non-adaptive codec configuration, the *DL Coding Model Selection* module is overridden, since only one single trained DL coding model is available to encode all blocks of each PC for each RD point. The RD performance for the DL coding models trained with different parameter α values is shown in Fig. 3 in comparison with the MPEG G-PCC benchmark. A vertical line is included to indicate the rates at which G-PCC can achieve lossless compression with octree coding only; in this context, it will be considered that lossy coding above these rates is 'meaningless' and thus the BD-Rate and BD-PSNR assessment will only be performed in the useful range of rates for lossy coding, this means up to the lossless coding rates. From the results, it is possible to observe:

- The proposed DL-PCC solution clearly outperforms MPEG G-PCC for all tested PCs at least for one α parameter value; most often DL-PCC outperforms G-PCC for many α parameter values, notably for the higher rates.
- The chosen value for α significantly impacts the RD performance, sometimes with quality differences of more than 5 dB.
- The best α value varies for each PC, notably depending on the PC density; for each PC, the best α value also varies with the rate. This shows how important the proposed adaptive approach is to maximize the RD performance for different PCs and rate ranges.
- For denser PCs, notably with a higher average number of points per block as shown in Table I, lower α values generally achieve better RD performance.
- For sparser PCs, such as *Bumbameuboi*, a higher α value is required since the class imbalance level is much higher than for the other PCs; for this example, α values lower than 0.9 even fail to outperform MPEG G-PCC.

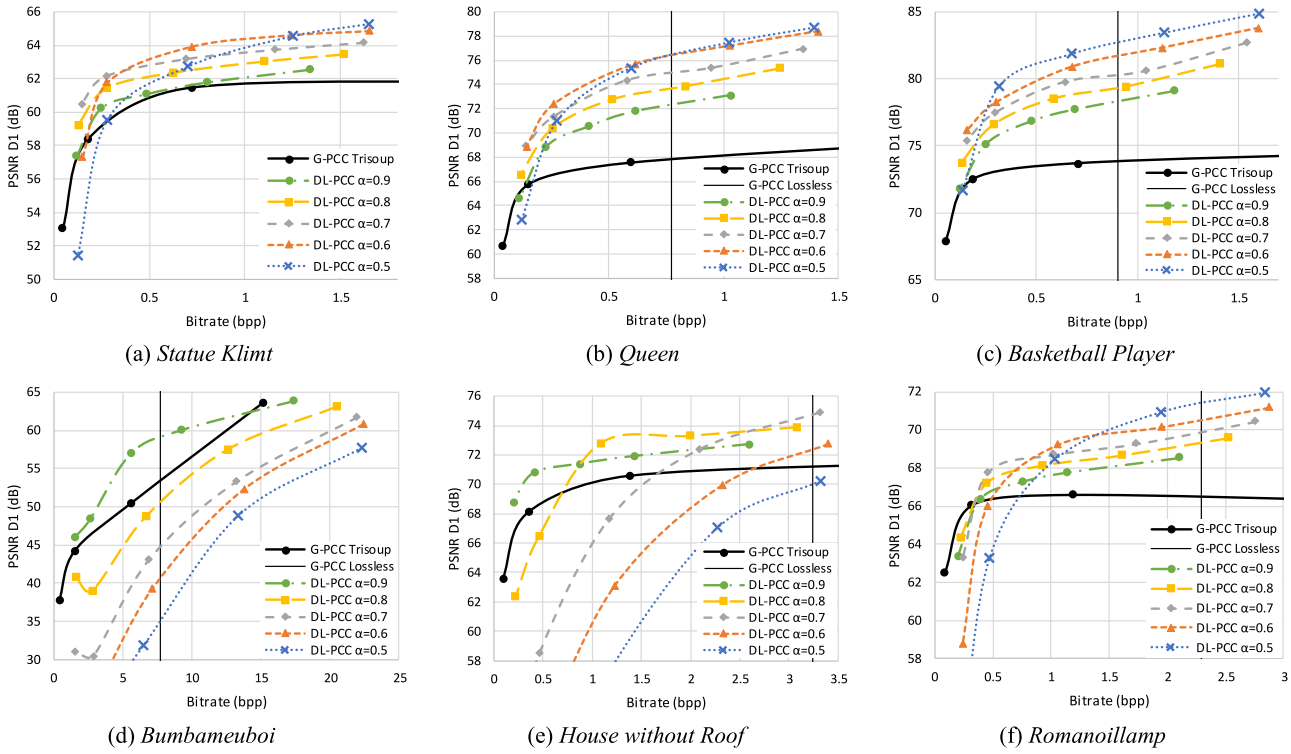


Fig. 3. RD performance comparison between the single model (non adaptive) DL-PCC solution and the MPEG G-PCC benchmark, for DL coding models trained with different α parameter values in the loss function of Equation (4). The vertical line indicates the rates at which G-PCC achieves lossless compression with octree coding only (for the *Statue Klimt* PC, this bitrate is larger than the shown bitrate range).

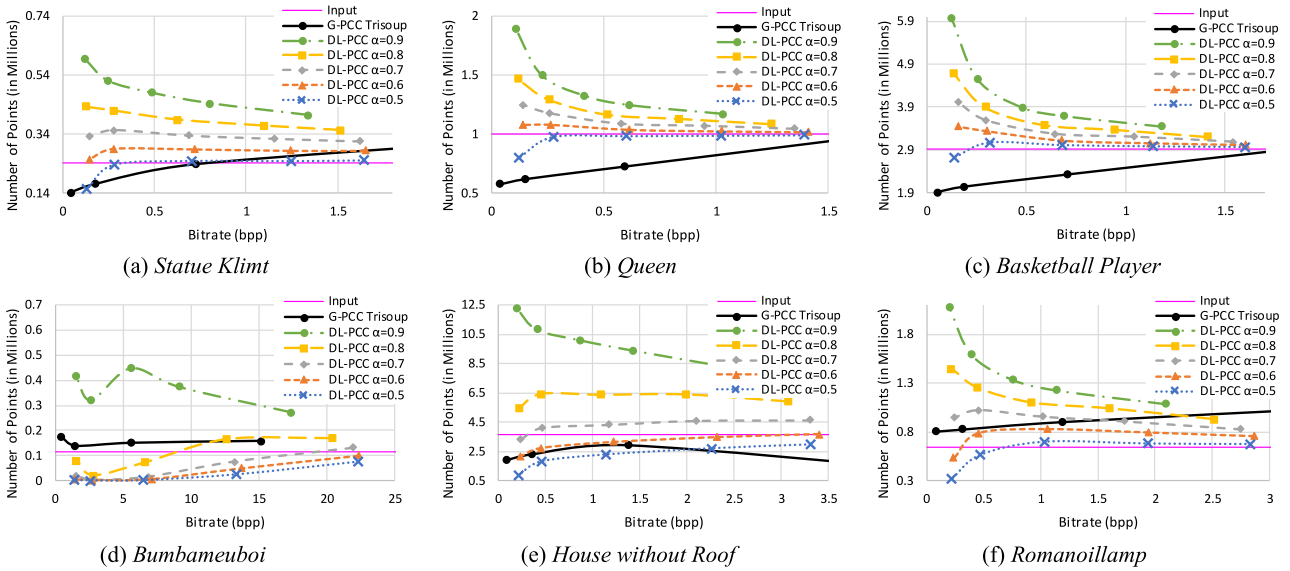
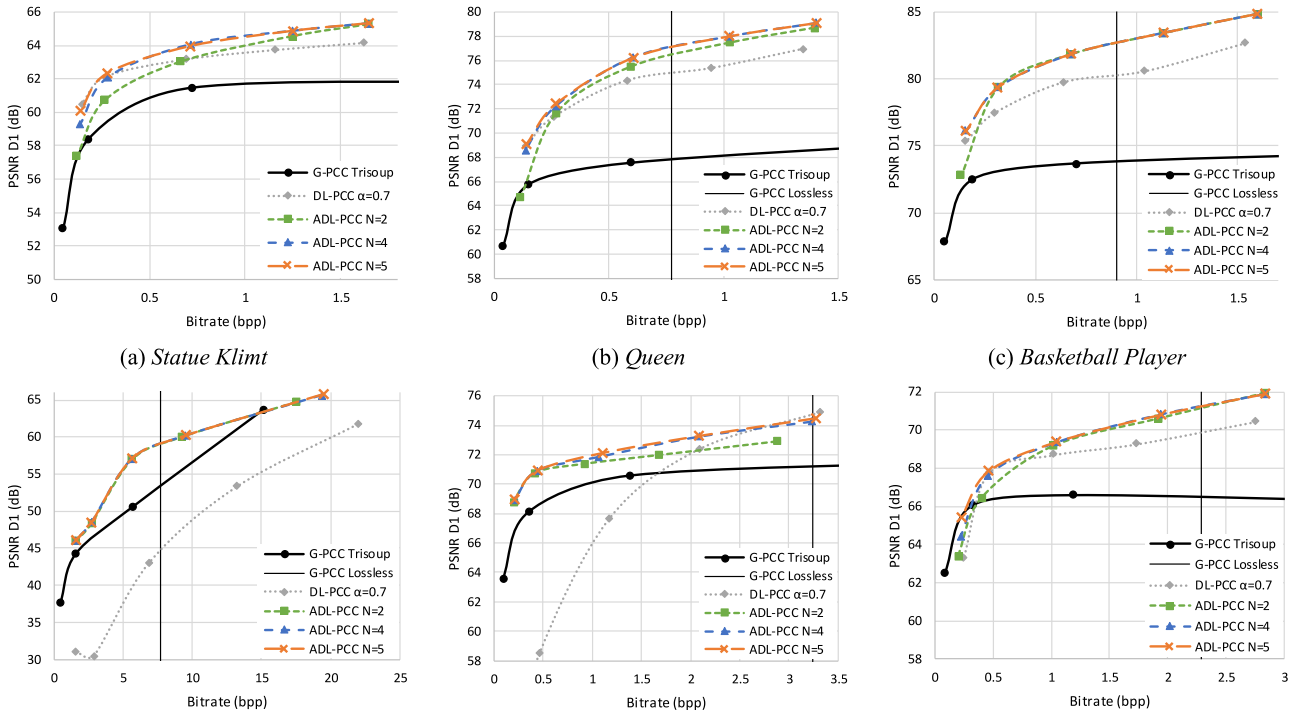


Fig. 4. Number of reconstructed points for different α parameter values in comparison with the original number of points of the input PC.

Fig. 4 presents the number of points of the DL-PCC reconstructed PCs for different α values. As can be seen, for the denser PCs, lower α values (0.5 and 0.6) are generally preferred since they allow to closely match the number of input points, producing a detailed reconstruction, while higher α values would needlessly overpopulate the surface. A less dense PC as *Romanoillamp* (Fig. 4f) starts to require higher α values at lower rates as otherwise the number of points would reduce beyond the input number. For the sparser *Bumbameuboi* (Fig. 4d), α values smaller than 0.9 tend to not be sufficient, causing the reconstruction to have a very reduced number of points, severely affecting the quality.

These results show that at higher rates the DL coding model tends to approximate the input number of points. However,



(a) *Statue Klimt* (b) *Queen* (c) *Basketball Player*

(d) *Bumbameuboi* (e) *House without Roof* (f) *Romanoillamp*

Fig. 5. RD performance comparison between the proposed ADL-PCC solution, using N=N_{models} available DL coding models, and the MPEG G-PCC benchmark.

at lower rates, naturally, the DL coding model does not have the capability to reconstruct finer details, especially for the sparser blocks (refer to Fig. 2b). To deal with this issue, the DL coding model tends to reconstruct more complete surfaces, filling them with more points to compensate the lack of detail and achieve more acceptable quality. While this is possible for higher α values, for lower α values, reconstructing many points in the originally empty space leads to a large training loss, and, thus, the learned model tends to reconstruct such sparse blocks without any points at all, leading to holes and poorer overall PC quality. Thus, higher α values are generally preferred at lower rates.

These results clearly demonstrate that none of the proposed single DL coding model can efficiently code all types of PCs. Due to the class imbalance issue, a specific DL coding model behaves very differently for denser and sparser PCs, meaning that different PC blocks require DL coding models with different α values in order to maximize the compression efficiency.

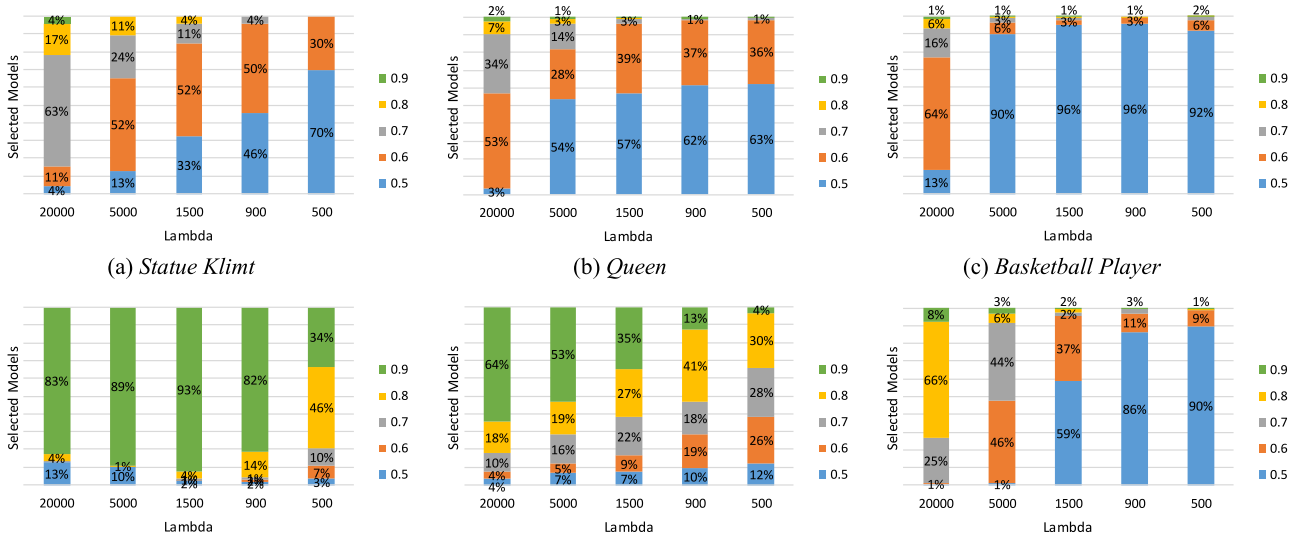
C. ADL-PCC RD Performance

The results above allow to clearly establish the importance of adopting an adaptive coding solution such as the proposed ADL-PCC solution. The development of an (automatic) adaptive coding solution aims to improve the compression performance for generic PCs and eliminate the difficult problem of deciding which of the α values is the best for each PC at each target rate. This problem is even more relevant for single PCs which include sparser and denser regions, since no single α value may be good for all blocks of this type of PC.

This subsection assesses and discusses the ADL-PCC RD performance, notably its efficiency in adapting to PCs with different characteristics. Besides the MPEG G-PCC benchmark, ADL-PCC is also compared with the DL-PCC solution using $\alpha = 0.7$. This α value was selected as a middle ground trade-off value between the previous extreme α values (0.5 and 0.9), for which the performance has very significant variations.

In this experiment, the proposed ADL-PCC solution was assessed for a varying number of DL coding models, notably $N_{models} = 2$, 4 and 5. For the DL coding model selection, the D1 distortion metric is initially used; the D2 distortion metric is also used in a latter experiment to evaluate the performance sensitivity to the model selection distortion metric. The RD performance results are presented in Fig. 5 and allow deriving the following conclusions:

- The adaptive version of the proposed solution, i.e. ADL-PCC, substantially outperforms the MPEG G-PCC standard overall;
- For $N_{models} = 2$, the selected DL coding models were trained with α values of 0.5 and 0.9; this solution offers an RD performance improvement over the $\alpha = 0.7$ single model, since it allows choosing more appropriate models for denser and sparser blocks. Nevertheless, some losses comparing to the single model may be observed for very small rate regions in some PCs;
- For $N_{models} = 4$, the α values of 0.5, 0.6, 0.8 and 0.9 were selected, which are in principle adequate to a wider range of block types, including mildly dense and sparse blocks. As expected, RD performance gains are observed for most PCs and all rates. For some cases, there is only a very small



(a) *Statue Klimt* (b) *Queen* (c) *Basketball Player*

(d) *Bumbameuboi* (e) *House without Roof* (f) *Romanoillamp*

Fig. 6. DL coding model selection per block in % for the proposed ADL-PCC solution for each RD point using $N_{models} = 5$.

gain when going from 2 to 4 DL coding models, namely for *Bumbameuboi*, *Basketball Player* and *Rhetorician*, particularly at higher rates. This can be explained by these PCs having more homogeneous blocks (in terms of sparsity) which are generally well matched with one single value of α. This may be confirmed in Fig. 3, where it can be seen that for the mentioned cases, the single DL coding model versions, with $\alpha = 0.9$ for *Bumbameuboi* and $\alpha = 0.5$ for *Basketball Player* and *Rhetorician*, are clearly better than the remaining ones for the corresponding RD regions.

- Finally, when using all the five DL coding models, the best RD performance is obtained, although with only a slight improvement. This is somewhat expected since the last added DL coding model, $\alpha = 0.7$, is generally good but often not the best of the available models for any PC, as seen in Fig. 3. Nonetheless, the key advantage of the proposed ADL-PCC solution is its ability to adapt to the diverse set of traits that PCs can present, in a fully automatic way.

In addition, Fig. 6 shows the percentage of blocks of each PC that used each DL coding model, for ADL-PCC using $N_{models} = 5$, for each RD point. It can be concluded that:

- The most used DL coding model for each PC generally matches the model with best RD performance in Fig. 3.
- For the highly sparse *Bumbameuboi* PC, an overwhelming majority of blocks selected the model with highest α, confirming the high class imbalance level of most blocks.
- The remaining PCs often use the DL coding model with lowest α; however, the usage of the various DL coding models varies significantly between PCs. This demonstrates the ADL-PCC capacity to adapt to different content and rate by selecting the best DL coding model for each specific case.

These results show that the ADL-PCC solution has two key advantages: i) for homogeneous PCs, it is able to automatically select the most appropriate α value, thus adjusting to the PC global characteristics; ii) for heterogeneous PCs, it is able to automatically select the most appropriate α value at block level, thus adjusting to the PC local characteristics; for both cases, the compression performance is maximized.

Fig. 7 shows the block-level quality for all blocks of all test PCs, measured with PSNR D1, as a function of the number of points in the block, where each selected DL coding model corresponds to a different color. For both the lower and higher rates, the relation between the selected DL coding models and the number of points in the blocks is clear. As expected, the lower the point count per block, the higher is the class imbalance level, thus leading to the selection of higher α values. Moreover, blocks coded with DL coding models using lower α values generally achieve better block quality. This can be related with the fact that blocks with a dense surface (Fig. 2a), which tend to use lower α values, are more easily reconstructed; on the other hand, blocks with a sparser surface (Fig. 2b), which tend to use higher α values, are generally reconstructed with larger errors. At lower rates in particular, it can be seen that blocks with lower number of points tend to be more poorly reconstructed, whereas for higher rates, this issue is mitigated. This is due to most blocks with low point count being generally sparser and harder to reconstruct, especially when the DL coding model has lower capacity as it is the case for lower rates.

D. Visual Assessment

Besides the objective evaluation, a few visual examples are included in Fig. 8, in an effort to show the coding artifacts obtained for PCs with different characteristics.

Fig. 8a demonstrates the expectable G-PCC Trisoup artifacts when reconstructing joint surfaces (e.g. hand plus the front and back surfaces of the tablet) at lower rates, even when surfaces are completely flat. G-PCC Trisoup creates triangles between the points originally located in two different surfaces, mixing them, thus resulting in a rather poor reconstruction. On the other

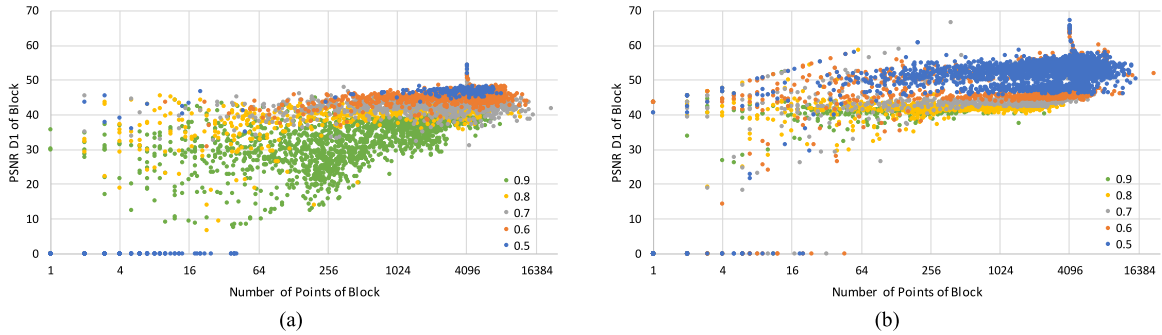


Fig. 7. Block PSNR D1 quality as a function of the number of points in the block, and respective model α value. (a) Lowest and (b) highest rate.

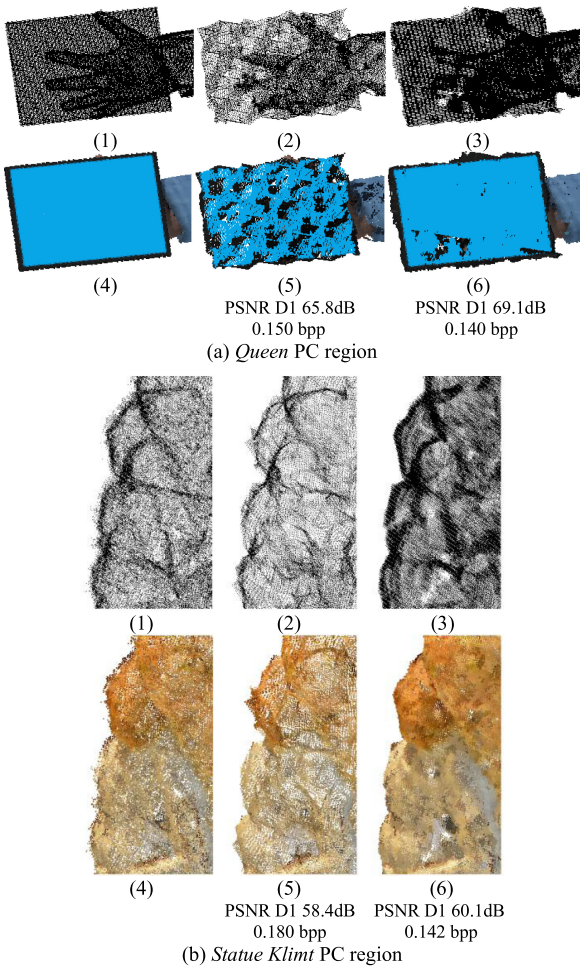


Fig. 8. Visual examples comparison between G-PCC Trisoup and ADL-PCC N_{models}=5, for different PC regions. Geometry only, with point size 1: (1) original PC; (2) MPEG G-PCC Trisoup; and (3) ADL-PCC. Recolored geometry, with point size 3: (4) originalPC; (5) MPEG G-PCC Trisoup; and (6) ADL-PCC. Recoloring was performed by transferring the original color from the closest point in the original PC.

hand, the corresponding ADL-PCC reconstruction is much more faithful and subjectively pleasant than G-PCC Trisoup, even for lower rates.

Fig. 8b shows results for the *Statue Klimt* PC, which has many noisy points close to the surface, making it rather irregular. In this case, G-PCC Trisoup generates triangles connecting the surface and the noisy points, once again producing rather visible artifacts. On the other hand, ADL-PCC is able to generate a smoother surface, filtering some of the noise and providing a more pleasant reconstruction.

E. DL Model Selection Distortion Metric Impact

In the previous experiment, the best DL coding model for each block was selected based on the D1 distortion metric. Thus, if the final RD performance is measured with the same distortion/quality metric, it is expected the performance to be maximized. However, other metrics like the PSNR D2 are often used to measure the PC reconstruction quality. This experiment aims at studying the impact of a distortion metric mismatch, i.e. using different distortion metrics for the DL coding model selection process and the final RD performance assessment.

With this target in mind, the proposed ADL-PCC RD performance was assessed for $N_{models} = 5$, using both the D1 (ADL-PCC N = 5 D1) and D2 (ADL-PCC N = 5 D2) distortion metrics for the DL model selection process. Fig. 9 shows the final RD performance, evaluated for both PSNR D1 and PSNR D2. As expected, the results demonstrate that the RD performance for a given metric is maximized when the same distortion metric is used for the DL Coding Model Selection process. Since the proposed ADL-PCC solution allows using any PC distortion metric for the DL Coding Model Selection process, it is not only able to adapt to the content itself but also to the PC distortion metric which is considered to be the most relevant for the application scenario at hand. Nevertheless, it should be noted that in the cases in which the two distortion metrics are not the same, the final RD performance impact is not very relevant, with very few exceptions.

F. Computational Complexity

In terms of computational complexity, the coding times per test PC were measured for both the MPEG G-PCC benchmark and ADL-PCC solution. The experiments were performed on an Intel Core i7-6700 CPU @ 3.40 GHz computer with NVidia GeForce RTX 2070 8 GB GPU and 32 GB of RAM, running Ubuntu 18.04. On average, G-PCC takes up to 76 (40) seconds to encode (decode) each test PC. As for ADL-PCC, the encoding time depends on the number of DL coding models; with

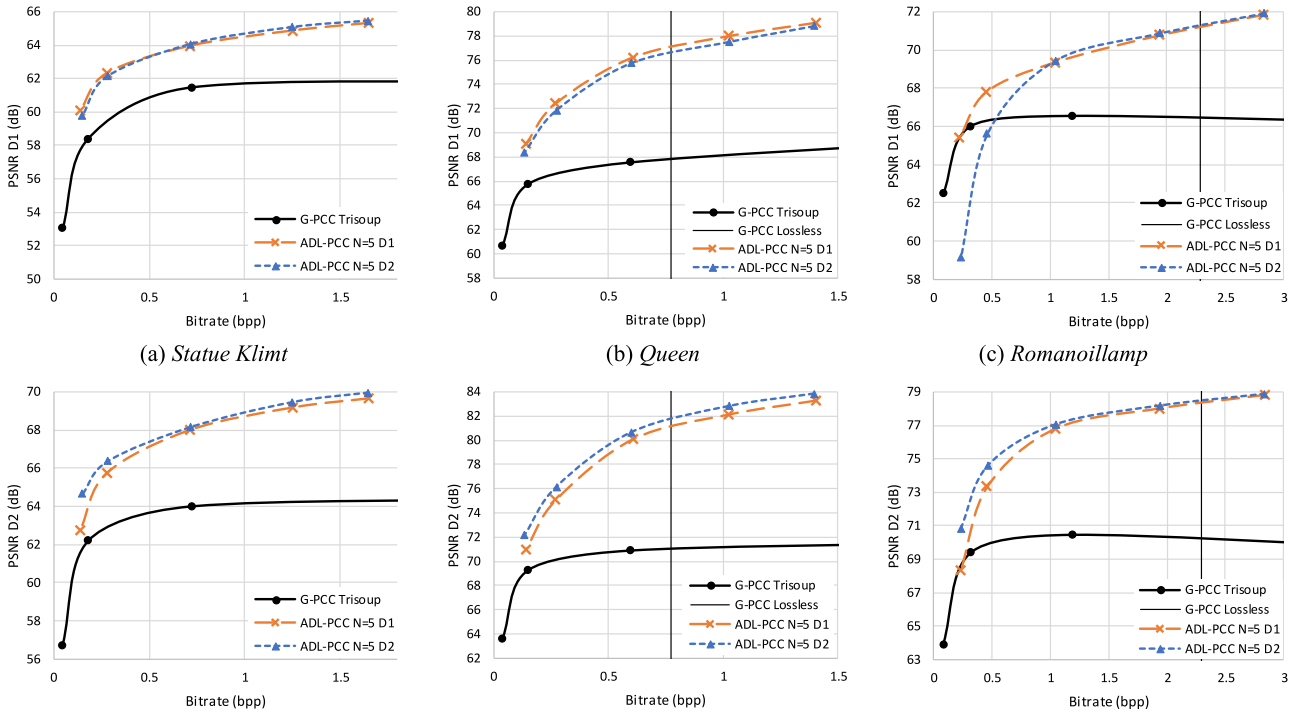


Fig. 9. RD performance comparison between the proposed ADL-PCC solution and the MPEG G-PCC benchmark, when different distortion metrics are used for the model selection stage: (a), (b), (c) PSNR D1 results; (d), (e), (f) PSNR D2 results.

$N_{models} = 5$, the average encoder (decoder) time for the selected PC dataset was 134 (52) seconds. This means that, with some software optimization, the ADL-PCC and G-PCC encoding and decoding times may not be that different. Naturally, as for every DL-based approach, the training time is much longer (several hours) but this is not a critical issue since all models are trained offline and made available for future use.

G. Comparison With Other DL-Based Solutions

In a final experiment, the ADL-PCC performance with $N_{models} = 5$ is compared with two recent DL-based PC coding solutions available in the literature, in this case [29] and [30] labelled in the charts and tables as *Quach et al*. and *Wang et al*, respectively. For both these DL-based coding solutions, results were obtained for all the test PCs listed in Table I using the publicly available software implementations and trained models, with their default parameters [29], [30]. Fig. 10 compares the ADL-PCC RD performance with the two previously mentioned solutions, using MPEG G-PCC Trisoup as benchmark, for the PSNR D1 metric, using D1 as optimization metrics for ADL-PCC and Quach *et al*.

When comparing ADL-PCC with Quach *et al*. [29], it is possible to observe that:

- For the denser PCs (*Queen*, *Basketball Player*, *Romanoillamp*), the proposed ADL-PCC solution presents similar RD performance, with either marginal gains or losses.
- For the sparser (*House without Roof*) and noisier (*Statue Klimt*) PCs, the proposed ADL-PCC presents a larger advantage, with more consistent gains.
- For the very sparse *Bumbameuboi* PC, the Quach *et al*. solution shows a very significant PSNR D1 gain. However, while seemingly offering a better objective quality, a more detailed analysis shows that this does not translate into a better subjective quality since there are very strong artifacts, which are not present in the ADL-PCC solution. To demonstrate this effect, Fig. 11 shows the *Bumbameuboi* reconstructed geometry for Quach *et al*. in comparison with ADL-PCC. It can be seen that, for Quach *et al*., the reconstructed PC is massively overpopulated, with almost 30 times the number of original points, and has a considerable number of noisy points appearing far from the surface, unlike for the proposed ADL-PCC solution. These artifacts are due to a very low binarization threshold, selected to optimize the PSNR D1 for each block, which causes the voxels that should be empty to get filled at the block's corners and edges. Contrary to what could be expected, these artifacts have a low impact on the PSNR D1 metric, because their large error is diluted in the millions of points that are closer to the original surface, since this distance metric considers the average error distance. However, when assessing using PSNR D2, which is less dependent on the number of points, the proposed ADL-PCC presents better RD performance than Quach *et al*. This effect highlights the importance of using more than one objective quality metric for performance assessment, and making at least some visual inspection, considering DL-based coding may bring new visual artifacts that have yet to be fully understood.

When comparing ADL-PCC with Wang *et al*. [30], it is possible to observe that:

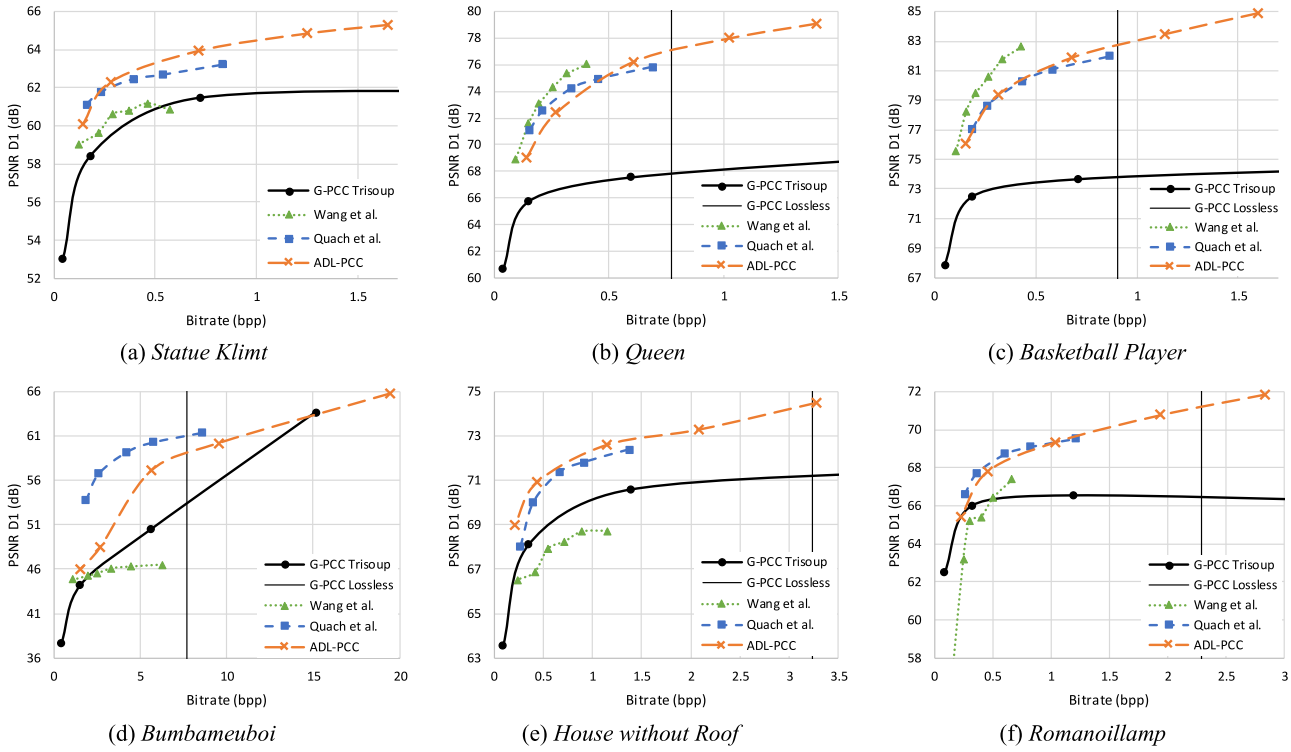


Fig. 10. PSNR D1 RD performance comparison between the proposed ADL-PCC, using N=5, and other DL-based solutions, namely Quach *et al*. [29] and Wang *et al*. [30], with the MPEG G-PCC benchmark. For ADL-PCC and Quach *et al*., the optimization distortin metric is D1.

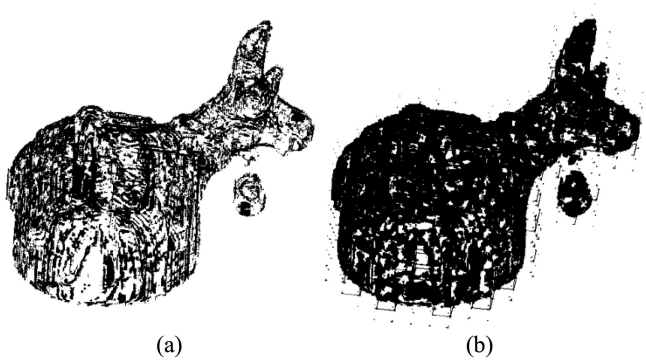


Fig. 11. *Bumbameuboi* PC example, decoded using: (a) ADL-PCC (5 models), with PSNR D1 46.0dB and PSNR D2 59.5dB, at 1.53 bpp, resulting in 415566 points; and (b) Quach *et al*. [29] solution with D1 optimization, with PSNR D1 53.8dB and PSNR D2 55.7dB, at 1.85 bpp, resulting in 3173556 points.

- For the very dense PCs with smooth surfaces (*Queen*, *Basketball Player*), Wang *et al*. presents significant performance gains.
- For the sparser (*Bumbameuboi*, *House without Roof*, *Romanoillamp*) and noisier (*Statue Klimt*) PCs, the performance of Wang *et al*. is severely penalized, being even inferior to MPEG G-PCC Trisoup, thus presenting overall a very unstable and varying RD performance behavior, heavily depending on the PC characteristics.

To summarize the full set of RD performance related experiments and associated results, Table II and Table III present the BD rate savings and PSNR gains, using as reference the MPEG G-PCC Trisoup benchmark, for PSNR D1 and PSNR D2, respectively. Results are shown for the proposed ADL-PCC, using both D1 and D2 distortion metric for DL model selection (ADL-PCC D1 and ADL-PCC D2), for Quach *et al*. using both D1 and D2 for the binarization threshold optimization (Quach *et al*. D1 and Quach *et al*. D2), and finally for Wang *et al*. In order to provide a more meaningful and fair comparison, the rate savings and PSNR gains are computed using only the rate range up to the G-PCC Octree lossless coding rates, as marked in the RD performance charts with a vertical line; this assessment approach is not common in the literature but allows to obtain a more reliable performance assessment, excluding rates above the lossless coding threshold where lossy coding performance is less relevant.

Both the BD-Rate and BD-PSNR results are reported as much as possible. Since the proposed ADL-PCC solution performs much better than the G-PCC Trisoup benchmark, the type of RD curves obtained lead to somewhat unreliable BD-Rate values, as there is a rather small curve overlap, if any, in terms of the quality ranges addressed by the two coding solutions. In fact, for some cases, marked with an asterisk '$*$' in Table II and Table III, it is even not possible to compute any BD-Rate value since the RD curves do not overlap at all in quality, i.e. the proposed ADL-PCC performance is sufficiently better than G-PCC Trisoup such that the minimum ADL-PCC quality is above the maximum G-PCC Trisoup quality; in spite of the penalty for ADL-PCC in terms of comparative results (since the rate gains are very large), these cases have not been used to compute the averages included in Table II and Table III. This occurs mostly for the PCs which present very large RD performance gains, as demonstrated by

TABLE II
BD-RATE (%) AND BD-PSNR (dB) PERFORMANCE GAINS OF THE PROPOSED ADL-PCC SOLUTION, IN COMPARISON WITH QUACH *ET AL.* [29] AND WANG *ET AL.* [30], USING MPEG G-PCC TRISOUP AS REFERENCE, **FOR THE PSNR D1 METRIC** (USING D1 AND D2 AS OPTIMIZATION DISTORTION METRICS)

Data set	Point Cloud Name	ADL-PCC D1		ADL-PCC D2		Quach et al. D1		Quach et al. D2		Wang et al.	
		BD-Rate	BD-PSNR	BD-Rate	BD-PSNR	BD-Rate	BD-PSNR	BD-Rate	BD-PSNR	BD-Rate	BD-PSNR
MPEG	*Statue Klimt*	-67.0%	**2.69**	-61.1%	2.66	-77.4%	2.11	-63.2%	1.67	-17.2%	0.71
	Queen	*	6.45	*	5.84	*	**7.00**	*	6.43	*	6.76
	House without Roof	-71.0%	**2.14**	-46.9%	1.95	-52.9%	1.64	-58.2%	1.70	81.5%	-1.32
	Longdress	*	5.31	-71.8%	5.30	*	5.51	*	4.70	-85.0%	**5.78**
	Basketball Player	*	6.63	*	6.25	*	6.69	*	6.21	*	**7.03**
	Dancer	*	6.30	-66.7%	5.82	*	**6.48**	*	6.00	-76.4%	6.24
JPEG	*Bumbameuboi*	-37.8%	3.69	-29.4%	2.40	*	**9.87**	-13.3%	2.59	24.3%	-1.04
	Guanyin	-79.7%	5.03	-68.7%	4.90	*	**5.35**	-22.0%	4.71	76.6%	-0.62
	Rhetorician	-76.5%	4.21	-57.7%	4.21	*	4.52	126.5%	-0.09	-54.9%	**4.98**
	Romanoillamp	13.7%	**2.24**	199.2%	0.77	*	2.09	205.2%	-0.51	133.1%	-1.83
MVUB	*Andrew*	-85.1%	**5.63**	-84.3%	5.61	*	5.03	*	4.97	-84.4%	5.28
	David	*	**6.11**	-79.2%	5.66	-78.4%	5.17	*	5.37	-43.7%	1.30
	Phil	-84.0%	5.59	-83.1%	5.47	-83.4%	4.93	*	5.17	-87.1%	**5.65**
	Ricardo	*	5.75	*	**5.85**	*	5.28	*	5.31	-87.6%	5.61
	Sarah	-80.9%	6.30	-82.1%	6.39	*	5.87	*	5.31	-85.3%	**6.83**
Average		-63.1%	4.94	-44.3%	4.61	-73.0%	**5.17**	29.2%	3.97	-23.5%	3.42
Average (no *Bumbameuboi*)		-66.3%	**5.03**	-45.7%	4.76	-73.0%	4.83	37.7%	4.07	-27.5%	3.74

TABLE III
BD-RATE (%) AND BD-PSNR (dB) PERFORMANCE GAINS OF THE PROPOSED ADL-PCC SOLUTION, IN COMPARISON WITH QUACH *ET AL.* [29] AND WANG *ET AL.* [30], USING MPEG G-PCC TRISOUP AS REFERENCE, **FOR THE PSNR D2 METRIC** (USING D1 AND D2 AS OPTIMIZATION DISTORTION METRICS)

Data set	Point Cloud Name	ADL-PCC D1		ADL-PCC D2		Quach et al. D1		Quach et al. D2		Wang et al.	
		BD-Rate	BD-PSNR	BD-Rate	BD-PSNR	BD-Rate	BD-PSNR	BD-Rate	BD-PSNR	BD-Rate	BD-PSNR
MPEG	*Statue Klimt*	-64.4%	3.44	*	**3.97**	-88.3%	2.44	*	2.70	-78.9%	2.11
	Queen	-81.7%	6.04	*	6.88	*	6.74	*	**7.51**	*	6.54
	House without Roof	-33.5%	1.94	-43.3%	**2.13**	-6.1%	0.80	-34.1%	1.00	20.5%	-0.32
	Longdress	-70.8%	5.49	*	5.90	*	5.65	*	6.22	-88.0%	**6.29**
	Basketball Player	*	7.27	*	7.38	*	7.20	*	**7.69**	*	7.49
	Dancer	*	6.82	*	6.94	*	6.89	*	**7.39**	*	6.81
JPEG	*Bumbameuboi*	8.5%	1.36	6.7%	1.33	97.5%	-0.24	-3.0%	**1.94**	-4.9%	0.04
	Guanyin	-71.2%	5.82	*	6.32	*	6.12	*	**6.47**	1.9%	1.82
	Rhetorician	-63.4%	4.89	*	5.36	*	5.18	*	5.44	*	**6.60**
	Romanoillamp	-77.7%	4.70	*	**5.55**	*	4.50	*	4.82	-97.1%	4.23
MVUB	*Andrew*	-84.4%	6.34	*	**6.72**	*	5.49	*	5.60	-86.0%	5.82
	David	*	7.00	*	**7.28**	-79.1%	5.74	*	6.02	-87.2%	4.42
	Phil	-81.1%	6.21	*	**6.67**	-80.4%	5.29	*	5.72	-88.4%	6.08
	Ricardo	*	6.41	*	**6.59**	*	5.62	*	5.77	-87.1%	5.77
	Sarah	-79.5%	6.64	*	**7.13**	*	5.92	*	5.62	-87.8%	6.37
Average		-63.6%	5.36	-18.3%	**5.74**	-31.3%	4.89	-18.6%	5.33	-62.1%	4.67
Average (no *Bumbameuboi*)		-70.8%	5.64	-43.3%	**6.06**	-63.5%	5.26	-34.1%	5.57	-67.8%	5.00

the very high corresponding BD-PSNR values. In fact, for the type of RD curves obtained here, BD-PSNR is a more reliable metric as there is a large range of rate overlapping between the curves under comparison.

As previously shown in Fig. 10, both Quach *et al*. and Wang *et al*. tend to perform similarly or slightly better than the proposed ADL-PCC solution for the denser and well-behaved PCs. However, for sparser and noisy PCs, the proposed ADL-PCC offers significant RD performance gains. On average, the proposed ADL-PCC shows a BD-PSNR of 4.94dB and 5.74dB, for PSNR D1 and PSNR D2 with matching optimization metric, respectively, whereas Quach *et al*. [29] shows 5.17dB and 5.33dB, and Wang *et al*. 3.42dB and 4.67dB. However, if the *Bumbameuboi* PC is discarded from the average, since it unreliably inflates the gain of Quach *et al*., as explained earlier when considering the subjective quality, the ADL-PCC gains increase to the point of outperforming the two alternative DL-based coding solutions and G-PCC benchmark, on average, both for PSNR D1 and PSNR D2.

Furthermore, the quality metric used for the binarization threshold optimization in Quach *et al*. seems to have a large RD performance impact, since there is a large disparity between the PSNR D1 and PSNR D2 results depending on the optimization metric, whereas for ADL-PCC results are more consistent for the two adopted optimization metrics.

Overall, these performance results allow to conclude that the proposed ADL-PCC solution indeed offers high adaptability to very diverse PC content, providing not only the most efficient solution, on average, but especially the most consistent RD performance over very diverse PCs, in comparison with the selected state-of-the art coding solutions. Finally, it should be noted that the proposed ADL-PCC coding solution and the optimization of the binarization threshold from [29], and also the improved entropy coding model from [30], are not mutually exclusive, implying that by integrating these tools better RD performance may potentially be achieved for ADL-PCC. In addition, there are no restrictions on the DL coding models that are available for selection, notably ADL-PCC allows the use of any number of DL

coding models, which may be trained with different parameters and even different distortion metrics in order to provide the features needed to improve the ADL-PCC adaptability power.

VI. Conclusions And Future Work

This paper presents a very adaptive deep learning-based point cloud geometry coding solution able to consider the wide range of characteristics of PC content associated to different application domains. The proposed adaptive solution allows different DL coding models to focus on specific PC characteristics, automatically selecting the most appropriate model for each PC region in order to maximize the RD performance. Results show that the proposed ADL-PCC solution is able to efficiently code both sparse and dense PCs, outperforming MPEG G-PCC Trisoup with average quality gains up to 4.9 dB and 5.7 dB for PSNR D1 and PSNR D2, respectively. Moreover, these compression gains do not arise at the cost of increasing the encoding and decoding complexities regarding G-PCC Trisoup. The unusual size of the DL-based coding solutions compression gains regarding the G-PCC Trisoup standard clearly highlights that DL-based coding is ready to play a key role in PC geometry coding.

Acknowledgment

The authors would like to thank the authors of [29] and [30] which were very helpful in providing their software and assistance to allow a more extensive performance comparison.

References

[1] C. Cao, M. Preda, and T. Zaharia, "3D point cloud compression: A survey," in *Proc. Int. Conf. 3D Web Technol.*, Jul. 2019.

[2] S. Schwarz *et al.*, "Emerging MPEG Standards for Point Cloud Compression," *IEEE Trans. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 1, pp. 133–148, Mar. 2019.

[3] L. Cui, R. Mekuria, M. Preda, and E. S. Jang, "Point-Cloud Compression: Moving Picture Experts Group's New Standard in 2020," *IEEE Consum. Electron. Mag.*, vol. 8, no. 4, pp. 17–21, Jul. 2019.

[4] 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.

[5] ISO/IEC JTC 1/SC29/WG1 N88014,"Final Call for Evidence on JPEG Pleno Point Cloud Coding," in *Proc. Online Meet.*, Jul. 2020.

[6] S. Pouyanfar *et al.*, "A Survey on Deep Learning: Algorithms, Techniques, Applications," *ACM Comput. Surv.*, vol. 51, no. 5, Jan. 2019, Art. no. 92.

[7] A. Voulodimos, N. Doulamis, A. Doulamis and E. Protopapadakis, "Deep Learning for Computer Vision: A Brief Review," *Comput. Intell. Neurosci.*, vol. 2018, Feb. 2018, Art. no. 7068349.

[8] ISO/IEC JTC 1/SC29/WG1 N86018. *Call for Evidence on Learning-based Image Coding Technologies (JPEG AI)*. Sydney, Australia, Jan. 2020.

[9] R. B. Rusu and S. Cousins, "3D is Here: Point Cloud Library (PCL)," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2011.

[10] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach, "Real-Time Compression of Point Cloud Streams," in *Proc. IEEE Int. Conf. Robot. Automat.*, May 2012.

[11] R. L. de Queiroz and P. A. Chou, "Motion-Compensated Compression of Dynamic Voxelized Point Clouds," *IEEE Trans. Image Process.*, vol. 26, no. 8, pp. 3886–3895, Aug. 2017.

[12] T. Ochotta and D. Saupe, "Compression of Point-Based 3D Models by Shape-Adaptive Wavelet Coding of Multi-Height Fields," in *Proc. Eurographics Symp. Point-Based Graph.*, Jun. 2004.

[13] H. Houshiar and A. Nüchter, "3D Point Cloud Compression Using Conventional Image Compression for Efficient Data Transmission," in *Proc. Int. Conf. Inf. Commun. Automat. Technol.*, Oct. 2015.

[14] G. J. Sullivan, J. Ohm, W. Han and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[15] J. Chen, M. Karczewicz, Y. Huang, K. Choi, J. Ohm, and G. J. Sullivan, "The Joint Exploration Model (JEM) for Video Compression With Capability Beyond HEVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 5, pp. 1208–1225, May 2020.

[16] J. Ballé, V. Laparra and E. P. Simoncelli, "End-to-End Optimization of Nonlinear Transform Codes for Perceptual Quality," in *Proc. Picture Coding Symp.*, Dec. 2016.

[17] G. Toderici *et al.*, "Variable Rate Image Compression with Recurrent Neural Networks," in *Proc. Int. Conf. Learning Representations*, May 2016.

[18] G. Toderici *et al.*, "Full Resolution Image Compression with Recurrent Neural Networks," in *Proc. IEEE Conf. Compu. Vis. Pattern Recognit.*, Jul. 2017.

[19] L. Theis, W. Shi, A. Cunningham, and F. Huszár, "Lossy Image Compression with Compressive Autoencoders," in *Proc. Int. Conf. Learning Representations*, Apr. 2017.

[20] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end Optimized Image Compression," in *Proc. Int. Conf. Learning Representations*, Apr. 2017.

[21] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational Image Compression with a Scale Hyperprior," in *Proc. Int. Conf. Learning Representations*, Apr. 2018.

[22] D. Minnen, J. Ballé, and G. Toderici, "Joint Autoregressive and Hierarchical Priors for Learned Image Compression," in *Proc. Adv. in Neural Inf. Process. Syst.*, Dec. 2018.

[23] ISO/IEC JTC 1/SC29/WG1 N89022, "Report on the JPEG AI Call for Evidence Results," in *Proc. Online Meet.*, Oct. 2020.

[24] W. Yan, Y. Shao, S. Liu, T. H. Li, Z. Li, and G. Li, "Deep AutoEncoder-based Lossy Geometry Compression for Point Clouds," *arXiv:1905.03691v1*, Apr. 2019.

[25] M. Quach, G. Valenzise, and F. Dufaux, "Learning Convolutional Transforms for Lossy Point Cloud Geometry Compression," in *Proc. IEEE Int. Conf. Image Process.*, Sep. 2019.

[26] A. Guarda, N. Rodrigues, and F. Pereira, "Point Cloud Coding: Adopting a Deep Learning-based Approach," in *Proc. Picture Coding Symp.*, Nov. 2019.

[27] A. Guarda, N. Rodrigues, and F. Pereira, "Deep Learning-Based Point Cloud Coding: A Behavior and Performance Study," in *Proc. Eur. Workshop Vis. Inf. Process.*, Oct. 2019.

[28] R. Mekuria, K. Blom, and P. Cesar, "Design, Implementation, and Evaluation of a Point Cloud Codec for Tele-Immersive Video," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, no. 4, pp. 828–842, Apr. 2017.

[29] M. Quach, G. Valenzise, and F. Dufaux, "Improved Deep Point Cloud Geometry Compression," in *Proc. IEEE Workshop Multimedia Signal Process.*, Sep. 2020. [Online] Available: https://github.com/mauriceqch/pcc_geo_cnn_v2, Accessed on: Nov. 22, 2020

[30] J. Wang, H. Zhu, Z. Ma, T. Chen, H. Liu, and Q. Shen, "Learned Point Cloud Geometry Compression," *arXiv:1909.12037*, Sep. 2019. [Online] Available: https://github.com/NJUVISION/PCGCv1, Accessed on: Nov. 22, 2020

[31] A. Guarda, N. Rodrigues, and F. Pereira, "Deep Learning-Based Point Cloud Coding: RD Control Through Implicit and Explicit Quantization," in *Proc. Int. Conf. Multimedia Expo Workshop*, Jul. 2020.

[32] E. Alexiou, K. Tung, and T. Ebrahimi, "Towards Neural Network Approaches for Point Cloud Compression," *SPIE Appl. Digit. Image Process. XLIII*, vol. 11510, pp. 18–37, Aug. 2020.

[33] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *Proc. IEEE Conf. Compu. Vis. Pattern Recognit.*, Jul. 2017.

[34] D. Tang *et al.*, "Deep Implicit Volume Compression," in *Proc. IEEE Conf. Compu. Vis. Pattern Recognit.*, Jun. 2020.

[35] ISO/IEC JTC1/SC29/WG11 N19084. *Common Test Conditions for Point Cloud Compression*. Brussels, Belgium, Jan. 2020.

[36] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic Coding for Data Compression," *Commun. ACM*, vol. 30, no. 6, pp. 520–540, Jun. 1987.

[37] T. Lin, P. Goyal, R. Girshick, K. He, and Piotr Dollár, "Focal Loss for Dense Object Detection," in *Proc. IEEE Int. Conf. Comput. Vision*, Oct. 2017.

[38] M. Abadi *et al.*, "Tensorflow: A System for Large-Scale Machine Learning," in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, Nov. 2016.

[39] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *Proc. Int. Conf. Learning Representations*, May 2015.

[40] ISO/IEC JTC 1/SC29/WG1 N87037. "JPEG Pleno Point Cloud Coding Common Test Conditions v3.2," in *Proc. Online Meet.*, Apr. 2020.

[41] ISO/IEC JTC 1/SC29 Joint WG11/WG1 M7201. "Microsoft Voxelized Upper Bodies – A Voxelized Point Cloud Dataset," Geneva, Switzerland, May 2016.