Learned Point Cloud Geometry Compression

Jianqiang Wang, Hao Zhu, Zhan Ma, Tong Chen, Haojie Liu, and Qiu Shen
Nanjing University

***Abstract*—This paper presents a novel end-to-end *Learned Point Cloud Geometry Compression* (a.k.a., Learned-PCGC) framework, to efficiently compress the point cloud geometry (PCG) using deep neural networks (DNN) based variational autoencoders (VAE). In our approach, PCG is first voxelized, scaled and partitioned into non-overlapped 3D cubes, which is then fed into stacked 3D convolutions for compact latent feature and hyperprior generation. Hyperpriors are used to improve the conditional probability modeling of latent features. A weighted binary cross-entropy (WBCE) loss is applied in training while an adaptive thresholding is used in inference to remove unnecessary voxels and reduce the distortion. Objectively, our method exceeds the geometry-based point cloud compression (G-PCC) algorithm standardized by well-known Moving Picture Experts Group (MPEG) with a significant performance margin, e.g., at least 60% BD-Rate (Bjöntegaard Delta Rate) gains, using common test datasets. Subjectively, our method has presented better visual quality with smoother surface reconstruction and appealing details, in comparison to all existing MPEG standard compliant PCC methods. Our method requires about 2.5MB parameters in total, which is a fairly small size for practical implementation, even on embedded platform. Additional ablation studies analyze a variety of aspects (e.g., cube size, kernels, etc) to explore the application potentials of our learned-PCGC.**

***Index Terms*—Point cloud compression, geometry, 3D convolution, classification, end-to-end learning.**

I. Introduction

POINT cloud is a collection of discrete points with 3D geometry positions and other attributes (e.g., color, opacity, etc), which can be used to represent the volumetric visual data such as 3D scenes and objects efficiently [1]. Recently, with the explosive growth of point cloud enabled applications such as 3D free viewpoint video and holoportation, high-efficiency Point Cloud Compression (PCC) technologies are highly desired.

Existing representative standard compliant PCC methodologies were developed under the efforts from the MPEG-I 3 Dimensional Graphics coding group (3DG) [1], [2], of which geometry-based PCC (G-PCC) for static point clouds and video-based PCC (V-PCC) for dynamic point clouds were two typical examples. Both G-PCC and V-PCC relied on conventional models, such as octree decomposition [3], triangulated surface model, region-adaptive hierarchical transform [4], [5], and 3D-to-2D projection. Other explorations related to the PCC are based on graphs [6], binary tree embedded with quardtree [7], or recently volumetric model [8].

In another avenue, a great amount of deep learning-based image/video compression methods [9]–[11] have emerged recently. Most of them have offered promising compression performance improvements over the traditional JPEG [12], JPEG 2000 [13], and even High-Efficiency Video Coding (HEVC) intra profile-based image compression [9], [11], [14]. These learned compression schemes have leveraged stacked DNNs to generate more compact latent feature representation for better compression [10], mainly for 2D images or video frames.

Motivated by facts that redundancy in 2D images can be well exploited by stacked 2D convolutions (and relevant non-linear activation), we have attempted to explore the possibility to use 3D convolutions to exploit voxel correlation efficiently in a 3D space. In other word, we aim to use proper 3D convolutions to represent the 3D point cloud compactly, mimicking the way that 2D image blocks can be well synthesized by stacked 2D convolutions [11], [14]. This paper focuses on static geometry compression, leaving other aspects (such as the compression of color attributes, etc) for our future study.

A high-level overview of our Learned-PCGC is given in Fig. 1a, consisting of 1) a pre-processing for point cloud voxelization, scaling, and partition; 2) a variational autoencoder (VAE) based compression network; and 3) a post-processing for proper voxel classification, inverse scaling, and extraction (for display and storage). Note that voxelization and extraction may be optional in the case that input point clouds are already in 3D volumetric presentation and not required to be stored in another non-volumetric format.

Generally, PCG data is typically voxelized for a 3D volumetric presentation. Each voxel uses a binary bit (1 or 0) to represent whether the current position at (i, j, k) is occupied as a positive and valid point (and its associated attributes). An analogous example of a voxel in a 3D space is a pixel in a 2D image. A (down)-scaling operation can be implemented to downsize input 3D volumetric model for better compression under a bit rate budget, especially at low bitrate scenarios. Corresponding (up)-scaling is required at another end for subsequent rendering.

Inspired by the successful block-based image processing, we propose to partition/divide the entire 3D volumetric model into non-overlapped *cubes*[1], each of which contains $W \times W \times W$ voxels. The compression process runs cube-by-cube. In this work, operations are contained within the current cube without exploiting the inter-cube correlation. This ensures the complexity efficiency for practical application, by offering the parallel processing and affordable memory consumption, on a cubic basis.

For each individual cube, we use Voxception-ResNet [15] (VRN) to exemplify the 3D convolutional neural network (CNN) for compact latent feature extraction. Similar as [9], [10], [14], a VAE architecture is applied to leverage *hyperpriors* for better conditional context (probability) modeling

[1]Each 3D cube is measured by its height, width and depth, similar as the 2D block represented by its height and width.

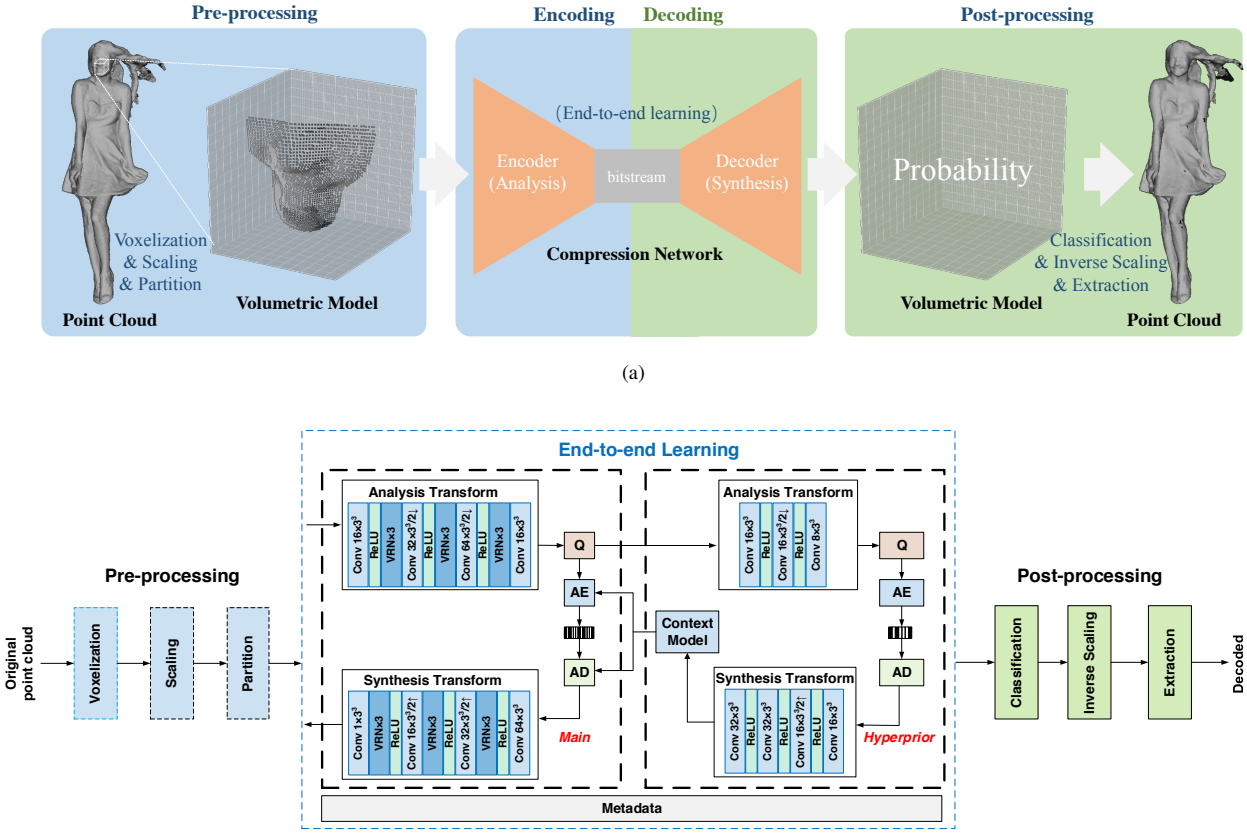


Fig. 1: **Learned-PCGC.** An illustrative overview in (a) and detailed diagram in (b) for point cloud geometry compression consisting of a pre-processing for PCG voxelization, scaling & partition, a compression network for compact PCG representation and metadata signaling, and a post-processing for PCG reconstruction and rendering. "Q" stands for "Quantization", "AE" and "AD" are Arithmetic Encoder and Decoder respectively. "Conv" denotes convolution layer with the number of the output channels and kernel size, "$\times 3$" means cascading three VRNs, "$s\uparrow$" and "$s\downarrow$" represent upscaling and downscaling at a factor of s. "ReLU" stands for the Rectified Linear Unit.

when encoding the latent features. For an end-to-end training, the weighted binary cross-entropy (WBCE) loss is introduced to measure the compression distortion for rate-distortion optimization, while an adaptive thresholding scheme is embedded for appropriate voxel classification in inference.

To ensure the model generalization, our learned-PCGC is trained using various shape models from ShapeNet [16], and is evaluated using the common test datasets suggested by MPEG PCC group and JPEG Pleno group. Extensive simulations have revealed that our Learned-PCGC exceeds existing MPEG standardized G-PCC by a significant margin, e.g., $\approx$ 67%, and 76% BD-Rate (Bjontegaard delta bitrate) gains using D1 (point2point) distance, and $\approx$ 62%, and 69% gains using D2 (point2plane) distance, against G-PCC using octree and trisoup models respectively. Our method also achieves comparable performance in comparison to another standardized V-PCC. In addition to the aforementioned objective measures, we have also reported that our method could offer much better visual quality (e.g., smoother surface and appealing details) when rendered using the surface model. This is mainly due to the inherently 3D structural representation using learned 3D transforms. A fairly small net model (e.g., 2.5MB) is used, and parallel cube processing is also doable. This offers low complexity requirement, which is friendly for hardware or embedded implementation.

The contributions of this paper are highlighted as follows:

- We have explored a novel direction to apply the learning-based framework, consisting of a pre-processing module for point cloud voxelization, scaling and partition, compression network for rate-distortion optimized representation, and a post-processing module for point cloud reconstruction and rendering, to represent point clouds geometry using compact features with the state-of-the-art compression efficiency;
- We have exemplified objectively and subjectively the efficiency by applying stacked 3D convolutions (e.g., VRN) in a VAE structure to represent the sparse voxels in a 3D space;
- Instead of directly using D1 or D2 distortion for optimization, a WBCE loss in training and adaptive thresholding

in inference are applied to determine whether the current voxel is occupied, as a classical classification problem. This approach works well for exploiting the voxel sparsity.
- Additional ablation studies have been offered to analyze a variety of aspects (e.g., partition size, kernel size, thresholding, etc) for our method to understand its capability for practical applications.

The rest of this paper is structured as follows: Section II reviews relevant studies on the compression of point clouds, learning-based image/video coding algorithms and recently emerged studies of autoencoders for point cloud processing; Our Learned-PCGC is given in Section III with systematic sketch and detailed discussions, followed by the experimental explorations and ablation studies to demonstrate the efficiency of our method; and concluding remarks are drawn in Section VI.

II. Related Work

Relevant researches of this work can be classified as point cloud geometry compression, learned image compression, and recent emerging autoencoder-based point cloud processing.

A. Point Cloud Geometry Compression

Prior PCGC approaches mainly relied on conventional models, including octree, trisoup, and 3D-to-2D projection based methodologies.

Octree Model. A very straightforward way for point cloud geometry illustration is using the octree-based representation [17] recursively. Binary labels (1, or 0) can be given to each node to indicate whether a corresponding voxel or 3D cube is positively occupied. Such binary string can be then compressed using statistical methods, with or without prediction [18], [19]. The octree-based approach has been adopted into the popular Point Cloud Library (PCL) [20], and referred to as benchmark solution extensively [21]. MPEG standard compliant G-PCC [22] has also applied the octree coding mechanism, which is known as the *octree geometry codec*. Most octree-based algorithms have shown decent efficiency for sparse point cloud compression, but limited performance for dense point cloud compression.

Mesh/Surface Model. Mesh/surface model could be regarded as the combination of point cloud and fixed vertex-face topology. Thus, an alternative approach is to use a surface model for point cloud compression, as investigated in [23], [24]. In these studies, 3D object surfaces are represented as a series of triangle meshes, where mesh vertices are encoded for delivery and storage. Point cloud after decoding is provided by sampling the reconstructed meshes. MPEG G-PCC has also included such triangulation-based mesh model, a.k.a., triangle soups representation of geometry [22] into the test model. This is known as the *trisoup geometry codec*. Such trisoup model is preferred for a dense point cloud.

Projection-Based Approach. Other attempts have tried to project the 3D object to multiple 2D planes from a variety of viewpoints. This approach can leverage existing and successful image and video codecs. The key issue to this solution are how to efficiently perform the 3D-to-2D projections. As exemplified in MPEG V-PCC [25], a point cloud is decomposed into a set of patches that are packed into a regular 2D image grid with minimum unused space. Padding is often executed to fill empty space for a piece-wise smooth image. With such projection, point cloud geometry can be converted into 2D depth images that can be compressed using the HEVC [26]. By far, V-PCC has exhibited the state-of-the-art coding efficiency compared with the G-PCC and PCL, etc, for geometry compression.

B. Learned Image Compression

Recent explosive studies [9]–[11], [27], [28] have shown that learned image compression offers better rate-distortion performance over the traditional JPEG [12], JPEG2000 [13], and even HEVC-based Better Portable Graphics (BPG)[2], etc [11], [14]. These algorithms are mainly based on the VAE structure with stacked 2D CNNs for compact latent feature extraction. Hyperpriors are used to improve the conditional probability estimation of latent features. While end-to-end learning schemes have been deeply studied for 2D image compression or even extended to the video [28], there lack systematic efforts to study effective and efficient neural operations for 3D point cloud compression. One reason is that pixels in the 2D grid are more well structured and can be predicted via (masked) convolutions, but voxels in 3D cube present more sparsity, and unstructured local and global correlation, which is usually difficult for compression.

C. Point Cloud Autoencoders

Existing point cloud representation and generation models using autoencoders serve as good references for point cloud compression. For example, Achlioptas *et al.* [29] proposed an end-to-end deep autoencoder that directly accepts point clouds for classification and shape completions. Brock *et al.* [15] introduced a voxel-based VAE architecture using stacked 3D CNNs for 3D object classification. Dai *et al.* [30] applied a 3D-Encoder-Predictor CNNs for shape completions, and Tatarchenko *et al.* [31] reported a deep CNN autoencoder for an efficient octree representation. These works are mainly developed for machine vision tasks but not for compression, but their autoencoder architectures provide references for us to represent 3D point clouds efficiently. Inspired by these studies, we try to design appropriate transforms using autoencoders for compact representation.

Quach *et al.* [32] proposed a convolutional transforms based PCG compression method recently, which is the most relevant literature to our work. When both compared with PCL, our work offers larger gains. This is mainly due to the fairly redundant features using shallow network structure with large convolutions, inaccurate context modeling of latent features, etc. We will show more details in subsequent ablation studies.

III. Learned-PCGC: An Exploration via Learning

This section details each component designed in our Learned-PCGC, shown in Fig. 1b, consisting of a pre-

[2] https://bellard.org/bpg/

processing, an end-to-end learning based compression network, and a post-processing.

A. Pre-processing

Voxelization. Point clouds may or may not be stored in its 3D volumetric representation. Thus, an optional step is converting its raw format to a 3D presentation, typically using a $(i,\ j,\ k)$-based Cartesian coordinate system. This is referred to as the *voxelization*. Given that our current focus is the geometry of point cloud in this work, a voxel at $(i,\ j,\ k)$ is set to 1, e.g., $V(i,j,k) = 1$, if it has positive attributes, and $V(i,j,k) = 0$ otherwise. Point cloud precision sets the maximum achievable value in each dimension. For instance, 10-bit precision allows $0 \leq i,j,k \leq 2^{10}-1$. PCG is referred to its volumetric representation throughout this paper unless pointed out specifically. With such a volumetric model for a PCG after voxelization, it captures inter-voxel correlations in a 3D space, which is better for us to apply the subsequent 3D convolutions to exploit the efficient and compact representation.

Scaling. Image downscaling was used in image/video compression [33] to preserve image/video quality under a constrained bit rate, especially at a low bit rate. Thus, this can be directly extended to point clouds for better rate-distortion efficiency at the low bit rate range. On the other hand, scaling can be also used to reduce the sparsity for better compression by zooming out the point cloud, where the distance between sparse points gets smaller, and point density within a fixed size cube increases. As will be revealed in later experiments, applying a scaling factor in pre-processing leads to noticeable compression efficiency gains for sparse point cloud geometry, such as Class C, and yields well-preserved performance at low bit rates for fairly dense Class A and B, shown in Fig. 5 and Table I.

In this work, we propose a simple yet effective operation via direct *downscaling* and *rounding* in advance. Let $\mathbf{X}_n = (i_n, j_n, k_n)$, $n = 1 \cdots N$ be the set of points of the input point cloud. We scale this point cloud by multiplying $\mathbf{X}_n$ with a scaling factor s, $s < 1$, and round it to the closest integer coordinate, i.e.,

$$\begin{aligned}\hat{\mathbf{X}}_n &= \mathsf{ROUND}(\mathbf{X}_n \times s) \\ &= \mathsf{ROUND}(i_n \times s, j_n \times s, k_n \times s). \end{aligned} \tag{1}$$

Duplicate points at the same coordinate after rounding are simply removed for this study. An interesting topic is to exploring the adaptive scaling within the learning network. However, it requires substaintial efforts and is deferred as our study. On the other hand, applying the simple scaling operations in pre-processing is already demonstrated as an effective scheme as will be unfolded in later experimental studies.

Partition. Typically a point cloud geometry presents a large volume of data, especially for it with large precision. It is difficult and costly to process an entire point cloud at a time. Thus, motivated by the successful block based processing pipeline adopted in popular image/video standards,

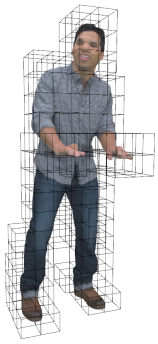


Fig. 2: Point cloud partitioned into non-overlapped cubes. Those cubes with occupied valid voxels are highlighted using bounding boxes.

we have attempted to partition the entire point cloud into non-overlapped cubes, as shown in Fig. 2. Each cube is at a size of $W \times W \times W$.

The geometry position of each cube can be signaled implicitly following the raster scanning order from the very first one to the last one, regardless of whether a cube is completely null or not. Alternatively, we can specify the position of each cube explicitly using the existing octree decomposition method, leveraging the sparse characteristics of the point cloud. Each valid cube (e.g., with at least one occupied voxel) can be seen as a super-voxel at a size of $W \times W \times W$. Thus, the number of super-voxel is limited, in comparison to the number of voxels in the same volumetric point cloud. As revealed in later ablation studies, signaling cube position explicitly using the existing octree compression method [20] only requires a very small percentage (e.g., $<$1%) overhead. In the meantime, the number of occupied voxels in each cube is also transmitted for later classification-based point reconstruction. In summary, we treat the geometry position and the number of occupied voxels of each individual (and valid) cube as the metadata that is encapsulated in the compressed binary strings explicitly.

In the current study, each cube is processed independently without exploring their intercorrelations. Massive parallelism can be achieved by enforcing the parallel cube processing. Assuming the geometry position of a specific cube is (i_c, j_c, k_c), global coordinates of a voxel can be easily converted to its local cubic coordinates,

$$\hat{\mathbf{X}}_n^{\mathsf{loc}} = \hat{\mathbf{X}}_n - (i_c \times W, j_c \times W, k_c \times W), \tag{2}$$

for the following learning-based compression.

B. Cube-based Learned-PCGC

We aim to find a more compact representation of any input cube with sparsely distributed voxels. It mainly involves the pursuit of appropriate transforms via stacked 3D CNNs, and accurate rate estimation, and a novel classification-based distortion loss measure for end-to-end optimization.

3D Convolution-based Transforms. Transforms are used for decades to represent the 2D image and video data in a more compact format, from the discrete cosine transform,

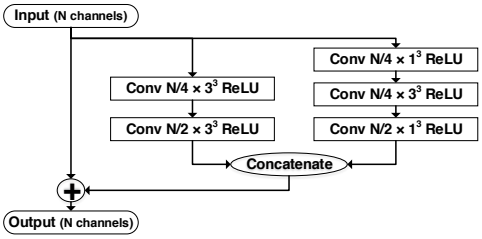


Fig. 3: Voxception-ResNet Blocks (VRN).

to recently emerged learned convolutions based approaches. Especially, those learned 2D transforms have demonstrated promising coding performance in image compression [10], [14] via stacked CNNs based autoencoders, by exploring the local and global spatial correlations efficiently.

Thus, an extension is to design proper transforms based on stacked 3D CNNs to represent the 3D point cloud. In the encoding process, forward transform is analyzing and exploiting the spatial correlation. Thus it can be referred to as the "Analysis Transform". Ideally, for any $W \times W \times W$ cube, we aim to derive compact latent features y, which are represented using a 4-D tensor with the size of $(channel, length, width, height)$. The *analysis transform* can be formulated as:

$$y = f_e(x; \theta_e), \tag{3}$$

with θ_e for convolutional weights.

Correspondingly, a mirroring *synthesis transform* is devised to decode quantized latent features $\hat{y}$ into a reconstructed voxel cube $\tilde{x}$, which can be formulated as:

$$\tilde{x} = f_d(\hat{y}; \phi_d) \tag{4}$$

with ϕ_d as its parameters.

Analysis and synthesis transformations are utilized in both main and hyper encoder-decoder pairs, shown in Fig. 1b.

In this work, we use Voxception-ResNet (VRN) structure proposed in [15] as the basic 3D convolutional unit in the main codec, for its superior efficiency inherited from both residual network [34] and inception network [35]. The architecture of VRN is illustrated in Fig. 3. In the main codec, nine stacked VRNs are used for both analysis and synthesis transform.

Given that hyperpriors are mainly used for latent feature entropy modeling, we apply three consecutive lightweight 3D convolutions (with further downsampling mechanism embedded) instead of in hyper codec. Decoded *hyperpriors* are then used to improve the conditional probability of latent features from the main codec. Details regarding the entropy rate modeling are given in Section III-D.

In this work, we have applied relative small kernels for convolutions, e.g., $1 \times 1 \times 1$ or $3 \times 3 \times 3$, which is then integrated with VRN model efficiently to capture the essential information for a compact representation. In the meantime, smaller convolution kernels are also implementation friendly with lower complexity.

C. Quantization

A simple yet effective *rounding* operation is used for feature quantization in inference, i.e.,

$$\hat{y} = \mathsf{ROUND}(y), \tag{5}$$

where y and $\hat{y}$ represent original and quantized representations respectively.

However, direct *rounding* is not differentiable for back-propagation in the end-to-end training scheme. Instead, we approximate the rounding process by adding uniform noise to ensure the differentiability,

$$\hat{y} = y + \mu, \tag{6}$$

where μ is random uniform noise ranging from $-\frac{1}{2}$ and $\frac{1}{2}$, $\hat{y}$ represents "noisy" latent representations with actual rounding error. $\hat{y}$ follows a uniform distribution $\mathcal{U}$ centered on y: $\hat{y} \sim \mathcal{U}(y-\frac{1}{2}, y+\frac{1}{2})$. Such approximation using added noise is also used in [10].

D. Entropy Rate Modeling

Entropy coding is critical for source compression to exploit statistical redundancy. Among existing approaches, arithmetic code is widely used and adopted in standards and products because of its superior performance. Thus we choose the arithmetic coding to compress each element of quantized latent feature. Theoretically, the entropy bound of the source symbol (e.g., feature element) is closely related to its probability distribution, and more importantly, accurate rate estimation plays a key role in lossy compression for rate-distortion optimization [36].

We can approximate the actual bit rate of the quantized latent feature via

$$R_{\hat{y}} = E_{\hat{y}}[-\log_2 p_{\hat{y}}(\hat{y})], \tag{7}$$

with $p_{\hat{y}}(\hat{y})$ as the self probability density function (p.d.f.) of $\hat{y}$. Rate modeling can be further improved from (7) if we can have more priors. Thus, in existing learned image compression algorithms [10], [14], a VAE structure is enforced to have both main and hyper codecs. In hyper codec, dimensions of latent features are further downscaled to provide hyperpriors z without the noticeable overhead. These hyperpriors $\hat{z}$ are decoded as the prior knowledge for better probability approximation of latent feature $\hat{y}$ when conditioned on the distribution of $\hat{z}$.

Note that the same quantization process will be applied to both latent features and hyperpriors. Following the aforementioned discussion, we can model the decoded hyperpriors (e.g., with assumed uniform rounding noise) using a fully factorized model, i.e.,

$$p_{\hat{z}|\psi}(\hat{z}|\psi) = \prod_i \left(p_{\hat{z}_i|\psi^{(i)}}(\psi^{(i)}) * \mathcal{U}(-\frac{1}{2}, \frac{1}{2}) \right)(\hat{z}_i), \tag{8}$$

where $\psi^{(i)}$ represents the parameters of each univariate distribution $p_{\hat{z}_i|\psi^{(i)}}$. Therefore, a Laplacian distribution $\mathcal{L}$ is used to approximate the p.d.f. of $\hat{y}$ when conditioned on the hyperpriors, i.e.,

$$p_{\hat{y}|\hat{z}}(\hat{y}|\hat{z}) = \prod_i \left(\mathcal{L}(\mu_i, \sigma_i) * \mathcal{U}(-\frac{1}{2}, \frac{1}{2}) \right)(\hat{y}_i). \tag{9}$$

The mean and variance parameters (μ_i, σ_i) of each element $\hat{y}_i$ are estimated from the decoded hyperpriors.

E. Rate-distortion Optimization

Rate-distortion optimization is adopted in popular image and video compression algorithms to trade-off the distortion (D) and bit rate (R). In our end-to-end learning framework, we follow the convention and define the Lagrangian loss for training, so as to maximize the overall rate-distortion performance, i.e.,

$$J_{\texttt{loss}} = R + \lambda D, \tag{10}$$

where λ controls the trade-off for each individual bit rate.

Rate Estimation: In our VAE structure based compression framework, a total rate consumption comes from the $\hat{y}$ and $\hat{z}$. Referring to (8) and (9), rate approximation can be written as

$$R_{\hat{y}} = \sum_i -\log_2(p_{\hat{y}_i|\hat{z}_i}(\hat{y}_i|\hat{z}_i)), \tag{11}$$

$$R_{\hat{z}} = \sum_i -\log_2(p_{\hat{z}_i|\psi^{(i)}}(\hat{z}_i|\psi^{(i)})). \tag{12}$$

The total rate can be easily derive via the summation, e.g., $R = R_{\hat{y}} + R_{\hat{z}}$. Here, rate spent by hyperpriors $\hat{z}$ could be regarded as the side information or overhead, occupying merely less bits than the the latent representations $\hat{y}$ in our design. Note that we only use hyperpriors for rate estimation, without including any autoregressive spatial neighbors [9], [14]. This is driven by the fact that voxels are distributed sparsely, thus, neighbors may not bring many gains in context modeling, but may break the voxel parallelism with large complexity.

Distortion Measurement: Existing image/video compression approaches use MSE or SSIM as the distortion measures. In this work, we have proposed a novel classification-based mechanism to measure the distortion instead. Such classification method fits the natural principle to extract valid point cloud data after decoding. More specifically, decoded voxels in each cube usually present in a predefined range, e.g., from 0 to 1 in this work, from 0 to 255 if 8-bit integer processing enforced. Recalling that each valid voxel in a point cloud geometry tells that this position is concretely occupied. Simple binary flag, "1" or "TRUE" often refers to the occupied voxel, while "0" or "FALSE" for the null or empty voxel. Therefore, decoded voxel needs to be classified into either 1 or 0 accordingly.

Towards this goal, we use a weighted binary cross-entropy (WBCE) measurement as the distortion in training, i.e.,

$$D_{\texttt{WBCE}} = \frac{1}{N_o}\sum^{N_o} -\log p_{\tilde{x}_o} + \alpha\frac{1}{N_n}\sum^{N_n} -\log(1-p_{\tilde{x}_n}), \tag{13}$$

where $p_{\tilde{x}} = \mathsf{sigmoid}(\tilde{x})$ is used in this work to enforce $p_{\tilde{x}} \in (0,1)$ as the probability of being occupied, $\tilde{x}_o$ represents occupied voxels, $\tilde{x}_n$ represents null voxels, and N_o, N_n represent the numbers of occupied and null voxels, respectively. Note that we do not classify voxel $\tilde{x}$ into a fixed 1 or 0, but let $\tilde{x} \in (0,1)$ to guarantee the differentiability in backpropagation used in training. Different from the standard BCE loss that weights positives and negatives equally, we calculate the mean loss of positive and negative samples separately with a hyperparameter α to reflect their relative importance and balance the loss penalty. We set α to 3 according to our experiments.

F. Post-Processing

Classification. In the inference stage, decoded voxels $\tilde{x}$ in each point cloud cube is presented as a floating number in (0,1), or an 8-bit integer in (0, 255), according to the specific implementation. Thus, we first need to classify it into binary 1 or 0. A fixed threshold can be easily applied, for example, a median value $t_h = 0.5$, however, performance often suffers as shown in Fig. 14. Instead, we propose an *adaptive thresholding* scheme for voxel classification, according to the number of occupied points in the original point cloud cube. This information is embedded for each cube as the metadata. Since $p_{\tilde{x}}$ can be also referred to as the probability of being occupied, we sort $p_{\tilde{x}}$ to extract the top k voxels, which are most likely to be occupied. Top-k selection fits the distortion criteria used in (13) for end-to-end training, e.g., minimizing the WBCE by enforcing processed voxel distribution (i.e., occupied or null) close to the original input distribution as much as possible.

Detailed discussion is given in subsequent ablation studies.

Inverse Scaling. A mirroring inverse scaling with a factor of $1/s$ is implemented in post-processing, in contrast to the scaling in pre-processing, when completing the inference of all cubes for rendering and display. This work applies a very simple linear scaling strategy. A complex scaling scheme could be used to retain reconstructed quality better, such as content adaptive scaling. This is an interesting topic to explore as our future study.

Extraction Extraction is an optional step in post-processing, as the voxelization part in pre-processing. This part is used to convert 3D point cloud into another file format for storage or exchange, such as the ASCII or polygon file format (ply) used by the MPEG PCC group. For the scenarios that original point clouds are already in 3D volumetric presentation, or decoded point clouds are used for direct display, extraction is not necessarily required.

G. Bitstream Specification

Following the above discussion, our Learned-PCGC runs iteratively for each cube in this work. It will encapsulate 1) cube position `cube_pos`, 2) the number of original occupied voxel `num_occupied_voxel`, 3) entropy coded features and hyperpriors, for each cube, into the binary bitstream for delivery and exchange. Here, we refer to part 1) and 2) as the *metadata* or (payload overhead), and 3) to as the main *payload*.

Metadata. For `cube_pos`, we simply use the octree model in [37] to indicate the location of the current cube in a volumetric point cloud. Since `num_occupied_voxel` is used for classification, we embed it directly here. For the worst case, we need $3\log_2^W$ bits for the cube at a size of $W \times W \times W$. In practice, `num_occupied_voxel` might be much less than $2^{3\log_2^W}$ because of its sparse nature.

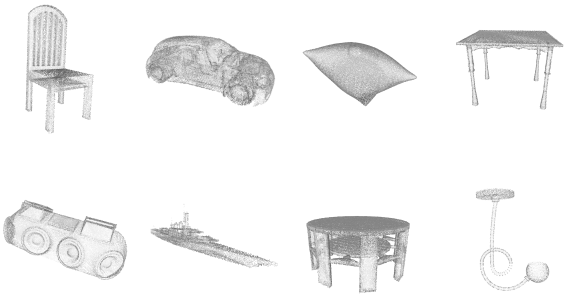


Fig. 4: Training examples from ShapeNet.

Alternatively, we can signal another syntax element, such as `max_num_occupied_voxel` for the entire point cloud to bound the number of voxels in each cube then.

Payload As seen, both features and hyperpriors are encoded for the Learned-PCGC. Syntax elements for hyperpriors and latent features (in corresponding fMaps) are encoded consecutively using arithmetic coder. Context probability of hyperpriors is based on a fully factorized distribution, while context probability of latent features is conditioned on the hyperpriors.

IV. Experimental Studies

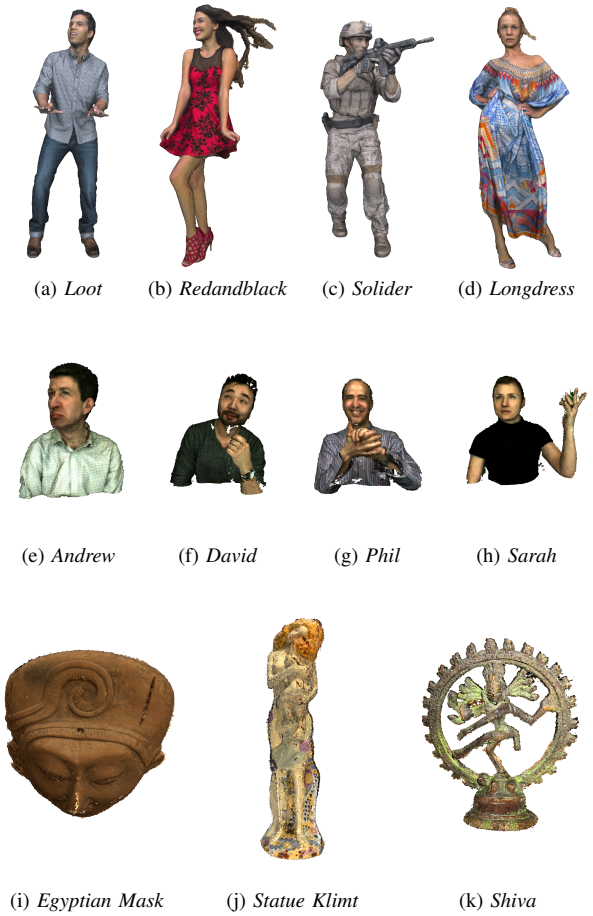
A. Training

Datasets. We randomly select $12,714$ 3D mesh models from the core dataset of ShapeNet [16] for training, including 55 categories of common objects. We sample the mesh model into point clouds by randomly generating points on the surfaces of the mesh. To ensure the uniform distribution of the points, we set the point density as 2×10^{-5} when sampling each mesh surface. Fig. 4 shows some examples of these point clouds used for training from ShapeNet. These point clouds are then voxelized on a $256 \times 265 \times 256$ occupancy space. We randomly collect $64 \times 64 \times 64$ cubes from each voxelized point cloud, resulting in 2×10^5 cubes in total used in this work.

Strategy. Loss function used for training is defined in Eq. (10). We set rate-distortion trade-off λ from 0.75 to 16 to derive various models with different compression performance. During training, we first train the model at high bit rate by setting λ to 16 and then use it to initialize model for lower bit rates. Applying the pre-trained model from higher bit rates for transfer learning not only ensures faster convergence but also guarantees reliable and stable outcomes. The learning rate is set to 10^{-5}, and the batch size is set to 8. Training iteration executes more than 2×10^5 batches for model derivation. Here, we use the Adam [38] to optimize the proposed network. We set its parameters β_1 and β_2 to 0.9 and 0.999, respectively.

B. Performance Evaluation

We apply trained models to do tests, aiming to validate the efficiency of our proposed method in subsequent discussions.



(a) *Loot* (b) *Redandblack* (c) *Solider* (d) *Longdress*

(e) *Andrew* (f) *David* (g) *Phil* (h) *Sarah*

(i) *Egyptian Mask* (j) *Statue Klimt* (k) *Shiva*

Fig. 5: **Testing Datasets.** Class A with full bodies shown in (a)-(d), Class C with inanimate objects for culture heritage in (i)-(k) used for MPEG PCC Common Test Condition (CTC) [39]; Class B with half bodies in (e)-(h) used by JPEG Pleno [40]. Note that Class C is static point cloud with one frame, and Class A and B are dynamic point clouds with multiple frames.

Testing Datasets. We choose three different test sets that are adopted by MPEG PCC [39] and JPEG Pleno [40] groups, to evaluate the performance of the proposed method, as shown in Fig. 5 and in Table I. These testing datasets present different structures and properties. Specifically, Class A (full bodies) exhibits smooth surface and complete shape, while Class B (upper bodies) presents noisy and incomplete surface (even having visible holes and missing parts). Another three inanimate objects in Class C have higher geometry precision but more sparse voxel distribution. Frames in I used for evaluation are also suggested by the MPEG PCC group.

Objective Comparison. We mainly compare our method with other PCGC algorithms, including 1) octree-based codec in Point Cloud Library (PCL) [20]; 2) MPEG PCC test model (TMs): TM13 for category 1 (static point cloud data), a.k.a., G-PCC; and 3) MPEG PCC TM2 for category 2 (dynamic content), a.k.a., V-PCC. Geometry model can be different in G-PCC method using respective *octree* or *trisoup* representation.

TABLE I: Details for Testing Datasets

Point Cloud		Points#	Precision	Frame#
A	Loot	805285	10	1200
	Redandblack	757691	10	1550
	Soldier	1089091	10	690
	Longdress	857966	10	1300
B	Andrew	279664	9	1
	David	330791	9	1
	Phil	370798	9	1
	Sarah	302437	9	1
C	Egyptian Mask	272684	12	-
	Statue Klimt	499660	12	-
	Shiva	1009132	12	-

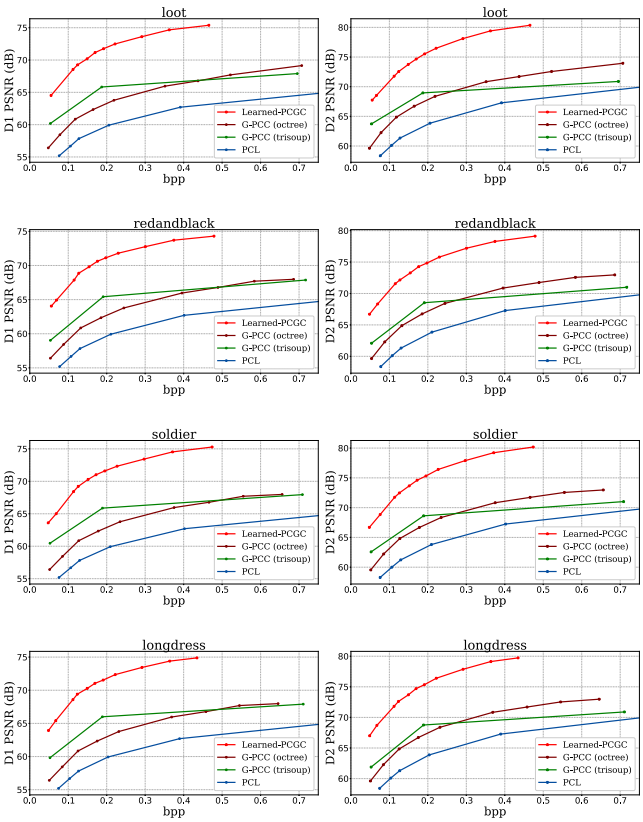


Fig. 6: R-D curves of Class A point clouds for PCL, G-PCC (octree), G-PCC (trisoup) and our Learned-PCGC: (left) D1 based PSNR, (right) D2 based PSNR.

The former one is using the octree coding mechanism similar to the implementation in PCL, and the latter is based on triangle soup representation of the geometry. They are noted as G-PCC (octree) and G-PCC (trisoup), respectively.

For a fair comparison, we have tried to enforce the similar bit rate ranges for PCL, G-PCC (octree), G-PCC (trisoup) and our method. Such bit rate range is applied as suggested by the MPEG PCC Common Test Condition (CTC) [39].

- For PCL, we use the *OctreePointCloudCompression* approach in PCL-v1.8.0 [20] for geometry compression only. We set octree resolution parameters from 1 to 64 to obtain serial rate points.
- For G-PCC, the latest TM13-v6.0 [37] is used with parameter settings following the CTC [39]. For G-PCC (octree), we set *positionQuantizationScale* from 0.75 to 0.015, leaving other parameters as default. For G-PCC (trisoup), we set *tirsoup_node_size_log2* to 2, 3, 4, and *positionQuantizationScale* to 1 for Class A and B, and 0.125 or 0.25 for Class C[3].
- Our Learned-PCC is trained in an end-to-end fashion for individual bit rates by adapting λ and scaling factor s.

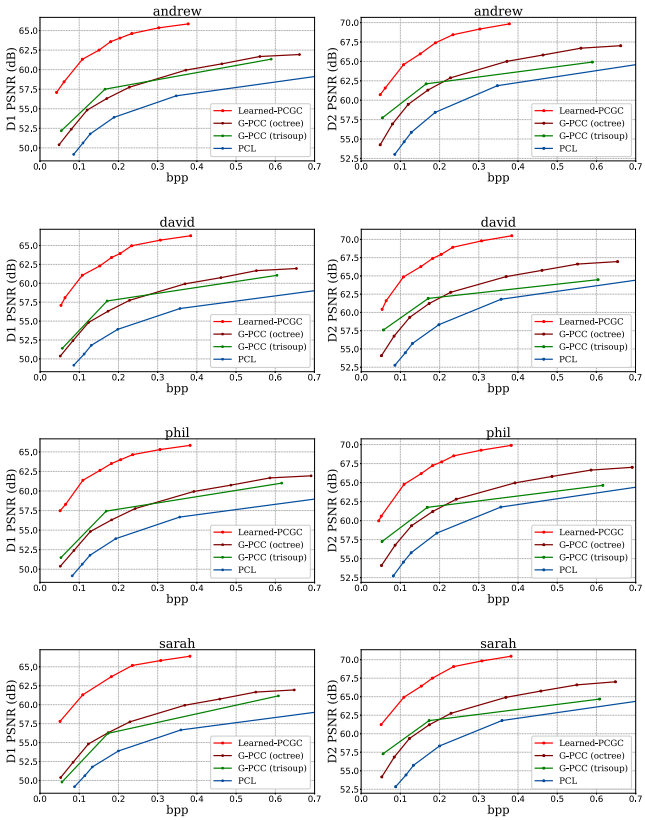


Fig. 7: R-D curves of Class B point clouds for PCL, G-PCC (octree), G-PCC (trisoup) and our Learned-PCGC: (left) D1 based PSNR, (right) D2 based PSNR.

Objective comparison is evaluated using the BD-Rate, shown in Table II. There are two distortion metrics widely used for point cloud geometry compression. One is the mean-squared-error (MSE) with point-to-point (D1-p2point) distance, and the other is the MSE with point-to-plane (D2-p2plane) distance measurement [41], [42]. Bit rate is represented using bits per input point (bpp), or bits per occupied voxel (bpov).

Our method offers averaged -88% and -82% gains against PCL, -77% and -69% gains against G-PCC (octree), -67% and -62% gains against G-PCC (trisoup), measured via respective D1 and D2 based BD-Rate. Illustrative Rate-distortion curves are presented in Figs. 6, 7, and 8.

[3]Downscaling is applied for Class C point cloud because they are typically sparse but with higher precision.

TABLE II: BD-Rate Gains against PCL, G-PCC (cotree), G-PCC (trisoup) in D1 and D2 based BR-Rate Measurement.

Point Cloud		D1 (p2point)			D2 (p2plane)		
		PCL	G-PCC (octree)	G-PCC (trisoup)	PCL	G-PCC (octree)	G-PCC (trisoup)
A	Loot	-91.50	-80.30	-68.58	-87.50	-73.49	-68.91
	Redandblack	-90.48	-79.47	-68.10	-86.70	-73.33	-68.22
	Soldier	-90.93	-79.67	-62.14	-87.07	-73.08	-67.39
	Longdress	-91.22	-80.46	-62.97	-87.34	-74.09	-68.35
	Average	**-91.03**	**-79.98**	**-65.44**	**-87.15**	**-73.49**	**-68.21**
B	Andrew	-88.64	-77.57	-74.63	-81.61	-66.79	-65.23
	David	-87.56	-75.25	-72.23	-82.52	-68.13	-66.95
	Phil	-88.31	-77.72	-75.42	-82.02	-68.74	-66.33
	Sarah	-88.62	-76.91	-79.42	-83.36	-69.51	-72.61
	Average	**-88.28**	**-76.86**	**-75.42**	**-83.30**	**-68.29**	**-67.78**
C	Egyptian Mask	-84.31	-73.53	-50.14	-85.12	-74.02	-40.80
	Statue Klimt	–83.45	-75.89	-60.33	-74.66	-62.33	-47.56
	Shiva	-77.30	-68.92	-64.91	–67.89	-56.42	-51.85
	Average	**-81.68**	**-72.78**	**-58.46**	**-75.89**	**-64.25**	**-46.73**
Overall average		**-87.48**	**-76.88**	**-67.17**	**-82.34**	**-69.08**	**-62.20**

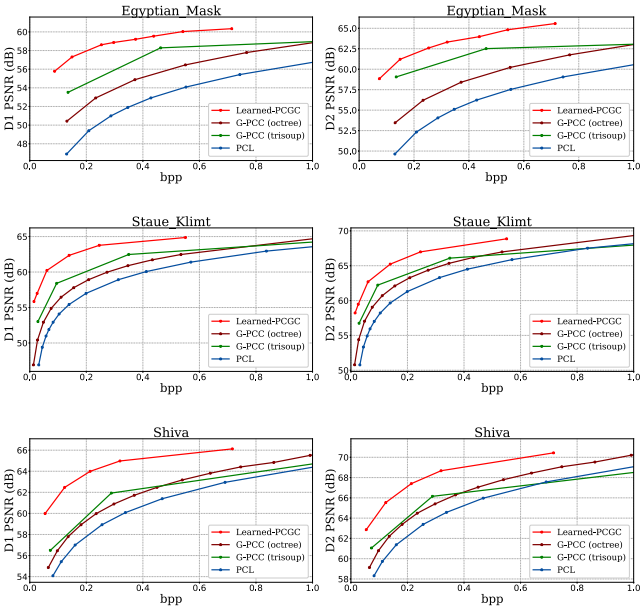


Fig. 8: R-D curves of Class C point clouds for PCL, G-PCC (octree), G-PCC (trisoup) and our Learned-PCGC: (left) D1 based PSNR, (right) D2 based PSNR.

As reported previously, our Learned-PCGC exceeds current G-PCC and PCL based geometry compression by a significant margin. For all testing Classes, e.g., dense or sparse voxel distributions, complete or incomplete surface, etc, the compression efficiency of our method consistently remains. On the other hand, our training dataset is the watertight point clouds generated from the ShapeNet [16], not directly covering the sparse voxel distribution as in Class C. However, our model still works by applying a simple scaling. All these observations justify the generalization of our method for various application scenarios.

In addition to the comparisons with those 3D model based geometry compression (e.g., PCL, G-PCC in Table II), we have also extended the discussion to the projection-based approach, e.g., V-PCC.

TABLE III: BD-Rate Efficiency of Learned-PCGC against V-PCC

Point Cloud	V-PCC	
	D1 (p2point)	D2 (p2plane)
Loot	21.00	8.59
Redandblack	-8.99	-21.87
Soldier	3.84	-7.47
longdress	16.81	3.51
Average	8.16	**-4.31**

We use the latest TM2-v6.0 [43] to demonstrate the efficiency of standard compliant V-PCC solution. For a fair comparison, we set the mode to "All-intra (AI)" and only compress the single frame of the dynamic point cloud, following the same test condition aforementioned [39]. We set a variety of quantization parameters (QP = 32, 28, 24, 20, and 16) to derive sufficient bit rates as well for coded geometry. Bit rates for geometry components (e.g., metadata, occupancy map, depth map [43]) are separated from the attributes for performance validation.

Again, our Learned-PCGC achieves comparable performance with V-PCC based geometry compression, as shown in Fig. 9. BD-Rate improvements are further put in Table III. Results have shown that averaged +8.16% D1 BD-Rate loss but -4.31% D2 BD-Rate gains are captured. V-PCC performs better on *Loot* and *Longdress*, while our Learned-PCGC works better on *Redandblack* and *Soldier*, as reported in Table III. Our analysis suggests that the more occluded region, the better compression efficiency of our Learned-PCGC. This is because our method inherently captures the voxel distribution in 3D space, regardless of occlusion or shape incompleteness that cannot be well exploited by the project-based method.

Subjective Evaluation We show the decoded point clouds from different methods and the ground truth in Figs. 10, 11,

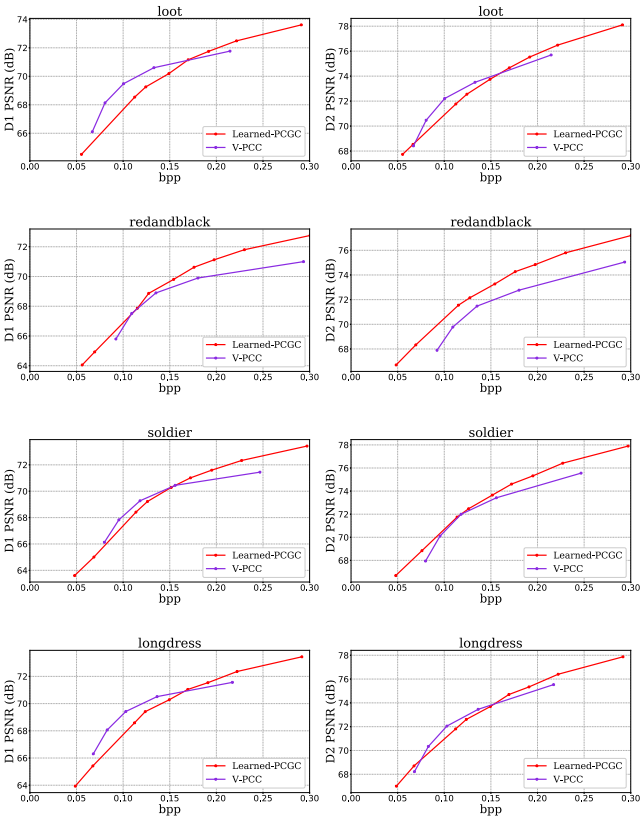


Fig. 9: Rate-Distortion Performance Comparision between V-PCC and our Learned-PCGC using Class A point clouds: (left) D1 PSNR, (right) D2 PSNR.

and 12, we recommend zooming in to see the detail. To visualize the point clouds, we first compute the normal for each point using 20 neighbor points, then we set parallel lighting in the front and render the points as Lambert unit. By this means, we could observe the detailed geometry which is more intuitive than vertex-color rendered image. We also plot the error map based on the point-to-point (P2point) D1 distance between decoded point clouds and ground truth to visualize the error distribution. We can see that our method preserves the detailed geometry and generates visually high-quality point clouds. Though V-PCC performs well in quantitative objective comparison, its reconstructed point clouds contain obvious seams as shown in the yellow dotted box, shown in zoomed-in area of Fig. 10 and 11. This is because its method encodes point clouds by projecting them to different views, so it is difficult to avoid seams when fusing projected point clouds in the decoding phase. We also find that G-PCC (tirsoup) codec may lose geometry details (e.g., visible holes shown in Figs. 10 and 11). The reconstructed point clouds of G-PCC (octree) codec and PCL are much sparser as they could only retain much fewer points at comparative bit rate budget.

An interesting observation is that our reconstructed point cloud fills some broken parts in the ground truth PC. The broken part is produced due to incomplete or failed scans. We highlight the repaired part using the blue dotted box in Fig. 11.

We think this is because we use ShapeNet [16] to generate the point cloud samples for training, where most of them are fine mesh models designed by CAD software. The high-quality training data make the distortion of our reconstruction is inclined to complete and smooth shapes with lower noise. In contrast, the distortion of other methods is inclined to random noise.

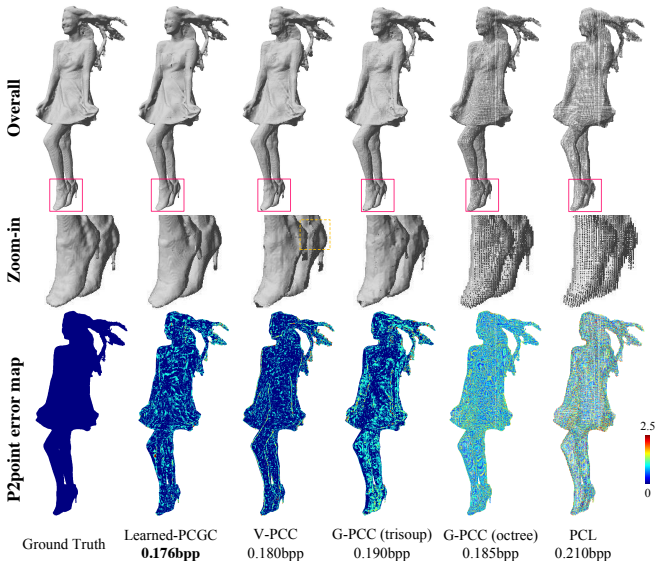


Fig. 10: Visual comparison of "redandblack" for ground truth, our Learned-PCGC, V-PCC, G-PCC (trisoup), G-PCC (octree), and PCL. Compressed bits are set closely for all methods.

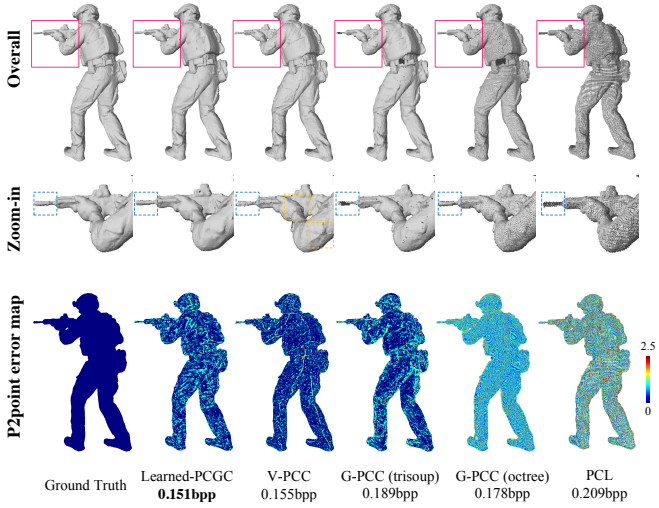


Fig. 11: Visual comparison of "soldier" for ground truth, our Learned-PCGC, V-PCC, G-PCC (trisoup), G-PCC (octree), and PCL. Compressed bits are set closely for all methods.

V. Ablation Studies

We further extend our studies by examining various aspects of our Learned-PCGC, including the partition size, hyper-

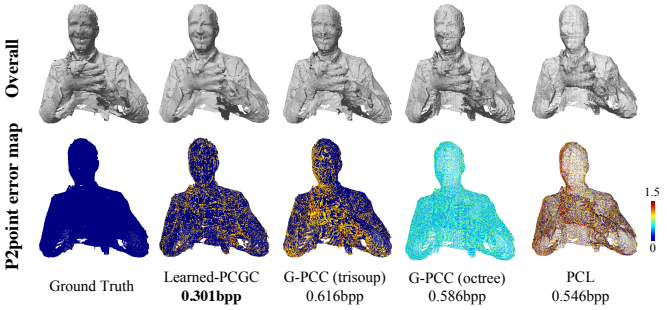


Fig. 12: Visual comparison of "phil" for ground truth, our Learned-PCGC, G-PCC (trisoup), G-PCC (octree), and PCL. Note that bits consumed by our method is about half of those existing methods.

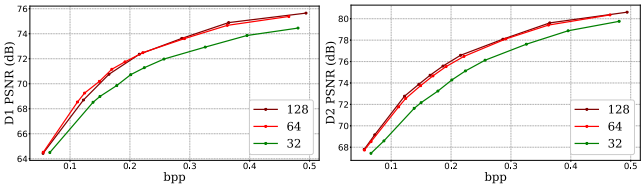


Fig. 13: Rate-distortion efficiency at different cube sizes (W). "Loot" is presented as an example.

priors, adaptive thresholding, to demonstrate the robust and reliable performance of our method.

Partition Size. Analogous to the size of the coding tree unit in HEVC, we could set different partition sizes W to explore its impact on the coding efficiency and implementation complexity for practice.

In the subsequent discussion, we have exemplified our studies using *Loot* at three different Ws, i.e., W =32, 64 and 128. Other testing materials share the similar outcomes. As illustrated in Fig. 13, BD-Rate gains about 20% from $W = 32$ to $W = 64$, but almost keeps the same from $W = 64$ to $W = 128$.

In addition to the BD-Rate, we have also provided other factors, e.g., the total number of cubes (cube#), metadata overhead (meta_bits), time (second) and memory consumption (mem) when executing the simulation, in Table IV. Time and memory consumption given here for processing each cube, are tested on a computer with an Intel i7-8700 CPU and a GTX1070 GPU (with 8G memory). All of these factors have substantial impacts on the algorithm complexity for implementation. For example, the smaller is W, the better is parallel processing with less memory consumption and computational time. However, it comes with more blocky artifacts and BD-Rate sacrifice. Thus, a good choice of W needs to balance the BD-Rate performance and implementation complexity. In this work, we choose $W = 64$.

Hyperpriors. Hyperpriors z have been used for accurate conditional entropy modeling for image compression in [10], [14]. Here we further examine its efficiency in our Learned-PCGC.

TABLE IV: Implementation Factors for Various W.

W	cube#	meta_bits (bpp)	time	mem
128	51	0.0015	0.78s	2208 MB
64	212	0.0046	0.13s	414 MB
32	790	0.0142	0.06s	252 MB

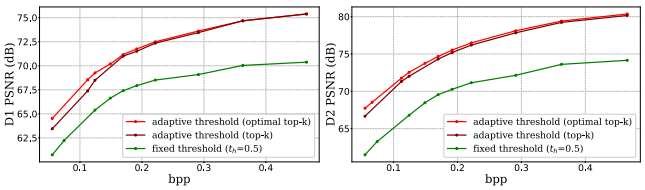


Fig. 14: BD-Rate illustration for "Loot" with adaptive thresholding based classification.

Compared with the scenario only using a *factorized entropy model* for latent representations $\hat{y}$, hyperpriors could improve the *context modeling* and lead to better rate-distortion performance with the conditional probability exploration, yielding about 14.65% BD-Rate gains from our experiments.

Adaptive Thresholding. Thresholding mechanism is applied to classify decoded voxel into a binary decision (e.g., 1 or 0) for its occupancy state. We aim to find a threshold that leads to the minimum distortion (e.g., D1 or D2) for reconstruction.

A straightforward way is to set a naïve value, such as the $t_h = 0.5$ as the global threshold to do classification for all cubes. However, performance suffers. Instead, we propose order the decoded voxels $\tilde{x}$ and select top-k ones, e.g., k = `num_occupied_voxel`, as the adaptive threshold, for each cube, leading to a noticeable BD-Rate gains in Fig. 14.

Since we are optimizing the top-k selection to minimize D1 or D2 for classification, we further deeply study whether adapting k can bring more gains by fine-tuning. Similarly as illustrated in Fig. 14, adjusting k for a fine-tuned k_f, i.e.,

$$k_f = \rho \cdot k, \quad 0.5 < \rho < 2, \tag{14}$$

would yield BD-Rate improvement. For example, on average, BD-Rate is gained about 8.9% with optimal ρ in (14), or equivalent k, when minimizing D1 distortion; while about 6.7% when minimizing D2 distortion. Optimal ρ differs for D1 and D2 measures respectively, due to their fundamental variations in distance calculation, as visualized in Fig. 15 for "Loot" at 0.11 bpp. More voxels are selected for optimal D1 distortion measurement, e.g., ρ =1.14, while less voxels, e.g., ρ = 0.91 are used for better reconstruction for D2 distortion. It indicates that D2 measurement is more suitable for sparser point cloud.

Convolution Kernels. Our Learned-PCGC, including both main and hyper encoder-decoder pairs, requires 658,092 parameters in total for all embedded convolutions. In the current implementation, each parameter is buffered using 4-byte floating format. It is about $\approx$ 2.52MB storage, which is fairly small on-chip buffer requirement compared with other popular algorithms, such as AlexNet [44] with 60 Million parameters, or GoogleNet [45] using 4 Million parameters.

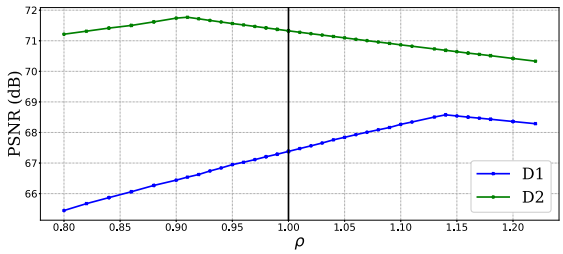


Fig. 15: Optimizing top-k selection for voxel classification to minimise the D1 or D2 distortion.

Our experiments have also revealed that current stacked VRN with small convolutional kernels and deep layers offer much better performance compared with an alternative approach using the shallow network with larger convolutional kernel sizes. This is mainly because that larger convolutions can not efficiently capture the spatial information due to sparse spatial distribution of voxels in a 3D space. But, deeper layers (with down-scaling) offers an effective way to exploit correlation in a variety of scales.

VI. Conclusion And Future Work

A learning-based point cloud geometry compression method, so-called Learned-PCGC, is presented in this work, which consists of stacked 3D convolutions for the exaction of latent features and hyperpriors, a VAE structure for accurate entropy modeling of latent features, and a weighted BCE loss in training and an adaptive thresholding scheme in inference for correct voxel classification.

We have demonstrated the state-of-the-art efficiency of proposed Learned-PCGC, for point cloud geometry compression, objectively, and subjectively, in comparison to those existing standardized methods, for example, over 62% and 67% BD-Rate gains over G-PCC (trisoup), and over 69% and 76% BD-Rate gains over G-PCC (octree), when the distortion is measured using D2 or D1 criteria respectively. On the other hand, our method also provides comparable compression efficiency against the projection-based MPEG V-PCC. Subjective evaluations have also evident the superior performance of our proposed method with noticeable perceptual improvements. Additional ablation studies deeply dive into a variety of aspects of our proposed method by carefully analyzing the performance and efficiency.

As for future studies, there are several interesting avenues to explore. For example, recent PointConV [46] might be borrowed to improve the efficiency of convolutions for the point cloud. On the other hand, traditional distortion measurements, such as D1 and D2, still suffer from a low correlation with subjective assessment. A better objective metric is highly desired.

VII. Acknowledgement

We are deeply grateful for the constructive comments from anonymous reviewers to improve this paper.

References

[1] S. Schwarz, M. Preda, V. Baroncini, M. Budagavi, P. Cesar, P. A. Chou, R. A. Cohen, M. Krivokuća, S. Lasserre, Z. Li *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, 2019. 1

[2] M. D. Graphics, "Call for proposals for point cloud compression v2," *ISO/IEC JTC1/SC29/WG11 MPEG2017/N16763*, 2017. 1

[3] D. Meagher, "Geometric modeling using octree encoding," *Computer graphics and image processing*, vol. 19, no. 2, pp. 129–147, 1982. 1

[4] R. L. de Queiroz and P. A. Chou, "Compression of 3d point clouds using a region-adaptive hierarchical transform," *IEEE Transactions on Image Processing*, vol. 25, no. 8, pp. 3947–3956, Aug 2016. 1

[5] ——, "Transform coding for point clouds using a gaussian process model," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3507–3517, July 2017. 1

[6] D. Thanou, P. A. Chou, and P. Frossard, "Graph-based compression of dynamic 3d point cloud sequences," *IEEE Transactions on Image Processing*, vol. 25, no. 4, pp. 1765–1778, April 2016. 1

[7] B. Kathariya, L. Li, Z. Li, J. Alvarez, and J. Chen, "Scalable point cloud geometry coding with binary tree embedded quadtree," in *2018 IEEE International Conference on Multimedia and Expo (ICME)*, July 2018, pp. 1–6. 1

[8] P. A. Chou, M. Koroteev, and M. Krivokua, "A volumetric approach to point cloud compression, part i: Attribute compression," *IEEE Transactions on Image Processing*, pp. 1–1, 2019. 1

[9] H. Liu, T. Chen, P. Guo, Q. Shen, and Z. Ma, "Gated context model with embedded priors for deep image compression," *arXiv preprint arXiv:1902.10480*, 2019. 1, 3, 6

[10] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational image compression with a scale hyperprior," *arXiv preprint arXiv:1802.01436*, 2018. 1, 3, 5, 11

[11] D. Minnen, J. Ballé, and G. D. Toderici, "Joint autoregressive and hierarchical priors for learned image compression," in *Advances in Neural Information Processing Systems*, 2018, pp. 10 771–10 780. 1, 3

[12] G. K. Wallace, "The jpeg still picture compression standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii–xxxiv, Feb 1992. 1, 3

[13] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The jpeg 2000 still image compression standard," *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 36–58, Sep. 2001. 1, 3

[14] H. Liu, T. Chen, P. Guo, Q. Shen, X. Cao, Y. Wang, and Z. Ma, "Non-local attention optimized deep image compression," *arXiv preprint arXiv:1904.09757*, 2019. 1, 3, 5, 6, 11

[15] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Generative and discriminative voxel modeling with convolutional neural networks," *arXiv preprint arXiv:1608.04236*, 2016. 1, 3, 5

[16] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su *et al.*, "Shapenet: An information-rich 3d model repository," *arXiv preprint arXiv:1512.03012*, 2015. 2, 7, 9, 10

[17] C. L. Jackins and S. L. Tanimoto, "Oct-trees and their use in representing three-dimensional objects," *Computer Graphics & Image Processing*, vol. 14, no. 3, pp. 249–270, 1980. 3

[18] R. Schnabel and R. Klein, "Octree-based point-cloud compression," in *Eurographics*, 2006. 3

[19] Y. Huang, J. Peng, C. C. Kuo, and M. Gopi, "A generic scheme for progressive point cloud coding," *IEEE Transactions on Visualization & Computer Graphics*, vol. 14, no. 2, pp. 440–453, 2008. 3

[20] "Point cloud library," https://pointclouds.org/, 2011. 3, 4, 7, 8

[21] J. Kammerl, N. Blodow, R. B. Rusu, M. Beetz, E. Steinbach, and S. Gedikli, "Real-time compression of point cloud streams," in *IEEE International Conference on Robotics & Automation*, 2012. 3

[22] K. Mammou, P. A. Chou, D. Flynn, M. Krivokuca, O. Nakagami, and T. Sugio, "G-pcc codec description v2," *ISO/IEC JTC1/SC29/WG11 N18189*, 2019. 3

[23] A. Anis, P. A. Chou, and A. Ortega, "Compression of dynamic 3d point clouds using subdivisional meshes and graph wavelet transforms," in *IEEE International Conference on Acoustics*, 2016. 3

[24] E. Pavez, P. A. Chou, R. L. D. Queiroz, and A. Ortega, "Dynamic polygon clouds: Representation and compression for vr/ar," *APSIPA Transactions on Signal and Information Processing*, vol. 7, 2018. 3

[25] V. Zakharchenko, "Algorithm description of mpeg-pcc-tmc2," *ISO/IEC JTC1/SC29/WG11 MPEG2018/N17526*, 2018. 3

[26] G. J. Sullivan, J. R. Ohm, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Transactions on Circuits & Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2013. 3

[27] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimized image compression," *arXiv preprint arXiv:1611.01704*, 2016. 3

[28] T. Chen, H. Liu, Q. Shen, T. Yue, X. Cao, and Z. Ma, "Deepcoder: a deep neural network based video compression," in *2017 IEEE Visual Communications and Image Processing (VCIP)*. IEEE, 2017, pp. 1–4. 3

[29] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, "Learning representations and generative models for 3d point clouds," *arXiv preprint arXiv:1707.02392*, 2017. 3

[30] A. Dai, C. Ruizhongtai Qi, and M. Nießner, "Shape completion using 3d-encoder-predictor cnns and shape synthesis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5868–5877. 3

[31] M. Tatarchenko, A. Dosovitskiy, and T. Brox, "Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2088–2096. 3

[32] M. Quach, G. Valenzise, and F. Dufaux, "Learning convolutional transforms for lossy point cloud geometry compression," *arXiv preprint arXiv:1903.08548*, 2019. 3

[33] A. M. Bruckstein, M. Elad, and R. Kimmel, "Down-scaling for better transform compression," *IEEE Transactions on Image Processing*, vol. 12, no. 9, pp. 1132–1144, Sep. 2003. 4

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778. 5

[35] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017. 5

[36] G. J. Sullivan and T. Wiegand, "Rate-distortion optimization for video compression," *IEEE signal processing magazine*, vol. 15, no. 6, pp. 74–90, 1998. 5

[37] "Mpeg-pcc-tmc13," http://mpegx.int-evry.fr/software/MPEG/PCC/TM/mpeg-pcc-tmc13, accessed: 2019. 6, 8

[38] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. 7

[39] "Common test conditions for point cloud compression," *ISO/IEC JTC1/SC29/WG11 N18474*, 2019. 7, 8, 9

[40] L. Charles, C. Qin, O. Sergio, and A. C. Philip, "Microsoft voxelized upper bodies - a voxelized point cloud dataset," *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document m38673/M72012*, May 2016. 7

[41] D. Tian, H. Ochimizu, C. Feng, R. Cohen, and A. Vetro, "Geometric distortion metrics for point cloud compression," in *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2017, pp. 3460–3464. 8

[42] R. Mekuria, S. Laserre, and C. Tulvan, "Performance assessment of point cloud compression," in *2017 IEEE Visual Communications and Image Processing (VCIP)*. IEEE, 2017, pp. 1–4. 8

[43] "Mpeg-pcc-tmc2," http://mpegx.int-evry.fr/software/MPEG/PCC/TM/mpeg-pcc-tmc2, accessed: 2018. 9

[44] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105. 11

[45] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9. 11

[46] W. Wu, Z. Qi, and L. Fuxin, "Pointconv: Deep convolutional networks on 3d point clouds," *arXiv preprint arXiv:1811.07246*, 2018. 12