# ADAE: ADVERSARIAL DISTRIBUTED SOURCE AUTOENCODER FOR POINT CLOUD COMPRESSION

*Simone Milani*

Dept. of Information Engineering, University of Padova, Padova, Italy
e-mail: simone.milani@dei.unipd.it

## ABSTRACT

The current paper presents an adversarial autoencoding strategy for voxelized point cloud geometry based on the principles of distributed source coding. The encoder characterizes the input voxel blocks with an array of hash bytes while the decoder combines them with side information blocks in order to reconstruct the original data. The reconstruction process is optimized by classifying the reconstructed block with an adversarial discriminator in order to make the recovered data as close as possible to an original block. Experimental results show that the proposed solution generalizes well while obtaining better coding performance with respect to other state-of-the-art solutions and allowing high flexibility in rate shaping and decoding operations.

***Index Terms***— point cloud compression, adversarial autoencoder, distributed source coding, decoding loss

## 1. INTRODUCTION AND RELATED WORKS

Recent years have witnessed a growing interest in point cloud (PC) coding and processing strategies. This attention has been nurtured by the possibility of acquiring clouds of 3D points in an easy and real-time manner with different types of sensors and systems, as well as by the wide range of possible applications from automotive to cultural heritage. Unfortunately, the massive amount of acquired data and the need of visualize them on heterogeneous devices has highlighted the need to effective and flexible compression strategies [1, 2].

Several coding solutions have been proposed so far re-adapting previous coding solutions that were designed for 2D data like images and videos Among these, it is worth mentioning the octree-based schemes [3, 4, 5] or coding engines employing a spatially-decorrelating transforms [6, 7, 8].

Other solutions rely on graph-based data processing [9, 10, 11], while other projection-based algorithms map spatialized data (like color information or attributes) to a texture image, which is then coded with traditional image/video coding strategies [12, 2, 13]. In case of dynamic point clouds, temporal correlation is considered as well in order to predict the information to be coded with respect to previous points [14, 15]. Moreover, during the last years, different works have highlighted that clustering and semantic processing of point cloud data enables optimizing the visualization routines and reducing the allocated bit rates; as a matter of fact, segmentation and partitioning algorithms enable some of these coding schemes to re-organize point cloud data and enhance the rate-distortion performance [16, 17, 18]

The work has been partially funded by the University of Padova SID project "SartreMR".

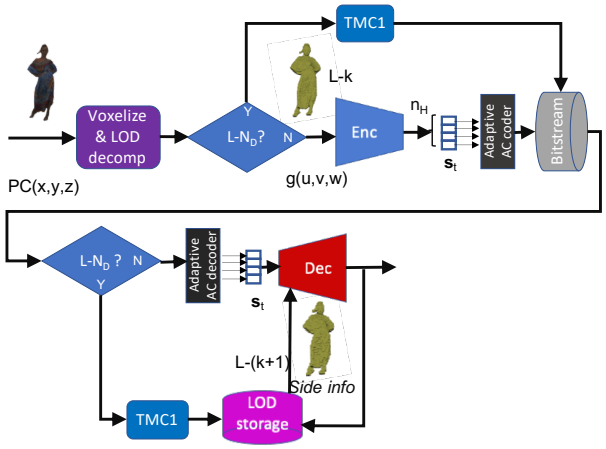



**Fig. 1**. Block diagram for the proposed encoding/decoding scheme.

The recent widespread of deep neural networks have triggered the design and the adoption of compression strategies based on deep learning (DL). Among the first approaches, it is worth mentioning some of the first image compression solutions based on autoencoders [19, 20] and RNNs [21]. End-to-end solutions have also been used for rate optimization purposes, like in [22]. Following strategies have been addressing other types of media such as video sequences [23], local descriptors [24]. and finally, 3D point cloud as well [25].

Autoencoders have proved to be extremely effective in the compression of point clouds [26]. The solution in [25] presents a three-dimensional convolutional autoencoder which allows outperforming the compression strategy of PCL library [27]. Instead, the work in [28] exploits the correlation among different Levels-Of-Detail (LODs) and implements the principles of distributed source coding (DSC) in a UNET hourglass architecture [29]. Following this trend, the solution in [30] implements an adversarial autoencoder for 3D representation, while Yang *et al.* in [31] include a folding operations to handle irregularly-distributed point clouds.

This work presents an adversarial training strategy for distributed source autoencoders, which is implemented by introducing a reconstructed-vs.-original detector for the reconstructed data in the training phase to regularize the decoder optimization and maximize the quality of the final point cloud. The proposed solution is applied to the approach in [28] combining the coding efficiency of a DSC-based network with the regularization capacity of the adversarial unit. In a nutshell, the main innovations brought by this paper can be summarized as follows:

- the presentation of an adversarial distributed source coding scheme for point cloud geometry data;

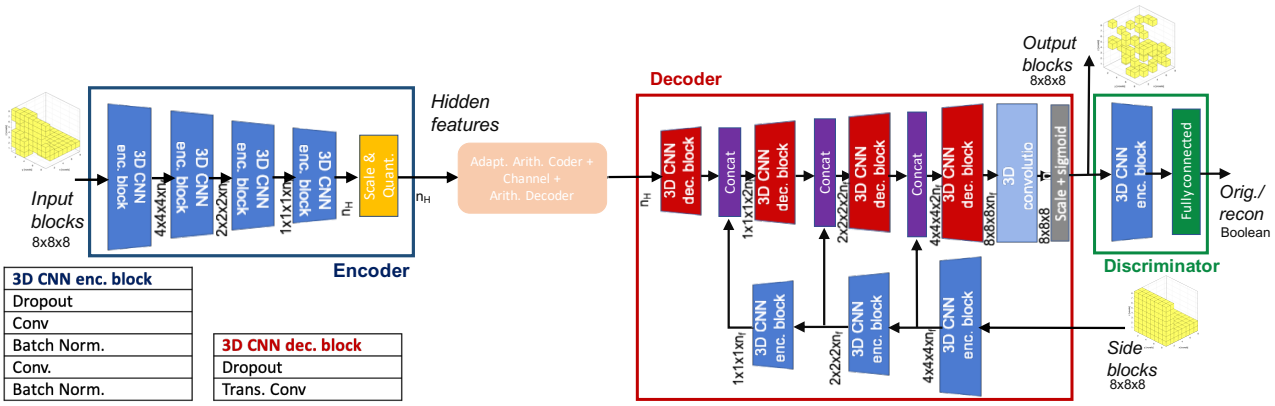



**Fig. 2**. Architecture of the Adversarial Distributed source Autoencoder.

- the introduction of novel loss functions to optimize the different elements in the autoencoder scheme;
- the inclusion of a traditional arithmetic coding scheme to reduce the residual correlation depending on the block-based processing.

In the following, Section 2 describes the overall architecture and the training strategy, while Section 3 reports the coding performance. Final conclusions are drawn in Section 4.

## 2. OVERVIEW OF THE COMPRESSION SCHEME

The proposed scheme is derived from the syndrome-based (or distributed) autoencoder reported in [28], which applies a traditional DSC scheme with no feedback channel ( see [32, 33, 34] for some examples) into an autoencoder architecture.

The main idea is to code each point cloud element independently and to decode it using a lower resolution reconstruction (available at the decoder) as side information. The decoder must be able to use multiple different blocks as side information provided that they are enough correlated with the one to be decoded.

Indeed, the encoder generates some hash symbols that can be associated to parity bits in a block channel decoder; it is possible to state that the hash values permit *correcting* the side information into a reconstructed version of the original block.

The general processing pipeline is reported in Fig. 1 and can be summarized as follows. Adopting a strategy similar to [25], the coordinates $x, y, z$ of the input point cloud $PC(x, y, z)$ are quantized and represented as $N$ bits integers; in this way, the total number of 3D points can be reduced and represented by a voxel volume $g(u, v, w)$ sized $N \times N \times N$[1]. This voxel volume represents the input data at the highest LOD (here denoted as L-0). Each voxel $g(u, v, w)$ assumes the value 1 or 0 depending on whether there is at least one point of $PC(x, y, z)$ associated to it. A progressive and iterative resampling of the voxel grid with factor 2 (along each dimension) generates different lower resolution versions of the original data until a minimum level L-$N_D$, which depends on the original value $N$ and to the desired decomposition level. In this paper, the LOD obtained at the $k$-th iteration of the resamplig will be denoted as $L$-$k$ and is sized $(N/2^k)^3$. The lowest LOD L-$N_D$ can be coded using a standard coding strategies (in this case, TMC1).

[1]In this paper, this resolution will be denoted as $N^3$ for the sake of conciseness.

Note that LOD $L$-$k$ can be approximated interpolating the data in $L$-$(k+1)$, e.g., the volume $g(u, v, w)$ at resolution $L0$ can be approximated interpolating $L$-1 with factor 2 generating the volume $g'(u, v, w)$. The proposed coding scheme shows that this approximation can be improved by including in the reconstruction process some additional hash or syndrome data; in this case, the interpolated LOD will act as side information in the DSC decoder.

In order to generate the hash data, the input volume $g(u, v, w)$ is then divided into blocks $\mathbf{b}_t$ with resolution $8 \times 8 \times 8$ voxels. Each block $\mathbf{b}_t$ is processed by a convolutional encoder which generates a byte array $\mathbf{s}_t = [s_i]$, $i = 0, \ldots, n_H - 1$. All the $i$-th features $s_{n,i}$ for all the blocks $\mathbf{s}_t$ are then gathered together and compressed using a 255-symbols adaptive Arithmetic coder (see Fig. 1).

The generated bit stream is then processed by the decoder which recovers the string $\mathbf{s}$ and decode it using the corresponding block in the previous LOD $(L-1)$ as side information. This scheme can be hierarchically iterated multiple times at different resolution layers until it reaches the minimum decomposition LOD $(L - N_D)$.

According to block channel code properties, this coding strategy enables a higher flexibility in the use of side information: whenever the number $n_s$ of syndrome values is enough with respect to the correlation between $\mathbf{b}_t$ and $\mathbf{b}_t^s$, the DSC decoder is capable of a perfect reconstruction (i.e., $\hat{\mathbf{b}}_t \equiv \mathbf{b}_t$). Following this rationale, provided that the correlation bound is respected, it could be possible to reconstruct LOD $L0$ from $L2$ instead of $L1$. This is possible whenever the model interpolated from resolution L-$(N/2^2)^3$ to $N^3$ is sufficiently close to $g(u, v, w)$. Moreover, in case of dynamic sequences, it is possible to use PC data of previous time instants to decode the current point clouds.

In the following sections, further details about the designed adversarial coding strategy will be provided.

### 2.1. Adversarial Distributed source Autoencoding (ADAE)

Adversarial autoencoders are among the most widely-diffused Generative Adversarial Networks (GANs). Their usual structure consists in a set of encoding layers generated a hidden representation, which is then processed by a decoding phase that aims at reversing the operations of the encoder phase minimizing the final distortion on the reconstructed data. Between these two sections, an adversarial role is played by a discriminator, which takes in input the hidden representation and classify it. In the iterative training phase of adversarial autoencoders, the autoencoder is tuned in order to generate finer and finer data that are capable of tricking the discriminator and leading

it to wrong classifications; at the same time, the discriminator is update in order to prevent this eventuality and pair the changes in the autoencoders. This process can be modeled as an iterative multi-stage non-cooperative game where one of the player (the autoencoder) aims at minimizing the gain of the other (the discriminator). As a result, the hidden representations are uniformly distributed in the vector space enabling an efficient compression.

Following this idea, the generic distributed source autoencoder architecture was modified as it is reported in Fig. 2 and described in detail by the following subsections.

#### *2.1.1. Encoder*

The encoder phase is made of 4 stacks of layers (named "3D CNN encoder block") which correspond to a Dropout layer (excluded in the first stack) followed by two three-dimensional convolutional layers with ReLU activation functions; each convolutional layer is characterized by $n_f$ filters and followed by a batch normalization stage. Kernel sizes are $3 \times 3 \times 3$ with padding equal to 1, while dropout probability is $0.5$; the number of filter is proportional to the number of syndrome symbols that are to be generated ($n_s$).

After the second batch normalization, a max pooling layer with width and stride equal to $2 \times 2 \times 2$ concludes the stack (made exception for the final one)

At the end, the generated hidden variables are reshaped into an array whose values are rescaled to the range $[0, 255]$ and casted to a `uint8` type array $\mathbf{s}$.

#### *2.1.2. Decoder*

The decoder implements a hourglass architecture with skip connections similar to UNET [29]; the proposed solution departs from the original UNET scheme since encoding and decoding phases are not paired with the same input/output. The input of the hourglass encoder is an $8\times8\times8$ side information voxel block $\mathbf{b}_t^s$ (see the decoder diagram in Fig. 2), but the central hidden variables are replaced by the hash array $\mathbf{s}_t$ received from the encoder. The $\mathbf{s}_t$ is processed by a sequence of layer stacks where Dropout is followed by a ReLU-activated transposed convolution with $n_f$ filters. After each decoding layers stack a concatenation unit joins the interpolated data with the corresponding depth layer generated from the side information $\mathbf{b}_t^s$. The final unit is a simple 3D convolutional layer that generates a reconstructed $8 \times 8 \times 8$ block of real values. These are then rescaled in the range $[0, 1]$ and processed by a sigmoid (in order to implement a differentiable step function).

The concatenation of encoder and decoder creates a distributed source autoencoder structure similar to that reported in [28]. One of the main innovations brought by the current paper is the inclusion of a discriminator used to implement an adversarial learning process. More details will be reported in the following subsection.

#### *2.1.3. Discriminator*

The final unit is a discriminator block that aims at distinguishing the original blocks from those reconstructed by the autoencoder. Note that in traditional adversarial autoencoders the input of the discriminator is the hidden layer. In this case, since the hash array $\mathbf{s}_t$ does not directly characterize the input block $\mathbf{b}_t$ (since it is intended to correct the side information block $\mathbf{b}_t^s$), the discriminator processed the reconstructed block $\hat{\mathbf{b}}_t$ directly. The discriminator combines a 3D CNN encoder stack and a fully connected layer with sigmoid activation function. The output is a binary variable $\mathbf{b}_t$ stating whether the input block is reconstructed or not.

### 2.2. Training phase

The training phase was carried on in multiple stages depending on the involved units.

#### *2.2.1. Autoencoder*

At first, the basic autoencoder is pretrained minimizing the loss function $\mathcal{L}_{AE}(\mathbf{b}_t, \hat{\mathbf{b}}_t)$ that can be defined as

$$\begin{aligned}\mathcal{L}(\mathbf{b}_t, \hat{\mathbf{b}}_t) = \quad & \lambda \left\| \mathbf{b}_t - \hat{\mathbf{b}}_t \right\|_2 + \frac{(1-\lambda)}{2}\, d_F(n_H, \mathbf{b}_t, \hat{\mathbf{b}}_t) \cdot \\ & \cdot \left[ d_P(\mathbf{b}_t, \hat{\mathbf{b}}_t) + d_P(\hat{\mathbf{b}}_t, \mathbf{b}_t) \right]\end{aligned} \tag{1}$$

which combines the Mean Square Error between $\mathbf{b}_t$ and $\hat{\mathbf{b}}_t$, the decoding failure coefficient $d_F$, and the distance point-to-plane $d_P\left(\mathbf{b}, \hat{\mathbf{b}}_t\right)$. This latter distance can be modelled via the equation

$$d_P\left(\mathbf{b}, \hat{\mathbf{b}}_t\right) = \sum_{u,v,w} \frac{|p_1 u + p_2 v + p_3 w + p_4|}{\sqrt{(p_1^2 + p_2^2 + p_3^2)}}\, \hat{\mathbf{b}}_t(u, v, w) \tag{2}$$

where $p_1, p_2, p_3, p_4$ are the parameters that define the plane fitting $\mathbf{b}_t(u, v, w)$. The purpose of this metric is to prevent the decoder from reconstructing geometrically-inconsistent blocks keeping the reconstructed points close to the fitted plane.

The decoding failure coefficient is instead computed considering that in a block channel code the number parity bits determines how many errors can be corrected in the transmitted codewords; similarly, the number $n_H$ of hash bytes $\mathbf{s}_t$ controls the amount of voxels in $\mathbf{b}_t^s$ that can be corrected into $\mathbf{b}_t$. As a matter of fact, it is possible to write that

$$d_F(n_H, \mathbf{b}_t, \hat{\mathbf{b}}_t) = \max\left\{ \frac{2 \cdot \|\mathbf{b}_t - \hat{\mathbf{b}}_t\|_0}{n_H}, 1.0 \right\} \tag{3}$$

where $\|\cdot\|_0$ is the $L_0$ norm. This term is deeply related to the DSC decoding paradigm; in this way, it is possible to weight the different blocks in each minibatch depending on its decodability.

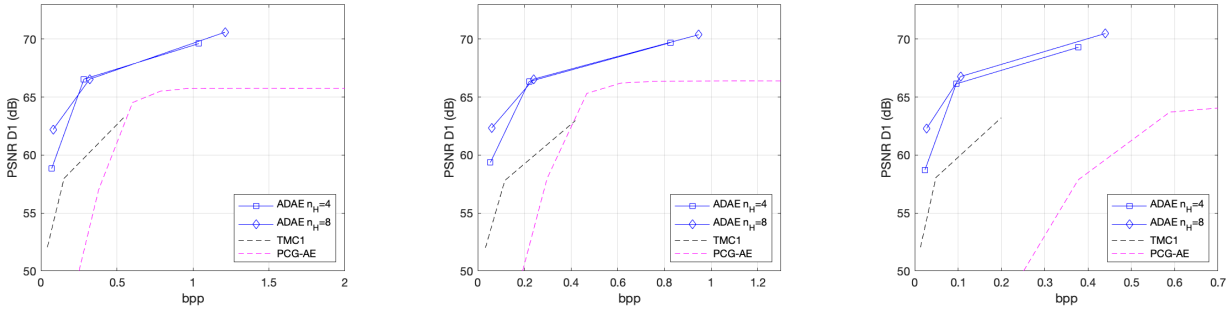
#### *2.2.2. Discriminator*

As for the discriminator, the loss function $L_D(\mathbf{b}_t, \mathbf{b}_t^r)$ corresponds to the binary cross-entropy function between the actual block label $\mathbf{b}_t^r$ (real=1 or reconstructed=0) and the outcome of the classifier $\mathbf{b}_t$. The training process operates as follows: 1) at first the autoencoder is pre-trained minimizing the loss function $\mathcal{L}_{AE}$. Then, a set of reconstructed blocks $\hat{\mathbf{b}}_t$ is generated and combined with the original blocks $\mathbf{b}_t$ in order to create a trainng set for the discriminator. Then, the discriminator block is trained minizing the loss $\mathcal{L}_D$.

#### *2.2.3. Adversarial training*

The last training phase concerns the adversarial refinement of the autoencoder parameters following the usual GAN optimization strategy, i.e., by alternatively optimizing the encoder parameters and the discriminator.

In the first phase of each iteration, network parameters for autoencoder are modified in order to maximize $\mathcal{L}_D$. In the second iteration, autoencoder parameters are fixed and the discriminator is trained minimizing $\mathcal{L}_D$.




(a) (b) (c)

**Fig. 3**. PSNR D1 (dB) vs. rate (bpp) performance on `longdress` (a), `queen` (b), and `house_without_roof` (c) models for ADAE, TMC1, and PCG-AE with 32 filters. Results for ADAE are reported for $n_H = 4, 8$.

## 3. EXPERIMENTAL RESULTS

A training dataset of static point cloud (acquired with different strategies) was adopted from the MPEG G-PCC repository [1]. Following the same training strategy in [1], the coordinates of the input data were rescaled and quantized in a 10-bits representation in order to have a uniform format for the input data. Each point cloud was then voxelized into a $N^3$ volume, and each $8 \times 8 \times 8$ block $\mathbf{b}_t$ was then associated with a corresponding block $\mathbf{b}'_t$ from the interpolated voxel volume of the previous LOD.

Similarly to the approach in [25], models `soldier`, `redandblack`, `loot`, `facade09`, `facade15`, `facade64`, `frog67`, `shiva35` were included in the training set, while models `long`, `queen`, `house_without_roof` are part of the test set. Note that training and test models are completely different and include point clouds with heterogeneous characteristics.

Training data were utterly partitioned into two parts: 55 % of the blocks were assigned to the training set, while the remaining were used as validation.

The quality of the reconstructed point cloud is measured using the PSNR D1 metric, while the bit rate is parameterized by the bit-per-point value.

Different network parameter values were trained for different resolutions ($N = 8, 9, 10$) and number $n_H$ of hash variables. Training was carried on a NVIDIA GeForce GTX 1070S GPU in 100 epochs with patience threshold of 10 (minimum difference $1e-04$) and varying the $\lambda$ parameter. Best rate-distortion results were obtained with $\lambda = 0.9$., and after an extensive set of trials, the number of filters $n_f$ was set to $2 \cdot n_H$ minimizing the final training loss.

Figure 3 reports the PSNR D1 values (expressed in dB) versus the bit rate (expressed in bpp) for different coding schemes and point cloud models. Graphs compare the proposed ADAE scheme, with TMC1 [35] (being one of the reference codec for point cloud compression), and the PCG-AE solution from [25] (which is based on a 3D CNN autoencoder).

It is possible to notice that the proposed solution outperforms the other strategies at different bit rates. Indeed, PCG-AE employs a 3D convolutional network, but it does not exploit the correlation with side information. On the contrary, the ADAE strategy allows implementing a sort of predictive strategy using the previous LOD as a reference. Note also that larger $n_H$ values permit achieving a higher quality at higher resolutions since a longer hash array is overdimensioned in order to correct LOD-$(k+1)$ into LOD-$k$. At higher resolutions, the characterization of $\mathbf{b}_t$ becomes more complex and therefore, larger $n_H$ are required. It is also worth noting that bit rate is very close since the features $s_{t,i}$ are more correlated along $t$ allowing the contexts of the adaptive arithmetic coders to converge more quickly.

**Table 1**. Bijontegaard $\Delta$rate, $\Delta$PSNR D1, $\Delta R_p$ for ablation test.

| Model | w/out $d_F$ | | with $d_F$ | |
|---|---|---|---|---|
| | $\mathbf{\Delta P}$ (dB) | $\mathbf{\Delta R}$ (%) | $\mathbf{\Delta P}$ (dB) | $\mathbf{\Delta R}$ (%) |
| `longdress` | $-0.04$ | $-6.34$ | 0.15 | $-6.10$ |
| `house` | $-0.02$ | $-5.76$ | 0.1 | $-5.80$ |

It is also worth noticing that the compression gain is more evident for noisy point clouds (see the plot for the `house_without_roof` model). In this case, the high irregularity of points distribution prevents a correct reconstruction using simple CNN layers. By including a plane fitting metric in the training phase, the network lean to fit the points into more regular structures.

*Ablation study*. In order to test the effectiveness of the proposed loss functions, we run different training and coding operations changing the loss function. More precisely, we evaluated the Bijontegaard $\Delta$rate and $\Delta$PSNR D1 using $\mathcal{L}_{AE}$ and $\mathcal{L}_{AE}$ with $d_F = 1$ (to disable the effect of $d_F$) with respect to the rate-distortion performance obtained using a simple MSE loss function (disabling $d_P$ and $d_F$). Table 1 reports the obtained values for the `longdress` model. It is possible to notice that most of the improvement is brought by $d_P$.

## 4. CONCLUSIONS

The paper presented a new adversarial autoencoding strategy for the compression of voxelized point cloud geometries. In the training phase, a distributed source autoencoder is combined with a reconstructed-vs.-original discriminator processing the decoded voxel blocks. This latter module acts as a regularizer for the reconstruction routine improving the quality of the reconstructed data in terms of PSNR D1. Rate-distortion performances showed that the proposed solution performs better than other state-of-the-art point cloud coders.

Future research will be focused on different directions. On one side, there is a need for effective rate-distortion optimization strategies that tunes the amount of coded data depending on the characteristics of the input geometry and the available transmission/storage capacity. Moreover, further investigations are required in order to test the deployability of the current architecture to low complexity embedded systems for mobile applications.

## 5. REFERENCES


[1] MPEG 3DG and Requirements, "Call for proposals for point cloud compression v2 - doc. n16763," in *ISO/IEC JTC1/SC29/WG11 Coding of Moving Pictures and Audio Meeting Proceedings*, Apr. 2017, files: w16763_PCC_CfP.docx.

[2] K. Mammou, "PCC Test Model Category 2 v0, in ISO/IEC JTC1/SC29/ WG11 Doc. N17248, Macau, China," 2017.

[3] R. Mekuria, K. Blom, and P. Cesar, "Design, implementation, and evaluation of a point cloud codec for tele-immersive video," *IEEE Trans. on CSVT*, vol. 27, no. 4, pp. 828–842, April 2017.

[4] Y. Huang, J. Peng, C. C. J. Kuo, and M. Gopi, "A generic scheme for progressive point cloud coding," *IEEE Trans. Vis. Comput. Graphics*, vol. 14, no. 2, pp. 440–453, March 2008.

[5] D. Thanou, P. A. Chou, and P. Frossard, "Graph-Based Compression of Dynamic 3D Point Cloud Sequences," *IEEE Trans. Image Process.*, vol. 25, no. 4, pp. 1765–1778, April 2016.

[6] S. Milani, "Fast Point Cloud Compression Via Reversible Cellular Automata Block Transform," in *Proc. of ICIP 2017*, Sept. 2017, pp. 2050–2054.

[7] P. A. Chou and R. L. de Queiroz, "Gaussian process transforms," in *Proc. of IEEE ICIP 2016*, Sept 2016, pp. 1524–1528.

[8] R. L. de Queiroz and P. A. Chou, "Compression of 3D Point Clouds Using a Region-Adaptive Hierarchical Transform," *IEEE Trans. Image Process.*, vol. 25, no. 8, pp. 3947–3956, Aug 2016.

[9] E. Pavez, P. A. Chou, R. L. de Queiroz, and A. Ortega, "Dynamic Polygon Cloud Compression," *CoRR*, vol. abs/1610.00402, 2016.

[10] P. de Oliveira Rente, C. Brites, J. Ascenso, and F. Pereira, "Graph-based static 3d point clouds geometry coding," *IEEE Trans. Multimedia*, vol. 21, no. 2, pp. 284–299, 2019.

[11] H. Houshiar and A. Nüchter, "3d point cloud compression using conventional image compression for efficient data transmission," in *Proc. of ICAT 2015*, 2015, pp. 1–8.

[12] L. Li, Z. Li, S. Liu, and H. Li, "Occupancy-map-based rate distortion optimization for video-based point cloud compression," in *Proc. of IEEE ICIP 2019*, Sep. 2019, pp. 3167–3171.

[13] E. S. Jang, M. Preda, K. Mammou, A. M. Tourapis, J. Kim, D. B. Graziosi, S. Rhyu, and M. Budagavi, "Video-based point-cloud-compression standard in mpeg: From evidence collection to committee draft [standards in a nutshell]," *IEEE Signal Process. Mag.*, vol. 36, no. 3, pp. 118–123, 2019.

[14] D. C. Garcia and R. L. de Queiroz, "Context-Based Octree Coding For Point-Cloud Video," in *Proc. of ICIP 2017*, Sep 2017, pp. 1412–1416.

[15] S. Limuti, E. Polo, and S. Milani, "A Transform Coding Strategy for Voxelized Dynamic Point Clouds," in *Proc. of IEEE ICIP 2018*, Oct. 2018, pp. 2954–2958.

[16] E. Di Palma and I. Tabus, "Compression of point cloud geometry with random access," in *Proc. of EUVIP 2018*, 2018.

[17] F. Capraro and S. Milani, "Rendering-aware point cloud coding for mixed reality devices," in *Proc. of IEEE ICIP 2019*, Sep. 2019, pp. 3706–3710.

[18] K. Zhang, W. Zhu, and Y. Xu, "Hierarchical segmentation based point cloud attribute compression," *Proc. of IEEE ICASSP 2018*, pp. 3131–3135, 2018.

[19] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston, "Variational image compression with a scale hyperprior," in *Proc. of ICLR 2018*, 2018.

[20] I. Schiopu and A. Munteanu, "Deep-Learning-Based Lossless Image Coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 7, pp. 1829–1842, 2020.

[21] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell, "Full resolution image compression with recurrent neural networks," in *Proc. of IEEE CVPR 2017*, 2017, pp. 5435–5443.

[22] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimized image compression," in *Proc. of ICLR 2017*, 2017.

[23] D. Liu, Z. Chen, S. Liu, and F. Wu, "Deep Learning-Based Technology in Responses to the Joint Call for Proposals on Video Compression With Capability Beyond HEVC," *IEEE Trans. on CSVT*, vol. 30, no. 5, pp. 1267–1280, 2020.

[24] X. Yu, Y. Tian, F. Porikli, R. Hartley, H. Li, H. Heijnen, and V. Balntas, "Unsupervised Extraction of Local Image Descriptors via Relative Distance Ranking Loss," in *Proc. of IEEE/CVF ICCVW 2019*, Oct.27–28 2019, pp. 2893–2902.

[25] A. F. R. Guarda, N. M. M. Rodrigues, and F. Pereira, "Point Cloud Coding: Adopting a Deep Learning-based Approach," in *Proc. of PCS 2019*, 2019, pp. 1–5.

[26] Maurice Quach, Giuseppe Valenzise, and Frederic Dufaux, "Improved deep point cloud geometry compression," in *Proc. of IEEE MMSP 2020*, 2020, pp. 1–6.

[27] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *Proc. of IEEE ICRA 2011*, May 2011, pp. 1–4.

[28] S. Milani, "A syndrome-based autoencoder for point cloud geometry compression," in *Proc. of IEEE ICIP 2020*, 2020, pp. 2686–2690.

[29] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *ArXiv*, vol. abs/1505.04597, 2015.

[30] M. Zamorski, M. Zieba, P. Klukowski, R. Nowak, K. Kurach, W. Stokowiec, and T. Trzcinski, "Adversarial autoencoders for compact representations of 3d point clouds," *Computer Vision and Image Understanding*, vol. 193, pp. 102921, 2020.

[31] Y. Yang, C. Feng, Y. Shen, and D. Tian, "Foldingnet: Point cloud auto-encoder via deep grid deformation," in *Proc. of IEEE/CVF CVPR 2018*, 2018, pp. 206–215.

[32] S. S. Pradhan and K. Ramchandran, "Distributed Source Coding Using Syndromes (DISCUS): Design and Construction," in *Proc. of DCC 1999*, Snowbird, UT, USA, Mar. 1999.

[33] S. Milani and G. Calvagno, "Distributed video coding based on lossy syndromes generated in hybrid pixel/transform domain," *Signal Processing: Image Communication*, vol. 28, no. 6, pp. 553 – 568, 2013.

[34] S. Milani, J. Wang, and K. Ramchandran, "Achieving H.264-like compression efficiency with distributed video coding," in *Proc. of SPIE VCIP 2007*, San Jose, CA, USA, Jan. 2007.

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