Static and Dynamic 3D Point Cloud Compression by TSPLVQ

Amira Filali, Vincent Ricordel, Nicolas Normand

Static and Dynamic 3D Point Cloud Compression by TSPLVQ

Amira Filali, Vincent Ricordel, and Nicolas Normand

LS2N Laboratory, Nantes University, Nantes, France
{Amira.Filali,Vincent.Ricordel,Nicolas.Normand}@univ-nantes.fr

Abstract. We have already addressed the issue of static point clouds geometry compression (G-PCC) by using Tree-Structured Point-Lattice Vector Quantization (TSPLVQ) [6]. Dynamic 3D point clouds are characterized by up millions of moving 3D positions and color attributes. Efficient point cloud (PC) compression is then fundamental. Temporally successive PC frames are very close, it however remains a challenging problem for coding as the PCs have varying numbers of points without explicit correspondence information. This paper improves and extends the prior work, which provided a new hierarchical geometry representation based on adaptive TSPLVQ. Firstly, a more robust Rate-Distortion optimization process is introduced in order to perform efficient and accurate rate-aware splitting decisions when building and coding the tree structure. Secondly, we focus on the compression of the geometry of dynamic point clouds (G-DPCC) and, the model enables to exploit the temporal dependencies of the 3D content. Exactly, TSPLVQ is a top-down method and permits to represent the PC geometry by using a scalable tree, so when quantizing the dynamic geometry of a given PCs sequence, the successive trees are represented as a trunk common for the 3D sequence, and branches added for each frame. Next, the trunk is first coded, followed by the branches that are differentially coded. Experimental results demonstrate that our method is able to bring significant improvement in terms of the overall compression performance compared to the state-of-the-art MPEG standard.

Keywords: 3D dynamic point cloud · G-PCC · differential compression.

1 Introduction

Among various newly acquired content types, 3D PC appears to be one of the most efficient representation of immersive media content, establishing a convergence between real and virtual realities and enabling more sophisticated immersive experience applications.
The development of even more precise capturing devices and the increasing requirements to realistically represent and to vividly render the 3D scenes, inevitably not only induce thousands up to billions of points, but also cause high complexity in the scattered random spatial distribution, which brings great challenges to the storage consumption and transmission system. Thus, more

advanced compression techniques are in urgent demand to make PC useful in practice.

The Moving Picture Experts Group (MPEG) led the process of building an open standard for point cloud compression (PCC) [4]. The standard adresses two main classes of solutions dealing with PCC: V-PCC which takes advantage of the usage of well-known 2D video technologies by projecting the point characteristics onto 2D frames, and a second class called G-PCC, for geometry-based compression of static PCs. G-PCC and V-PCC are both appropriate for the context of this paper, due to the large potential of improvements of compression techniques to store and transmit this type of data. Both G-PCC and V-PCC are based on conventional models, such as octree decomposition [11], triangulated surface model and region-adaptive hierarchical transform [15], [16]. Other explorations related to the V-PCC are relied on incorporating state-of-the-art technologies from both 2D video coding and computer graphics. To compare the compression solutions, MPEG provided quality evaluation metrics for PCC leading to the selection of the point-to-point and point-to-plane as baseline metrics [22].

In the present paper, our work is mainly related to G-PCC. Many solutions for PC geometry compression have been explored using the octree structure to partition the whole PC into smaller voxel volumes in order to better structure and represent the 3D PC [21] [20]. A lossless intra encoder of voxelized PCs is introduced in [14] that views the PC geometry as an array of bi-level images using a dyadic decomposition instead of the popular octree decomposition.

Some dynamic PC compression methods are specially designed for immersive 3D human body compression. Thanou et al. [23] introduced a compression framework for a dynamic 3D PC sequence, where they designed a novel approach for motion estimation and compensation for geometry and color attribute. Similarly, Nguyen et al. [13] proposed graph wavelet filter banks to compress moving human body sequences. Mekuria et al. [12] introduced a generic compression method for real-time 3D tele-immersive video, which is suitable for mixed reality applications. In their works, the octree structure is adopted in intra frames, and a prediction method is performed for inter coding by splitting the octree voxel space into macroblocks. In [5], authors focused on voxel-to-image projection methodology, where the video blocks are readily tiled into an image for efficient color compression through a traditional video codec. Recently, Guarda et al. extended in [8] the deep-learning coding approach to point cloud coding using an autoencoder network design.

Inspired by the formulation of quantization in [18, 19], we proposed in [6] to significantly expand it by using new quantization tools, different rate-distortion optimizations and several practical adaptations to the case of G-PCC.

The present paper builds on the proposed work in [6] to introduce a more acurate rate computation based on the entropic cost estimation of the tree for the purpose of efficient 3D PC geometry coding. Firstly, we improve the TSPLVQ using two splitting schemes (2×2×2 or 3×3×3) of a cubic Voronoï cell, by adapting the distortion versus rate trade-off. Thus, the purpose of our method is to reduce more the amount of data of a 3D point set while preserving as much information

as possible by considering the distortion in the rendered 3D content from the decoded PC. It is therefore better for rate-distortion optimization during the TSPLVQ procedure. Secondly, we also extend our method to the compression of the geometry of dynamic point clouds (G-DPCC) where we apply a process of comparison between successive PC frames inside each temporal segment of the PCs sequence in order to exploit temporal redundancies.

2 Static Point Cloud Compression by TSPLVQ

Our prior work in [6] is at the crossroads of static G-PCC, vector quantization of 3D data and rate-distortion optimization driven compression. The TSPLVQ approach, based on the embedding of truncated cubic lattices, permits hierarchical description of the 3-D PC through an unbalanced tree structure where at each node a cube is associated. The tree growing structure is achieved by using an iterative process such as, at each loop the best choice has to be done, between the node (namely its associated cube) to split and according to two cube splitting schemes (2×2×2 versus 3×3×3) to better map the PC splitting. This choice is based on a rate-distortion criterion, the optimization is then performed locally, where the Lagrange multiplier λ controls the trade-off between rate and geometric distortion. The rate considered in [6], is either a constant cost of node splitting (8 bits if the splitting is 2×2×2, 27 bits if the splitting is 3×3×3), or a basic entropy estimation taking into account the entropy of the population in relation to node points.

2.1 Optimized estimation of the tree rate cost

Entropy coding is critical for source compression and exploit statistical redundancies. Theoretically, the entropy bound of the source symbol (e.g., splitting bitstream in our case) is closely related to its probability distribution, and accurate rate estimation plays a major role in rate-distortion optimization driven compression.
We propose to use a more accurate estimation of the entropic cost of the node splitting during the tree growing process, where exactly, for each node splitting are computed: the increase in rate calculated in terms of entropic encoding cost of the node splitting, and the decrease in geometric distortion.

At this level of process, our objective is to code the PC geometry stored in its 3D volumetric representation, this is referred as the *voxelization*. Considering this geometry, a voxel V in the 3D representation at (i, j, k) position, corresponds also to a node (n_j) in the tree structure, is set to 1, e.g., $V(i, j, k) = 1$, if it is occupied (contains one or more PC points) and split, and $V(i, j, k) = 0$ otherwise. Each splitting scheme partitions the 3D cube associated to a node n_j either into 8 or 27 embedded smaller cubes (so children nodes of n_j). For each branch connecting n_j to its child, one bit is used, let's define it as the splitting state. Splitting state is a bitstream associated with each voxel node. Each

voxel node is divided into several voxels (children) depending on the quantization scheme being considered (namely, 2×2×2 or 3×3×3) and every leaf node has in turn an associated voxel value (1 or 0). Note also that each node indicates by a position index where it lies inside its parent's splitted 3D cube (for instance when considering 2×2×2 splitting case, the position index are simply listed from 1 up to 8). If after a loop, a leaf node undergoes splitting (namely, its corresponding cube splitting), we update the splitting state of its parent node. The splitting state determines using 0s and 1s the children nodes that are split, e.g. a given splitting by 2×2×2, could have $splittingstate = 01001101$. So children nodes with position index 1, 4, 5, 7 are split and children nodes with position index 0, 2, 3, 6 are not split. At each level of quantization step, every splitting state has associated occurency probability which gives how likely the splitting state occurs in the tree. In other words, the occurency probability of a splitting state is the ratio between the number of time this splitting state occurs in the tree, and the number of splitted nodes. We can then approximate the actual rate of every splitting state by computing its entropy.
For instance, the rate $R(s_j)$, used to calculate λ score in equation (4) in [6], is equal to the entropy of the splitting states of the tree nodes at the $j-th$ loop in the TSPLVQ growing tree s_j.
The splitting states entropies of the tree nodes has to be updated dynamically after each loop of the TSPLVQ splitting process, by taking into account of the incrementing and decrementing of the splitting states counters.
In order to count and to store the probabilities of all the splitting states, we use a hash table. Advantage of this strategy is that splitting state of any length can be dynamically stored in the table to avoid storing all the possible combinatories of all the splitting states (2^8 for 2×2×2 splitting and 3^{27} for 3×3×3 splitting). Due to this, we are able to run a hybrid quantization method using 2×2×2 and 3×3×3 in competition with each other and locating probability occurency of any state becomes fast due to hashing.

2.2 Final Bit-stream and attribute Compression

In the tree, each occupied leaf node corresponds exactly to one representant point which 3D location is set to the average value of the initial cloud points contained within the cube associated to the leaf. The mean colour is used to represent the colour information of all the cloud points inside the leaf cell. To further compress the corresponding reproduction vectors position and colour at the leaves level, we propose in this work to perform range coding by using Lempel–Ziv–Markov chain algorithm (LZMA) [3] for a lossless compression.
In other context, we could consider the cubes centers, instead of the mean points coordinates, to represent the input PC position and colour using the optimized TSPLVQ. In this case, we do not need to encode any explicit geometry information at nodes level, only build the tree structure by using recursive splittings described arithmetically in the bit stream when scanning the tree at different levels. Thus to encode the PC geometry : either we proceed progressively by using the scalable descriptions obtainable at the tree nodes (for instance by scanning

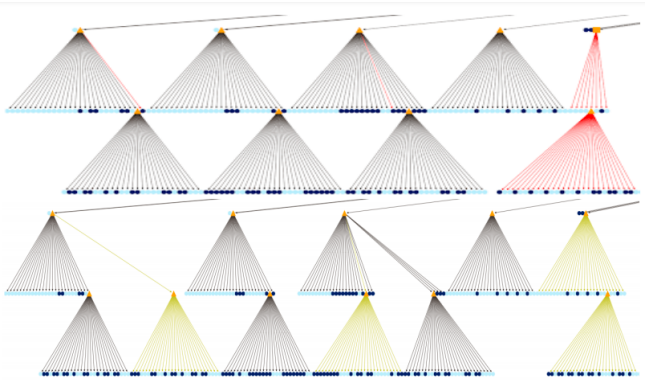

Fig. 1: Extract of TSPLVQ-based G-DPCC merging process for 2 successive frames. Common trunk is in black, the differential branches are in red and yellow.

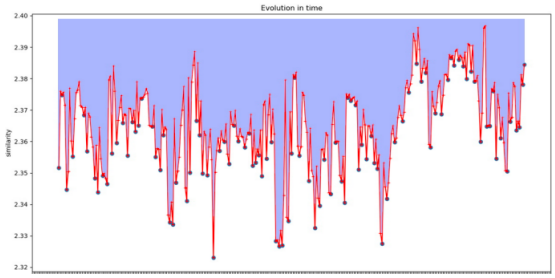

Fig. 2: Illustration of the inverse similarity curve for the 300 frames of "Loot" sequence: each stroke is a frame and minima local values are in blue.

its successive depth levels), the PC points in each leaf node are then represented by the corresponding center point, either we consider directly and only the final positions and colours associate to the tree leaves.

3 Geometry-based Dynamic Point Cloud Compression (G-DPCC) by TSPLVQ

We extend the previous work, destined to static PC compression, to dynamic PC compression where successive PC frames are close.

3.1 G-DPCC based on trunks and branches representations

A PC sequence of length T frames is defined as a T-tuple of 3D point clouds $P = (P_1, P_2, ..., P_T)$. Each element of this tuple is a PC sub-sequence, namely a 3D point set $P_t = p_i^t \mid i = 0, 1, 2, ..., n_t$ such as $\sum_{t=1}^{T} n_t = N$, where the frame $p_i(n)$ is represented by its geometric coordinates and features vector (e.g. colour). Our

proposed G-DPCC method consumes P as input, and produces S and D as outputs, defined respectively as M-tuple of trunks and N-tuple of corresponding branches. Exactly, the model firstly applies separately the optimized TSPLVQ method on each individual PC of P in order to represent its geometry by using a tree. Then the model aims at segment the P into temporal segments homogeneous in geometry (i.e. sub-sequences). The geometry homogeneity measure is based on the comparison between pairs of successive frames trees because (nearly) similar frames geometry content will produce close trees: the bigger their common tree part (namely common trunk), the higher their similarity in geometry. When quantizing the dynamic geometry of a given PCs sequence, the successive corresponding trees share a common trunk and they differentiate by their respective branches added to the trunk (as shown in Fig.1).
Our goal is then to get temporal segment where the frames all share a common trunk that is large enough, we proceed by using a greedy approach: we start by computing the common trunk between two first frames, next we compute the similarities (the similarity measure is detailed after in the 3.2) between the resulting common trunk and a third frame which is added to the segment if its similarity with the trunk is high enough, and the greedy process continues with the next frame tree. The process stops when the new frame tree similarity drops, then a new segment is created. When encoding the dynamic PC geometry of a given temporal segment, first its trunk is coded, next the branches of the successive frames trees are coded by using a differential approach (where only the differences between the trees are coded).

3.2 Watershed-based similarity measure for the segment design

We heuristically compute the common trunk by analyzing the structured trees corresponding to the PCs from the first to the last frame, by extracting the similarities between each pair of consecutive frames. This heuristic method requires to find the best size of the segment in order to better represent the corresponding PCs frames by a single common trunk. To do that, we propose to exploit the idea of watershed technique [9] which is able to detect valleys (i.e minimum local values) of a curve presenting in our case the similarity function F_{sim} applied to the successive PCs trees sequence.

$$F_{sim} = 1 - \frac{\sum_{i=1}^{D} w(n_i)}{\sum_{j=1}^{S} w(n_j)}$$

where,

$$w(n_i) = \frac{1}{\sum_{i=1}^{D} b_i^{d_i}}$$
$$w(n_j) = \frac{1}{\sum_{j=1}^{S} b_j^{d_j}}$$

Where w is a similarity weight calculated, for each node n_i in the D and n_j in the S, in function of the splitting scheme b_i and depth d_i in the tree structure.

Note that b_i equals to 2 if the chosen splitting scheme is 2×2×2 and equals to 3 otherwise with 3×3×3.
The idea consists of adapting the watershed algorithm to our objective. Because we aim at find the minima local values of the similarity curve in function of successive sequence frames instead of maxima local values (as shown in Fig. 2), so we invert the curve before using the algorithm. The positions of time-varying valleys determines the borders of each segment in the sequence. A smoothing filter is used in order to avoid an over-segmentation. For that purpose, a Savitzky-Golay (SavGol) filter [7] is applied on the similarity curve since it is the best for retaining the small scale features for the original curves.

Table 1: Comparison of symmetric PSNR and Bitrate metrics results between the optimized TSPLVQ based on Point-to-Point distortion and MPEG reference model.

Point Cloud	Number of points	MPEG G-PCC			Our approach			
		Number of points	PSNR	Bitrate (bpov)	Number of points	PSNR	Leaves cost (bpov)	Tree nodes cost (bpov)
Soldier	1089091	20065	52,79	0.029	73103	58.86	0.0002	0.1473
LongDress	857966	15685	52.78	0.031	48869	59.74	0.0002	0.0418
Loot	805285	14828	52.78	0.029	47616	60.08	0.0003	0.0444
Redandblack	757691	13832	25.40	0.032	45034	60.08	0.0003	0.3203

4 EXPERIMENTAL RESULTS

For Static PCC performance evaluation, we used our optimized TSPLVQ, relied on Point-to-Point geometry distortion, to encode 3D PCs. We compared our approach against the MPEG octree-based reference test model (lossy G-PCC model) [10]. All parameters in the G-PCC test model are kept unchanged for fair comparisons. We selected four static PCs from people object dataset: *Soldier*, *LongDress*, *Loot* and *LongDress* suggested by MPEG-3DG group [24] as a test dataset. The performance is then measured in terms of the point-to-plane symmetric PSNR metric given in [17].

A selection of results is presented in Table 1. The analysis of the PSNR metric of the four PCs, *Soldier*, *LongDress*, *Loot* and *Redandblack*, shows that our method when considering the leaves cost only, obviously outperforms the reference model on all the testing point clouds in term of PSNR and Bitrate. Considering these 4 points clouds, the reference model has an average bitrate of 0.03 *bpov* (bits per occupied voxel) and an average PSNR of 45.93 *dB* while our method has an average bitrate of 0.0002 *bpov* and an average PSNR of 59.90 *dB*. It is worth observing that the entropy cost of the built trees of all PCs, arithmetically encoded, is greater than the reference model. Moreover, we should take into consideration the colour information that also has to be transmitted, it is

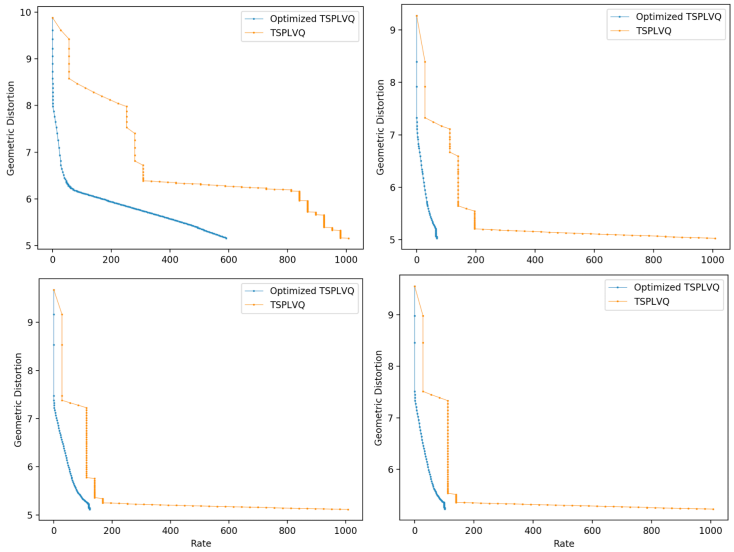

Fig. 3: RD curves showing the performance of the different steps of the optimized TSPLVQ. (first row: *Soldier* and *RedandBlack*; second row: *Loot* and *LongDress*). Distortion is computed using the Point-to-Point geometric distortion [6] and the rate is the entropic cost of the splitting scheme.

why we add this information to the leaves, when we consider the leaves cost in Table 1.

We compute RD curves for each sequence of the test PCs. We compared both rate-distortion driven TSPLVQ: the adaptive TSPLVQ [6] and the rate-adaptive TSPLVQ. The performance comparisons in term of rate and distortion between the 2 methods are reported in Fig. 3. The optimized method outperforms the previous TSPLVQ on all PCs providing a maximal decrease in distortion, and a minimal increase in rate.

In Fig. 4, we show examples on the four selected 3D objects. These particular examples show that our method produces more points than the reference model at lower bitrates. The MPEG G-PCC model does not show details in complex

Table 2: Comparison between the G-DPCC method and V-PCC reference model.

Methods	Attributes	Rendering	PC sequence: Loot		PC sequence: Soldier	
			Number of frames	average bpov	Number of frames	average bpov
V-PCC (V9.0)	colored	raw points	300 (30 fps)	0.93	300 (30 fps)	0.81
G-DPCC (+LZMA)	colored	raw points/ tree structure	300 (30 fps)	0.58	300 (30 fps)	0.80

Fig. 4: Rendering results for original PCs (first row: compressed PCs using MPEG lossy PCC; second row: (a), (b), (c), (d) and using our optimized TSPLVQ; third row: (e), (f) (g), (h)).

regions. It also leads to halo artifacts and the resulting PC requiring therefore a post-processing. For instance, for the *Soldier* PC, the MPEG reference model cannot show details in complex and smooth regions inducing a great loss of visual details (see Fig. 4 (a), (b), (c), (d)) while with our method using a lower bitrate for the leaves (0.0002 *bpov*), reasonable quality was achieved on all PCs, as shown in Fig. 4(e), (f), (g), (h). Hence, the optimized method can preserve the details and the global look of the original PC.
In addition, we could obtain very close rendering to the original PC when we produce more points thanks to the proposed multiscale approach with low bitrate. Our visual rendering seems qualitatively very close to the original rendered PCs compared to the reference test model. The benefits of the optimized method over the reference model in terms of both objective and subjective quality are easily

observable.

For dynamic PCC performance evaluation, we used our G-DPCC method to encode 3D point clouds. We compared our approach against the MPEG V-PCC (V.9.2) reference test model (lossy model) [2]. Here we show the compression performance of the compression of dynamic point cloud sequences, such as *Loot* ans *Soldier* sequences [1]. These PC sequences are composed of a 10s point cloud sequence of a human subject at 30 frames per second. Each point cloud frame has approximately 1 million points, with geometry positions and RGB color attributes. Table 2 shows that our method outperforms the V-PCC model on the *Loot* and *Soldier* PCs in term of Bitrate. The V-PCC reference model has an average bitrate of 0.93 *bpov* (bits per occupied voxel) for *Loot* PC sequence, while our method has an average bitrate of 0.58 *bpov*. The gain of our proposed G-DPCC over the V-PCC (V9.0) model is 37% and 1.2% for *Loot* and *Soldier* PC sequences respectively.

5 CONCLUSION

In this paper, we proposed a novel framework to compress static and dynamic point cloud geometry efficiently. Our methods take advantage of well-established principles that have been paramount to reach promising levels of static/dynamic PC compression performance.

References

1. 8i voxelized surface light field (8ivslf) dataset, available online at: https://mpeg-pcc.org/index.php/pcc-content-database
2. The tmc2 reference software, available online at: https://github.com/MPEGGroup/mpeg-pcc-tmc2
3. Akoguz, A., Bozkurt, S., Gozutok, A., Toprakci, G., E. Turan, M.B., Kent, S.: Comparison of open source compression algorithms on VHR remote sensing images for efficient storage hierarchy. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences (July 2016)
4. et al., S.S.: Emerging MPEG standards for point cloud compression. IEEE Journal on Emerging and Selected Topics in Circuits and Systems **9**(1), 133–148 (2019)
5. Dorea, C., de Queiroz, R.L.: Adaptive block partitioning of point clouds for video-based color compression. In: 2020 IEEE International Conference on Image Processing (ICIP). pp. 2746–2749 (2020)
6. Filali, A., Ricordel, V., Normand, N., Hamidouche, W.: Rate-distortion optimized tree-structured point-lattice vector quantization for compression of 3D point clouds geometry. International Conference on ImageProcessing (ICIP) (sept 2019)
7. Golay, M.J.E.: Smoothing of data by least squares procedures and by filtering. IEEE Transactions on Computers **C-21**(3), 299–301 (1972)
8. Guarda, A.F.R., Rodrigues, N.M.M., Pereira, F.: Point cloud coding: Adopting a deep learning-based approach. In: Picture Coding Symposium (PCS). pp. 1–5 (Ningbo, China 2019)

9. Hagyard, D., Razaz, M., Atkin, P.: Analysis of watershed algorithms for greyscale images. In: Proceedings of 3rd IEEE International Conference on Image Processing. vol. 3, pp. 41–44 (1996)
10. Mammou, K., Chou, P.A., Flynn, D., Krivokuća, M., Nakagami, O., Sugio, T.: G-PCC codec description v1. ISO/IEC JTC1/SC29/WG11 N18015 (October 2018)
11. Meagher, D.: Geometric modeling using octree encoding. Computer graphics and image processing **19**(2), 129–147 (1982)
12. Mekuria, R., Blom, K., Cesar, P.: Design, implementation, and evaluation of a point cloud codec for tele-immersive video. IEEE Transactions on Circuits and Systems for Video Technology **27**(4), 828–842 (April 2017)
13. Nguyen, H.Q., Chou, P.A., Chen, Y.: Compression of human body sequences using graph wavelet filter banks. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) pp. 6152–6156 (2014)
14. Peixoto, E.: Intra-frame compression of point cloud geometry using dyadic decomposition. IEEE Signal Processing Letters **27**, 246–250 (2020)
15. de Queiroz, R.L., Chou, P.A.: Compression of 3D point clouds using a region-adaptive hierarchical transform. IEEE Transactions on Image Processing **25**(8), 3947–3956 (Aug 2016)
16. de Queiroz, R.L., Chou, P.A.: Transform coding for point clouds using a gaussian process model. IEEE Transactions on Image Processing **26**(7), 3507–3517 (July 2017)
17. Ricard, J.: PCC CE 0.3 on new metrics. ISO/IEC JTC1/SC29/WG11 N18032 (October 2018)
18. Ricordel, V., Labit, C.: Vector quantization by packing of embedded truncated lattices. In: Proceedings., International Conference on Image Processing, Washington, DC, USA. vol. 3, pp. 292–295 vol.3 (1995). https://doi.org/10.1109/ICIP.1995.538545
19. Ricordel, V., Labit, C.: Tree-structured lattice vector quantization. European Signal Processing Conference (EUSIPCO), Italy (Sept 1996)
20. Rusu, R.B., Cousins, S.: 3D is here: Point cloud library (PCL). International Conference on Robotics and Automation, Shanghai (2011)
21. Schnabel, R., Klein, R.: Octree-based point-cloud compression. Eurographics Symposium on Point-Based Graphics pp. 111–120 (2006)
22. Schwarz, S., Cocher, G., Flynn, D., Budagavi, M.: Common test conditions for point cloud compression. in ISO/IECJTC1/SC29/WG11 MPEG output document N17766 (July 2018)
23. Thanou, D., Chou, P.A., Frossard, P.: Graph-based motion estimation and compensation for dynamic 3d point cloud compression. In: 2015 IEEE International Conference on Image Processing (ICIP). pp. 3235–3239 (2015). https://doi.org/10.1109/ICIP.2015.7351401
24. Tulvan, C., Mekuria, R., Li, Z.: Draft dataset for point cloud coding (PCC). ISO/IEC JTC1/SC29/WG11N16333 (June 2016)