Point Cloud Compression for Efficient Data Broadcasting: A Performance Comparison

Francesco Nardo, Davide Peressoni, Paolo Testolina, Marco Giordani, Andrea Zanella
Department of Information Engineering, University of Padova, Italy, email: {name.surname}@dei.unipd.it

***Abstract*—The worldwide commercialization of fifth generation (5G) wireless networks and the exciting possibilities offered by connected and autonomous vehicles (CAVs) are pushing toward the deployment of heterogeneous sensors for tracking dynamic objects in the automotive environment. Among them, Light Detection and Ranging (LiDAR) sensors are witnessing a surge in popularity as their application to vehicular networks seem particularly promising. LiDARs can indeed produce a three-dimensional (3D) mapping of the surrounding environment, which can be used for object detection, recognition, and topography. These data are encoded as a point cloud which, when transmitted, may pose significant challenges to the communication systems as it can easily congest the wireless channel. Along these lines, this paper investigates how to compress point clouds in a fast and efficient way. Both 2D- and a 3D-oriented approaches are considered, and the performance of the corresponding techniques is analyzed in terms of (de)compression time, efficiency, and quality of the decompressed frame compared to the original. We demonstrate that, thanks to the matrix form in which LiDAR frames are saved, compression methods that are typically applied for 2D images give equivalent results, if not better, than those specifically designed for 3D point clouds.**

***Index Terms*—LiDAR, point cloud, compression, autonomous driving, data broadcasting, performance comparison.**

I. Introduction

The Light Detection and Ranging (LiDAR) sensor is a remote scanner that determines the distance with an object by measuring the time between the emission of a light pulse and the reception of the back-scattered signal. LiDAR pulses are generated by an array of lasers that fire thousands of times per second at different vertical inclinations, and that continuously rotate to produce a three-dimensional (3D) omnidirectional representation of the surrounding environment in the form of a *point cloud*. Specifically, a LiDAR point cloud consists of a set of 3D data points in space corresponding to the projections of the laser beams on the surface of shapes or objects, and may also provide additional information including the laser intensity, scan angle, and reflectance properties of the surface.

In the last decades, LiDARs have been extensively applied to different research fields, including agriculture (e.g., for topographic analysis and prediction of soil properties), military (e.g., for ground surveillance, navigation, search and rescue) and architecture (e.g., for detecting subtle topographic features). More recently, LiDAR scanners have also been playing an increasingly important role for connected and autonomous vehicles (CAVs) to enhance detection and recognition of road entities, and enable a safer driving environment [1]. Compared to other types of sensors such as RADARs or color/thermal cameras [2], LiDARs are robust under almost all lighting and weather conditions, with or without glare and shadows, and are currently the most precise sensors to measure range [3]. On the other hand, LiDAR acquisitions may produce very large volumes of data that can be challenging to handle with standard Vehicle-to-Everything (V2X) technologies [4], [5]. One possible method to solve capacity issues is by leveraging the millimeter wave (mmWave) spectrum, as promoted by recent IEEE and 3GPP standardization activities for future vehicular networks [6]. At the same time, sensor data should be carefully selected as a function of the available channel bandwidth, so as to save (already limited) network resources for the most valuable transmissions [7]. However, this typically requires machine learning methods to be trained and validated for identifying the critical data, which may be difficult to do on board of vehicles [8]. In any case, data rates could be further reduced if the LiDAR point clouds were efficiently compressed before data are validated and broadcast [9].

In these regards, the most challenging aspect for data compression lies in the way the point cloud is represented. Given the 3D nature of LiDAR perceptions, geometric compression algorithms, based on Point Cloud Data (PCD), LAS-Comp/LASzip [10] and Octree [11] formats, are the most common in the literature. More recent techniques based on deep learning, e.g., OctSqueeze [12], have been developed to enhance compressibility in 3D scenes. Even though these methods preserve accuracy after compression, they require point-level processing of data, which may not be implemented in real time. As a result, the scientific community is considering applying bi-dimensional (2D) transformations to the point cloud, using graph algorithms, and then exploit image-oriented compression techniques, such as Lossless JPEG (J-LS) [13] or Portable Network Graphics (PNG) [14], as well as video-oriented and dictionary-based compression techniques, such as Motion JPEG 2000 (MJ2) [15] and Lempel–Ziv–Welch (LZW) [16] respectively, to reduce computational complexity. Despite these studies, however, there is no accepted standard for point cloud compression, thus stimulating further research.

Based on the above introduction, in this paper we provide a comparison between 2D and 3D compression methods for point clouds, shedding light on the most promising scheme(s) to guarantee accurate though efficient compression before data broadcasting. Compared to prior work, e.g., [17], our performance analysis is assessed not only in terms of average compression ratio (which generally indicates how accurately

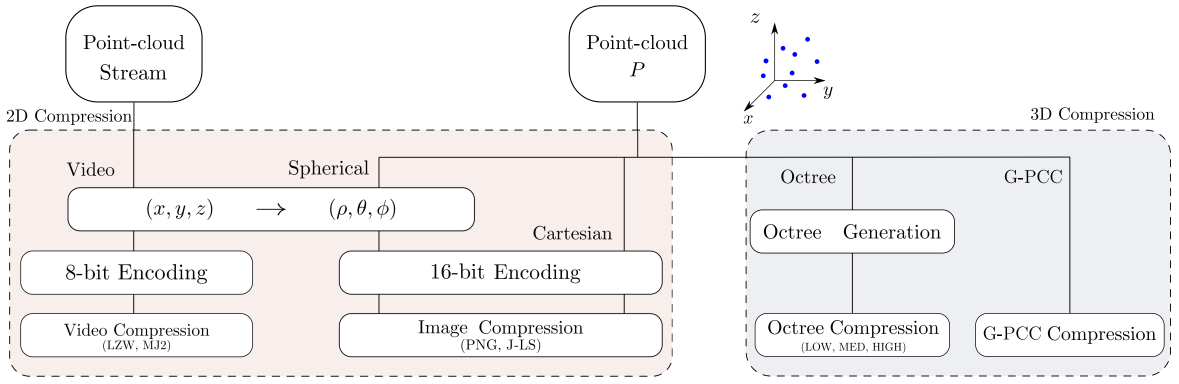

Figure 1: A schematic representation of the 3D and 2D compression methods reviewed in this paper.

data is compressed), but also in terms of Peak Signal to Noise Ratio (PSNR) (which measures the quality of the compressed point cloud, a critical requirement to ensure precise autonomous driving operations) and (de)compression time (to verify whether the point cloud can be (de)compressed in low latency, as is the case in safety-critical applications). Moreover, we study both 3D and image/video-based compression strategies, and investigate whether representing the point cloud with spherical coordinates, as natively supported in LiDAR data, would result in better compression than considering Cartesian coordinates. Our simulation results, validated on the public Veloview Sample Dataset, demonstrate that 2D compression methods are orders of magnitude more efficient and up to $20\times$ faster than the considered 3D methods, especially when spherical coordinates are adopted, while incurring limited accuracy degradation in the reconstructed point cloud.

The remainder of this paper is organized as follows. In Sec. II and Sec. III we describe some of the most common 2D and 3D methods, respectively, to compress LiDAR point clouds. In Sec. IV we present our simulation setup and numerical results. Finally, in Sec. V we summarize our main conclusions and suggestions for future work.

II. 3D Compression Methods

LiDARs emit light pulses and record the backscattered waveforms. In general, from each return pulse, we can estimate the Cartesian coordinates (x, y, z) and the angle of arrival of each point, the received signal intensity, the registered time, as well as other side information. In this work, we consider the data returned by the Velodyne sensors, i.e., a collection of User Datagram Protocol (UDP) packets encoded in a Packet Capture (PCAP) file, and try to compress the file size.

A. *3D Data Representation*

The most challenging aspect of collecting point clouds is related to their unordered and sparse structure, which makes classical storage methods inefficient. For this reason, new solutions have been specifically designed to represent point cloud data. Octrees [18], in particular, are an extension of binary trees in which each internal node has exactly eight children, and that can be used to partition 3D spaces. The root of the Octree is associated to the bounding box containing the whole point cloud. Then, the space volume is partitioned in eight parts, each assigned to a children of the root node. Each level of the space is subsequently split in eight parts. Thus, each child represents $1/2^3$ of the parent space. With this approach, each point collected by the LiDAR is represented by the leaf which contains it, so the encoding precision grows with the number of levels. A similar structure is the Voxel Grid (VG) that has been traditionally used in computer graphics to reduce both the input space dimensionality and the number of points in the raw point cloud.The VG sub-sampling technique is based on a grid of 3D *voxels*.[1] For each voxel, a centroid is chosen as the representative of all the points that lie on the corresponding partition of the space. [19]. Clearly, both the Octree and the VG representations introduce a quantization error, which depends on the granularity of the space partition.

B. *3D Data Compression*

Several 3D compression algorithms were developed, depending on how the point cloud is represented. For example, a compression algorithm exploiting the Octree data structure to perform predictive decoding based on local surface approximations was proposed in [11]. A deep neural network, also based on Octree data, was then introduced in [12]. Notably, a fast compression algorithm was developed in [20] considering spherical voxels, while the Moving Picture Expert Group (MPEG) has released specifications for the video-based (V-PCC) and the geometry-based (G-PCC) point cloud compression standards [21].

In this work, we analyze the efficiency and accurateness of G-PCC, as a possible standard for 3D point cloud compression, and of the Octree representation, as an efficient 3D storage method for point clouds, as illustrated in Fig. 1. For the Octree generation, the Point Cloud Library (PCL)[2] [22] was used. Each node of the Octree is represented by 8 bits, each stating whether the corresponding space partition is empty (0) or contains at least one point (1).Then, all non-zero bytes are saved in breadth-first order [19]. PCL offers 12 different

[1]A voxel is a discrete volumetric element used in the visualization and analysis of 3D data. It represents the equivalent of a 2D-image pixel but on a regular grid in the 3D space.

[2]The PCL is a standalone, large scale, open C++ library for point cloud processing and management. The PCL can be publicly accessed at https://pointclouds.org/documentation/tutorials/pcd_file_format.html.

▨ LOW (1 cm^2) 🟦 MEDIUM (5 mm^2) ⬜ HIGH (1 mm^2)

Profile	Resolution	Color	Encode Method
1	1 cm^3	No	Online
2	1 cm^3	Yes	Online
3	5 mm^3	No	Online
4	5 mm^3	Yes	Online
5	1 mm^3	No	Online
6	1 mm^3	Yes	Online
7	1 cm^3	No	Offline
8	1 cm^3	Yes	Offline
9	5 mm^3	No	Offline
10	5 mm^3	Yes	Offline
11	1 mm^3	No	Offline
12	1 mm^3	Yes	Offline

Table I: List of Octree compression profiles according to the PCL [22].

resolution profiles (corresponding to 12 different levels of compression), which can be grouped into 3 categories, i.e., HIGH, MEDIUM, and LOW, as reported in Table I.

III. 2D Compression Methods

In this section we discuss how 3D LiDAR point clouds can be transformed into 2D representations (Sec. III-A), and then compressed via 2D methods originally designed to compress images (Sec. III-B) and videos (Sec. III-C).

A. 3D-to-2D Data Representation

The 3D LiDAR data can be stored into a 2D image array, represented through Cartesian or spherical coordinates.

- *Cartesian representation.* According to prior work [17], the original Cartesian (x, y, z) point-cloud coordinates can be mapped into the 2D plane according to one of the following strategies:
  - *Single-channel Cartesian representation*: The points are saved in three single-channel (grayscale) images by assigning Cartesian coordinates to each image.
  - *Tri-channel Cartesian representation*: The points are saved in one Red-Green-Blue (RGB) colored image by assigning the x coordinate to the R channel, the y coordinate to the G channel, and the z coordinate to the B channel.
- *Spherical representation.* We propose to represent the point cloud through the (ρ, θ, ϕ) spherical coordinates (where ρ is the radial distance, θ is the polar/elevation angle and ϕ is the azimuth angle), computed as $\rho = \sqrt{x^2+y^2+z^2}$, $\theta = \arctan\left(z/(\sqrt{x^2+y^2})\right)$, and $\phi = \arctan(y/x)$. The data is then stored into a 2D image array.

Notice that converting 3D data into a 2D image requires the point-cloud coordinates to be converted from `float` to `unsigned integers`, thus introducing a quantization error. Considering a n-bit encoding, a floating point value u_f can be easily converted to `unsigned integer` u_i as

$$u_i = \left\lfloor \frac{u_f - \min(u_f)}{\max(u_f) - \min(u_f)} \cdot (2^n - 1) \right\rceil . \qquad (1)$$

In our trials, we observed that encoding with more than 16 bits would not bring any significant improvement in terms of accuracy, thus we set $n = 16$ in our simulations.

This image representation of LiDAR frames makes the application of existing 2D compression algorithms quite straightforward. Furthermore, the image encoding preserves the value continuity of the scene, i.e., neighboring pixels have similar values, an important property when applying image compression algorithms: each row in the matrix-form representation of the point cloud contains the points having the same elevation angle, i.e., acquired by the same laser, whereas the columns scan the azimuth space, according to the laser rotation. In the following, we present the 2D compression algorithms that we considered in this work, as illustrated in Fig. 1.

B. 2D (Image-Based) Data Compression

For image-based compression, we consider the well-known PNG and J-LS image formats. Specifically, PNG uses DEFLATE, a compression algorithm that combines LZW [16] with the Huffman coding [23]. Similarly to other dictionary coders, LZW employs a sliding window to scan the data: whenever a new sequence of bytes is observed, the corresponding dictionary entry is created and all the subsequent occurrences of the same sequence are substituted with the corresponding dictionary index. The dictionary is then compressed with the Huffman coding.

The J-LS algorithm [13], instead, predicts the value of each pixel in the image from the values of the neighboring pixels, thus leveraging the correlation among consecutive frames. This information is then modeled through a two-sided geometric distribution, and encoded using the Golomb coding, which is similar to the Huffman one.

C. 2D (Video-Based) Data Compression

Once the 3D LiDAR frames are converted into their 2D representation, video-based compression techniques (either inter- or intra-frame) can be applied. Specifically, we analyze the performance of an adaptation of LZW (DEFLATE) for videos, and the MJ2 algorithm. In both cases, 8 bit-encoding was applied, as typically considered in the most common video encoding standards.

We easily extended LZW [16] (specifically DEFLATE) in order to be applied inter-frame compression, i.e., taking into account the temporal correlation among consecutive frames to improve compression rates. Namely, for a given sequence of N LiDAR frames, three N-long vectors are generated, one per coordinate – either Cartesian (x, y, z) or spherical (ρ, θ, ϕ). DEFLATE is then applied to each coordinate vector separately.

MJ2 is another popular video coding scheme [15]. In this case, the video frames are generated as a sequence of images, according to the representation strategies described in Sec. III-A. Then, each frame is independently encoded using JPEG 2000. Because of the intra-frame encoding, the MJ2 is more resilient to propagation of errors over time, more scalable, and better suited to networked and point-to-point environments than DEFLATE. Also, it permits random access to individual frames.

IV. Performance Comparison

In this section we first describe our simulation scenario and performance metrics (Sec. IV-A), then we present our main performance results (Sec. IV-B).

A. Simulation Scenario and Parameters

The performance of compression algorithms has been compared on the Veloview Sample Dataset[3], that contains data from seven heterogeneous road environments acquired with a Velodyne VLP-16 and a Velodyne HDL-32 LiDAR, so as to consider different point cloud resolutions. In particular, the former sensor uses 16 laser beams with an angular resolution of 2 degrees and 0.1 degrees on the elevation and azimuth dimensions, respectively, while the latter configures up to 32 laser beams at around twice the resolution. Furthermore, the data was acquired using two rotation frequencies, i.e., 600 and 1200 rpm, thus further increasing the data diversity and the robustness of the results. The datasets were converted from the original PCAP format into the CSV or Binary PCD formats with the Matlab `velodyneFileReader` module, to be then used in our custom Python code for performance evaluation.

We compare the performance of the compression algorithms reviewed in Secs. II and III. For 3D methods, we consider the Octree compression levels (HIGH, MEDIUM, and LOW), and G-PCC with default parameters. For 2D methods, we compare image-based (PNG and J-LS, considering both tri-channel Cartesian and spherical representations) and video-based (LZW and MJ2, considering spherical representation only) solutions. The following metrics have been used to evaluate the compression algorithms.

a) Compression rate: Let $\bar{\mathcal{S}}$ be the size of the compressed point cloud, and $\mathcal{S}_{\text{raw}}$ be the size of the raw point cloud, which is the PCAP file from the LiDAR acquisition. The compression rate measures the reduction in size of the data representation produced by compression, and is given by

$$\text{Compression rate} = 1 - (\bar{\mathcal{S}}/\mathcal{S}_{\text{raw}}). \tag{2}$$

b) Bytes per Point (BPP): Let $|\bar{P}|$ be the total number of points contained in the compressed point cloud of size $\bar{\mathcal{S}}$. The BPP is defined as the number of bytes used to compress each point in the original point cloud, and is quantified as

$$\text{BPP} = \bar{\mathcal{S}}/|\bar{P}|. \tag{3}$$

c) Point-to-plane Peak Signal to Noise Ratio (PSNR): The PSNR is proportional to the quality of reconstructed point clouds/images/videos subject to compression, and is thus related to the accuracy of autonomous driving operations like object detection [24]. Let $\boldsymbol{p} \in P$ be one point in the original point cloud P, and $\boldsymbol{q} \in \hat{P}$ be its nearest neighbor in the reconstructed point cloud $\hat{P}$. The point-to-plane Mean Square Error (MSE) can be computed with respect to P as

$$\text{MSE}_{P \to \hat{P}} = \frac{1}{|P|} \sum_{\forall \boldsymbol{p} \in P} \left(\langle \boldsymbol{p} - \boldsymbol{q}, \boldsymbol{n}_{\boldsymbol{q}} \rangle \right)^2, \tag{4}$$

[3]The Veloview Sample Dataset can be publicly accessed at https://data.kitware.com/#collection/5b7f46f98d777f06857cb206.

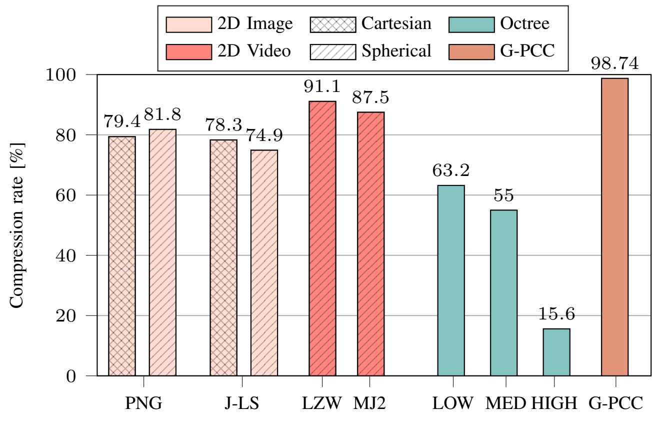

Figure 2: Compression rate for different 2D vs. 3D compression methods.

where $\boldsymbol{n}_{\boldsymbol{q}}$ is the surface tangent in $\boldsymbol{q} \in \hat{P}$, and $\langle \boldsymbol{p} - \boldsymbol{q}, \boldsymbol{n}_{\boldsymbol{q}} \rangle$ is the projection of vector $\boldsymbol{p} - \boldsymbol{q}$ on $\boldsymbol{n}_{\boldsymbol{q}}$. Accordingly, the point-to-plane PSNR with respect to P can be written as

$$\text{PSNR}_{P \to \hat{P}} = 10 \log_{10} \left(\frac{(\theta_P^*)^2}{\text{MSE}_{P \to \hat{P}}} \right) \tag{5}$$

where θ_P^* represents the peak value in the original point cloud P. Generally, θ_P^* is selected according to the nearest neighbor distances d_p for all points $\mathbf{p}$ in P, i.e., $\theta_P^* = \max_{\forall \boldsymbol{p} \in P} \{d_p\}$. Then, the PSNR between P and $\hat{P}$ is given by

$$\text{PSNR}_{P,\hat{P}} = \min \left\{ \text{PSNR}_{P \to \hat{P}},\ \text{PSNR}_{\hat{P} \to P} \right\}. \tag{6}$$

d) Computation time: It refers to the time required to compress/decompress the point cloud (from when the raw LiDAR output is produced in the form of a PCAP file, until the compressed point cloud is generated, or vice versa) using one of the techniques presented in the paper. This quantity has been measured on a machine executing an Intel Core i5-4210U processor at 1.70 GHz, running Linux 5.4.74-1, Python 3.8.6, g++ 10.2.0, using PCL 1.10 for Octree compression and G-PCC 13. All the trails have been run single threaded.

B. Numerical Results

Compression efficiency. In Fig. 2 we plot the compression rate for different compression methods. First, we observe that G-PCC achieves the best compression rate (98.74%), thus imposing as the standard for point cloud compression. Second, 2D compression and, in particular, PNG and J-LS, outperforms the Octree-based compression. In fact, unlike their 2D counterparts, Octree methods tend to overfit the data and cannot detect and appropriately remove redundant information hidden in the point cloud representations [9], resulting in a dramatic drop in the compression rate when increasing the resolution. On the contrary, PNG still guarantees a promising 80% compression rate, up to 25% better than Octree.

Third, Fig. 2 shows that representing the point cloud with spherical coordinates can result in better compression than using Cartesian coordinates, e.g., in case of PNG. In fact, while the former approach tries to store only radius and azimuth for each point in the LiDAR data (the elevation angle is indeed constant for each LiDAR laser beam), raw

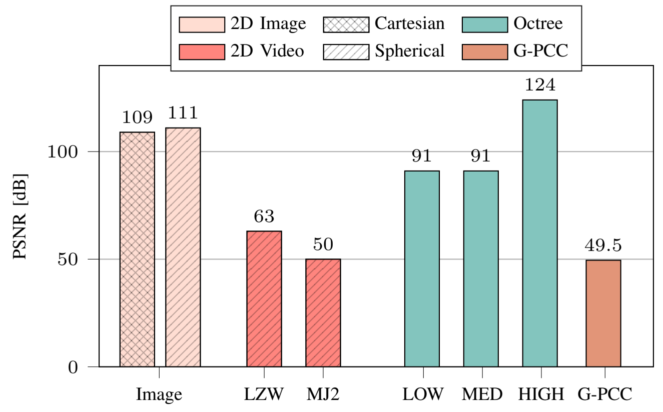

Figure 3: PSNR for different 2D vs. 3D compression methods. “Image” compression is obtained by averaging PNG and J-LS compression.

Cartesian files encode three geometric coordinates as a tri-channel image, thus using about 1/3 more BPPs than in the spherical methods. We also tried to convert the point cloud into spherical coordinates using the radius only, thus representing the LiDAR’s input as a single-channel image. While this approach permits to reduce the BPPs by 2/3 compared to Cartesian files, the final compression rate was unsatisfactory.

Third, Fig. 2 illustrates that video-based methods like LZW can compress efficiently by taking advantage of the temporal correlation between neighboring frames in the 2D point cloud representation, for example tracking the movement of cars: compared to PNG, LZW achieves a $+11\%$ improvement, just 8% less than G-PCC.

Compression accuracy. Compression accuracy is measured in terms of PSNR, as depicted in Fig. 3 (where the “Image” bars are obtained by averaging PNG and J-LS schemes, that gave similar results). It appears clear that Octree with HIGH profile exhibits the best performance ($+14\%$ against PNG, however in the face of a significant degradation in terms of compression rate), even though both LOW and MEDIUM profiles underperform image-based methods (-17%). In any case, the PSNR is guaranteed to be above 100 dB, thereby resulting in basically *lossless* compression; this ensures that the reconstructed point cloud after decompression can be considered the same as the original dataset. Notably, for image-based methods, both Cartesian and spherical representations give similar PSNR performance.

On the other hand, Fig. 3 shows that video-based compression, despite the high compression rate, suffers from very bad accuracy compared to both image- (up to -55%) and Octre-based (up to -60%) schemes. In fact, while static images are encoded with 16 bits, video frames are designed to operated with 8 bits, as illustrated in Sec. III-C. Even though updates to both LZW and MJ2 standards have been made to increase the bit-depth, commercially available implementations are still limited to 8 (or sometimes 12) bits per sample, which make the compression *lossy*.

Similarly, G-PCC exhibits a low PSNR, thus revealing the accuracy cost (up to 74 dB vs. Octree and 60 dB vs. 2D solutions) required to achieve its outstanding compression rate.

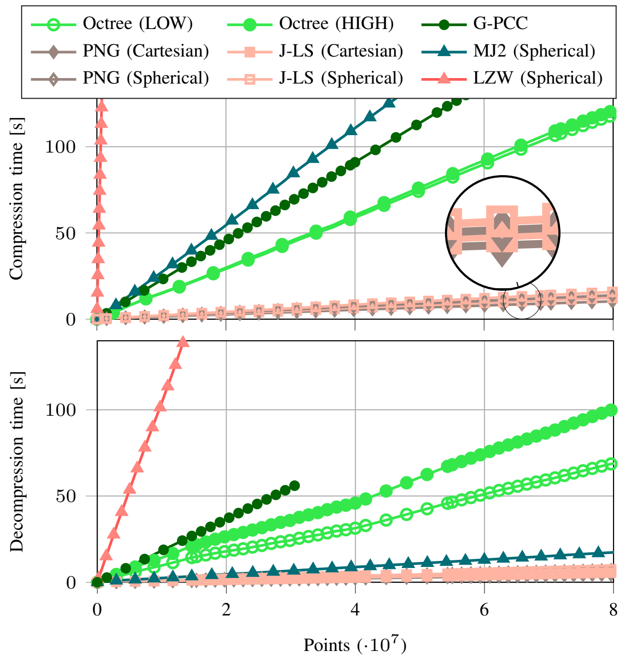

Figure 4: Compression (above) and decompression (below) times for different 2D vs. 3D compression methods.

(De)compression time. Timely compression and decompression is of utmost importance for communication systems to ensure that sensor data is broadcast in real time. From Fig. 4 (above), we observe that image-based methods achieve up to $10\times$ and $20\times$ faster compression than Octree and G-PCC. In particular, PNG works slightly better than J-LS, achieving an improvement of 20%. In both cases, the compression time grows linearly with the number of points in the point cloud, as expected. On average, Octree and G-PCC can compress around 670k and 440k point/s respectively, against the 5.5M points/s for PNG. In comparison, the HDL-32 sensor captures 695k points/s, thereby making image-based compressors the only methods capable of processing the data at the frame rate of the LiDAR, thus achieving real-time performance. Interestingly, video-based strategies (LZW and MJ2) are significantly slower than their competitors, which make them undesirable for most applications.

In terms of decompression, Fig. 4 (below) illustrates that image-based methods are still faster than the 3D ones. Notably, decompression takes less time than compression, a critical feature for autonomous driving since decompression is generally executed on-board of cars [9].

Compression guidelines for data broadcasting. To summarize our conclusions, Fig. 5 compares the compression performance of the investigated algorithms in terms of PNSR (to quantify the accuracy of the reconstructed point cloud) and BPP (to quantify the size of the compressed point cloud). As anticipated, image-based methods, in particular PNG, achieve the best trade-off. On one side, Octree-based solutions at HIGH profile could guarantee up to $+14\%$ better PSNR, while requiring in turn $3\times$ more BPPs for compression, making this solution ineffective for efficient data broadcasting. A LOW profile would exhibit worse PSNR and BPP performance, and

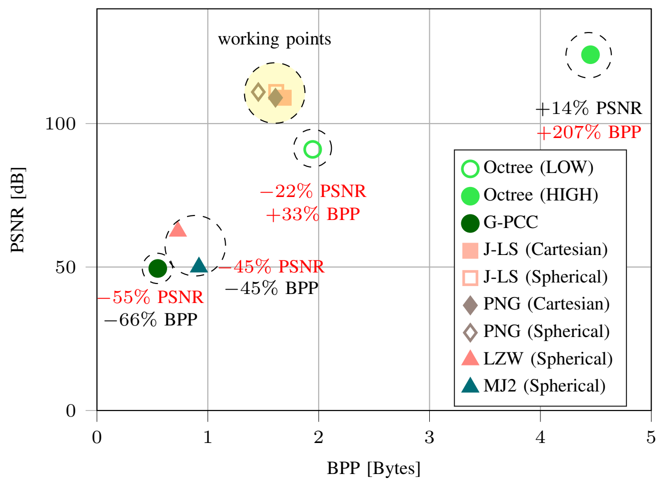

Figure 5: Compression quality (in terms of PSNR) vs. efficiency (in terms of BPP) trade-off for different 2D vs. 3D compression methods.

involve up to 10× slower compression. On the other side, video-based compression methods and G-PCC permit to represent the point cloud with −45% and −66% BPPs compared to PNG, even though degrading the PSNR by an impressive −45% and −55%, which makes these solutions lossy. Also, MJ2, LZW and G-PCC are not compatible with low latency for data dissemination, since compression and decompression may take tens/hundreds of seconds to complete. For both PNG and J-LS schemes, employing spherical coordinates before compression guarantees better compressibility than Cartesian-based methods, despite involving slower compression.

V. Conclusions and Future Work

In this paper we faced the challenge of compressing LiDAR data to facilitate efficient data broadcasting. To do so, we compared 3D compression methods (Octrees and G-PCC) specifically designed for point clouds, and 2D methods (PNG, J-LS, LWZ and MJ2) typically used to compress image and video frames. We showed that 2D algorithms, even though requiring the raw point cloud to be first transformed into its two-dimensional representation, can achieve a high compression rate and up to 20× faster compression than G-PCC, while guaranteeing a PNSR greater than 100 dB, thus supporting lossless compression.

In our future work we will investigate whether more advanced solutions, e.g., other settings of G-PCC [21] or methods based on artificial intelligence, may improve compression accuracy by operating directly on the raw 3D point cloud.

Acknowledgments

This work was partially supported by MIUR (Italian Ministry for Education and Research) under the initiative "Departments of Excellence" (Law 232/2016). Paolo Testolina's activities were also supported by Fondazione CaRiPaRo under the grants "Dottorati di Ricerca" 2019.

References

[1] N. Lu, N. Cheng, N. Zhang, X. Shen, and J. W. Mark, "Connected Vehicles: Solutions and Challenges," *IEEE Internet of Things Journal*, vol. 1, no. 4, pp. 289–299, Aug 2014.

[2] X. Yue, B. Wu, S. A. Seshia, K. Keutzer, and A. L. Sangiovanni-Vincentelli, "A LiDAR point cloud generator: from a virtual world to autonomous driving," in *ACM on International Conference on Multimedia Retrieval*, 2018.

[3] Y. Li and J. Ibanez-Guzman, "LiDAR for autonomous driving: The principles, challenges, and trends for automotive LiDAR and perception systems," *IEEE Signal Processing Magazine*, vol. 37, no. 4, pp. 50–61, Jul 2020.

[4] M. Giordani, A. Zanella, T. Higuchi, O. Altintas, and M. Zorzi, "On the Feasibility of Integrating mmWave and IEEE 802.11p for V2V Communications," in *IEEE Connected and Automated Vehicles Symposium (CAVS)*, 2018.

[5] M. Giordani, A. Zanella, and M. Zorzi, "LTE and millimeter waves for V2I communications: An end-to-end performance comparison," in *IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, 2019.

[6] T. Zugno, M. Drago, M. Giordani, M. Polese, and M. Zorzi, "Toward Standardization of Millimeter-Wave Vehicle-to-Vehicle Networks: Open Challenges and Performance Evaluation," *IEEE Communications Magazine*, vol. 58, no. 9, pp. 79–85, Sep. 2020.

[7] M. Giordani, T. Higuchi, A. Zanella, O. Altintas, and M. Zorzi, "A framework to assess value of information in future vehicular networks," in *1st ACM MobiHoc Workshop on Technologies, mOdels, and Protocols for Cooperative Connected Cars*, 2019.

[8] M. Giordani, A. Zanella, T. Higuchi, O. Altintas, and M. Zorzi, "Investigating Value of Information in Future Vehicular Communications," *IEEE 2nd Connected and Automated Vehicles Symposium (CAVS)*, 2019.

[9] A. Varischio, F. Mandruzzato, M. Bullo, M. Giordani, P. Testolina, and M. Zorzi, "Hybrid Point Cloud Semantic Compression for Automotive Sensors: A Performance Evaluation," *IEEE International Conference on Communications (ICC)*, 2021.

[10] M. Isenburg, "LASzip: lossless compression of LiDAR data," *Photogrammetric Engineering & Remote Sensing*, vol. 79, February 2013.

[11] R. Schnabel and R. Klein, "Octree-based Point-Cloud Compression," *Eurographics Symposium on Point-Based Graphics*, 2006.

[12] L. Huang, S. Wang, K. Wong, J. Liu, and R. Urtasun, "OctSqueeze: Octree-Structured Entropy Model for LiDAR Compression," in *Conference on Computer Vision and Pattern Recognition*, 2020.

[13] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS," *IEEE Transactions on Image Processing*, vol. 9, no. 8, pp. 1309–1324, Aug 2000.

[14] T. Boutell and T. Lane, "PNG (portable network graphics) specification," *Network Working Group*, 1997.

[15] Joint Photographic Experts Group (JPEG), "T.802 : Information technology - JPEG 2000 image coding system: Motion JPEG 2000," in *International Telecommunication Union (ITU)*, Jan 2005.

[16] T. A. Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 06, pp. 8–19, 1984.

[17] P. V. Beek, "Image-based compression of LiDAR sensor data," *Electronic Imaging*, vol. 2019, no. 15, pp. 43–1, 2019.

[18] H. Samet, "An overview of quadtrees, octrees, and related hierarchical data structures," in *Theoretical Foundations of Computer Graphics and CAD*, R. A. Earnshaw, Ed. Springer, 1988, pp. 51–68.

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

[20] J. Smith, G. Petrova, and S. Schaefer, "Encoding normal vectors using optimized spherical coordinates," *Computers Graphics*, vol. 36, no. 5, pp. 360–365, Aug. 2012.

[21] D. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, and A. Tabatabai, "An overview of ongoing point cloud compression standardization activities: video-based (V-PCC) and geometry-based (G-PCC)," *APSIPA Trans. on Signal and Information Proc.*, vol. 9, 2020.

[22] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation*, 2011.

[23] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.

[24] G. Li, Y. Yang, X. Qu, D. Cao, and K. Li, "A deep learning based image enhancement approach for autonomous driving at night," *Knowledge-Based Systems*, vol. 213, p. 106617, 2021.