Pushing Point Cloud Compression to the Edge

Ziyu Ying
Computer Science and Engineering
The Pennsylvania State University
State College, PA, USA
zjy5087@psu.edu

Shulin Zhao*
Computer Science and Engineering
The Pennsylvania State University
State College, PA, USA
suz53@psu.edu

Sandeepa Bhuyan
Computer Science and Engineering
The Pennsylvania State University
State College, PA, USA
sxb392@psu.edu

Cyan Subhra Mishra
Computer Science and Engineering
The Pennsylvania State University
State College, PA, USA
cyan@psu.edu

Mahmut T. Kandemir
Computer Science and Engineering
The Pennsylvania State University
State College, PA, USA
mtk2@psu.edu

Chita R. Das
Computer Science and Engineering
The Pennsylvania State University
State College, PA, USA
cxd12@psu.edu

***Abstract*—As Point Clouds (PCs) gain popularity in processing millions of data points for 3D rendering in many applications, efficient data compression becomes a critical issue. This is because compression is the primary bottleneck in minimizing the latency and energy consumption of existing PC pipelines. Data compression becomes even more critical as PC processing is pushed to edge devices with limited compute and power budgets. In this paper, we propose and evaluate two complementary schemes, intra-frame compression and inter-frame compression, to speed up the PC compression, without losing much quality or compression efficiency. Unlike existing techniques that use sequential algorithms, our first design, intra-frame compression, exploits parallelism for boosting the performance of both geometry and attribute compression. The proposed parallelism brings around 43.7× performance improvement and 96.6% energy savings at a cost of 1.01× larger compressed data size. To further improve the compression efficiency, our second scheme, inter-frame compression, considers the temporal similarity among the video frames and reuses the attribute data from the previous frame for the current frame. We implement our designs on an NVIDIA Jetson AGX Xavier edge GPU board. Experimental results with six videos show that the combined compression schemes provide 34.0× speedup compared to a state-of-the-art scheme, with minimal impact on quality and compression ratio.**

***Keywords*-point cloud compression; edge computing; video processing; energy-efficiency;**

I. Introduction

As the world is increasingly becoming virtual and moving closer towards automation, accurate 3D representation of real-life objects in the virtual domain, be it for life-like graphics or efficient autonomous driving, is becoming essential. Recently, *Point Cloud* (PC) consisting of millions of points, which capture the 3D geometry and attributes (e.g. RGB colors), has become an important modality for such realistic representations for applications like AR/VR, gaming, autonomous driving, etc. Moreover, with the recent pandemic, as telepresence is becoming a norm, people are virtually attending meetings, visiting arts, heritage sites and tourist places across the globe, and even living in a virtual universe. All these applications rely on high quality PC capturing, processing and displaying for a more realistic experience. Additionally, with the new generation mobile phones, capable of capturing PC and then streaming them into an AR/VR enabled head mounted displays (HMDs), capturing 3D PC now is becoming as common as capturing a photograph. With this trend, the PC business is expected to reach a 10 Billion dollar industry by 2024 [71].

Since capturing PC no longer requires sophisticated, commercial and expensive devices and people are equipped with mobile PC capturing devices (like iPhone 13 Pro [3]), the application providers are also pushing many of the PC processing tasks like compression or rendering to the edge, to avoid the use of expensive cloud resources, minimize the data transfer latency, and/or protect user's privacy. Considering the dense features, 3D geometry and the visual attributes captured in PC, especially for the media applications like telepresence and virtual visits, pre-processing [21], [44], [61], [84], compressing and storing [14], [16], [19], [47], [48], [74], post-processing and streaming [25], [40], [66], [76], [90] PC using a mobile device, while maintaining a reasonable quality of service (QoS), are fast becoming challenging tasks. Specifically, *PC compression* (*PCC*, or *PC encoding*) consists of both geometry (e.g., x, y, z coordinates in the 3D space) and attribute (e.g., RGB colors) compression. Our experiments show that PCC is the most expensive computation in a PC processing pipeline that takes $\approx 4 seconds$, especially when deployed in mobile/edge devices, and hence, is a major contributor to the performance, video quality, and transmission energy, for the entire PC pipeline.

However, it is challenging to design an optimal point cloud

[*]Work was done while at Penn State.

compression (PCC) pipeline which is *fast* (within or close to real-time), *accurate* (with good quality), and *efficient* (with high compression ratio). The state-of-the-art PCC pipeline typically utilizes tree structures like *Octree* [63] or *kd-tree* [62] for compression, and often, the tree construction becomes a bottleneck due to lack of parallelization. Moreover, the conventional PC typically stores the geometry, while a wide array of applications, especially the ones meant for content consumption, infotainment and gaming, need the attributes to be stored as well, hence making the compression even more complex. For example, TMC13 [56] and CWIPC [48] – two state-of-the-art (SOTA) PCC techniques – take 4.1s and 4.2s, respectively, to compress one PC frame on an edge platform, which are significantly higher than the real-time requirement ($\approx 100ms$ [19]), making them even more challenging to employ in emerging edge devices.

To address this, we study the SOTA compression pipelines and observe that the main reason behind their performance inefficiencies is their *sequential updates* to the global result *with each intermediate local runtime state* in a *point-by-point* fashion. Moreover, there has been little effort in parallelizing them on the state-of-the-art commercial systems, let alone on any edge/mobile devices. Prior works on PCC acceleration [19], [33] only consider the PC with geometry data and/or have limited parallelism, and thus, could neither leverage GPU nor benefit from other types of accelerators. In this context, this paper explores the following three **opportunities**: ① The points can be processed *in parallel* by using Morton codes [30] (which mathematically represent the geometry relationship among points) to identify the spatial-locality[1] within one frame for geometry compression. ② Further, this locality also exists in attributes (RGB pixels), i.e., spatial locality leads to attribute similarities, and hence opening opportunities for fast attribute compression. ③ And, finally, the locality extends beyond a single frame, i.e., the temporal locality, which can be leveraged by sorting the points in the Morton code order, creating further opportunities to improve the compression efficiency.

Motivated by these opportunities, we propose and evaluate a two-pronged compression approach, where the *intra-frame* approach leverages the opportunities described in ① and ②, and the *inter-frame* approach takes advantage of ③. The intra-frame approach speeds up the geometry and attribute compression by 37$\times$ and 49$\times$ respectively, while the inter-frame approach further improves the compression ratio by $\approx 1.75\times$ by reusing the matched blocks in reference frame.

To the best of our knowledge, this is the first work that targets to push the PCC to the edge by taking edge device-specific constraints into account and targeting four critical metrics – latency, energy, quality, and compression ratio. The major **contributions** of this work are the following:

- We identify the spatio-temporal redundancies for optimizing the PCC using a public PC dataset [18]. We also demonstrate that, such spatio-temporal localities can be precisely captured by Morton codes [30]. Specifically, we find that ① the points with similar Morton codes within one frame tend to have little variances in both geometry and attribute values (*spatial locality*), and ② the points with adjacent Morton codes (for instance, a cluster of geometrically close points) are likely to move in a certain direction, as a whole block, across frames (*temporal locality*).
- We propose two complementary designs to capture and utilize such spatio-temporal localities. First, we propose a Morton code-assisted intra-frame compression scheme, where *both* the geometry and attribute can be compressed in a *highly parallel fashion.* We believe this is the first work that applies the Morton code-based parallel octree construction algorithm [31] to speed up PC geometry compression. On the other hand, for attribute compression, we propose to sort the points in the Morton code order with the goal of capturing the attribute similarities. Also, to utilize temporal locality, we propose an inter-frame compression scheme which further increases the compression efficiency.
- We implement and evaluate our proposals on an edge device – NVIDIA Jetson AGX Xavier board [58]. Our extensive experimental results show that, compared to a state-of-the-art intra-frame PCC technique [56], our intra-frame proposal can accelerate the PCC by 43.7$\times$ and save 96.6% energy. While with our inter-frame compression design, the compression ratio can be further improved (increasing from 5.95 in intra-frame design to 10.43) with 35$\times$ speedup and 97.4% energy savings with respect to a state-of-the-art inter-frame PCC scheme [13]. Moreover, our proposal not only accelerates the PC encoding stage, but also can improve the performance of the decoding stage which involves inverse encoding operations (e.g., reduces decoding latency to $\approx 70ms$), thus enabling the end-to-end PC processing in near real-time (i.e., 10FPS).

[1] In this paper, we use "locality" and "frame similarity" alternatively.

II. Background and Related Work

A. Background

Point Cloud in Real Life: Point Cloud (PC) is a set of points which represent objects or shapes in a 3D space where each point/voxel (3D equivalent of a 2D pixel) contains its 3D location (x, y, z coordinates), as well as some attributes (e.g., colors, normal, etc.). Capturing PC representation of the real world typically requires millions of voxels, far more than the amount of pixels required for 2D images. While PCs containing only the 3D geometry data are commonly used in LiDAR-based 3D imaging for autonomous vehicles or robotics path planning, the lack of attributes nullify their usage for visual media consumption. Therefore, any PC application meant for visual media like

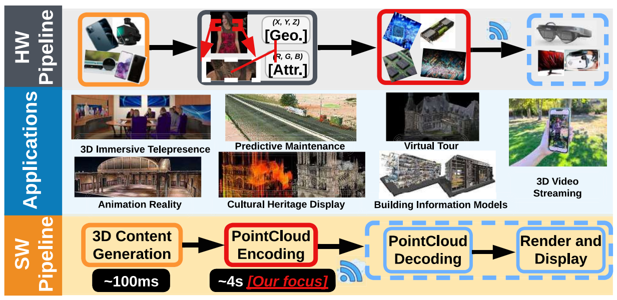


Figure 1: Example PC applications and processing pipelines.

immersive telepresence, telemedicine, video streaming etc., needs the attributes to be stored along with the 3D coordinates.

Since PC generation requires sophisticated instruments like LiDAR or 3D cameras, it was typically done on server-class computers with high compute and storage capabilities. However, with the advent of the modern mobile devices, capturing 3D image and PC on these tiny and battery-backed devices is becoming increasingly common. For example, the recent iPhone 12/13 Pro features LiDAR camera for PC recording, and similarly, Samsung Galaxy S20+/S20 Ultra contains ToF (Time of Flight) camera for the same. This makes PC-based media recording a common commodity, rather than a sophisticated pipedream. Moreover, applications like Record3D [44] enable seamless PC media streaming from phone to a wearable, encouraging a perpetually increasing PC content generation and consumption. However, the sheer volume of the data captured in these PC applications coupled with the limited compute and storage capabilities of these handheld devices pose a challenge in high quality PC media capture, storage and consumption [26]. To understand these challenges, we analyze an end-to-end PC pipeline.

End-to-end Pipeline: The PC video processing pipeline, as shown in Fig. 1, typically consists of 5 stages: 3D content generation, PC encoding, data transmission, PC decoding, render and display. In the *3D Content Generation* stage, the capturing device (e.g., the iPhone) uses LiDAR scanning or photogrammetry for the PC data acquisition. LiDAR maps spatial relationships and shapes by measuring the time taken by signals to bounce off objects and return to the scanner, while photogrammetry takes many photos from different angles to capture the target's geometry [9]. This process typically takes 10s of milliseconds [26]. Further, each point in the PC is associated with 3 coordinates (x, y, z) for the geometry and 3 colors (R, G, B) for the attribute. Thus, to represent one point, $4byte \times 3 + 1byte \times 3 = 15bytes$ are needed ($4bytes$ per coordinate and $1byte$ per color component). Thus, a typical PC frame containing 10^6 points [49] require $120M$ bits of data, which is impossible to transmit in real-time to the end-user's display, from both the latency and energy standpoints, considering a steady 30-60fps requirement. Therefore, the PC video frame is compressed in the *PC Encoding* stage, before being transmitted over the *network* to the end-user. The received frame is decoded in the *PC Decoding* stage and the decoded PC frame is forwarded to the *Render and Display* stage where it is finally rendered and displayed on the screen. Note that although the same method is followed and has been well established for streaming $2D$ or 360° videos, given that the PC data is much denser, compression becomes essential as well as the primary bottleneck, often taking several seconds to compress one PC frame [26](see Fig. 1ⓐ).

B. Related Work

1) Point Cloud Use-cases: Recently, PC is being widely used in various fields, such as AR/VR [46], [81], telepresence [43], [57], [86], virtual tourism [12], [50], teleoperation [83], telemedicine [51], video streaming [24], [37] and gaming [81], [87], etc. where both geometry and attributes are essential as the contents are consumed by people for infotainment purpose. Almost all of these applications can be categorized as interactive volumetric video streaming. On the other hand, for applications such as autonomous driving [1], [4], robotics [79], motion planning [34] or path planning [42], attributes like RGB info, at most times, are not necessary as the PC is used in the compute pipeline (by the machine) to extract features and make decisions.

Especially, **interactive volumetric video streaming** is starting to become mainstream, as edge devices (e.g., iPhones) facilitate recording and streaming the PC video which provide end-users with real-time 6-degrees of freedom (6-DoF) experiences. Streaming such PC videos in real-time involves capturing both the attributes and geometry data making it a challenging task even without user-object interaction. Towards this, Han et al. proposed the viewpoint-dependent PCC scheme (termed as "ViVo") which only sends the 3D tiles within user's field of view [24], thereby reducing the data volume. Such optimizations [37], [59], [68] are extremely important for applications like virtual tourism, video streaming etc. More complex optimizations are needed when *human-object* interactions and *human-object-sensors* interactions are involved (for applications like telepresence, telemedicine, virtual shopping and gaming) and need the help of PC data analytics to recognize/classify the interactable objects/scenes.

2) Point Cloud Analysis: To analyze objects/scenes in PCs, 3D convolutional neural networks (CNNs) have been widely used in techniques like 3D shape classification [66], [67], [89], [91], object detection [38], [45], [65], tracking [22], [78], or segmentation [10], [66], [67], [89]. While most prior works target accuracy, Mesorasi [20] improves the compute and memory efficiency of 3D CNNs using delayed-aggregation and software-hardware co-design, and PointAcc [40] proposes special mapping unit and memory management for optimizations. However, the huge data volume of the PC still remains a bottleneck in data movement and sharing, creating hurdles in high quality-low latency streaming/analysis. Therefore, there have been several works focusing on compressing the PC, as discussed next.

3) Point Cloud Compression (PCC): The prior PCC works can be classified as follows:

G-PCC utilizes special structures like octree or kd-tree to represent and compress the geometry [17], [23], [74]. For example, with octree-based PCC, considering a PC is contained in a $D \times D \times D$ cube, the cube is recursively divided into 8 $D/2 \times D/2 \times D/2$ sub-cubes until D=1. The occupied/non-empty voxels in level$_n$ can be indicated by the *occupy* bits of its "parent voxel" (voxel in level$_{n-1}$). Each branch node in the octree stores 8 occupy bits, indicating the occupancy of its children/sub-cubes. The attribute compression in the G-PCC depends on the geometry. As a result, the attribute and geometry are compressed separately. There are 3 methods for attribute compression in the G-PCC – *RAHT* [14], *Predicting Transform* [52], and *Lifting Transform* [52]. The main idea behind RAHT is to use the attribute values in a lower octree level to predict the values in the upper level. In contrast, Predicting Transform and Lifting Transform are based on the hierarchical nearest-neighbor interpolation [80]. Apart from these methods that compress a static PC, there also exist several attempts at optimizing the compression for dynamic PCs by exploring the "temporal redundancy" across the PC video frames. For example, a macro block (a $S \times S \times S$ cube) based motion estimation and compensation is proposed in [15], [16], [48], [73], to further improve the compression efficiency.

V-PCC[2] targets compressing PC videos. Specifically, given a PC video stream, V-PCC first performs 3D to 2D projection on each frame [29], [32], [39], [75], and then encodes these 2D projections via traditional 2D image codec. Both G-PCC and V-PCC are widely adopted in MPEG standard [53], and since our proposals begin with G-PCC, thus, is also compliant with the MPEG PCC standard.

NN-PCC[2] takes the raw PC as input, and feeds it into a pretrained 3D CNN, which outputs the compressed PC stream [28], [82]. Several recent efforts have been put into optimizing the 3D CNN to increase the compression ratio and/or decrease the number of parameters in the neural network model [27], [69]. However, based on the results reported in [40], even with a custom 3D-CNN accelerator, only 2.5× speedup could be achieved for 3D-CNN compared to edge GPU. Considering that NN-PCC can take thousands of seconds to compress one PC frame [88], such a huge gap between the long execution latency of NN-PCC and the real-time refresh requirement of vision applications is yet to close and prevents the deployment of NN-PCC on edge devices. Moreover, NN-PCC mainly focuses on compressing the geometry data and hence is not very useful for the PC with attributes [88].

[2]Although V-PCC and NN-PCC have high compression efficiencies, they are compute-intensive [41], [88], and consequently, are not the best option for mobile devices and are not considered in this work. Besides, most of the NN-PCC only focus on compressing geometry data [88], thus, is not applicable for this paper's target (i.e., mainly for vision applications where the attributes are essential).

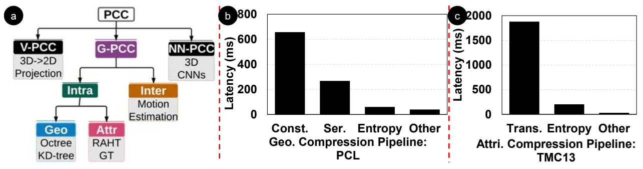


Figure 2: Prior PC compression technique categories and latency breakdown for prior techniques on compressing one PC frame from [55].

To the best of our knowledge, most of these works focusing on PCC with attributes target the compression ratio, and overlook the latency or energy consumption. However, as PC is moving to mobile, one cannot ignore the latency/energy constraints, thus demanding the need for mobile friendly PCC techniques which offer the best compression, latency and energy savings while preserving the video quality.

III. Motivation

A. Reasons for Inefficiency

To better understand the performance of the PCC pipeline, we characterize the "latency breakdown" of two state-of-the-art G-PCC techniques, i.e., PCL [72] and TMC13 [56], on a typical edge SoC platform (NVIDIA AGX Xavier) in Figs. 2ⓑ and ⓒ. Overall, the entire PCC pipeline takes around 3.5 seconds[3], which prevents one from employing such techniques in an edge device. Further, among the five stages in the pipeline, octree construction & serialization for geometry compression and RAHT for attribute compression are the two major bottlenecks which take 1s and 2s, respectively. Driven by these observations, we investigate the reasons behind such inefficiencies, and further explore the potential opportunities for speeding up the PC compression.

Before delving into the details of our approach which aims to close the performance gap between "seconds" in practical and "hundreds of milliseconds" in ideal settings, we first investigate the reasons behind the inefficiencies of the prior techniques. Towards this, we studied three state-of-the-art PCC pipelines – octree-based pipeline for intra-frame geometry compression (Sec. IV-A1), RAHT for intra-frame attribute compression (Sec. IV-C1), and macro block-based motion compensation pipeline for inter-frame compression

[3]We use PCL [72] and TMC13 [56] library for our profiling, where the geometry is compressed by the octree structure in PCL, and the attributes (RGB colors) are compressed through RAHT in TMC13.

(Sec. V) – and reached the following *conclusions*: the primary reason behind their performance inefficiencies is what can be termed as "under-parallelism", i.e., not being able to fully exploit parallelism during compression. Especially, many levels of dependencies (i.e, various regularities of locks) exist in their pipelines – e.g., the entire octree needs to acquire a "macro lock" before inserting a point and updating the tree (as shown in Fig. 5), during the intra-frame geometry compression; similarly, in the attribute compression shown in Fig. 6, the points at one layer in the octree have dependencies with those at other layers, thus their processing requires acquiring locks at a "layer granularity". Even with the optimizations in [33], where the octree construction stage can be performed in parallel, there are still several synchronization points, resulting to limited parallelism. To summarize, the performance inefficiencies in prior works can be primarily attributed to the lack of parallelism of these algorithms. Motivated by this observation, we next plan to improve the compression performance by exploiting various parallelism opportunities, which have been ignored, to the best of our knowledge, by the prior research but are essential in employing PCC in edge device settings.

B. What are the potential opportunities?

Increasing Geometry Compression Parallelism Using Morton Code: As mentioned earlier, the reason why the "sequential update" is necessary is that, during the intermediate stages, the *global* Octree (the final tree constructed at the last step) is unknown until the last point is inserted in the tree. To relax this constraint, if the PCs can be *sorted* based on a geometrical order, then the topographic structure of the global tree can be known at the beginning, thus fixing the tree structure and not requiring to be updated in a point-by-point fashion. As a result, these points can processed in *parallel*. In fact, there is a mathematical concept called *Morton Code* [30] (essentially, a space filling curve that maps a multidimensional data to one dimension while preserving the locality of the data points), which describes the geometrical location relationships between points, and thus can serve this purpose perfectly. There have been prior works like N-body application [5] which utilize the Morton code for parallel octree construction[4]; however, we believe ours is the first work that tries to apply such technique in the PCC pipeline.

Morton Code Can Also Assist Attribute Compression: As discussed above, Morton code naturally describes the geometrical relations among points; thus, intuitively, it makes sense to utilize the Morton code to improve the geometry compression. However, our goal is to go beyond just optimizing the geometry compression. Specifically, in the traditional 2D video compression domain, the video frames are usually rich in spatial locality (similar neighboring

[4]We do not claim the parallel octree construction as our contribution.

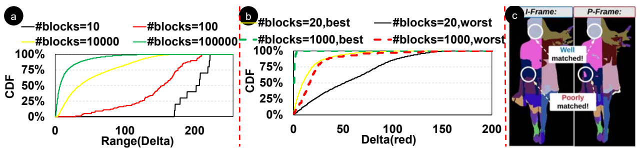


Figure 3: a) Spatial locality within one frame. b) Temporal locality among two frames. c) An example of macro blocks segmented using Morton codes in two frames.

pixels) within one frame, as well as temporal locality (similar pixel values in corresponding locations across consecutive frames) [36]. This observation motivates us to ask the question: *Do such similarities also exist in the PC streams? If so, can we leverage Morton code (containing location information) to capture such similarities as well?*

Spatial Locality in Attributes: Towards exploring the attribute similarity within one frame, we partition a frame (whose points are first sorted in Morton-code order) from the 8iVFB dataset [18] into 10, 10^2, 10^4 and 10^5 *macro blocks*, plot the CDF of the range for attribute delta ($Max_{red} - Min_{red}$) within one segment/macro block in Fig. 3ⓐ, and observe that:

- Overall, with more segments/macro blocks whose size is smaller (compared to a frame), more similarity exists in a block (delta is small). Specifically, compared to the black line (only 10 blocks), the attribute in yellow line (10^4 blocks, each of them is $1000\times$ smaller) exhibits a better similarity (i.e., left-shift towards the y-axis).
- When partitioning the macro blocks in an even more fine grain fashion, as shown in the green line with 10^5 segments, now the CDF curve is pushed towards left even further. This again indicates that, within a smaller macro block, the voxels have richer similarity with their neighbors.

Temporal Locality in Attributes: To study the temporal attribute locality across two frames, we plot the CDF of the attribute deltas among two segments in an I-frame and a P-frame in Fig. 3ⓑ, and a visual view of how these segments look like in Fig. 3ⓒ, and we observe that:

- Compared to the two solid lines (the frame is partitioned into 20 blocks), the two dotted lines (when partitioned into 1000 blocks) are closer to the y-axis, indicating that a finer segment can better capture the temporal-locality.
- Considering the dotted lines with 1000 segments partitioned from I- and P- Frames, the green line represents the smallest delta between two segments, which indicates the upper-bound/the scope of the attribute similarity, whereas the red line represents the largest delta/the least similarity among the segments. Further, the gap between the dotted red and green (1000 blocks) is smaller compared to that between the solid yellow and black lines (corresponding to a 20 block partitioning), thus indicating that a finer partition granularity can observe less variance in the temporal

locality opportunities (i.e., the smaller area, the better).

- Moreover, a vertical line can be drawn in this gap range, i.e., $x = \alpha$, and the macro blocks on the left side have "enough" temporal similarities and thus can be compressed with the I-frame (e.g., simply discard the deltas and represent/compress these blocks by the pointers to the matched I-blocks), whereas those on the right have to employ an extra intra-compression step to further compress the deltas. Note that α can be adjusted based on the application preference, e.g., shifting the $x = \alpha$ line to the right results in more macro blocks in the I-frame being directly reused for compressing the P-frame, i.e., higher compression ratio, with a cost of quality drop (more details in Sec. V).
- Fig. 3ⓒ illustrates an example with 20 segments in I-Frame and P-Frame. Due to the limited number of segments, one can observe that some highlighted blocks are not well matched. This again confirms that a finer segment can yield a better temporal locality, as also discussed above.

Takeaway: The Morton codes generated as an intermediate result during geometry compression not only improve the geometry compression by increasing pipeline parallelism, but also help to capture/identify the attribute similarities within a frame as well as across frames. Motivated by this observation, we next propose schemes that can utilize Morton codes for *both* geometry and attribute compression in point clouds.

IV. Intra-Frame Compression Design

As discussed in Sec. III-A, PCC takes several seconds to execute, which is significantly higher than the ideal/real-time demand (100ms). The main reason for this inefficiency is the "sequential updates" in the state-of-the-art octree-based algorithms [47] (illustrated in Fig. 2). Furthermore, we also observed in Sec. III-B that Morton codes can reveal opportunities for both geometry similarity (owing to the fact that the Morton code itself is the reflection of the geometrical relationship between points) and attribute similarity (the RGB attributes of two adjacent points are more likely to be similar). Unlike the prior octree-based works [56], [72] which mainly focus on the compression efficiency (i.e., attaining higher compression ratio and good quality simultaneously) with sequential updates and longer execution latency, in this work, we focus primarily on speeding up the PCC at the *edge* and achieving the real-time target mentioned above without losing much quality or compression ratio.

A. Intra-frame Compression

In this subsection, we first present the state-of-the-art intra-frame geometry and attribute compression techniques and discuss their inefficiencies. We then introduce our proposed intra-frame geometry and attribute compression schemes which are discussed in detail in Sec. IV-B and Sec. IV-C.

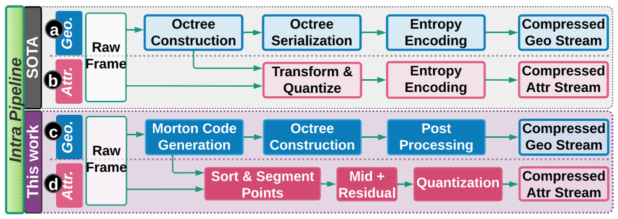


Figure 4: Intra-frame PCC pipelines.

1) Prior Intra-Frame Compression Inefficiencies: **State-of-the-Art Intra-Geometry Compression[5]:** As discussed in Fig. 2 and Sec. III, most of the existing G-PCC techniques [56], [72] are based on octree data structure. We illustrate the generic pipelines (for ① geometry compression, and for ② attribute compression) employed by the state-of-the-art intra-frame compression techniques in Fig. 4ⓐ and 4ⓑ. Specifically, the SOTA geometry compression pipeline includes five stages which can be summarized as follows:

- *Raw Frame (Input):* The input raw PC frame contains several (usually millions of) points, carrying both geometry and attribute information. Only the geometry data are forwarded to the upper geometry compression pipeline.
- *Octree Construction:* With the input geometry data, the octree construction algorithm is invoked to add the points and update the tree (e.g., the maximum depth required for inclusion of a point, occupancy information for nodes, etc.) in a point-by-point fashion. This point-by-point "update" makes this stage difficult to parallelize.
- *Octree Serialization:* After the octree has been constructed, the tree is traversed in a top-to-bottom manner, in order to extract the *occupy* bits for each node, and record them in a predefined order (e.g., via depth-first traversal), such that the decoder can recover the octree with these occupy bits as well as the serialization order. Note that this step is also time-consuming, as shown in Fig. 2. This is because that all the nodes in the tree are traversed sequentially.
- *Entropy Encoding:* To further compress the generated occupy bits vector, a typical encoding technique – Entropy Encoding [35], [60] – is employed.
- *Compressed Geometry Stream (Output):* The final compressed geometry output stream is ready to be stored in the memory or streamed over the network.

State-of-the-Art Intra-Frame Attribute Compression: As shown in Fig. 4ⓑ, to compress the attribute data, similar steps – *Raw Frame Input*, *Attribute Transform and Quantize*, *Entropy Encoding*, and *Compressed Attribute Output Stream* – are employed. Only the ***Transform and Quantize*** step differs from the geometry compression pipeline, which takes both the raw frame's attribute data as well as the constructed octree as its inputs. With these inputs, the *Transform* step

[5]We consider the octree-based technique [56], [72] and RAHT [14], [56] as SOTAs for geometry and attribute compression respectively.

performs linear transformations on the attribute data of each voxel pair (the voxel in level$_n$ and its siblings along x, y, and z dimensions) to obtain a low-pass component and a high-pass component. The high-pass component is quantized and entropy encoded, while the low-pass component proceeds to the next level (level$_{n-1}$) and serves as a prediction for the voxel's attribute in this upper level [14]. Note that, this step also needs to be performed sequentially across the tree layers.

Takeaway: The prior octree-based works for both geometry and attribute compression suffer from performance inefficiencies, mainly because the octree construction, serialization and attribute transformations involve sequential computations. To improve the performance, next we want to explore the hidden spatio-temporal locality opportunities (missed by the prior works), and speed up *both* the geometry and attribute compression from *both* the intra- and inter-frame perspectives.

2) Optimizing the Intra-Frame Compression:

Driven by the observations above, next we relax the "sequential update" approach that exists in the prior works, and employ the (intermediate) generated *Morton Codes* to reveal the "hidden parallelism" opportunities for compressing a PC frame (shown in Figs. 4c and 4d for the geometry and attribute compression, respectively).

Proposed Intra-Frame Geometry Compression: As can be seen from Fig. 4c, the modified components in our pipeline compared to the previously-proposed geometry compression approach (depicted in Fig. 4a) include the following:

- ***Morton Code Generation:*** Given the raw PC, instead of constructing the octree point-by-point, now the first step is to generate the Morton codes in one shot (note that this can be performed in parallel and only takes 0.5*ms*). This additional pre-processing step can draw an overall layout for all the points, which will further help to parallelize the octree construction.
- ***Octree Construction***[4]***:*** Using the Morton codes generated in the previous step, now the octree can be constructed in parallel by employing techniques similar to [31], [64]. Note that this step is slightly different from the one in the prior pipeline shown in Fig. 4a. Instead of updating and storing the occupy bits for each node during the process of adding points, now the outputs of this step are several arrays (Morton codes array, parent array, etc.), which reveal the geometrical relationship across the nodes.
- ***Post Processing:*** Using these relationship arrays, the final step is to post-process them to obtain the occupy bits for each node, and output the compressed geometry stream.

Proposed Intra-Frame Attribute Compression: As shown in Fig. 3 and discussed in Sec. III-B, apart from identifying the locality existing in geometry data, the Morton codes can also help us capture the attribute locality. Motivated by this, we further optimize the attribute compression pipeline for a given frame, as shown in Fig. 4d. Specifically, our proposed new pipeline includes the following three steps:

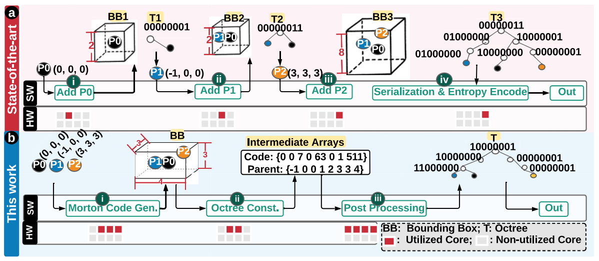


Figure 5: Intra-Frame geometry compression example.

- ***Sort and Segment Points:*** Unlike the prior works which utilize the octree to capture the spatial locality between the points when compressing the attributes, we use the Morton codes to cluster the points which are spatially close to each other. Specifically, with the Morton codes for all the points (i.e., the intermediate results from geometry compression without any additional overhead), we first sort these points in the Morton code order, and then segment these sorted points into several blocks which can help to gather the points with similar positions/coordinates into one segment.
- ***Mid + Residual:*** Within each segment, since the points are located in small regions, their attribute values tend to have similar numbers. Therefore, instead of recording the exact attribute values for all the points within a segment, we only need to find the "*median value*" of these attributes (as base) and then compute and compress the *residual values* (as deltas) for these points. Fortunately, these computations are light-weight, and can be performed in parallel.
- ***Quantization:*** Finally, these small residual values are quantized to further improve the compression ratio.

B. Intra-Frame Geometry Compression

In above section, we have discussed our overall proposals for both geometry and attribute compression pipelines. And, as will be shown later in Sec. VI, such optimizations are able to bring around 37× speedup w.r.t. the state-of-the-art techniques (1.55s latency in prior works vs. 42ms latency in ours). To better understand where the benefit comes from, next we go over a simple example, given in Fig. 5, and answer the following three critical questions: *i) how to increase parallelism for the bottleneck steps?*, *ii) how to integrate such optimizations into the entire geometry compression pipeline?*, and *iii) what are the resulting benefits and overheads?*

1) How to Increase Parallelism?: Fig. 5 shows an example of geometry compression. Specifically, there are three points in the this frame: P_0's coordinates are $[0,0,0]$, P_1's are $[-1,0,0]$, and P_2's are $[3,3,3]$. Consider the geometry compression pipeline in PCL [72], where the points are added one-by-one when constructing the octree. Initially, the bounding box is infinitely small (side length is 0, corresponding to no data); and the octree only has one *root* node, which is just a "virtual placeholder" containing no information. When inserting the first P_0 point into the bounding box and the octree, two actions are taken: (1) expanding the bounding box with a step-size of 2^n, where $n = 1, 2, 3, \cdots$, until P_0 is wrapped inside the bounding box. In this case, the side length of the bounding box cube becomes 2, and now P_0 is located inside the bounding box. And (2), inserting the new point into the current octree. In this case, P_0 is located in the $7th$ child of the *root* node, and the *root* node now stores the occupy information, which is 00000001 (the right-most 1 indicates a "child" in $7th$ leaf node). Similarly, P_1 is located inside the current bounding box and inserted into the octree as the $6th$ child of the *root*. Interestingly, in order to include P_2, the current bounding box has to expand its side length by 4×, i.e., enlarging from 2 to 8, and now the octree also contains more levels with all three points being in its leaf level. Obviously, both the bounding box and the octree are updated point-by-point, which forces the pipeline to be sequential.

In our proposal shown in the lower figure, instead of constructing the octree in a point-by-point fashion, we process all three points as one "batch" in the *Morton Code Generation* step in parallel, and output the final bounding box cuboid with side lengths 4×3×3 (x-axis: 3-(-1) =4, y-axis: 3-0 =3, and z-axis 3-0 =3). With the Morton code in place, next we invoke the parallel octree construction technique (for more details, please refer to [31], [64]) to construct the octree. Note that this step is also amenable to parallelism.

2) How to Apply to PCC?: As discussed in Sec. IV-A2, the *octree Construction* step returns several arrays containing the relationship among octree nodes; e.g., the *code array* contains the Morton codes for all the nodes, while the *parent array* contains the index of the current node's parent in the code array (e.g., in Fig. 5, -1 in the parent array means that the root node has no parent, whereas $parent[7] = 4$ means that for the $7th$ node (whose code is $code[7] = 511$), the index for its parent node in the code array is 4 (whose code is $code[4] = 63$)). Although such arrays already contain the necessary information to decode the geometry information for the PC, they are *not suitable for compression tasks*. For example, to store/transmit these two arrays, $4bytes{\times}16{=}64bytes$ are needed. However, even without any compression, we only need $4bytes{\times}3{\times}3{=}36bytes$ to represent these 3 points. Therefore, an extra post-processing step is needed to merge these two arrays in the "occupy bits" style. As shown in Algo. 1, to obtain the occupy bits for one branch node, we first calculate which branches its children should be on (e.g., $C[j]\%8$ in Line #5), and then merge all of its occupied branches via the "|" operation. This inexpensive post-processing step can be applied to all branch nodes in parallel, thus does not bring much overhead.

3) What are the Benefits and Drawbacks?: To summarize, compared to the prior schemes like PCL [72] and TMC13 [56], the most obvious benefit from our proposal is the potential performance improvement in terms of latency

Algorithm 1: Octree Occupy Bits Generation Algo.

```
Input  : C: Code Array; P: Parent Array; N: Number of Points
1 Occupy Bits Array: $O = \{\}$
2 $L = len(C) - N$
3 for $i$ in $L$ do
4     $p = P[i]$
5     $O[p] \mid= (C[j]\%8)$, $P[j] = p$
Output : O: Occupy Bits Array
```

and energy savings, due to embracing more parallelism. In fact, the CPU-based PCC pipeline in PCL [72] requires $O(N \times D)$ time complexity to process N points, with a D-layer tree. In comparison, given a GPU-based system with k parallel cores, our design requires only $O(\sum_{i=1}^{D} N_i/k)$ time (N_i is #nodes in layer$-i$). And, as we show later in Sec. VI, our results indicate 37× speedup for the geometry compression. Apart from compression, the de-compression stage can also be run in parallel after applying our proposal, and can be even faster than the compression stage due to its reduced complexity (note that, this also applies for the attribute compression proposals as discussed later in Sec. IV-C1 and Sec. V). Specifically, with our current "sub-optimal" implementation (e.g., the codes are not fully optimized), the de-compression stage (including both geometry and attribute de-compression) for Redandblack video [55] only takes $\approx 70ms$ per PC frame, which is less then the PC compression latency as we will discuss later in Sec. VI-C.

Such significant speedup comes with a reduction in quality. In the example shown in Fig. 5, the octree constructed based on the Morton codes is slightly different from the one generated by the sequential algorithm (which is lossless). In fact, in our octree, the P_0 node now contains geometry information of $[-0.43, 0, 0]$ (-0.43=-1+1/7×4), which is slightly different from the original $[0, 0, 0]$, whereas the other two points, P_1 and P_2, are exactly same as the original ones. Thus, as we discuss later in Sec. VI, our proposal drops the quality a little bit (PSNR ≈ 80dB in our design). This quality degradation comes as a result of the parallel algorithm, and we argue that the PSNR values resulting from our proposal are still very good for most video and AR/VR applications [6], [77].

Another comparison parameter is the *compression ratio*. Our proposal provides similar compressed size as TMC13 [56] (≈ 0.1× larger) when exploiting the entropy encoding. However, this entropy encoding consumes $\approx 100ms$, which halves our performance gains. Thus, in order to harvest most of the speedup benefits ($42ms$ vs $1.55s$), in our design, we discard the entropy encoding and still achieve reasonable compressed size, which is ≈ 0.5× larger than that of TMC13 [56].

C. Intra-Frame Attribute Compression

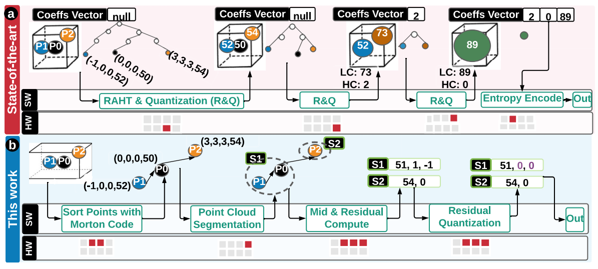


Figure 6: Intra-Frame attribute compression example.

The above discussion has explored the opportunity of utilizing Morton codes to speedup the geometry compression. Recall from Fig. 3ⓐ that, there also exists spatial locality for attribute compression, which can be identified with the help of the Morton code. Let us now consider the example in Fig. 6 with three points – P_0 with geometry data of $[0,0,0]$ and one attribute value of 50 (in this example, we set the attribute as a scalar for simplicity; normally, the attribute should be a vector, e.g., RGBs), P_1 with a $[-1,0,0]$ geometry and 52 as the attribute, and P_2 with a $[3,3,3]$ geometry and 54 as the attribute. Here, we assume that the octree for these points has already been established using the geometry pipeline, and we next go through our proposed attribute pipeline step by step and explain where the envisioned benefits will come from.

1) How to Speedup?: First, RAHT [14] takes the initial octree (which is deepest now) as input, and invokes *RAHT and Quantization* to perform the linear transformation on the leaves with their siblings along the x, y and z dimensions, and shrinks the tree layer-by-layer. Each transformation emits out one low-coefficient (LC) and one high-coefficient (HC) by the following equation:

$$\begin{bmatrix} LC \\ HC \end{bmatrix} = \frac{1}{\sqrt{w_1 + w_2}} \begin{bmatrix} \sqrt{w_1} & \sqrt{w_2} \\ -\sqrt{w_2} & \sqrt{w_1} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2, \end{bmatrix} \quad (1)$$

where w_1 and w_2 are the weights for the two leaves (#occupied voxels in this leaf), and a_1 and a_2 are the attributes. Now, the HC is quantized and entropy encoded, while the LC is further sent to the next *RAHT and Quantization* round and serves as the attribute of the large voxel/leaf in the upper layer. This procedure is repeated until we reach the root node. In this example, eventually the coeffs vector contains $[2,0,89]$, which can be further compressed by entropy encoding. This entire pipeline also requires sequential processing across the octree layers, which is obviously time-consuming when the number of points is large and the octree is deep. In fact, our profiling shows that RAHT takes around 2 seconds to process a typical frame with around 1M points, on a typical edge device.

Towards addressing this significant performance inefficiency, we investigate the insight from the above discussion in Sec. III-B, which indicates that the locality revealed by the Morton codes does not only exists among the geometry data, but it can also help with the attribute compression. Note that, the Morton codes have already been generated as intermediate results during the geometry compression, thus, can be (re)used to identify the spatio-locality for the attribute compression without any extra cost. Specifically, as shown in Fig. 6, in order to capture the spatial locality in attributes (e.g., points with similar Morton codes tend to have similar colors), our proposed pipeline first sorts the points using the Morton codes and then partition/group them into multiple segments. The points within one segment are geometrically close to each other, and hence their attributes are also likely to be similar. Thanks to this, for each segment, we just need to store one *medium* value as *Base* and several *residual* values as *Deltas* (which are mostly small, due to similarity). And finally, we quantize these deltas for achieving higher compression ratio. In this example, two vectors (as there are two segments) store the final data, including $Mid = 51, Delta = [0,0]$ for the first one segment, and $Mid = 54, Delta = [0]$ for the second.

2) What are the Pros and Cons?: As indicated in Fig. 6, compared to RAHT, our proposal reuses the intermediate Morton codes, which have been computed during the geometry compression, to precisely identify the points with similar attributes from a set of irregular points. This is expected to be much faster than RAHT, and in fact, our experimental results show $\approx 49\times$ speedup ($53ms$ vs $2.6s$).

However, as also shown in Fig. 6, the storage size after our compression is larger than RAHT, since each segment requires one vector storage to store its median/base and (quantized) delta values. Although the $48\times$ speedup brought by our proposal is promising in terms of performance gain, the observed $2\times$ compression inefficiency needs to be addressed.

3) How to Further Improve the Compression Efficiency for Attributes?: Towards further improving the compression efficiency, one could consider different options. Instead of throwing more compute power, we want to emphasize that, the discussion in this section only focuses on the attribute locality within one frame, which has ignored the potential localities among consecutive frames. In fact, if frame-2 does not vary much with respect to frame-1, intuitively, there would be temporal locality between the two frames. Thus, to further improve the attribute compression efficiency, in the next section, we investigate the inter-frame similarity opportunity.

V. Inter-Frame Attribute Compression Design

Motivated by the above discussion, in an attempt to further improve the compression efficiency from an inter-frame perspective, in this section we explore the "attribute similarity" that exists across consequent frames in a PC video, and explain the design details of our proposed inter-frame attribute compression scheme. Similar to the intra-frame proposals discussed in Sec. IV, we again use a simple

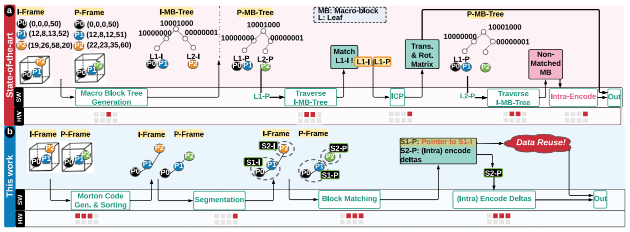


Figure 7: Inter-Frame attribute compression example.

example using a state-of-the-art inter-frame compression technique [13] and our proposal, and study the following important questions: i) *what* is the opportunity?, ii) how do we *capture* and *exploit* such opportunity?, and iii) what are the potential *benefits*?

A. *Inter-Frame Attribute Compression*

1) What is the Temporal Opportunity?: As we have shown earlier in Fig. 3b, two blocks (a set of points) which are located close to one another are likely to contain similar color pixels. In the example shown in Fig. 7, the first frame, I-Frame, contains three points – P_0 with geometry data $[0,0,0]$ and an attribute value 50, P_1 with $[12,8,13]$ for geometry and 52 for attribute, and P_2 with $[19,26,58]$ for geometry and 20 for attribute. Obviously, the two P_0 points in I-Frame and P-Frame are exactly the same, which could be completely reused during the compression of the P-Frame. Moreover, the two P_1 points are located closely (i.e., $[12,8,13]$ vs $[12,8,12]$), and contain very similar attribute values (52 vs 51). Thus, the P-Frame can also be further compressed by reusing the P_1 data in the previous I-Frame, without losing too much quality. On the other hand, the two P_2 points are relatively far away from each other and their attribute inputs are quite different, offering little reuse opportunity. To summarize, in this example, the first two points in I-Frame, P_0 and P_1, could be reused for compressing the P-Frame, thus reducing the compressed output size.

2) How to Capture the Temporal Opportunity?: To identify such similarity/reuse opportunities across frames (shown in Fig. 7), the macro-block based state-of-the-art approach [48] first needs to generate two macro block trees (where the minimum voxel dimension in this tree is of a predefined size) – one for I-Frame and the other for P-Frame. Next, for each leaf node/block in the P-MB-Tree, the entire I-MB-Tree needs to be traversed in a top-to-down fashion, and the exactly-matched leaf in the I-MB-Tree is found. In this case, the found leaf $L1$-I, which contains two points (P_0 and P_1), is a perfect match; however, no match can be found for the $L2$-P leaf. This process is repeated for $O(N)$ times, where N is the number of macro-blocks in the P-Frame. This processing can be quite time-consuming, and our profiling shows that it usually takes $\approx 5.9s$ to compress one predicted PC frame even when running on 4 CPU threads.

Instead, our proposal takes advantage of the Morton code generated in the geometry compression, which is a good indicator for attribute similarity (as discussed earlier in Sec. III-B). Specifically, our proposal consists of the following 4 steps:

PC sorting: As shown in Fig. 7, compared to the irregular raw PC, after sorting the PC via Morton-code, the adjacent points are more "regular" and geometrically closer, and thus share rich attribute similarities.

Segmentation: The next step is to partition the sorted PCs (i.e., I-frame and P-frame) into several blocks/segments (similar to the term "macro-blocks" in 2D image encoding).

Block/segment match (BM): for each block in the P-frame, we iterate through all the candidate blocks in the I-frame and calculate the difference between these $<I,P>$ block pairs. Finally, the candidate I-block which differs minimally with the P-block is picked as its "best-matched/reference" block. Specifically, given two blocks with K-points, we use 2-norm attribute distances (see Equ. 2) to measure their difference:

$$Diff(I_{block},P_{block}) = \sum_{i=1}^{K}(r_{iP}-r_{iI})^2+(g_{iP}-g_{iI})^2+(b_{iP}-b_{iI})^2 \quad (2)$$

where $P_{block}=\{(x_{iP},y_{iP},z_{iP},r_{iP},g_{iP},b_{iP})\}$, $I_{block}=\{(x_{iI},y_{iI},z_{iI},r_{iI},g_{iI},b_{iI})\}$, for $i=\{1,...,K\}$. Note that, the BM can be performed in parallel as there is no dependence across blocks.

Reuse: for blocks for which the reference blocks are similar enough (e.g., the 2-norm differences are less than the pre-defined thresholds), only the pointers to their reference blocks will be recorded (in our proposal, for each P-block, we set the number of candidate blocks as 100, thus, 6 bits are sufficient for encoding one P-block). On the other hand, if the "best matched I-block" is not as similar as the P-block (e.g., the 2-norm differences are larger than the threshold), simply approximating the P-block with its reference block will significantly degrade the quality; instead, we compute and store the deltas for such block pairs, and then invoke the *Base+Deltas* technique, as mentioned in Sec. IV-A2 for intra-frame compression, to further compress these deltas.

3) What are the Pros and Cons?: The proposed inter-frame attribute compression further improves the compression efficiency by skipping the redundant storage for the same/similar segments matched across frames, with extra latency overhead (but still much better than the state-of-the-art – 139ms vs 5.9s). In this example, for compressing the P-Frame, one pointer (for $S1$ which contains P_0 and P_1) and only one post-intra-encoded compressed delta (for P_2) are required for storage, instead of storing all three. However, the proposed inter-frame compression pipeline has additional steps (PC sorting and block matching), which collectively take about 139ms for a typical PC frame.

B. Combining Inter-frame and Intra-frame Compression

Simply put, our intra-frame compression proposal can significantly reduce the execution latency, while the inter-frame compression proposal can further improve the compression efficiency. We emphasize here that, these two proposals can work in an interleaved fashion (with a frame-level granularity) for a PC video stream. Specifically, in our design, the PC frames are encoded in an "IPP" fashion, where each I-frame is followed by two P-frames. Further, for the P-frame, as discussed above, the pre-defined threshold for determining the block matching (i.e., is the $<I,P>$ block pair good/similar match or bad/dissimilar match) can be tuned according to the application preference, e.g., for applications which favor good quality, more post-intra-encoded blocks (e.g., store and encode the deltas instead of simply reusing) are preferred. On the other hand, for applications that demand very small compressed data sizes to transfer through the network, the threshold condition can be relaxed to favor the *direct reuse* more. For example, in this paper, we pick two different thresholds to strike a balance between latency, quality and compression efficiency, as will be discussed in detail later in Sec. VI-C. We also investigate how different thresholds/preferences shape the behavior of our approach in a sensitivity study in Sec. VI-E.

VI. Experimental Results

In this section, we compare our proposed intra-frame and inter-frame designs against two different PCC techniques, by evaluating four metrics critical for the PC-based applications – execution latency, energy consumption, video quality, and compression ratio. Towards this, we first describe the configurations (Sec. VI-A) used for our analyses, e.g., experimental platform, dataset, and different designs (Sec. VI-B). We then compare those design schemes on our platform (Sec. VI-C). Finally, we provide detailed insights on how to tailor the PCC pipeline to cater to various application preferences (Sec. VI-E).

A. Methodology

1) Evaluation Platform: To evaluate and compare the proposed intra- and inter-compression designs with the state-of-the-art works, we use the NVIDIA Jetson AGX Xavier board [58], which is an edge development board, and is well-known to simulate the realistic edge development environment. Specifically, it is equipped with a 512-core Volta GPU, a 8-core ARMv8 64-bit CPU, and 32GB 256-Bit LPDDR4x Memory. In our implementation, we start the application from CPU (reading the PC data), and then offload the computations to GPU (octree construction, block matching, etc.), and the compute mode of Jetson AGX Xavier board is set to be 15W.

2) Point Cloud Dataset: We use two dynamic PC video datasets – the 8i Voxelized Full Bodies (8iVFB) [18] and the Microsoft Voxelized Upper Bodies (MVUB) [8] datasets in our evaluations. Specifically, we pick four videos from 8iVFB, and two videos from MVUB. The 8iVFB dataset contains the PC data of four persons, captured by 42 RGB cameras placed at different angles, while the MVUB dataset consists of five subjects captured by four frontal RGBD cameras. All these videos used are captured at 30fps, and voxelized into 1024×1024×1024 voxels (3D points), with each point containing three *float-pointing* coordinates and three *unsigned char* RGBs.

Table I: Six videos in 8iVFB [18] and MVUB [8] datasets used in this paper.

Video	**Redandblack**	**Longdress**	**Loot**	**Soldier**	**Andrew10**	**Phil10**
#Frames	300	300	300	300	318	245
#Points/Frame	727070	834315	793821	1075299	1298699	1486648

B. PCC Design Configurations

To demonstrate the effectiveness of our proposal, we evaluate the following five PCC designs:

- **TMC13** [56]: We use TMC13 (G-PCC codec from MPEG), as the *state-of-the-art* approach for *intra-frame compression*. Especially, by tuning the parameters (e.g., octree depth, compression algorithm, etc.), we utilize the octree-based method to compress the geometry data losslessly, while the attributes are compressed by the predictive RAHT lossily. We use this tool to compress every single PC frame, and measure the encoding latency, energy consumption, the compressed stream size, and finally measure the quality (PSNR) of the decoded frame by *pc_error_d* tool [85].
- **CWIPC** [13], [48]: CWIPC is a PCC library that supports the inter-frame compression (encoding the predicted frame via macro block (MB)-based motion estimation). We use CWIPC as the *state-of-the-art* approach for *inter-frame compression* and build it with multi-thread option. In our setup, one I-frame(intra-compressed frame) is followed by two P-frames(predicted frames), and the number of threads for MB matching is set to 4. Similarly, we have opted to use octree-based algorithm for geometry compression, and directly applied entropy encoding to the raw attributes.[6]
- **Intra-Only**: We apply our intra-frame compression method discussed in Sec.IV to each of the PC frames. Specifically, we choose to segment each PC frame into 30000 blocks[7], and use a 2-layer encoder (more specifically, we first encode the attributes via the proposed intra-frame encoder

[6]Based on our profiling, the provided attribute compression APIs (e.g., JPEG-Turbo-based) would degrade the quality of PC significantly; thus, we do not use such APIs/Libs in our evaluations.

[7]We decide the parameters for intra- and inter-compression by profiling several frames in the 8iVFB [18] dataset, to obtain a relatively balanced design point between compressed size and quality.

as described in Sec. IV, and then treat the obtained delta values as new attributes, and finally feed them again to the encoder to further increase the compression efficiency).

- **Intra-Inter-V1**: As mentioned in Sec. V, only part of the blocks/segments can be directly approximated by the reference block. Therefore, we add one more parameter to control the ratio of such blocks. Especially, for each P-block, we first calculate its 2-Norm distance to its *best matched block* in I-frame. We then compare this distance with a preset threshold (e.g., 300[7]) to determine if this segment can be approximated by *direct reuse*; otherwise, we mark it as an post-intra-encoded block. Also, the total number of blocks is 50000, and the search step is set to be the size of the current P-block (i.e., to find the best matched block for current P-block, each time, we traverse the search region in the reference frame by this step size).
- **Intra-Inter-V2**: For this configuration, we choose a larger threshold (1200) for the "direct-reuse decision making" such that the ratio of the *direct reuse* will increase with slight drops in quality. Other settings are not changed (they remain the same as in the Intra-Inter-V1 version).

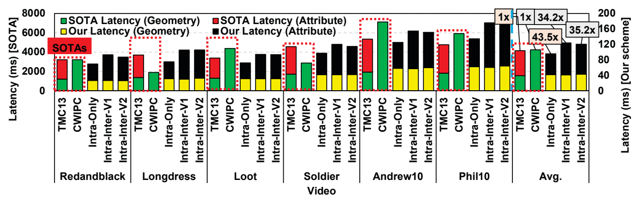
C. Results

We first compare the execution latency, energy consumption, quality (PSNR [peak signal-to-noise ratio]), and compression efficiency (compressed size) in Fig. 8, when using various designs explained in Sec. VI-B to compress the six PC videos from the 8iVFB and MVUB datasets (described in Table I). Then, we discuss and present the validity of our results on the smartphones.

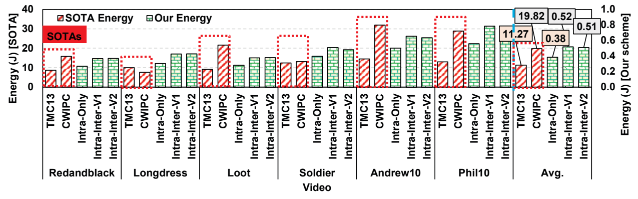
Execution Latency: We first compare two SOTA schemes (TMC13 [56] for intra-frame compression, and CWIPC [13] for inter-frame compression) with our three proposals (intra-only, (better) quality-oriented Intra-Inter-V1, and (better) compression-oriented Intra-Inter-V2), and present the collected execution latencies in Fig. 8a. For each video, these five designs are listed on the x-axis. The primary y-axis (left) shows the latency in *ms* for SOTAs, whereas the secondary y-axis (right) gives the latency in *ms* for our proposals.

From this plot, the following observations can be made:

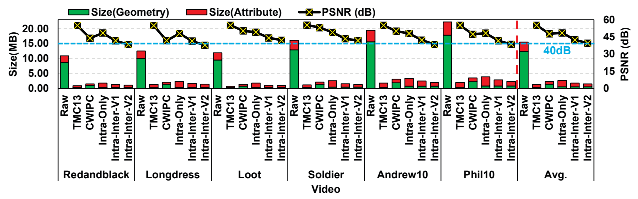
- *TMC13:* TMC13 takes around 4152*ms* to compress one PC frame, including 1552*ms* for geometry compression and 2600*ms* for attribute compression.
- *CWIPC:* CWIPC takes about 4229*ms* (mainly for geometry compression as the attributes are directly entropy-encoded without any other efforts, as mentioned in Sec. VI-B).
- *Our Intra-only:* The performance of above two techniques has two orders of gap with the ≈ 100*ms* real-time requirement [19]. On the other hand, our intra-only scheme takes only 95*ms* (42*ms* for geometry and 53*ms* for attribute compression), which is 43× faster w.r.t. TMC13. This is due to: 1). the octree can be constructed



(a) Latency breakdown for geometry and attribute compression



(b) Energy consumption



(c) Compressed size & PSNR

Figure 8: Results: (a) Latency breakdown. (b) Energy consumption. (c): Compression efficiency.

in parallel with the help of Morton codes, which speeds up the geometry compression by 37×; 2). by utilizing the spatial locality with Morton code for attribute compression, instead of performing transforms through octree layers, in our proposal, only simple subtractions are needed for computing deltas. Further, all the points are processed in parallel; 3). we discard the entropy encoding for further speedup.

- *Our Intra-Inter-V1 (Quality-oriented):* This design favors the quality over compression efficiency, and it only takes 124*ms* (41*ms* for geometry compression, and 83*ms* for attribute compression), contributing to around 34× speedup w.r.t. CWIPC. This speedup comes from: 1). instead of searching the matched macro block from the entire space (the I-MB-Tree traversal as described in Sec. V), given the points sorted in the Morton code order, the search space for block matching in our proposal is minimized (in the I-frame, now, we only need to search the neighboring regions for the current P-block); 2). for the matched macro block, instead of executing the complex iterative closest point (ICP) [7] algorithm for the translation and rotation matrix, we only need to record a pointer to the matched block in I-frame, without any extra computation overhead.
- *Our Intra-Inter-V2 (Compression efficiency-oriented):* Similarly, our Intra-Inter-V2 scheme (oriented towards high

compression ratio) takes 121ms (43ms for geometry and 78ms for attribute compression), which represents about 35$\times$ speedup w.r.t. CWIPC. There is not much performance difference between V2 and V1, because we have to run the block matching algorithm on all the blocks before making the "direct-reuse" decision, which dominates the entire pipeline in both the design variants.

Energy Consumption: Fig. 8b plots the energy consumptions (in J) for the SOTA works on the primary y-axis (left), and our proposals on the secondary y-axis (right). Clearly, TMC13 and CWIPC are two of the most energy-consuming schemes, which consume 11.3J and 19.8J, respectively, for one PC frame. This is mainly due to their long execution latencies. Further, as mentioned in Sec. VI-B, 4 CPU threads are invoked to perform the macro-block matching in CWIPC, this results in even higher CPU power. For example, the average CPU power for TMC13 is 1687mW whereas 3622mW for CWIPC. On the other hand, our Intra-Only scheme only consumes 0.38J per PC frame, which represents 96.6% energy saving w.r.t. TMC13, while our Intra-Inter-V1 and Intra-Inter-V2 only consume 0.52J and 0.5J energy, respectively, which translate to $\approx$97% energy savings w.r.t. CWIPC. Note that, although our schemes employ the GPU with an extra overhead (e.g., the GPU power is about 1065mW), the CPU power is reduced (e.g., around 1310mW, lower than that in TMC13 and CWIPC) since most of the computations are offloaded to GPU. Therefore, the overall energy savings brought by our schemes are similar to the corresponding execution latency reductions discussed above.

Compression Efficiency: To investigate how the compression efficiency changes with the above schemes, in Fig. 8c, we plot the compressed size (in megabytes) shown on the primary y-axis (left), and the PSNRs (in dB) for attributes[8] on the secondary y-axis (right), and observe that:

- *TMC13:* compresses the input frame size to be only 8% of the original while preserving the best video quality (PSNR is 55dB). This is mainly because TMC13 performs lossless geometry compression and almost-lossless attribute compression in our settings. Additionally, the transform and quantization in TMC13 can output (near-)zero coefficients, which significantly increases the compression ratio. Note however that, this comes at the cost of longer processing latency, as shown in Fig. 8a.
- *CWIPC:* Overall, when employing the CWIPC, the output frame size reduces to around 14% of the original input frame (including 63% of geometry and 37% of attribute data). The reason for such low compression ratio is because – ① for intra-attribute compression, only entropy encoder is applied; and ② even with the inter-frame compression, only few macro blocks are matched and inter-encoded, which limits the benefits from inter-frame compression. As for the quality, it drops the PSNR by 7.2dB when compared to TMC13, due to the macro block-based approximation for the inter-frame compression.
- *Our Intra-only:* this design emits out a compressed frame with $\approx$ 17% of the original data size (including 19% of geometry and 81% of attribute) and provides PSNR values up to 48.5dB (only 6.5dB drop compared to TMC13). Clearly, the geometry data has been compressed very well, as opposed to the attribute data. Recall from our intra-frame design discussion in Sec. IV that, the Morton codes can precisely describe the geometry relations among points (thus, ensuring a good geometry compression), but sometimes they may not work well for the attributes, especially when the spatial locality is not rich for some blocks/frames.
- *Our Intra-Inter-V1 (Quality-oriented):* To further reduce the frame size, this design exploits the temporal locality across PC frames. As a result, the data size becomes 5% less than our intra-only design (e.g., only 12% of the original size), while dropping the quality by 6.1dB due to the block-level approximations in the inter-frame compression.
- *Our Intra-Inter-V2 (Compression efficiency-oriented):* Similarly, this design further reduces the compressed data size by 2%, by adjusting the threshold for "direct-reuse decision making", as discussed in Sec. VI-B. At the same time, the quality further drops by 2.9dB. Still, we argue that, even with the Intra-Inter-V2 option (see one demo in Fig. 10a ⓘⓥ) which has the worst quality, the absolute PSNR is close to 40dB, which is sufficient for many of the video applications that do not require very high resolution [6]. For other high-demanding applications like AR-based surgery [11], our proposals may jeopardize the video quality, and consequently, we may need further software and/or hardware optimizations for improved user experience.

Correlation with the evaluations on smartphones: Based on our profiling, the power consumption of our proposal is $\approx 4W$, which is below the peak discharge power of modern smartphones (10W [2], [70]), meaning that our proposal will work fine on smartphones as well. To further prove this, we also change the compute mode of Jetson AGX Xavier board to 10W, and measure the execution latency for loot video [54]. We observe that the total execution latency when using 10W mode is 1.29x of that when using 15W mode (the mode for collecting the main results). Such similar performance demonstrates that our proposal is expected to work well for low-power edge devices like smartphones as well.

D. Architectural Insights:

In this section, we further investigate the energy efficiency characteristics of the proposed optimizations by dissecting the total energy consumption for the inter-frame attribute compression proposal (which is the most time- and energy-

[8]The geometry PSNR are excellent for all designs (e.g., $>$ 70dB), so we only compare the PSNR for attributes in the evaluations.

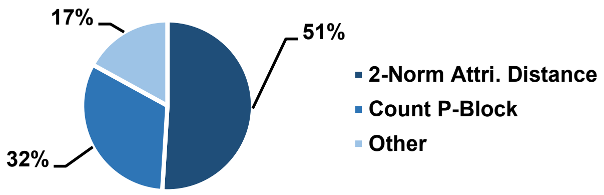
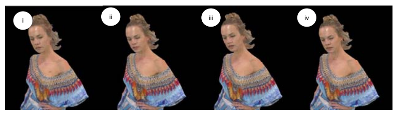
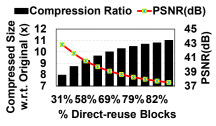


Figure 9: Energy consumption breakdown for inter-frame attribute compression for Loot video [54].



(a) Original vs. decoded PCs with our proposals.



(b) Sensitivity study.

Figure 10: (a) comparison between i: raw PC and our proposals (ii: intra, iii: intra-inter-V1, iv: intra-inter-V2). (b) PSNR v.s. compression ratio (i.e., input size / compressed size).

consuming one among the proposed geometry, intra- and inter-frame attribute compression techniques). We then analyze the bottleneck of our proposal and provide insights for potential architectural support to make PCC even more energy efficient. As shown in Fig. 9, the address generation stage for storing P-blocks' deltas/residuals consumes 32% of total energy, while the computation for the 2-norm attribute distance consumes 51% energy, which dominates the total energy consumption, and is, therefore, our target for next-step optimization. Specifically, to compute the 2-norm attribute distance, two kernel functions are invoked (*Diff_Squared* and *Squared_Sum*), which consume 35% and 16%, respectively of the total energy. Such high energy consumption of these two kernels can be attributed to two reasons. First, these are the most frequently invoked kernels during the block matching stage. Second, processing a typical PC with $1M$ points in a fully-parallelized fashion poses very high demands on the GPU resources (e.g., the number of available threads or the memory budget), which are quite limited, especially in edge devices. More interestingly, software-level optimizations for this step have been fully exploited (e.g., the kernel functions are invoked in a fully-parallelized manner), yet it still dominates the latency and energy. This motivates us to further look into architecture-level optimizations in future work, including 1) replacing GPU with ASIC to improve the power efficiency for *Diff_Squared* computation kernel; 2) customizing the accelerator (e.g., number of layers of the tree-structured adder) for the *Squared_Sum* kernel; and 3) minimizing data movements such as inter-SoC (e.g., between GPU and CPU) or intra-SoC (e.g., across L2/L3 caches in a GPU) memory copies.

E. Sensitivity Study

Our proposed intra-frame PCC utilizes the Morton code to capture the spatial locality, and significantly speeds up the compression (44×), with high compressed quality (48.5dB PSNR). Additionally, by exploiting the temporal locality across frames, our inter-frame compression further increases compression efficiency with the cost of longer processing latency and lower quality. To study how the inter-compressed frames/blocks would affect the compression efficiency (the compressed size w.r.t. the raw PC frame) and the quality (PSNR), we *reconfigured* the number of "direct-reuse" blocks by adjusting the threshold as discussed in Sec. VI-B. As shown in Fig. 10b, with fewer "direct-reuse" blocks (e.g., only 31% of the I-blocks are directly reused in the left-most bar), the PSNR drops slightly when compared to the intra-frame compression, while the compression ratio is also the worst. On the other side, by increasing the percentage of the "direct-reuse" blocks, the compression efficiency also increases, at the cost of a PSNR degradation (e.g., the PSNR reduces to 38dB with 83% "direct-reuse" blocks). Hence, to enhance the flexibility of our proposed design for trading off the compression efficiency with the quality, we can use the *percentage of "direct-reuse" blocks* as a *tunable design knob*, for which, users can choose the appropriate value based on their preferences (i.e., fewer "direct-reuse" blocks with higher PSNR vs. more "direct-reuse" blocks with higher compression efficiency).

VII. Concluding Remarks

PC processing has become the trend for many video applications spanning scientific computing, education, health-care and entertainment, and is recently being offloaded to the edge. PC compression is an essential component of PC processing (and a critical performance bottleneck), which affects video quality, user experience, and energy efficiency. Unfortunately, prior works mainly focused on compression ratio, but did not consider the performance and energy implications, particularly for edge devices. This paper exploits the data similarity opportunities in *both* geometry and attribute data from *both* intra-frame and inter-frame perspectives, and proposes two complementary designs for minimizing the compression latency and energy requirements for pushing the PC compression to the edge. And, more importantly, these proposals are compliant with the emerging MPEG PCC standards [53]. The experimental results with six PC videos show that our proposals provide 34× speedup (latency reduces from 4.2s to 121ms) and 96% improvement in energy efficiency, with only 13% compression ratio drop and a minimal degradation in video quality with respect to the state-of-the-art schemes. Note however that, even with our proposals, the execution latency per PC frame is still slightly beyond the real-time requirement (i.e., $\geq 100ms$). Towards this, in the future, we plan to explore GPU-specific optimizations such as compile-time instruction fusion for better parallelism, or provide additional architectural support for our proposal by investigating the hardware designs with respect to FPGA modules or customized ASICs, to optimize the bottleneck stage and make PCC on edge devices even faster/more efficient (e.g., $\approx 33ms$ for $30fps$ display refresh rate).

Acknowledgment

We thank the anonymous reviewers for their helpful feedback and suggestions towards improving the paper content. This research is supported in part by NSF grants #1629915, #1763681, #1931531, #2008398, #2116962, #2122155, #2028929 and #2211018.

References

[1] E. E. Aksoy, S. Baci, and S. Cavdar, "Salsanet: Fast road and vehicle segmentation in lidar point clouds for autonomous driving," in *2020 IEEE intelligent vehicles symposium (IV)*. IEEE, 2020, pp. 926–932.

[2] Andrei Frumusanu, "The Snapdragon 888 vs The Exynos 2100: Cortex-X1 & 5nm, Who Does It Better?" "https://bit.ly/3OF66Tw", 2021.

[3] Apple Inc., "https://www.apple.com/iphone-13-pro/", 2021.

[4] S. Ayukawa, N. Tokudome, S. Enokida, and T. Nishida, "Time-series lidar data superimposition for autonomous driving," *Proc. of ICT-ROBOT, ThBT3*, vol. 3, 2016.

[5] J. Bédorf, E. Gaburov, and S. P. Zwart, "A sparse octree gravitational n-body code that runs entirely on the GPU processor," *Journal of Computational Physics*, pp. 2825–2839, 2012.

[6] M. Beg, Y. C. Chang, and T. F. Tang, "Performance evaluation of error resilient tools for mpeg-4 video transmission over a mobile channel," in *2002 IEEE International Conference on Personal Wireless Communications*, 2002, pp. 285–289.

[7] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Sensor fusion IV: control paradigms and data structures*, 1992, pp. 586–606.

[8] Charles Loop, Qin Cai, Sergio Orts Escolano, and Philip A Chou, "JPEG Pleno Database: Microsoft Voxelized Upper Bodies - A Voxelized Point Cloud Dataset," "http://plenodb.jpeg.org/pc/microsoft".

[9] Charles Thomson, "Reality capture 101: point clouds, photogrammetry and LiDAR," "https://bit.ly/3xfqvqy", 2019.

[10] C. Choy, J. Gwak, and S. Savarese, "4d spatio-temporal convnets: Minkowski convolutional neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[11] A. Corning, "AR/VR in the OR – Surgical Applications of Augmented and Virtual Reality," "shorturl.at/fqwAJ", 2021.

[12] T. Cosso, I. Ferrando, and A. Orlando, "High-precision laser scanning for cave tourism 3d reconstruction of the pollera cave, italy," *GIM INTERNATIONAL-THE WORLDWIDE MAGAZINE FOR GEOMATICS*, vol. 29, no. 3, pp. 23–25, 2015.

[13] CWI-DIS Group, "cwipc-CWI Point Clouds software suite," "https://github.com/cwi-dis/cwipc", 2021.

[14] R. L. de Queiroz and P. A. Chou, "Compression of 3d point clouds using a region-adaptive hierarchical transform," *IEEE Transactions on Image Processing*, pp. 3947–3956, 2016.

[15] R. L. de Queiroz and P. A. Chou, "Motion-compensated compression of dynamic voxelized point clouds," *IEEE Transactions on Image Processing*, vol. 26, no. 8, pp. 3886–3895, 2017.

[16] C. Dorea and R. L. de Queiroz, "Block-based motion estimation speedup for dynamic voxelized point clouds," in *2018 25th IEEE International Conference on Image Processing (ICIP)*, 2018, pp. 2964–2968.

[17] J. Elseberg, D. Borrmann, and A. Nüchter, "One billion points in the cloud – an octree for efficient processing of 3d laser scans," *ISPRS Journal of Photogrammetry and Remote Sensing*, pp. 76–88, 2013.

[18] Eugene d'Eon, Bob Harrison, Taos Myers and Philip A. Chou, "JPEG Pleno Database: 8i Voxelized Full Bodies (8iVFB v2) - A Dynamic Voxelized Point Cloud Dataset," "https://bit.ly/3cJQ61a", 2017.

[19] Y. Feng, S. Liu, and Y. Zhu, "Real-time spatio-temporal lidar point cloud compression," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 10 766–10 773.

[20] Y. Feng, B. Tian, T. Xu, P. Whatmough, and Y. Zhu, "Mesorasi: Architecture support for point cloud analytics via delayed-aggregation," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2020, pp. 1037–1050.

[21] Geo Week News Staff, "Cloud Chamber: Low-cost smartphone app captures point clouds for AEC," "https://bit.ly/3DP1Qvc", 2018.

[22] S. Giancola, J. Zarzar, and B. Ghanem, "Leveraging shape completion for 3d siamese tracking," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[23] Google, "Draco," "https://github.com/google/draco", 2021.

[24] B. Han, Y. Liu, and F. Qian, *ViVo: Visibility-Aware Mobile Volumetric Video Streaming*. Association for Computing Machinery, 2020.

[25] X. He, H. L. Cao, and B. Zhu, "Advectivenet: An eulerian-lagrangian fluidic reservoir for point cloud processing," *arXiv preprint arXiv:2002.00118*, 2020.

[26] J. Hu, A. Shaikh, A. Bahremand, and R. LiKamWa, "Characterizing real-time dense point cloud capture and streaming on mobile devices," in *Proceedings of the 3rd ACM Workshop on Hot Topics in Video Analytics and Intelligent Edges*, 2021, pp. 1–6.

[27] L. Huang, S. Wang, K. Wong, J. Liu, and R. Urtasun, “Octsqueeze: Octree-structured entropy model for lidar compression,” in *CVPR*, 2020.

[28] T. Huang and Y. Liu, “3d point cloud geometry compression on deep learning,” in *Proceedings of the 27th ACM International Conference on Multimedia*, 2019, p. 890–898.

[29] 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 Processing Magazine*, pp. 118–123, 2019.

[30] Jeroen Baert, “Morton encoding/decoding through bit interleaving: Implementations,” ”https://bit.ly/30Rf506”, 2013.

[31] T. Karras, “Maximizing parallelism in the construction of bvhs, octrees, and k-d trees,” in *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics*, 2012, p. 33–37.

[32] J. Kim, J. Im, S. Rhyu, and K. Kim, “3d motion estimation and compensation method for video-based point cloud compression,” *IEEE Access*, pp. 83 538–83 547, 2020.

[33] N. Koh, P. K. Jayaraman, and J. Zheng, “Parallel point cloud compression using truncated octree,” in *2020 International Conference on Cyberworlds (CW)*, 2020, pp. 1–8.

[34] A. Kuntz, C. Bowen, and R. Alterovitz, “Fast anytime motion planning in point clouds by interleaving sampling and interior point optimization,” in *Robotics Research*. Springer, 2020, pp. 929–945.

[35] G. G. Langdon, “An introduction to arithmetic coding,” *IBM Journal of Research and Development*, pp. 135–149, 1984.

[36] D. Le Gall, “Mpeg: A video compression standard for multimedia applications,” *Commun. ACM*, p. 46–58, 1991.

[37] K. Lee, J. Yi, Y. Lee, S. Choi, and Y. M. Kim, *GROOT: A Real-Time Streaming System of High-Fidelity Volumetric Videos*. Association for Computing Machinery, 2020.

[38] B. Li, “3d fully convolutional network for vehicle detection in point cloud,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1513–1518.

[39] L. Li, Z. Li, V. Zakharchenko, J. Chen, and H. Li, “Advanced 3d motion prediction for video-based dynamic point cloud compression,” *IEEE Transactions on Image Processing*, pp. 289–302, 2020.

[40] Y. Lin, Z. Zhang, H. Tang, H. Wang, and S. Han, “Pointacc: Efficient point cloud accelerator,” in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2021, pp. 449–461.

[41] H. Liu, H. Yuan, Q. Liu, J. Hou, and J. Liu, “A comprehensive study and comparison of core technologies for mpeg 3-d point cloud compression,” *IEEE Transactions on Broadcasting*, pp. 701–717, 2020.

[42] M. Liu, “Robotic online path planning on point cloud,” *IEEE transactions on cybernetics*, vol. 46, no. 5, pp. 1217–1228, 2015.

[43] Marek Kowalski, Jacek Naruniec, “LiveScan3D-Hololens,” ”https://github.com/MarekKowalski/LiveScan3D-Hololens”, 2020.

[44] Marek Simonik, “Record3D-Record your own 3D Videos!” ”https://record3d.app/”, 2021.

[45] S. Martin, M. Stefan, A. Karl *et al.*, “Complex-yolo: real-time 3d objectdetection on point clouds,” in *Computer vision and pattern recognition*, 2018.

[46] Mayank Raj, “Point Clouds and its significance in AR,” ”https://bit.ly/3uknBjT”, 2020.

[47] D. Meagher, “Geometric modeling using octree encoding,” *Computer Graphics and Image Processing*, pp. 129–147, 1982.

[48] R. Mekuria, K. Blom, and P. Cesar, “Design, implementation, and evaluation of a point cloud codec for tele-immersive video,” *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 828–842, 2017.

[49] G. Meynet, Y. Nehmé, J. Digne, and G. Lavoué, “Pcqm: A full-reference quality metric for colored 3d point clouds,” in *2020 Twelfth International Conference on Quality of Multimedia Experience (QoMEX)*, 2020, pp. 1–6.

[50] A.-M. S. Mohamed, “Potential of 3d laser point cloud data usage for the tourism industry,” in *The International Conference on Civil and Architecture Engineering*, vol. 12, no. 12th International Conference on Civil and Architecture Engineering. Military Technical College, 2018, pp. 1–13.

[51] C. Moreno, Y. Chen, and M. Li, “A dynamic compression technique for streaming kinect-based point cloud data,” in *2017 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2017, pp. 550–555.

[52] MPEG, “G-PCC codec description v2,” ”https://bit.ly/3nN43C8”, 2019.

[53] MPEG, “MPEG Point Cloud Compression,” ”https://mpeg-pcc.org”, 2022.

[54] MPEG, “Point Cloud Video: Loot,” ”https://bit.ly/3QXDsPf”, 2022.

[55] MPEG, “Point Cloud Video: Redandblack,” ”https://bit.ly/3NyQq2R”, 2022.

[56] MPEGGroup, “Geometry based point cloud compression (G-PCC) test model,” ”https://github.com/MPEGGroup/mpeg-pcc-tmc13”, 2021.

[57] New Farmer Blogger, “3D points clouds for immersive real estate and telepresence experiences,” ”https://bit.ly/3AhWQR5”, 2015.

[58] Nvidia Corporation, “Jetson AGX Xavier Developer Kit,” ”https://bit.ly/3oWQNtH”, 2018.

[59] Park, Jounsup and Chou, Philip A. and Hwang, Jenq-Neng, “Rate-utility optimized streaming of volumetric media for augmented reality,” 2018.

[60] W. A. Pearlman and A. Said, *Entropy coding techniques*. Cambridge University Press, 2011, p. 41–76.

[61] Pix4D SA, “PIX4Dcatch: Turn your mobile device into a professional 3D scanner using the power of photogrammetry,” ”https://www.pix4d.com/product/pix4dcatch”, 2021.

[62] Point Cloud Library Contributors, “Module kdtree - Point Cloud Library (PCL),” ”https://pointclouds.org/documentation/group__kdtree.html”, 2022.

[63] Point Cloud Library Contributors, “Module octree - Point Cloud Library (PCL),” ”https://pointclouds.org/documentation/group__octree.html”, 2022.

[64] Point Cloud Library Contributors, “Pcl gpu octree,” ”https://github.com/PointCloudLibrary/pcl/tree/master/gpu/octree”, 2022.

[65] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, “Frustum pointnets for 3d object detection from rgb-d data,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[66] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “Pointnet: Deep learning on point sets for 3d classification and segmentation,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[67] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “Pointnet++: Deep hierarchical feature learning on point sets in a metric space,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, p. 5105–5114.

[68] F. Qian, B. Han, J. Pair, and V. Gopalakrishnan, “Toward practical volumetric video streaming on commodity smartphones,” in *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*, 2019, p. 135–140.

[69] Z. Que, G. Lu, and D. Xu, “Voxelcontext-net: An octree based framework for point cloud compression,” in *CVPR*, 2021, pp. 6042–6051.

[70] Ricci Rox, “Power-hungry Snapdragon 8 Gen 1 gets trounced by the Apple A15 Bionic in real-world gaming test,” ”https://bit.ly/3P086pp”, 2022.

[71] Richard Durant, “Pointerra: Attractive Opportunity If Growth Can Be Sustained,” ”https://bit.ly/3nvfRrx”, 2021.

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

[73] C. Santos, M. Gonçalves, G. Corrêa, and M. Porto, “Block-based inter-frame prediction for dynamic point cloud compression,” in *2021 IEEE International Conference on Image Processing (ICIP)*, 2021, pp. 3388–3392.

[74] R. Schnabel and R. Klein, “Octree-based point-cloud compression,” in *Proceedings of the 3rd Eurographics / IEEE VGTC Conference on Point-Based Graphics*, 2006, p. 111–121.

[75] S. Schwarz, M. Preda, V. Baroncini, M. Budagavi, P. Cesar, P. A. Chou, R. A. Cohen, M. Krivokuća, S. Lasserre, Z. Li, J. Llach, K. Mammou, R. Mekuria, O. Nakagami, E. Siahaan, A. Tabatabai, A. M. Tourapis, and V. Zakharchenko, “Emerging mpeg standards for point cloud compression,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, pp. 133–148, 2019.

[76] J. Shao, H. Zhang, Y. Mao, and J. Zhang, “Branchy-gnn: A device-edge co-inference framework for efficient point cloud processing,” in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 8488–8492.

[77] R. Shumaker and L. Stephanie, *Virtual, Augmented and Mixed Reality: Designing and Developing Augmented and Virtual Environments: 6th International Conference, VAMR 2014, Held as Part of HCI International 2014, Heraklion, Crete, Greece, June 22-27, 2014, Proceedings, Part I*. Springer, 2014.

[78] J. Sun, Y. Xie, S. Zhang, G. Zhang, H. Bao, and X. Zhou, “You Don’t Only Look Once: Constructing spatial-temporal memory for integrated 3d object detection and tracking,” *ICCV*, 2021.

[79] M. A. S. Teixeira, H. B. Santos, A. S. d. Oliveira, L. V. Arruda, and F. Neves, “Robots perception through 3d point cloud sensors,” in *Robot Operating System (ROS)*. Springer, 2017, pp. 525–561.

[80] TheAILearner Blogger, “Image Processing – Nearest Neighbour Interpolation,” ”https://bit.ly/3l4iY95”, 2018.

[81] TopoDOT Blogger, “Using Point Clouds for Augmented and Virtual Reality,” ”https://bit.ly/3OFdA97”, 2022.

[82] C. Tu, E. Takeuchi, A. Carballo, and K. Takeda, “Point cloud compression for 3d lidar sensor using recurrent neural network with residual blocks,” in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 3274–3280.

[83] D. Valenzuela-Urrutia, R. Muñoz-Riffo, and J. Ruiz-del Solar, “Virtual reality-based time-delayed haptic teleoperation using point cloud data,” *Journal of Intelligent & Robotic Systems*, vol. 96, no. 3, pp. 387–400, 2019.

[84] veesus.com, “INTRODUCING ZAPPCHA – MOBILE POINT CLOUD CAPTURE AND CLOUD-BASED STORAGE,” ”https://bit.ly/3xiSw0v”, 2021.

[85] Vision Lab, Nanjing University, “Multiscale Point Cloud Geometry Compression,” ”https://bit.ly/3xiAxah”, 2020.

[86] Z.-R. Wang, C.-G. Yang, and S.-L. Dai, “A fast compression framework based on 3d point cloud data for telepresence,” *Int. J. Autom. Comput.*, vol. 17, no. 6, p. 855–866, 2020.

[87] We are Luminous, “Interitum - Sureal VR Point Cloud Game,” ”https://www.youtube.com/watch?v=P5BgrdXis68”, 2016.

[88] C.-H. Wu, C.-F. Hsu, T.-K. Hung, C. Griwodz, W. T. Ooi, and C.-H. Hsu, “Quantitative comparison of point cloud compression algorithms with pcc arena,” *IEEE Transactions on Multimedia*, pp. 1–1, 2022.

[89] W. Wu, Z. Qi, and L. Fuxin, "Pointconv: Deep convolutional networks on 3d point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[90] T. Xu, B. Tian, and Y. Zhu, "Tigris: Architecture and algorithms for 3d perception in point clouds," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2019, p. 629–642.

[91] X. Yan, C. Zheng, Z. Li, S. Wang, and S. Cui, "Pointasnl: Robust point clouds processing using nonlocal neural networks with adaptive sampling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.