

This is the **accepted version** of the journal article:

Tzamarias, Dion Eustathios Olivier; Chow, Kevin; Blanes Garcia, Ian; [et al.]. «Fast Run-Length Compression of Point Cloud Geometry». IEEE transactions on image processing, Vol. 31 (June 2022), p. 4490-4501. DOI 10.1109/TIP.2022.3185541

This version is available at <https://ddd.uab.cat/record/290134>

under the terms of the  **IN COPYRIGHT** license

Fast Run-Length Compression of Point Cloud Geometry

Dion E. O. Tzamarías, Kevin Chow, Ian Blanes, and Joan Serra-Sagrà, *Senior Member, IEEE*

Abstract—The increase in popularity of point-cloud-oriented applications has triggered the development of specialized compression algorithms. In this paper, a novel algorithm is developed for the lossless geometry compression of voxelized point clouds following an intra-frame design. The encoded voxels are arranged into runs and are encoded through a single-pass application directly on the voxel domain. This is done without representing the point cloud via an octree nor rendering the voxel space through an occupancy matrix, therefore decreasing the memory requirements of the method. Each run is compressed using a context-adaptive arithmetic encoder yielding state-of-the-art compression results, with gains of up to 15% over *TMC13*, MPEG’s standard for point cloud geometry compression. Several proposed contributions accelerate the calculations of each run’s probability limits prior to arithmetic encoding. As a result, the encoder attains a low computational complexity described by a linear relation to the number of occupied voxels leading to an average speedup of 1.8 over *TMC13* in encoding speeds. Various experiments are conducted assessing the proposed algorithm’s state-of-the-art performance in terms of compression ratio and encoding speeds.

Index Terms—Point cloud geometry compression, run-length encoding, context-adaptive encoder.

I. INTRODUCTION

Recent advances in 3D acquisition technologies favor point clouds over polygonal meshes for the digital representation of three-dimensional objects. Consisting of a set of attributes and a dense collection of points, placed at precise 3D coordinates, point clouds achieve, in a simple structure, the realistic portrayal of static or even dynamic time-varying scenes. Attending to the advantages of such data are the numerous point cloud applications in areas such as Augmented/Virtual Reality (AR/VR), self-driving cars, or heritage preservation, to name but a few [1]–[5]. The importance of these applications coupled with the large size and irregular nature of point cloud data have sparked the emergence of an accrescent family of new specialized compression algorithms [6]. As the target of such applications are often smaller devices, such as smartphones in the case of AR/VR, there is an increasing necessity for fast, memory-efficient, low-complexity point cloud compression algorithms [7].

The information that a point cloud carries can be split into its *geometry* data, referring to the 3D coordinates of each point,

and its *attribute* data, referring to the value that each point carries, such as the color, the normal or its reflectivity [8]. Therefore, depending on the target, point cloud compression schemes are grouped into geometry or attribute algorithms.

A standard preprocessing step, after acquiring the point cloud data, is the quantization of its geometry information into integer coordinates [9]. This process is referred to as voxelization, where any point in the 3D space is represented as a cubic cell, called voxel and placed in a regular 3D cubic grid. A voxel is said to be occupied when it corresponds to a point of the point cloud, in contrast to unoccupied voxels that form the empty space. Although the voxelization of a point cloud reduces somewhat its original geometry information, additional processing is required for meaningful compression.

The lossy compression of point clouds can attain relatively low bit rates at the expense of introducing distortion on the decompressed data, whereas lossless compression schemes preserve flawlessly the original data at the cost of a higher rate. In the case of dynamically time-varying point cloud scenes, inter-frame encoders exploit redundancies among consecutive point cloud frames. These methods typically lead to higher compression ratios (lower bit rates), though their intra-frame counterparts, which exploit redundancies only within each individual frame, often perform at a lower computational complexity.

The taxonomy of existing lossless point cloud geometry compression designs can be organized according to the manner in which a point cloud is represented throughout the encoding process. These particular point cloud representations, apart from laying the foundations of the encoding process, aim to restrict the number of encoded non-occupied voxels.

The most prevalent representation of point clouds among compression schemes follows the octree decomposition, where the cubic voxel space is recursively subdivided into eight identically-sized cubes called octants [10]. At each iteration, only octants that contain at least one occupied voxel within their volume are further subdivided into children octants, whose occupancy status is indicated through a single bit. The resulting tree structure achieves an a priori compression of the point cloud geometry, though the additional use of transforms, contexts and entropy encoders drastically ameliorates an algorithm’s performance.

Some intra-frame methods combine octrees with context-based arithmetic or LZW encoders [11], while other transform-based

This paper has supplementary downloadable material available at <http://ieeexplore.ieee.org>, provided by the author. The material includes the core algorithms in pseudocode notation, as well as several proofs of equations. Contact dion.tzamarías@gmail.com for further questions about this work.

methods have octets reversibly transformed by a context-adaptive binary 3D Cellular Automata [12]–[14]. For dynamic point cloud scenes, inter-frame-based designs employ operations on octrees of adjacent frames [15], [16], or sort the octree based on previously encoded frames to promote efficient contexts [17]. In certain approaches, such is the large amount of contexts used in the entropy encoder, that even the context frequency tables have been compressed using an octree representation [11], [18], [19].

Aside from the octree decomposition, there are several different point cloud representations such as the recent G-Arrays known for reduced memory requirements and fast lookup speeds [20]. Unlike the TMC13 lossless geometry compression variant of MPEG, which is also based on the octree, the standardization working group has provided a lossy alternative based on projecting the point cloud on 2D surfaces and employing standard 2D video encoders [5], [8], [21], [22]. Most projection-based compression algorithms represent point cloud geometry through a series of 2D projections using 2D occupancy and depth maps such as the lossy [23]–[25] and lossless methods [26].

Other lossless geometry compression designs can operate straight on the voxel domain by directly processing the coordinates of the occupied voxels, e.g. encoding the residual differences between coordinates of occupied voxels [27]. The advantage of voxel-domain-type encoders lies on the circumvention of intricate preprocessing procedures that are dictated by the representation method, such as the construction of an octree or the calculation of various projection maps.

A sub-family of voxel domain compression methods utilizes the occupancy matrix to represent the point cloud structure. The entire cubic voxel space composed by N^3 voxels is modeled as a three-dimensional $N \times N \times N$ sparse boolean matrix whose elements of value 1 correspond to occupied voxels. Each 2D *slice* of the occupancy matrix, perpendicular to the x , y or z axis is represented by a $N \times N$ boolean array referred to as an image.

Using this representation, the authors of [28] compress the occupancy matrix of a point cloud through a boolean decomposition, and later via a dyadic decomposition and 3D contexts [29]. The latter's impressive results were further improved by extending its intra-frame design to an inter-frame variant [30], as well as by refining the context selection process [31], [32]. An amelioration of the intra-frame design was presented in [33] by removing the dyadic decomposition and utilizing the $N \times N$ projection matrix, calculated through a boolean OR operation across all images of the occupancy matrix perpendicular to the x , y or z axis. The binary projection matrix, as illustrated in Fig. 1b, appears as the shade of the point cloud that falls on the plane perpendicular to the chosen x , y or z axis. Transmitting the binary matrix to the decoder allows to limit the number of encoded voxels as one can deduce the absence of occupied voxels at precise 2D coordinates of all images.

Recent proposals deploy Deep Neural Networks for the ge-

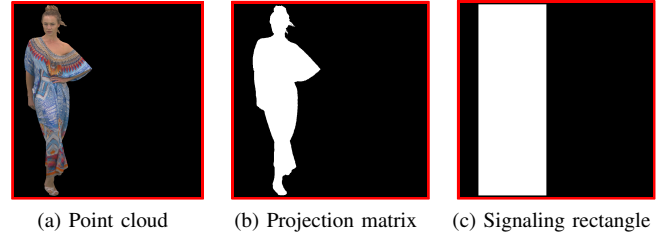


Fig. 1. Depiction of a point cloud along with different options to limit encoded voxels. Voxels being encoded are colored white in (b) and (c).

ometry compression of point cloud data, achieving very high compression ratios. In the case of the lossless designs, authors follow a hybrid representation of the point cloud, processing voxels through an octree as well as rendering sub-volumes of the voxel space through occupancy matrices [34]–[37]. Though it is important to mention that the complexity of these approaches leads to increased encoding times.

Although providing competitive results in terms of rate, these hybrid and occupancy-type methods render multiple sub-volumes or the entirety of the voxel space, which lead to increased memory requirements and large computational costs. These methods' sequential nature of processing one voxel at a time coupled with the unavoidable encoding of a large number of non-occupied voxels drastically decelerates the compression. As an example, in occupancy-type methods such as [33], it is common that only 2.3% of the processed and encoded voxels are occupied. Likewise, hybrid-type methods such as [37] show that the percentage of occupied voxels in the first level of processed sub-volumes is below 5%.

Our proposed lossless intra-frame geometry compression scheme solves the precedent drawbacks of hybrid and occupancy-type methods while maintaining state-of-the-art compression performance. Using only the point coordinates, the discussed method encodes the point cloud directly on the voxel domain without rendering the voxel space, contributing to its memory-efficient design. Furthermore, through grouping the encoded voxels in runs, the encoding time is drastically decreased. The introduction of several improvements accelerate key calculations, which lower the encoder's computational complexity to the order of occupied voxels, regardless of the number of encoded non-occupied voxels. Lastly, the employment of 3D contexts and an adaptive arithmetic encoder yields significant compression results.

The remainder of this paper is structured in the following manner. Section II summarizes the proposed method. Section III provides descriptions of the core processes. Experimental results are presented and analyzed in Section IV. Conclusions are discussed in Section V. The provided Supplementary material includes pseudocode algorithms and several proofs of equations.

II. PROPOSED METHOD

The proposed compression scheme, coined *FRL* –for Fast Run-Length–, operates directly on the voxel domain, in the

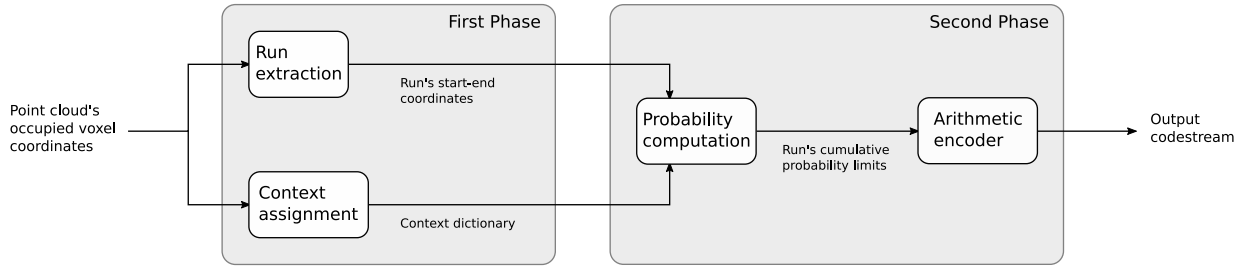


Fig. 2. Simplified version of the proposed single-pass compression scheme indicating the four main processes as well as their inputs and outputs. The processes' names match the title of their dedicated section in the manuscript.

absence of an octree or an occupancy matrix, by encoding runs of voxels through a context-adaptive arithmetic encoder. Its general structure bares similarities to our previous publication [33], which has been improved by virtue of several key novelties. The occupancy matrix is discarded entirely and the compression of individual voxels is replaced in favor of encoding sequences of voxels. Also novel is the acceleration of arduous calculations, required during the encoding process. As a consequence, the memory consumption as well as the overall computational complexity of the algorithm are significantly reduced, yielding a fast, high-performing point cloud geometry compression scheme. The simplified scheme of Fig. 2 indicates the main processes, grouped into two phases, along with each of their inputs and outputs that enable the encoding of sorted point clouds through a single pass. For point clouds that are not already sorted along a desired axis, a sorting preprocessing is required.

The first phase involves the *run-extraction* and *context-assignment* processes. Both receive as input the coordinates of the point cloud's occupied voxels. An essential component that stimulates the accelerated speed of *FRL* consists of forming symbols from runs of voxels rather than individual voxels. This takes place in the process *run-extraction*, which scans the point cloud's geometry information, forming runs of voxels that begin and end within a single 2D plane of the voxel space called a slice. Each run is entirely contained within a single slice and is composed by all non-occupied voxels which are positioned after the end of the previous run and one occupied voxel. I.e., the run ends as soon as the first occupied voxel is found following raster-scan order. The process is concluded after returning the voxel coordinates of each run's starting and ending points.

Similar to previous approaches [28], [29], [33], to enhance the prediction accuracy associated to the occupancy of any voxel in a run, the status of its neighboring voxels are also taken into account by means of contexts. The contexts that are employed in this paper are known as 3D contexts, composed by 5 voxels located on the current encoded slice and 9 voxels located on the previous one; cumulating to 14 voxels in total [29]. In Fig. 3, an illustration of the *context map* used in the proposed compression scheme depicts the location of the voxels that form the context, relative to the target voxel whose status is currently examined. Each context is identified according to a unique integer provided by a 14-bit-long bitmask that

is defined by the occupancy status of its constituent voxels. An occupied voxel at the position i of the context map contributes 2^i to the context's integer representation. Therefore, 2^{14} unique contexts are deployed in the proposed scheme, whose integer representations range from 0, in the case of the empty context composed solely of non-occupied voxels, to $2^{14} - 1$.

On account of contexts playing such an important role in the proposed encoder, the next key process of the phase-one stage involves the assignment of contexts to each voxel. The use of an occupancy matrix coupled with frequent matrix lookups would facilitate the allocation of contexts to each voxel. However, this would impact negatively the overall computational complexity as well as the memory consumption of the algorithm. To our advantage, the proposed *context-assignment* process, in absence of an occupancy matrix, constructs efficiently and inexpensively the context dictionary, a queue data structure that links voxels to their corresponding context. Therefore, the context dictionary accelerates drastically the proposed compression scheme as its size is of the order of occupied voxels and each of its elements is accessed a single time throughout the compression of the point cloud.

The second phase involves the *probability computation* and *arithmetic encoder* processes. Prior to encoding the runs through the *arithmetic encoder* process, each symbol should be associated with an upper and lower cumulative probability limit. This takes place in the *probability computation* process, which utilizes both outputs of the first phase. The introduction of several numerical shortcuts accelerate the calculations of the limits by restricting their complexity to the order of occupied voxels, regardless of the number of non-occupied voxels transmitted. Each run's occurrence probability is calculated through the independent joint probability involving the occupancy status of its component voxels. Furthermore, by means of contexts, the likelihood of a voxel's status is estimated adaptively, using conditional probabilities, through tracking and updating the number of times each specific context occurred to an occupied or a non-occupied voxel.

III. CORE PROCESSES

The first and second phase processes of the encoder's compression scheme act as the foundations of the proposed method. The functionality, motivation and novelties of each step are

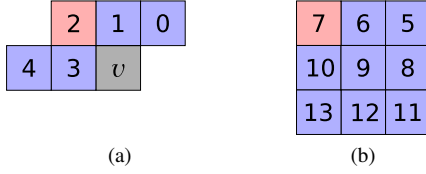


Fig. 3. Context map employed for encoding voxel v . The 3D contexts are formed by (a) five voxels from the current slice and (b) nine voxels of the previously encoded slice. The 2D coordinates of v and the 9th context coincide. If the non-occupied voxels are noted in blue and the occupied in red, v 's context's integer representation is $2^2 + 2^7 = 132$.

detailed in the following subsections. Simplified versions of their related algorithms as well as illustration of the pseudocode notation are provided as supplementary material to demonstrate their functionality.

A. Run extraction

Similar to classical image or text compression, processing groups of data rather than individual symbols is crucial to the speed of an encoder. As it is expected in most hybrid or occupancy-type point cloud geometry compression algorithms, the encoding of non-occupied voxels is unavoidable and often vastly surpass the number of occupied voxels. Therefore, processing and encoding one voxel at a time burdens the computational complexity and encoding speeds of the compression algorithm since they depend fully on the surplus of encoded non-occupied voxels. However, in the hereby introduced method, voxels are not processed individually, but in sequences, accelerating the encoding process and lowering the computational complexity of the algorithm to the order of occupied voxels.

In order to form the runs, the voxel space is first organized into 2D planes, i.e., slices, which are all perpendicular to a predefined axis. Such planes are never rendered as matrices and therefore do not constitute a type of point cloud rendering for the proposed method. For simplicity's sake, throughout the paper, it is assumed that the z -axis has been selected; therefore, the 2D coordinates of any voxel on a slice can be expressed through the x and y axis. Occupied voxels are then grouped by their respective slice such that all of those within a group contain the same z -coordinate. It is with the group's voxel coordinates that runs are formed such that collectively they express the full contents of a slice. Each run is entirely contained within a single slice and is composed primarily by non-occupied voxels, ending on the first encountered occupied voxel, following raster-scan order of the 2D plane. The unique case when a run is permitted to end prematurely on a non-occupied voxel is when it would otherwise extend beyond the end of a slice to reach an occupied voxel. Only in that particular case is the run devoid of an occupied voxel. Thus, given a slice composed of K occupied voxels, the number of runs in said slice is K , unless the slice ends in a non-occupied voxel, in which case it is $K + 1$. Treating such sequences as individual symbols puts in motion the first step of accelerating the encoding process.

It is clear that compression gains can be achieved by reducing the number of non-occupied voxels that are present in the runs. To this end, the encoded area of a slice's 2D voxel space is restricted through the introduction of a signaling rectangle. By registering the x and y coordinates of the north-, south-, west-, and east-most occupied voxels of the point cloud, the occupied voxels are enclosed within a tangential rectangle, indicating the absence of occupied voxels outside the confined area. A realistic example of a point cloud's signaling rectangle is depicted in Fig. 1c. Only the voxels located within the signaling rectangle participate in the runs as all voxels found outside the marked perimeter are dismissed. In the case where no occupied voxels are located within the signaling rectangle, the slice is flagged as empty, and the next plane is processed. Once the encoding process has been completed, the signaling rectangle can be then reproduced in the decoder by transmitting the x and y values of its top, bottom, left and right limits.

An example of the run extraction process is provided in Fig. 4 where each run of the slice is indicated with a different color. All runs are confined within the signaling rectangle, traced with a bold black line. Following raster-scan order, all runs end at occupied voxels, which are represented as squares marked with a dot, except for the last one in blue. The blue run is permitted to end at a non-occupied voxel as it is the last run that is contained within the slice.

The uniform nature of the signaling rectangle makes the computation of the run's length fast and effortless and permits the assignment of contexts without the use of matrices. Yet, the use of the signaling rectangle leads to the processing and encoding of additional non-occupied voxels, compared to the projection matrix; nevertheless, as justified later on and demonstrated in section IV, this does not affect the compression performance of the encoder nor its encoding speed. Since the proposed method encodes the point cloud using directly the coordinates of the occupied voxels, the compression scheme is clearly a voxel-domain-type encoder.

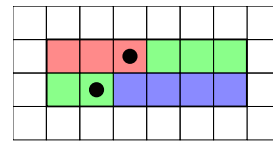


Fig. 4. Example of runs in a slice. Squares correspond to voxels, with the ones marked with a dot being occupied. The signaling rectangle is traced with a bold line. Each run is marked with a different color.

B. Context assignment

The identification of each voxel's context can significantly burden the overall compression scheme in terms of computational complexity. Competitive hybrid-type schemes and occupancy-array-type methods often identify the context of each voxel by rendering multiple volumes or the entire voxel-space through a large matrix such as an occupancy array. The usage of such a matrix burdens the algorithm's memory consumption and impinges on its execution speed. To obtain the context of each

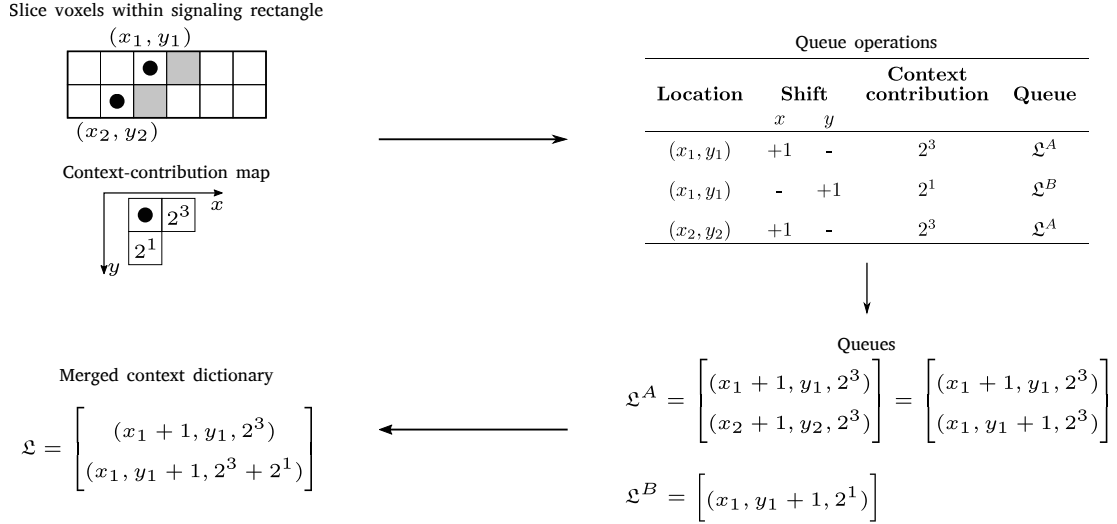


Fig. 5. Example demonstrating the calculation of the context dictionary. Dotted squares correspond to occupied voxels, specified by their coordinates, and shaded squares correspond to voxels linked to a non-empty context. This simplified example considers only the context map positions 1 and 3.

of the W encoded voxels, the status of several elements of the occupancy matrix is assessed, resulting in a complexity of $O(W)$. Given that, usually, the number of occupied voxels V is much smaller than the number of encoded voxels, the use of the occupancy matrix heavily strains the encoding speed and increases the overall computational complexity. In the proposed compression scheme, the complexity of this step is drastically decreased by discarding the occupancy matrix and hence alleviating the memory consumption.

To bypass the aforementioned issues of the context-assignment process, the devised method retrieves the necessary context information through, exclusively, a single-pass over the occupied voxels of a point cloud. This way, a slice's *context dictionary* is defined. Its role is to link the voxels included in any run of the slice with their non-empty context, having the particularity that voxels that are related to the empty context are entirely omitted from the dictionary. Algorithmically, the context dictionary is represented as a queue data structure, \mathcal{L} , composed by a series of triplets that include the two-dimensional coordinates of the voxel as well as the integer representation of its corresponding, non-empty, context. Following the example from the previous subsection and using a simplified set of contexts, Fig. 5 illustrates an instance of a context dictionary, in a slice composed by two occupied voxels (marked with a dot), as well as only two voxels with non-empty contexts (marked in gray). Therefore, if a voxel corresponds to a (non-empty) context, both the coordinates of the voxel, as well as the integer identifier of its context can be accessed. Voxels that are not present in the context dictionary are deduced to be linked to the empty context or outside the boundaries of the signaling rectangle.

To construct a slice's context dictionary without rendering the voxel space, only the 2D coordinates of occupied voxels are used. The computation of \mathcal{L} is based on the principle that only context map positions related to occupied voxels are required to deduce the integer representation of a voxel's

context. For example, referencing Fig. 3, it is only the context map positions of the occupied voxels, i.e., 2 and 7, that are required for the calculation of the integer representation of v 's context as $2^2 + 2^7 = 132$. Therefore, a voxel's context can be computed as a sum of the independent contributions of the occupied voxels in its context map.

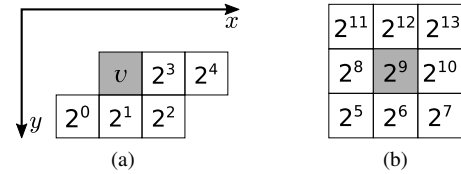


Fig. 6. Context-contribution map: Contribution of the occupied voxel v to the integer representation of the contexts of its surrounding voxels depending on their relative position. (a) Contribution to contexts of voxels located at the same slice as v range from 1 to 2^4 , while (b) the ones of voxels located at the next slice of v range from 2^5 to 2^{13} . The darker colored voxels, although located on different slices, share identical (x, y) coordinates.

To uphold the aforementioned single-pass characteristic, it should be possible from the coordinates of an occupied voxel v to retrieve the coordinates of all the voxels that contain v in any position of their context map. Furthermore, each of these voxel coordinates should be linked to the exact contribution of v to their respective contexts. This is achieved via the *context-contribution map* of Fig. 6. By appropriately shifting the coordinates of an occupied voxel, one can match its contribution with the coordinates of the affected voxel. For example, considering an occupied voxel with coordinates (x, y, z) , using the context-contribution map it is observed that it contributes 2^2 to the context of the voxel with coordinates $(x + 1, y + 1, z)$, or that it contributes 2^4 to the context of the voxel with coordinates $(x + 2, y, z)$. Regardless of an occupied voxel's global position, each unique shift on its coordinates is therefore linked to a unique contribution. Additionally, the shifted coordinates correspond to those of a voxel whose context has been impacted by said contribution.

Thus, using only the coordinates of occupied voxels, an

individual queue is constructed for each type of shift. By applying a given shift on all occupied voxels within a slice, a queue groups all the coordinates of voxels that have received a unique and identical context-contribution, storing them as triplets. Two examples of such queues are illustrated in Fig. 5 containing in each triplet the aforementioned 2D voxel coordinates and a unique context contribution. In the end, 14 such queues are utilized for the construction of the slice's context dictionary, one for each shift, i.e., one for each unique context contribution.

To illustrate the computation of such queues, the schematic of Fig. 5 provides a visual explanation. For simplicity's sake, only the context map positions 1 and 3 are taken into account, resulting in the simplified context-contribution map indicated in the schematic. As a result, the context dictionary will be calculated using only two queues, \mathcal{L}^A and \mathcal{L}^B , one for each type of shift as indicated in Fig. 5. Therefore, by shifting the x -coordinate of all occupied voxels by $+1$, it can be seen that the queue \mathcal{L}^A groups all the coordinates that have an occupied voxel at position 3 of their context-map, therefore receiving a context contribution of 2^3 .

By design, each such queue is sorted in ascending order, with the intent to expedite their merging into the context dictionary, composed by triplets with unique coordinates. As illustrated in Fig. 5, the elements of the queues with the same two-dimensional coordinates are integrated into a single triplet whose context integer representation is calculated as the sum of context-contributions of the merged triplets.

The use of the signaling rectangle facilitates the action of determining if a voxel should be added into the context dictionary, devoid of any matrix lookup, rendering the process fast and inexpensive. This would not be possible with a projection matrix since the voxel selection process would require a complexity-raising matrix search to determine if the considered voxel is to be encoded.

Considering that the context dictionary is constructed upon iterating only over occupied voxels, the context identification procedure has a computational complexity of the order of occupied voxels, $O(V)$. This is important as the majority of encoded voxels are found in long runs, of lengths above 80, and most of them in each run are related to the empty context whose frequency is independent from the number of occupied voxels (see Table I). By discarding the use of the occupancy matrix, we significantly decrease the memory usage of our algorithm. The computational complexity and the run time are drastically reduced thanks to the use of context dictionaries.

C. Probability computation

Ensuing the extraction of the runs and the calculation of the context dictionary, each run is compressed via an arithmetic encoder, which first estimates the upper and lower cumulative probability bounds for each run from the context dictionary,

TABLE I

ON THE LEFT, THE PERCENTAGE OF ENCODED VOXELS FOUND IN RUNS OF LENGTH HIGHER THAN 80 VOXELS. ON THE RIGHT, THE PERCENTAGE OF VOXELS OF EMPTY CONTEXT IN RUNS OF LENGTH ABOVE 80. RESULTS CORRESPOND TO AVERAGES ON ALL FRAMES OF EACH SEQUENCE.

Sequence	Encoded voxels included in runs of length ≥ 80 voxels (%)	Voxels linked to empty contexts in runs ≥ 80 voxels (%)
andrew9	97.81	97.25
david9	99.09	98.00
phil9	97.72	97.01
ricardo9	98.31	96.52
sarah9	97.70	96.94
<i>Average</i>	98.13	97.14
longdress	94.44	96.75
loot	96.20	97.24
redandblack	95.21	96.81
soldier	96.92	97.21
<i>Average</i>	95.69	97.00
basketballplayer	98.59	99.35
dancer	98.95	99.53
<i>Average</i>	98.77	99.44
Egyptianmask	99.98	99.99
Shiva	99.97	99.95
<i>Average</i>	99.98	99.97

and subsequently encodes it. For the purpose of modeling run probabilities, the proposed method takes into account the length of the sequence as well as the contexts of all its composing voxels as illustrated in the Markov chain of Fig. 7.

The Markov chain is expressed by the tree of $2n$ states shown in Fig. 7, where n is the length of the longest-possible run (i.e., the number of remaining uncoded voxels in a slice). The probability of a run is obtained by traversing the tree according to the occupancy of its voxels as follows. Starting at its root, whenever a voxel is occupied, the left child shall be taken; otherwise, the right child shall be taken. The transition probability leading to the i th state is given by the conditional probability of the voxel being occupied given its context c_i .

Each run is composed by a sequence of non-occupied voxels that ends at the first encountered occupied voxel, except for the special case of the last run of a slice. Therefore, the probability of any run is expressed by the independent joint probability of all states that are being traversed starting from the root and ending at any childless node of the tree. Hence, the probability of a run whose length is l and terminates at an occupied voxel is expressed by

$$P(l) = p(1|c_l) \cdot \prod_{i=1}^{l-1} p(0|c_i). \quad (1)$$

In order to arithmetically encode such a sequence of l voxels, one requires the upper and lower cumulative probability limits

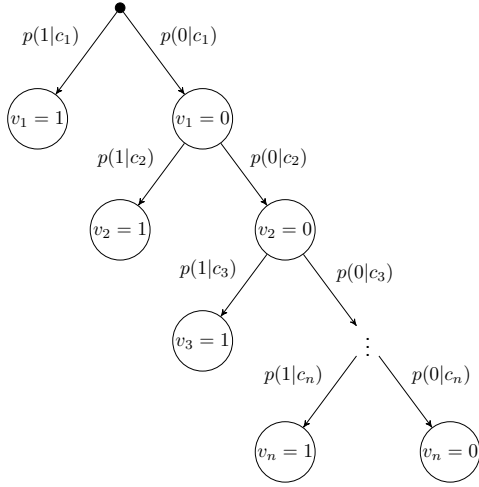


Fig. 7. Markov chain tree, modeling the states of voxels affiliated with runs of lengths shorter or equal to n . Occupied or empty voxels acquire the value of 1 or 0 respectively. The state probability of a given voxel i depends on its context c_i .

of the sequence:

$$\begin{aligned} \text{Low}(l) &= \sum_{r=1}^{l-1} P(r) = \sum_{r=1}^{l-1} p(1|c_r) \cdot \prod_{i=1}^r p(0|c_i), \\ \text{High}(l) &= \sum_{r=1}^l P(r) = \sum_{r=1}^l p(1|c_r) \cdot \prod_{i=1}^r p(0|c_i). \end{aligned} \quad (2)$$

At first glance, the calculation of the run's upper and lower cumulative probability bounds appears long and arduous. Therefore, the encoder presented in this paper offers two alternative ways of resolving this problem, each one described in the following subsections.

D. Fast Calculation of Probability Limits

While a direct computation of Eq. 2 is of notable complexity, it can be reformulated into the somewhat simpler Eq. 3 (see Appendix A in the Supplementary Material for the mathematical derivation).

$$\begin{aligned} \text{Low}(l) &= 1 - \prod_{i=1}^{l-1} p(0|c_i), \\ \text{High}(l) &= 1 - \prod_{i=1}^l p(0|c_i). \end{aligned} \quad (3)$$

However, by exploiting the adaptive nature of the encoder as described below, the complexity of Eq. 3 can be further decreased to the order of occupied voxels. Instead of examining each voxel in a run, this is achieved by only considering relevant run locations, while maintaining various counts that still allow to obtain identical probability limits.

At each run location linked to a non-empty context c , the counts of non-occupied and occupied voxels linked to any voxel are stored in, respectively, lists C^\square and C^\blacksquare , such that

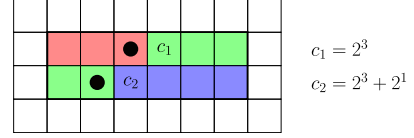


Fig. 8. Example for demonstrating the calculation of the cumulative probability limits of each run. Dotted squares correspond to occupied voxels and voxels with non-empty contexts are marked with their context integer representation, calculated at the example of the *Context assignment* section. Each run is indicated with a separate color.

$p(0|c) = \frac{C_c^\square}{C_c^\square + C_c^\blacksquare}$. I.e., for a non-occupied voxel with context c , the probability $p(0|c)$ is updated by increasing C_c^\square by one; for an occupied voxel with context c , the probability $p(0|c)$ is updated by increasing C_c^\blacksquare by one.

Conversely, the frequency of non-occupied voxels linked to the empty context is tracked by a single variable \mathcal{C} , which is updated at each occurrence of a non-occupied voxel associated to the empty context; i.e., the variable \mathcal{C} is incremented by one. On the contrary, variable \mathcal{C} is divided by two whenever the empty context is linked to an occupied voxel. The count of occupied voxels related to the empty context is assumed to be always equal to 1.

Employing this approach, the calculation of the lower cumulative probability limit for a run composed by k voxels of non-empty context and m voxels linked to the empty context, is given by (see proof in Appendix B in the Supplementary Material):

$$\text{Low}(l) = 1 - \frac{\mathcal{C}}{\mathcal{C} + m} \cdot \prod_{i=1}^k p(0|c_i). \quad (4)$$

For example, by applying Eq. 4 one can compute the probability limits of the colored runs marked in Fig. 8, where dotted squares correspond to occupied voxels, and voxels with non-empty contexts are marked with their integer context representation. Therefore, the cumulative probability limits of the red, green and blue run are calculated as $[1 - \frac{\mathcal{C}}{\mathcal{C}+2}, 1 - \frac{\mathcal{C}}{\mathcal{C}+3}]$, $[1 - \frac{\mathcal{C}}{\mathcal{C}+3} \cdot p(0|c_1), 1 - \frac{\mathcal{C}}{\mathcal{C}+4} \cdot p(0|c_1)]$ and $[1 - \frac{\mathcal{C}}{\mathcal{C}+3} \cdot p(0|c_2), 1]$ respectively. It should be highlighted that the upper cumulative limit of the blue run can be deduced to be equal to 1 as it ends on a non-occupied voxel, hence being represented by the final state of the Markov chain tree.

Meanwhile, the employment of the unique manner of tracking the frequencies related to the empty context does not influence the estimation of its conditional probabilities. On the occurrence of an occupied voxel linked to $c = 0$, by forcing its count N_1 to be equal to 1 and dividing \mathcal{C} by 2, the probability $p(1|c = 0)$ can still be computed accurately since

$$p(1|c = 0) = \frac{N_1}{N_1 + \mathcal{C}} = \frac{1}{1 + \mathcal{C}/2} = \frac{2}{2 + \mathcal{C}}.$$

This method of encoding the runs is coined fast-mode encoder as opposed to the slower single-mode encoder that is described in the following section.

E. Single-Mode Encoder

During the encoding of runs, the limited numerical precision of processors can cause small numerical errors in the calculations of the probability limits. Most of these numerical errors are harmless to the encoding process as they are recreated identically at the decoder. In some cases though, the difference between the upper and lower probability limits of long runs becomes so small that the arithmetic encoder interprets them as the same value. If left unchecked, this in turn would propagate underflow errors and render the decoder unable to reproduce the original point cloud. Therefore, these underflow cases should be correctly identified and an alternative to the fast-mode encoder should be devised to correctly encode such runs.

The identification of a problematic run simply involves the computation of its probability limits using Eq. 4 and comparing their difference against a specific threshold Q . Only in the cases where the difference becomes lower than the threshold will the fast-mode encoder trigger underflow errors and therefore an alternative type of encoder should be employed.

As the component voxels of a problematic run are unable to be encoded as a single sequence, a process should be carefully devised to rearrange them into admissible runs. One such strategy requires meticulously splitting the run into shorter sub-runs such that their probability limits are appropriately spaced. This procedure is coined single-mode encoding.

The mechanism of splitting a problematic run into sub-runs is performed by having the algorithm iterate over the voxels of the original run, in the order of their appearance in the sequence. At each iteration, a voxel is added to the current sub-run only if the difference of its updated probability limits is larger than the threshold Q . Obviously, the fast calculation of Eq. 4 is abandoned in favor of Eq. 3, which is updated at each inclusion of a new voxel into the current sub-run. In case the addition of a voxel would trigger an underflow, it is added in a new empty sub-run, whereas the old sub-run is terminated at the previous voxel. The following voxels are iteratively added to the new sub-run unless a potential underflow error triggers the initialization of a new sub-run.

The complexity of the single-mode encoder is of the order of the number of encoded voxels $O(W)$, significantly higher than the complexity of the fast-mode encoder. Nevertheless, the single-mode encoder is triggered in the rarest of cases since, on average, it is being employed 0.55 times per point cloud.

IV. EXPERIMENTAL RESULTS

To examine the characteristics and behavior of the proposed *FRL* method, several experiments are conducted including a performance comparison against other state-of-the-art geometry compression schemes. The experimentation and analysis are performed on several dynamic and static scenes. The dynamic scenes are provided by the Microsoft upper body [38] and the 8i Labs full body [39] datasets, containing voxelized

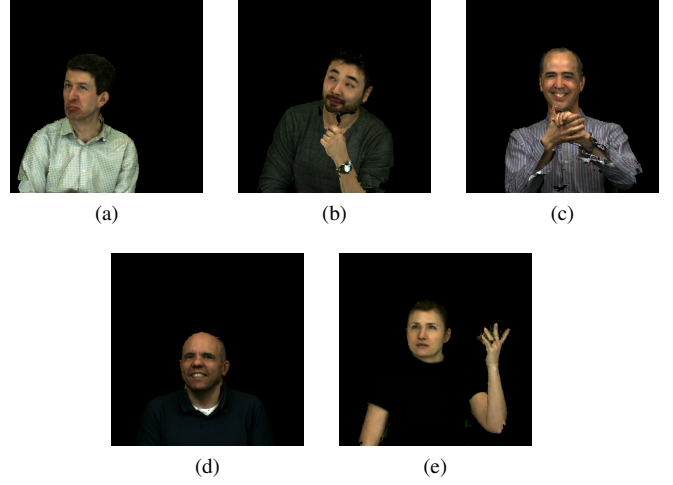


Fig. 9. 2D depiction of single frames from the Microsoft upper body database from the sequences (a) Andrew, (b) David, (c) Phil, (d) Ricardo and (e) Sarah.

point cloud sequences placed in a cubic space of dimensions of $512 \times 512 \times 512$ or $1024 \times 1024 \times 1024$ voxels, respectively known as 9 and 10 bit-depth resolution. Two-dimensional depictions of individual point cloud frames of each sequence are provided in Figs. 9 and 10. Each sequence's number of frames, average number of occupied voxels and size of voxel space is given in Table II.

The static voxelized scenes are provided by MPEG's test datasets; consisting of the larger point clouds basketball_player_vox11_00000200 and dancer_vox11_00000001 of the Owlii dataset [40], as well as the sparse Egyptian_mask_vox12 and Shiva_00035_vox12 of CTC [41]. These voxelized point clouds are contained in a $2048 \times 2048 \times 2048$ or $4096 \times 4096 \times 4096$ cubic voxel space, respectively known as 11 and 12 bit-depth resolution.

Although the proposed method follows a voxel-domain-based approach, the compression scheme bares many similarities to recent occupancy-array-type algorithms. Therefore, the following high performing –in terms of rate– lossless geometry compression algorithms are included in the comparisons: the intra-frame [33] and [29] methods, as well as inter-frame [32] method, noted as *SP*, *S3D* and *S4D* respectively. The proposed *FRL* method is also compared against competitive lossless hybrid-type methods such as [35] (noted as *MSVDNN*) along with both *VDNN* approaches with and without context extension, respectively [37] and [36]. The *TMC13* v14 MPEG variant [42], which is a recent standard for lossless geometry compression, is also considered in the comparison.

A. Rate Comparison

The comparison, in terms of data compression performance, against state-of-the-art approaches is carried out using all frames of the dynamic scenes as well as the single-frame



Fig. 10. 2D depiction of single frames from sequences of the 8i Labs full body database.

TABLE II
POINT CLOUD INFORMATION PER SEQUENCE.

Sequence	Frame count	Average occupied voxels	Size of voxel space
Dynamic scenes			
Microsoft upper body [38]			
andrew9	318	$2.84 \cdot 10^5$	$512 \times 512 \times 512$
david9	216	$3.49 \cdot 10^5$	$512 \times 512 \times 512$
phil9	245	$3.34 \cdot 10^5$	$512 \times 512 \times 512$
phil10	245	$1.49 \cdot 10^6$	$1024 \times 1024 \times 1024$
ricardo9	216	$2.26 \cdot 10^5$	$512 \times 512 \times 512$
ricardo10	216	$1.00 \cdot 10^6$	$1024 \times 1024 \times 1024$
sarah9	207	$2.60 \cdot 10^5$	$512 \times 512 \times 512$
8i Labs full body [39]			
longdress	300	$8.34 \cdot 10^5$	$1024 \times 1024 \times 1024$
loot	300	$7.94 \cdot 10^5$	$1024 \times 1024 \times 1024$
redandblack	300	$7.27 \cdot 10^5$	$1024 \times 1024 \times 1024$
soldier	300	$1.07 \cdot 10^6$	$1024 \times 1024 \times 1024$
Static scenes			
Owlii data [40]			
basketballplayer	1	$2.93 \cdot 10^6$	$2048 \times 2048 \times 2048$
dancer	1	$2.59 \cdot 10^6$	$2048 \times 2048 \times 2048$
CTC data [41]			
Egyptianmask	1	$2.73 \cdot 10^5$	$4096 \times 4096 \times 4096$
Shiva	1	$1.01 \cdot 10^6$	$4096 \times 4096 \times 4096$

static scenes listed in Table II. The reported results, which are measured in bits per occupied voxel (bpov) and indicated in Table III, are averaged over all the frames of each point cloud sequence.

Before delving into the analysis of the results, it should be noted that the choice of the axis in *FRL*, which determines the order in which the occupied voxels are being processed, is

selected through an empirically derived rule. Unless the variance of the points' coordinates along the y -axis is sufficiently larger than along the z -axis, the former axis is always selected. Therefore, the z -axis has been selected for all the sequences of the Microsoft upper body dataset, whereas the y -axis is selected for the rest of the sequences.

Regarding the dynamic scenes, from Table III it is visible that the proposed compression scheme yields very low bit rates, achieving the best reported results for 5 out of the 11 sequences. The herein described encoder significantly outperforms the *TMC13* standard, reporting gains of up to 15%. Additionally, *FRL* outperforms the two recent intra-frame occupancy array-type methods: although the proposed method surpasses *S3D* by a considerable margin, reporting average gains of 9.8%, that margin narrows when focusing on the comparison against *SP*. As mentioned later in section IV-B, *FRL* and *SP* methods follow a similar order of processing the point cloud and use identical 3D contexts, therefore achieving similar results. It appears though, that the usage of the signaling rectangle positively impacts the compression performance of *FRL*, leading to a consistent gain over *SP*. The competitive performance of *FRL* is also highlighted by its comparison against the inter-frame encoder *S4D*: *S4D* is more competitive on the upper body sequences while *FRL* performs better on the full body sequences. Be that as it may, the much larger computational complexity of *S4D*, aggravated by its context selection procedure, renders *FRL* the preferred choice.

Shifting the focus to the neural network hybrid algorithms, the updated version of *VDNN* [37], employing context extension, appears as the most competitive hybrid approach. The authors report compression rates, for a single frame per sequence, of 0.73 and 0.62 bpov on the 10 bit-depth resolution sequences of the upper and full body datasets respectively [38], [39]. The results of the neural network methods [36] and [35], presented in Table III, are directly taken from the cited papers, however the authors do not specify the number of frames that were tested in each sequence. From Table III it is clear that *FRL* is outperformed by *VDNN* for the sequences used for its evaluation in [36]. Though, this competitive performance is penalized by its lengthy run time, provided in [35], which is many orders of magnitude larger than *FRL*'s, as indicated later in Section IV-C. On the other hand, *FRL* outperforms *MSVDNN* on sequences of the Microsoft upper body dataset while the latter is more competitive only on sequences of the 8i Labs' full body dataset.

Although the previously mentioned neural network approach is an accelerated version of *VDNN*, *MSVDNN* is several orders of magnitude more computationally expensive than the proposed approach. This is highlighted by the encoding speed measurements of *MSVDNN* when implemented on the optimized PyTorch software and executed on a high-performance GeForce RTX 2080 GPU [35].

In the case of larger, dense static scenes, such as basketball or dancer, the performance of *FRL* does not appear to be hindered. On these point clouds it is observed that the average

TABLE III

RATE COMPARISON BETWEEN PROPOSED METHOD *FRL* AND STATE OF THE ART. MEASUREMENTS ARE AVERAGED OVER ALL AS WELL AS THE 100 FIRST FRAMES OF EACH SEQUENCE AND DISPLAYED IN BITS PER OCCUPIED VOXEL [BPOV]

Sequence	All frames (Intra coders)				Unspecified number of frames (Intra coders)		First 100 frames	
	FRL	SP [33]	S3D [29]	TMC13 [42]	VDNN [36]	MSVDNN [35]	(Intra coder) FRL	(Inter coder) S4D [32]
andrew9	1.00	1.02	1.12	1.13	-	-	1.01	0.95
david9	0.94	0.96	1.05	1.07	-	-	0.94	0.94
phil9	1.02	1.04	1.14	1.17	0.92	-	1.02	1.02
phil10	0.95	-	-	-	0.83	1.02	0.95	-
ricardo9	0.93	0.95	1.03	1.07	0.72	-	0.94	0.90
ricardo10	0.89	-	-	-	0.75	0.95	0.90	-
sarah9	0.95	0.97	1.06	1.07	-	-	0.96	0.92
Average	0.95	0.99	1.08	1.10	0.81	0.99	0.96	0.95
longdress	0.86	0.89	0.95	1.02	-	-	0.86	0.88
loot	0.83	0.86	0.92	0.97	0.64	0.63	0.82	0.84
redandblack	0.94	0.96	1.03	1.09	0.73	0.87	0.93	0.94
soldier	0.88	0.91	0.97	1.04	-	-	0.88	0.65
Average	0.88	0.91	0.97	1.03	0.69	0.75	0.87	0.83
basketballplayer	0.80	-	-	0.90	-	-	-	-
dancer	0.77	-	-	0.89	-	-	-	-
Average	0.79	-	-	0.90	-	-	-	-
Egyptianmask	18.20	-	-	11.78	-	-	-	-
Shiva	15.11	-	-	9.68	-	-	-	-
Average	16.66	-	-	10.73	-	-	-	-

compression gains over TMC13 are at 12%. However, in sparse point clouds such as Egyptianmask and Shiva, FRL is in deficit in terms of compression performance. This might very well be due to the limited sub-volume of the voxel space that is covered by the employed contexts, since they are composed only by 14 voxels. That is to say, in sparse point clouds such is the isolation of individual voxels that the contexts struggle to extract meaningful information from local voxel neighborhoods. Such a phenomenon has been studied in the hybrid geometry encoder [37], where a decrease in performance over sparse data was also reported. To somewhat alleviate such a drawback, the authors proposed a context extension strategy which increases the volumetric space covered by the contexts.

B. Projection matrix vs signaling rectangle

The similarity in terms of rate between the proposed *FRL* and *SP*, whose performance are compared in section IV-A, is due to the use of identical contexts. The main differences between these two approaches that could affect their performance in terms of rate is the use of different parameters in the arithmetic encoder and the replacement of the projection matrix with the signaling rectangle in *FRL*. Therefore, the demonstrated gains over *SP* are attributed to both the signaling rectangle and an optimized arithmetic encoding. That said, if the two methods both employed a signaling rectangle, and identical encoder parameters, the final upper and lower probability limits determined by their arithmetic encoder related to the entire point cloud would be identical.

In the results of Table IV it is noticeable that the encoding of the signaling rectangle is much more cost-efficient than the

encoding of the projection matrix. This is because the corners of the signaling rectangle are transmitted to the decoder using 16 bits apiece. On the other hand in [33], the 2D projection matrix, which resembles a binary image with intricate curves and forms, is much more costly to encode.

However, the advantage of the aforementioned matrix is linked to its ability to reduce the number of encoded voxels significantly more than the signaling rectangle. In the last column of Table IV it is evident that the proposed method, through the signaling rectangle, encodes at least 76% more voxels than in [33] using the projection matrix. That said, almost the entirety of the additionally encoded non-occupied voxels are associated with the empty context, which predicts extremely accurately the occupancy status of a voxel. Therefore, the low coding cost of the signaling rectangle counteracts the small increase in rate attributed to the surplus of encoded voxels. Hence, by introducing the signaling rectangle in the proposed compression scheme, a significant reduction in memory consumption, computational complexity and encoding speed can be achieved without compromising its rate-related performance. Due to such drawbacks in memory consumption, the *SP* implementation (in Matlab) is unable to process very large point clouds, which is why the results for the static scenes are absent from Table IV.

C. Run-Time Comparison

The proposed point cloud encoder is designed with the aim to significantly reduce encoding times and computational complexity while maintaining a very competitive geometry compression performance. In the following experiment, the

TABLE IV
CODING COST OF PROJECTION MATRIX VERSUS SIGNALING RECTANGLE.
THE RIGHT-MOST COLUMN INFORMS ON THE PERCENTAGE SURPLUS OF
ENCODED VOXELS IF THE SIGNALING RECTANGLE IS USED INSTEAD OF
THE PROJECTION MATRIX. MEASUREMENTS ARE AVERAGED OVER ALL
FRAMES OF EACH SEQUENCE.

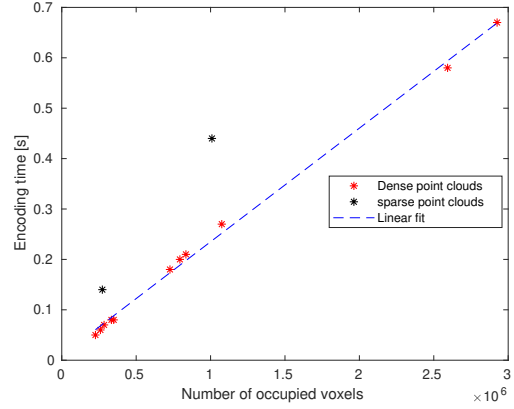
Sequence	Projection matrix (bpov)	Signaling rectangle (bpov)	Surplus of encoded voxels (%)
andrew9	0.0170	$22.5 \cdot 10^{-5}$	145.15
david9	0.0188	$18.3 \cdot 10^{-5}$	178.19
phil9	0.0221	$19.2 \cdot 10^{-5}$	80.31
ricardo9	0.0168	$28.3 \cdot 10^{-5}$	84.29
sarah9	0.0186	$24.6 \cdot 10^{-5}$	120.32
Average	0.0187	$22.6 \cdot 10^{-5}$	121.65
longdress	0.0031	$7.7 \cdot 10^{-5}$	75.52
loot	0.0049	$8.1 \cdot 10^{-5}$	89.50
redandblack	0.0060	$8.8 \cdot 10^{-5}$	89.68
soldier	0.0034	$6.0 \cdot 10^{-5}$	92.05
Average	0.0044	$7.7 \cdot 10^{-5}$	86.69

proposed *FRL* is coded in C and the timing performance of each of the algorithms was evaluated on an 8 core 2.8GHz Intel Core i7-7700HQ CPU.

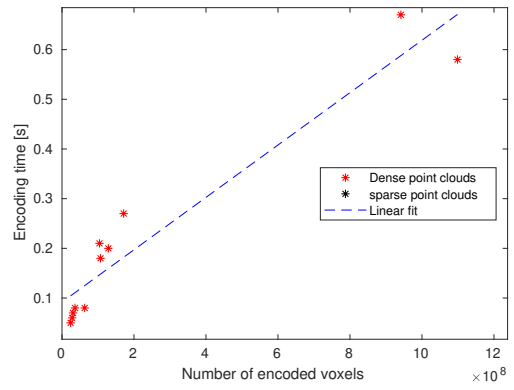
Given that the complexity of the proposed algorithm is of the order of occupied voxels $O(V)$, the increase in the number of encoded non-occupied voxels caused by the use of the signaling rectangle does not affect the overall encoding time. As illustrated in Fig. 11, the encoding time of the proposed algorithm on dense point clouds (red dots), is linearly correlated with the number of occupied voxels and not the number of encoded voxels, confirming the effect of Eq. 4 on the speed of the compression scheme. As reported in Table II, the percentage of occupied voxels is between 0.0004% and 0.26% of the total number of voxels, which illustrate the very competitive behavior of our approach. In the case of sparse data (black dots) the linear correlation of Fig. 11 seems disrupted even though the single mode encoder is not triggered more often than in the case of dense data. This increase in encoding time is due to the negligible terms in the complexity calculation related to the increased number of voxels linked to non-empty contexts (which can be at most 14 times the number of occupied voxels on a slice).

Table V marks the encoding time measurements on static as well as on dynamic scenes, which are provided as averages over all frames of each sequence. The proposed *FRL* method has only been compared against *TMC13* v14 (implemented in C++) since the latter is the most competitive among the algorithms used in the experiments, in terms of execution time. To assure a fair comparison, both encoders have been build using exactly same optimization level: -O3 and -march=native.

In addition to the average encoding time of each sequence, the initial sorting of the point cloud along a selected axis is also taken into account in the case of *FRL*. Following the reported results it is observed that for dense point clouds, such as the dynamic scenes and the OwlII data, *FRL* is on average 1.4



(a) Occupied voxels vs encoding time (average)



(b) Encoded voxels vs encoding time (average). Sparse data points are out of view with an encoding time of sub 0.44s and a number of encoded voxels of $2.9 \cdot 10^{10}$ and $5.6 \cdot 10^{10}$

Fig. 11. Observed relation between encoding time and voxel count.

times faster than *TMC13*. In the case of sparse scenes such as Egyptianmask and Shiba, the increase in encoding speed of the proposed method over *TMC13* is almost three-fold.

In the case of the hybrid-based methods, the authors report that, on average, the encoding speeds of *MSVDNN* as well as both *VDNN* with and without context extension, were respectively, 14, 3186 and 658 times slower than *TMC13* [35]. These averages were computed on a test set of dynamic and static scenes of a bit depth of 10 and 9. Therefore, since our experimental results have demonstrated that, on average, *FRL* is more than 1.3 times faster than *TMC13*, on a similar test set, it is safe to say that the proposed encoder is many orders of magnitude faster than the aforementioned hybrid approaches.

V. CONCLUSION

A novel lossless point cloud geometry compression algorithm is proposed in this paper. Its intra-frame design permits its use on static and dynamic scenes. The proposed encoder is applied directly on the voxel domain of the point cloud, therefore limiting requirements on memory consumption by avoiding the use

TABLE V

ENCODING TIME COMPARISON BETWEEN PROPOSED METHOD *FRL* AND STATE OF THE ART. MEASUREMENTS ARE AVERAGED ON ALL FRAMES OF EACH SEQUENCE AND DISPLAYED IN SECONDS.

Sequence	FRL		TMC13 [42]	Speedup over TMC13
	Sorting	Encoding	Total	
andrew9	0.005	0.07	0.07	0.10
david9	0.006	0.08	0.09	0.13
phil9	0.006	0.08	0.08	0.12
ricardo9	0.004	0.05	0.06	0.08
sarah9	0.005	0.06	0.06	0.09
Average	0.005	0.07	0.07	0.10
longdress	0.017	0.21	0.22	0.31
loot	0.016	0.20	0.21	0.29
redandblack	0.014	0.18	0.20	0.28
soldier	0.021	0.27	0.29	0.40
Average	0.017	0.22	0.23	0.32
basketball	0.061	0.67	0.74	1.05
dancer	0.054	0.58	0.64	0.92
Average	0.058	0.63	0.69	0.99
Egyptianmask	0.009	0.14	0.15	0.48
Shiva	0.025	0.44	0.47	1.30
Average	0.017	0.29	0.31	0.89

of an occupancy matrix or of an octree representation. Instead of encoding one voxel at a time, entire runs of voxels are encoded using a run-length-adaptive arithmetic encoder. The proposal of several novelties that accelerate key calculations, regarding the cumulative probability limits of each run, render the algorithm, on average, 1.8 times faster than *TMC13* and of low computational complexity. The introduction of the signaling rectangle restricts the number of encoded voxels while the employment of 3D contexts yields competitive state-of-the-art performance in terms of bit-rate. A series of experimental results confirm the preceding claims, including compression ratio (or bit-rate) and execution time comparison against state-of-the-art compression schemes. Future research will be steered towards ameliorating the algorithm's performance on sparse data, improving the contexts for each point cloud and optimizing the axis selection choice without impacting heavily the overall complexity of the algorithm.

ACKNOWLEDGMENT

This research was partially funded by Universitat Autònoma de Barcelona under grant 472-02-1/2017, by the Secretary of Universities and Research (Government of Catalonia) under grant 2017SGR-463, and by the Spanish Ministry of Economy and Competitiveness, the Spanish Ministry of Science and Innovation, and the European Regional Development Fund under grants RTI2018-095287-B-I00 (MINECO/FEDER, UE) and PID2021-125258OB-I00 (MICINN/FEDER, UE).

REFERENCES

- [1] D. Ni, A. Y. Nee, S.-K. Ong, H. Li, C. Zhu, and A. Song, "Point cloud augmented virtual reality environment with haptic constraints for teleoperation," *Transactions of the Institute of Measurement and Control*, vol. 40, no. 15, pp. 4091–4104, 2018.
- [2] Y. Wang, S. Zhang, B. Wan, W. He, and X. Bai, "Point cloud and visual feature-based tracking method for an augmented reality-aided mechanical assembly system," *The International Journal of Advanced Manufacturing Technology*, vol. 99, no. 9, pp. 2341–2352, 2018.
- [3] K. Kohira and H. Masuda, "Point-cloud compression for vehicle-based mobile mapping systems using portable network graphics," *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, vol. IV-2/W4, pp. 99–106, 2017.
- [4] A. Ullah, M. Watanabe, A. Kubo *et al.*, "Analytical point-cloud based geometric modeling for additive manufacturing and its application to cultural heritage preservation," *Applied Sciences*, vol. 8, no. 5, p. 656, 2018.
- [5] D. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, and A. Tabatabai, "An overview of ongoing point cloud compression standardization activities: video-based (v-pcc) and geometry-based (g-pcc)," *APSIPA Transactions on Signal and Information Processing*, vol. 9, 2020.
- [6] C. Cao, M. Preda, V. Zakharchenko, E. S. Jang, and T. Zaharia, "Compression of sparse and dense dynamic point clouds—methods and standards," *Proceedings of the IEEE*, 2021.
- [7] 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, pp. 135–140.
- [8] 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*, vol. 66, no. 3, pp. 701–717, 2019.
- [9] 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.
- [10] D. Meagher, "Geometric modeling using octree encoding," *Computer graphics and image processing*, vol. 19, no. 2, pp. 129–147, 1982.
- [11] D. C. Garcia and R. L. de Queiroz, "Intra-frame context-based octree coding for point-cloud geometry," in *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2018, pp. 1807–1811.
- [12] S. Milani, "Fast point cloud compression via reversible cellular automata block transform," in *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2017, pp. 4013–4017.
- [13] S. Limuti, E. Polo, and S. Milani, "A transform coding strategy for voxelized dynamic point clouds," in *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2018, pp. 2954–2958.
- [14] S. Milani, E. Polo, and S. Limuti, "A transform coding strategy for dynamic point clouds," *IEEE Transactions on Image Processing*, vol. 29, pp. 8213–8225, 2020.
- [15] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach, "Real-time compression of point cloud streams," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 778–785.
- [16] R. L. de Queiroz, D. C. Garcia, P. A. Chou, and D. A. Florencio, "Distance-based probability model for octree coding," *IEEE Signal Processing Letters*, vol. 25, no. 6, pp. 739–742, 2018.
- [17] D. C. Garcia and R. L. de Queiroz, "Context-based octree coding for point-cloud video," in *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2017, pp. 1412–1416.
- [18] D. C. Garcia, T. A. Fonseca, R. U. Ferreira, and R. L. de Queiroz, "Geometry coding for dynamic voxelized point clouds using octrees and multiple contexts," *IEEE Transactions on Image Processing*, vol. 29, pp. 313–322, 2019.
- [19] D. C. Garcia, T. A. Fonseca, and R. L. De Queiroz, "Example-based super-resolution for point-cloud video," in *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2018, pp. 2959–2963.

[1] D. Ni, A. Y. Nee, S.-K. Ong, H. Li, C. Zhu, and A. Song, "Point cloud augmented virtual reality environment with haptic constraints for

- [20] H. Roodaki, M. Dehyadegari, and M. N. Bojnordi, “G-arrays: Geometric arrays for efficient point cloud processing,” in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 1925–1929.
- [21] S. Schwarz, M. Preda, V. Baroncini, M. Budagavi, P. Cesar, P. A. Chou, R. A. Cohen, M. Krivokuća, S. Lasserre, Z. Li *et al.*, “Emerging MPEG standards for point cloud compression,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 1, pp. 133–148, 2018.
- [22] 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*, vol. 36, no. 3, pp. 118–123, 2019.
- [23] I. Tabus, E. C. Kaya, and S. Schwarz, “Successive refinement of bounding volumes for point cloud coding,” in *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*. IEEE, 2020, pp. 1–6.
- [24] J. Xiong, H. Gao, M. Wang, H. Li, K. N. Ngan, and W. Lin, “Advanced geometry surface coding for dynamic point cloud compression,” *arXiv preprint arXiv:2103.06549*, 2021.
- [25] Y. Xu, W. Zhu, Y. Xu, and Z. Li, “Dynamic point cloud geometry compression via patch-wise polynomial fitting,” in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 2287–2291.
- [26] E. C. Kaya, S. Schwarz, and I. Tabus, “Refining the bounding volumes for lossless compression of voxelized point clouds geometry,” in *2021 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2021, pp. 3408–3412.
- [27] W. Zhu, Y. Xu, L. Li, and Z. Li, “Lossless point cloud geometry compression via binary tree partition and intra prediction,” in *2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP)*. IEEE, 2017, pp. 1–6.
- [28] R. Rosário and E. Peixoto, “Intra-frame compression of point cloud geometry using boolean decomposition,” in *2019 IEEE Visual Communications and Image Processing (VCIP)*. IEEE, 2019, pp. 1–4.
- [29] E. Peixoto, “Intra-frame compression of point cloud geometry using dyadic decomposition,” *IEEE Signal Processing Letters*, vol. 27, pp. 246–250, 2020.
- [30] E. Peixoto, E. Medeiros, and E. Ramalho, “Silhouette 4d: An inter-frame lossless geometry coder of dynamic voxelized point clouds,” in *2020 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2020, pp. 2691–2695.
- [31] E. Ramalho and E. Peixoto, “Context selection for lossless compression of bi-level images,” in *XXXVIII Simpósio Brasileiro de Telecomunicações e Processamento de Sinais*, 2021.
- [32] E. Ramalho, E. Peixoto, and J. E. G. de Medeiros, “Silhouette 4d with context selection: Lossless geometry compression of dynamic point clouds,” *IEEE Signal Processing Letters*, 2021.
- [33] D. E. Tzamaras, K. Chow, I. Blanes, and J. Serra-Sagristà, “Compression of point cloud geometry through a single projection,” in *2021 Data Compression Conference (DCC)*. IEEE, 2021, pp. 63–72.
- [34] E. C. Kaya and I. Tabus, “Neural network modeling of probabilities for coding the octree representation of point clouds,” *arXiv preprint arXiv:2106.06482*, 2021.
- [35] D. T. Nguyen, M. Quach, G. Valenzise, and P. Duhamel, “Multiscale deep context modeling for lossless point cloud geometry compression,” in *2021 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*. IEEE, 2021, pp. 1–6.
- [36] —, “Learning-based lossless compression of 3d point cloud geometry,” in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 4220–4224.
- [37] —, “Lossless coding of point cloud geometry using a deep generative model,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 31, no. 12, pp. 4617–4629, 2021.
- [38] C. Loop, Q. Cai, S. O. Escolano, and P. Chou, “Microsoft voxelized upper bodies - a voxelized point cloud dataset,” *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document m38673/M72012*, Geneva, May 2016.
- [39] E. d’Eon, B. Harrison, T. Myers, and P. A. Chou, “8i voxelized full bodies - a voxelized point cloud dataset,” *ISO/IEC JTC1/SC29/WG1 input document M74006 and ISO/IEC JTC1/SC29/WG11 input document m40059*, Geneva, January 2017.
- [40] Y. Xu, Y. Lu, and Z. Wen, “OwlII Dynamic human mesh sequence dataset,” *ISO/IEC JTC1/SC29/WG11*, 120th MPEG Meeting, Macau, Input document m41658, Oct. 2017.
- [41] C. Tulvan, A. Gabrielli, and M. Preda, “Datasets update on Point Cloud compression for cultural objects,” *ISO/IEC JTC1/SC29/WG11*, 115th MPEG meeting, Geneva, Input document m38678, May 2016.
- [42] “C++ MPEG group TMC13 implementation,” [Online] Available: <https://github.com/MPEGGroup/MPEG-pcc-tmc13>, accessed: May 2022.