Tampere University

EMRE CAN KAYA

Visual and Geometric Data Compression for Immersive Technologies

Tampere University Dissertations 687

Tampere University Dissertations 687

EMRE CAN KAYA

Visual and Geometric Data Compression for Immersive Technologies

ACADEMIC DISSERTATION
To be presented, with the permission of
the Faculty of Information Technology and Communication Sciences
of Tampere University,
for public discussion at Tampere University
on 4 November 2022, at 12 o’clock.

ACADEMIC DISSERTATION
Tampere University, Faculty of Information Technology and Communication Sciences
Finland

Responsible supervisor and Custos	Professor Ioan Tabus Tampere University Finland	
Pre-examiners	Professor Søren Forchhammer Technical University of Denmark Denmark	Professor Titus Zaharia Télécom SudParis France
Opponent	Assoc. Professor Simone Milani University of Padova Italy	

ISBN 978-952-03-2605-0 (print)
ISBN 978-952-03-2606-7 (pdf)
ISSN 2489-9860 (print)
ISSN 2490-0028 (pdf)
http://urn.fi/URN:ISBN:978-952-03-2606-7

HIILINEUTRAALI PAINOTUOTE
ClimateCalc CC-000025/FI
PunaMusta Printing

Carbon dioxide emissions from printing Tampere University dissertations have been compensated.

PunaMusta Oy – Yliopistopaino
Joensuu 2022

PREFACE/ACKNOWLEDGEMENTS

This dissertation is based on my research at Tampere University during 2019 to 2022. The purpose of this dissertation is to present novel compression algorithms for visual and geometric data related to immersive technologies.

Firstly, I wish to express my sincere appreciation and my highest level of gratitude to Prof. Ioan Tabus, for all the strong guidance, advice and cooperation he has provided regarding all theoretical and practical aspects of my studies during this fruitful research period. Secondly, I wish to thank to Sebastian Schwarz from Nokia Technologies, with whom I had the privilege to cooperate. Next, I wish to sincerely thank Prof. Søren Forchhammer for acting as pre-examiner, Prof. Titus Zaharia for acting as pre-examiner, and Prof. Simone Milani for acting as opponent.

I wish to express my gratitude to my colleague Emanuele Palma for his helpful, constructive and positive attitude which always enabled a peaceful and friendly workplace atmosphere. Moreover, I am also deeply thankful to Pekka Astola, with whom I had the privilege to work in the same office in my initial year and who also provided guidance in several aspects regarding my studies. I would particularly like to thank my parents and sister for their love and care. I am very grateful to Kari Suomela and Virve Larmila for all the administrative, technical assistance and more importantly for helping me master the Finnish language. Finally, I am very grateful to all the wonderful people from all around the world that I shared the work environment with, in the Computing Sciences Unit of the Faculty of ITC in Tampere University.

ABSTRACT

The contributions of this thesis are new compression algorithms for light field images and point cloud geometry. Light field imaging attracted wide attention in the recent decade, partly due to emergence of relatively low-cost handheld light field cameras designed for commercial purposes whereas point clouds are used more and more frequently in immersive technologies, replacing other forms of 3D representation. We obtain successful coding performance by combining conventional image processing methods, entropy coding, learning-based disparity estimation and optimization of neural networks for context probability modeling.

On the light field coding side, we develop a lossless light field coding method which uses learning-based disparity estimations to predict any view in a light field from a set of reference views. On the point cloud geometry compression side, we develop four different algorithms. The first two of these algorithms follow the so-called bounding volumes approach which initially represents a part of the point cloud in two depth maps where the remaining points of the cloud are contained in a bounding volume which can be derived using only the two depth maps that are losslessly transmitted. One of the two algorithms is a lossy coder that reconstructs some of the remaining points in several steps which involve conventional image processing and image coding techniques. The other one is a lossless coder which applies a novel context arithmetic coding approach involving gradual expansion of the reconstructed point cloud into neighboring voxels. The last two of the proposed point cloud compression algorithms use neural networks for context probability modeling for coding the octree representation of point clouds using arithmetic coding. One of these two algorithms is a learning-based intra-frame coder which requires an initial training stage on a set of training point clouds. The lastly presented algorithm is an inter-frame (sequence) encoder which incorporates the neural network training into the encoding stage, thus for each sequence of point clouds, a specific neural network model is optimized which is also transmitted as a header in the bitstream.

CONTENTS

List of Figures

List of Tables

ABBREVIATIONS

2D	Two Dimensional
3D	Three Dimensional
bpov	bits per occupied voxel
BV	Bounding Volumes
CL	Codelength
CNN	Convolutional Neural Network
G-PCC	Geometry based Point Cloud Compression (Standardization Process)
LBV	Lossy Bounding Volumes
MLP	Multi Layer Perceptron
MSE	Mean Square Error
MVUB	Microsoft Voxelized Upper Bodies
NN	Neural Network
PC	Point Cloud
PCC	Point Cloud Compression
PSNR	Peak Signal to Noise Ratio
V-PCC	Video based Point Cloud Compression (Standardization Process)

ORIGINAL PUBLICATIONS

Publication I	E. C. Kaya and I. Tabus. Corner view disparity estimation for lossless light field compression. *2019 1st European Light Field Imaging Workshop (ELFI)*. 2019.
Publication II	I. Tabus, E. C. Kaya and S. Schwarz. Successive Refinement of Bounding Volumes for Point Cloud Coding. *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*. 2020, 1–6. DOI: `10.1109/MMSP48831.2020.9287106`.
Publication III	E. C. Kaya, S. Schwarz and I. Tabus. Refining The Bounding Volumes for Lossless Compression of Voxelized Point Clouds Geometry. *2021 IEEE International Conference on Image Processing (ICIP)*. 2021, 3408–3412. DOI: `10.1109/ICIP42928.2021.9506767`.
Publication IV	E. C. Kaya and I. Tabus. Neural Network Modeling of Probabilities for Coding the Octree Representation of Point Clouds. *2021 IEEE 23rd International Workshop on Multimedia Signal Processing (MMSP)*. 2021, 1–6. DOI: `10.1109/MMSP53017.2021.9733658`.
Publication V	E. C. Kaya and I. Tabus. Lossless Compression of Point Cloud Sequences Using Sequence Optimized CNN Models. *IEEE Access* 10 (2022). DOI: `10.1109/ACCESS.2022.3197295`.

Author's contribution

This study is conducted under the supervision of Prof. Ioan Tabus at the Computing Sciences Unit of Tampere University. The author contributed to all stages of research and preparation of the enlisted publications. The author contributed to development of concepts, to the proposal of original algorithms and methodology, implementation of the proposed algorithms in C and Python environments, and to evaluation of the proposed methods.

Publication I	The author has contributed to the design and development of the LLFC lossless light field image coding scheme. The author implemented the coding scheme in Python environment. The author performed experiments to evaluate the performance of the proposed codec, and contributed to the writing of the manuscript.
Publication II	The author contributed to the design and development of the LBV lossy point cloud compression scheme. The author was the main contributor to the implementation of the coding scheme in C environment. The author performed experiments to evaluate the performance of the proposed codec, and contributed to the writing of the manuscript.
Publication III	The author contributed to the design and development of the BVL lossless point cloud compression scheme. The author was the main contributor to the implementation of the proposed scheme in C environment. The author performed experiments to evaluate the performance of the proposed codec, and contributed to the writing of the manuscript.
Publication IV	The author has contributed to the design and development of the NNOC lossless point cloud coding scheme. The author was the main contributor to the implementation of the coding scheme in Python environment. The author performed experiments to evaluate the performance of the proposed codec, and contributed to the writing of the manuscript.

Publication V	The author has contributed to the design and development of SeqNOC lossless point cloud sequence compression scheme. The author was the main contributor to the implementation of the coding scheme in Python environment. The author performed experiments to evaluate the performance of the proposed codec, and was the main contributor to the preparation of the manuscript.

1 INTRODUCTION

In this chapter, we introduce the motivation behind this study and we present the objectives in Publications I to V, and finally we provide an outline of the structure of this thesis.

1.1 Motivation of the thesis

Immersive technologies aim to create virtual worlds for us to experience or to extend our perception of *reality*. Defining immersion is not an easy task and it relates to one's subjective senses of being surrounded which is certainly not limited to the visual experience [3]. Several concepts that are closely related to immersive technologies include augmented reality, virtual reality, mixed reality, holography, volumetric video, telepresence and digital twin applications. A few decades ago, such technologies were merely considered as sci-fi visions. In recent years however, immersive technologies started to attract a wide audience and for some of us they have already become a part of the daily life through consumer products such as VR glasses.

In immersive technologies, transmission of 3D geometry of objects in real-time is essentially important. Since a few decades, 3D geometry of objects is represented with triangular meshes [48, 63, 73] whereas, recently, point clouds also proved to be an efficient means of representing the 3D geometry. Point clouds are useful for augmented and virtual reality applications such as telepresence [47, 88], volumetric video [75], self-driving vehicles [1, 38], 3D printing [59] and digital preservation of cultural heritage assets [64].

1.2 Objectives of the thesis

The main objective of this thesis is to develop compression algorithms for point cloud geometry and light field images which are two types of immersive data. Our approach leverages the well established image compression paradigms whereas it also aims to build upon the more recently introduced learning-based methods which prove to be successful in several tasks related to computer science and computer vision. In addition to the aim of contributing to academic knowledge, we aim to develop software complying with practical considerations. In the following, we summarize our objectives in each publication. The publications are numbered in chronological order according to respective submission dates.

- *Publication I.* Redundancy in light field images can be efficiently represented by the usage of disparity maps. For compression purposes, it is valuable to describe the scene geometry through various disparity maps corresponding to different views of the scene. In Publication I, our first objective is to develop a disparity estimation method for the corner views of a light field image by building upon an existing learning-based scheme which was proposed recently. Secondly, we build a lossless light field image coder that makes use of disparity map estimation at five view points.
- *Publication II.* In certain real-life usage cases of point clouds such as telepresence, geometry of a human's body is represented using point clouds. For such point clouds, most of the points can be represented in two depth maps obtained by projecting the point cloud in opposite directions on a certain axis. In Publication II we develop a method which utilizes projections to generate depth maps. The points represented in the depth maps are used to define what we call a *bounding volume* which provides us anchors to represent encode the remaining points in an efficient manner. The proposed method builds upon the well established depth map and image compression techniques to obtain perceptually plausible lossy reconstructions having competitive rate-distortion performance and running times.
- *Publication III.* In this publication, we follow the *bounding volume* approach which was proposed in Publication II and develop a lossless point cloud compression method dubbed BVL. BVL treats a point cloud as a combination of

what we call *shells* where each *shell* is a point cloud that is efficiently encoded using depth map compression combined with a novel context arithmetic coding scheme that iteratively expands into the neighboring locations of the depth map points. Achieving lossless compression is the main objective which requires the proper treatment of all possible arrangements of points in a voxel grid, unlike the lossy compression scenario in Publication II.

- *Publication IV.* Computer science community witnessed the strong success of machine learning in most topics of research in the recent decade and data compression was no exception. On the other hand, octree representation has been extensively used for point cloud compression since around fifteen years. In Publication IV, our objective was to develop a machine learning scheme dubbed NNOC with a novel octree-based context coding approach which yielded better compression rates than existing coding schemes.
- *Publication V.* The NNOC scheme proposed in Publication IV was comparing favorably with the other learning-based approaches whereas it was aiming at single point cloud compression although in many usage scenarios one is interested in compression of dynamic (i.e. sequences of) point clouds. Moreover it had its own limitations arising from its learning-based nature. In Publication V we aimed to develop a lossless sequence (of point clouds) compression scheme which incorporates the well studied neural network optimization techniques as part of the encoding stage without necessiating a long preliminary training phase which is typical in learning-based approaches. The results are competitive in both bitrates and encoding/decoding times.

1.3 Outline of the thesis

This dissertation is structured in the following way. In Chapter 2, background information on the compression methodologies that are referred to in the later chapters of the dissertation are conveyed. In Chapter 3, contributions to depth estimation and lossless light field compression in Publication I are presented. In Chapter 4, contributions to lossy and lossless compression of point cloud geometry in Publications II, III, IV and V are presented. Finally, In Chapter 5, the conclusions and summary of the dissertation are presented.

2 PRELIMINARIES ON COMPRESSION METHODOLOGIES

This chapter provides the relevant technical background for the subsequent chapters which are devoted to describe the main contributions in the thesis. Section 2.1, describes the relevant concepts that are common to all fields of data compression. Section 2.2, describes the octree representation of point clouds and reviews the literature related to point cloud compression.

2.1 General Data Compression Concepts

In this section, we shall overview some of the basic data compression concepts which are not specific to any type of data. These concepts are relevant for understanding the methods that we describe in the contribution chapters as well.

2.1.1 Dictionary-Based Methods

In computer science terminology, dictionary is an unordered set of unique key-value pairs. In dictionary-based compression, the idea is to substitute pieces of data with keys (codewords) that take up less storage space than the data itself. A dictionary is called static, semi-adaptive or adaptive based on whether it changes while the input is being processed or not. A static dictionary can be preferable when there is prior knowledge about the data. For instance, if the compressor is going to be used to compress only texts written in English, a meaningful strategy would be to assign the shortest codewords to the most frequent words in English [92]. If there is no prior information about the data, i.e. if we want to have a generic compression tool, adaptive dictionary is the most preferable choice. In this approach, the encoder and the decoder should be able to construct the same dictionary on the fly, i.e. while processing the data.

Most of the adaptive dictionary-based methods that are currently in widespread use are variants of two methods called LZ77 and LZ78 which are also referred to as the Lempel-Ziv Algorithms [99, 100].

2.1.2 Entropy Coding

Data Compression has its theoretical roots in information theory. Information theory is a branch of probability theory that mainly originates from the seminal works of C.E. Shannon [78]. In information theory, entropy is a measure of how much we are uncertain about the possible outcomes of a probabilistic data source. The higher is the entropy of the source, the lower are our chances to compress the generated data. Thus, entropy establishes an upper limit to how much compression can be achieved with a certain probabilistic model of the data source. A group of methods in lossless compression which are called entropy coding methods aim to get as close as possible to the limit dictated by the entropy.

One notable example of entropy coding is Huffman coding [32], where one constructs a prefix-free code having optimal codeword lengths for each symbol based on its probability of occurrence. The optimal value for the codeword length of a symbol is calculated from its probability p of occurence as $-log(p)$. This optimal value is, in general, a real number whereas length of a codeword can only take integer values.

Arithmetic coding [72] is the most efficient entropy coding approach, which instead of assigning codewords to individual symbols, partitions the 0 to 1 probability interval, corresponding to all possible outcomes, into probability intervals assigned to sequences of symbols. Thus, instead of having a codeword that uniquely represents a symbol, the encoding of the entire signal results in a single floating point number between 0 and 1 that corresponds to a probability interval uniquely assigned to one of all the possible sequences (signals) that can be generated by the data source. The more probable the sequence is to be generated by the data source, the larger the probability interval assigned to it and the less number of bits are needed in the final floating point number written to the bitstream to distinguish it from other intervals corresponding to other possible sequences.

We cannot emphasize enough that entropy tells something about a certain probabilistic model that we devise and use to explain the data, rather than the data itself. Entropy coding can yield the best outcome for that particular model which on the other hand might be inferior to many other possible models that could have been used instead. With the discovery of entropy coding, the real challenge in modern data compression has became to design powerful probabilistic models which yield low entropy.

2.1.2.1 Context Coding

An important advantage of arithmetic coding is that it is not restricted to fixed probabilities as in Huffman coding. In other words, the probability of encountering a symbol can be defined as a function of the neighboring symbols rather than a constant. This enables us to design more realistic probabilistic models. In the most general sense, context is the surroundings of something. One can define context for various types of data in various ways. For a text, context of a word can be a function of the words that occur in the same sentence or paragraph. Similarly, in a raster image, neighboring pixels can serve as a context for each other.

In arithmetic coding, as in many other coding frameworks, the bitstream is traversed (written and read) by the encoder and the decoder in the same order. When the encoder and the decoder are at the same part of the bitstream, the already encoded parts of the data are identical to the already decoded parts of the data. These already encoded/decoded parts are past inputs for the encoder and they are available to both the encoder and the decoder. On the other hand, the not yet encoded/decoded parts of the data are not available to the decoder and they are the future inputs for the encoder. A context that is to be utilized in arithmetic coding has to be *causal* in the sense that it is defined using the past inputs of the encoder only (not the future inputs, since they are not available to the decoder).

In context coding, the goal is to estimate the conditional probability $P(X|C)$ of occurrence of a symbol X given its context C. In many classical approaches [2, 71, 82, 94], the estimation of conditional probabilities is achieved by collecting statistics from the input data by the encoder and the decoder in synchrony. This adaptive strategy has the disadvantage that, at the beginning of encoding/decoding, the estimates are based on a small number of observations hence they might be inaccurate. Moreover, collection of statistics requires keeping of frequency tables containing entries for each possible context. This requirement puts a limit to the maximum number of possible contexts due to memory limitations. If the size of the context window (template) is set to be very large, number of possible contexts also becomes large and the probability of observing any of the contexts decreases. As a result of this, the estimations based on the collected statistics become inaccurate. This is referred to as the context dilution problem. One possible solution to the context dilution problem is to apply context quantization [7, 17, 89, 93] where similar contexts are classified into a set of classes.

2.1.2.2 Run-Length Encoding

In certain cases, there are prior information available about the data which can be used to improve the compression performance significantly. Run-Length Encoding is a good option when the data is a string which is known to contain large blocks of repeated characters. In that approach, the string is first expressed as a series of integers each representing a character and the number of times it repeats. The integers can then be efficiently compressed using an entropy coding scheme such as Golomb-Rice Coding.

2.1.3 Performance Evaluation for Lossy and Lossless Compression

Lossless compression methods perfectly reconstruct the original input data. In lossless compression, the goal is simply to reduce the bitrate and one can express the compression performance with a single number which is generally referred to as bitrate. When the compression performance of the lossless case is not satisfactory, one can resort to lossy compression.

In lossy compression, performance evaluation is more complex. The goal is to reduce the bitrate while keeping the output reconstruction as *similar* to the input data as possible. The degree of similarity between the original data and the reconstruction is quantified in dB scale. dB expresses the ratio of the power of one signal to the power of another signal. In lossy compression, the intention is to measure the similarity of the output signal to the input (reference) signal. The degree of similarity between the two signals is quantified in dB scale where the ratio is taken between the power of the reference signal and the power of the error between the reference and the non-reference (output, reconstructed) signal. The error between two signals can be computed in many ways. One frequently used error function is the mean square error (MSE) which is defined for a reference signal x_r and a non-reference signal x_{nr} as

$$\mathrm{MSE}(x_r, x_{nr}) = \frac{1}{n_r} \sum_{i_r=1}^{n_r} (x_r(i_r) - x_{nr}(i_{nr}(i_r)))^2, \tag{2.1}$$

where n_r is the number of data points in the reference signal, i_r is an index associated with the reference signal and $i_{nr}(i_r)$ is an index associated with the non-reference signal. One needs to define the function that associate i_{nr} and i_r. For example, considering point cloud compression, the reference signal x_r can be chosen as the original (input) point cloud and x_{nr} can be the output of a lossy compression algorithm. If the metric for associating the points in the two point clouds is chosen as the euclidean distance $d(p^1, p^2)$ between points p^1 and p^2, $i_{nr}(i_r)$ would be

$$i_{nr}(i_r) = \arg\min_i d(x_r(i_r), x_{nr}(i)) \tag{2.2}$$

where

$$d(p^1, p^2) = \sqrt{\sum_{c \in (x,y,z)} (p_c^1 - p_c^2)^2} \tag{2.3}$$

and p_x, p_y, p_z are the x,y,z coordinates of a point p.

Considering that the error between the two signals is defined as $\mathrm{MSE}(x_r, x_{nr})$, the similarity between x_r and x_{nr} is computed as

$$\mathrm{PSNR}(x_r, x_{nr}) = 10 \cdot \log_{10}\left(\frac{X_{r,max}^2}{\mathrm{MSE}(x_r, x_{nr})}\right), \tag{2.4}$$

where $X_{r,max}$ is the maximum value the reference signal x_r can take. If x_r is a voxelized point cloud having resolution R bits per dimension, $X_{r,max}$ is $2^R - 1$. PSNR is one way to quantify how much a signal is *distorted*, has undergone an unwanted change. Note that, the value of the $PSNR(x_r, x_{nr})$ computed by (2.4) depends on which of the two signals is chosen to be the reference signal x_r. Therefore it is not symmetric,

$$\mathrm{PSNR}(x_r, x_{nr}) \neq \mathrm{PSNR}(x_{nr}, x_r). \tag{2.5}$$

For measuring the distortion for evaluating lossy compression, it is desirable to use a symmetric measure. A commonly used distortion measure for point clouds is the

D1 metric [83, 95] which is defined as

$$\mathrm{D1}(\mathscr{P}_i,\mathscr{P}_o) = \min(\mathrm{PSNR}(\mathscr{P}_i,\mathscr{P}_o),\mathrm{PSNR}(\mathscr{P}_o,\mathscr{P}_i)) \tag{2.6}$$

such that PSNR is computed as in (2.4) once by taking the original (input) point cloud $\mathscr{P}_i$ as the reference signal and once by taking the output point cloud (reconstruction) $\mathscr{P}_o$ as the reference signal. Note that, here, the word *metric* is not used in the strict mathematical sense but it is used only to be consistent with the common usage in the literature for D1. Plotting a distortion measure such as D1 versus the bitrate one obtains a so-called *rate-distortion curve*. Rate-distortion curve characterizes the compression performance of a compression algorithm in an objective manner.

2.2 Point Cloud Representation and Compression

A point cloud is an unordered set of three dimensional points possibly having attributes such as color and surface normals. The coordinates of the points can in general take any real value. These real values may correspond to actual physical distances if the point cloud is obtained through a physical measurement process such as LIDAR. On the other hand, a point cloud may also represent a purely virtual entity such as a 3D model created by an artist. In any case, in this unconstrained form, where the coordinates can take any real value, point clouds are not computer-friendly. Therefore, a process called *voxelization* is employed, which maps the coordinates to integers in the $[0, 2^r)$ interval where r is an integer specifying the resolution of the output *voxelized* point cloud. Depending on how densely the 3D space is initially populated by the points, this mapping may result in multiple points being mapped to the same voxel. Since more than one points having the same coordinates are redundant, it is customary to remove the duplicate points after voxelization. It is possible to voxelize a point cloud without generating any duplicate points by selecting the resolution sufficiently high.

Point cloud compression can be divided into subtopics as geometry compression and attribute compression according to which part of the data is being compressed. Geometry compression deals with compression of the 3D coordinates only, whereas attribute compression deals with attributes of points such as their normals and colors. According to whether the data is a single point cloud or consisting of multiple frames (a sequence of PCs), it can be called static or dynamic point cloud compression. In this thesis, we deal with both static (single frame) and dynamic (multi frame) geometry compression.

2.2.1 Octree Representation

Octree or 8-ary tree is a rooted tree where each node has at most 8 children. A voxelized point cloud can be efficiently represented with an octree having depth equal to the bits-per-dimension resolution of the point cloud [46].

Suppose the point cloud is fully contained in a $2^r \times 2^r \times 2^r$ voxel grid where r is the bits-per-dimension resolution of the point cloud. By downsampling the point cloud by 2, as $(x,y,z) \rightarrow (\lfloor x/2 \rfloor, \lfloor y/2 \rfloor, \lfloor z/2 \rfloor)$, one obtains the one step lower resolution version of the point cloud. Here we assumed that $x,y,z \in \{0,\ldots,2^r-1\}$. This coarser point cloud lives in a $2^{r-1} \times 2^{r-1} \times 2^{r-1}$ voxel grid, hence its resolution is $r-1$. Downsampling can be applied further until obtaining a $1 \times 1 \times 1$ grid thus a single voxel.

One voxel being occupied in resolution $\hat{r}-1$, where $0 < \hat{r} <= r$, corresponds to 8 possibly occupied voxels in resolution $\hat{r}$. This relationship is represented in the octree as an internal node having 8 children in the one step higher depth level. Every node in a depth level $\hat{r}$ of the tree corresponds to a voxel in the voxel grid with resolution $\hat{r}$, where the binary value assigned to the node denotes the occupancy of the voxel. For convenience, we shall refer to the nodes having value 0 and 1 as 0-nodes and 1-nodes, respectively. The 0-nodes which represent the unoccupied voxels do not have any children nodes because they are already known to be 0-nodes, as well. An example octree and corresponding voxel grids are depicted in Fig. 2.1.

Octree representation is frequently used in point cloud compression [15, 18, 19, 20, 35, 36, 68]. The simplest lossless point cloud compression algorithm based on octree representation can be devised in the following manner. The entire octree constructed for a point cloud can be represented in a bitstream consisting of octets (8 bit portions) where each octet corresponds to 8 children of a 1-node. The octree bitstream has a length $CL_{oct} = N_1 \times 8$ where N_1 is the total number of 1-nodes with children.

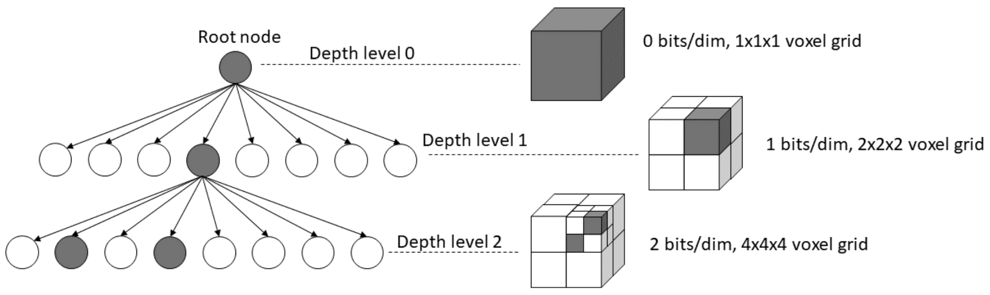

Root node
Depth level 0
Depth level 1
Depth level 2
0 bits/dim, 1x1x1 voxel grid
1 bits/dim, 2x2x2 voxel grid
2 bits/dim, 4x4x4 voxel grid

Figure 2.1 Octree Representation. Occupied voxels and the nodes corresponding to them (1-nodes) are shown in gray.

2.2.2 An Initial Look at the Benchmark Datasets

Several benchmark datasets are proposed to assess and compare the performances of different methods as part of the standardization processes conducted by MPEG and JPEG such as 8i Voxelized Full Bodies [14] and Microsoft Voxelized Upper Bodies (MVUB) [44] datasets. In Table 2.1, a list of 10 point clouds from benchmark datasets are provided. In Fig. 2.2, the numbers of points obtained at lower resolutions when each of these point clouds are downsampled are plotted in log2. From Fig. 2.2, it is observed that, at resolutions between 2 to 10, the curves are close to linear suggesting that the number of points are increasing exponentially in that interval. On the other hand, at resolutions higher than 10 bits/dimension, the number of points increase at a much slower pace, suggesting that the higher resolution point clouds are much more sparse.

Applying the basic octree-based compression scheme described in Section 2.2.1, to benchmark point clouds, we obtain $bpov_{oct}$'s in Table 2.2. In order to exemplify the efficiency of the octree representation, note, for instance, redandblack has 10 bits/dimension, thus without any compression applied, it can be stored by consuming $3 \times 10 = 30$ bits for each point, in other words, the point cloud has a bitrate of 30 bits-per-occupied-voxel [bpov] in its raw form. Octree bitrate obtained for this point cloud (according to Table 2.2) is around 3 bpov, hence, by simply using the octree representation one is able to compress this particular point cloud roughly 10 times already.

An octet is an 8-bits piece of information so it can take $2^8 = 256$ different values. In an octree, since the all-zero octet cannot appear, there are 255 possible val-

Table 2.1 Properties of Several Benchmark Point Clouds

Point Cloud	Dataset	Resolution	Approx. Nr. of Points
Phil	MVUB	10	1.66M
Ricardo	MVUB	10	0.96M
Sarah	MVUB	10	1.36M
Loot	8i	10	0.81M
Redandblack	8i	10	0.76M
Longdress	8i	10	0.86M
House-without-roof-00057	Cat 1-B	12	4.85M
Shiva-00035	Cat 1-B	12	1.01M
Facade-00015	Cat 1-A	14	8.91M
Stanford-Area-2	Cat 1-C	16	47.06M

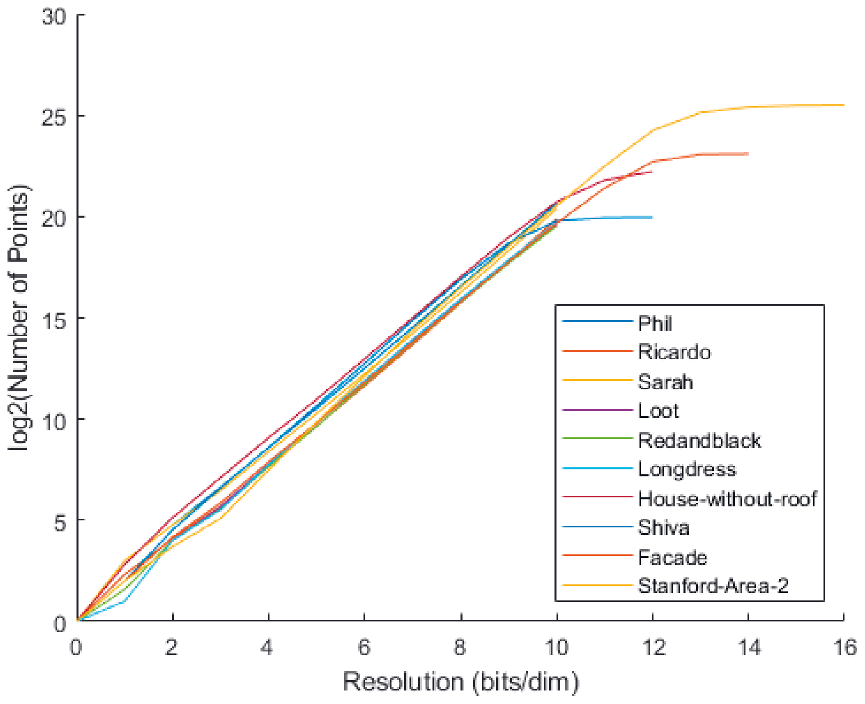


Figure 2.2 log2(Number of Points) vs. Resolution for the point clouds in Table 2.1

Table 2.2 Bitrates (bpov) obtained for benchmark point clouds, R: Resolution, N_p: Number of Points

Point Cloud	R	N_p	$bpov_{oct}$	$bpov_H$
Phil	10	1.66M	2.5417	2.0086
Ricardo	10	0.96M	2.4949	1.9270
Sarah	10	1.36M	2.5260	1.9641
Loot	10	0.81M	2.9828	2.3357
Redandblack	10	0.76M	3.0056	2.4040
Longdress	10	0.86M	2.9899	2.3336
House-without-roof	12	4.85M	9.9892	6.3488
Shiva-00035	12	1.01M	19.6640	10.0627
Facade-00015	14	8.91M	17.5234	9.2316
Stanford-Area-2	16	47.06M	26.3682	12.7442

ues. Thus, it can be formulated as a source having an alphabet with 255 symbols. The probability of occurence of each of these symbols can be estimated in the simplest way by counting their numbers of occurences and dividing these with the total number of octets. Note that, in this estimation, we do not make use of any possible dependencies among the neighboring octets hence they are independent probabilities. Using these independent and fixed probabilities, one can estimate an entropy H_r for each resolution level r. H_r's estimated with independent probabilities establish a lower bound for the average codelength that can be obtained with an entropy coding method that can make use of these fixed probabilities such as Huffman Coding. Using H_r's, we estimate a theoretical $bpov_H$ for encoding a point cloud with resolution R having N_p points as,

$$bpov_H = \frac{1}{N_p} \sum_{r=1}^{R} H_r * N_{oct,r} \tag{2.7}$$

where, $N_{oct,r}$ is the number of octets at depth level r in the octree representation of the point cloud. $bpov_H$'s obtained for the selected benchmark point clouds are presented in Table 2.2.

The numbers of occurences of 255 unique octets at the 4 highest depth levels for 3 different point clouds (Phil, Loot, Stanford-Area-2) are presented as histograms in

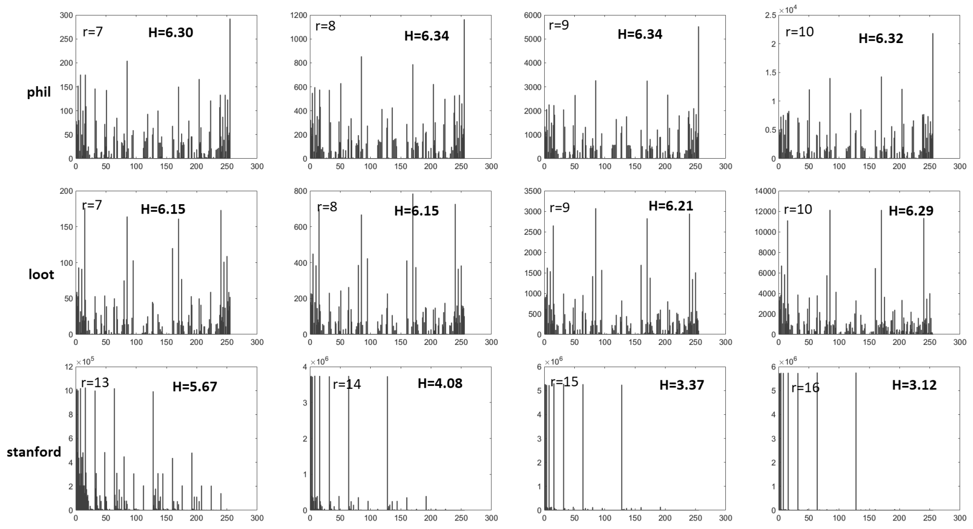


Figure 2.3 Number of occurence of octets at 4 highest resolutions (r). Each row corresponds to one point cloud. The rightmost column corresponds to the octets at the final resolution of the respective point cloud (10 for phil and loot and 16 for stanford).

Fig. 2.3 where the estimated entropies (H) are also given for each of the histograms. From Fig. 2.3, it is observed that Stanford-Area-2 is much more sparse when compared to the other two point clouds. Moreover, for stanford, only a few octet configurations (out of 255 possible configurations) are encountered at the highest depth levels of the octree. As a result of this, the entropy estimated with independent probabilities is significantly lower than the entropies at the lower resolutions.

2.2.3 Literature on Point Cloud Geometry Compression

Point cloud compression (PCC) has been an active area of research in the recent years. Also, two standards have been developed, namely G-PCC [23], and V-PCC [33]. In a recent survey, Cao et al. [9] (2019) provide an extensive walkthrough over the existing methods until that time. A more recent comprehensive survey is in [10]. Quach et al. [65] provide a general overview of the topic with a focus on the learning-based methods. In this section we shall overview some of the most prominent works in the field.

Gumhold et al. [26] construct predictive trees through one dimensional traversal of the points, in which the nodes are associated with points in point cloud where the neighboring points correspond to nodes connected with a single edge. The encoding of the trees is achieved with arithmetic coding. Merry et al. [49] employ a similar predictive tree approach with better heuristics.

Although the definition of point cloud does not necessitate that the points are distributed in the 3D space as a 2D manifold, most point clouds which are of practical interest, possess such a manifold like structure. Therefore, it is feasible to decompose a point cloud into 2d patches and apply the well established 2d compression techniques on those patches. Many methods in the literature employ patch generation [58, 77, 96].

Octree representation, which is described in detail in Section 2.2.1, has been introduced in early 1980s [46] as a geometric modeling technique for arbitrary 3-D objects. It was utilized in a point cloud compression scheme for the first time in 2006 [74]. Several point cloud compression methods adopt octree representation [15, 18, 19, 20, 35, 36, 68]. In [35], exclusive disjunction operator (XOR) is employed for differential encoding of octrees corresponding to consecutive frames of a point cloud sequence. In another inter-frame octree-based scheme, de Queiroz et al. [69] define the context through distances of voxels to occupied voxels in a reference frame. Garcia and de Queiroz [20], apply arithmetic coding with context and Lempel-Ziv-Welch (LZW) algorithm [90] (a Lempel-Ziv [100] variant) on the octree representation. More recently, Garcia et al. [18] propose a multiple context octree coding method in which, a reference octree is used as a reference to encode the current octree and the resulting frequency histogram is viewed as a discrete 3D surface and is encoded using another octree.

As a notably different approach than the octree-based methods, Peixoto [62] proposes to recursively apply dyadic decomposition. In this approach, a so-called silhouette image of the input point cloud is obtained by sweeping the point cloud along a user-defined axis and this image is encoded in a context adaptive binary arithmetic coding scheme. After that, the point cloud is split into two where each of the resulting point clouds are processed using the silhouette image obtained from their parent point cloud as a 2d-mask telling which locations are known to be unoccupied. The procedure is recursively applied to each of the newly obtained children point clouds where the silhouette of the previously encoded children serve as a context for the currently encoded ones. In Silhouette 4D [61], the dyadic decomposition based approach in [62] was extended into dynamic point clouds (sequences of point clouds).

In [70], Ramalho et al. develop a context selection preprocessing stage for the Silhouette 4D coder. Focusing on a certain type of point cloud such as humanoid avatars, one might benefit from the appropriate usage of a specific representation such as the skeleton model representation of the human body for efficent motion estimation as done in [11].

In recent years, the majority of methods for several types of processing of point clouds involved machine learning [8, 13, 24, 41, 42, 43, 87, 97, 98]. In particular, learning-based schemes started to prove successful in compression of point cloud geometry [4, 25, 30, 37, 45, 50, 51, 55, 56, 67, 85]. In OctSqueeze [30], a tree-structured entropy model, consisting of several MLP stages, is employed to encode the octree representation of LIDAR point clouds. Quach et al. [66] propose a 3D convolutional autoencoder architecture for extracting latent representations for the input point cloud which is then quantized to yield competitive lossy compression results. Guarda et al. [25] propose the PCG-AE scheme which employs autoencoders to extract latent representations from 3D blocks of point clouds which are encoded with entropy coding. Milani [50] proposes a convolutional autoencoder scheme that applies the principles of distributed source coding to deep representations of voxelized point cloud geometry. In [51], the distributed source autoencoder approach in [50] is further developed and enhanced with an adversarial training strategy. In [91], rate-distortion optimization is achieved through an end-to-end learned analysis-synthesis transform pair combined with an adaptive decomposition stage. Wang et al. [86] employ a variational autoencoder scheme for lossy geometry compression. In VoxelContextNet [68] a deep neural network, consisting of several 3D convolutional stages followed by several fully connected stages, is employed to predict probability distributions from a local 3D "voxel context".

In VoxelDNN [55], 3D masked convolutions are employed to enforce causality in the context. The input of the deep convolutional neural network is a fixed size block of voxels and the network predicts occupancy probabilities for all of the voxels in the block. VoxelDNN achieves competitive lossless bitrates for MVUB [44] and 8i [14] datasets. More recently, a fast and less accurate version of VoxelDNN called MSVoxelDNN [57] is proposed. In MSVoxelDNN, voxels are grouped in a certain way and their occupancies are fed to the network in parallel such that some of the context information is sacrified for speed.

3 CONTRIBUTIONS TO LIGHT FIELD DISPARITY ESTIMATION AND LIGHT FIELD COMPRESSION

In this chapter, the contributions to light field processing are presented. The contributions in Publication I are two-folded. In Section 3.1, we focus on disparity estimation from light field images. In Section 3.2, we present the lossless light field compression method dubbed LLFC, which uses a set of reference views and corresponding estimated disparities to predict any view in a light field image.

Disparity estimation is a widely studied problem in computer vision with countless practical application cases. Disparity estimation can be performed on different types of data such as single images (monocular) [22], stereo images [6] or light field images [60]. Our mammalian brains perform depth estimation from the stereo image pair generated in our eyes involuntarily whenever our eyes are open in a sufficiently illuminated environment. Thanks to our built-in depth estimation algorithm, we perceive the surrounding objects as three dimensional entities. Whether it is human eyes or stereo cameras, depth estimation process is based on the disparity of objects, which is the location difference of the same object in the two images. The disparity of the object is related to its depth via camera parameters such as baseline distance (the distance between two cameras) and focal length of the camera. Therefore, for a setup with known camera parameters, depth estimation problem reduces to disparity estimation. For this reason, depth estimation and disparity estimation are the two terms that may be encountered in the literature when referring to the same problem.

Light field image compression has also been an active area of research in recent years [5, 12, 16, 28, 29, 39, 40, 80]. Light field images are considered to be highly redundant when compared to other types of visual data. The redundancy in light field images can be efficiently represented with the so called Epipolar Plane Images (EPI). For this reason, several light field compression algorithms adapt the EPI approach [31, 53, 54].

3.1 Disparity Estimation from Light Field

Disparity is the distance between the positions of corresponding pixels in two images of a scene which are obtained from different view angles. In the context of light fields, the term disparity may sound ambiguous since there are more than two views. Therefore, when speaking about disparity in light fields, one should also mention between which two views the disparity is being measured. Notable works related to light field disparity estimation in recent years include [34, 79, 84].

In this chapter, we build on a learning-based disparity estimation scheme for light fields called EPINET [79]. It is a supervised learning-based scheme in which the parameters of the model are tuned based on the error between the estimated disparity and the ground truth disparity provided in the training dataset. Therefore, the disparity output by the model is to be interpreted according to how the ground truth disparities are defined. As its name implies, EPINET employs EPI approach. EPINET [79] is trained on a synthetic dataset called 4D Light Field Benchmark (HCI) [27] which features ground truth disparities. For HCI data, the disparities provided for one view are the differences with respect to the closest right horizontal neighbor view. HCI dataset features a very idealized 4D light field representation unlike the actual light fields produced by light field cameras such as Lytro cameras [21]. The main advantage of using a synthetic dataset is the availability of very accurate ground truth disparities. On the other hand, it should be noted that the actual light field images are quite different than the HCI images in terms of the positioning of the viewpoints with respect to each other and in terms of the visual qualities.

EPINET is summarized in Fig. 3.1. It is a deep multistream fully convolutional network. The first stage of the network consists of structurally identical convolutional blocks devoted to each of the input stacks. In the first stage, there are six 2×2 convolutional layers each with 70 filters for each of the stacks. Outputs of the 1st stage are concatenated to yield a stack of convolutional features with 280 channels. The 2nd stage consists of 15 convolutional layers (2×2) each with 280 filters and a 2×2 convolutional layer with a single channel output at the end.

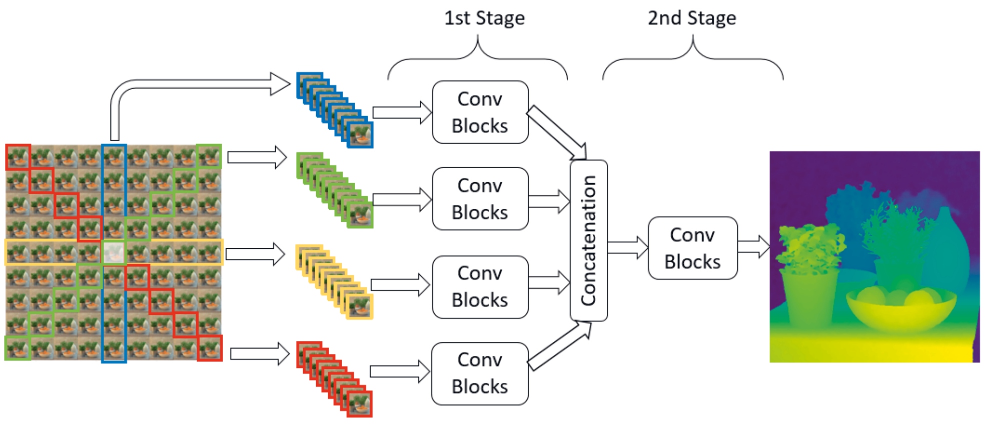


Figure 3.1 EPINET [79] method for disparity estimation from light fields. Disparity map that corresponds to the center (highlighted) view is generated with a fully convolutional neural network that operates on four stacks of input images collected from vertical, horizontal and diagonal directions.

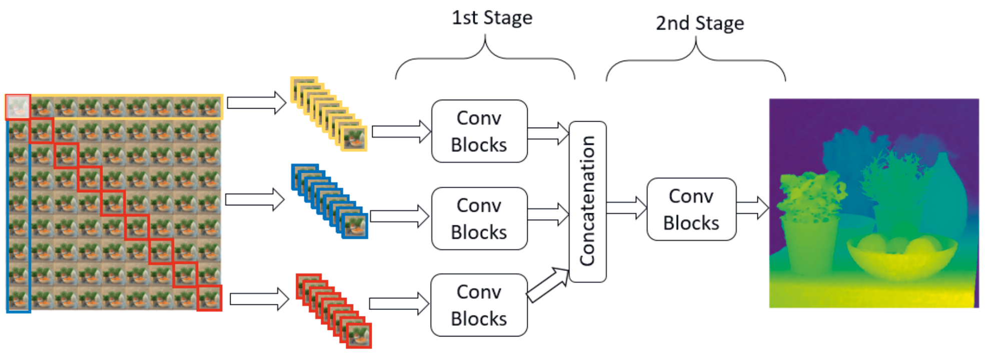


Figure 3.2 CEPINET is our variant of EPINET [79] that estimates the disparity for the upper-left corner view of the light field. The original network architecture is modified such that there are 3 input stacks.

Since the EPINET structure is not directly applicable to the corner reference views, in Publication I, we propose the corner variant CEPINET that operates on three stacks of input as shown on Fig. 3.2. The structure of CEPINET differs in that it has 3 input streams instead of 4 and since after concatenation there are 210 feature maps, the number of filters in the 2nd stage is 210 (instead of 280). In Fig. 3.2, the disparity estimation is exemplified with the upper-left corner view whereas one is able use the same network (having the same weights) to estimate the disparity for all four corner views by applying the appropriate rotations.

3.2 Lossless Light Field Coding Using Estimated Disparities

In Publication I, we propose a lossless light field image compression method called LLFC. The key ingredient in LLFC is the learning-based disparity estimation scheme that estimates disparities for corner and center reference angular views using EPINET [79] and CEPINET models described in the previous section. The estimated disparities and the corresponding reference view color images are losslessly compressed and transmitted. Using the corresponding disparity map, one can warp a reference view to any target view. All of the non-reference (target) views are reconstructed by performing a prediction using the warped reference view color images.

An overview of LLFC encoding and decoding schemes is presented in Fig. 3.3 and 3.4, respectively. At the encoder side, referring to Fig. 3.3, five reference views are selected from the input light field; four of them being the corner views and one being the center view. Center view disparity map is estimated using EPINET and corner view disparities are estimated using CEPINET. The five reference disparity maps and the corresponding five reference color images are encoded with JPEG2000. Next, the operations that are shown inside the red box in Fig. 3.3 are repeated for each target view that we want to encode. Each of the reference view color images are warped to the target view resulting in warped reference view color images $I_{W,r}$ where r is the reference index. Out of the five reference disparity maps, the one that is closest (in euclidian distance over the angular plane) to the target view is picked and it is warped to the target view. This warped reference disparity is quantized and divided into connected components. Each connected component is assigned a best reference label which are marked in a Best Reference Labels Image I_B. Let CC_i denote the group of pixels that belong to the i'th connected component and let $MSE_{CC_i}(I_{W,r}, I_T)$ denote the mean square error computed over CC_i between the target view ground truth color image I_T and the warped reference view color image $I_{W,r}$ obtained from the r'th reference view. Best reference label of all pixels in CC_i is the reference index r that minimizes the aforementioned MSE, hence,

$$I_B(x,y) = \arg\min_r (MSE_{CC_i}(I_{W,r}, I_T)), \; \forall (x,y) \in CC_i \tag{3.1}$$

where $I_B(x,y)$ can take 5 different values. I_B can be compressed by either a generic lossless image compression method such as JPEG2000 or a more suited compression

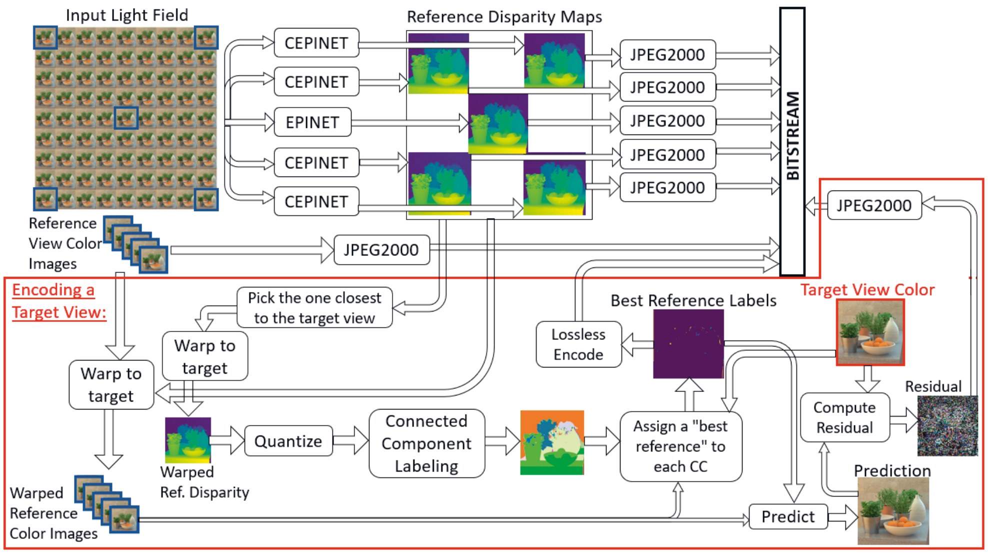


Figure 3.3 Block diagram of the proposed LLFC encoder. Four corners and center views of the light field are selected as reference views and the corresponding color images are transmitted. For each reference view, a disparity map is estimated and transmitted. A non-reference view which is denoted target view is predicted using the reference views and estimated disparities. Residuals to this prediction are transmitted to ensure lossless compression.

method such as CERV [81]. The results with CERV are slightly better when compared to the JPEG2000 case. Here we present experimental results obtained by compressing I_B with CERV. In Publication I, results for compressing I_B with JPEG2000 are also available. A prediction for the target view color is constructed by combining the parts from the warped reference images $I_{W,r}$ according to I_B resulting in the prediction image $\hat{I}_T$ which can be formally expressed as,

$$\hat{I}_T(x,y) = I_{W,I_B(x,y)}(x,y). \tag{3.2}$$

Since $\hat{I}_T$ is a lossy reconstruction of I_T, we also need to compress the residual for all target views in order to achieve lossless compression.

The decoding scheme is much simpler as seen from Fig. 3.4. Initially, the reference color images and disparity maps are decoded. For decoding a target view, reference color images are warped to obtain $I_{W,r}$'s which are used in conjunction with the I_B of the target to perform the prediction expressed in (3.2). Finally, residual of the target is added.

In Fig. 3.5 are shown the warped reference views, corresponding best reference

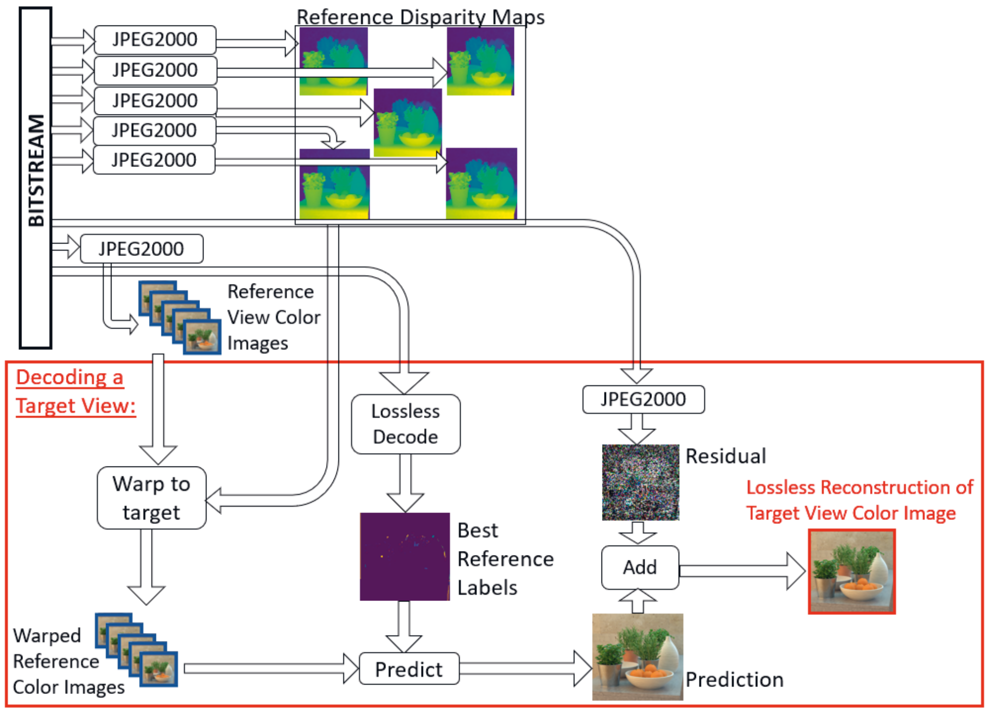


Figure 3.4 Block diagram of the proposed LLFC decoder. Prediction of a target view is performed using five reference views and corresponding disparity maps. A target view is losslessly reconstructed using the corresponding residual.

labels, prediction and ground truth for a particular target view. Warped reference view color images contain empty regions (shown in black). The prediction step expressed in (3.2) combines the information available from all reference views such that eventually there are no empty regions in the predicted target view.

3.3 Experimental Results

In this section we provide an overview of the experimental results that we obtained. With the same training set, we train an EPINET and a CEPINET model. The training set consists of 12 samples from the default training split of HCI dataset [27]. The data augmentation employed in the original EPINET [79] scheme is performed also during CEPINET training. In Fig. 3.6, the disparity maps obtained with both the EPINET (first column) and CEPINET (remaining columns) are presented for a number of test samples for which the ground truth disparities are not available.

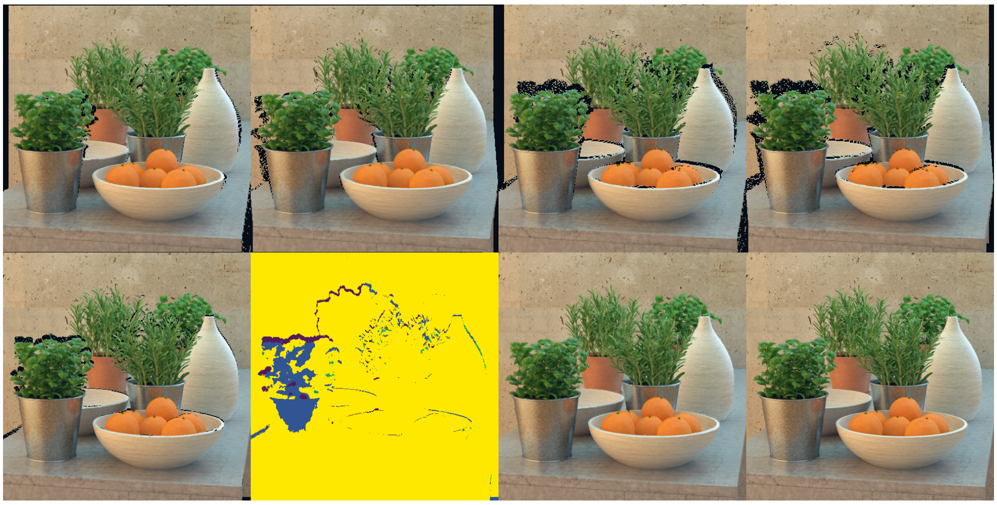


Figure 3.5 Top Row: Warped corner reference views(From left to right: Upper-left, upper-right, lower-left, lower-right corners.) Bottom Row, from left to right: Warped center reference, best reference labels, predicted target view, ground truth target view.

From Fig. 3.6, it is evident that our corner view disparity estimations are in accordance with the EPINET center view disparities.

Table 3.1 shows bitrates obtained with LLFC and JPEG2000 for a number of test samples for which the ground truth disparity maps for all five reference views are available. Bitrates obtained with LLFC are much better than the baseline JPEG2000 results. The last column contains the results for the case when the ground truth disparities are used instead of EPINET/CEPINET estimations. Hence, rather than an implementable result, the last column represents a lower bound for the bitrates that can be converged to by improving only the disparity estimation in LLFC scheme. The differences between the actual results (2nd column) and the last column demonstrates how much the deviation of our disparity estimations from the ground truth disparities effect the final compression results. In Table 3.2, results for the remaining test samples (for which ground truth disparities are not available) are presented.

The experimental results obtained with the proposed scheme look promising, yet it is worth emphasizing that these results are obtained with synthetic data for which there was the possibility to use quite accurate ground truth disparities. On the other hand, for real light fields, this sort of accurate ground truth disparities are not available and the networks trained on synthetic data are not expected to perform well on the real data. Therefore, the obtained results should be seen as a promising

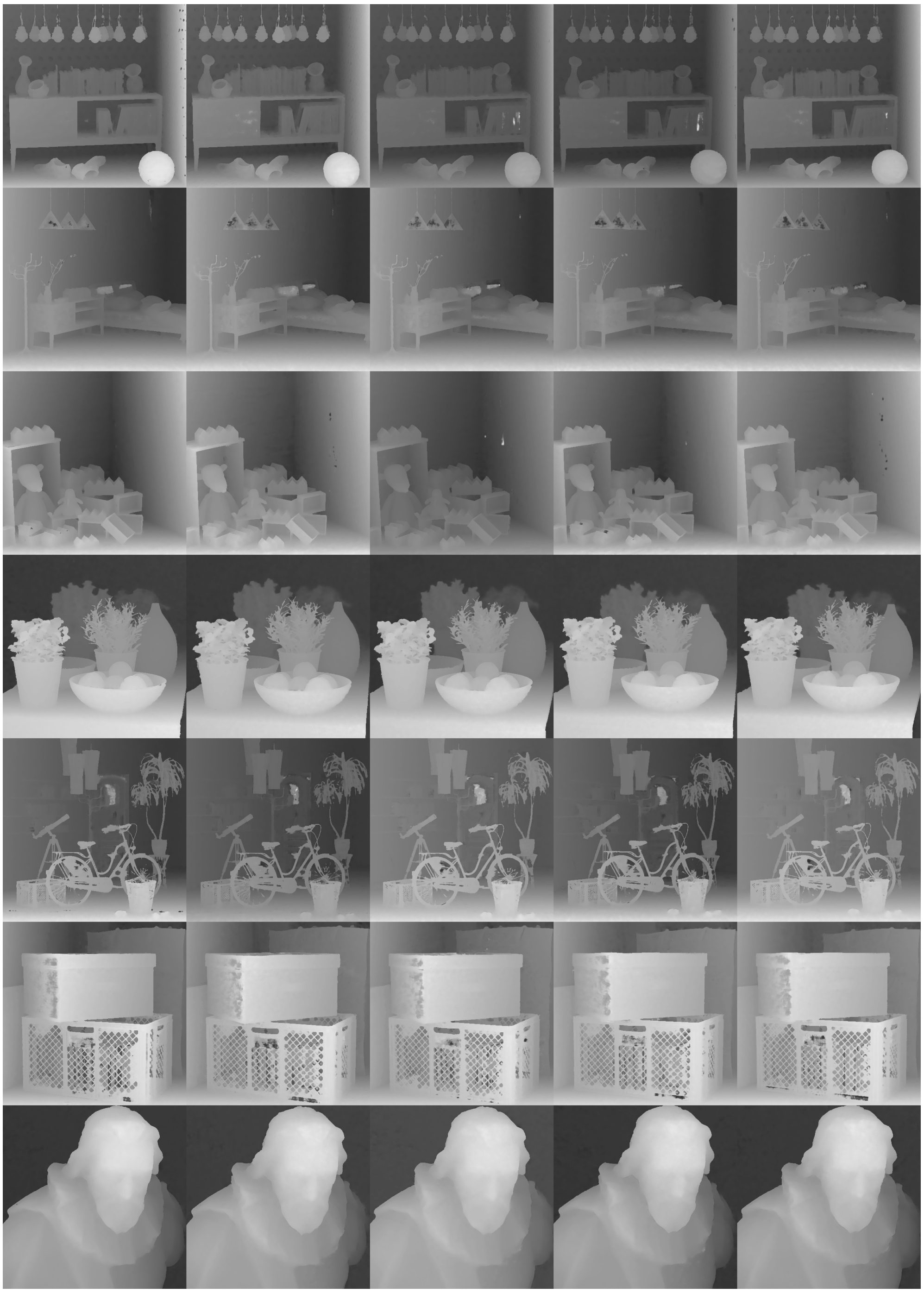


Figure 3.6 Each row corresponds to a light field, from top to bottom: sideboard, bedroom, dino, herbs, bicycle, boxes and cotton. First column: EPINET (Center View) output disparity map. Remaining columns: CEPINET (Upper-left, upper-right, lower-left, lower-right) output disparity maps.

Table 3.1 Bits-per-pixel results for test samples for which ground truth disparities are available

Sample	JPEG2000	LLFC	LLFC (with GT Disparities)
vinyl	7.38	4.30	**4.08**
kitchen	9.07	6.15	**5.78**
museum	10.97	6.92	**6.67**
greek	8.15	5.10	**4.94**

Table 3.2 Bits-per-pixel results for the test samples

Sample	JPEG2000	LLFC
dino	9.65	**5.79**
dots	24.16	**20.06**
bedroom	10.13	**6.90**
pyramids	19.88	**13.32**
stripes	3.44	**1.92**
bicycle	12.85	**8.65**
backgammon	16.62	**11.30**
origami	10.53	**6.83**
boxes	11.34	**7.95**
cotton	6.96	**3.21**
sideboard	13.93	**9.42**
herbs	11.93	**8.01**

initial step towards the actual goal of compressing real light field images.

4 CONTRIBUTIONS TO POINT CLOUD GEOMETRY COMPRESSION

In this chapter, we present our contributions to point cloud geometry compression in Publications II, III, IV and V. Section 4.1, discusses a lossy and a lossless method which are based on the concept of bounding volumes. In Section 4.2, we present a method based on octrees and context coding with neural network estimated probabilities.

4.1 Geometry Compression using Bounding Volumes

A bounding volume is an approximation for a 3D object that fully contains the object and possibly additional (empty) volume. It can be thought of as the three dimensional analog of a bounding box. In Publications II and III, a bounding volume is defined for a point cloud using two depth maps which are obtained by projecting the point cloud along two opposite directions. In this manner, a portion of the points are represented and encoded in the form of depth maps. The remaining points are known to be contained in the bounding volume defined by the two depth maps. In this section, we present the two different methods described in Publications II and III, to encode a point cloud with the bounding volume approach.

4.1.1 Lossy Compression using Bounding Volumes

In Publication II, a lossy point cloud geometry compression algorithm which we refer to as LBV (Lossy Bounding Volumes) is presented. An overview of LBV encoding scheme is given in Fig. 4.1. In this scheme, different bitrates are achieved by adjusting a single parameter q, which is the quantization step, where $q = 1$ is equivalent to no quantization taking place and corresponds to the highest bitrate and lowest

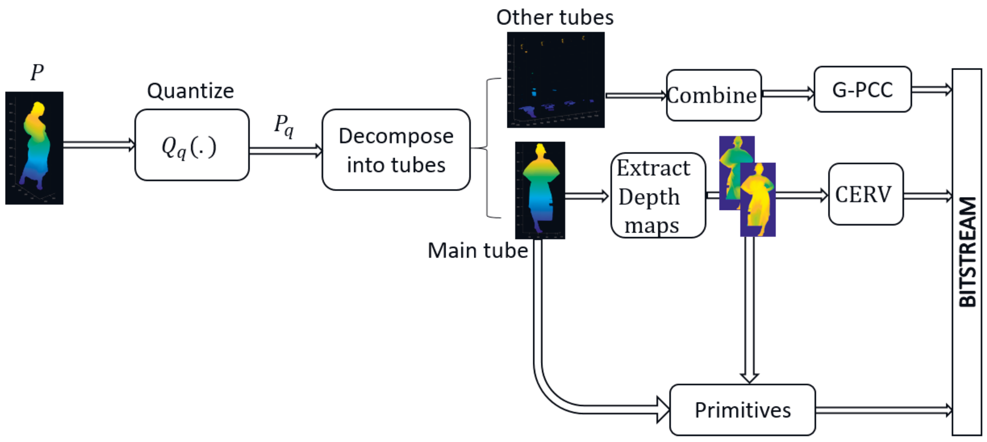


Figure 4.1 Overview of the proposed encoding scheme in the Lossy Bounding Volumes (LBV) algorithm.

distortion case. After quantization, the quantized point cloud $\mathscr{P}_q$ is decomposed into what we call *tubes*. We define *tube* as a point cloud such that when it is traversed along an axis, all of its cross section binary occupancy images contain at most one connected component. We call the axis, through which the tube is traversed, the sweeping axis. Before deciding on the sweeping axis, initially, a projection axis, which we shall denote z, is selected. The projection axis is selected from the three inherent axes of the point cloud as the one that captures the most number of projection points. Projection of a point cloud in two directions along one axis and the resulting front and back view depth maps are visualized in Fig. 4.2.

After deciding the projection axis z, one of the two remaining inherent axes of the point cloud which are orthogonal to the projection axis is selected as the sweeping axis, denoted y. According to the selected sweeping axis y, the point cloud is decomposed into what we call *tubes*. Decomposition of the input point cloud into tubes is visualized on Fig. 4.3. When decomposing the input point cloud into tubes, the points in consecutive cross sections are included in the same tube if their connected components (in the consecutive cross section binary images) have large overlapping regions. While each of the tubes are valid inputs for the subsequent algorithm stages, in our experiments, we compress only the main tube (the one with the most number of points) for all test point clouds and compress all of the remaining points with G-PCC.

The main tube is then projected along the projection axis in the opposite directions to obtain two depth maps which are then encoded by a depth map compression

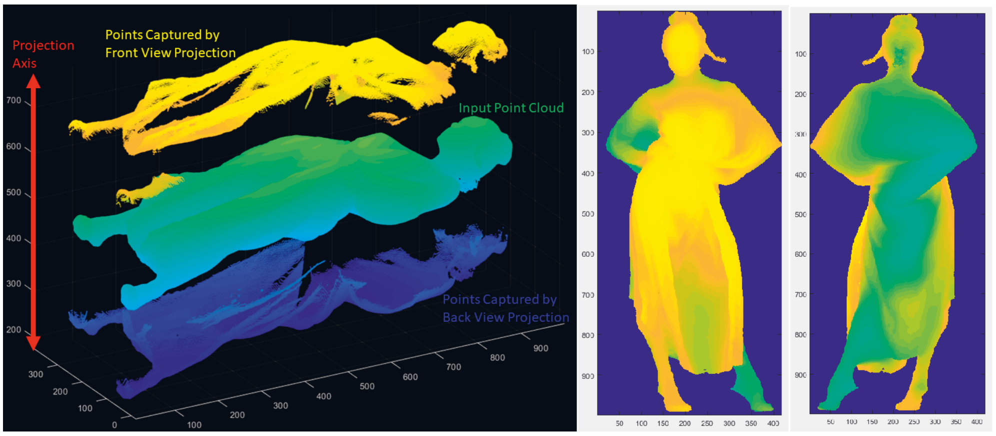

Projection Axis
Points Captured by Front View Projection
Input Point Cloud
Points Captured by Back View Projection

Figure 4.2 Projection of the input point cloud along the projection axis results in front (middle) and back (right) view depth maps.

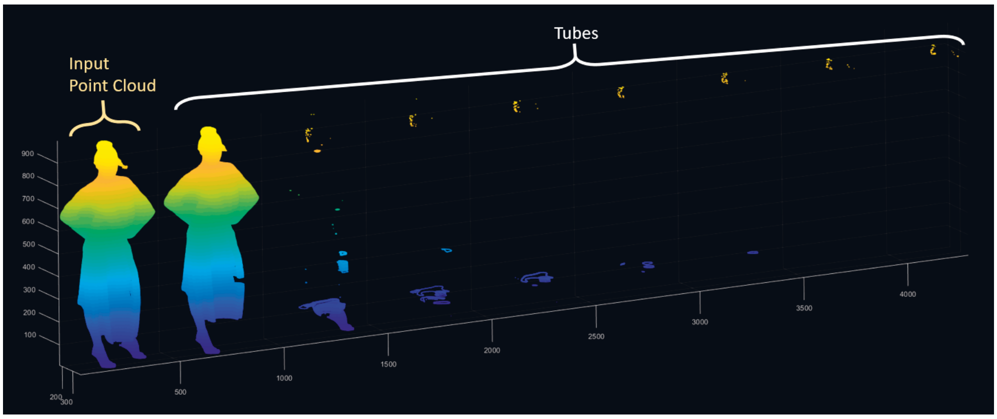


Figure 4.3 Initially, the input point cloud is decomposed into so-called tubes, which are point clouds such that when traversed in a certain axis called the sweeping axis, their 2D cross section binary occupancy images contain at most one connected component each.

algorithm called CERV [81]. In CERV, the depth (or disparity) map is divided into constant depth regions and the crack edges between the regions are encoded using optimally pruned context trees which are transmitted first. The constant values inside the regions are predicted and encoded by making use of the already transmitted neighboring region information as context. The remaining (not projected) points in the main tube are encoded in a number of stages which are called Primitives.

The main tube is traversed (swept) along the sweeping axis and encoding/decoding is performed section-by-section in several stages as explained in the following.

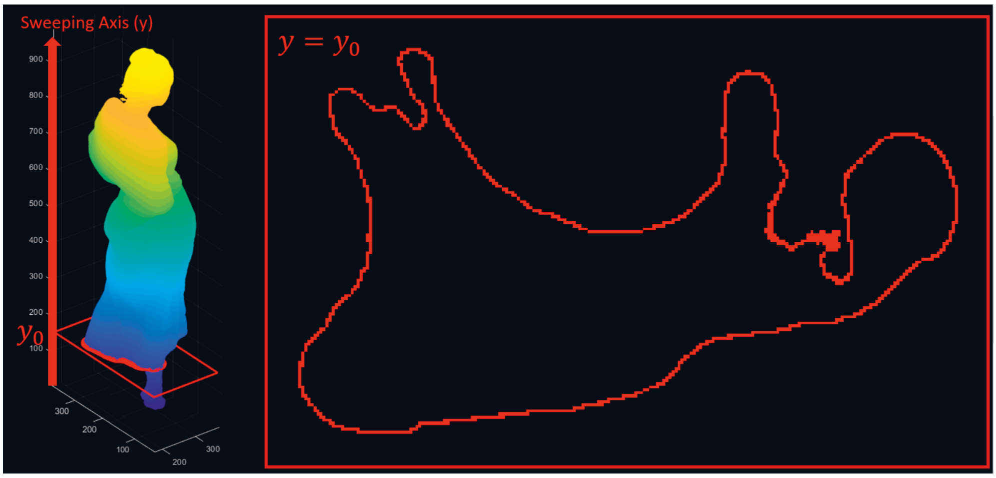


Figure 4.4 A two dimensional section of the main tube.

Suppose the ground truth for a two dimensional section of the main tube is as shown in Figure 4.4. The points in the two dimensional section of the tube that are projected to depth maps are shown in Fig. 4.5 in green. Projected points were already encoded/decoded as two depth maps. After decoding the depth maps, the decoder is able to reconstruct the points shown in green in Fig. 4.5. Furthermore, using the projected points, the encoder/decoder constructs a so-called feasible region in which all the remaining points of the current 2D section of the tube have to be confined. In other words, the feasible region is a two dimensional section of the bounding volume defined by the two depth maps. The feasible region obtained from the projected points is shown in Fig. 4.6 in pink.

In the next step which we refer to as Primitive I, the boundary of the feasible region is traced such that one obtains an ordered list of possibly occupied locations $\mathscr{L}_b$ that constitute the boundary of the feasible region. Note that the decoder is able to obtain the same $\mathscr{L}_b$ by using only the feasible region information. $\mathscr{L}_b$ contains 3 sorts of locations: Firstly, there are the locations of projected points which are already encoded as depth maps. No further action is required for them. Secondly, there are unoccupied locations (gray locations in Fig. 4.6) and finally, there are locations which are occupied but not yet encoded (blue points in Fig. 4.6). All of the not projected feasible boundary locations (blue and gray) are collected in a new list L_{b2}. The goal of Primitive I is to efficiently encode the not projected occupied locations (true points) in the boundary. The occupancy of feasible region boundary locations,

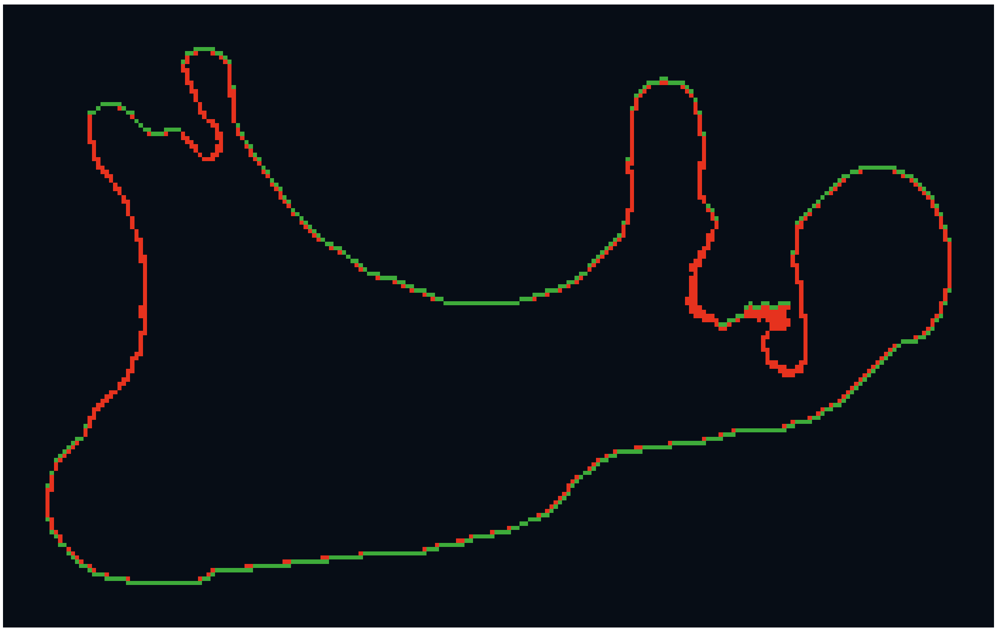


Figure 4.5 A two dimensional section of the main tube (same as in Fig. 4.4) where the projected points are shown in green.

excluding the projected (already encoded) points, is represented with a binary vector v_{P1} where the i'th element of v_{P1} denotes the occupancy status of the i'th location in $\mathscr{L}_{b2}$. v_{P1} can be efficiently encoded via run-length encoding or arithmetic coding.

After Primitive I, the current reconstruction consists of the green and the blue points in Fig. 4.6. The next stage is Primitive II. In Primitive II, the pieces of contours that we have as the current reconstruction are completed into one big closed contour consisting of true points residing at the outer surface of the tube. For the datasets that we experiment with, the overwhelming majority of the true points lie in such contours.

In Primitive II, all end points of the contours in the current reconstruction are called anchor points. Anchor points are denoted with red circles in Fig. 4.7. Primitive II encodes the points that bind the anchor points (shown in purple) using chain codes.

After Primitive II, we have an additional stage which is not mentioned in Publication II. Here we call this Primitive A. In Primitive A, the current reconstruction after Primitive II which is a closed contour, is traversed from the inside. For each cross (diagonally) connected consecutive true point pair, there is one location in the

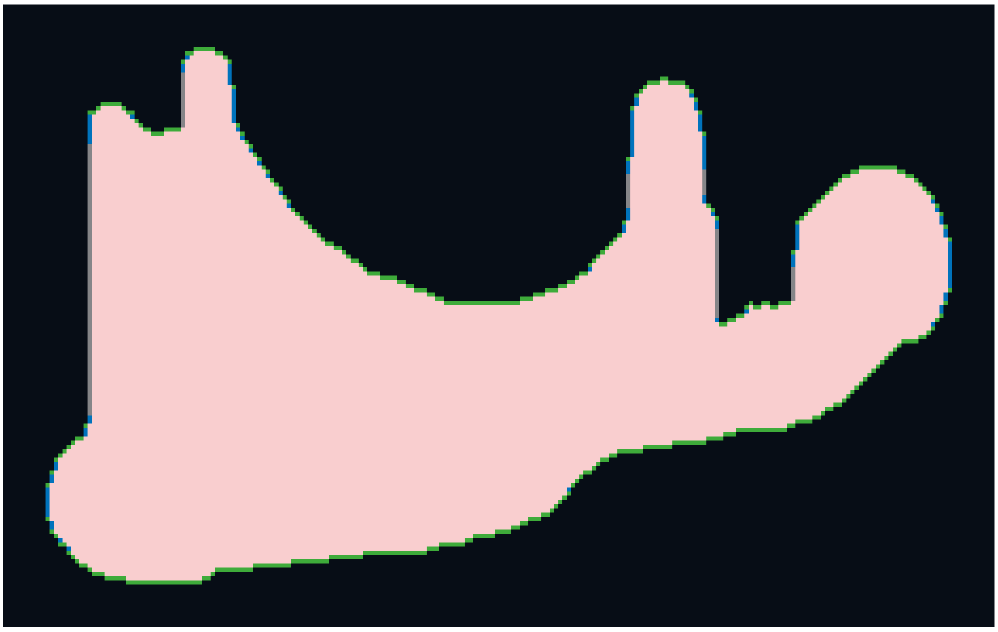


Figure 4.6 Green: Projected points. Blue: Points encoded by Primitive I. Gray: Unoccupied locations in feasible region boundary. Pink: Feasible region.

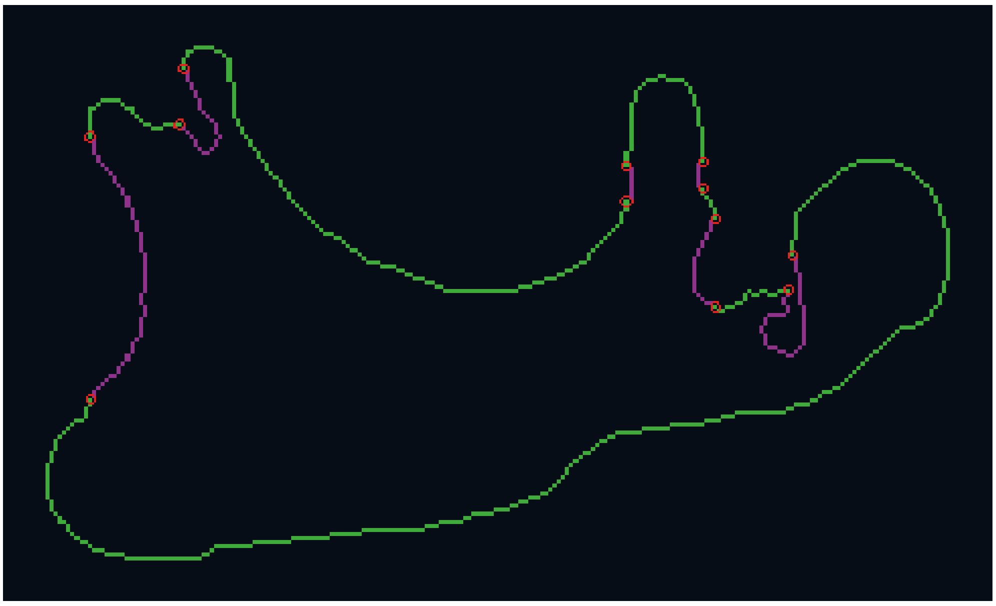


Figure 4.7 Green: Current Reconstruction after Primitive I. Red circles: Anchor points. Purple: Points encoded at Primitive II.

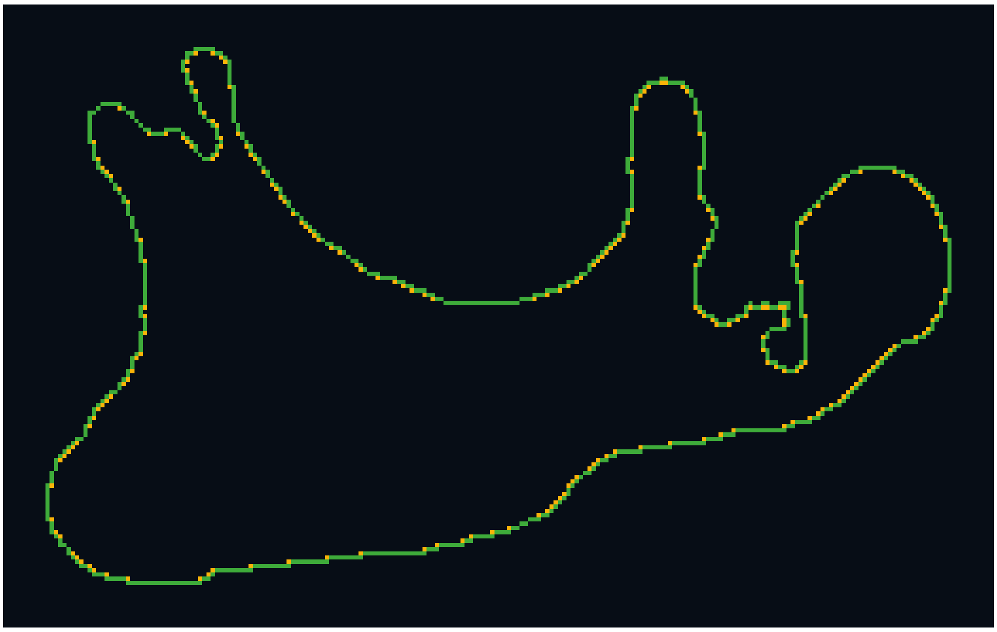


Figure 4.8 Green: Current reconstruction after Primitive II. Yellow: Points encoded by Primitive A.

feasible region that is 4-connected to both of the already encoded points. The occupancies of these locations are represented in a binary vector which is encoded using arithmetic coding. The points encoded at Primitive A are shown in Fig. 4.8.

The final stage of encoding in LBV, called Primitive III, involves not only the current section of the tube ($y = y_0$) but also the previously encoded/decoded section ($y = y_0 - 1$). In this stage, we make the assumption that in quite many cases, a point cloud is a water tight surface. Water tight assumption allows us to reconstruct groups of neighboring points efficiently. By overlaying the reconstructions from two sections y_0 and $y_0 - 1$, one obtains connected component regions. For each of these regions, we encode a single binary flag stating whether these regions are occupied or not at the currently being encoded section (y_0). Primitive III points are depicted at Fig.s 4.9 and 4.10. As an overview, the points reconstructed at different stages of LBV are visualized in Fig. 4.11.

For demonstrating the bitrate performance of different stages, we find it useful to plot bpov-vs-number of points after each stage and compare these with the bpovs obtained with the G-PCC codec. In Fig. 4.12, two such plots are shown for two different point clouds. The vertical axis denotes the bitrate. In the blue curves, each point corresponds to an instant after a certain stage of LBV (CERV, Primitive I,II,A

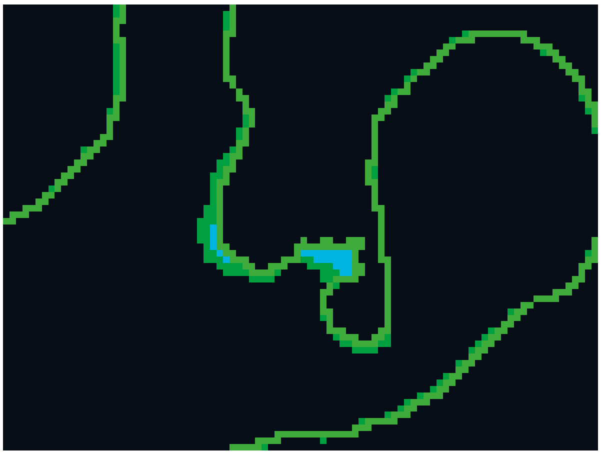


Figure 4.9 Light green: Current reconstruction at $y = y_0$ after Primitive A. Dark green: Some of the points reconstructed at the previous section ($y = y_0 - 1$). Blue: Points reconstructed at Primitive III.

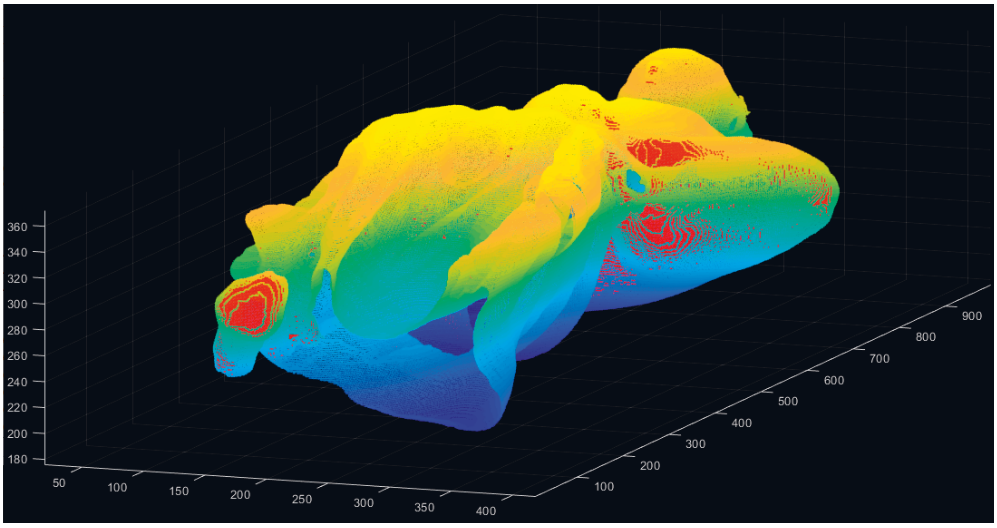


Figure 4.10 The main tube where points reconstructed at Primitive III are highlighted in red.

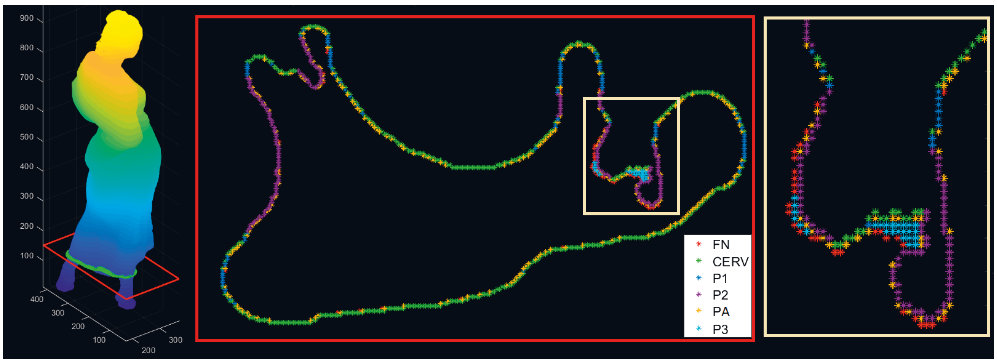


Figure 4.11 Left: Input point cloud. Center: Points at a single section (FN: Missed points, CERV: Points reconstructed by CERV; P1, P2, PA, P3: Points reconstructed by Primitives I, II, A, III). Right: Close up view of the section.

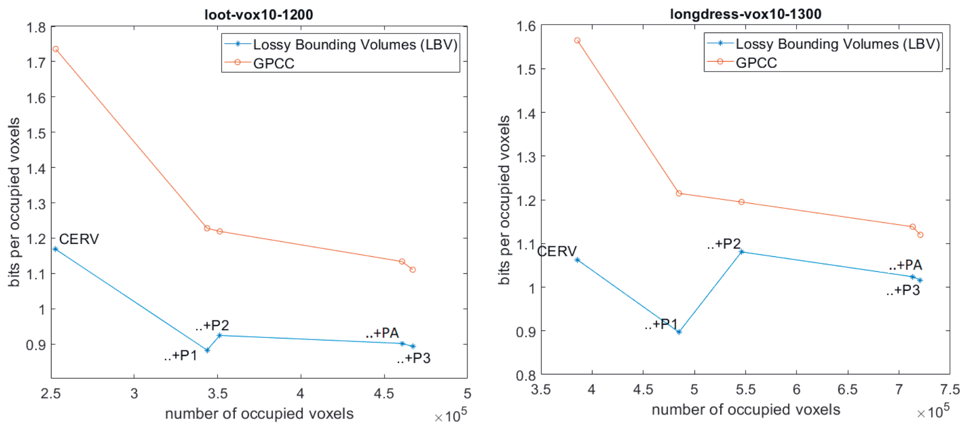


Figure 4.12 Blue Curves: Bitrate vs number of reconstructed points obtained after individual stages of LBV (CERV, Primitives I,II,A,III). Orange Curves: Bitrate vs number of reconstructed points when the same reconstructions obtained after each stage are compressed with G-PCC.

or III) is completed. The G-PCC (orange curves) results are obtained by compressing the reconstructions obtained after each LBV stage using G-PCC codec. Such a comparison aims to demonstrate the bitrate efficiency of the primitives.

4.1.1.1 Quantization and Unquantization

For the lower bitrate modes of operation, we employ quantization at various levels. Quantization $Q_q(\mathscr{P})$ of a point cloud $\mathscr{P}$ with the quantization step size $q > 1$ is the first stage of encoding and is a two step operation such that, $\mathscr{P}_q = Q_q(\mathscr{P})$. In the first step, the quantized points

$$(x_q, y_q, z_q) = (\lfloor \frac{x}{q} \rfloor, \lfloor \frac{y}{q} \rfloor, \lfloor \frac{z}{q} \rfloor) \tag{4.1}$$

are obtained from all points $(x, y, z) \in \mathscr{P}$. Next, duplicate points are removed in the resulting set to obtain a unique set of points $\mathscr{P}_q$.

Unquantization function $Q_q^U(.)$ is the decoder counterpart and is the final stage of decoding. Let $\mathscr{P}_q^{DEC}$ denote the lossy reconstruction of the quantized point cloud $\mathscr{P}_q$ at the decoder side, consisting of the decoded points (x_q^D, y_q^D, z_q^D). For each decoded point $(x_q^D, y_q^D, z_q^D) \in \mathscr{P}_q^{DEC}$, unquantization function $Q_q^U(.)$ generates q^3 points as

$$x^D, y^D, z^D = (x_q^D + k_x, y_q^D + k_y, z_q^D + k_z), \forall k_c = 0, ..., q-1, \forall c \in x, y, z. \tag{4.2}$$

The resulting set of points is called the decoded point cloud $\mathscr{P}^{DEC} = Q_q^U(\mathscr{P}_q^{DEC})$ where $\mathscr{P}_q^{DEC} = Q_q(\mathscr{P}^{DEC}) = Q_q(Q_q^U(\mathscr{P}_q^{DEC}))$. Hence, the unquantized point cloud $\mathscr{P}^{DEC}$ contains all the points x, y, z that may yield a point x_q, y_q, z_q in $\mathscr{P}_q^{DEC}$ when (4.1) is applied.

4.1.1.2 Experimental Results

Performance of LBV is evaluated with the D1 metric which was defined in (2.6). Rate-D1 curves for Cat 1A samples are presented in Fig. 4.13 in dB scale. It is observed from Fig. 4.13 that LBV outperforms G-PCC baseline and trisoup codecs for wide ranges of bitrates. Furthermore, it is observed that distortion is most of the time monotonically decreasing with the bitrate, which is a very desirable quality for a lossy encoder. Rate-D1 curves of LBV in Fig. 4.13 contain the operating points obtained by adjusting the quantization parameter as $1 <= q <= 4$, where $q = 1$ is equivalent to having no quantization.

Besides distortion and bitrates, another important consideration is encoding/decoding times. In Fig. 4.14 is plotted quantization parameter q vs. encoding and decoding times for the longdress sample for the same q values as in Fig. 4.13 ($1 <= q <= 4$). It is observed that, for all quantization levels, decoding proceeds faster than encoding.

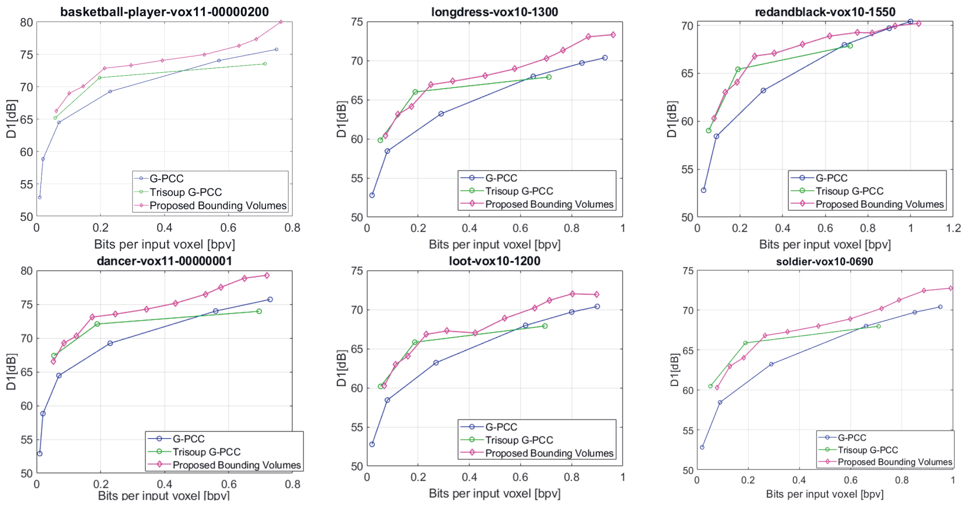


Figure 4.13 Results obtained with the proposed Lossy Bounding Volumes (LBV) scheme (pink) compared against the G-PCC baseline and trisoup codecs.

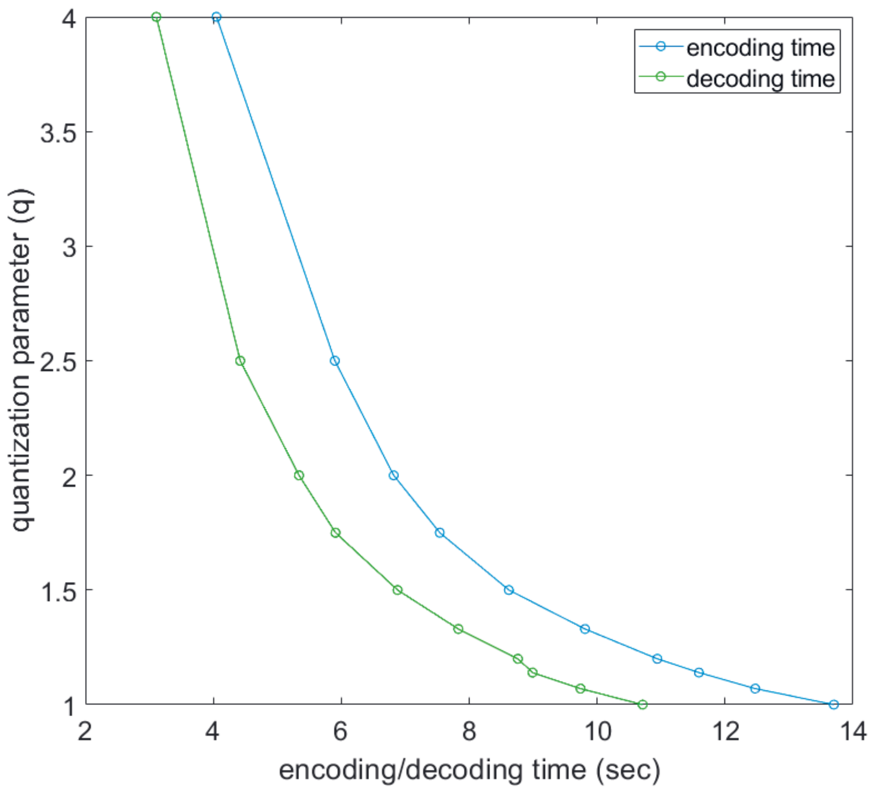


Figure 4.14 Quantization parameter q vs. encoding/decoding times of LBV for longdress from Cat1A.

4.1.2 Lossless Compression using Bounding Volumes

LBV, described in the previous section, employed a projection (depth map extraction) stage that generates depth maps in which the points, which are visible from the two opposite directions in the projection axis, are represented. In the subsequent stages, which were called primitives, we were able to encode some of the remaining points in an efficient way by making certain assumptions. Considering the 2D section-wise processing of the point cloud, all of the encoded points in primitives were, by construction, 8-connected to the projected points in the currently processed section.

In Publication III, is presented a lossless scheme called BVL that starts with the same projection stage as in LBV where the projected points are represented and encoded as depth maps in the same way as in LBV. Similar to LBV, the point cloud is swept along one axis and processed section-by-section. However, unlike LBV, this time, at every 2D section, all of the points which are connected to the projection points (via a path of 8-connected points in the 2D section) are encoded in the same way with a context coding scheme which will be described in detail. The projected points and the points which are connected to the projected points with an 8-connectivity path, together form what we call a *shell*. Once a *shell* is encoded, one takes out the *shell*, like a matryoshka doll, and the procedure is repeated for the remaining points. While in general, a point cloud, by definition can consist of a large number of shells, for the common types of point clouds that are of practical use, just a few number of these shell iterations are sufficient to encode the entire point cloud. In fact, in all our experiments, we encode only 2 shells and the remaining points (if any) are directly written to bitstream without any compression. Shells for 3 point clouds are visualized in Fig. 4.15. For relatively simpler point clouds, all of the points can be contained in two shells. However, for more complex point clouds, the method starts to become inefficient.

An overview of the scheme for encoding a shell is presented in Fig. 4.16. In the 2D cross section, all of the not projected points which are connected to the projected (depth map) points are shown in blue and they belong to the currently encoded shell. The small group of points that are shown in red are not part of this shell hence they will be encoded as part of the second shell.

Encoding of the remaining shell points in a 2D section which are shown in blue

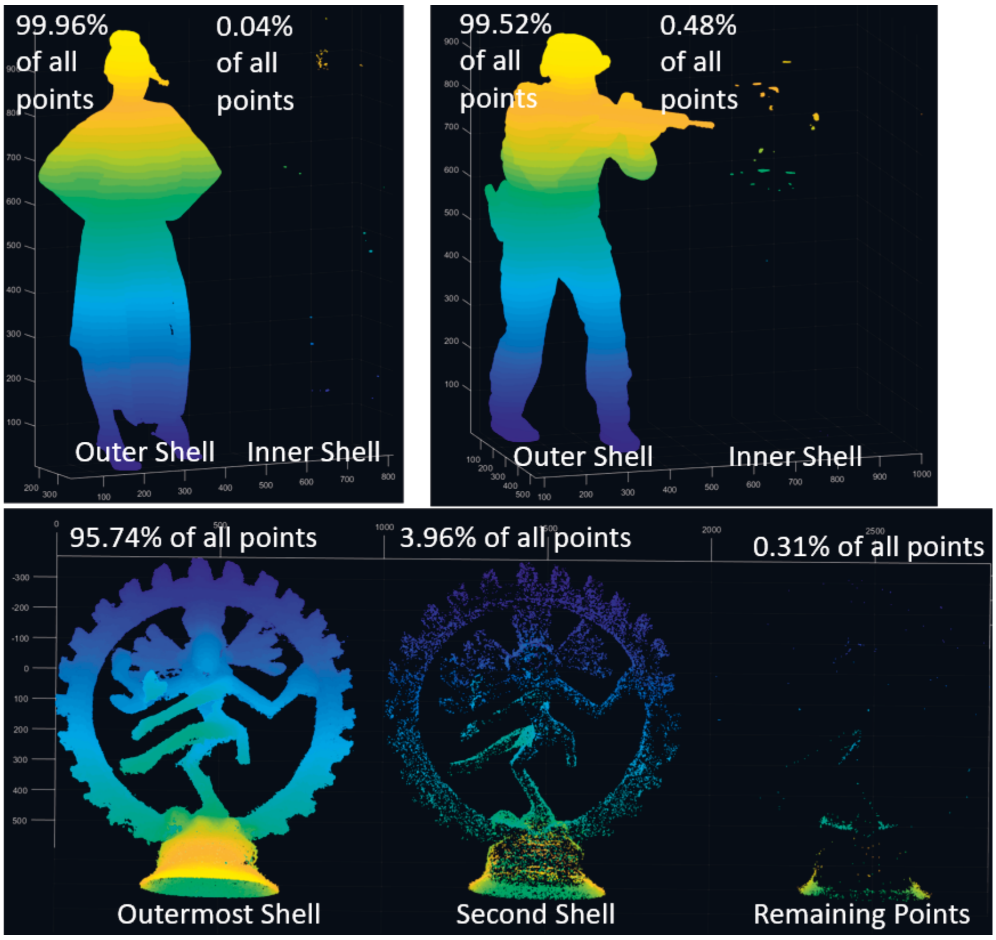


Figure 4.15 Shells of point clouds. For the point clouds in top row, all of the points are contained in two shells. For the shiva point cloud (2nd row), the structure of the inner shells is much more complex than the former point clouds.

in Fig. 4.16 is summarized in Algorithm 1. Here we shall go through the algorithm steps in detail and refer to the related lines in Algorithm 1 where necessary.

Let $\mathscr{P}$ denote the input point cloud defined in a voxel grid of size $N_x \times N_y \times N_z$. We denote the sweeping axis as y. A 2D section of $\mathscr{P}$ with $y = y_0$ can be represented with a binary image $T_{y_0}(z,x)$ where $T_{y_0}(z,x) = 1$ if $(x, y_0, z) \in \mathscr{P}$ and $T_{y_0}(z,x) = 0$ otherwise. In the first step of Algorithm 1, the encoder/decoder initializes the reconstruction of the current cross section R_{y_0} with the already encoded/decoded depth map points. Next, the encoder/decoder constructs the feasible regions image F from R_{y_0}. Note that, unlike the situation in LBV where we had a so-called tube as

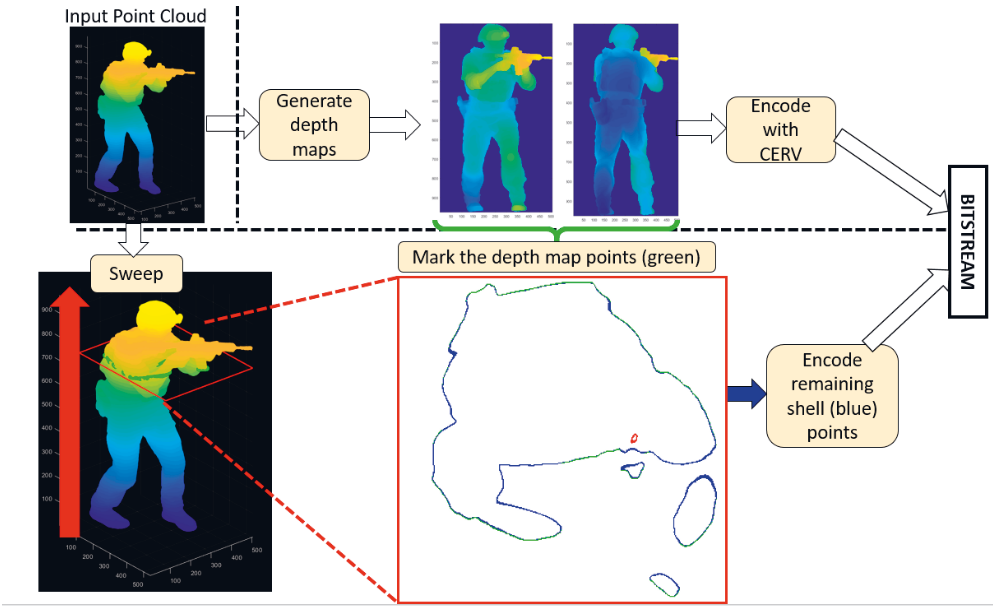

Input Point Cloud
Generate depth maps
Encode with CERV
BITSTREAM
Sweep
Mark the depth map points (green)
Encode remaining shell (blue) points

Figure 4.16 Encoding a shell in BVL. Encoding of the remaining shell points is summarized in Algorithm 1.

input, T_{y_0} is not constrained to contain a single connected component therefore F can contain more than one connected component as well.

It is useful to represent our knowledge of occupancies of all the locations in the current section with a binary image K. All of the locations, for which $F(z,x) = 0$, are known to be unoccupied and the locations, for which $R_{y_0}(z,x) = 1$ (the projected points), are known to be occupied. Therefore, initially K is simply $\overline{F}+R_{y_0}$ where $\overline{F}$ is the logical inverse of F. K is updated whenever a new location is encoded/decoded.

With the projected points, we initialize a list of locations to be processed called L_p. The processing of a location in L_p involves two steps: Firstly, encoding/decoding its occupancy if it's not already known and secondly, appending its 8-connected neighbors to L_p if their occupancies are not already known. Once a location is processed, it is removed from L_p. Processing continues until L_p is empty and all of the shell points are encoded. Since initially all of the locations in L_p are known locations, the lines 8-17 in Algorithm 1 are inactive and we simply add the neighboring unknown locations (line 20).

The occupancy of an unknown location in L_p has to be encoded/decoded. The encoding is performed using arithmetic coding with context. The context of a location (x, y_0, z) is extracted from a $3 \times 3 \times 2$ neighborhood in the following manner. Let the location $l_n = (x_n, y_0, z_n)$ be one of the 8 neighbors of (x, y_0, z) at the current section. Regarding occupancy and our knowledge of occupancy, there are three possibilities: l_n's occupancy may be unknown, l_n may be known to be unoccupied or l_n may be known to be occupied. We consider all three possibilities as context information. The context information regarding all 8 neighbors can be represented in a ternary 3×3 matrix C_{y_0} which can be computed by simply adding a R_{3x3} and a K_{3x3} cropped from R_{y_0} and K from the 8-neighborhood of (x, y_0, z). The second part of the context called C_{y_0-1} is extracted from the reconstruction of the previous section R_{y_0-1}. Regarding the previous section, there are 9 neighbors for which $R_{y_0-1}(z_n, x_n)$ can be either 0 or 1. These are represented with the 3×3 binary matrix C_{y_0-1} cropped from R_{y_0-1}. In order to make use of rotational invariances, we apply a normalizing rotation $\alpha^*(.)$ such that the four rotated versions of the 3D context around the sweeping axis are treated as the same context. Eventually, a context is represented with a unique label ζ which consists of a pair of integers.

$$\zeta = (I(C_{y_0,\alpha^*}), J(C_{y_0-1,\alpha^*})) \tag{4.3}$$

where,

$$I(C) = \sum_{j=0}^{2} \sum_{i=0}^{2} C_{ij} 3^{i+3j} \tag{4.4}$$

and

$$J(C) = \sum_{j=0}^{2} \sum_{i=0}^{2} C_{ij} 2^{i+3j}. \tag{4.5}$$

Considering that the highest value $J(C_{y_0-1,\alpha^*})$ can take is $2^9 - 1$, one can also express the unique label ζ as an integer as $\zeta = I(C_{y_0,\alpha^*})2^9 + J(C_{y_0-1,\alpha^*})$. Lines 8-14 in Algorithm 1 are the context formation steps.

4.1.2.1 Experimental Results

In this section we shall present the experimental results obtained with BVL. In Table 4.1, bitrates obtained with BVL are compared with G-PCC for 11 and 10 bit point clouds from Category 1A of MPEG. With most of the point clouds, BVL

Algorithm 1: Encoding the Remaining Shell Points

```
Require: $T_{y_0}$: Current section ($y = y_0$) ground truth
       $R_{y_0-1}$: Reconstruction of the previously encoded section ($y = y_0 - 1$)
 1: Initialize reconstruction of the current section $R_{y_0}$ with projected points
 2: Construct the feasible regions image $F$ using $R_{y_0}$
 3: Initialize $K$, the binary image of known locations $K \leftarrow \overline{F} + R_{y_0}$
 4: Initialize $L_p$, the list of pixels to be processed $L_p \leftarrow \{(z,x)|R_{y_0}(z,x) = 1\}$
 5: while $L_p \neq \emptyset$ do
 6:     Read $(z,x)$ from the top of $L_p$
 7:     if $K(z,x) == 0$ then
 8:         Extract a $3 \times 3$ matrix $R_{3x3}$ from $R_{y_0}$ centered at $(z,x)$
 9:         Extract a $3 \times 3$ matrix $K_{3x3}$ from $K$ centered at $(z,x)$
10:         $C_{y_0} \leftarrow R_{3x3} + K_{3x3}$
11:         Extract a $3 \times 3$ matrix $C_{y_0-1}$ from $R_{y_0-1}$ centered at $(z,x)$
12:         Find normalizing rotation $\alpha^*(C_{y_0})$ and form $C_{y_0,\alpha^*}$
13:         Use $\alpha^*(C_{y_0})$ to rotate $C_{y_0-1}$ as $C_{y_0-1,\alpha^*}$
14:         Form the context $\zeta = (I(C_{y_0,\alpha^*}), J(C_{y_0-1,\alpha^*}))$
15:         Encode $T_{y_0}(z,x)$ using the context $\zeta$
16:         Update the current reconstruction: $R_{y_0}(z,x) \leftarrow T_{y_0}(z,x)$
17:         Update K: $K(z,x) \leftarrow 1$
18:     end if
19:     if $R_{y_0}(z,x) == 1$ then
20:         Append to $L_p$ every 8-connected neighbor $(z_n,x_n)$ of $(z,x)$ for which
            $K(z_n,x_n) = 0$
21:     end if
22:     Remove $(z,x)$ from $L_p$
23: end while
```

outperforms G-PCC codec. In Table 4.2, the average bitrate results for sequences from MVUB [44] and 8i [14] datasets are presented and compared with other recent codecs. With the MVUB point clouds, Dyadic Decomposition based method [62] yields the best results, whereas BVL results are close to G-PCC. With the 8i dataset, BVL outperforms the other methods.

Table 4.1 Average Rate [Bpov] results on point clouds from CAT1A

Point Cloud	Bitdepth	G-PCC	BVL
basketball player	11	0.885	**0.852**
dancer	11	0.876	**0.826**
facade 00064	11	**1.1969**	1.3331
longdress	10	1.032	**0.9223**
loot	10	0.9818	**0.8991**
queen 0200	10	**0.783**	0.7883
redandblack	10	1.1055	**1.0418**
shiva 00035	10	**3.6966**	5.5642
soldier	10	1.0419	**0.9577**

Table 4.2 Average Rate for the first 200 frames for all sequences from MVUB [44] and 8i [14] for the proposed BVL encoder compared to recent codecs.

Sequence	Average Rate [bpv]			
	P(PNI)[18]	**G-PCC[52]**	**DD[62]**	**BVL**
Microsoft Voxelized Upper Bodies [44]				
Andrew9	1.83	1.14	**1.12**	1.17
David9	1.77	1.08	**1.06**	1.10
Phil9	1.88	1.18	**1.14**	1.20
Ricardo9	1.79	1.08	**1.03**	1.05
Sarah9	1.79	**1.07**	**1.07**	1.08
Average	1.81	1.11	**1.08**	1.12
8i Voxelized Full Bodies [14]				
Longdress	1.75	1.03	0.95	**0.91**
Loot	1.69	0.97	0.91	**0.88**
Redandblack	1.84	1.11	**1.03**	**1.03**
Soldier	1.76	1.04	**0.96**	**0.96**
Average	1.76	1.04	0.96	**0.94**

4.2 NNOC: Neural Networks and Octrees

In Publication IV, we have proposed a lossless point cloud geometry compression scheme called NNOC (short for neural network and octree). In this scheme, octree representation of a point cloud is encoded with arithmetic coding using probabilities estimated by a neural network. Octree representation is a multiresolution representation in which, different depth levels in the octree representation correspond to different resolutions of the point cloud. The octree is processed in breadth-first order starting from a very low depth level and reaching the original resolution of the point cloud in a resolution-progressive manner. An overview of NNOC is presented in Fig. 4.17. At every resolution level, candidate (possibly occupied) voxels are obtained from the one step lower resolution and they are traversed at a certain order which establishes the causality status for the voxels. According to the causality status determined by the scanning order, for some of the voxels only the candidacy information is available (coming from the previously encoded/decoded resolution), whereas for the remaining voxels, occupancy status are available. For the currently encoded candidate voxel, a hybrid context, which consists of neighboring occupancies (where available) and candidacies, is formed as a binary vector. The probability of occupancy of the currently encoded candidate voxel is estimated based on the context vector. Traversing of candidate voxels and context extraction are made efficient by utilizing binary images (buffers) that hold the relevant information.

The geometry-based point cloud compression (G-PCC) test model (TMC13) developed by MPEG also encodes octree representation with arithmetic coding. The two major differences between our model and G-PCC are worth pointing out: Firstly, the way we extract the context is fundamentally different from G-PCC. In G-PCC, the location and orientation of the context region with respect to the corresponding voxel is not the same for all voxels. Different child nodes of the same parent voxel share the same context elements. The second major difference is, our scheme employs a neural network for estimating the probability of occupancy of a voxel given its context. As a result of this, unlike many other arithmetic coding schemes including G-PCC, we do not need to keep track of the number of occurences for each of the contexts to estimate a probability. Using a neural network allows us to handle a huge number of possible contexts which would not be possible by keeping occurences in lookup tables explicitly. Specifically, our context can take around 2^{95}

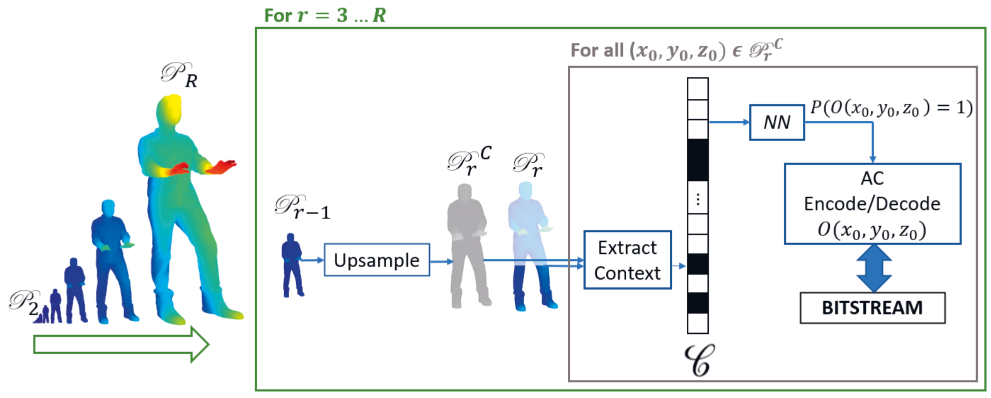


Figure 4.17 Overview of NNOC. Encoding and decoding proceeds from low resolution to high resolution. At every resolution r, the candidate voxels $\mathscr{P}_r^C$ obtained by upsampling the reconstruction at the previous resolution $\mathscr{P}_{r-1}$ are traversed. Probability of occupancy of a candidate voxel is estimated by a neural network based on a context derived from current and past resolution information.

different configurations.

In Fig. 4.17, the input point cloud is denoted P_R where R is the resolution. Encoding and decoding proceeds from low resolution to high resolution. At every resolution r, the occupancies of the octree nodes are encoded with arithmetic coding where the probability model is a neural network for which the weights are obtained through offline training.

NNOC encoder and decoder are summarized in Algorithms 2 and 3, respectively, which are conceptual descriptions that aim to maintain a certain level of readability while conveying the most significant aspects of the method. For a more detailed description of the NNOC encoder, we refer the reader to Algorithm 6 which generates the same bitstream as in Algorithm 2 in a faster way.

Suppose the input point cloud is $\mathscr{P}_R$ with R bits per dimension resolution consisting of points $(x_R, y_R, z_R) \mid x_R, y_R, z_R \in \mathbb{Z} \cap [0, 2^R - 1]$. NNOC encoder generates the lower resolution versions of $\mathscr{P}_R$; $\mathscr{P}_{R-1} \dots \mathscr{P}_2$ by downsampling $\mathscr{P}_R$ repeatedly by 2 in the following manner. Let $\mathscr{P}_r$ be a point cloud with resolution $r \in \{2, .., R\}$. $\mathscr{P}_{r-1}$ is the unique set of points obtained by mapping every point (x_r, y_r, z_r) in $\mathscr{P}_r$ to a point $(x_{r-1}, y_{r-1}, z_{r-1})$ such that

$$(x_{r-1}, y_{r-1}, z_{r-1}) = (\lfloor \frac{x_r}{2} \rfloor, \lfloor \frac{y_r}{2} \rfloor, \lfloor \frac{z_r}{2} \rfloor). \tag{4.6}$$

$\mathscr{P}_2$ is a point cloud in $4 \times 4 \times 4$ voxel grid and it is written to the bitstream in 64 bits where each bit corresponds to the occupancy of one voxel in the grid. Hence, the first thing the decoder reads from the bitstream is $\mathscr{P}_2$. Starting with $r = 3$, for all resolutions $r = 3, .., R$, the encoder/decoder generates candidate locations (possibly occupied voxels) in resolution r called $\mathscr{P}_r^C$ using $\mathscr{P}_{r-1}$, where

$$\mathscr{P}_r^C = \{(x_c, y_c, z_c) \,|\, (x_c, y_c, z_c) = (2x_{r-1} + \alpha, 2y_{r-1} + \beta, 2z_{r-1} + \gamma);\ \alpha, \beta, \gamma \in \{0, 1\}\} \tag{4.7}$$

and

$$(x_{r-1}, y_{r-1}, z_{r-1}) \in \mathscr{P}_{r-1}. \tag{4.8}$$

$\mathscr{P}_r$ is a subset of $\mathscr{P}_r^C$. After generating $\mathscr{P}_r^C$, encoder and decoder traverse $\mathscr{P}_r^C$ in the same order. Here, the fastest and slowest traversal directions are denoted $+y$ and $+z$, respectively. The occupancy status $O(x, y, z)$ of candidate locations are encoded/decoded using arithmetic coding with 2 symbols. Probability of occupancy of a candidate location $(x_0, y_0, z_0) \in \mathscr{P}_r^C$ is the probability of (x_0, y_0, z_0) being an element of $\mathscr{P}_r$:

$$P(O(x_0, y_0, z_0) = 1) = P((x_0, y_0, z_0) \in \mathscr{P}_r) \tag{4.9}$$

and is estimated with a neural network (NN) such that

$$P(O(x_0, y_0, z_0) = 1) = NN(\mathscr{C}_0) \tag{4.10}$$

where, $\mathscr{C}_0$ is the context vector extracted around location (x_0, y_0, z_0) and

$$\mathscr{C}_0 = [c_0..c_{4w^2}]. \tag{4.11}$$

The context vector $\mathscr{C}_0$ is obtained by flattening a 3D binary context region. The dimensions of the 3D context region is $w \times w \times 4$ in x, y, z coordinates, respectively where w is set to be 5. The context region is symmetric around the current candidate location (x_0, y_0, z_0) in x and y directions but it is not symmetric in the z direction because the locations at $z = z_0 + 2$ plane are excluded from the context while locations with $z = z_0 - 2$ are part of the context. It is in a way justified in the sense that

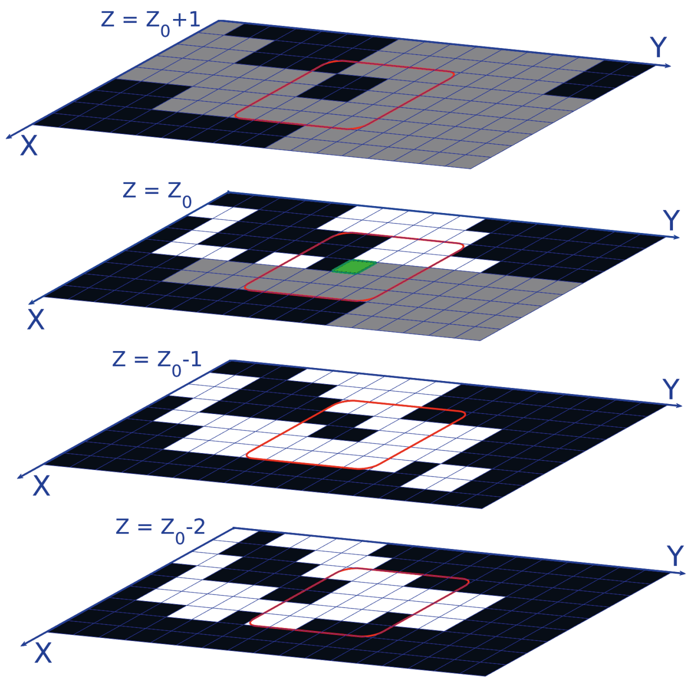


Figure 4.18 Context extraction in NNOC. Green Tile: Candidate location currently being encoded/decoded. White Tiles: Locations already encoded/decoded with occupancy status 1. Black Tiles: Locations with candidacy/occupancy status 0. Grey Tiles: Candidate locations that are not yet encoded/decoded. Red squares: $w \times w$ context windows surrounding the context region. © 2021 IEEE

for the $z = z_0 + 2$ locations, only the candidacy (lower resolution) information is available whereas for the $z = z_0 - 2$ locations, the true occupancy status are known. Candidacy is for sure less informative when compared to the occupancy. Context extraction in NNOC is visualized in Fig. 4.18.

When constructing the context vector, context elements are scanned in the same order as the traversal order of candidates, where $+y$ is the fastest and $+z$ is the slowest scanning direction, respectively. Scanning order of context elements can be expressed formally as a one-to-one mapping from an integer i to a triplet of integer coordinate shifts as $i \rightarrow (\hat{x}(i), \hat{y}(i), \hat{z}(i))$. According to this, the first $(5w^2 - 1)/2$ elements in $\mathscr{C}_0$ are expressed as,

$$c_i = O(x_0 + \hat{x}(i), y_0 + \hat{y}(i), z_0 + \hat{z}(i)), \quad i \in \mathbb{Z} \cap [0, \frac{5w^2 - 3}{2}] \tag{4.12}$$

where $O(.)$ denotes the occupancy of a voxel considering $\mathscr{P}_r$. The next $(3w^2+1)/2$ elements in $\mathscr{C}_0$ are expressed in a similar way,

$$c_i = O^C(x_0+\hat{x}(i), y_0+\hat{y}(i), z_0+\hat{z}(i)), \quad i \in \mathbb{Z} \cap [\frac{5w^2-1}{2}, 4w^2-1] \tag{4.13}$$

where $O^C(.)$ denotes the candidacy of a voxel considering $\mathscr{P}_r^C$.

In order to be able to fetch the occupancies and candidacies that constitute the context vector in an efficient manner, during traversal of the candidates, four binary images I_z with $z = z_0-2, z_0-1, z_0, z_0+1$ are constructed both at the encoder and the decoder side. Here, z_0 denotes the z coordinate of the currently processed candidate locations.

$$I_{z_0+j}(x,y) = O(x,y,z_0+j),\ j \in \{-1,-2\} \tag{4.14}$$

$$I_{z_0}(x,y) = \begin{cases} O(x,y,z_0), & \text{if } x < x_0 \\ O(x,y,z_0), & \text{if } x = x_0, y < y_0 \\ O^C(x,y,z_0), & \text{otherwise} \end{cases} \tag{4.15}$$

$$I_{z_0+1}(x,y) = O^C(x,y,z_0+1) \tag{4.16}$$

Additionally, the encoder constructs a binary image of true occupancies T_{z_0} as

$$T_{z_0}(x,y) = O(x,y,z_0) \tag{4.17}$$

4.2.1 Parallel NN execution and the fast model

For the sake of speed, parallel computation can be employed during NN execution so that if $\mathscr{L}_{\mathscr{C}}$ is a list of context vectors, a single network execution can generate a list of occupancy probabilities $\mathscr{L}_P$,

$$\mathscr{L}_P = NN(\mathscr{L}_{\mathscr{C}}) \tag{4.18}$$

At the encoder side, parallel execution is possible since all the occupancies are available. Thus, occupancy probabilities for all locations having the same z are estimated with a single network pass. However, at the decoder side, the situation is different. For instance, the context vector $\mathscr{C}_0$ contains the occupancy $O(x_0, y_0 - 1, z_0)$ which is available to the decoder only after decoding it. Therefore, the decoding proceeds by passing only one context vector to the neural network at a time which turns out to be much slower than the encoding.

Alternatively, we propose a fast version called fNNOC which enables the parallel NN execution at the decoder side by defining the context in a slightly different way. fNNOC encoder and decoder are summarized in Algorithms 4 and 5, respectively. In fNNOC, context vector $\mathscr{C}_0$ contains only candidacy values $O^C(x, y, z_0)$ thus no occupancies $O(x, y, z_0)$ are involved. Hence, the decoder can collect the same list of contexts $\mathscr{L}_{\mathscr{C}}$ for all candidates having $z = z_0$ as in the encoder side. The context extraction in fNNOC is depicted on Fig. 4.19. Note that, only the $z = z_0$ part of the context is changed and accordingly, in fNNOC,

$$I_{z_0}(x, y) = O^C(x, y, z_0) \tag{4.19}$$

whereas $I_{z_0-2}, I_{z_0-1}, I_{z_0+1}$ are the same as in (4.14) and (4.16).

4.2.2 Neural Network Structure and Training

The neural network structure is depicted in Fig. 4.20. It is a multi layer perceptron with a single hidden layer. The input to the neural network is a binary vector of length $4w^2$ and the number of neurons in the hidden layer is set as twice the input length which is $8w^2$. The number of output neurons is set as 2 where the output values are denoted α_1 and α_2. Output probability estimation is computed with a nonlinear transformation as,

$$P(O(x_0, y_0, z_0)) = \frac{e^{\alpha_1}}{e^{\alpha_1} + e^{\alpha_2}}. \tag{4.20}$$

The nonlinear activation function at the output ensures that the output value is a valid probability estimation between 0 and 1. The total number of trainable parameters in NN is $32w^4 + 24w^2 + 2$. For $w = 5$, it is 20602 parameters.

Input binary context vector $\mathscr{C}_0$ combines the information at the neighborhood of (x_0, y_0, z_0) coming from $\mathscr{P}_r^C$ and current reconstruction of $\mathscr{P}_r$. While some of the elements in $\mathscr{C}_0$ represent the candidacy status $O^C(.)$ of a neighbor location, the

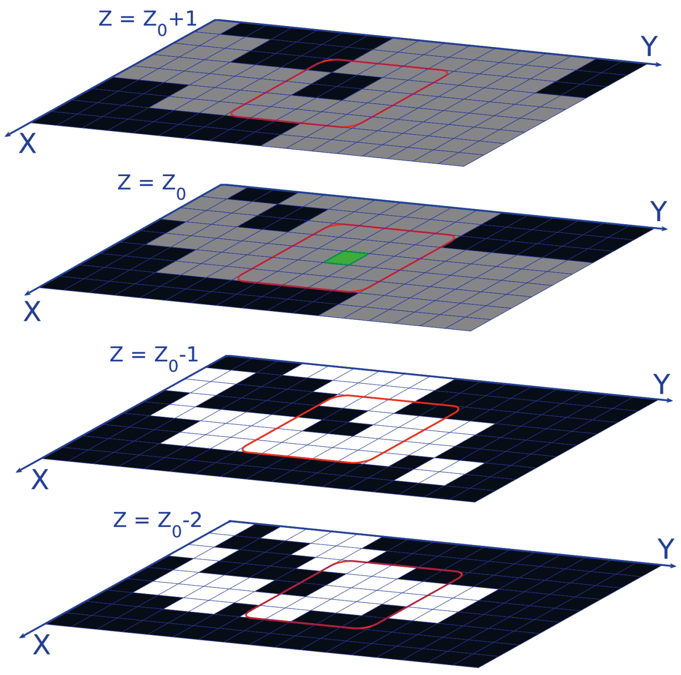
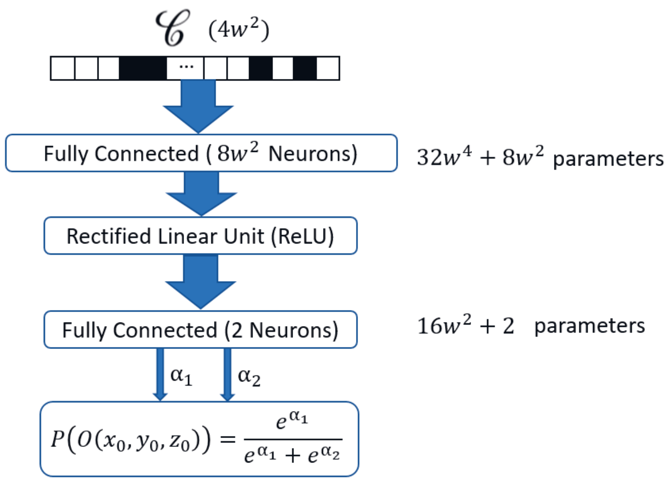


Figure 4.19 Context extraction in fNNOC. Green Tile: Candidate location currently being encoded/decoded. White Tiles: Locations already encoded/decoded with occupancy status 1. Black Tiles: Locations with candidacy/occupancy status 0. Grey Tiles: Candidate locations that are not yet encoded/decoded. Red squares: $w \times w$ context windows surrounding the context region. © 2021 IEEE



$\mathscr{C}$ $(4w^2)$
...
Fully Connected ($8w^2$ Neurons)
$32w^4 + 8w^2$ parameters
Rectified Linear Unit (ReLU)
Fully Connected (2 Neurons)
$16w^2 + 2$ parameters
α_1
α_2
$P(O(x_0, y_0, z_0)) = \frac{e^{\alpha_1}}{e^{\alpha_1} + e^{\alpha_2}}$

Figure 4.20 Structure of the Neural Network in NNOC.

Algorithm 2: Encoding with NNOC

```
Require: A point cloud $\mathscr{P}_R$ with resolution $R$ bits/dimension
1: Construct lower resolution point clouds $\mathscr{P}_{R-1},\ldots,\mathscr{P}_2$ representing the full
   nodes at octree depth levels $R-1,\ldots,2$.
2: Encode $\mathscr{P}_2$ in 64 bits
3: Encode iteratively $\mathscr{P}_3$ to $\mathscr{P}_R$ as follows:
4: for $r=3,\ldots,R$ do
5:     Generate for each point $(x_{r-1},y_{r-1},z_{r-1})\in\mathscr{P}_{r-1}$ 8 candidate voxels in the
       resolution $r$, resulting in the set of candidate points $\mathscr{P}_r^C$
6:     Traverse candidates $\mathscr{P}_r^C$ and encode occupancies as follows:
7:     for $z_0=0\ldots 2^r-1$ do
8:         Initialize empty list of context vectors $\mathscr{L}_{\mathscr{C}}=\{\}$
9:         Update $I_{z_0+1},I_{z_0},I_{z_0-1},I_{z_0-2}$ and $T_{z_0}$ using points from $\mathscr{P}_r^C$ and $\mathscr{P}_r$
10:        for $x_0=0\ldots 2^r-1$ do
11:            for $y_0=0\ldots 2^r-1$ do
12:                if $(x_0,y_0,z_0)\in\mathscr{P}_r^C$ then
13:                    Extract the context vector $\mathscr{C}_0$ from $I_{z_0+1},I_{z_0},I_{z_0-1},I_{z_0-2}$
14:                    Append $\mathscr{C}_0$ to $\mathscr{L}_{\mathscr{C}}$
15:                    Update $I_{z_0}\leftarrow T_{z_0}(x_c,y_c,z_0)$
16:                end if
17:            end for
18:        end for
19:        Pass all contexts in $\mathscr{L}_{\mathscr{C}}$ to NN in parallel to obtain $\mathscr{L}_P=NN(\mathscr{L}_{\mathscr{C}})$
20:        for all $(x_c,y_c,z_0)\in\mathscr{P}_r^C$ do
21:            Encode the occupancy $O(x_c,y_c,z_0)$ using $P(O(x_c,y_c,z_0)=1)$ from $\mathscr{L}_P$
22:        end for
23:    end for
24: end for
```

remaining elements represent the true occupancy status $O(.)$ at the current resolution. Therefore, it is worth noting that the neural network has devoted weights to operate on these two different types of elements.

As an entropy coding method, arithmetic coding can achieve bits per symbol codelengths very close to the entropy. This allows us to estimate the total codelength and optimize the probabilistic model such that it minimizes the total codelength. Our probabilistic model produces conditional probabilities such that the probability of occurence of a symbol s_j is expressed as $P(s_j|\mathscr{C})$ where $\mathscr{C}$ is a context vector. There are only 2 symbols, that is, a candidate voxel is either occuppied or unoccupied

Algorithm 3: Decoding with NNOC

```
Require: Bitstream
1: Reconstruct $\mathscr{P}_2$ from first 64 bits
2: Decode iteratively $\mathscr{P}_3$ to $\mathscr{P}_R$ as follows:
3: for $r = 3,\ldots,R$ do
4:   Initialize $\mathscr{P}_r = \{\}$
5:   Generate for each point $(x_{r-1}, y_{r-1}, z_{r-1}) \in \mathscr{P}_{r-1}$ the 8 candidate voxels in
     the resolution $r$, resulting in the set of candidate points $\mathscr{P}_r^C$
6:   Traverse candidates $\mathscr{P}_r^C$ and decode occupancies as follows:
7:   for $z_0 = 0\ldots 2^r - 1$ do
8:     Update $I_{z_0+1}, I_{z_0}, I_{z_0-1}, I_{z_0-2}$ using points from $\mathscr{P}_r^C$ and $\mathscr{P}_r$
9:     for $x_0 = 0\ldots 2^r - 1$ do
10:      for $y_0 = 0\ldots 2^r - 1$ do
11:        if $(x_0, y_0, z_0) \in \mathscr{P}_r^C$ then
12:          Extract the context vector $\mathscr{C}_0$ from $I_{z_0+1}, I_{z_0}, I_{z_0-1}$ and $I_{z_0-2}$
13:          Obtain the occupancy probability $P(O(x_0, y_0, z_0) = 1) = NN(\mathscr{C}_0)$
14:          Decode the occupancy $O(x_0, y_0, z_0)$ using $P(O(x_0, y_0, z_0) = 1)$
15:          Update $I_{z_0}(x_0, y_0) \leftarrow O(x_0, y_0, z_0)$
16:          if $O(x_0, y_0, z_0) = 1$ then
17:            Add $(x_0, y_0, z_0)$ to $\mathscr{P}_r$
18:          end if
19:        end if
20:      end for
21:    end for
22:  end for
23: end for
```

hence

$$s_j \equiv O(x_0, y_0, z_0) = j,\ j \in \{0, 1\}. \tag{4.21}$$

Let different context vectors be denoted as C_i where i is an integer index, probability of occupancy of a candidate voxel given a context $\mathscr{C}_i$ is expressed as $P_{1i} = P(s_1|\mathscr{C}_i)$ and similarly $P_{0i} = P(s_0|\mathscr{C}_i)$. P_{1i} is what is produced by the neural network $P_{1i} = NN(\mathscr{C}_i)$ and $P_{0i} = 1 - P_{1i}$. If the probability of occurence of a symbol is P_{ji}, arithmetic coding can achieve an average (bits per symbol) codelength close to $-\log(P_{ji})$. Thus, if the number of occurences corresponding to a symbol-context pair $(s_j, \mathscr{C}_i)$ is denoted n_{ji}, the estimated total codelength for arithmetic coding is

Algorithm 4: Encoding with fNNOC

```
Require: A point cloud $\mathscr{P}_R$ with resolution $R$ bits/dimension
 1: Construct lower resolution point clouds $\mathscr{P}_{R-1},\dots,\mathscr{P}_2$ representing the full
    nodes at octree depth levels $R-1,\dots,2$.
 2: Encode $\mathscr{P}_2$ in 64 bits
 3: Encode iteratively $\mathscr{P}_3$ to $\mathscr{P}_R$ as follows:
 4: for $r = 3,\dots,R$ do
 5:   Generate for each point $(x_{r-1},y_{r-1},z_{r-1}) \in \mathscr{P}_{r-1}$ 8 candidate voxels in the
      resolution $r$, resulting in the set of candidate points $\mathscr{P}_r^C$
 6:   Traverse candidates $\mathscr{P}_r^C$ and encode occupancies as follows:
 7:   for $z_0 = 0\dots 2^r - 1$ do
 8:     Initialize empty list of context vectors $\mathscr{L}_{\mathscr{C}} = \{\}$
 9:     Update $I_{z_0+1}, I_{z_0}, I_{z_0-1}, I_{z_0-2}$ and $T_{z_0}$ using points from $\mathscr{P}_r^C$ and $\mathscr{P}_r$
10:     for $x_0 = 0\dots 2^r - 1$ do
11:       for $y_0 = 0\dots 2^r - 1$ do
12:         if $(x_0,y_0,z_0) \in \mathscr{P}_r^C$ then
13:           Extract the context vector $\mathscr{C}_0$ from $I_{z_0+1}, I_{z_0}, I_{z_0-1}, I_{z_0-2}$
14:           Append $\mathscr{C}_0$ to $\mathscr{L}_{\mathscr{C}}$
15:         end if
16:       end for
17:     end for
18:     Pass all contexts in $\mathscr{L}_{\mathscr{C}}$ to NN in parallel to obtain $\mathscr{L}_P = NN(\mathscr{L}_{\mathscr{C}})$
19:     for all $(x_c,y_c,z_0) \in \mathscr{P}_r^C$ do
20:       Encode the occupancy $O(x_c,y_c,z_0)$ using $P(O(x_c,y_c,z_0) = 1)$ from $\mathscr{L}_P$
21:     end for
22:   end for
23: end for
```

$$\mathrm{CL} \approx -\sum_{i=1}^{N_{\mathscr{C}}} n_{0i}\log_2 P_{0i} + n_{1i}\log_2 P_{1i} \tag{4.22}$$

where $N_{\mathscr{C}}$ is the number of unique contexts.

Training set consists of $(\mathscr{C}_i, n_{0i}, n_{1i})$ triplets where $\mathscr{C}_i$ is a context vector that occurs in the training samples (point clouds) $n_{0i} + n_{1i}$ times. In n_{1i} of these occurences, occupancy associated with $\mathscr{C}_i$ is $O(x_0,y_0,z_0) = 1$ and in n_{0i} of them, it is $O(x_0,y_0,z_0) = 0$. Training triplets are collected from two datasets called 8i Voxelized Full Bodies [14] and Microsoft Voxelized Upper Bodies (MVUB) [44].

While total codelength (CL) is what we ideally want to minimize, due to practical

Algorithm 5: Decoding with fNNOC

```
Require: Bitstream
 1: Reconstruct $\mathscr{P}_2$ from first 64 bits
 2: Decode iteratively $\mathscr{P}_3$ to $\mathscr{P}_R$ as follows:
 3: for $r = 3, \ldots, R$ do
 4:   Initialize $\mathscr{P}_r = \{\}$
 5:   Generate for each point $(x_{r-1}, y_{r-1}, z_{r-1}) \in \mathscr{P}_{r-1}$ the 8 candidate voxels in
      the resolution $r$, resulting in the set of candidate points $\mathscr{P}_r^C$
 6:   Traverse candidates $\mathscr{P}_r^C$ and decode occupancies as follows:
 7:   for $z_0 = 0 \ldots 2^r - 1$ do
 8:     Initialize empty list of context vectors $\mathscr{L}_{\mathscr{C}} = \{\}$
 9:     Update $I_{z_0+1}, I_{z_0}, I_{z_0-1}, I_{z_0-2}$ using points from $\mathscr{P}_r^C$ and $\mathscr{P}_r$
10:     for $x_0 = 0 \ldots 2^r - 1$ do
11:       for $y_0 = 0 \ldots 2^r - 1$ do
12:         if $(x_0, y_0, z_0) \in \mathscr{P}_r^C$ then
13:           Extract the context vector $\mathscr{C}_0$ from $I_{z_0+1}, I_{z_0}, I_{z_0-1}$ and $I_{z_0-2}$
14:           Append $\mathscr{C}_0$ to $\mathscr{L}_{\mathscr{C}}$
15:         end if
16:       end for
17:     end for
18:     Pass all contexts in $\mathscr{L}_{\mathscr{C}}$ to NN in parallel to obtain $\mathscr{L}_P = NN(\mathscr{L}_{\mathscr{C}})$
19:     for all $(x_c, y_c, z_0) \in \mathscr{P}_r^C$ do
20:       Decode the occupancy $O(x_c, y_c, z_0)$ using $P(O(x_c, y_c, z_0) = 1)$ from $\mathscr{L}_P$
21:     end for
22:     if $O(x_0, y_0, z_0) = 1$ then
23:       Add $(x_0, y_0, z_0)$ to $\mathscr{P}_r$
24:     end if
25:   end for
26: end for
```

limitations the training loss is defined over a randomly selected batch of contexts rather than the entire training set as

$$\text{Loss} = -\frac{1}{N_b} \sum_{i=1}^{N_b} (n_{0i} \log_2 P_{0i} + n_{1i} \log_2 P_{1i}) \tag{4.23}$$

where N_b is the training batch size.

4.2.3 Additional Details on NNOC and fNNOC Algorithms

Algorithms 2,3,4,5 describe NNOC and fNNOC encoders and decoders completely, however, taking a closer look and noting the 3 loops of coordinates z_0, y_0, x_0 over the entire voxel grid (in lines 7,10,11 of Algorithm 2) would make one think that the method is cursed to be quite slow. In order to respond to these concerns and show how the method can be made feasible, we provide a more detailed description of the NNOC encoder in Algorithm 6. In Algorithm 6, after generating the candidate points (line 5), points in $\mathscr{P}_r$ and $\mathscr{P}_r^C$ are sorted into lists $\mathscr{L}_r$ and $\mathscr{L}_r^C$, such that z coordinate is the slowest changing coordinate. Next, two matrices each with size $(2^r - 1) \times 2$ called S and S^C are constructed where S is related to $\mathscr{P}_r$ and S^C is related to $\mathscr{P}_r^C$. The z_0'th row of S and S^C hold two indices that can be used to access the $z = z_0$ points in L_r and $\mathscr{L}_r^C$ such that $\mathscr{L}_{r,z_0} = \mathscr{L}_r(S(z_0, 0) : S(z_0, 1))$ and $\mathscr{L}_{r,z_0}^C = \mathscr{L}_r^C(S^C(z_0, 0) : S^C(z_0, 1))$ are sorted lists of points having $z = z_0$. To sum up, prior sorting of the points allows us to access the relevant data in an efficient way.

4.2.4 Experimental Results

Results for 11 and 10 bit point clouds from Category 1A of Common Test Conditions of MPEG [76] are presented in Table 4.3.

Since the NNOC encoder, the fNNOC encoder and the fNNOC decoder allow for parallel execution of neural network as in (4.18); for these three cases, neural network executions are performed on GPU. On the other hand, NNOC decoder cannot execute the NN in parallel on multiple contexts so the usage of GPU is not improving the speed. Therefore, NNOC decoder is executed using CPU only. The encoding and decoding durations obtained with different resolution samples are presented in Table 4.5. A breakdown of encoding and decoding times of NNOC and fNNOC is presented in Table 4.6. All of the durations are measured on an Intel(R) Xeon(R) Processor E5-2620 v4 with GeForce RTX 2080.

In order to investigate the effect of context size and network structure, we performed an ablation study. We trained 5 variants of fNNOC which we will shortly refer to as fN1-5. The differences of these models compared to fNNOC are as follows: In fN1, context region does not include the $z = z_0 + 1$ plane hence $\mathscr{C}_0$ has

Algorithm 6: Encoding with NNOC - A More Detailed Description

```
Require: A point cloud $\mathscr{P}_R$ with resolution $R$ bits/dimension
1: Construct lower resolution point clouds $\mathscr{P}_{R-1},\ldots,\mathscr{P}_2$ representing the full
   nodes at octree depth levels $R-1,\ldots,2$.
2: Encode $\mathscr{P}_2$ in 64 bits
3: Encode iteratively $\mathscr{P}_3$ to $\mathscr{P}_R$ as follows:
4: for $r=3,\ldots,R$ do
5:    Generate for each point $(x_{r-1},y_{r-1},z_{r-1})\in\mathscr{P}_{r-1}$ 8 candidate voxels in the
      resolution $r$, resulting in the set of candidate points $\mathscr{P}_r^C$
6:    Sort points in both $\mathscr{P}_r$ and $\mathscr{P}_r^C$ in ascending order such that, z coordinate is
      the slowest and y coordinate is the fastest increasing coordinate, resulting in
      sorted lists of points $\mathscr{L}_r$ and $\mathscr{L}_r^C$
7:    Construct two $(2^r-1)\times 2$ matrices $S$ and $S^C$ such that
      $\mathscr{L}_{r,z_0}=\mathscr{L}_r(S(z_0,0):S(z_0,1))$ and $\mathscr{L}_{r,z_0}^C=\mathscr{L}_r^C(S^C(z_0,0):S^C(z_0,1))$ are
      sorted lists of points with $z=z_0$.
8:    Traverse candidates $\mathscr{P}_r^C$ and encode occupancies as follows:
9:    for $z_0=0\ldots 2^r-1$ do
10:      Initialize empty list of context vectors $\mathscr{L}_{\mathscr{C}}=\{\}$
11:      Update $I_{z_0+1},I_{z_0},I_{z_0-1},I_{z_0-2}$ and $T_{z_0}$ using $\mathscr{L}_r^C,\mathscr{L}_r$, $S$ and $S^C$.
12:      for $(x_0,y_0,z_0)\in\mathscr{L}_{r,z_0}^C$ do
13:         Extract the context vector $\mathscr{C}_0$ from $I_{z_0+1},I_{z_0},I_{z_0-1},I_{z_0-2}$
14:         Append $\mathscr{C}_0$ to $\mathscr{L}_{\mathscr{C}}$
15:         Update $I_{z_0}\leftarrow T_{z_0}(x_0,y_0,z_0)$
16:      end for
17:      Pass all contexts in $\mathscr{L}_{\mathscr{C}}$ to NN in parallel to obtain $\mathscr{L}_P=NN(\mathscr{L}_{\mathscr{C}})$
18:      for $(x_0,y_0,z_0)\in\mathscr{L}_{r,z_0}^C$ do
19:         Encode the occupancy $O(x_0,y_0,z_0)$ using $P(O(x_0,y_0,z_0)=1)$ from $\mathscr{L}_P$
20:      end for
21:   end for
22: end for
```

$3\times5\times5=75$ elements. In fN2, context region does not include the $z=z_0-2$ and $z=z_0+1$ planes hence $\mathscr{C}_0$ has $2\times5\times5=50$ elements. In fN3, $w=3$ hence $\mathscr{C}_0$ has $4\times3\times3=36$ elements. In fN4, the number of neurons in the final layer is 1 instead of 2 and the output probability is defined as

$$P(O(x_0,y_0,z_0))=\frac{1}{1+e^{-\alpha}} \tag{4.24}$$

Table 4.3 Average Rate [Bpov] results on point clouds from CAT1A

	Bitdepth	**G-PCC**	**BVL**	**fNNOC**	**NNOC**
basketball player	11	0.885	0.852	*0.6908*	**0.5934**
dancer	11	0.876	0.826	*0.6907*	**0.5751**
facade 00064	11	*1.1969*	1.3331	1.2216	**1.1053**
queen	10	*0.7817*	0.7883	0.7573	**0.6897**
redandblack	10	1.1055	1.0418	*0.8854*	**0.7353**
loot	10	0.9818	0.8991	*0.7615*	**0.5989**
Average		0.971	0.957	*0.8616*	**0.7163**

Table 4.4 Ablation Study: Average Rate [Bpov] results on point clouds from CAT1A

	fNNOC	**fN1**	**fN2**	**fN3**	**fN4**	**fN5**
length of $\mathscr{C}_0$	100	75	50	36	100	100
# of neurons by layer	(200,2)	(150,2)	(100,2)	(72,2)	(200,1)	(200,200,2)
basketball player(11)	**0.6908**	0.8924	1.2689	0.9098	0.6945	0.7083
dancer(11)	**0.6907**	0.8874	1.2443	0.8931	0.6936	0.7037
facade 00064(11)	1.2216	1.3888	1.4973	1.3102	**1.2098**	1.2291
queen(10)	**0.9196**	1.1327	1.4814	1.1773	0.9309	0.9390
redandblack(10)	0.8854	1.107	1.514	1.1009	0.8932	**0.8738**
loot(10)	0.7615	0.975	1.3785	0.9798	0.7761	**0.7557**
Average	**0.8616**	1.0639	1.3974	1.0619	0.8663	0.8683

where α is the output value of the single output neuron. In fN5, the neural network has an additional hidden layer with 200 elements. For all of the variants, number of hidden layer elements are set to be twice the length of the input context $\mathscr{C}_0$. Comparing the results for fNNOC and its variants in Table 4.4, it is observed that, none of the variants performs consistently better than fNNOC.

Table 4.5 Durations of encoding and decoding with NNOC and fNNOC

			NNOC		fNNOC	
P. Cloud	**Bitdepth**	**Nr. of Pts**	**ENC**	**DEC(CPU)**	**ENC**	**DEC**
phil-0000	9	371k	35s	7m 38s	26s	27s
loot-1000	10	784k	1m 23s	20m 41s	1m	1m 6s
basket. player	11	2.926M	5m 14s	68m 3s	3m 42s	3m 53s

Table 4.6 Breakdown of durations of encoding and decoding with NNOC and fNNOC (Per-frame average over the loot sequence)

	NNOC		fNNOC	
Time Breakdown	**ENC**	**DEC(CPU)**	**ENC**	**DEC**
Total time	1m 16s	25m 18s	1m 4s	1m 8s
Neural Network	10s	20m	3s	2s
Arithmetic Coding	15s	45s	14s	21s
Remaining operations	51s	4m 33s	47s	45s

4.3 SeqNOC: Optimizing a CNN Model for Compressing a Sequence of Point Clouds

In the NNOC scheme, which was described in Sec. 4.2, is trained a NN model using contexts and corresponding statistics collected from a training set of point clouds, which then becomes a part of the encoder and the decoder. The model architecture and the learned parameters are not transmitted since they are assumed to remain the same. The probability model which is obtained by using a specific training set is used to compress any point cloud. In this approach, the point cloud to be compressed needs to obey similar probability distributions as the training set, such that good compression can be achieved. Because of this, the compression performance of NNOC is dependent on the training data.

Alternatively, one can optimize a NN model on the point cloud to be com-

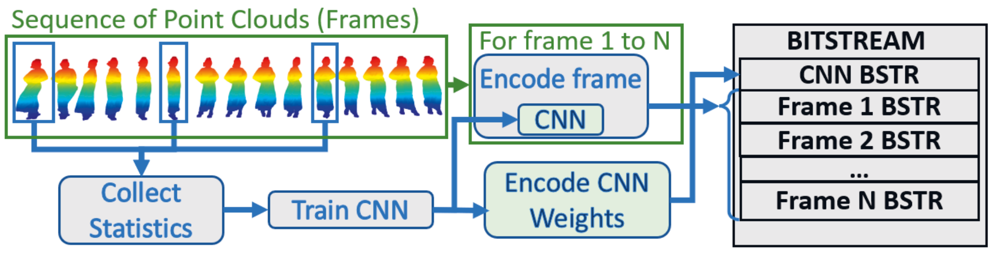


Figure 4.21 Overview of the proposed SeqNOC point cloud sequence encoder.

pressed. In such a scenario, the model should be transmitted as well, since it would be different for each point cloud. In NNOC and fNNOC, the NN model contained 20602 trainable parameters where each parameter is a 32 bit floating point number. Thus, assuming the model structure is known to the decoder, transmitting the model would consume 82 kB which would be a huge cost for a single point cloud. Moreover, the optimization of the NN, which usually is a time-consuming process, becomes in this case a part of the encoding, contributing to the encoding time. Therefore, optimizing such a NN model for the point cloud to be compressed, does not seem to be a feasible approach.

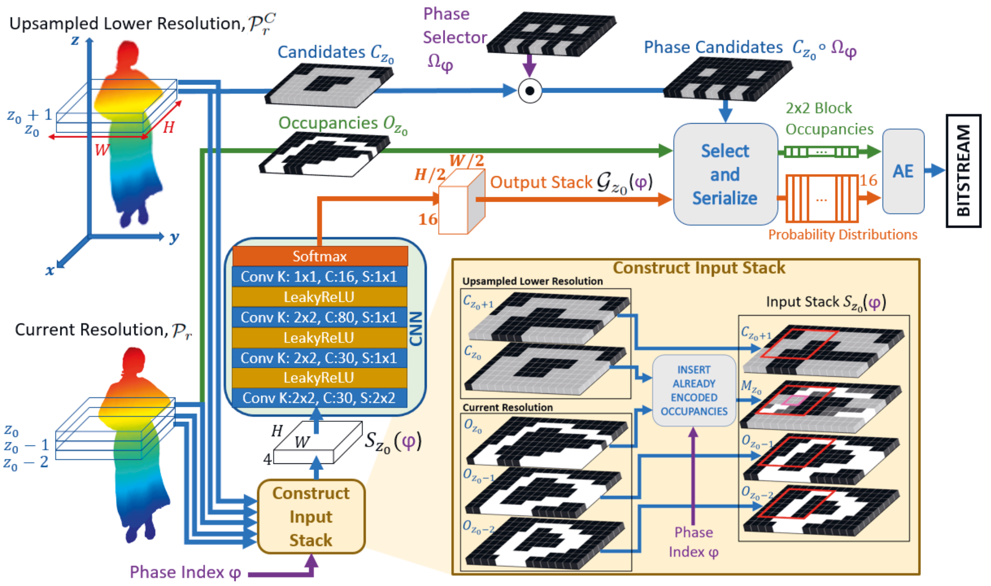
On the other hand, for several practical usage scenarios for point clouds, it is required to transmit a sequence of point clouds rather than a single point cloud which is sometimes also referred to as dynamic point clouds. Unlike the single point cloud case, for a sequence of point clouds, optimizing and transmitting a NN model can be rather feasible if the sequence is sufficiently long.

In Publication V, we introduce a scheme for compressing a sequence of point clouds which we call SeqNOC, which optimizes a NN model on the sequence to be compressed and the optimized NN parameters are transmitted as a header in the bitstream. An overview of SeqNOC is depicted in Fig. 4.21. The NN structure is known to the decoder hence it is not transmitted. Here we refer to the point clouds in a sequence as frames. Using the same probability model for all franes in a sequence is sensible because the frames in a sequence usually contain similar contexts obeying similar probability distributions.

Unlike the case in NNOC, in the proposed SeqNOC method, NN training is a part of the encoding. SeqNOC encoder consists of 3 stages. The first stage is the collection of training data. Training data are collected from a number of equidistant point clouds taken from the sequence starting from the first frame, which we call the training frames. From statistical perspective, it is expected that the compression ratio improves as we use a larger amount of training data and if the compression performance were to be the only concern, it would be reasonable to include the entire sequence in the training data. However, from a practical perspective, besides compression performance, we need to consider also the encoding time and the memory consumption. Having more training frames may increase both the time spent for training data collection and the time spent during training. Both of these times are considered as parts of the total encoding time. Therefore, we train the network using data collected from only a few training frames.

Encoding of a frame in SeqNOC proceeds in a similar way to fNNOC with the major difference being that the neural network stage is formulated as a CNN which allows for a much faster implementation. Moreover, instead of encoding the occupancy of each voxel in one symbol as was the case in fNNOC, here we encode the occupancy of 2×2 groups of candidate voxels. That is, for each 2×2 block of voxels, the CNN estimates a 16-element probability mass function (pmf) vector $\mathbf{G}(m, n)$ where each element corresponds to a certain 2×2 occupancy pattern. Around the 2×2 block of voxels, the context region is a $6 \times 6 \times 4$ block and it is implicit in the structure of the CNN. In other words, for an input stack having shape $6 \times 6 \times 4$, the CNN output is a single pmf vector. The last dimension which has size 4 is the channel dimension and the CNN consists of 2D convolutional layers sliding over the other two dimensions. The number of output channels is 16 where each channel corresponds to the predicted probability of occurence of an occupancy pattern for a given context.

Encoding of a single point cloud (frame) in SeqNOC is described in Algorithm 7. Algorithm 7 proceeds in a similar way to Algorithms 2 and 4 from lines 1 to 7. The encoding of occupancies in a 2D section $z = z_0$ is achieved in four phases $\varphi = 1, ..., 4$ as summarized in lines 10-18. Each phase φ is associated with a different so-called phase selector image Ω_φ which is a binary image. Using Ω_φ, the phase candidates image $\Omega_{\varphi,C}$ is generated which is also a binary image and it determines the candidates to be encoded at phase φ. Operations at a single phase φ are illustrated in Fig. 4.22. At each phase φ, the CNN is called once with a different input stack denoted $S_{z_0}(\varphi)$. $S_{z_0}(\varphi)$ is an ordered set of 4 images $O_{z_0-2}, O_{z_0-1}, M_{z_0}(\varphi), C_{z_0+1}$ which carry the context information for the candidate voxels in the $z = z_0$ plane. What differs in $S_{z_0}(\varphi)$ between phases is the mixing image $M_{z_0}(\varphi)$. It is a four-level image where the levels



Upsampled Lower Resolution, $\mathcal{P}_r^C$
Candidates C_{z_0}
Phase Selector Ω_φ
Phase Candidates $C_{z_0} \circ \Omega_\varphi$
Occupancies O_{z_0}
2x2 Block Occupancies
Select and Serialize
AE
BITSTREAM
Output Stack $\mathcal{G}_{z_0}(\varphi)$
Probability Distributions
Softmax
Conv K: 1x1, C:16, S:1x1
LeakyReLU
Conv K: 2x2, C:80, S:1x1
LeakyReLU
Conv K: 2x2, C:30, S:1x1
LeakyReLU
Conv K:2x2, C:30, S:2x2
CNN
Current Resolution, $\mathcal{P}_r$
Construct Input Stack
Phase Index φ
Upsampled Lower Resolution
Current Resolution
Input Stack $S_{z_0}(\varphi)$
INSERT ALREADY ENCODED OCCUPANCIES

Figure 4.22 Encoding of a single frame in SeqNOC proceeds in a similar way to NNOC where the current resolution voxel grid is swept along a so-called sweeping dimension. Encoding of a 2D section of the current resolution PC is achieved in four phases. The workflow in a single phase is shown here.

are 0 to 3. A pixel (x,y) in $M_{z_0}(\varphi)$ having value 0 corresponds to a non-candidate voxel in $z = z_0$ plane. If (x,y,z_0) is an unoccupied candidate voxel which is already encoded (in a former phase), $M_{z_0}(\varphi)$ at (x,y) is 1. If (x,y,z_0) is a not yet encoded candidate voxel, $M_{z_0}(\varphi)$ at (x,y) is 2. Finally, if (x,y,z_0) is an occupied candidate voxel, which is already encoded (in a former phase), $M_{z_0}(\varphi)$ at (x,y) is 3. Construction of $M_{z_0}(\varphi)$ can be expressed in a single equation as

$$M_{z_0}(\varphi) = 2C_{z_0} \circ (1 - \Omega^s_{\varphi-1}) + (2O_{z_0}(\varphi) + 1) \circ \Omega^s_{\varphi-1}, \tag{4.25}$$

where $\Omega^s_{\varphi-1}$ represents the already processed voxels at $z = z_0$ after phase $\varphi - 1$, i.e, $\Omega^s_{\varphi-1} = \Omega_1 \vee \ldots \vee \Omega_{\varphi-1}$ ($\vee$ denotes the element-wise OR operation). The four levels in $M_{z_0}(\varphi)$ are ordered in such a way that the value of a pixel being higher corresponds to a stronger occupancy status. A non-candidate voxel has a weaker occupancy status than a candidate voxel which is discovered to be unoccupied because, for the non candidate voxel, not only the voxel itself but also its siblings, which decend from the same parent voxel in the lower resolution, are known to be unoccupied. The order of four levels is important because there is a single CNN model which applies the

Algorithm 7: Encoding of a frame (point cloud) in SeqNOC

```
Require: A point cloud $\mathscr{P}_R$ with resolution $R$ bits/dimension
 1: Construct lower resolution point clouds $\mathscr{P}_{R-1},\dots,\mathscr{P}_2$ representing the full
    nodes at octree depth levels $R-1,\dots,2$.
 2: Encode $\mathscr{P}_2$ in 64 bits
 3: Encode iteratively $\mathscr{P}_3$ to $\mathscr{P}_R$ as follows:
 4: for $r = 3,\dots,R$ do
 5:     Generate for each point $(x_{r-1}, y_{r-1}, z_{r-1}) \in \mathscr{P}_{r-1}$ 8 candidate voxels in the
        resolution $r$, resulting in the set of candidate points $\mathscr{P}_r^C$
 6:     Traverse the voxel grid along the sweeping dimension $z$ and encode each 2D
        section $z = z_0$ as follows:
 7:     for $z_0 = 0 \dots 2^r - 1$ do
 8:         Construct the occupancy images $O_{z_0}, O_{z_0-1}, O_{z_0-2}$ using $\mathscr{P}_r$
 9:         Construct the candidacy images $C_{z_0}, C_{z_0+1}$ using $\mathscr{P}_r^C$
10:         for $\varphi = 1 \dots 4$ do
11:             Construct the phase candidates image $\Omega_{\varphi,C}$ as $\Omega_{\varphi,C} = C_{z_0} \circ \Omega_\varphi$
12:             Construct the mixing image
                $M_{z_0}(\varphi) = 2C_{z_0} \circ (1 - \Omega^s_{\varphi-1}) + (2O_{z_0}(\varphi) + 1) \circ \Omega^s_{\varphi-1}$
13:             Construct the input stack $S_{z_0}(\varphi)$ using $O_{z_0-2}, O_{z_0-1}, M_{z_0}(\varphi), C_{z_0+1}$
14:             Run $CNN(S_{z_0}(\varphi))$ to generate the output stack $\mathscr{G}_{z_0}(\varphi)$.
15:             From $O_{z_0}$, collect the $2 \times 2$ occupancy patterns $Q_{m,n}$ of the phase
                candidates using $\Omega_{\varphi,C}$
16:             From $\mathscr{G}_{z_0}$, collect the pmf vectors $\mathbf{G}(m,n)$ using $\Omega_{\varphi,C}$
17:             Encode $Q_{m,n}$ of the phase candidates using $\mathbf{G}(m,n)$ (The arithmetic
                encoding step)
18:         end for
19:     end for
20: end for
```

same weights to contexts coming from different phases.

4.3.1 Experimental Results

Bpp (bits-per-point, same as bpov) results optained with SeqNOC are compared with other methods in Table 4.7. Since the results for Silhouette 4D with Context Selection (S4DCS) [70] was reported for 100 consecutive frames of each benchmark sequence starting from the second frames, we also report for the same 100 frames, for the sake of fairness in comparison. From Table 4.7, it is evident that SeqNOC performs significantly better than the other methods which are not trained on a generic dataset. The only inferior result was obtained for the Soldier sequence for which S4DCS [70] yields 0.65 bpov.

In Table 4.7 and 4.8 we refer to the SeqNOC results obtained with what we call the default configuration as SeqNOC (default). The default configuration has four phases as described in Section 4.3 and the number of training frames (PCs) is set to 5. In Table 4.8, the default configuration is compared with two other configurations which are referred to as SeqNOC-10 and SeqNOC-SP. Each of these two configurations differ in only one aspect from the default configuration so that we can observe the impact of those aspects. SeqNOC-10 employs 10 training frames and SeqNOC-SP has only one phase instead of four phases just like fNNOC (except fNNOC encodes each voxel occupancy as a separate symbol whereas all SeqNOC versions encode 2×2 groups of neighboring voxels). From Table 4.8, it is evident that employing 10 training frames (as in SeqNOC-10) instead of 5 (as in SeqNOC (default)), improves the bitrate performance significantly. However, it also increases the encoding times ($t_e[s]$), since the time required for training statistics collection and the time required for training both become longer. The differences in decoding times ($t_d[s]$) between SeqNOC and SeqNOC-10 are not to be explained by the change in the number of training frames because this is a change that is related to the encoding procedure only. Comparing the results for SeqNOC (default) and SeqNOC-SP in Table 4.8, one can observe that if a single phase φ of encoding is employed (as in SeqNOC-SP) instead of four phases (as in SeqNOC (default)), the bitrates are significantly worse. However, the single phase configuration SeqNOC-SP has the advantage of faster encoding and decoding since the estimation of the distributions for all candidates in a 2D section is performed simultaneously. SeqNOC (default) appears to offer a reasonable balance between speed and bitrate.

Table 4.7 Comparing Average Bitrates of SeqNOC (the default configuration) with Other Recent Methods

	TMC13 [52]	**DD** [62]	**P(Full)** [18]	**S4DCS** [70]	**SeqNOC**
Andrew	1.14	1.12	1.37	0.95	**0.85**
David	1.07	1.06	1.31	0.94	**0.78**
Phil	1.17	1.14	1.42	1.02	**0.86**
Ricardo	1.09	1.04	1.34	0.90	**0.79**
Sarah	1.07	1.07	1.37	0.92	**0.80**
Average$_{\text{MVUB}}$	1.11	1.09	1.36	0.95	**0.82**
LongDress	1.02	0.95	1.13	0.88	0.70
Loot	0.96	0.92	1.02	0.84	**0.66**
RedAndBlack	1.08	1.02	1.23	0.94	**0.77**
Soldier	1.03	0.96	0.85	**0.65**	0.70
Average$_{\text{8i}}$	1.02	0.96	1.06	0.83	**0.71**

Table 4.8 Bitrates [bpp], Encoding Times ($t_e[s]$) and Decoding Times ($t_d[s]$) for three different configurations of SeqNOC

	SeqNOC (default)			**SeqNOC-10**			**SeqNOC-SP**		
	bpp	$t_e[s]$	$t_d[s]$	bpp	$t_e[s]$	$t_d[s]$	bpp	$t_e[s]$	$t_d[s]$
Andrew	0.85	8.4	7.0	**0.83**	9.5	7.1	0.90	**4.9**	**2.4**
David	0.78	10.0	9.0	**0.77**	11.9	9.0	0.78	**6.4**	**2.6**
Phil	0.86	10.0	8.0	**0.84**	12.5	8.1	0.88	**6.6**	**2.2**
Ricardo	0.79	9.2	6.5	**0.77**	11.8	6.6	0.80	**4.6**	**2.0**
Sarah	0.80	9.2	8.5	**0.78**	10.9	7.7	0.79	**5.9**	**2.2**
Average$_{\text{MVUB}}$	0.82	9.4	7.8	**0.80**	11.4	7.7	0.83	**5.7**	**2.3**
Longdress	0.70	10.9	9.0	**0.69**	14.2	9.1	0.75	**6.5**	**2.7**
Loot	0.66	10.0	9.0	**0.64**	16.4	9.0	0.70	**6.9**	**2.6**
Redandblack	0.77	9.8	8	**0.75**	16.0	8.5	0.81	**7.6**	**2.5**
Soldier	**0.70**	12.7	10.0	**0.70**	13.4	10.2	0.77	**6.4**	**3.0**
Average$_{\text{8i}}$	0.71	10.8	9.1	**0.70**	15.0	9.2	0.76	**6.9**	**2.7**

5 CONCLUSIONS AND SUMMARY

In this final chapter, the contributions of this dissertation work to compression of point cloud geometry and compression of light field images are summarized. Moreover, we present our main conclusions drawn from the studies which were previously reported in five publications. In the previous chapters, we went over these five publications which are also attached to the end of this document.

In Publication I, our contributions to light field processing were two-folds. Initially, we developed a CNN structure for estimating disparities for the corner views of a light field image. The proposed CEPINET scheme was obtained by adapting a method in the literature called EPINET [79], which estimates a disparity map associated to the center view of a light field image, to the case of corner views. Corner views are the most extreme locations in a camera array and therefore the disparity estimation is arguably more difficult. The second contribution was the development of a lossless light field compression scheme which uses multiple disparity estimations. By using the disparities associated with several different reference views in conjunction, one is able to predict any target view color image in the light field with highly compressible prediction errors.

Publications II and III were dealing with point cloud compression (PCC) based on a novel representation which we refer to as *bounding volumes*. In Publication II is presented a lossy PCC method called LBV, whereas in Publication III a lossless PCC method called BVL is presented. In both LBV and BVL, the initial step is to define a bounding volume using two depth maps obtained by projecting the input point cloud in two opposite directions. The depth maps are efficiently compressed with a method called CERV [81] and they represent the "outermost" points in the point cloud which are then taken as anchors to describe the remaining points in the point cloud in an efficient manner. This initial representation of a point cloud is quite different from the well established and commonly used octree representation. As evident from the experimental results presented in Chapter 4, the novel BV approach is performing favorably with certain types of point clouds such as those involving an entire human figure geometry whereas it is underperforming the G-PCC [23] standard codec with more complex point clouds.

In Publication IV, we proposed NNOC, which is a novel learning based lossless geometry coding scheme operating on the octree representation. NNOC uses a simple MLP i.e. a fully-connected NN for estimating the probability of voxel occupancies given 3D contexts derived from two consecutive levels of resolution (in the literature, also referred to as levels of detail). The NN model parameters are learned from a set of training point clouds. Since the compression performance is expected to depend on the choice of training PCs, in our experiments, we used a training set of PCs that is very similar to that which was used in VoxelDNN [55] which we regarded as a competitor method. The compression performance of NNOC is highly competitive and running (encoding/decoding) times for the fast version fNNOC are in an acceptable level considering that the implementation was done in a high-level programming environment unlike TMC13 [52] implementation of the G-PCC standard codec [23] or our C implementation of BVL. Yet still, the fact that the reported results depend on the choice of training set, causes NNOC to be a less general method when compared to the methods which do not involve machine learning such as G-PCC or BVL. Furthermore, NNOC is an intra-frame codec whereas several practical applications of PCs require compression of a sequence of point clouds instead of a single frame. Motivated by these critical viewpoints, in Publication V, we proposed a sequence coder called SeqNOC which is similar to NNOC in the way it handles the octree representation. SeqNOC is fundamentally different from NNOC in the way the probabilities are modeled, where for a sequence to be compressed, a CNN is trained from scratch as part of the encoding procedure, using the statistics collected from the sequence itself. CNN training needs to take much shorter than usual training times for similar CNN architectures in the literature such that the per-frame encoding time is in the same level as the competitor methods. The decoding times of SeqNOC are even shorter since decoding does not involve training data collection and training.

To sum up, in this dissertation, several methods for lossy and lossless compression of immersive data were developed where the emphasis was on the point cloud geometry compression side.

REFERENCES

[1] R. Abbasi, A. K. Bashir, H. J. Alyamani, F. Amin, J. Doh and J. Chen. Lidar Point Cloud Compression, Processing and Learning for Autonomous Driving. *IEEE Transactions on Intelligent Transportation Systems* (2022), 1–18. DOI: `10.1109/TITS.2022.3167957`.

[2] M. D. Adams. The JPEG-2000 still image compression standard. (2001).

[3] S. Agrawal, A. Simon, S. Bech, K. Bæntsen and S. Forchhammer. Defining Immersion: Literature Review and Implications for Research on Audiovisual Experiences. *Journal of the Audio Engineering Society* 68.6 (June 2020), 404–417. DOI: `https://doi.org/10.17743/jaes.2020.0039`.

[4] E. Alexiou, K. Tung and T. Ebrahimi. Towards neural network approaches for point cloud compression. *Applications of Digital Image Processing XLIII*. Vol. 11510. SPIE. 2020, 18–37.

[5] P. Astola and I. Tabus. Wasp: Hierarchical warping, merging, and sparse prediction for light field image compression. *2018 7th European Workshop on Visual Information Processing (EUVIP)*. IEEE. 2018, 1–6.

[6] M. Bleyer, C. Rhemann and C. Rother. Extracting 3D scene-consistent object proposals and depth from stereo images. *European Conference on Computer Vision*. Springer. 2012, 467–481.

[7] M. Cagnazzo, M. Antonini and M. Barlaud. Mutual information-based context quantization. *Signal Processing: Image Communication* 25.1 (2010), 64–74.

[8] E. Camuffo, D. Mari and S. Milani. Recent advancements in learning algorithms for point clouds: an updated overview. *Sensors* 22.4 (2022), 1357.

[9] C. Cao, M. Preda and T. Zaharia. 3D point cloud compression: A survey. *The 24th International Conference on 3D Web Technology*. 2019, 1–9.

[10] 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* 109.9 (2021), 1537–1558.

[11] C. Chao, C. Tulvan, M. Preda and T. Zaharia. Skeleton-based motion estimation for Point Cloud Compression. *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*. IEEE. 2020, 1–6.

[12] J. Chen, J. Hou and L.-P. Chau. Light field compression with disparity-guided sparse coding based on structural key views. *IEEE Transactions on Image Processing* 27.1 (2017), 314–324.

[13] Y. Cui, R. Chen, W. Chu, L. Chen, D. Tian, Y. Li and D. Cao. Deep learning for image and point cloud fusion in autonomous driving: A review. *IEEE Transactions on Intelligent Transportation Systems* (2021).

[14] E. d'Eon, B. Harrison, T. Myers and P. A. Chou. 8i voxelized full bodies-a voxelized point cloud dataset. *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document WG11M40059/WG1M74006* 7 (2017), 8.

[15] R. L. De Queiroz and P. A. Chou. Compression of 3D point clouds using a region-adaptive hierarchical transform. *IEEE Transactions on Image Processing* 25.8 (2016), 3947–3956.

[16] E. Dib, M. Le Pendu, X. Jiang and C. Guillemot. Local low rank approximation with a parametric disparity model for light field compression. *IEEE Transactions on Image Processing* 29 (2020), 9641–9653. DOI: `10.1109/TIP.2020.3029655`.

[17] S. Forchhammer, X. Wu and J. D. Andersen. Optimal context quantization in lossless compression of image data sequences. *IEEE Transactions on Image Processing* 13.4 (2004), 509–517.

[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* 29 (2019), 313–322.

[19] D. C. Garcia and R. L. de Queiroz. Context-based octree coding for point-cloud video. *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2017, 1412–1416.

[20] D. C. Garcia and R. L. de Queiroz. Intra-frame context-based octree coding for point-cloud geometry. *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE. 2018, 1807–1811.

[21] T. Georgiev, Z. Yu, A. Lumsdaine and S. Goma. Lytro camera technology: theory, algorithms, performance analysis. *Multimedia content and mobile devices*. Vol. 8667. International Society for Optics and Photonics. 2013, 86671J.

[22] C. Godard, O. Mac Aodha and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, 270–279.

[23] 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* 9 (2020).

[24] F. Groh, P. Wieschollek and H. Lensch. Flex-convolution (million-scale point-cloud learning beyond grid-worlds). *arXiv preprint arXiv:1803.07289* (2018).

[25] A. F. Guarda, N. M. Rodrigues and F. Pereira. Point cloud coding: Adopting a deep learning-based approach. *2019 Picture Coding Symposium (PCS)*. IEEE. 2019, 1–5.

[26] S. Gumhold, Z. Kami, M. Isenburg and H.-P. Seidel. Predictive point-cloud compression. *ACM SIGGRAPH 2005 Sketches*. 2005, 137–es.

[27] K. Honauer, O. Johannsen, D. Kondermann and B. Goldluecke. A dataset and evaluation methodology for depth estimation on 4D light fields. *Asian Conference on Computer Vision*. Springer. 2016, 19–34.

[28] J. Hou, J. Chen and L.-P. Chau. Light field image compression based on bi-level view compensation with rate-distortion optimization. *IEEE Transactions on Circuits and Systems for Video Technology* 29.2 (2019), 517–530. DOI: `10.1109/TCSVT.2018.2802943`.

[29] X. Hu, J. Shan, Y. Liu, L. Zhang and S. Shirmohammadi. An adaptive two-layer light field compression scheme using GNN-based reconstruction. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 16.2s (2020), 1–23.

[30] L. Huang, S. Wang, K. Wong, J. Liu and R. Urtasun. Octsqueeze: Octree-structured entropy model for lidar compression. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, 1313–1323.

[31] X. Huang, P. An, Y. Chen, D. Liu and L. Shen. Low bitrate light field compression with geometry and content consistency. *IEEE Transactions on Multimedia* (2020).

[32] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE* 40.9 (1952), 1098–1101.

[33] 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* 36.3 (2019), 118–123.

[34] H.-G. Jeon, J. Park, G. Choe, J. Park, Y. Bok, Y.-W. Tai and I. So Kweon. Accurate depth map estimation from a lenslet light field camera. *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, 1547–1555.

[35] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz and E. Steinbach. Real-time compression of point cloud streams. *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, 778–785.

[36] M. Krivokuća, P. A. Chou and M. Koroteev. A volumetric approach to point cloud compression–part II: geometry compression. *IEEE Transactions on Image Processing* 29 (2020), 2217–2229. DOI: `10.1109/TIP.2019.2957853`.

[37] M. Li, W. Zuo, S. Gu, D. Zhao and D. Zhang. Learning convolutional networks for content-weighted image compression. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, 3214–3223.

[38] Y. Li, L. Ma, Z. Zhong, F. Liu, M. A. Chapman, D. Cao and J. Li. Deep learning for lidar point clouds in autonomous driving: A review. *IEEE Transactions on Neural Networks and Learning Systems* 32.8 (2020), 3412–3432.

[39] D. Liu, P. An, R. Ma, W. Zhan, X. Huang and A. A. Yahya. Content-based light field image compression method with Gaussian process regression. *IEEE Transactions on Multimedia* 22.4 (2020), 846–859. DOI: `10.1109/TMM.2019.2934426`.

[40] D. Liu, L. Wang, L. Li, Z. Xiong, F. Wu and W. Zeng. Pseudo-sequence-based light field image compression. *2016 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*. IEEE. 2016, 1–4.

[41] X. Liu, M. Yan and J. Bohg. MeteorNet: Deep learning on dynamic 3D point cloud sequences. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2019.

[42] Y. Liu, B. Fan, G. Meng, J. Lu, S. Xiang and C. Pan. Densepoint: Learning densely contextual representation for efficient point cloud processing. *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, 5239–5248.

[43] Z. Liu, S. Zhou, C. Suo, P. Yin, W. Chen, H. Wang, H. Li and Y.-H. Liu. Lpd-net: 3d point cloud learning for large-scale place recognition and environment analysis. *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, 2831–2840.

[44] C. Loop, Q. Cai, S. O. Escolano and P. A. Chou. Microsoft voxelized upper bodies-a voxelized point cloud dataset. *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document m38673 M* 72012 (2016), 2016.

[45] G. Lu, X. Zhang, W. Ouyang, L. Chen, Z. Gao and D. Xu. An end-to-end learning framework for video compression. *IEEE Transactions on pattern analysis and machine intelligence* (2020).

[46] D. Meagher. Geometric modeling using octree encoding. *Computer graphics and image processing* 19.2 (1982), 129–147.

[47] 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* 27.4 (2016), 828–842.

[48] R. Mekuria, M. Sanna, S. Asioli, E. Izquierdo, D. C. Bulterman and P. Cesar. A 3d tele-immersion system based on live captured mesh geometry. *Proceedings of the 4th ACM Multimedia Systems Conference*. 2013, 24–35.

[49] B. Merry, P. Marais and J. Gain. Compression of dense and regular point clouds. *Proceedings of the 4th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*. 2006, 15–20.

[50] S. Milani. A syndrome-based autoencoder for point cloud geometry compression. *2020 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2020, 2686–2690.

[51] S. Milani. ADAE: Adversarial Distributed Source Autoencoder For Point Cloud Compression. *2021 IEEE International Conference on Image Processing (ICIP)*. 2021, 3078–3082. DOI: `10.1109/ICIP42928.2021.9506750`.

[52] *MPEG Group TMC13*. `https://github.com/MPEGGroup/mpeg-pcc-tmc13`. Accessed: 2020-03-20.

[53] M. U. Mukati and S. Forchhammer. Epipolar plane image-based lossless and near-lossless light field compression. *IEEE Access* 9 (2020), 1124–1136.

[54] M. U. Mukati and S. Forchhammer. EPIC: Context adaptive lossless light field compression using epipolar plane images. *2020 Data Compression Conference (DCC)*. IEEE. 2020, 43–52.

[55] D. T. Nguyen, M. Quach, G. Valenzise and P. Duhamel. Learning-based lossless compression of 3d point cloud geometry. *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2021, 4220–4224.

[56] D. T. Nguyen, M. Quach, G. Valenzise and P. Duhamel. Lossless coding of point cloud geometry using a deep generative model. *IEEE Transactions on Circuits and Systems for Video Technology* 31.12 (2021), 4617–4629.

[57] D. T. Nguyen, M. Quach, G. Valenzise and P. Duhamel. Multiscale deep context modeling for lossless point cloud geometry compression. *2021 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*. IEEE. 2021, 1–6.

[58] T. Ochotta and D. Saupe. Compression of point-based 3D models by shape-adaptive wavelet coding of multi-height fields. *SPBG'04 Symposium on Point - Based Graphics 2004*. Ed. by M. Gross, H. Pfister, M. Alexa and S. Rusinkiewicz. The Eurographics Association, 2004. ISBN: 3-905673-09-6. DOI: `10.2312/SPBG/SPBG04/103-112`.

[59] W. Oropallo, L. A. Piegl, P. Rosen and K. Rajab. Point cloud slicing for 3-D printing. *Computer-Aided Design and Applications* 15.1 (2018), 90–97.

[60] I. K. Park, K. M. Lee et al. Robust light field depth estimation using occlusion-noise aware data costs. *IEEE Transactions on pattern analysis and machine intelligence* 40.10 (2017), 2484–2497.

[61] E. Peixoto, E. Medeiros and E. Ramalho. Silhouette 4D: An inter-frame lossless geometry coder of dynamic voxelized point clouds. *2020 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2020, 2691–2695.

[62] E. Peixoto. Intra-frame compression of point cloud geometry using dyadic decomposition. *IEEE Signal Processing Letters* 27 (2020), 246–250.

[63] J. Peng, C.-S. Kim and C.-C. Jay Kuo. Technologies for 3D mesh compression: A survey. *Journal of Visual Communication and Image Representation* 16.6 (2005), 688–733. ISSN: 1047-3203. DOI: `https://doi.org/10.1016/j.jvcir.2005.03.001`. URL: `https://www.sciencedirect.com/science/article/pii/S1047320305000295`.

[64] R. Pierdicca, M. Paolanti, F. Matrone, M. Martini, C. Morbidoni, E. S. Malinverni, E. Frontoni and A. M. Lingua. Point cloud semantic segmentation using a deep learning framework for cultural heritage. *Remote sensing* 12.6 (2020), 1005.

[65] M. Quach, J. Pang, D. Tian, G. Valenzise and F. Dufaux. Survey on deep learning-based point cloud compression. *Frontiers in Signal Processing* (2022).

[66] M. Quach, G. Valenzise and F. Dufaux. Learning convolutional transforms for lossy point cloud geometry compression. *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2019, 4320–4324.

[67] M. Quach, G. Valenzise and F. Dufaux. Improved deep point cloud geometry compression. *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*. 2020, 1–6. DOI: `10.1109/MMSP48831.2020.9287077`.

[68] Z. Que, G. Lu and D. Xu. VoxelContext-Net: An octree-based framework for point cloud compression. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, 6042–6051.

[69] 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* 25.6 (2018), 739–742.

[70] E. Ramalho, E. Peixoto and E. Medeiros. Silhouette 4D with context selection: lossless geometry compression of dynamic point clouds. *IEEE Signal Processing Letters* 28 (2021), 1660–1664.

[71] J. Rissanen. A universal data compression system. *IEEE Transactions on Information Theory* 29.5 (1983), 656–664. DOI: `10.1109/TIT.1983.1056741`.

[72] J. Rissanen and G. G. Langdon. Arithmetic coding. *IBM Journal of research and development* 23.2 (1979), 149–162.

[73] J. Rossignac. Edgebreaker: connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics* 5.1 (1999), 47–61. DOI: `10.1109/2945.764870`.

[74] R. Schnabel and R. Klein. Octree-based point-cloud compression. *PBG@SIGGRAPH.* 2006, 111–120.

[75] O. Schreer, I. Feldmann, S. Renault, M. Zepp, M. Worchel, P. Eisert and P. Kauff. Capture and 3D video processing of volumetric video. *2019 IEEE International Conference on Image Processing (ICIP).* 2019, 4310–4314. DOI: `10.1109/ICIP.2019.8803576`.

[76] S. Schwarz, G. Martin-Cocher, D. Flynn and M. Budagavi. Common test conditions for point cloud compression. *Document ISO/IEC JTC1/SC29/WG11 w17766, Ljubljana, Slovenia* (2018).

[77] 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* 9.1 (2018), 133–148.

[78] C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal* 27.3 (1948), 379–423.

[79] C. Shin, H.-G. Jeon, Y. Yoon, I. S. Kweon and S. J. Kim. Epinet: A fully-convolutional neural network using epipolar geometry for depth from light field images. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, 4748–4757.

[80] M. Stepanov, M. U. Mukati, G. Valenzise, S. Forchhammer and F. Dufaux. Learning-based lossless light field compression. *IEEE International Workshop on Multimedia Signal Processing (MMSP'2021).* 2021.

[81] I. Tabus, I. Schiopu and J. Astola. Context coding of depth map images under the piecewise-constant image model representation. *IEEE Transactions on Image Processing* 22.11 (2013), 4195–4210.

[82] D. Taubman. High performance scalable image compression with EBCOT. *IEEE Transactions on image processing* 9.7 (2000), 1158–1170.

[83] D. Tian, H. Ochimizu, C. Feng, R. Cohen and A. Vetro. Geometric distortion metrics for point cloud compression. *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2017, 3460–3464.

[84] Y.-J. Tsai, Y.-L. Liu, M. Ouhyoung and Y.-Y. Chuang. Attention-based view selection networks for light-field disparity estimation. *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 07. 2020, 12095–12103.

[85] J. Wang, D. Ding, Z. Li and Z. Ma. Multiscale point cloud geometry compression. *2021 Data Compression Conference (DCC)*. IEEE. 2021, 73–82.

[86] J. Wang, H. Zhu, H. Liu and Z. Ma. Lossy point cloud geometry compression via end-to-end learning. *IEEE Transactions on Circuits and Systems for Video Technology* 31.12 (2021), 4909–4923.

[87] Y. Wang and J. M. Solomon. Deep closest point: Learning representations for point cloud registration. *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, 3523–3532.

[88] Z.-R. Wang, C.-G. Yang and S.-L. Dai. A fast compression framework based on 3D point cloud data for telepresence. *International Journal of Automation and Computing* 17.6 (2020), 855–866.

[89] M. J. Weinberger, J. J. Rissanen and R. B. Arps. Applications of universal context modeling to lossless compression of gray-scale images. *IEEE Transactions on Image Processing* 5.4 (1996), 575–586.

[90] T. A. Welch. A technique for high-performance data compression. *Computer* 17.06 (1984), 8–19.

[91] X. Wen, X. Wang, J. Hou, L. Ma, Y. Zhou and J. Jiang. Lossy geometry compression of 3d point cloud data via an adaptive octree-guided network. *2020 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE. 2020, 1–6.

[92] H. White. Printed English compression by dictionary encoding. *Proceedings of the IEEE* 55.3 (1967), 390–396.

[93] X. Wu, P. A. Chou and X. Xue. Minimum conditional entropy context quantization. *IEEE International Symposium on Information Theory*. Citeseer. 2000, 43–43.

[94] X. Wu and N. Memon. Context-based, adaptive, lossless image coding. *IEEE transactions on Communications* 45.4 (1997), 437–444.

[95] 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).

[96] Y. Xu, W. Zhu, Y. Xu and Z. Li. Dynamic point cloud geometry compression via patch-wise polynomial fitting. *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, 2287–2291.

[97] J. Zhang, X. Zhao, Z. Chen and Z. Lu. A review of deep learning-based semantic segmentation for point cloud. *IEEE Access* 7 (2019), 179118–179133.

[98] Z. Zhang, Y. Dai and J. Sun. Deep learning based point cloud registration: an overview. *Virtual Reality & Intelligent Hardware* 2.3 (2020), 222–246.

[99] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory* 23.3 (1977), 337–343.

[100] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory* 24.5 (1978), 530–536.

PUBLICATIONS

PUBLICATION

I

Corner view disparity estimation for lossless light field compression
E. C. Kaya and I. Tabus

*2019 1st European Light Field Imaging Workshop (ELFI)*2019

CORNER VIEW DISPARITY ESTIMATION FOR LOSSLESS LIGHT FIELD COMPRESSION

Emre Can Kaya, Ioan Tabus

Tampere University
Computing Sciences Unit
Tampere, Finland

ABSTRACT

In this paper we study disparity estimation for high density camera arrays using convolution neural networks (CNN) with the goal of achieving an overall description of all disparity images needed in the warping process for light field compression. Furthermore, we present a lossless compression scheme that makes use of the obtained disparity estimates. The scheme provides random access to individual views, as opposed to most published lossless light field compression, which require the simultaneous decoding of all views. Following similar steps as in EPINET, which is a recently published CNN based estimator of the disparity at the center view, we design a CNN for estimating the disparity of the views in corner positions. EPINET uses several data augmentation techniques through rotation and flipping of light field and we study similar transformations of the light field that provide data augmentation when estimating corner views disparities. The performance of the estimated disparities is evaluated first in terms of the traditional mean square error and percentage of pixels above a certain threshold. Additionally, we validate the quality of the disparity estimates in terms of their successful usage for the warping stage in a lossless color view compression scheme. The lossless compression achieved by the estimated disparities is shown to be close to the lossless compression achieved when using ground truth disparities. Codes are available at *https://github.com/marmus12/CornerView*.

I. INTRODUCTION

Disparity estimation is an active area of research that has several applications in computer vision. Disparity maps contain the essential information about how the consecutive views in a certain angular direction of the camera array are related, making them very useful for light field compression.

Recently, CNN based methods proved to be successful in several light field processing tasks [1], [2]. EPINET [3] is a CNN model for estimating a disparity map of a light field image. The disparity map generated by EPINET corresponds to the center view which is the disparity between the center view and its closest right horizontal neighbour view. In the following, we use the term corner view disparity to refer to the disparity between the corner view and its closest right horizontal neighbor, or equivalently the negative disparities to the left neighbor. EPINET has a multi-stream architecture which efficiently combines the information coming from different view stacks. In the network, several stacks of views are considered, extracting information from the epipolar plane images at different angular directions in the camera array. The four input view stacks are intersecting at the center view. For estimating the center view disparity with EPINET, it is needed that input views from around the center view are available. Here, we present a different scheme, in which the disparities can be estimated for the most extreme locations in a camera array, namely for the corner views. It should be noted that once the corner and center view estimates are available, reliable estimates can be obtained for any desired view, since reliable information exists for displacements on various angular directions.

II. CORNER VIEW DISPARITY ESTIMATION

The proposed CNN architecture for estimating a corner view disparity map is depicted on Fig. 1. We call this architecture CEPINET, short for Corner EPINET. This architecture is derived from the original EPINET architecture by replacing the 45° and 135° diagonal subnetworks with a single diagonal subnetwork, since in CEPINET there is a single diagonal stack. Moreover, the number of filters in the merge network is reduced to 210 from 280 to remain consistent with the number of input stacks. The CEPINET network encompasses 2.96 million trainable parameters, while the number for EPINET is 5.12 millions.

In EPINET, data augmentation of the light field through rotation is performed by rotating the light field image around the center view. Therefore, the center view remains at the center after rotation. Hence, the resulting light field is still a valid input for the network that estimates the center disparity. In CEPINET, the situation is different in the sense that whenever the light field is rotated about the center of view array, the position of every corner changes. In order to train a single network, which by proper preprocessing can estimate any of the four corners, we introduce rules for forming the input stacks, with the convention that the disparity in the upper left corner has to be estimated. According to this convention, when estimating the other three corners, the input light field is rotated by multiples of 90 degrees to bring the corner view that is being estimated to the upper left position. On the other hand, we also apply transposition of the light field as an augmentation which also yields a valid

Table I. MSE*10^3 with Town as Validation

View	Train Mean	Vinyl	Kitchen	Museum	Town
NW	**7.17**	97.13	141.73	65.34	3.67
NE	7.55	**79.11**	**138.28**	**60.66**	**2.95**
SW	7.69	124.50	157.88	83.01	6.03
SE	9.21	156.59	153.87	107.58	5.14
Center	15.13	112.09	164.57	116.68	3.30

Table II. MSE*10^3 with Greek as Validation

View	Train Mean	Vinyl	Kitchen	Museum	Greek
NW	22.74	114.53	148.17	66.88	**95.12**
NE	20.65	**90.33**	**138.69**	82.28	206.86
SW	21.43	115.13	175.99	**52.73**	250.77
SE	23.79	126.98	152.06	65.29	272.86
Center	**12.00**	117.70	154.84	76.05	118.61

light field. Upper left corner view remains in the same place after transposing the light field, thus not violating our convention. In summary, eight different training samples are obtained by rotating and transposing a light field image: two for each corner (original and transpose). The other augmentation strategies such as color scaling are kept the same as in EPINET.

III. LOSSLESS COMPRESSION USING CORNER AND CENTER VIEW DISPARITIES

Once the disparity maps for the corner views and the center view are obtained, any target view can be reconstructed with small error by warping reference images and combining the warped images. The proposed lossless light field compressor, dubbed here LLFC, is depicted in Fig. 2. The scheme is close in spirit to the disparity based and region based plenoptic image compression from [4], but has a more refined combining of warping from several references, based on a region based best performance switch, as described next. Unlike the previous work in [4], LLFC makes use of the corner view disparity estimations in addition to the center view for predicting one general position view.

For each target view, a disparity image is generated by warping the closest reference's disparity to the target location. The target disparity is quantized and divided into connected regions. For each region, one marks in an image, called *best reference labels image*, the index of that warped image which yields the smallest MSE over the region. The best reference labels image is constructed for each target and it is used in conjunction with the warped references to predict the target view. Best reference labels image has all elements as integers 1 to 5 (indexing the winner, out of 4 corners and center). First the side information is encoded: 5 reference disparity maps, 5 reference color images, and 1 best reference labels image for each target. Furthermore, since we want to perform lossless compression, residual images for each target are also transmitted. Best reference labels images are very sparse allowing them to be compressed efficiently. We compress reference color, disparity images and residual images with lossless JPEG 2000 [5] prior to transmission. The best reference labels image can be encoded either with JPEG 2000 or CERV [6]. We present results for both cases.

IV. EXPERIMENTAL WORK

In this section, we present experimental results for corner view disparity estimation and lossless compression. Our training and validation set consists of 13 samples from the HCI Benchmark [7]. Three samples that contain reflective surfaces (*vinyl*, *kitchen* and *museum*) are chosen as test samples. Since reflective surface disparities are hard to estimate, MSEs for these samples are much higher than the validation samples in Tables I-II.

We train 2 CEPINET and 2 corresponding baseline EPINET models, each time leaving out 1 sample (first *greek*, then *town*) for validation so that the training set consists of 12 samples. From each light field image, 8 different training samples are obtained by rotating the light field with multiples of 90 degrees and taking the transpose. These 8 different cases are randomly sampled during training. One training batch consists of 48 randomly chosen multi-scale patches with size 25x25. Learning rate is initially set as 10^{-4} and it is dropped by a factor of 0.5 whenever training loss reaches a plateau. We present MSE and Bad Pixel Ratio results for 5 different views, 4 being corners and 1 being center on Tables I-IV. Qualitative results obtained with 2 EPINET and 2 CEPINET models with the corresponding validation samples are presented on Figure 3.

We report the compressed size, as total compressed file size over the total number of pixels $(9 \times 9 \times 512 \times 512)$ of the light field, expressed in bits per pixel (bpp), for 16 test samples out of the training set averaged over all target views with LLFC and JPEG2000 are presented on Tables V-VI. According to Tables V-VI LLFC compression method yields superior results to JPEG2000 for all samples. Using CERV [6] for compressing best reference labels, instead of JPEG2000, yields slightly better results as evident on Tables V-VI. CERV provides a specialized framework for encoding constant value regions in an image, therefore it is expected to yield better results when compared to JPEG 2000 which is a generic image compression scheme. On the other hand, JPEG 2000 has the advantage of providing a less complicated encoder and decoder architecture while not sacrificing a lot from accuracy.

The compressed size obtained with ground truth (GT) corner and center disparities are also presented as an ideal reference on Table V. GT disparity results provide an

Table III. Bad Pixel Ratio with Town as Validation (Threshold =0.07)

View	Train Mean	Vinyl	Kitchen	Museum	Town
NW	**0.037**	**0.229**	0.251	0.113	0.049
NE	**0.037**	0.237	**0.238**	**0.107**	**0.043**
SW	0.043	0.254	0.246	0.130	0.047
SE	0.043	0.230	0.244	0.132	0.052
Center	0.043	0.260	0.259	0.132	0.047

Table IV. Bad Pixel Ratio with Greek as Validation (Threshold = 0.07)

View	Train Mean	Vinyl	Kitchen	Museum	Greek
NW	0.108	0.272	0.279	0.133	0.309
NE	0.108	0.267	0.278	**0.125**	0.295
SW	0.103	0.294	0.275	0.139	0.313
SE	0.109	0.273	0.271	0.143	0.337
Center	**0.075**	**0.245**	**0.253**	0.132	**0.279**

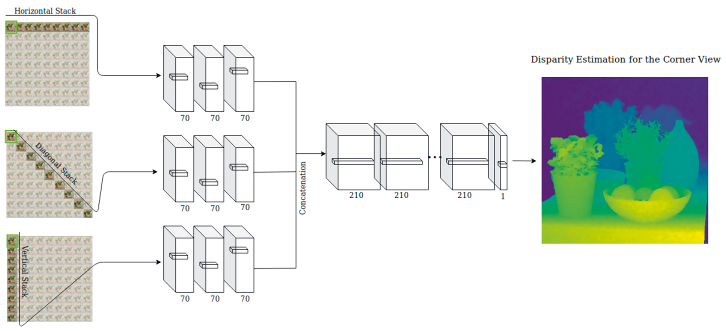


Figure 1. CEPINET: the angular directions for estimating the disparity map of the upper left corner view (green square) are depicted as black arrows.

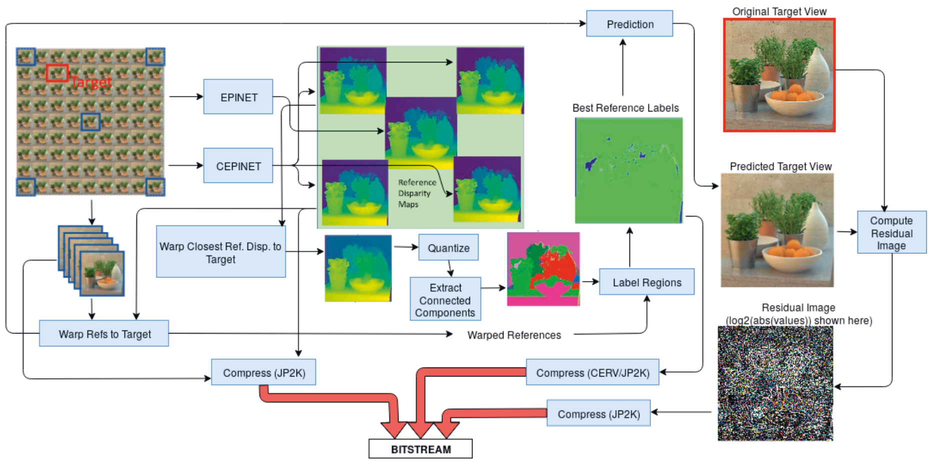


Figure 2. Proposed encoder architecture for lossless compression.

Table V. The compressed size for samples with GT disparity available

Sample	JP2K	LLFC(JP2K/CERV)	LLFC(GT Ds)
vinyl	7.38	4.41/4.30	**4.08**
kitchen	9.07	6.27/6.15	**5.78**
museum	10.97	7.01/6.92	**6.67**
greek	8.15	5.22/5.10	**4.94**

Table VI. The compressed size for samples without GT disparity

Sample	JP2K	LLFC(JP2K)	LLFC(CERV)
dino	9.65	5.84	**5.79**
dots	24.16	20.43	**20.06**
bedroom	10.13	6.94	**6.90**
pyramids	19.88	13.40	**13.32**
stripes	3.44	2.01	**1.92**
bicycle	12.85	8.80	**8.65**
b.gammon	16.62	11.40	**11.30**
origami	10.53	6.97	**6.83**
boxes	11.34	8.12	**7.95**
cotton	6.96	3.26	**3.21**
sideboard	13.93	9.56	**9.42**
herbs	11.93	8.15	**8.01**

Table VII. The compressed size at different Views (LLFC(CERV))

Ref	7.81	7.96	8.01	8.10	7.95	7.91	7.80	**Ref**
7.92	8.70	8.77	8.82	8.23	8.83	8.76	8.70	7.93
8.02	8.73	8.80	8.77	8.14	8.76	8.83	8.75	8.02
8.04	8.75	8.73	8.69	7.94	8.68	8.75	8.79	8.06
7.95	7.89	7.86	7.72	**Ref**	7.73	7.89	7.94	8.00
8.21	8.72	8.69	8.66	7.88	8.64	8.72	8.75	8.19
8.12	8.81	8.71	8.67	7.97	8.67	8.73	8.81	8.13
7.96	8.73	8.80	8.71	7.99	8.68	8.77	8.73	7.97
Ref	7.79	7.94	7.98	7.92	7.93	7.89	7.77	**Ref**

upper limit for the performance of our framework that can be attained by improving the disparity estimation stage. Best reference labels are encoded using CERV.

The compressed size results of our lossless compression scheme at every target view averaged over 16 samples (listed on Tables V-VI) are presented on Table VII. These values are computed by averaging the common costs due to transmission of reference color and disparities over all targets and adding each target's own cost for best reference labels and residuals. It should be noted that this computation corresponds to the scenario when whole light field image is being transmitted. Lossless JPEG2000 yields an average bpp of 11.7 for the same data for all views. Hence, corner view based compression scheme yields better results when compared to JPEG2000 at all camera array locations as evident from Table VII.

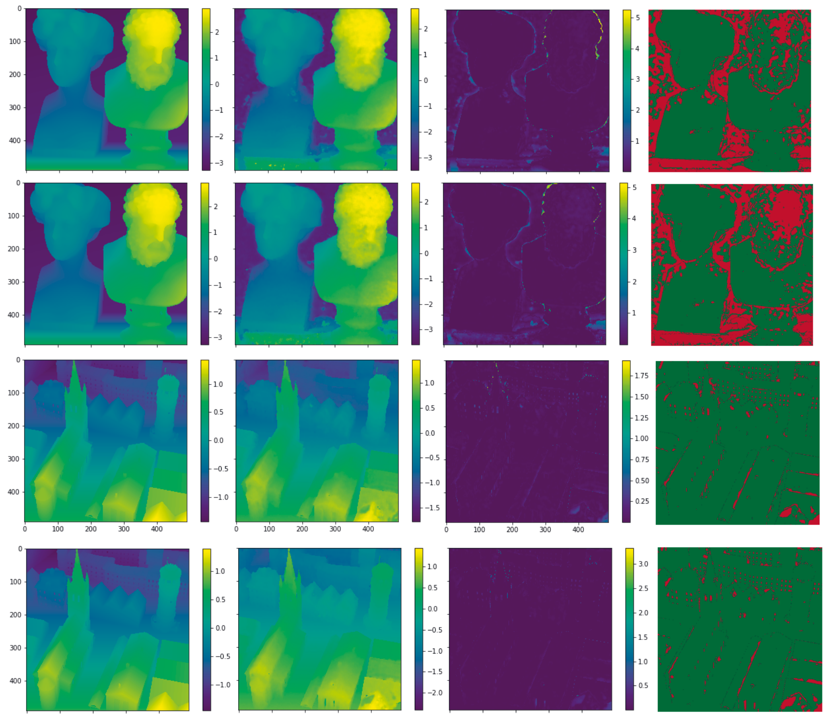


Figure 3. Columns from left to right: Ground Truth, Estimation, Absolute Error, Bad Pixel Mask (0.07). Rows from top to bottom: Center, NW Corner, Center, NW Corner.

V. CONCLUSION

In this work, we constructed the CEPINET for estimating corner view disparity maps of a light field image. It is observed that this variant is able to generate corner view disparities at a similar precision or even better than the center view estimates by EPINET. The proposed lossless compression method provides random access to individual targets, at the cost of transmitting first only the references, and its compression ratio is expected to be lower than the methods that don't possess the random access feature. On the other hand, the compression ratio of LLFC is better than the independent encoding of views by JPEG 2000, which provides instantaneous random access.

VI. REFERENCES

[1] Nima Khademi Kalantari, Ting-Chun Wang, and Ravi Ramamoorthi, "Learning-based view synthesis for light field cameras," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, pp. 193, 2016.

[2] Youngjin Yoon, Hae-Gon Jeon, Donggeun Yoo, Joon-Young Lee, and In So Kweon, "Learning a deep convolutional network for light-field image super-resolution," in *Proceedings of the IEEE international conference on computer vision workshops*, 2015, pp. 24–32.

[3] Changha Shin, Hae-Gon Jeon, Youngjin Yoon, In So Kweon, and Seon Joo Kim, "Epinet: A fully-convolutional neural network using epipolar geometry for depth from light field images," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4748–4757.

[4] Ioan Tabus, Petri Helin, and Pekka Astola, "Lossy compression of lenslet images from plenoptic cameras combining sparse predictive coding and jpeg 2000," in *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2017, pp. 4567–4571.

[5] Charilaos Christopoulos, Athanassios Skodras, and Touradj Ebrahimi, "The jpeg2000 still image coding system: an overview," *IEEE*, vol. 46, no. 4, pp. 1103–1127, 2000.

[6] Ioan Tabus, Ionut Schiopu, and Jaakko Astola, "Context coding of depth map images under the piecewise-constant image model representation," *IEEE Transactions on Image Processing*, vol. 22, no. 11, pp. 4195–4210, 2013.

[7] Katrin Honauer, Ole Johannsen, Daniel Kondermann, and Bastian Goldluecke, "A dataset and evaluation methodology for depth estimation on 4d light fields," in *Asian Conference on Computer Vision*. Springer, 2016, pp. 19–34.

PUBLICATION

II

Successive Refinement of Bounding Volumes for Point Cloud Coding
I. Tabus, E. C. Kaya and S. Schwarz

*2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*2020, 1–6
DOI: 10.1109/MMSP48831.2020.9287106

Successive Refinement of Bounding Volumes for Point Cloud Coding

Ioan Tabus
Computing Sciences Unit
Tampere University
Tampere, Finland
ioan.tabus@tuni.fi

Emre Can Kaya
Computing Sciences Unit
Tampere University
Tampere, Finland
emre.kaya@tuni.fi

Sebastian Schwarz
Media Technology Research
Nokia Technologies
Munich, Germany
sebastian.schwarz@nokia.com

***Abstract*—The paper proposes a new lossy way of encoding the geometry of point clouds. The proposed scheme reconstructs the geometry from only the two depth maps associated to a single projection direction and then proposes a progressive reconstruction process using suitably defined anchor points. The reconstruction from the two depth images follows several primitives for analyzing and encoding, several of which are only optional. The resulting bitstream is embedded and can be truncated at various levels of reconstruction of the bounding volume. The encoding tools for encoding the needed entities are extremely simple and can be combined flexibly. The scheme can also be combined with the G-PCC coding, for reconstructing in a lossless way the sparse point clouds. The experiments show improvement of the rate-distortion performance of the proposed method when combined with the G-PCC codec as compared to G-PCC codec alone.**

***Index Terms*—point cloud coding, lossy geometry compression, anchor points.**

I. Introduction

The compression of the geometry of point clouds is subject to a renewed strong interest, especially since it became a main topic of the standardization bodies [1], [2]. This follows decades of intense coverage in academic literature, where the octree point cloud compression was introduced some four decades ago and then steadily developed, with some landmark references [3]–[10]. However, other alternative approaches are recently considered, with a large number of contributions in the area of deep learning methods, e.g., investigated in [11], [12] and also other new approaches, e.g., using 2D image processing for encoding a collection of 2D images as explored in [13], [14].

In Section II we present the principle of our method, while in Section III we detail the main steps of the algorithm. In Section IV we present experimental results and finally Section V contains a discussion and conclusions.

II. Principle of the method

In this paper the geometry encoding is done through a sequential process, where tight bounding volumes around the points of the point cloud are iteratively constructed and refined. The point cloud is translated so that x, y, and z are strictly positive and bounded by N, M, and L, respectively. We select a first axis, say Oz, for defining the depth, and a second axis, say Oy, for drawing perpendicular plane sections. The starting stage is the encoding of two 2D depth images: one for $\max z(x,y)$ and one for $\min z(x,y)$, for all (x,y) in the $N \times M$ rectangular grid. A reconstruction bounding volume (RBV) is initialized to include all points (x,y,z) having z between max $z(x,y)$ and min $z(x,y)$. The reconstruction bounding volume is then sectioned perpendicular to the second axis, at the planes $y = y_0$, and for each resulting point cloud section the boundary is extracted and refined in two stages, so that it becomes identical to the true boundary of the section $y = y_0$. The first refining stage marks as ones the points of the reconstructed boundary contour (RBC) that are true points, and as zeros the points that are not true boundaries, and the resulting binary string (BS) is encoded using a Markov model. The starts and ends of the zero runs in the BS form the potential anchor points, between which the true boundary is different from the RBC. These true segments are encoded as chain codes, resulting finally in the true border of the entire section $y = y_0$. Next are considered the connections between the two successive sections along the plane $y = y_0$ and the plane $y = y_0 + 1$ (for which we know only the true boundaries, but we don't know yet the possible interior points). By examining the necessary connectivity relations, one obtains sets of candidate interior points engulfed between the two boundaries, which are checked against ground truth and their validity state is transmitted to the decoder. At this stage the outer shape of a reconstructed bounding volume is fully defined. One can perform additional iterations for clarifying if among the marked candidate interior points within each section there are points that do not belong to the point cloud, hence transmitting refinements for the interior points, until all points within the true bounding volume are correctly transmitted. The successive markings of the true points along candidate boundary contours or across boundary surfaces at various stages give rise to exclusion information, which is stored so that no point is tested twice if it belongs to the point cloud or not.

The transmission of the markings along a sequence, or of the chain codes for a sequence between two anchor points, or of the interior points, have each some associated code length, per occupied voxel transmitted. The transmission of the refinements of the RBV is organized in the increasing order of the bits per voxels of the elements, resulting in an embedded bitstream, that can be truncated at various points, for obtaining an optimized RD overall performance. We note that a point cloud can be pre-processed so that it is sectioned in several bounding volumes, where each section, say at $y = y_0$, is decomposed in several 2D connected components, resulting in simple processing along boundaries of connected components, and also facilitating random access to the different parts of the point cloud. For sparse point clouds, the proposed method can be applied at lower than full resolution levels, while for a full lossless reconstruction one may call several stages of further octree decomposition and encoding. However, good lossy performance can be obtained already using the proposed scheme alone.

III. Description of the algorithm stages

We consider a point cloud defined by a $(n_{PC} \times 3)$ matrix $\mathbf{B}$, with the i'th row a vector containing the i'th pixel coordinates $\mathbf{B}_i = [x_i \ y_i \ z_i]$ for all $i = 1, \ldots, n_{PC}$. We consider voxelized point clouds with 2^n pixels per coordinate, where $B_{ij} \in \{0, \ldots, 2^n - 1\}$. We also refer to the point cloud as a set $\mathcal{B} = \{(x_i, y_i, z_i) | i = 1, \ldots, n_{PC}\}$.

When convenient we might consider the six possible permutations of the components so that the point cloud is viewed from any of the six different direction. In this presentation we associate z to "depth" coordinate, y to the direction for sectioning the point cloud with a plane $y = y_0$, and (z, x) to the order of coordinates in the 2D image representing a section (where z is the height or row index and x is the column index in the 2D image).

A. Encode two depth images describing the upper cover and lower cover of the bounded volume across the vertical Oz axis

This stage can be described simply as encoding the depth image of the highest points seen above the Oxy plane and then encoding the depth image of the lowest points above the Oxy plane. The empty part of each depth image is the same, so one can improve the performance by utilizing this information when encoding the two images. We define formally the two depth images as follows. We construct first the outer maximum depth image $\mathbf{Z}_{\max}$: associate to each 2D-point $(x, y) \in \{0, \ldots, 2^n - 1\}^2$ of the square grid the value $Z_{\max}(x, y)$ which a) either indicates a non-existence marker $Z_{\max}(x, y) = -1$ (in the case that $(x, y) \neq (x_i \ y_i)\}$ for all $(x_i, y_i, z_i) \in \mathcal{B}$) or b) $Z_{\max}(x, y)$ specifies an existent depth (in case that (x, y) appears in any triplet (x_i, y_i, z_i)). The existent depth is set as $Z_{\max}(x, y) = \max\{z_i | (x, y, z_i) \in \mathcal{B}\}$.

Then similarly consider the outer minimum half-shell, as $Z_{\min}(x, y) = \min\{z_i | (x, y, z_i) \in \mathcal{B}\}$ (in case that (x, y) appears in any triplet (x_i, y_i, z_i)), or mark the non-existence marker $Z_{\min}(x, y) = -1$ for the pairs (x, y) that are not appearing in any true point (x_i, y_i, z_i).

We use for encoding the depth images the algorithm (crack-edge,region,value) (CERV) [15] which is very efficiently encoding first the geodesic contours of the depth image, then it constructs out of the contours the connected component regions having the same depth value, and in the end transmits the depth value for each region.

In Figure 1 we show the two depth images that are encoded in the first stage. One can see the potential benefits of encoding the geodesic contours first and the transmitting the values in the constant regions. The similarities of the contour shapes (elongated along the outer contours of the shapes) make the algorithm to collect very specific statistics when encoding the geodesic shapes, and hence capture important regularities in the depth image.

B. Construct the feasible region on the section at the plane $y = y_0$

We now consider the second chosen axis, Oy, and construct perpendicular sections at every y_0. The true occupancy image $\mathbf{S}_{y_0}$ is defined at the pixel coordinates (z, x) as $\mathbf{S}_{y_0}(z, x) = 1$ if $(x, y_0, z) \in \mathcal{B}$ and otherwise $\mathbf{S}_{y_0}(z, x) = 0$. We show in Fig. 2a) the true occupancy image at y_0. We proceed to reconstruct this true image out of the already transmitted information, and using additionally transmitted entities, as follows.

From the currently transmitted depth images we can already mark in a first section reconstruction $\overline{\mathbf{S}}_{y_0}^{[0]}$ some known boundary points: select all existent points, at which $Z_{\min}(x, y_0) > -1$, and mark $\overline{S}_{y_0}^{[0]}(Z_{\min}(x, y_0), x) = 1$. Similarly the information in $\mathbf{Z}_{\max}$ can be transferred to the section reconstruction, by marking $\overline{S}_{y_0}^{[0]}(Z_{\max}(x, y_0), x) = 1$ for all $Z_{\max}(x, y_0) > -1$. In Fig. 2b) we show the occupied pixels in $\overline{\mathbf{S}}_{y_0}^{[0]}$, known after decoding the two depth images, for a given value of y_0.

We notice that at section y_0, for a given x_0 we already know that any occupied pixel (z, x_0) needs to have the coordinate z obeying the constraint $Z_{min}(x_0, y_0) \leq z \leq Z_{max}(x_0, y_0)$. Hence we define as feasible region of occupancy the area containing pixels (z, x_0), with $Z_{min}(x_0, y_0) \leq z \leq Z_{max}(x_0, y_0)$. We denote the binary feasible image $\overline{\mathbf{S}}_{y_0}^{[1]}$. We show in Fig. 2c) the feasible occupancy image at y_0.

C. Encoding by Primitive One: Encode a binary mask for recovering vertical stretches of existent pixels on the section at the plane $y = y_0$

We show in Fig. 3a) in white the boundary of the feasible occupancy image at y_0. The boundary is stored as a list of pixels, when traversing in clock-wise sense the boundary. We mark each element of the list by the occupancy status of that pixel. In Fig. 3b) is shown by red a pixel marked as one and by blue a pixel marked by zero. The list of this markings is a binary sequence that can be very efficiently coding by run-length coding or Markov model of order 1 and arithmetic coding. After this stage, the decoder knows that all pixels marked in red a true contour pixels.

D. Encoding by Primitive Two: Encode the missing elements of the true boundary using anchored chain codes

As a very useful outcome of the previous stage, one can extract the anchor points, between which we need to reconstruct the unknown yet pieces of boundary. All the possible anchor points are the ends of the red segments in Fig. 3b). These anchor points are shown by cyan and red circles (cyan for a starting pixel and red for a landing pixels), along the clockwise traversing of the true boundary. At the encoder, the segments of true boundaries between the anchor points are encoded as chain codes and transmitted to the decoder. The length of each segment does not need to be transmitted, since we check during decoding of the chain that we arrived in the landing pixel, and stop the chain encoding or decoding.

The anchor points can be visualized as points having coordinates (x, y_0, z) on the outer surface of the bounding volume, and each chain code segment reconstructs an inner segment of arbitrary complexity, see, e.g., the cyan contour at the bottom right part of Fig. 6, which has a convoluted shape, requiring a complex transformation process if one wanted to project it on a plane. We do not project the points to a plane, but instead we keep them organized as anchored segments, at known locations of the outer surface.

E. Encoding by Primitive Three: Infer possible inner points in two consecutive sections

Two consecutive sections, at $y = y_0$ and $y = y_0 + 1$ might have quite different outer boundary of the true region. In order for the bounding volume to maintain 4 or 8 connectivity, between the outer boundary of two sections one needs to add inner points in one or both of the sections.

In Figure 5 are shown the operations that are needed for adding inner pixels to the boundary of a section, when its boundary forms un-occupied islands between it and the boundary of the previous section.

The reason to presume the existence of some inner pixels is that they would be necessary for ensuring connectivity between the boundary pixels at two consecutive sections, which is the case if the point cloud surface is water tight. However, if this water tight hypothesis is not true, the decoder will be announced to not include the candidate inner points. In this primitive we find the candidate inner pixels to add, without any coding cost, only by comparing two successive boundaries and marking the islands (each island is a connected component) of black pixels engulfed in the interior of the red boundary, between the red boundary and the blue boundary (see Figure 5a). After the candidate sets are identified, they are checked against the true points existing in the section (the true points include boundary points and inner points) as in 5b). If an island contains true inner points it is declared valid, and is marked with a one, so the decoder will know to add those inner points in the reconstructed section (and consequently in the reconstructed overall point cloud), as in 5c). This marking cost is very small and pays off in adding very important group of points to the reconstructed point cloud.

The process by which we added inner points to the section at $y = y_0 + 1$ based on the boundary of the section at $y = y_0$ can then be iterated also for adding inner points to $y = y_0 + 1$, based on the boundary of $y = y_0 + 2$, when that boundary will become available. Hence each section, say at $y = y_0 + 1$, can be enriched and made closer to the true section, by using the information about the boundary of the two sections, above and below it.

Some interior points can still be missing after this phase of Primitive Three, and they can be added in the next phases, if the available bitrate allows transmitting additional information, with transmission of new pairs of anchor points and of chains of pixels formed from inner pixels.

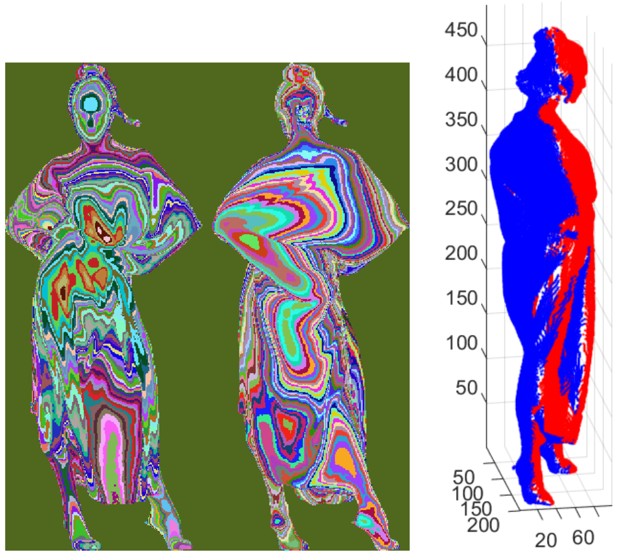


Fig. 1. Stage 1: Encoding a front and a back depth map image. (Left) The two depth images, shown in pseudocolor, can be represented in the most natural way by encoding first the geodesic contours, and then for each region a value. (Right) The resulting point cloud map, using only the two depth map images, where points from the front depth map are shown in red and the points from the back depth map are shown in blue. The outer surface is incomplete, which can be seen well from the side views.

IV. Experimental results

The implementation of the method is illustrated for several full-body point clouds. The datasets are from [16]. Each point cloud is first decomposed in a number of "tubes" along the Oy axis, so that each of their cross-sections is a single 2D-connected component. For example, for "longdress" dataset, the original number of points is 857966 and the first tube covers 790833 points, the next largest covers 53795 points, so together the two largest cover 844628 points. The algorithm can be applied to each of the tubes, however we resorted to a hybrid solution, where we encode with our algorithm only the largest tube and collect all other points of the smaller tubes into a point cloud which is encoded by G-PCC.

We tested the resulted scheme at more quantization points than required in the CTC [17] and the evaluation of distortion is done according to the CTC [17] [18].

In Fig. 7 and 8 is shown the performance of the proposes bounding volume method. It exceeds the performance of the G-PCC alone, and hence it is shown to capture in a more efficient way the regularities of the point cloud surfaces. Further refining of the method and experiments are underway

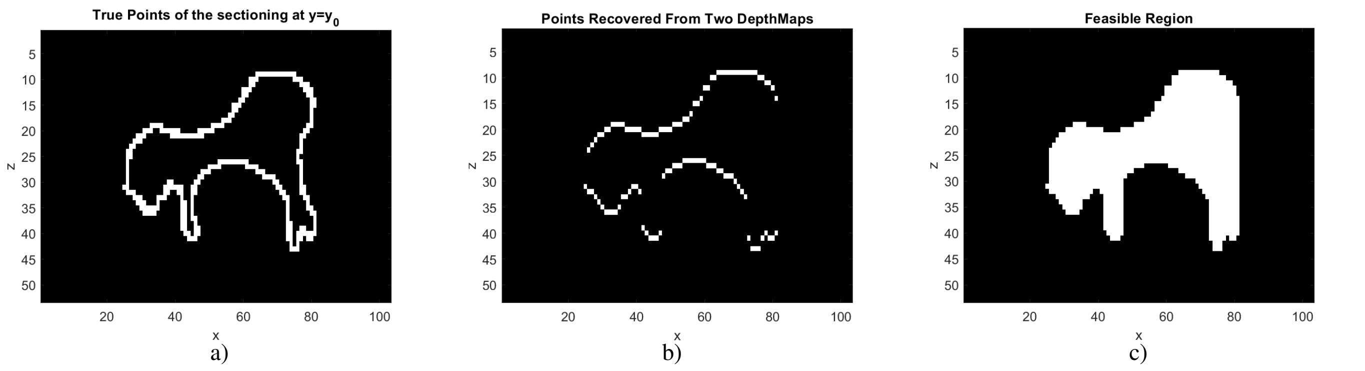


Fig. 2. Section on the point cloud at the plane $y = 50$ (for a PC with resolution $n = 8$ bits. Stage 2: Analyzing the existing points from Stage 1 for constraints on the occupied points of current section. a) True occupied points on this section; b) Points known to be occupied from Stage 1, after decoding the two depth map images; c) Feasible region for the occupied points, drawn by uniting the vertical maximum and minimum points on each vertical at the image in Panel b).

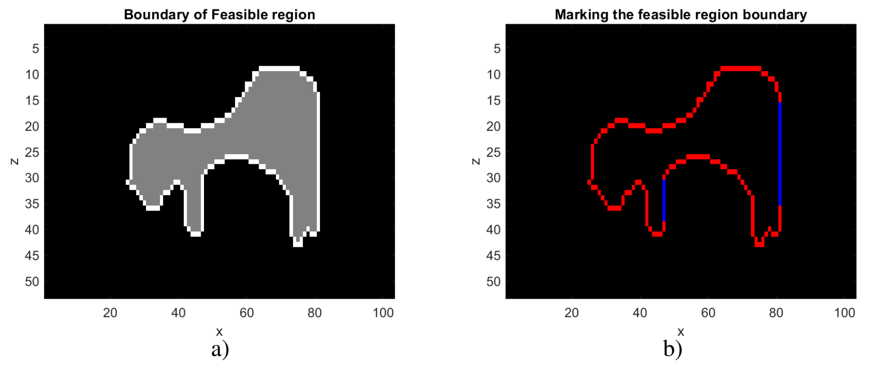


Fig. 3. Stage 3: Encoding with Primitive One: a) Boundary of the feasible region marked in white; b) Encoding the incorrect pixels on the boundary of the feasible region (the pixels marked in blue); The run length coding or Markov chain arithmetic coding of the long stretches of ones and zeros are very efficient.

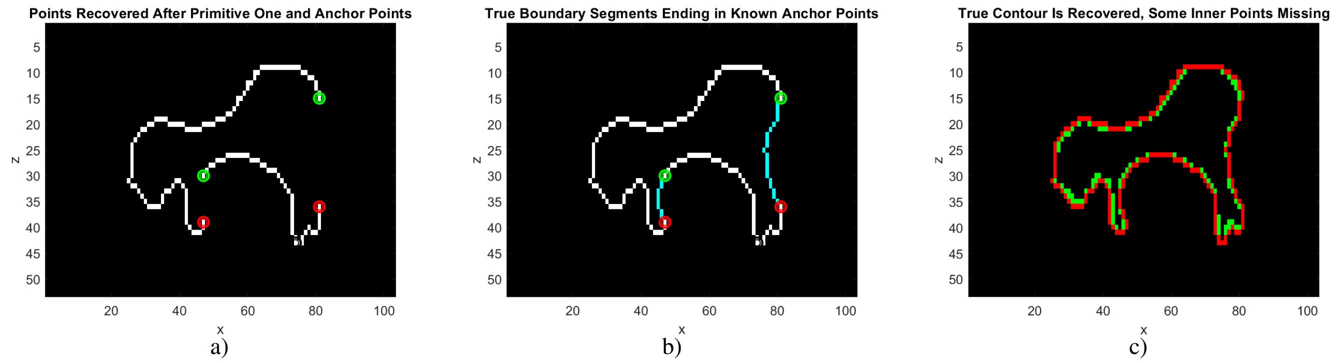


Fig. 4. Stage 4: Encoding with Primitive Two: a) True boundary points recovered after Stage 1 (Primitive Coding One) with their open ends marked as anchor points. The boundary indexing starts at the bottom-leftmost pixel and goes clockwise around the boundary of Figure 3b), marking along when a recovered stretch of ones ends (green circle) and when it restarts (red circle). b) Encoding the missing true boundary segments, marked in cyan, as chain codes. The ends of each chain code are known, so its length does not need to be transmitted. c) The end of Stage 2, with the full true boundary recovered (marked in red) and with the additional missing inner true points marked in green.

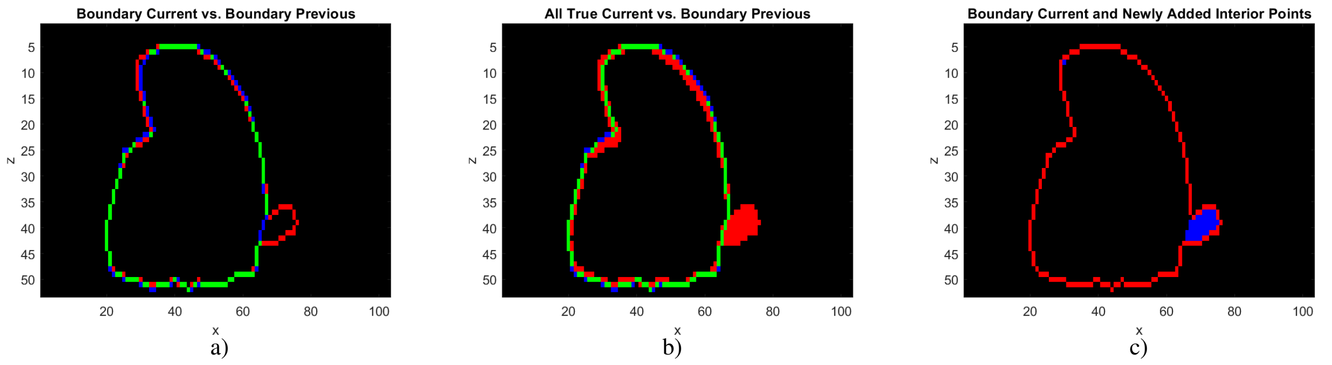


Fig. 5. Stage 5: Encoding with Primitive Three: Two consecutive sections, at $y = 143$ and $y = 144$. a) True boundaries obtained at $y = 143$ and $y = 144$ with Primitive One and Two. The boundary at 143 is shown in blue and green, the boundary at 144 is shown in green and red (green are the common points). All black points stranded between red (outer) and blue (inner) are candidates for interior points at $y = 144$; b) Indeed all the true points at $y = 144$ (shown in red and green) include the candidate interior points, so they should be added to section $y = 144$ (in blue and green is shown the boundary at $y = 143$); c) Finally the new interior points, shown in blue, are added to the boundary (shown in red) of the section at $y = 144$.

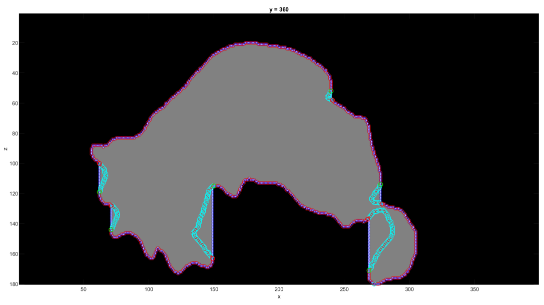


Fig. 6. A detail of encoding with Primitives One and Two for encoding the outer boundary of a section at $y = 360$ at a higher resolution PC ($n = 9$). Blue ‘x’ marks the feasible boundary obtained after transmitting the two depth images; the red circle marks the true boundary pixels recovered after encoding by Primitive One and the cyan circles mark the pixels recovered after encoding by Primitive Two.

for improving the efficiency and for adding new features, able to extract regularities from more complex point clouds.

V. Discussion and conclusions

This proposal is intended for lossy compression of the geometry of point clouds. Compared to the existing methods, there are several new features, listed below.

The proposed scheme reconstructs the geometry from only the two depth maps associated to a single projection direction. Hence there is no need to organize an expanded pad or mosaic image. Instead, one uses well defined anchor points, that are recovered as side information of the progressive reconstruction process.

The reconstruction from the two depth images follows several primitives for analyzing and encoding, several of which are only optional. The resulting bitstream is embeded, it can be truncated at various levels of reconstruction of the bounding volume.

The encoding tools for encoding the needed entities (for the binary string representing the occupancy mask, for the chain codes representing the anchor segments) are extremely simple and elementary: run length coding, or Markov models of order one with arithmetic coding, and finally chain coding which can be implemented in many efficient ways.

The scheme can be combined with the G-PCC coding, for reconstructing point clouds in a lossless way.

References

[1] 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.

[2] T. Ebrahimi, S. Foessel, F. Pereira, and P. Schelkens, “Jpeg pleno: Toward an efficient representation of visual reality,” *Ieee Multimedia*, vol. 23, no. 4, pp. 14–20, 2016.

[3] C. L. Jackins and S. L. Tanimoto, “Oct-trees and their use in representing three-dimensional objects,” *Computer Graphics and Image Processing*, vol. 14, no. 3, pp. 249–270, 1980.

[4] D. Meagher, “Geometric modeling using octree encoding,” *Computer graphics and image processing*, vol. 19, no. 2, pp. 129–147, 1982.

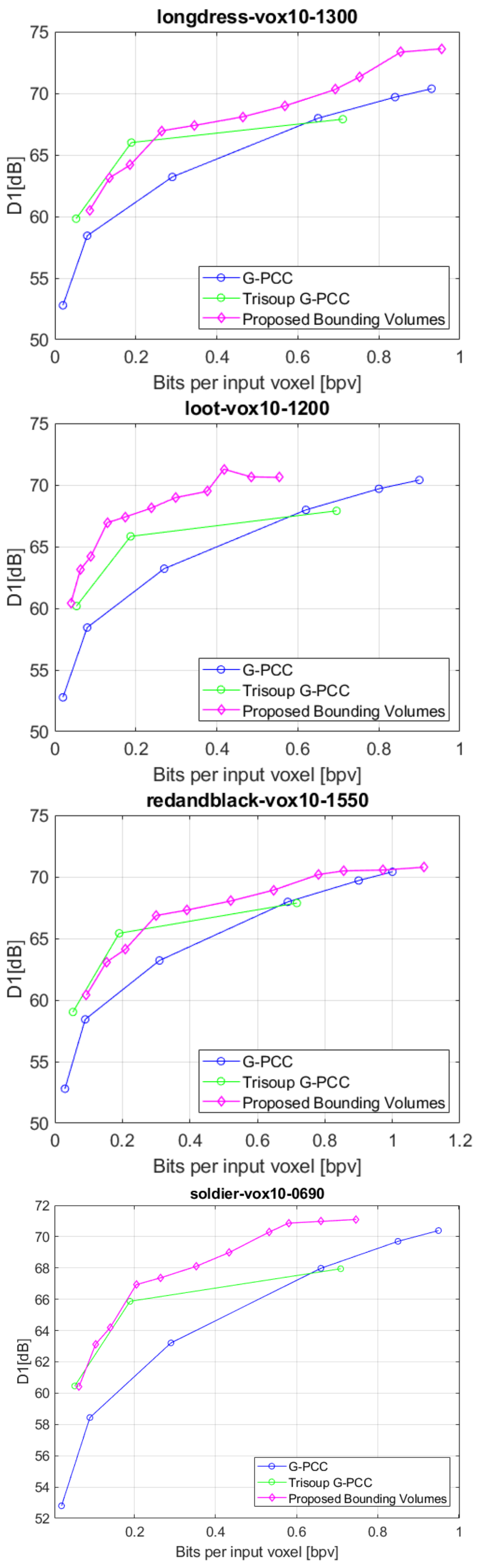


Fig. 7. Results of encoding full body point clouds using the proposed method, compared against the G-PCC baseline and G-PCC tri-soup.

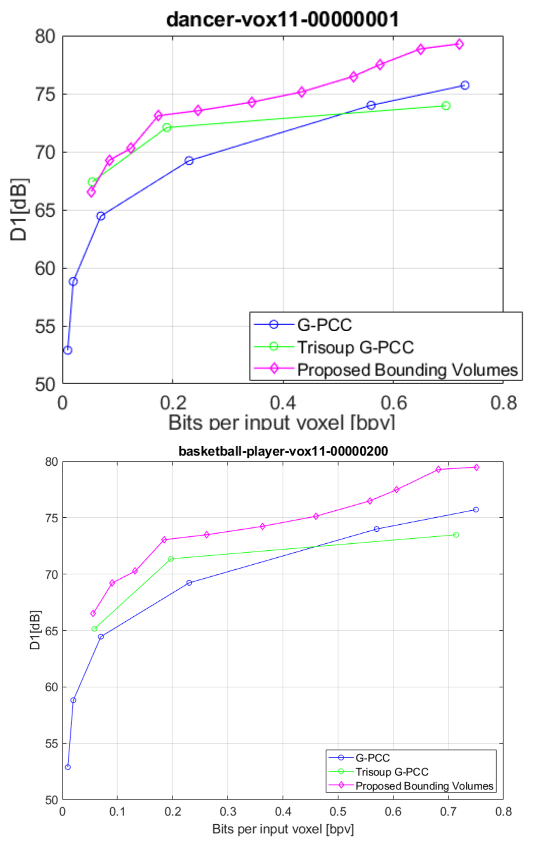


Fig. 8. Results of encoding full body point clouds using the proposed method, compared against the G-PCC baseline and G-PCC tri-soup.

[5] Y. Huang, J. Peng, C.-C. J. Kuo, and M. Gopi, "Octree-based progressive geometry coding of point clouds." *SPBG*, vol. 6, pp. 103–110, 2006.

[6] R. Schnabel and R. Klein, "Octree-based point-cloud compression." *Spbg*, vol. 6, pp. 111–120, 2006.

[15] I. Tabus, I. Schiopu, and J. Astola, "Context coding of depth map

[7] 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*, vol. 27, no. 4, pp. 828–842, 2016.

[8] 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.

[9] 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.

[10] 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.

[11] M. Quach, G. Valenzise, and F. Dufaux, "Learning convolutional transforms for lossy point cloud geometry compression," in *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2019, pp. 4320–4324.

[12] A. F. R. Guarda, N. M. M. Rodrigues, and F. Pereira, "Deep learning-based point cloud geometry coding: Rd control through implicit and explicit quantization," in *2020 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*, 2020, pp. 1–6.

[13] 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.

[14] E. Peixoto, "Intra-frame compression of point cloud geometry using dyadic decomposition," *IEEE Signal Processing Letters*, vol. 27, pp. 246–250, 2020.
images under the piecewise-constant image model representation," *IEEE Transactions on Image Processing*, vol. 22, no. 11, pp. 4195–4210, Nov 2013.

[16] E. d'Eon, B. Harrison, T. Myers, and P. A. Chou, "8i voxelized full bodies, version 2–a voxelized point cloud dataset," *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document m40059 M*, vol. 74006, 2017.

[17] S. Schwarz, G. Martin-Cocher, D. Flynn, and M. Budagavi, "Common test conditions for point cloud compression," *Document ISO/IEC JTC1/SC29/WG11 w17766, Ljubljana, Slovenia*, 2018.

[18] D. Tian, H. Ochimizu, C. Feng, R. Cohen, and A. Vetro, "Geometric distortion metrics for point cloud compression," in *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2017, pp. 3460–3464.

PUBLICATION

III

Refining The Bounding Volumes for Lossless Compression of Voxelized Point Clouds Geometry

E. C. Kaya, S. Schwarz and I. Tabus

*2021 IEEE International Conference on Image Processing (ICIP)*2021, 3408–3412
DOI: 10.1109/ICIP42928.2021.9506767

REFINING THE BOUNDING VOLUMES FOR LOSSLESS COMPRESSION OF VOXELIZED POINT CLOUDS GEOMETRY

Emre Can Kaya[⋆] *Sebastian Schwarz*[†] *Ioan Tabus*[⋆]

[⋆] Tampere University, Tampere, Finland
[†] Nokia Technologies, Munich, Germany

ABSTRACT

This paper describes a novel lossless compression method for point cloud geometry, building on a recent lossy compression method that aimed at reconstructing only the bounding volume of a point cloud. The proposed scheme starts by partially reconstructing the geometry from the two depthmaps associated to a single projection direction. The partial reconstruction obtained from the depthmaps is completed to a full reconstruction of the point cloud by sweeping section by section along one direction and encoding the points which were not contained in the two depthmaps. The main ingredient is a list-based encoding of the inner points (situated inside the feasible regions) by a novel arithmetic three dimensional context coding procedure that efficiently utilizes rotational invariances present in the input data. State-of-the-art bits-per-voxel results are obtained on benchmark datasets.

***Index Terms*—** Point Cloud Compression, Context Coding, Lossless Compression, Arithmetic Coding

1. INTRODUCTION

The lossless compression of point clouds was thoroughly studied and is currently under standardization under MPEG [1],[2] and JPEG activities [3]. The research literature on point cloud compression includes a lot of methods based on octree representations, e.g., [4, 5, 6, 7, 8]. The current GPCC lossless method [9] is also based on octree representations [10], where a point cloud is represented as an octree, which can be parsed from the root to the leaves and at each depth level in the tree, one obtains a lossy reconstruction of the point cloud at a certain resolution, while the lossless reconstruction is obtained at the final resolution level in the octree. At each resolution level, an octree node corresponds to a cube at a particular 3D location and the octree node is labeled by one if within that cube there is at least one final point of the point cloud. By traversing in a breadth-first way, one can obtain a lossy reconstruction at each depth of the tree, while the lossless reconstruction is obtained at the final resolution. Octree-based approach is attractive for providing a progressive-in-resolution reconstruction.

Recently, good lossless performance was achieved by methods based on successive decomposition of the point cloud data into binary images [11, 12], without using octree representations.

This paper describes a novel lossless point cloud compression algorithm which is based on Bounding Volumes [13] for G-PCC. In the Bounding Volumes, all reconstructed points truly belonged to the point cloud, but some points, specifically inner points in the transversal sections of the point cloud, were not encoded at all. In this work, a complete lossless process, overlapping in the first stage with the Bounding Volumes method (encoding a front and a back projection), but diverging from the previous method, already at the second stage, that of encoding the true boundary of the feasible region and making it less restrictive, getting rid of the requirement of decomposing the point cloud into tubes having single connected component sections.

Compared to the lossy Bounding Volume method, the newly introduced encoding process includes an extensive context coding scheme that can finally provide a lossless reconstruction of the point cloud. This novel scheme is based on carefully selected context coding methods and intensively uses exclusion to avoid unnecessary testing for the occupancy of the locations which are already known.

2. PROPOSED METHOD

Consider a point cloud having the resolutions N_x, N_y, N_z along the axes x, y, z, respectively. The points are encoded in two stages. In Stage I, two projections of the points cloud are encoded; the front and the back projections on xy plane. These projections are two depthmap images, each with N_xxN_y pixels. Then, the coding proceeds along the Oy axis of the 3D coordinate system, drawing transverse sections of the point cloud parallel to the zOx plane and encoding each such section in Stage II. An overview of the method is presented on Fig. 1. The regularity of the geometric shapes, including smoothness of the surface and the continuity of the edges, are exploited by using context models, where the probability of occupancy of a location is determined by the occupancy of its neighbor locations in 3D space, by including the causal (previously encoded) 3D neighbors from the current and the past sections.

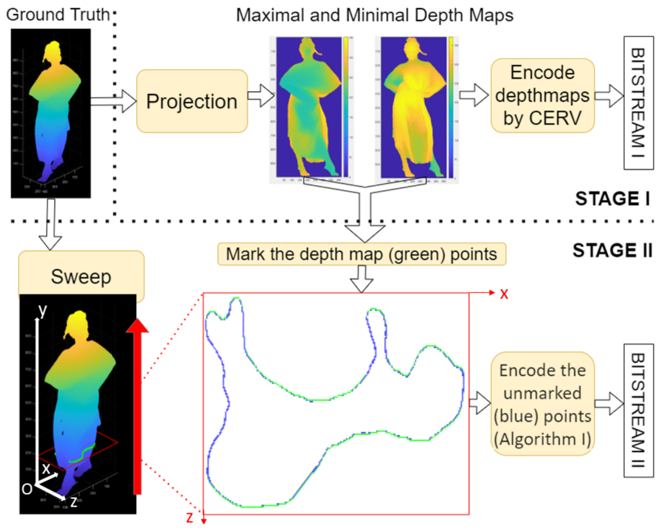
2.1. Stage I: Encoding a front and a back depthmap projection

The first encoding stage is intended for defining and encoding the maximal and minimal depthmaps (representing heights above the Oxy plane) resulting in exclusion sets containing large parts of the space. The minimal depthmap, Z_{min}, has at the pixel (x, y) the value, $Z_{min}(x, y)$ equal to the minimum z for which (x, y, z) is a point in the original point cloud. Similarly, the maximal depthmap, Z_{max} has at the pixel (x, y) the value, $Z_{max}(x, y)$ equal to the maximum z for which (x, y, z) is a point in the original point cloud. If no point with (x, y) exists in the point cloud, then it is set $Z_{min}(x, y) = Z_{max}(x, y) = 0$. The encoding of these depthmaps is performed by CERV [14], which encodes the geodesic lines using contexts very rich in information.

2.2. Stage II: Encoding the remaining points

At Stage II, we sweep the point cloud along the y dimension, stepping y_0 from 0 to $N_y - 1$, and we reconstruct all the points in the current section y_0 in the $(N_z\text{x}N_x)$ binary image R (current reconstruction). At every y_0, points projected to the depthmaps are already known to the decoder and we initialize the current reconstruction R with the projected points such that, $R(Z_{max}(x, y_0), x) = 1$ and $R(Z_{min}(x, y_0), x) = 1$ for every $Z_{max}(x, y_0) > 0$ and $Z_{min}(x, y_0) > 0$. R will be updated whenever a new point is encoded or decoded. We note that, in the binary image R, we know at each column x which are the lowest and the highest occupied points (minimal and maximal depths). We consequently construct the binary image F of feasible regions, i.e., of locations in the plane that are possibly occupied (the magenta pixels in Fig. 2(b)). Formally, $F(z, x) = 1$ for all (z, x) pair satisfying $Z_{min}(x, y_0) \leq z \leq Z_{max}(x, y_0)$. Using R and F, we initialize a binary image K, where 1 denotes that the occupancy of a location is known. For example, the locations outside the feasible region are known to be unoccupied, hence, $K(z, x) = 1$ for those locations. The true points in the section, that we need to losslessly encode, are marked in a binary image denoted T (see Fig. 2(a)), and the reconstructed points in the past section (at $y_0 - 1$, that is already known to the decoder) are marked in a binary image P.

In the image R the already reconstructed true points belonging to depthmaps form a set ϕ of pixels. We perform a morphological dilation of the set ϕ using as structural element the 3×3 element. This obtained set of locations is traversed along the rows of the 2D image and is stored in a list L. We also initialize a binary marker image M to mark the pixels already in the list L. After this initialization step, the list L is processed sequentially, starting from the top of the list, processing a current pixel (z, x). Both encoder and decoder check whether $K(z, x) = 1$, and if yes, the point is



Ground Truth
Maximal and Minimal Depth Maps
Projection
Encode depthmaps by CERV
BITSTREAM I
STAGE I
STAGE II
Mark the depth map (green) points
Sweep
y
x
O
z
x
z
Encode the unmarked (blue) points (Algorithm I)
BITSTREAM II

Fig. 1. Overview of the proposed method: In Stage I, minimal and maximal depthmaps Z_{min} and Z_{max} are generated by projecting the point cloud along z axis. The depthmaps are encoded by CERV[14]. In Stage II, the point cloud is swept through y axis. The points that are not already encoded by CERV (blue points) are encoded as described in Sections 2.2 and 2.3.

removed from the list, since its occupancy status is already known. Otherwise, if $K(z, x) = 0$, we transmit the value of $T(z, x)$ using arithmetic coding with the coding distribution stored at the context ζ. After that, the counts of symbols associated with the context ζ are updated. K is updated as $K(z, x) \leftarrow 1$, and the reconstructed image is updated as $R(z, x) \leftarrow T(z, x)$. If the value $T(z, x) = 1$, we include to the list any neighbor (z_n, x_n) of (z, x), (in 8-connectivity) for which $K(z_n, x_n) = 0$ and for which the marked status is 0, $M(z_n, x_n) = 0$. After inclusion, the marked status is set to 1, $M(z_n, x_n) = 1$. This procedure is repeated until the list becomes empty. At the end, we have encoded all the points that are connected to the boundary of the feasible region by a path of connected pixels (in 8-connectivity). After the final section $y_0 = N_y - 1$ is processed, all the points that are connected to the points contained in the initial two depthmap images are encoded. For the voxelized point clouds, this outer shell of points contains the majority of the points in the point cloud. The remaining points (if any) are encoded by processing additional shells, as described in Section 2.4.

2.3. Normalized Contexts

One of the most efficient ways of utilizing the contextual information that is needed in Stage II is described here. In order to show the elements forming the context, it is illustrated in Fig. 2(c) the ground truth occupancies for the cur-

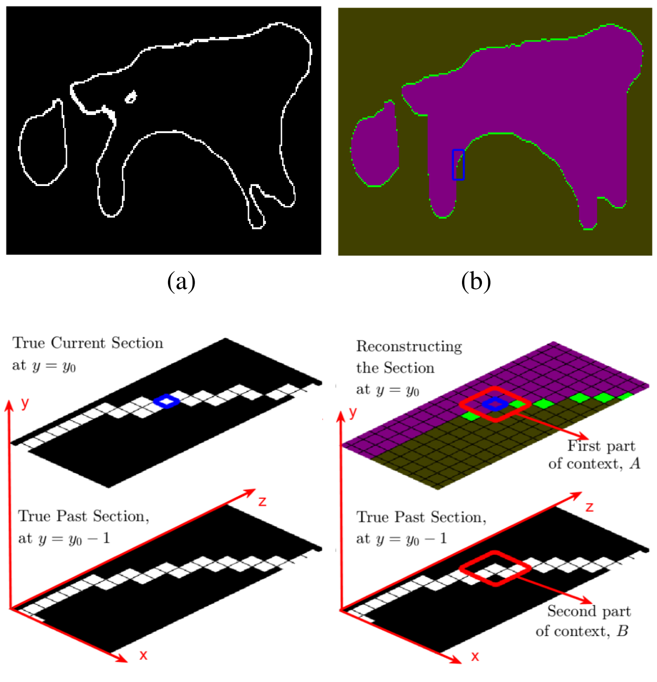


Fig. 2. (a) True current section; (b) The feasible region (magenta), points from the depth maps (green), forbidden region (khaki); (c) Details of the ground truth inside the blue rectangle from (b); (d) Selection of the two part context for encoding the blue marked pixel

Algorithm 1 Stage II of encoding

```
Require: $T$: True section binary image at $y = y_0$ ($N_z$ x $N_x$)
  $P$: True section binary image at $y = y_0 - 1$ ($N_z$ x $N_x$)
  $R$: Current reconstruction at $y = y_0 - 1$ ($N_z$ x $N_x$)
  $F$: Feasible Regions bin. image derived from $R$ (Sect. 2.2)
  $K$: Binary image of known locations $K \leftarrow \overline{F} + R$
  $L$: Pixels to be processed $L \leftarrow \{(z,x) \ni R(z,x) = 1\}$
  $M$: Binary image of pixels that has been to $L$. $M \leftarrow R$
  while $L \neq \emptyset$ do
    Read $(z,x)$ from the top of $L$
    Extract a 3x3 matrix $R_{3x3}$ from $R$ centered at $(z,x)$
    Extract a 3x3 matrix B from $P$ centered at $(z,x)$
    Extract a 3x3 matrix $K_{3x3}$ by cropping K around $(z,x)$
    $A \leftarrow R_{3x3} + K_{3x3}$
    Find normalizing rotation $\alpha^*(A)$ and form $A_{\alpha^*}$
    Use $\alpha^*(A)$ to rotate B as $B_{\alpha^*}$
    Form the context $\zeta = (I(A_{\alpha^*}), J(B_{\alpha^*}))$
    Encode $T(z,x)$ using the context $\zeta$
    if $T(z,x) == 1$ then
      Append to $L$ every neighbor $(z_n, x_n)$ of $(z,x)$ for which $K(z_n,x_n) = 0$ and $M(z_n,x_n) = 0$
      If $(z_n,x_n)$ is appended, $M(z_n,x_n) \leftarrow 1$
    end if
    Update R: $R(z,x) \leftarrow T(z,x)$
    Update K: $K(z,x) \leftarrow 1$
    Remove $(z,x)$ from $L$
  end while
```

rent section ($y = y_0$) and the past section ($y = y_0 - 1$), showing in white the (true) occupied pixels in these sections. On Fig. 2(d), we show the same area during the reconstruction process, which advances section by section such that, at the moment of reconstructing the section $y = y_0$, the section $y = y_0 - 1$ is fully known, and the second part of the context, called matrix B, can be extracted and contains the true occupancy status at section $y = y_0 - 1$. The context A from the current section for every candidate pixel (z,x) is extracted from the 3×3 neighbourhood of the pixel (i.e., the pixels in the red square).

When encoding the blue marked pixel, the pixels considered as the 3×3 A matrix part of context are those surrounded by the red contour. Each pixel might be green (already encoded in Stage I), for which the status is known and occupied ($K(z,x) = 1$ and $R(z,x) = 1$), khaki for forbidden pixels with status known and unoccupied ($K(z,x) = 1$ and $R(z,x) = 0$), and finally magenta for feasible i.e., not yet known ($K(z,x) = 0$). The context extracted from the current section forms the matrix A and that from the past section forms the matrix B (which are later rotated for normalizing and are combined to form the final contexts).

The procedure for encoding the points at a section $y = y_0$ with normalized contexts is summarized in Algorithm 1. Consider that we need to encode $T(z,x)$. The first part of the context uses the values of the already reconstructed pixels that are 8-neighbors of (z,x) and also the information about which of the pixels were already known. The values of the pixels in the ternary image $R_k = R + K$ have the following significance: $R_k(z,x) = 0$ if the value of $T(z,x)$ is not known yet; $R_k(z,x) = 1$ if the value of $T(z,x)$ was encoded and $T(z,x) = 0$; $R_k(z,x) = 2$ if the value of $T(z,x)$ was encoded, and $T(z,x) = 1$. We consider first the 3×3 square, centered at (z,x), cropped from the image R_k, and we denote it as a 3×3 matrix A. The elements of A belong by construction to 0, 1, 2. The second part of the context is the 3×3 binary matrix B formed from P at (z, x). The information from A and B is used to form the context. For example scanning by columns we get a one-to-one correspondence between A and $I(A) = \sum_{j=0}^{2} \sum_{i=0}^{2} A_{ij} 3^{i+3j}$. Similarly there is a one-to-one correspondence between B and $J(B) = \sum_{j=0}^{2} \sum_{i=0}^{2} B_{ij} 2^{i+3j}$. We combine them to a context label $\zeta = (I(A), J(B))$.

The context information in A and B is further normalized in the following way: We consider performing context collapsing operations, such that if we would perform a rotation by $\alpha \in \{0, \pi/2, \pi, 3\pi/2\}$ of each of the images R, T and K around the pixel (z,x), the value of the resulting normalized context is the same. We consider first the 3×3

Table 1. Average Rate for the first 200 frames from MVUB [15] and 8i [16] datasets for the proposed Bounding Volume Lossless (BVL) encoder compared to recent codecs.

Sequence	Average Rate [bpv]			
	P(PNI)[8]	**TMC13[9]**	**DD[12]**	**BVL**
Microsoft Voxelized Upper Bodies [15]				
Andrew9	1.83	1.14	**1.12**	1.17
David9	1.77	1.08	**1.06**	1.10
Phil9	1.88	1.18	**1.14**	1.20
Ricardo9	1.79	1.08	**1.03**	1.05
Sarah9	1.79	**1.07**	**1.07**	1.08
Average	1.81	1.11	**1.08**	1.12
8i Voxelized Full Bodies [16]				
Longdress	1.75	1.03	0.95	**0.91**
Loot	1.69	0.97	0.91	**0.88**
Redandblack	1.84	1.11	**1.03**	**1.03**
Soldier	1.76	1.04	**0.96**	**0.96**
Average	1.76	1.04	0.96	**0.94**

matrix A. Apply the α rotation around the middle pixel and denote A_α the resulting 3×3 matrix. Compute for each of $\alpha \in \{0, \pi/2, \pi, 3\pi/2\}$, the matrix A_α and the weighted score of it $W(A_\alpha)$ and pick as canonical rotation that α_* for which the weighted score $W(A_\alpha)$ is the largest. Hence, the four rotated matrices A_α with $\alpha \in \{0, \pi/2, \pi, 3\pi/2\}$ will be represented only by $A_{\alpha*}$. This process of collapsing the four matrices into a single one is performed offline once, resulting in a tabulated mapping $\alpha_* \leftrightarrow A$ and another mapping $I_* \leftrightarrow A$, which realize the mappings of the context to the canonical one, stored in look-up tables. As an example of the weighting score $W(A)$, we consider the vector $v = [A_{00}A_{01}A_{10}A_{02}A_{11}A_{20}A_{12}A_{21}A_{22}]$ and form $W(A) = \sum_{k=0}^{8} v_k 3^k$, giving in this way a larger weight to those elements which are close to the corner $(0, 0)$ of A. The normalized context is found in the following way: At each point (z, x) the matrix A is formed from R+K and the canonical rotation index α_* for this matrix is computed. Also the corresponding rotated matrix A_0 is computed. The second part of the context is the 3×3 matrix B formed from P around (z, x). The matrix B is rotated by the previously determined α_* around its center, yielding a matrix B_0. Now the context to be used for encoding $T(z, x)$ is constructed from A_0 and B_0 as context $\zeta = (I(A_0), J(B_0))$.

2.4. Repetitive peeling-off process for complete reconstruction of more complex point clouds

After Stage II, the reconstruction contains all the points forming the outer surfaces of the point cloud and all the inner points connected to these outer surface points, i.e., all points that are connected by a path in 3D space (in 26-voxel connectivity), to the initial points recovered in Stage I from the two depthmap images. However, there are complex point clouds, for example those representing a building and objects inside, where some objects are not connected by a 3D path to the outermost points. In that case one can repeat the encoding process shell by shell, in a peeling-off operation, where we encode first the outermost shell, defined by the points represented in the maximal and minimal depthmaps and all points connected to these points, and then we reapply the same process to the remaining un-encoded points. If needed, this peeling-off process can be applied several times. In this work, there are maximally 2 shells peeled-off and the remaining points (if any) are simply written in binary representation into the bitstream.

3. EXPERIMENTAL WORK

The algorithm is implemented in C and the experiments were carried out on two point cloud datasets namely, 8i Voxelized Full Bodies [16] and Microsoft Voxelized Upper Bodies [15]. The average bits per occupied voxel results (average rates) are presented on Table 1. For each point cloud, all 6 possible permutations of the 3 dimensions are tried and the best rate obtained is kept and reported here. It is observed that, proposed method performs better than the other methods on the 8i dataset. On the other hand, on MVUB dataset, our results are slightly worse than TMC13 [9] and Dyadic Decomposition [12]. Additionally, we test BVL and TMC13 on all the point clouds from the Cat1A MPEG Database [17] having original resolutions of 10 and 11 bits. These were quantized to 10, 9 and 8 bits as well to test the performance in lower resolutions. BVL outperformed TMC13 in average at 10 bits by 6.6%, at 9 bits by 5.2%, at 8 bits by 2%. At 11 bits, TMC13 outperformed BVL by 1.8%. For all of the point clouds mentioned in this work, the decoding resulted in a perfect lossless reconstruction.

Encoding and decoding durations for BVL were measured to be 7.3 sec and 7.8 sec, respectively. On the same machine, encoding with TMC13 took 1.1 sec. All durations are measured on a single frame of the 10 bits longdress sequence by running the algorithm 10 times and taking the median. While the durations are not competitive with TMC13, it should be noted that the execution time is not yet carefully optimized.

4. CONCLUSIONS

We proposed a lossless compression method where the first stage is constructing a bounding volume for the point cloud and the following steps succeed at adding all the remaining points at a competitive bitrate, achieving state-of-the-art results for the full body datasets, and comparable results to the current GPCC standard on the upper body dynamic datasets.

5. REFERENCES

[1] 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.

[2] L. Cui, R. Mekuria, M. Preda, and Eu. S. Jang, "Point-cloud compression: Moving picture experts group's new standard in 2020," *IEEE Consumer Electronics Magazine*, vol. 8, no. 4, pp. 17–21, 2019.

[3] T. Ebrahimi, S. Foessel, F. Pereira, and P. Schelkens, "JPEG Pleno: Toward an efficient representation of visual reality," *Ieee Multimedia*, vol. 23, no. 4, pp. 14–20, 2016.

[4] R. L. De Queiroz and P. A. Chou, "Compression of 3d point clouds using a region-adaptive hierarchical transform," *IEEE Transactions on Image Processing*, vol. 25, no. 8, pp. 3947–3956, 2016.

[5] 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*, vol. 27, no. 4, pp. 828–842, 2016.

[6] 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.

[7] Diogo C Garcia and Ricardo 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.

[8] 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.

[9] "MPEG Group TMC13," `https://github.com/MPEGGroup/mpeg-pcc-tmc13`, Accessed: 2020-03-20.

[10] D. Meagher, "Geometric modeling using octree encoding," *Computer graphics and image processing*, vol. 19, no. 2, pp. 129–147, 1982.

[11] 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.

[12] E. Peixoto, "Intra-frame compression of point cloud geometry using dyadic decomposition," *IEEE Signal Processing Letters*, vol. 27, pp. 246–250, 2020.

[13] 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.

[14] I. Tabus, I. Schiopu, and J. Astola, "Context coding of depth map images under the piecewise-constant image model representation," *IEEE Transactions on Image Processing*, vol. 22, no. 11, pp. 4195–4210, 2013.

[15] C. Loop, Q. Cai, S. O. Escolano, and P. A. Chou, "Microsoft voxelized upper bodies - a voxelized point cloud dataset," *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document m38673/M72012*, 2016.

[16] E. d'Eon, B. Harrison, T. Myers, and P. A. Chou, "8i voxelized full bodies - a voxelized point cloud dataset," *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document WG11M40059/WG1M74006*, 2017.

[17] S. Schwarz, G. Martin-Cocher, D. Flynn, and M. Budagavi, "Common test conditions for point cloud compression," *Document ISO/IEC JTC1/SC29/WG11 w17766, Ljubljana, Slovenia*, 2018.

PUBLICATION
IV

Neural Network Modeling of Probabilities for Coding the Octree Representation of Point Clouds

E. C. Kaya and I. Tabus

*2021 IEEE 23rd International Workshop on Multimedia Signal Processing (MMSP)*2021, 1–6
DOI: `10.1109/MMSP53017.2021.9733658`

Neural Network Modeling of Probabilities for Coding the Octree Representation of Point Clouds

Emre Can Kaya
Computing Sciences Unit
Tampere University
Tampere, Finland
emre.kaya@tuni.fi

Ioan Tabus
Computing Sciences Unit
Tampere University
Tampere, Finland
ioan.tabus@tuni.fi

***Abstract*—This paper describes a novel lossless point cloud compression algorithm that uses a neural network for estimating the coding probabilities for the occupancy status of voxels, depending on wide three dimensional contexts around the voxel to be encoded. The point cloud is represented as an octree, with each resolution layer being sequentially encoded and decoded using arithmetic coding, starting from the lowest resolution, until the final resolution is reached. The occupancy probability of each voxel of the splitting pattern at each node of the octree is modeled by a neural network, having at its input the already encoded occupancy status of several octree nodes (belonging to the past and current resolutions), corresponding to a 3D context surrounding the node to be encoded. The algorithm has a fast and a slow version, the fast version selecting differently several voxels of the context, which allows an increased parallelization by sending larger batches of templates to be estimated by the neural network, at both encoder and decoder. The proposed algorithms yield state-of-the-art results on benchmark datasets. The implementation will be available at https://github.com/marmus12/nnctx**

***Index Terms*—Lossless Compression, Context Coding, Point Cloud, Arithmetic Coding, Point Cloud Compression**

I. Introduction

In the recent years, point cloud compression (PCC) has became a very active field of study, in an effort to provide efficient coding solutions for the very large point clouds available nowadays. The major contributions to the area are the standardization projects initiated by JPEG [1] and MPEG [2], from which the video PCC (V-PCC) and the geometry PCC (G-PCC) standards are already finalized. Meanwhile, several contributions have appeared in the technical literature, showing improvements over the standardized solutions, for some specific classes of point clouds. Encoding the geometry of voxelized point clouds is a first task, solved both in V-PCC and G-PCC, and for which several recent publications provided alternative solutions, see e.g.: the lossless codec using dyadic decomposition [3]; the bounding volumes by depthmap projections BVL [4]; and more recently, VoxelDNN [5] and MSVoxelDNN [6], based on deep neural networks for providing the arithmetic coder with coding probabilities.

We propose a neural network based lossless coder for voxelized point clouds. In the algorithm, the internal representation for point clouds is selected to be the octree model, offering multiresolution reconstructions, where at resolution r all points are given with a precision of r bits per dimension. In the octree model each point from the resolution $r-1$ is split into 8 candidate points at resolution r, and specifying for each of these 8 points the occupancy status by 1 bit, one can retrieve all points at resolution r. Iteratively in the same way, one obtains all resolutions up to the final resolution R. Hence, the octree representation recursively constructs the set of points, by specifying an octet for each node at the current resolution. What needs to be encoded for transmitting the point cloud is the octet that represents the splitting pattern at each node, for each resolution level, starting from resolution 0 up to resolution $r-1$.

Encoding of the splitting octet can be done in several ways. For example, G-PCC encodes the splitting (octet) pattern $[b_1, b_2, \ldots, b_8]$ at a voxel n_k at resolution $r-1$, by considering for each bit b_i a context defined based on the following: the occupancy status of the neighbors of the current voxel n_k to be splitted, the location i of the bit b_i inside the octet, the values of the already encoded bits $b_1, \ldots, b_{i-1}$, and some already encoded splitting patterns at some neighbor voxels of n_k. These contexts are further merged, e.g., based on rotation invariance, in order to obtain the most relevant probability models at each context. In here we do not use the context construction of G-PCC, specifically we do not use the probability distribution $p(b_i|b_1, \ldots, b_{i-1}, C)$ conditional at some collapsed context C, but instead we associate each b_i to the corresponding voxel at resolution r, and establish the context $\mathcal{C}$ in the same way, uniformly for all i, based on spatially neighbor voxels in the resolution $r-1$ and r, as we explain below. This has the effect of using a single conditional distribution $p(b_i|\mathcal{C})$ instead of 8 distinct conditional distributions.

VoxelDNN [5], a recent DNN-based method, generated probabilities for the occupancy of voxels, using a wider conditional template than in the octree model, by splitting the space in very large cubes, e.g. $64 \times 64 \times 64$, and generating conditional probabilities $p(b_i|b_1, \ldots, b_{i-1})$ for each of $i = 1, \ldots, 64^3$, hence for each voxel inside the large cube. Each such specific distribution leads to an equivalent template with variable 3D shape, handled during the training process by masked convolutions. The equivalent template becomes asymmetrical with respect to the voxel b_i to be encoded, especially near the facets of the $64 \times 64 \times 64$ cube. In our

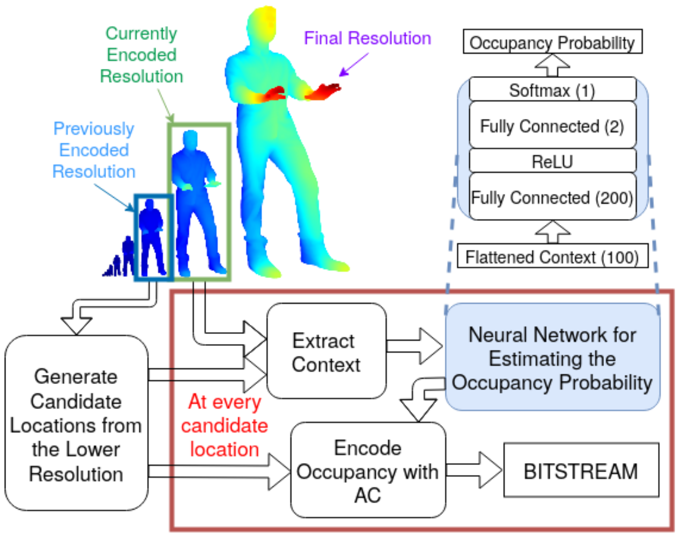

Currently Encoded Resolution
Final Resolution
Previously Encoded Resolution
Occupancy Probability
Softmax (1)
Fully Connected (2)
ReLU
Fully Connected (200)
Flattened Context (100)
Extract Context
Neural Network for Estimating the Occupancy Probability
Generate Candidate Locations from the Lower Resolution
At every candidate location
Encode Occupancy with AC
BITSTREAM

Fig. 1. The proposed lossless encoding scheme.

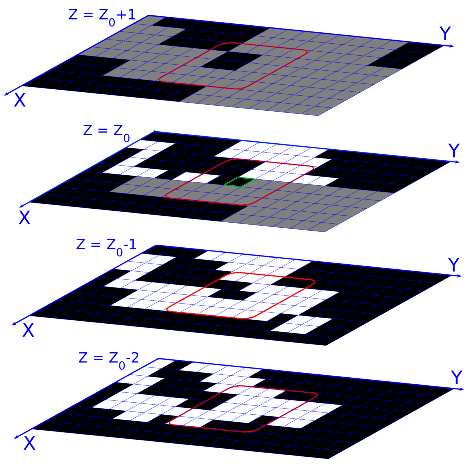


Fig. 2. Context definition in the proposed NNOC. Black locations are unoccupied voxels and those that fall into the red bordered windows are represented each by a 0 in the context vector. White and gray locations are occupied and candidate voxels, respectively. Each of those that are contained in the red bordered windows are represented by a 1 in the context vector. The currently encoded/decoded voxel is shown with a green border.

approach we keep the template of the context the same for most voxels to be encoded, with asymmetries in the template occurring only at the boundary of the overall bounding cube for the point cloud.

II. Proposed Method

The proposed method, which we dub NNOC, is illustrated in Fig. 1. When encoding a point cloud with r bits resolution, the encoder encodes additionally all the $(r-1)$ lower resolution versions. Initially, the 2 bits resolution point cloud, having $4\times4\times4$ voxels, is simply encoded in 64 bits where each bit represents the occupancy of a voxel. Then, the encoding and decoding proceeds by inferring, from the voxels of the point cloud at 2 bits resolution, the possibly occupied (candidate) locations in 3 bits resolution. In general, for every voxel in a resolution level $r-1$, there are 8 possibly occupied voxels in resolution r. Hence, if the 2 bits resolution point cloud has n_p points, there are $8n_p$ candidate locations for the 3 bits resolution. The locations other than the candidate locations are known to be unoccupied.

After marking all candidate voxels at resolution r starting from the known voxels at resolution $r-1$, the encoder and decoder proceed to encoding the occupancy status of these candidate voxels, by scanning the candidate locations in a given scanning order (explained below), and at every candidate location a context is constructed, identically at encoder and decoder, encompassing voxels from resolution r that are either candidates or that have their occupancy status known (since they were already scanned). The scanning order of the candidate pixels at resolution r is defined by considering the regular lexicographic one, considering one-by-one the sections across the point cloud at planes $z=z_0$, and in each section scanning is done row-wise (interpreting the section as an image with row index x and column index y). The scanning order establishes the causality status for the voxels in the considered contexts. A neural network (NN) model is employed to estimate the probability of occupancy of a candidate voxel given the voxel's context. Then, the occupancy of the candidate voxel is encoded by 2-symbol arithmetic coding using the probability distribution generated by the network. The procedure described above is repeated for all the resolutions starting from 3 bits octree depth until the final resolution of the input point cloud is encoded. Apart from the occupancies of voxels, a necessary side information is the binary occupancy status of each section, stating whether the section contains any points. This is expressed in a binary vector having each element associated to one section, which is encoded with run-length encoding.

A. *Collecting the contexts of candidate voxels*

Our approach is based on the octree structure and hence we need to encode each bit of the splitting pattern at each node. What first distinguishes our approach is that at each resolution $r-1$, we traverse the existing nodes and encode their splitting patterns in a two-pass manner as explained next. Suppose we have all nodes at resolution $r-1$, and need to transmit the splitting patterns to reconstruct resolution r. For each node $(x_0^{[r-1]}, y_0^{[r-1]}, z_0^{[r-1]})$, we have 8 possibly occupied children voxels at resolution r, namely $(2x_0^{[r-1]}+\alpha, 2y_0^{[r-1]}+\beta, 2z_0^{[r-1]}+\gamma)$, with $\alpha,\beta,\gamma\in\{0,1\}$, and we store all these points in a sorted list (in the sorting, key z has the highest priority). Having the candidate voxels sorted by section enables us to fetch the section points efficiently. We then reconstruct the points at resolution r, plane by plane, by keeping a 4 sections buffer (two already encoded sections, the current one, and the next one not encoded yet) to which we associate 4 binary images as depicted in Fig. 2. After the status of all candidates in the section $z=z_0$ is encoded, the buffer slides upwards by 1 section, to encode the occupancies of the section $z=z_0+1$.

In terms of the octree coding, what is peculiar in our approach is that we transmit at each resolution $r-1$ the occupancy patterns of the nodes lying in the plane $z = z_0^{[r-1]}$, but in two phases: First we transmit the 4 bits of the splitting pattern having $\gamma = 0$ (corresponding to the lower plane in the resolution r), and after we finish transmitting all these half-splitting patterns for all nodes, we continue transmitting the remaining half-splitting patterns, having $\gamma = 1$ (corresponding to the higher plane in the resolution r). The process is repeated for the plane $z = z_0^{[r-1]} + 1$. In G-PCC for instance, the scanning order is different, since one transmits at once the eight bits corresponding to the occupancy of the children of $(x_0^{[r-1]}, y_0^{[r-1]}, z_0^{[r-1]})$, and only then the process moves to a different node at resolution $r-1$.

In the four sections of the buffer, we mark the current state in the reconstruction process: For the past two sections we know the true occupancy status of all the candidate points, so we mark in the past two sections the true occupancy of voxels at resolution r. In the next section $z = z_0 + 1$, we do not yet know any occupancy status, but we know the status of being a candidate, which is marked on the binary image of the section.

We therefore have organized in a convenient way the scanning order for traversing the candidate voxels at each resolution level, in the following order: section-by-section (traversing all voxels lying on a section $z = z_0$), and in each such 2D section taking the scanning order to be row-by-row. Using such a scanning order, the current candidate (x_0, y_0, z_0) whose occupancy needs to be encoded, has some of its 3D neighbors already encoded (they are at those voxels belonging to sections $z = z_0 - 1$ and $z = z_0 - 2$, and from the section $z = z_0$ the voxels having the row index $x < x_0$, and finally those voxels having $x = x_0$, $y < y_0$ and $z = z_0$). The other neighbor voxels either are not candidates (they are not children of some existing point in the resolution $r-1$), or are known to be unoccupied or they are candidates but their occupancy is not known yet.

We define the context as a cuboid with $5 \times 5 \times 4$ 3D neighbors of (x_0, y_0, z_0) at resolution r as shown in Fig. 2. We note that this cuboid changes its position at each encoding of a candidate occupancy, because we place it such that it keeps its alignment with the current candidate. Due to the selected scanning order, the status of being already encoded or not for each pixel remains the same, so the contexts' elements keep the same type of information or significance. For example, the neighbor $(x_0 - 2, y_0 - 2, z_0 - 2)$ has always its occupancy known (either because it was not a candidate for testing i.e., not being a child of an existing node at $r-1$), or because it was a candidate and we have transmitted its occupancy status. So the state of $(x_0 - 2, y_0 - 2, z_0 - 2)$ can be either 0, if it is not occupied, or 1 if it is occupied. On the other hand, $(x_0, y_0 + 2, z_0)$ is not encoded yet. We already can know its occupancy, if it was not a candidate then for sure it is not occupied) but if it was a candidate, its occupancy is not known yet. So the state that can be associated to $(x_0, y_0 + 2, z_0)$ is 1 if it is a candidate, and 0 if it is not.

Algorithm 1 Encoding with NNOC

```
Require: A point cloud $\mathcal{P}_R$ with resolution $R$ bits/dimension
  1. Construct lower resolution point clouds $\mathcal{P}_{R-1}, \dots, \mathcal{P}_2$ representing the nodes at octree depth level $R-1, \dots, 2$.
  2. Encode $\mathcal{P}_2$ in 64 bits
  3. Encode iteratively $\mathcal{P}_3$ to $\mathcal{P}_R$ as follows:
  for $r = 3, \dots, R$ do
    3.1 Generate for each point $P \in \mathcal{P}_{r-1}$ the eight candidate voxels in the resolution $r$, resulting in the set of candidate points $\mathcal{P}_r^C$
    3.2 Traverse candidates $\mathcal{P}_r^C$ section-by-section and encode occupancies as follows:
    for $z_0 = 0, \dots, 2^r - 1$ do
      3.2.1 Construct 4 binary images as in Fig. 2
      3.2.2 Traverse the candidates for which $z = z_0$ as described in Section II.A.
      for all $(x_c, y_c, z_0) \in \mathcal{P}_r^C$ do
        3.2.2.1 Extract the context vector $\mathcal{C}$ from 4 images
        3.2.2.2 Obtain the coding distribution $NN(\mathcal{C})$
        3.2.2.3 Encode the occupancy $O(x_c, y_c, z_0)$ using $NN(\mathcal{C})$
        3.2.2.4 Save the true occupancy in the image $z = z_0$
      end for
    end for
  end for
```

To conclude, in NNOC we consider a 100-element binary context vector $\mathcal{C}$, where each element corresponds to the binary status of one location in the $5 \times 5 \times 4$ template and this status means different things for different voxels: For the already encoded voxels it means occupancy, for not yet encoded voxels it means candidacy.

Due to practical reasons, the context vector contains the candidacy of also the currently encoded voxel (x_0, y_0, z_0) which is always 1. It is experimentally found that accessing a block of voxels in the current section altogether and writing them to context vector is running faster than going through the positions inside the block one by one to exclude the currently encoded position.

The significance of the neighbors within the context remains the same for all contexts, and for all resolutions. So we decide to define the probability distribution for the voxel (x_0, y_0, z_0) being occupied ($O(x_0, y_0, z_0) = 1$, or not, $O(x_0, y_0, z_0) = 0$, $p(O(x_0, y_0, z_0) = 1) = NN(\mathcal{C})$ and propose to implement this function using a neural network, having parameters obtained in a training process. The structure of the algorithm is presented in Algorithm 1.

B. Generating the coding distribution by a NN having the context $\mathcal{C}$ at its input

The input to the neural network is a binary vector consisting of occupancies or candidancies (as explained in II.A) in the causal context of the location being encoded/decoded. We employ a 3 dimensional context such that the number of inputs elements to the NN is $n_C = 5 \times 5 \times 4 = 100$. In order to ensure a reasonable encoding and decoding speed, the neural network structure is kept simple, adopting here a Multilayer Perceptron consisting of 2 fully connected layers. The first layer has $2n_C = 200$ neurons with ReLU activations and the 2nd layer has 2 neurons, with outputs α_1 and α_2, and finally there is a softmax activation giving as output

$$p(O(x_0, y_0, z_0) = 1) = \frac{e^{\alpha_1}}{e^{\alpha_1} + e^{\alpha_2}}. \qquad (1)$$

The output of the neural network is interpreted as an estimation of the probability distribution of occupancy of the current location given its causal context. This floating point estimation provided by the network is multiplied by 2^{14} and rounded to yield integer counts which are used in arithmetic coding.

The training set for the neural network consists of causal contexts that have occurred at least once in the point clouds selected for training. Let no_{0i} and no_{1i} denote the number of occurences of the i'th context in a training batch of contexts (including $b.size$ such contexts), where the current "true" location's occupancy was 0 or 1, respectively. Let $p_{1i} = p(O(x_0, y_0, z_0) = 1)$ be the output of the NN and $p_{0i} = 1 - p_{1i}$. The neural network is trained to minimize the following criterion:

$$Loss = -\sum_{i=1}^{b.size} no_{0i} \log_2 p_{0i} + no_{1i} \log_2 p_{1i}. \qquad (2)$$

Note that, if the batch would contain all the causal contexts with corresponding number of occurrences in one point cloud, $Loss$ would be equal to the codelength obtained when compressing that point cloud with arithmetic coding using the estimated probabilities p_{0i} and p_{1i}. Therefore, $Loss$ reflects our goal to minimize the codelength in the most natural way. Although the loss formulation looks quite similar to the commonly used cross entropy loss function that is encountered in supervised classification schemes, there are important distinctions worth pointing out: The contexts that occur in the training set are only a small portion of the set of all possible contexts. However, what is expected from the network is to generate a probability distribution for any input context, whether seen or not. Another distinction is that the number of occurrences shouldn't be perceived as a ground truth information. They are obtained from a number of point clouds which are likely to have different number of occurrences than the input point cloud to be compressed.

The network is trained with batches formed by randomly selected contexts possibly coming from different training point clouds.

C. Parallelization and the fast model

Parallelization can be utilized efficiently at the encoder, where all the contexts at one section $z = z_0$, can be collected and fed to the network to obtain the probability distributions for all section candidates at once. For high resolutions, this may result in efficient computation of probabilities in GPU in large batches.

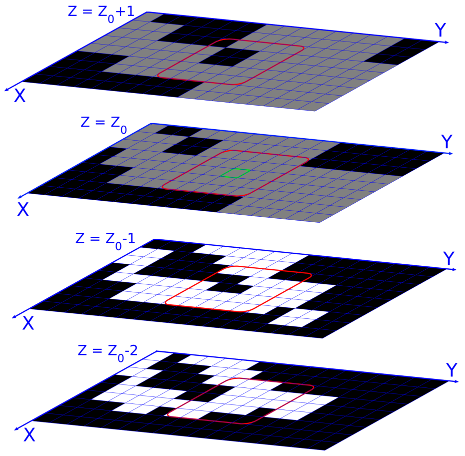


Fig. 3. Context definition in fNNOC (the faster version of the proposed NNOC). Black locations are unoccupied voxels and those that fall into red bordered windows are represented each by a 0 in the context vector. White and gray locations are occupied and candidate voxels, respectively. Each of those that are contained in the red bordered windows are represented by a 1 in the context vector. Currently encoded/decoded voxel is denoted with a green window.

However, at the decoder the situation is different. NNOC requires for forming the context at (x_0, y_0, z_0) the knowledge about the occupancy of $(x_0, y_0 - 1, z_0)$, which needs to be decoded by arithmetic coding. The only possible sequencing of operations at the decoder is: The decoding of the occupancy of $(x_0, y_0 - 1, z_0)$ is done and it is used to define the context $\mathcal{C}(x_0, y_0, z_0)$. This context is fed to the network which generates the coding distribution of $p(O(x_0, y_0, z_0) = 1) = NN(\mathcal{C})$ that is used to decode the occupancy of (x_0, y_0, z_0) and so on. In this manner, the network operates with batches of size 1, which is simply sequential, with no parallelism. In order to accelerate the process and to be able to utilize batches of contexts at the decoder, we modify the context in NNOC so that it does not need to make use of the current resolution occupancies at section $z = z_0$. This is realized in the faster version of NNOC called fNNOC. The collection of the context in fNNOC is depicted in Fig. 3. This new type of context is for sure less informative and fNNOC performs worse than NNOC in terms of bpov. In fNNOC, the decoder collects all candidates from the current section at once, and then it feeds them to the GPU implementation of the NN, in a batch having the size equal to the number of candidate voxels in section $z = z_0$. As a consequence, durations of encoding and decoding with fNNOC are similar.

III. EXPERIMENTAL RESULTS

Training and tests are performed on two publicly available datasets: Microsoft voxelized upper bodies (MVUB) [8] and 8i voxelized full bodies (8i) [9]. The method is implemented in Python and Tensorflow. Arithmetic coding is adapted from an open source Python implementation [11]. The neural network is trained with contexts collected from 18 randomly chosen frames in Andrew10, David10, Sarah10 sequences (6 from

TABLE I
COMPARING **VOXELDNN** [5] AND **NNOC** (PROPOSED) IN TERMS OF AVERAGE RATE [BPOV] AND GAINS OVER **G-PCC** [7]

Point Cloud(s)	**Number of frames**	**Single Frame**			**Average over all frames**		
		G-PCC	**VoxelDNN**	**Gain over G-PCC**	**G-PCC**	**NNOC**	**Gain over G-PCC**
Microsoft Voxelized Upper Bodies [8]							
Phil9	245	1.2284	0.9201	25%	1.1785	0.8095	**31%**
Phil10	245	1.1617	0.8307	28%	1.135	0.7817	**31%**
Ricardo9	216	1.0422	0.7173	31%	1.0836	0.6805	**37%**
Ricardo10	216	1.0672	0.7533	29%	1.0723	0.7006	**35%**
Average	-	1.1248	0.8053	28%	1.1173	0.7431	**33%**
8i Voxelized Full Bodies [9]							
Loot10	300	0.9524	0.6387	33%	0.9801	0.5904	**40%**
Redandblack10	300	1.0889	0.7317	33%	1.1047	0.723	**35%**
Boxer9	1	1.0815	0.756	30%	0.9683	0.6439	**34%**
Boxer10	1	0.9	0.59	34%	0.9619	0.5507	**43%**
Thaidancer9	1	1.0677	0.8078	24%	1.1253	0.7309	**35%**
Thaidancer10	1	-	-	-	1.0061	0.6839	32%
Average	-	1.0476	0.7334	30%	1.0244	0.6538	36%

TABLE II
COMPARING **MSVOXELDNN** [6] AND **FNNOC** (PROPOSED) IN TERMS OF AVERAGE RATE [BPOV] AND GAINS OVER **G-PCC** [7]

Point Cloud(s)	**Number of frames**	**Single Frame**			**Average over all frames**		
		G-PCC	**MSVoxelDNN**	**Gain over G-PCC**	**G-PCC**	**fNNOC**	**Gain over G-PCC**
Microsoft Voxelized Upper Bodies [8]							
Phil9	245	-	-	-	1.1785	0.9974	15%
Phil10	245	1.1617	1.02	**12%**	1.135	1.0206	10%
Ricardo9	216	-	-	-	1.0836	0.861	21%
Ricardo10	216	1.0672	0.95	11%	1.0723	0.941	**12%**
Average	-	1.1249	0.985	11%	1.1174	0.955	14%
8i Voxelized Full Bodies [9]							
Loot10	300	0.9524	0.73	23%	0.9801	0.7427	**24%**
Redandblack10	300	1.0889	0.87	20%	1.1047	0.8661	**22%**
Boxer9	1	-	-	-	0.9683	0.7466	23%
Boxer10	1	0.9	0.7	22%	0.9619	0.6815	**29%**
Thaidancer9	1	-	-	-	1.1253	0.8672	23%
Thaidancer10	1	1.00	0.85	15%	1.0061	0.8069	**20%**
Average	-	0.9853	0.7875	20%	1.0244	0.78517	23.5%

TABLE III
AVERAGE RATE [BPOV] RESULTS ON POINT CLOUDS FROM CAT1A [10]

		NNOC	**Other codecs**		**Fast versions of NNOC**					
	Bitdepth		**G-PCC**	**BVL** [4]	**fNNOC**	**fNNOC1**	**fNNOC2**	**fNNOC3**	**fNNOC4**	**fNNOC5**
Input context size		100			100	75	50	36	100	100
# of neurons by layer		(200,2)			(200,2)	(150,2)	(100,2)	(72,2)	(200,1)	(200,200,2)
basketball player	11	**0.5934**	0.885	0.852	*0.6908*	0.8924	1.2689	0.9098	0.6945	0.7083
dancer	11	**0.5751**	0.876	0.826	*0.6907*	0.8874	1.2443	0.8931	0.6936	0.7037
facade 00064	11	**1.1053**	1.1969	1.3331	1.2216	1.3888	1.4973	1.3102	*1.2098*	1.2291
queen	10	**0.6897**	0.7817	0.7883	*0.9196*	1.1327	1.4814	1.1773	0.9309	0.9390
redandblack	10	**0.7353**	1.1055	1.0418	0.8854	1.107	1.514	1.1009	0.8932	*0.8738*
loot	10	**0.5989**	0.9818	0.8991	0.7615	0.975	1.3785	0.9798	0.7761	*0.7557*
Average		**0.7163**	0.971	0.957	*0.8616*	1.0639	1.3974	1.0619	0.8663	0.8683

each) from MVUB and 18 randomly chosen frames in Longdress and Soldier sequences (9 from each) from 8i. All of the training contexts are collected from the original resolution of the point clouds (10 bits). The training and validation sequences are not used for tests. Our selection of training data is similar to the recently introduced VoxelDNN [5] and its fast version MSVoxelDNN [6].

For NNOC, the training set contains around 40 million unique contexts with number of occurences ranging from 1 to 278000 whereas for fNNOC, there are 31 million training contexts with number of occurences ranging from 1 to 135000. These are relatively small training sets considering that the number of possible different contexts with 100 elements is about 2^{100}. During training we also employ validation sets through which we decide when to stop training. The validation sets consist of randomly chosen frames from Andrew10 and Soldier sequences. Training was performed using ADAM [12] optimizer with batches of 30k contexts.

Average bpov results of the proposed NNOC and fNNOC on sequences and individual point clouds from MVUB and 8i

TABLE IV
DURATIONS OF ENCODING AND DECODING WITH NNOC AND fNNOC

		NNOC		fNNOC	
P. Cloud	**B.depth**	**ENC**	**DEC**	**ENC**	**DEC**
phil (1st fr.)	9	1m 17s	8m 53s	42s	46s
loot (1st fr.)	10	3m 11s	19m 31s	1m 46s	1m 53s
basket. player	11	10m 37s	3hrs 21m 42s	5m 44s	6m 7s

datasets are presented in Tables I and II. In Tables I and II, we also show the bpov results obtained with G-PCC, VoxelDNN and MSVoxelDNN. Since the VoxelDNN and MSVoxelDNN bpovs are for single frames from the sequences, they are not fully comparable with the bpovs for the whole sequences. For a better comparison basis, we show the respective gains in bpov over G-PCC. It should be noted that, Boxer and Thaidancer are in fact individual point clouds (single frames) with bitdepths of 12. We present results for their downsampled versions with bitdepths 9 and 10.

In Table III, left side, we compare the average rates obtained for some of the CAT1A [10] point clouds with G-PCC, BVL [4] which is a recent lossless method, the proposed NNOC and fNNOC. It is observed that NNOC outperforms all the other methods on all point clouds. fNNOC outperforms BVL on 4 out of 6 point clouds and G-PCC on 5 out of 6 point clouds. A future study might look more into better ways to consider point clouds for training, so that more of the typical existing datasets can be encoded in an efficient way.

A. Ablation Study

In order to investigate the effects of the selection of context and network structure, we have performed an ablation study where we train 5 fNNOC variants. Variants fNNOC1-3 have smaller input contexts. fNNOC1 has the same context elements as fNNOC, except removing the voxels coming from the future section (z_0+1), hence it has 75 context bits. The second variant called fNNOC2 has the same context elements as fNNOC1 except those coming from the section z_0-2, so it has a context vector of length 50. The third variant fNNOC3 has the same 4 context sections as in fNNOC but the context window size is 3×3 instead of 5×5, so it contains $3 \times 3 \times 4 = 36$ context elements in total. fNNOC4 has a single output neuron representing the occupancy probability, followed by sigmoid instead of 2 neurons and softmax. Finally, fNNOC5 has 2 hidden layers instead of 1. The average rates obtained with the variants are presented in Table III (right side).

Comparing fNNOC, fNNOC1, fNNOC2 and fNNOC3, one can see that the currently selected context $5 \times 5 \times 4 = 100$ is justified, improving consistently and significantly over the less complex versions. Moreover, comparing fNNOC and fNNOC4, it is observed that having 2 output neurons with softmax instead of 1 with sigmoid yields a slightly better performance. Comparing fNNOC and fNNOC5, it is seen that adding one extra hidden layer does not have a major impact on the bitrate performance.

B. Encoding and Decoding Times

The encoding and decoding times of NNOC and fNNOC for point clouds with three different bitdepths are presented in Table IV. While the reported times are not comparable with G-PCC, which can encode and decode in the order of seconds, it might be worth emphasizing that the implementation is done in a high-level environment which is not ideal for speed.

IV. CONCLUSIONS

We proposed a lossless compression method that utilizes octree representation and neural network estimation of occupancy probability distributions for splitting the octree nodes, based on contexts obtained through combining information from current and past resolutions. The method has a fast and a slow version and is shown to provide much better compression results than G-PCC, however at a complexity cost which is currently very high. The proposed method compares favorably with other recently proposed neural network solutions. Further study is needed for expanding the solutions to larger classes of point clouds and for higher ranges of resolutions.

REFERENCES

[1] T. Ebrahimi, S. Foessel, F. Pereira, and P. Schelkens, "JPEG Pleno: Toward an efficient representation of visual reality," *IEEE Multimedia*, vol. 23, no. 4, pp. 14–20, 2016.
[2] 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.
[3] E. Peixoto, "Intra-frame compression of point cloud geometry using dyadic decomposition," *IEEE Signal Processing Letters*, vol. 27, pp. 246–250, 2020.
[4] E.C. Kaya, S. Schwarz, and I. Tabus, "Refining the bounding volumes for lossless compression of voxelized point clouds geometry," *arXiv preprint arXiv:2106.00828*, 2021.
[5] D. T. Nguyen, M. Quach, G. Valenzise, and P. Duhamel, "Learning-based lossless compression of 3d point cloud geometry," in *2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 4220–4224.
[6] D. T. Nguyen, M. Quach, G. Valenzise, and P. Duhamel, "Multiscale deep context modeling for lossless point cloud geometry compression," *arXiv preprint arXiv:2104.09859*, 2021.
[7] Moving Picture Experts Group, "TMC13," https://github.com/MPEGGroup/mpeg-pcc-tmc13, Accessed: 2020-03-20.
[8] C. Loop, Q. Cai, S. O. Escolano, and P. A. Chou, "Microsoft voxelized upper bodies - a voxelized point cloud dataset," *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document m38673/M72012*, 2016.
[9] E. d'Eon, B. Harrison, T. Myers, and P. A. Chou, "8i voxelized full bodies - a voxelized point cloud dataset," *ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document WG11M40059/WG1M74006*, 2017.
[10] S. Schwarz, G. Martin-Cocher, D. Flynn, and M. Budagavi, "Common test conditions for point cloud compression," *Document ISO/IEC JTC1/SC29/WG11 w17766, Ljubljana, Slovenia*, 2018.
[11] Project Nayuki, "Reference arithmetic coding," https://github.com/nayuki/Reference-arithmetic-coding, Accessed: 2021-04-03.
[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

PUBLICATION

V

Lossless Compression of Point Cloud Sequences Using Sequence Optimized CNN Models

E. C. Kaya and I. Tabus

IEEE Access 10.(2022)
DOI: 10.1109/ACCESS.2022.3197295

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 17 July 2022, accepted 1 August 2022, date of publication 8 August 2022, date of current version 12 August 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3197295

RESEARCH ARTICLE

Lossless Compression of Point Cloud Sequences Using Sequence Optimized CNN Models

EMRE C. KAYA AND IOAN TABUS, (Senior Member, IEEE)
Computing Sciences Unit, Tampere University, 33720 Tampere, Finland
Corresponding author: Emre C. Kaya (emre.kaya@tuni.fi)

ABSTRACT In this paper we propose a new paradigm for encoding the geometry of dense point cloud sequences, where a convolutional neural network (CNN), which estimates the encoding distributions, is optimized on several frames of the sequence to be compressed. We adopt lightweight CNN structures, we perform training as part of the encoding process and the CNN parameters are transmitted as part of the bitstream. The newly proposed encoding scheme operates on the octree representation for each point cloud, consecutively encoding each octree resolution level. At every octree resolution level, the voxel grid is traversed section-by-section (each section being perpendicular to a selected coordinate axis), and in each section, the occupancies of groups of two-by-two voxels are encoded at once in a single arithmetic coding operation. A context for the conditional encoding distribution is defined for each two-by-two group of voxels based on the information available about the occupancy of the neighboring voxels in the current and lower resolution layers of the octree. The CNN estimates the probability mass functions of the occupancy patterns of all the voxel groups from one section in four phases. In each new phase, the contexts are updated with the occupancies encoded in the previous phase, and each phase estimates the probabilities in parallel, providing a reasonable trade-off between the parallelism of the processing and the informativeness of the contexts. The CNN training time is comparable to the time spent in the remaining encoding steps, leading to competitive overall encoding times. The bitrates and encoding-decoding times compare favorably with those of recently published compression schemes.

INDEX TERMS Convolutional neural networks, lossless geometry compression, octree coding, point cloud compression.

I. INTRODUCTION

The compression of the voxelized point clouds recently became a hot research topic, owing to the need to develop immersive technologies, underlined, e.g., in the programs launched by the MPEG [1] and JPEG [2] standardization bodies, which have already resulted in two well-engineered standards, V-PCC [3] and G-PCC [4] (having the test model TMC13 [5]). The scientific literature has witnessed a strong interest in improving the compression performance of G-PCC, with many scholarly contributions in recent years, e.g., [6]–[22]. These methods differ in several aspects, such as the representation used for the point cloud (e.g., octree [15]–[20], dyadic decomposition [6], [7], [12] or projections onto 2D planes [13], [23]), the selection of the context used for the conditional probability model for arithmetic coding and the method to define the symbols to be encoded.

The probability model can be based on adaptively maintained counts for various contexts [4], [7] or on a neural network (NN) model [11], [17], having a binary context at the input and the probability mass function of the symbol at the output.

In the last few years, machine learning approaches using neural networks have been proven to be competitive for both lossy [14], [16], [20]–[22] and lossless [11], [17], [20], [24] point cloud geometry compression. A comprehensive survey of the recent methods with a focus on the learning-based approaches is provided in [25].

In [14], an autoencoder architecture involving 3D convolutional layers is employed to generate a latent representation

The associate editor coordinating the review of this manuscript and approving it for publication was Gulistan Raja.

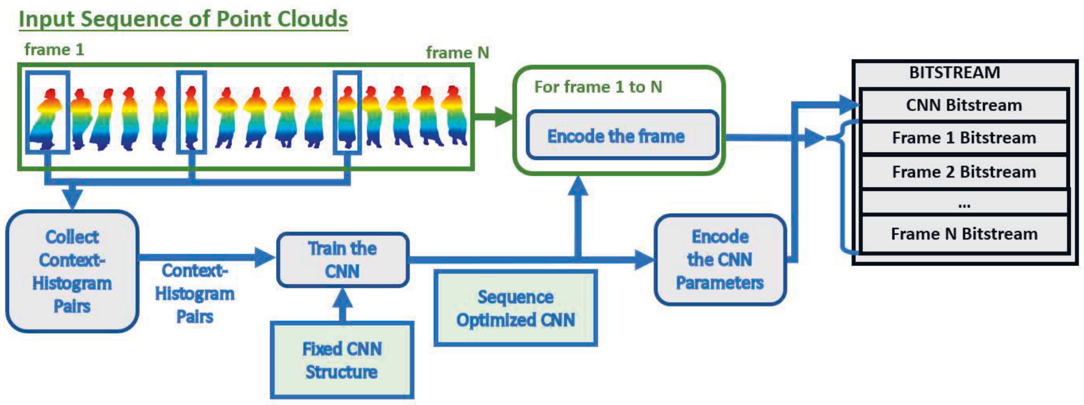


FIGURE 1. **Overview of the proposed Specifically Trained Model (STM) compression method for encoding a point cloud sequence.**

of the point cloud, which is further compressed using range coding. In [21], a variational autoencoder model is employed in an end-to-end learning scheme. In [22], adaptive octree-based decomposition of the point cloud is performed prior to encoding with a multilayer perceptron-based end-to-end learned analysis-synthesis architecture. VoxelDNN [11] uses 3D masked convolutional filters to enforce the causality of the 3D context from which the occupancy of $64 \times 64 \times 64$ blocks of voxels are estimated. VoxelContext-Net [20] employs an octree-based deep entropy model for both dynamic and static LIDAR point clouds. NNOC [17] operates on the octree representation, where hybrid contexts are formed by combining the information from two consecutive resolution levels. In [24], a deep generative model is employed for lossless geometry compression.

As a variation from the previous approaches that utilize neural networks, we optimize a specific neural network model for the sequence to be encoded. In the proposed scheme, the optimization (training) of the network is a part of the encoding stage, and the optimized CNN model parameters are transmitted as a header to the decoder, being then used for decoding any point cloud from the sequence.

We consider here two distinct paradigms for using a neural network as a coding probability model. In the first, which we dub *Generically Trained Model* (GTM), some generic training set is selected and is used for optimizing an NN model to be used by both the encoder and decoder for all the compression tasks that will be required in the future [8], [11], [14], [16], [17], [20]–[22], [24], [26], [27]. The compression performance of the methods involving this approach depends on the selection of a suitable training set. The model is an integral part of the encoding program, and an identical copy is assumed to exist in the decoding program, so the NN model is not considered as a part of the encoded data. Hence, the model can be made as complex as needed, since its size does not contribute to the size of the bitstream. Additionally, the training time of the model is not accounted for in the encoding time, although it can be rather important, of the order of hours.

Here, we propose a second paradigm, *Specifically Trained Model* (STM), where the NN model to be used for the compression of a sequence of point clouds (PC) is optimized for the sequence and is transmitted as a part of the encoded stream. One cannot simply adhere to this paradigm by selecting the same model structure as in the GTM case and train it on the sequence, because the complexity-compression ratio trade-off needs to be different. The model needs to be trained quickly, and the cost of transmitting the model parameters needs to be sufficiently low. Using the specifically trained model paradigm as a basis for the compression of a point cloud sequence has the advantage that the NN model that generates the coding distribution at each context can be trained to match very closely the real distributions found in the point clouds that form the sequence, even with a less complex NN structure than in the general trained model case. Another advantage is that the point clouds in a sequence are similar, one to another, from the point of view of their real probability distributions at the contexts. Whereas in the case of a general trained model, the distributions generated by the model at each context can differ quite significantly from the distributions learned from the generic training data.

In this paper, we propose a solution that belongs to the STM paradigm and has the following main features: It uses the octree representation [28]; encodes the occupancy of groups of 2×2 voxels at each arithmetic encoding operation; uses a context based on the occupancies of lower resolution voxels (translated as candidate voxels at the current resolution) and on sets of voxels already encoded at the current resolution; computes the probability using a CNN model having at the input a multivalued context (which becomes gradually more relevant along the four phases).

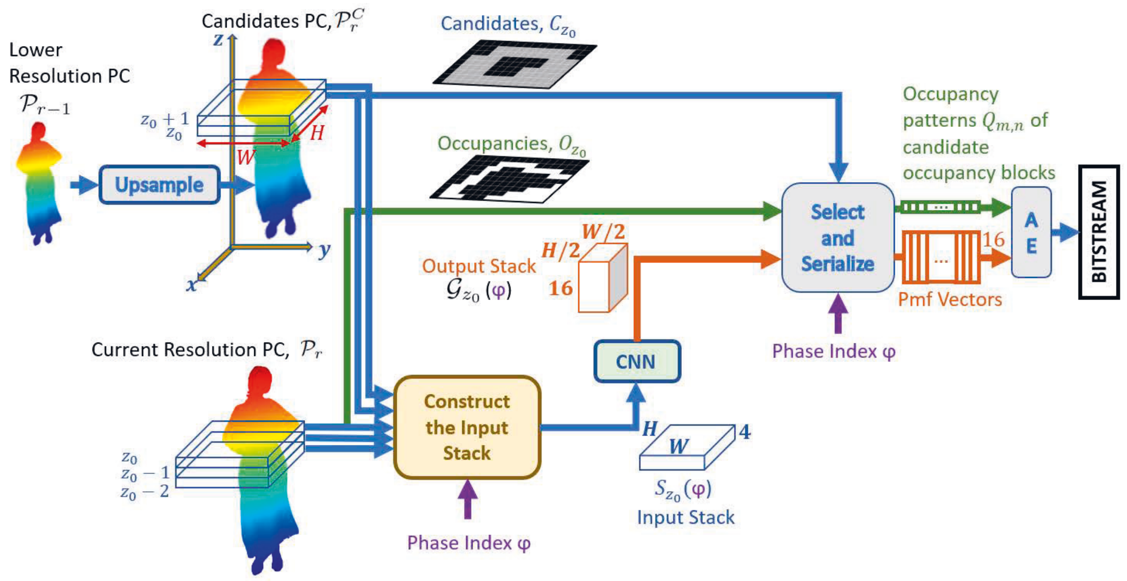

Lower Resolution PC $\mathcal{P}_{r-1}$
Candidates PC, $\mathcal{P}_r^C$
Upsample
Candidates, C_{z_0}
Occupancies, O_{z_0}
Output Stack $\mathcal{G}_{z_0}(\varphi)$
W/2 H/2 16
Select and Serialize
Occupancy patterns $Q_{m,n}$ of candidate occupancy blocks
Pmf Vectors
A E
BITSTREAM
Phase Index φ
CNN
Current Resolution PC, $\mathcal{P}_r$
Construct the Input Stack
$\mathcal{S}_{z_0}(\varphi)$ Input Stack
H W 4
Phase Index φ

FIGURE 2. Encoding of the occupancy image O_{z_0} (the section $z = z_0$ through the point cloud $\mathcal{P}_r$ at resolution r) at phase φ using an arithmetic encoder (AE) for 16-valued symbols. "Construct the Input Stack" is detailed in Fig. 3, CNN is detailed in Fig. 4 and "Select and Serialize" is detailed in Fig. 5.

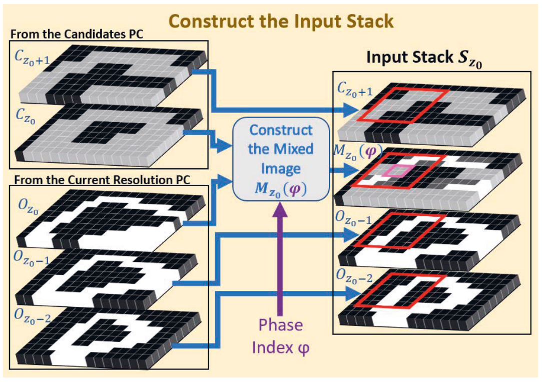

Construct the Input Stack
From the Candidates PC
From the Current Resolution PC
Construct the Mixed Image $M_{z_0}(\varphi)$
Input Stack S_{z_0}
Phase Index φ

FIGURE 3. Construction of the Input Stack $S_{z_0}(\varphi)$. $M_{z_0}(\varphi)$ is formed by combining the available information from the current and the lower resolution (candidates PC). For more detail on how $M_{z_0}(\varphi)$ is formed, see Fig. 7 and Eqn. (3).

The structure of the paper is as follows. We introduce our method in Section II and present the experimental results in Section III, and we present the conclusions in Section IV.

II. PROPOSED METHOD

A. ENCODING A SEQUENCE OF FRAMES BY OPTIMIZING A CNN MODEL ON A FEW FRAMES

We consider lossless compression of the geometry of dense point clouds forming a sequence. Each point cloud in the sequence is referred to as a frame. An overview of the proposed lossless sequence encoding scheme is provided in Fig. 1. The lossless encoding scheme consists of three stages. In the first stage, we collect the contexts and corresponding histograms from a small number of frames. For example, we show in the Experimental Work section the performance when selecting different numbers of training frames at equal temporal distances to each other. More elaborate selection strategies might be considered but we noticed that the overall performance does not improve significantly for the sequences that we experiment with. Each context is associated with a 16-element histogram where each element corresponds to an occupancy pattern as described in Section II-C in detail. This first stage is shown as a block called "Collect Context-Histogram Pairs" in Fig. 1. In the second stage, we train a fixed structure CNN model using the contexts and corresponding histograms that were collected in the first stage (see "Train CNN" in Fig. 1). We emphasize that the structure of the CNN is fixed to make it clear that it is not subject to optimization in the algorithm, however it is chosen *a priori* from a set of possible structures, given the intended performance: for better compression performance one can use a heavier model, while for low overhead of the model bitstream a lighter structure is preferable (e.g., in the case of short sequences). In Section III we illustrate four choices of CNN model structures of different complexities and report their performance. The set of possible structure choices should be known to the decoder and the choice made at the encoder needs to be signaled (by a couple of bits) to the decoder.

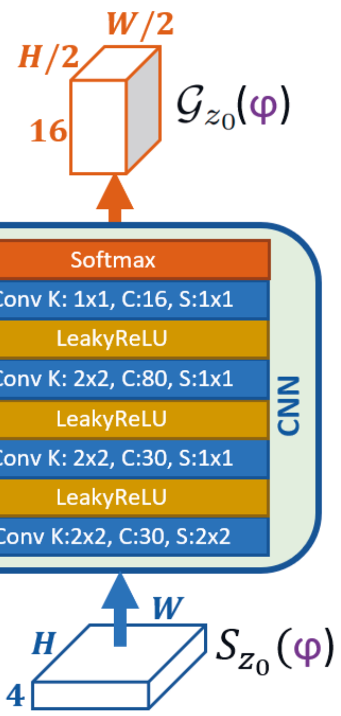


FIGURE 4. Structure of the sequence optimized CNN used for estimating the occupancy probabilities. For the convolutional layers, K denotes the 2D kernel size, C denotes the number of output channels and S denotes the 2D stride.

The optimal parameters (weights and biases) of the CNN resulting from the training stage, are losslessly transmitted, consuming 32 bits for each floating point parameter. This corresponds to the "Encode the CNN Parameters" block in Fig. 1. The decoder has available the structure parameters of the CNN (number of layers, number of channels, strides), and by reading the header transmitted by the encoder, it can reconstruct the CNN model to be exactly the same as the one used by the encoder. The CNN model is used for independently encoding/decoding each frame of the sequence, and since the encoding is done only intraframe, the starts and ends of the bitstream for each frame can be stored (incurring a little overhead) so that each frame can be decoded without the need of decoding the previous frames. In the final stage, we encode the entire sequence frame-by-frame using the same CNN for each frame. The frames can be encoded independently from one another; hence, they can be decoded in any order, independently from one another, resulting in random access to the point clouds of the sequence. We note that the methods using interframe coding do not possess this random access property.

B. ENCODING OF A FRAME

At each frame, the octree representation of the point cloud is processed iteratively at the resolution r, starting from the resolution (octree depth level) $r = 2$ and eventually reaching the original resolution $r = R$, thus enabling lossless reconstruction. First, the PC at $r = 2$ is written to the bitstream in 64 bits, where each bit represents the occupancy of a voxel.

The encoding steps at resolutions higher than 2 are illustrated in Fig. 2. Let $\mathcal{P}_R$ denote the input point cloud, where $\mathcal{P}_r$ with $2 \leq r < R$ is a lower resolution version of $\mathcal{P}_R$. At each resolution level $r \leq R$, we have available at both the encoder and the decoder the point cloud $\mathcal{P}_{r-1}$, and we create from it an upsampled version called $\mathcal{P}_r^C$ by splitting each occupied voxel of $\mathcal{P}_{r-1}$ into eight candidate voxels in $\mathcal{P}_r^C$, now having a resolution of r bits per dimension. The point cloud $\mathcal{P}_r$ is a subset of $\mathcal{P}_r^C$. For encoding $\mathcal{P}_r$, we need to encode and transmit the occupancy status of each candidate voxel in $\mathcal{P}_r^C$. This is performed sequentially by sweeping z_0 along the sweeping dimension z and at each z_0 considering the sectioning by the plane $z = z_0$ of the point cloud $\mathcal{P}_r^C$ and for each candidate voxel transmitting the occupancy status.

To use suggestive geometric interpretation, when encoding the voxels at the section $z = z_0$ in $\mathcal{P}_r$, we refer to the plane in the x and y coordinates as a binary $(W \times H)$-occupancy image O_{z_0}, where $O_{z_0}(x, y) = 1$ indicates that $(x, y, z_0) \in \mathcal{P}_r$ (see Fig. 2). The occupancies of the candidate voxels in the current section $z = z_0$ are encoded in four phases such that, at each phase, only some of the candidate voxels in the current section are encoded. The context used at a phase φ for encoding the candidate voxels is constructed using the most up-to-date information, containing the occupancies that are encoded in the previous phases $1, \ldots, \varphi - 1$. Thus, having four phases instead of a single phase provides more informative contexts for the candidates that are not encoded in the first phase.

Section $z = z_0$ through the upsampled candidate point cloud $\mathcal{P}_r^C$ is called the $(W \times H)$-candidate image, C_{z_0} (see Fig. 2). During the decoding process, the reconstructed occupancy image $R_{z_0}(\varphi)$ (a binary $(W \times H)$-image) is maintained for each of the four phases $\varphi = 1, 2, 3,$ and 4, as described in Subsection II-C. After phases $\varphi = 1, 2, 3$, $R_{z_0}(\varphi)$ is a partial reconstruction of O_{z_0}, and after phase $\varphi = 4$, O_{z_0} is fully reconstructed. In $R_{z_0}(\varphi)$, the decoded occupied voxels (true points) are represented as pixels with a value of 1, and all the remaining pixels have a value of 0. When encoding the pixels of O_{z_0}, the planes O_{z_0-1} and O_{z_0-2} were already encoded, and we rely, when building conditioning distributions by the CNN, on the most relevant available information, namely, the images $C_{z_0+1}, C_{z_0}, O_{z_0-1}, O_{z_0-2}$, and for the information in the current plane $z = z_0$ reconstructed as $R_{z_0}(\varphi)$. The information from the candidate image C_{z_0} and the reconstructed image R_{z_0} are combined into a four-level mixed image, $M_{z_0}(\varphi)$, as described in Subsection II-C.

The mixed image $M_{z_0}(\varphi)$ together with the three binary images C_{z_0+1}, O_{z_0-1} and O_{z_0-2} are used to construct a $4 \times W \times H$ array called Input Stack $S_{z_0}(\varphi)$. Input Stack $S_{z_0}(\varphi)$ is constructed by the block "Construct the Input Stack" in Fig. 2, which is detailed in Fig. 3, and it is input to a CNN for estimating the probabilities of occupancy of the voxels at $z = z_0$.

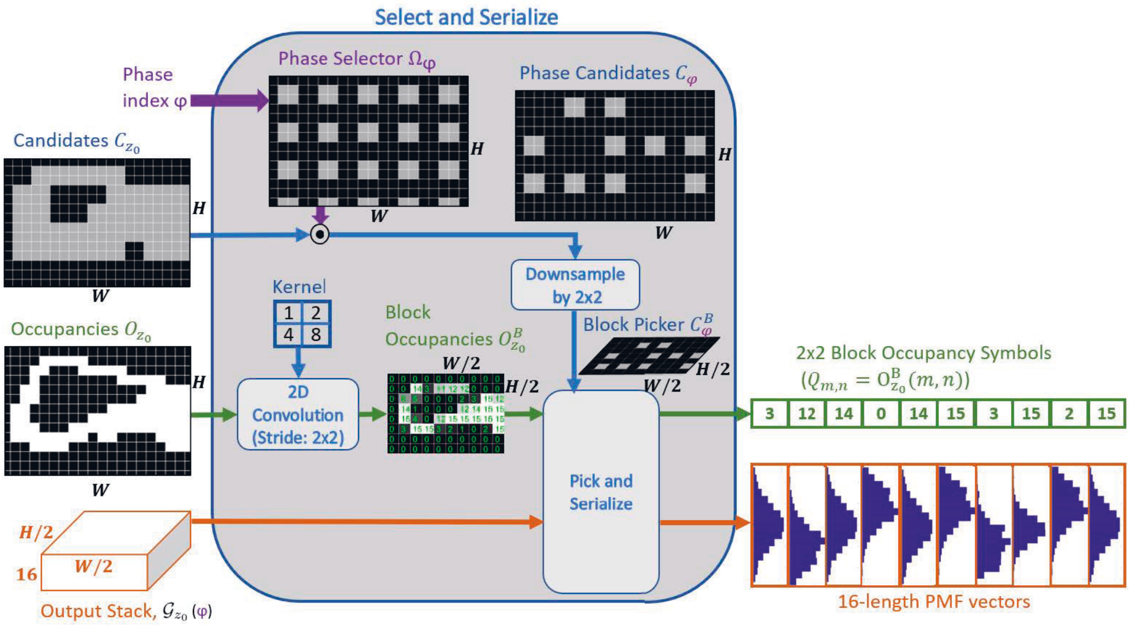

Select and Serialize
Phase index φ
Phase Selector Ω_φ
Phase Candidates C_φ
Candidates C_{z_0}
H
W
Downsample by 2x2
Kernel
1 2
4 8
Block Occupancies $O^B_{z_0}$
Block Picker C^B_φ
Occupancies O_{z_0}
2D Convolution (Stride: 2x2)
W/2
H/2
2x2 Block Occupancy Symbols
$(Q_{m,n} = O^B_{z_0}(m,n))$
3 12 14 0 14 15 3 15 2 15
Pick and Serialize
16
Output Stack, $\mathcal{G}_{z_0}(\varphi)$
16-length PMF vectors

FIGURE 5. The structure of the block "Select and Serialize" from Fig. 2. The candidate voxels encoded in the current phase (φ) are determined by the phase candidate image C_φ, which is obtained by multiplying the two binary images C_{z_0} and Ω_φ. By downsampling C_φ, we obtain the so-called block picker C^B_φ, which is a binary image where $C^B_\varphi(m,n) = 1$ corresponds to a 2×2 block of occupancies $B_{m,n}$ that needs to be encoded in phase φ. Occupancies image O_{z_0} is convolved with a 2×2 kernel to generate the block occupancies image $O^B_{z_0}$, which contains numbers from 0 to 15 that symbolize the occupancy patterns of 2×2 blocks. Block Picker's shape is consistent with both $O^B_{z_0}$ and the output stack $\mathcal{G}_{z_0}(\varphi)$ coming from the CNN. The element $O^B_{z_0}(m,n)$ for which $C^B_\varphi(m,n) = 1$ and the corresponding column $\mathbf{G}(m,n)$ in the output stack $\mathcal{G}_{z_0}(\varphi)$ are picked and form a (symbol, pmf) pair. These pairs are serialized in a certain scanning order, which is also obeyed by the decoder.

We encode the occupancy of candidate voxels in section $z = z_0$ in blocks of 2×2 voxels, such that a block of occupancies at a position m, n is defined as

$$B_{m,n} = \begin{bmatrix} O_{z_0}(2m, 2n) & O_{z_0}(2m, 2n+1) \\ O_{z_0}(2m+1, 2n) & O_{z_0}(2m+1, 2n+1) \end{bmatrix}.$$

We define the occupancy pattern of a block $B_{m,n}$ as $Q_{m,n} = O_{z_0}(2m, 2n) + 2O_{z_0}(2m, 2n+1) + 2^2 O_{z_0}(2m+1, 2n) + 2^3 O_{z_0}(2m+1, 2n+1)$. The entire $W \times H$ image is covered by nonoverlapping blocks by setting $m = 0, 1, \ldots, W/2 - 1$ and $n = 0, 1, \ldots, H/2 - 1$ (W and H are enforced to be even).

For encoding with arithmetic coding the occupancy pattern $Q_{m,n}$ of a generic $(2 \times 2)-$ block of pixels $B_{m,n}$, we utilize a probability mass function (pmf), denoted as a 16-length vector $\mathbf{G}(m,n)$, specifying at element q the probability $G_q(m,n) = \text{Prob}(Q_{m,n} = q|\mathcal{C}_{m,n})$, for each $q \in \{0, \ldots, 15\}$, conditioned on a context $\mathcal{C}_{m,n}$ (see Fig. 6). We implement the probability model by a CNN with the set of parameters $\mathcal{W}$ and 16 output channels, which will output at every location (m,n) the pmf vector denoted

$$\mathbf{G}(m,n) = CNN_{\mathcal{W}}(\mathcal{C}_{m,n}) = \mathbf{g}(\mathcal{W}, \mathcal{C}_{m,n}). \tag{1}$$

Hence, the output of the CNN is an array of size $16 \times W/2 \times H/2$, called the output stack, denoted $\mathcal{G}_{z_0}$. At location (m,n) and channel q, $\mathcal{G}_{z_0}(q,m,n) = G_q(m,n)$.

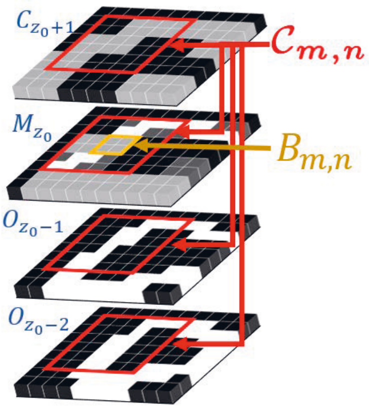


FIGURE 6. The context $\mathcal{C}_{m,n}$ (the 4 × 6x6 block of voxels marked in red) for computing the 16-element probability mass function $Prob(Q_{m,n} = q|\mathcal{C}_{m,n})$ for $q = 0, \ldots, 15$, needed for encoding and decoding the occupancy pattern $Q_{m,n}$ for the 2×2 block $B_{m,n}$ (the block marked by the yellow square). The CNN computes in parallel all these probabilities at all (m,n) locations. At the training stage, for each context observed in the training set, a 16-element histogram $h(q|\mathcal{C}_{m,n})$, $q = 0, \ldots, 15$, is collected and used in the loss function from (5).

The conditioning is done on a context $\mathcal{C}_{m,n}$, which is defined by the receptive field of the CNN, i.e., by the set of

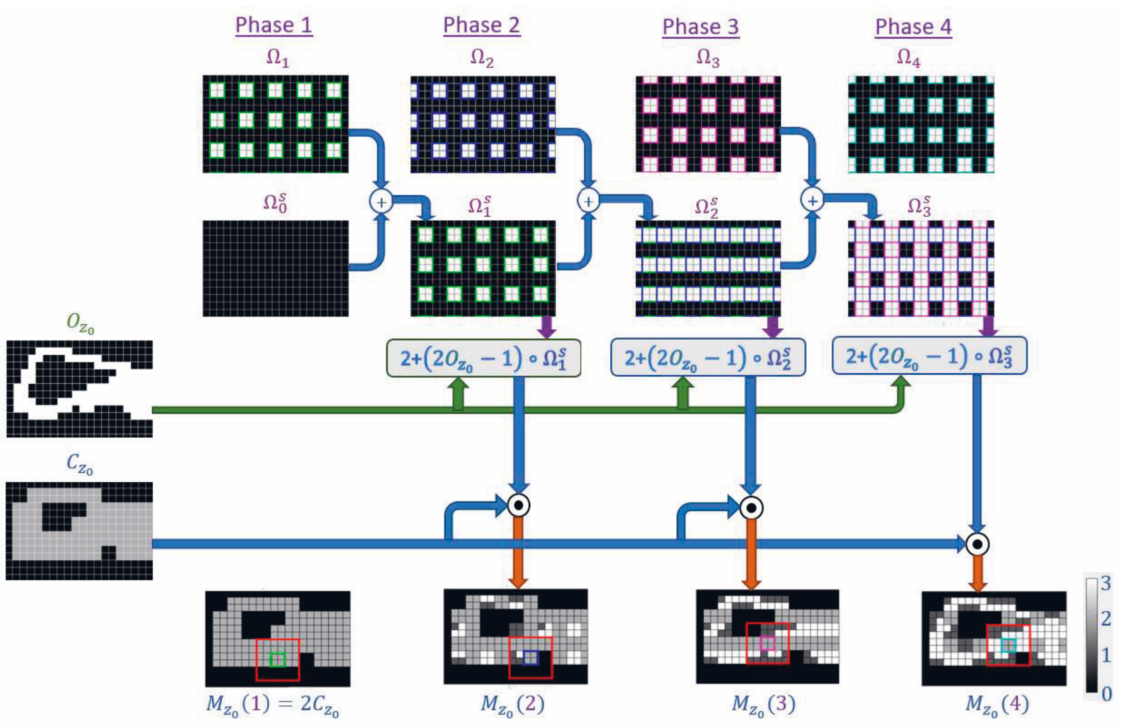


FIGURE 7. Constructing the mixed quaternary image $M_{z_0}(\varphi)$ at each phase φ by combining the available information from the already encoded parts of the binary occupancy image O_{z_0} and the binary candidates image C_{z_0}. The quaternary value for each pixel (i,j) is obtained by setting either $M_{z_0}(i,j) = 2O_{z_0}(i,j) + 1$ or $M_{z_0}(i,j) = 2C_{z_0}(i,j)$ depending on whether $O_{z_0}(i,j)$ is known in the current phase (i.e., $\Omega^s_{\varphi-1}(i,j) = 1$). Each phase is associated with a different phase selector image Ω_φ, which are binary images. The already processed voxels at the beginning of each phase φ are expressed as a binary image $\Omega^s_{\varphi-1}$. The mixed image, $M_{z_0}(\varphi)$, which is to be used in the input stack at phase φ, is obtained by applying $M_{z_0}(\varphi) = C_{z_0} \circ (2 + (2O_{z_0} - 1) \circ \Omega^s_{\varphi-1})$ (3) (which rephrases the mechanism of constructing the quarternary values described above). A pixel in $M_{z_0}(\varphi)$ can have one of the four values 0,1,2,3. If it was not yet encoded, it can be 0 (if it is not a candidate, $C_{z_0}(i,j) = 0$) or 2 (if it is a candidate, $C_{z_0}(i,j) = 1$). If it is already encoded, it can be 1 (if it is not occupied, $O_{z_0}(i,j) = 0$) or 3 (if it is occupied, $O_{z_0}(i,j) = 1$). At each phase, we show in $M_{z_0}(\varphi)$ one example context window as a red bounding box around its corresponding 2×2-block to be encoded.

pixels from the Input Stack S_{z0}, which affects the computation of $\mathbf{G}(m,n)$. For the selected structure of the CNN shown in Fig. 4, the receptive field $\mathcal{C}_{m,n}$ can be seen to be the $4 \times 6 \times 6$ subblock from the input stack S_{z0}, ranging on x coordinates from $2m-2$ to $2m+3$ and on y coordinates from $2n-2$ to $2n+3$. In Fig. 3 and Fig. 6, the context of $4 \times 6 \times 6$ pixels from the images $O_{z0-2}, \ldots, C_{z0+1}$, corresponding to the $2 \times 2-$ candidate block marked in yellow on the mixed image $M_{z0}(\varphi)$, are shown by the red contours.

The CNN consists of four 2D convolutional stages where the convolutions operate along the W and H dimensions and the number of input channels is four (the number of 2D images in $S_{z0}(\varphi)$). As a nonlinear activation function at the hidden stages, we employ LeakyReLU [29] with a constant slope α for the negative inputs so that

$$\text{LeakyReLU}(x) = \max(0,x) + \alpha * \min(0,x). \tag{2}$$

We set the negative region slope as $\alpha = 0.01$ following [29]–[31]. The output layer activation is chosen as softmax to ensure the output $\mathbf{G}(m,n)$ to be valid probability mass functions.

C. ENCODING THE VOXELS IN A 2D SECTION IN FOUR PHASES

To improve the informativeness of the contexts, we split the encoding process into four phases such that in each phase, the CNN is called once, having at the input a different input stack, $S_{z0}(\varphi)$, which gradually becomes more informative at each phase. In all four phases, we employ the same CNN model which is trained with context-histogram pairs collected from all four phases. We do not devote different CNN models to each phase because in such a scenario, the CNN bitstream would be four times longer and the CNN training time would also increase significantly. Moreover, running four different CNNs would significantly increase the consumption of GPU memory during encoding and decoding. Thus, having a single model which handles all the four phases is advantageous in terms of the bitrate, the execution times and the memory

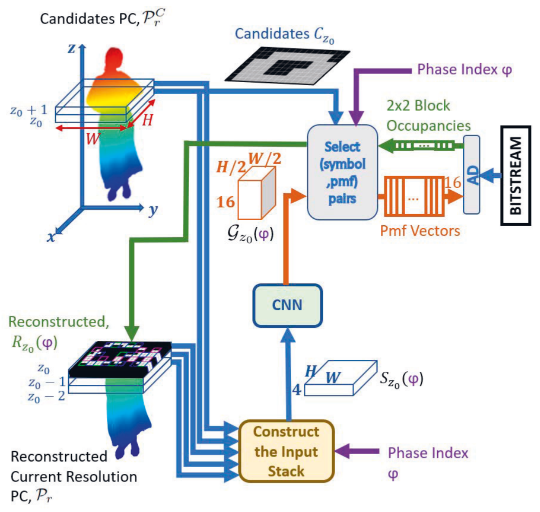

Candidates PC, $\mathcal{P}_r^C$
Candidates C_{z_0}
Phase Index φ
2x2 Block Occupancies
Select (symbol ,pmf) pairs
AD
BITSTREAM
Pmf Vectors
16
$H/2$
$W/2$
16
$\mathcal{G}_{z_0}(\varphi)$
CNN
Reconstructed, $R_{z_0}(\varphi)$
z_0
z_0-1
z_0-2
z_0+1
z_0
W
H
z
y
x
$S_{z_0}(\varphi)$
4
Construct the Input Stack
Phase Index φ
Reconstructed Current Resolution PC, $\mathcal{P}_r$

FIGURE 8. Decoding of the occupancy image O_{z_0} (the section $z = z_0$ through the point cloud $\mathcal{P}_r$ at resolution r) at a phase φ. The block "Select (symbol,pmf) pairs" provides the arithmetic decoder (AD) with the pmf vectors of the blocks to be decoded and it inserts the decoded block occupancies into the reconstructed $R_{z_0}(\varphi)$. In the next phase, $R_{z_0}(\varphi)$ is used to form the $M_{z_0}(\varphi+1)$ in the input stack $S_{z_0}(\varphi+1)$.

consumption. Associated with each phase φ is a different phase selector image Ω_φ that selects the candidate blocks to be encoded in the current phase (the selected candidate blocks are called the phase candidates blocks) (see Fig. 5 and 7).

Phase selector images Ω_φ are visualized in Fig. 7. In each phase φ, a quarter of the 2×2 blocks, all located in a squared grid, are selected for possible encoding by setting $\Omega_\varphi(i,j) = 1$ for every pixel (i,j) belonging to a selected 2×2 block (see the four phase selectors $\Omega_1,\ldots,\Omega_4$ at the top of Fig. 7). The elements of Ω_1 are 1 for all pairs (i,j) with $i = 4k, i = 4k+1, j = 4l, j = 4l+1$, where k,l are integers, so that the elementwise product $O_{z_0}\circ\Omega_1$ ($\circ$ denotes element- wise multiplication) forces to zero all the pixels that do not belong to blocks $B_{2k,2l}$.

Similarly, the selector image Ω_2 selects in $O_{z_0}\circ\Omega_2$ all the blocks $B_{2k,2l+1}$, Ω_3 selects all the blocks $B_{2k+1,2l}$, and Ω_4 selects all the blocks $B_{2k+1,2l+1}$.

In each phase, the CNN uses the Input Stack $S_{z_0}(\varphi)$ and generates pmf vectors $\mathbf{G}(m,n) = CNN_{\mathcal{W}}(\mathcal{C}_{m,n})$ for all blocks $B_{m,n}$, with $m = 0,1,\ldots W/2-1$ and $n = 0,1,\ldots H/2-1$, hence covering all blocks of the $(W\times H)$ images. The candidate image C_{z_0} specifies which of the blocks are already known to be zero and hence do not need to be encoded.

In the first phase, out of all $2\times 2-$ blocks of C_{z_0} and O_{z_0}, only a quarter of the blocks are selected by using Ω_1; namely, from O_{z_0}, only the blocks $B_{2k,2l}$, with $k = 0,\ldots,W/4-1$ and $l = 0,\ldots,H/4-1$. The pmf corresponding to each block is read from the pmf vector $\mathbf{G}(2k,2l)$ and is used by the arithmetic encoder. The encoder and the decoder are now accounting that one quarter of the image O_{z_0} was reconstructed, and those values can be inserted for the next phase into S_{z_0}.

Phases 2, 3 and 4 proceed in a similar way, after which the entire O_{z_0} pixels can be reconstructed so that $R_{z_0} = O_{z_0}$, and the algorithm moves to the processing of the next section, $z = z_0+1$.

Fig. 7 shows how the candidate image C_{z_0} and the occupancy image O_{z_0} (or equivalently the reconstructed image R_{z_0}) are combined differently at each phase, resulting in the mixed $W\times H$-image $M_{z_0}(\varphi)$. Initially, since from O_{z_0} nothing is encoded yet, M_{z_0} contains only the candidacy information. The already processed part of the current section after phase φ is denoted with a binary image $\Omega_\varphi^s = \Omega_1 \vee \ldots \vee \Omega_\varphi$ (elementwise OR), where Ω_0^s is all-zeros. The mixed image $M_{z_0}(\varphi)$ at the beginning of phase φ is obtained at the encoder as

$$M_{z_0}(\varphi) = C_{z_0}\circ(2+(2O_{z_0}-1)\circ\Omega_{\varphi-1}^s), \tag{3}$$

where $\circ$ is pixelwise multiplication, by the process of obtaining the quaternary values out of the two binary values, as described in the caption of Fig. 7. Equivalently, at the decoder, we have

$$M_{z_0}(\varphi) = C_{z_0}\circ(2+(2R_{z_0}(\varphi)-1)\circ\Omega_{\varphi-1}^s), \tag{4}$$

since $O_{z_0}\circ\Omega_{\varphi-1}^s = R_{z_0}(\varphi)\circ\Omega_{\varphi-1}^s$. Eqn. (3) is the mathematical expression for the block "Construct the Mixed Image $M_{z_0}(\varphi)$" in Fig. 3, whereas (4) is the decoder version of the same block.

The "Select and Serialize" block, which appears in Fig. 2, carries out the operations required to feed the occupancy symbols and the corresponding probability mass functions to the arithmetic encoder. "Select and Serialize" operations are schematized in Fig. 5. The two binary images C_{z_0} and Ω_φ are element-wise multiplied to yield the so-called phase candidates C_φ. Occupancies image O_{z_0} is convolved with a 2D kernel having elements [1, 2; 4, 8] to yield the so-called block occupancies $O_{z_0}^B$. $O_{z_0}^B$ is a 16-level image with dimensions $(W/2, H/2)$, and each pixel corresponds to the occupancy pattern of a 2×2 block in O_{z_0}. In phase φ, only those candidate blocks that are indicated in C_φ need to be encoded. C_φ is downsampled to the so-called block picker C_φ^B, which has the same shape $(W/2, H/2)$ as $O_{z_0}^B$. Finally, the "Pick and Serialize" block in Fig. 5 picks the relevant elements from $O_{z_0}^B$ and from the output stack $\mathcal{G}_{z_0}$ and serializes them such that for each block $B_{m,n}$ for which $C_\varphi^B(m,n) = 1$, the 16-element pmf vector $\mathbf{G}(m,n)$ is extracted from the output stack, and the corresponding block occupancy symbol $O_{z_0}^B(m,n)$ is extracted from the block occupancy image. Then, the (symbol,pmf) pairs are sent to the encoder in a predefined scanning order.

Fig. 8 shows the decoder's flow diagram using the same probability modeling and the same CNN as the encoder. The decoder counterpart of the "Select and Serialize" block in the encoder is called the "Select (symbol,pmf) pairs" block. It performs perfectly aligned with the encoder. First, it provides the arithmetic decoder with the pmf of the block to be decoded, and second, it reads from the output of the arithmetic decoder the decoded 2×2 block occupancies and inserts them into the correct locations in $R_{z_0}(\varphi)$. The essential difference from the way the input to the CNN is constructed at the encoder is the replacement of O_{z_0} by R_{z_0}.

D. OPTIMIZATION OF THE CNN

The CNN is trained using the data collected from a number of frames, which we call the training frames. The training set consists of $(4 \times 6 \times 6)$-shaped contexts that have occurred in the training frames at least once. Each context is associated with a 16-element histogram h containing the number of occurrences of 16 possible occupancy patterns of 2×2 candidate blocks. The training data are collected from the final resolution only.

During training, the contexts from the collected set are fed in batches of size N_b, and the loss is expressed as

$$Loss = -\frac{1}{N_b} \sum_{i=1}^{N_b} \sum_{q=0}^{15} h(q|\mathcal{C}_i) \log_2 g_q(\mathcal{W}, \mathcal{C}_i), \tag{5}$$

where $h(q|\mathcal{C}_i)$ is the number of occurrences of the q'th occupancy pattern in the training set for the i'th context, $\mathcal{C}_i$, of the batch and $g_q(\mathcal{W}, \mathcal{C}_i)$ is the corresponding output of the CNN; see (1).

III. EXPERIMENTAL WORK

We have performed experiments with sequences from Microsoft Voxelized Upper Bodies (MVUB) [33] and 8i Voxelized Full Bodies [34] datasets, which are dense point cloud datasets. MVUB sequences have a resolution of 9 bits, whereas 8i sequences have a resolution of 10 bits per dimension.

In our default CNN architecture (shown in Fig. 4), the total number of optimized parameters is 15116. Each parameter is transmitted in 4 bytes, leading to a model codelength of $\text{CL}_m \cong 60.5$ kB. Thus, for a frame with 500k points in a 100-frame sequence, the model's contribution to the bitrate is less than 1%. Due to this small size, we did not consider, in this paper, the problem of entropy coding the CNN parameters, which will not significantly improve the currently achieved bitrates.

In all the experiments, training is performed using the ADAM optimizer with batch size $N_b = 10^4$ and an initial learning rate of 0.001. Since the training time is counted as part of the encoding time, the training phase is kept short. When no improvement in loss is observed for 20 epochs for the first time, the learning rate is halved. When no improvement in loss is observed once again, the training ends. The number of training frames is by default set to 5. To maximize parallelism, the sweeping axis Oz is selected as the shortest dimension of the bounding box, tightly enclosing all the points of the first frame. The algorithm is implemented using PyTorch and TorchAC [35] on an NVIDIA RTX 2080. The implementation is made available on Github.[1]

The bitrate br for the sequence is measured as the average of the bitrates br_f over F consecutive frames, where the bitrate br_f for a frame f is measured as bits-per-point (bpov)

$$\text{br}_f = (\text{CL}_f + \text{CL}_m/F)/n_{p,f}, \tag{6}$$

and CL_f is the codelength for encoding frame f, CL_m is the codelength for encoding the CNN model and $n_{p,f}$ is the number of points in frame f.

In addition to the bitrates, we also report the average encoding and decoding times per frame, where the encoding time per frame t_e includes the time spent collecting training data and training the CNN divided by the number of frames. That is,

$$t_e = \frac{1}{F}(t_{tr} + \sum_{f=1}^{F} t_{e,f}), \tag{7}$$

where t_{tr} is the total time spent training the CNN (including the training data collection time) and $t_{e,f}$ is the time spent encoding frame f. For our default configuration, the average t_{tr} over all 9 sequences used in our experiments is 348 seconds.

In Table 1, we compare the bitrates br obtained with the proposed SeqNOC (default model) and with some recently published algorithms. The methods on the left side of the table have to resort to optimization (as in our method) or adaptively track the count numbers in each context to build efficient coding probability distributions. The right side of Table 1, under the header "Methods using models trained on a generic dataset" contains methods following the GTM paradigm [11], [17], [24], [26]. We note that for the *generically trained models*, there are no results listed for some sequences because some frames of those sequences were part of the training data; therefore, the cited publications did not include results for them. TMC13 [4], S3D [6], P(Full) [18], S4DCS [12] and SeqNOC results are the average bitrates over the 100 frames starting from the 2nd frame, whereas the results for NNOC, fNNOC [17] are for the entire sequence. VoxelDNN [11], MSVoxelDNN [26] and DGM [24] results are for certain representative frames taken from the sequences.

To investigate the effect of model complexity, we experiment with CNN models with different complexities. To that end, we devise three additional CNN model structures having the same number of convolutional layers as our default model, which is illustrated in Fig. 4 but with a different number of output channels at the hidden layers. These models are named SNLM, SNLM2 and SNHM (the first two of them being lighter than the default model and the last one being heavier), and their structures are summarized in Table 2, where n_w is the number of CNN parameters to be transmitted and $n_{hidden,l}$ is the number of output channels at the l'th hidden layer.

[1]https://github.com/marmus12/seqnoc

TABLE 1. Comparing the average bitrates of the proposed SeqNOC with those of the other recent methods.

	Methods using models adapting to the PC or to the sequence to be compressed						Methods using models trained on a generic dataset				
	TMC13 (v12.0) [5]	**S3D** [6]	**P(Full)** [18]	**S4DCS** [12]	**FRL** [32]	**SeqNOC**	**fNNOC** [17]	**NNOC** [17]	**VDNN** [11]	**MSVDNN** [26]	**DGM** [24]
Andrew	1.14	1.12	1.37	0.95	1.01	**0.85**	-	-	-	-	-
David	1.07	1.06	1.31	0.94	0.94	**0.78**	-	-	-	-	-
Phil	1.17	1.14	1.42	1.02	1.02	**0.86**	1.00	0.81	0.92	-	***0.76***
Ricardo	1.09	1.04	1.34	0.90	0.94	**0.79**	0.86	***0.68***	0.72	-	0.69
Sarah	1.07	1.07	1.37	0.92	0.96	**0.80**	-	-	-	-	-
Average$_{\text{MVUB}}$	1.11	1.09	1.36	0.95	0.97	**0.82**	-	-	-	-	-
LongDress	1.02	0.95	1.13	0.88	0.86	**0.70**	-	-	-	-	-
Loot	0.96	0.92	1.02	0.84	0.82	**0.66**	0.74	0.59	0.64	0.73	***0.58***
RedAndBlack	1.08	1.02	1.23	0.94	0.93	**0.77**	0.87	0.72	0.73	0.87	***0.66***
Soldier	1.03	0.96	0.85	**0.65**	0.88	0.70	-	-	-	-	-
Average$_{\text{8i}}$	1.02	0.96	1.06	0.83	0.87	**0.71**	-	-	-	-	-

TABLE 2. Structure of CNN models having different complexities.

	SeqNOC (default)	**SNLM**	**SNLM2**	**SNHM**
$n_{hidden,1}$	30	20	10	60
$n_{hidden,2}$	30	20	10	60
$n_{hidden,3}$	80	40	20	100
n_w	15116	5856	1736	41196

The bitrates and per-frame encoding-decoding times obtained with four different models are presented in Table 3, where the sequence length is $F = 100$. From Table 3, one can observe that the default model yields the best bpov results for the 9-bit MVUB sequences, whereas for the 10-bit 8i sequences, SNHM performs the best in bitrates. This is because, for the 9-bit sequences, the number of points at each point cloud is much less than the 10-bit point clouds so that the average bpov model cost of the heavy model (SNHM) is much more significant than the 10-bit case. A more complex model is more suitable when the number of points is high. On the other hand, the best decoding time is obtained with the lightest model (SNLM2); however, the bitrates of SNLM2 are the worst. Regarding the encoding time, there does not seem to be a clear-cut winner, but on average, SNLM performs the best. This is because the per-frame encoding time t_e is composed of both the time spent training the CNN t_{tr} (divided by the number of frames F) and the time spent encoding one frame. The time spent encoding one frame is shorter for the lighter models, whereas the CNN training time can sometimes be shorter when the model is more complex. Note that for all the experimental models, we employ the same training policy, which decides to end the training when the training loss stops to improve.

In Table 4, we compare the results obtained with different numbers of training frames. In our default configuration, the number of training frames is set to 5. Since the number of training frames does not affect the decoding times, we only present the encoding times in Table 4. According to Table 4, the per-frame encoding time t_e increases dramatically with the increasing number of training frames since the CNN training takes longer and the best encoding times are obtained in the single training frame case. For MVUB, the best bitrates are obtained with 20 training frames, whereas for the 8i dataset, the best bitrates are obtained with 10 training frames, yet the bitrate differences between the 10 and 20 training frame scenarios are rather small.

In Table 5, the results obtained with the default SeqNOC are compared with the single phase version of SeqNOC, which is called SeqNOC-SP. In SeqNOC-SP, we do not employ the four-phase strategy; instead, there is a single phase in which the probability of occupancies for a section $z = z_0$ is estimated. The SeqNOC-SP operates on less informative contexts, yet it has the advantage of speed. We perform this comparison mainly to show the improvement brought by the usage of phases.

Since the algorithms run at different speeds on different hardware, we compare the speed performance of different algorithms by comparing the ratios between the runtime of an algorithm and the runtime of TMC13 (reported in the publication describing the algorithm). The ratios between the average (per-frame) encoding/decoding times of the different methods and the average (per-frame) encoding/decoding times of TMC13 for PCs from MVUB and the 8i sequences are presented in Table 6. The encoding time ratio for FRL is taken from [32]. The runtimes for DGM, NNOC, fNNOC, MSVDNN, VoxelDNN and SeqNOC were measured on an NVIDIA RTX 2080, and the runtimes for TMC13 were measured as 2.9 seconds for encoding and 2.8 seconds for decoding on an Intel(R) Xeon(R) Silver 4110 CPU @ 2.10 GHz [24]. From Table 6, it is evident that FRL [32] is the fastest method, whereas SeqNOC-SP and SeqNOC are the third and fourth fastest, respectively. We note that the remaining methods are significantly slower than SeqNOC, and out of the NN-based methods, only the SeqNOC and SeqNOC-SP encoding times include the time spent for CNN training.

In Figure 9, we plot the average per-frame bitrate of several methods vs. the encoding time ratios to the TMC13 encoding time. Such 2D visualization allows us to roughly demonstrate the trade-off between bitrates and runtimes for different algorithms. In Fig. 9, the convex hull formed by

TABLE 3. Bitrates [bpov], encoding times (t_e[s]) and decoding times (t_d[s]) obtained with four different CNN models.

	SeqNOC (default)			**SNLM**			**SNLM2**			**SNHM**		
	bpov	t_e[s]	t_d[s]	bpov	t_e[s]	t_d[s]	bpov	t_e[s]	t_d[s]	bpov	t_e[s]	t_d[s]
Andrew	**0.85**	8.4	7.0	0.87	7.3	6.8	0.94	7.7	**6.6**	0.88	**7.1**	7.2
David	**0.78**	10.0	9.0	0.79	10.2	**8.5**	0.84	10.2	**8.5**	0.81	**9.3**	9.1
Phil	**0.86**	10.0	8.0	0.88	**9.1**	7.9	0.93	11.1	**7.3**	0.89	10.8	8.4
Ricardo	**0.79**	9.2	6.5	**0.79**	8.1	6.5	0.88	**5.8**	**6.2**	0.85	8.0	7.0
Sarah	**0.80**	9.2	8.5	0.82	**8.2**	7.9	0.87	10.0	**7.6**	0.85	9.1	8.2
Average$_{\text{MVUB}}$	**0.82**	9.4	7.8	0.83	**8.6**	7.5	0.89	9.0	**7.2**	0.86	8.9	8.0
Longdress	0.70	10.9	9.0	0.73	8.7	8.3	0.87	**7.9**	**8.1**	**0.68**	13.5	10.9
Loot	0.66	10.0	9.0	0.68	7.9	8.1	0.78	**7.4**	**7.9**	**0.65**	18.7	10.6
Redandblack	0.77	9.8	8.5	0.79	10.2	8.0	0.88	**8.7**	**7.5**	**0.76**	14.2	10.2
Soldier	0.70	12.7	10.0	0.74	**10.2**	9.3	0.8	11.8	**9.0**	**0.68**	15.8	12.1
Average$_{\text{8i}}$	0.71	10.8	9.1	0.73	9.3	8.4	0.83	**9.0**	**8.1**	**0.69**	15.6	11.0

TABLE 4. Bitrates [bpov] and encoding times (t_e[s]) for SeqNOC obtained with different numbers of training frames.

Number of Training Frames:	**1**		**5 (default)**		**10**		**20**	
	bpov	t_e[s]	bpov	t_e[s]	bpov	t_e[s]	bpov	t_e[s]
Andrew	0.91	**5.9**	0.85	8.4	**0.83**	9.5	**0.83**	10.0
David	0.84	**7.8**	0.78	10.0	**0.77**	11.9	**0.77**	11.2
Phil	0.93	**7.1**	0.86	10.0	0.84	12.5	**0.83**	21.0
Ricardo	0.86	**5.5**	0.79	9.2	0.77	11.8	**0.76**	16.5
Sarah	0.89	**6.5**	0.80	9.2	0.78	10.9	**0.77**	22.5
Average$_{\text{MVUB}}$	0.89	**6.6**	0.82	9.4	0.80	11.3	**0.79**	16.2
Longdress	0.72	**8.4**	0.70	10.9	**0.69**	14.2	0.70	16.7
Loot	0.67	**8.3**	0.66	10.0	**0.64**	16.4	0.65	17.5
Redandblack	0.79	**7.8**	0.77	9.8	**0.75**	16.0	0.76	16.9
Soldier	0.71	**11.3**	**0.70**	12.7	**0.70**	13.4	0.72	13.6
Average$_{\text{8i}}$	0.72	**9.0**	0.71	10.9	**0.70**	15.0	0.71	16.2

TABLE 5. Bitrates [bpov], encoding times (t_e[s]) and decoding times (t_d[s]) obtained with the default (4-phase) model vs. the single phase model.

	SeqNOC (default)			**SeqNOC-SP**		
	bpov	t_e[s]	t_d[s]	bpov	t_e[s]	t_d[s]
Andrew	**0.85**	8.4	7.0	0.90	**4.9**	**2.4**
David	**0.78**	10.0	9.0	**0.78**	**6.4**	**2.6**
Phil	**0.86**	10.0	8.0	0.88	**6.6**	**2.2**
Ricardo	**0.79**	9.2	6.5	0.80	**4.6**	**2.0**
Sarah	0.80	9.2	8.5	**0.79**	**5.9**	**2.2**
Average$_{\text{MVUB}}$	**0.82**	9.4	7.8	0.83	**5.7**	**2.3**
Longdress	**0.70**	10.9	9.0	0.75	**6.5**	**2.7**
Loot	**0.66**	10.0	9.0	0.70	**6.9**	**2.6**
Redandblack	**0.77**	9.8	8.5	0.81	**7.6**	**2.5**
Soldier	**0.70**	12.7	10.0	0.77	**6.4**	**3.0**
Average$_{\text{8i}}$	**0.71**	10.8	9.1	0.76	**6.9**	**2.7**

the datapoints corresponding to various algorithms is also drawn as a blue dashed line. The bitrates are averaged over 4 sequences, namely, phil, ricardo, loot and redandblack, for which the bitrates were available for all of the plotted methods. Similarly, in Fig. 10 are plotted the bitrates vs ratios of decoding times. From Figures 9 and 10, it is evident that both the default SeqNOC and its single-phase version SeqNOC-SP provide good trade-offs between encoding/decoding times and bitrates.

A. COMPRESSING A SINGLE POINT CLOUD WITH SeqNOC

Although the SeqNOC scheme is intended for the compression of sequences, it can also be used to compress a single point cloud, where the CNN optimization is performed with the context-histogram pairs extracted from the single point cloud. In the single point cloud case, the cost of transmitting the CNN parameters becomes quite significant; therefore, we choose to employ a CNN model with a small number of parameters. The model we use for the single point cloud case is SNLM2. The encoding time for a single PC is on the order of minutes since the time spent for optimization (t_{tr}) is on the order of minutes, whereas the decoding time is still on the order of seconds for a single point cloud. This property makes encoding of single frames by the STM strategy well suited for ‘‘encoding once decoding many times’’ scenarios (i.e., for broadcasting).

The bitrates obtained for the point clouds from Cat1A [36] are presented in Table 7. The point clouds with resolutions higher than 10 bits are downsampled to 10 bits. The bitrates reported in the SNLM2 column include the model cost (the cost of transmitting the model). Since the model costs for the single point cloud case are much more significant than those for the sequence case, the model costs for each PC are also shown separately in the Model Cost column. Note that for all of the PCs, the model structure and thus the model codelength $CL_m = 32 \times n_w$ are exactly the same, and the model cost in bpv (CL_m/n_p, where n_p is the number of points) varies due to the differences in the number of points in each PC. From Table 7, one can see that SNLM2 outperforms TMC13 for all the point clouds except three of them. Furthermore, SNLM2 offers significantly better bitrates on average when

TABLE 6. Ratios between the average (per-frame) runtime of different methods and the average (per-frame) runtime of TMC13.

	FRL [32]	SeqNOC-SP	SeqNOC (default)	fNNOC [17]	NNOC [17]	VoxelDNN [11]	MSVDNN [26]	DGM [24]
ENC	0.7	2.2	3.5	24.2	65.9	658.3	13.9	305.2
DEC	n.a.	2.5	3.0	27.5	418.2	2075.8	18.7	228.6

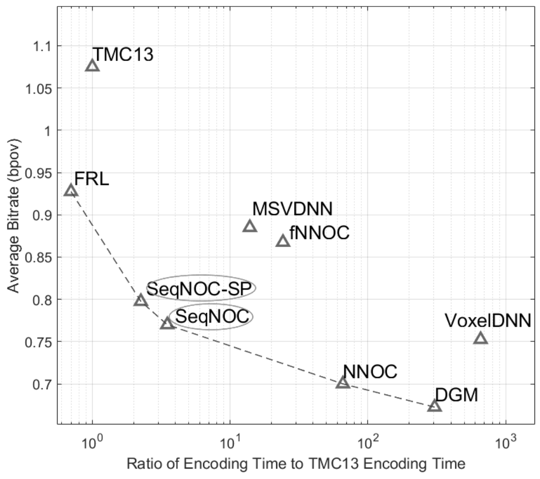


FIGURE 9. Average bitrate vs. encoding time ratio to TMC13 encoding time.

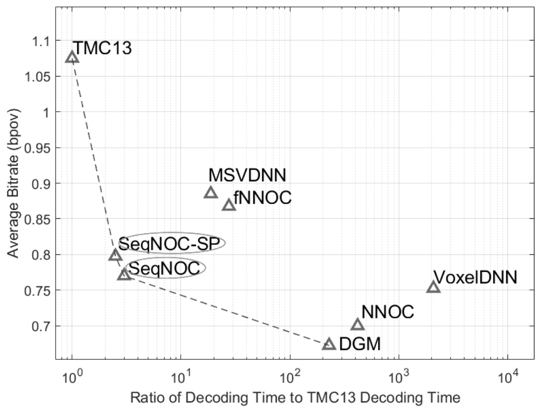


FIGURE 10. Average bitrate vs. decoding time ratio to TMC13 decoding time.

compared to TMC13. On the other hand, DGM [24] performs significantly better on shiva and frog.

IV. DISCUSSION

Our proposed scheme SeqNOC is a follow-up version of the schemes NNOC and fNNOC proposed in [17], with many architectural changes made with the main goal of achieving a reasonable per-frame *decoding time*. Additionally, from the encoder point of view, the structure of the neural network was changed and chosen to obtain a reasonable time for the optimal design of the CNN model based on several frames of the sequence. In a nutshell, we changed the schemes from [17] to allow a specific design of the CNN for each sequence with a *faster encoding time* (including even the time for optimizing a specific CNN for each sequence) and a faster decoding time, retaining almost as good lossless coding performance as in [17]. The similarities and differences are reviewed below in detail.

First, we discuss the similarities. The encoding of the occupancies of the voxels is performed in a multi-resolution fashion, sequentially encoding the octree representation at increasing resolutions. As with any context-based compression scheme, we tried to ensure the most informative contexts around the voxels to be encoded, using the voxels at the previous resolution (all of them being already encoded) and the voxels that are already encoded from the current resolution. In all SeqNOC and NNOC versions, we scan the voxels at a current resolution plane-by-plane (with each plane perpendicular on a chosen coordinate axis). Looking at the $4 \times L \times L$ ($L = 5$ for NNOC) neighbors of the voxel to be encoded in the current resolution, we see in the planes below it, $z = z_0 - 2$ and $z = z_0 - 1$, voxels for which the occupancy status is known, providing good contextual information. In the plane above, at $z = z_0 + 1$, the current resolution status is not known, but the previous resolution level pixels are already encoded, so we can create the candidate voxels at the current resolution (those that resulted from the split of an occupied voxel at the previous resolution). This is still useful contextual information but less useful than the information at $z = z_0 - 2$ and $z = z_0 - 1$.

In the current plane, $z = z_0$, one can use the voxels from the current resolution that have been already encoded (i.e., $\lfloor (L \times L - 1)/2 \rfloor$ voxels), as is done in NNOC, which results in a good coding performance, but this implies that the computation of the occupancy probabilities must be done at the decoder, one-by-one sequentially, for all the voxels in a plane $z = z_0$, resulting in very slow decoding. The fNNOC did not use all the voxels that have been already encoded from the current resolution at $z = z_0$, instead using only the candidate status at $z = z_0$, which resulted in faster decoding than NNOC, but with a lower compression performance. In SeqNOC, we use in a similar way the occupancy status of the neighbor voxels at $z = z_0 - 2$ and $z = z_0 - 1$ and the candidacy status of the voxels at $z = z_0 + 1$; however, the main change is in forming the context from the neighboring voxels at $z = z_0$.

TABLE 7. Bitrates [bpov] for TMC13, DGM, and for SNML2 (the lighter complexity version of SeqNOC, see Table 2) for the point clouds from Cat1A [36] in 10 bits resolution. The bitrate costs of transmitting the CNN parameters are included in the SNLM2 column but are also shown separately in the model cost column.

	TMC13	DGM [24]	SNLM2	Model Cost
shiva	3.70	**3.46**	3.58	0.06
loot	0.98	-	**0.81**	0.07
redandblack	1.11	-	**0.99**	0.07
longdress	1.03	-	**0.85**	0.06
queen	0.78	-	**0.70**	0.06
soldier	1.04	-	**0.85**	0.05
basketball player	0.87	-	**0.64**	0.07
head	1.54	-	**1.39**	0.02
boxer	0.96	-	**0.80**	0.06
facade9	2.41	-	**2.20**	0.06
facade15	1.58	-	**1.42**	0.07
facade64	1.03	-	**0.88**	0.05
dancer	0.86	-	**0.69**	0.08
frog	1.91	**1.70**	1.82	0.04
house without roof	**1.81**	-	2.00	0.03
thaidancer	**1.01**	-	1.02	0.20
arco valentino	**4.85**	4.99	4.86	0.04
Average	1.62	-	**1.50**	0.06

Some of the differences are listed next. To make a scheme even faster than fNNOC at the decoder and to improve the compression performance, in SeqNOC, we made two changes. First, we defined the element to be encoded at each arithmetic coding operation to be a block of 2×2 voxels (leading to a context of $4 \times 6 \times 6$ voxels around the 2×2 voxel block), instead of a single voxel in NNOC and fNNOC (for which the context had $4 \times 5 \times 5$ voxels). The prediction capabilities of the enhanced contexts in SeqNOC were shown in the ablation study to be better, not being affected too much by the dilution effect of the larger contexts. Another consequence of having a different symbol definition is the change in the shape of the context window.

The second change was to use a convolutional neural network instead of the multilayer neural network, which allowed us to compute all the probabilities at all the voxels very fast in parallel from a plane $z = z_0$. In NNOC, a two-layer MLP was employed, whereas in SeqNOC, a four-layer CNN, having fewer parameters than the MLP of NNOC, was employed. The CNN formulation enables faster execution, which helps to reduce the encoding/decoding times significantly.

Additionally, to further improve the compression performance, the pass of the encoding through plane $z = z_0$ was divided into four phases so that in each phase, the computation of the encoding probabilities can be performed in parallel but the contexts from the available voxels in $z = z_0$ are more informative, including the voxels $z = z_0$ already encoded in the previous phases.

The training stage is different. First, NNOC is a generically trained model, and therefore, the results obtained with NNOC are dependent on the training data. If the test data are not similar to the training data, the bitrate performance may deteriorate. In SeqNOC, the entropy model adapts to the input sequence itself. Since the symbols are defined in a different way than it was in NNOC, the loss function is also formulated in a different way.

In NNOC, during decoding, NN was executed for each candidate location in the current section; hence, the number of phases was as high as the number of candidates in the current section, resulting in very slow decoding. The four-phase approach in SeqNOC provides a good balance between bitrates and decoding speed.

Compared to the current state-of-the-art, our method is distinguished by two main features. First, the selection of the encoding unit (a 2×2 block of voxels) and its context (in the neighborhood of the block, utilizing the already encoded voxels at both the current resolution and the previous resolution of the octree), which allows encoding and decoding with rich contextual information in a parallel fashion along the plane-by-plane scanning of the point cloud at each resolution, which reaches competitive encoding and decoding speeds. Second, the introduction of lightweight CNN models, which can be trained quickly at the encoder and can be attached as a header to the bitstream for the full sequence without greatly affecting the overall bitrate, ensuring a very specific CNN model for the task at hand and alleviating the question of whether a generically trained CNN is suitable for the sequence at hand.

V. CONCLUSION

We have introduced a lossless geometry encoding scheme for sequences of dense point clouds using CNN models that are designed in a new paradigm, named *specifically trained models*, which has not been used until now. The learning of the CNN model can be done fast enough at the encoder, so that the learning plus sequence encoding time divided by the number of frames, gives very competitive per frame encoding times, at a bitrate performance better than that obtained in the competing paradigm of *generically trained models*. The coding probability models at each 2×2 block are computed in parallel by the convolutional neural network at each section $z = z_0$ through the point cloud, contributing to a fast encoding and decoding performance. To improve the decoding time, which has unreasonably large values for the published solutions [11], [17], [26], we adopt a four-phase encoding at each section, such that the content of the contexts improves from one phase to another, including the recently encoded/decoded voxels inside the context after each phase. Several variations of the method were proposed that covered various interesting trade-offs (e.g. compression ratio vs. time complexity). We discussed the performance of the proposed solution compared to the recently published schemes and found that the introduced features produce significant improvements.

REFERENCES

[1] S. Schwarz, M. Preda, V. Baroncini, M. Budagavi, P. Cesar, P. A. Chou, R. A. Cohen, M. Krivokuća, S. Lasserre, and Z. Li, "Emerging MPEG standards for point cloud compression," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 1, pp. 133–148, Mar. 2018.

[2] T. Ebrahimi, S. Foessel, F. Pereira, and P. Schelkens, "JPEG Pleno: Toward an efficient representation of visual reality," *IEEE Multimedia*, vol. 23, no. 4, pp. 14–20, Oct./Dec. 2016.

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

[4] D. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, and A. Tabatabai, "An overview of ongoing point cloud compression standardization activities: Video-based (V-PCC) and geometry-based (G-PCC)," *APSIPA Trans. Signal Inf. Process.*, vol. 9, no. 1, pp. 1–15, 2020.

[5] Moving Picture Experts Group. *TMC13*. Accessed: Mar. 20, 2020. [Online]. Available: https://github.com/MPEGGroup/mpeg-pcc-tmc13

[6] E. Peixoto, "Intra-frame compression of point cloud geometry using dyadic decomposition," *IEEE Signal Process Lett.*, vol. 27, pp. 246–250, 2020.

[7] E. Peixoto, E. Medeiros, and E. Ramalho, "Silhouette 4D: An inter-frame lossless geometry coder of dynamic voxelized point clouds," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2020, pp. 2691–2695.

[8] J. Wang, D. Ding, Z. Li, and Z. Ma, "Multiscale point cloud geometry compression," in *Proc. Data Compress. Conf. (DCC)*, Mar. 2021, pp. 73–82.

[9] 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 Trans. Broadcast.*, vol. 66, no. 3, pp. 701–717, Sep. 2020.

[10] S. Milani, E. Polo, and S. Limuti, "A transform coding strategy for dynamic point clouds," *IEEE Trans. Image Process.*, vol. 29, pp. 8213–8225, 2020.

[11] D. T. Nguyen, M. Quach, G. Valenzise, and P. Duhamel, "Learning-based lossless compression of 3D point cloud geometry," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Jun. 2021, pp. 4220–4224.

[12] E. Ramalho, E. Peixoto, and E. Medeiros, "Silhouette 4D with context selection: Lossless geometry compression of dynamic point clouds," *IEEE Signal Process. Lett.*, vol. 28, pp. 1660–1664, 2021.

[13] E. C. Kaya, S. Schwarz, and I. Tabus, "Refining the bounding volumes for lossless compression of voxelized point clouds geometry," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2021, pp. 3408–3412.

[14] M. Quach, G. Valenzise, and F. Dufaux, "Learning convolutional transforms for lossy point cloud geometry compression," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2019, pp. 4320–4324.

[15] D. C. Garcia and R. L. de Queiroz, "Context-based octree coding for point-cloud video," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2017, pp. 1412–1416.

[16] L. Huang, S. Wang, K. Wong, J. Liu, and R. Urtasun, "OctSqueeze: Octree-structured entropy model for LiDAR compression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 1310–1320.

[17] E. C. Kaya and I. Tabus, "Neural network modeling of probabilities for coding the octree representation of point clouds," in *Proc. IEEE 23rd Int. Workshop Multimedia Signal Process. (MMSP)*, Oct. 2021, pp. 1–6.

[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 Trans. Image Process.*, vol. 29, pp. 313–322, 2019.

[19] R. L. de Queiroz, D. C. Garcia, P. A. Chou, and D. A. Florencio, "Distance-based probability model for octree coding," *IEEE Signal Process. Lett.*, vol. 25, no. 6, pp. 739–742, Jun. 2018.

[20] Z. Que, G. Lu, and D. Xu, "VoxelContext-Net: An octree based framework for point cloud compression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 6042–6051.

[21] J. Wang, H. Zhu, H. Liu, and Z. Ma, "Lossy point cloud geometry compression via end-to-end learning," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 12, pp. 4909–4923, Dec. 2021.

[22] X. Wen, X. Wang, J. Hou, L. Ma, Y. Zhou, and J. Jiang, "Lossy geometry compression of 3D point cloud data via an adaptive octree-guided network," in *Proc. IEEE Int. Conf. Multimedia Expo (ICME)*, Jul. 2020, pp. 1–6.

[23] D. E. O. Tzamarias, K. Chow, I. Blanes, and J. Serra-Sagristà, "Compression of point cloud geometry through a single projection," in *Proc. Data Compress. Conf. (DCC)*, Mar. 2021, pp. 63–72.

[24] D. T. Nguyen, M. Quach, G. Valenzise, and P. Duhamel, "Lossless coding of point cloud geometry using a deep generative model," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 12, pp. 4617–4629, Dec. 2021.

[25] M. Quach, J. Pang, D. Tian, G. Valenzise, and F. Dufaux, "Survey on deep learning-based point cloud compression," *Frontiers Signal Process.*, vol. 2, pp. 1–15, Feb. 2022.

[26] D. T. Nguyen, M. Quach, G. Valenzise, and P. Duhamel, "Multiscale deep context modeling for lossless point cloud geometry compression," in *Proc. IEEE Int. Conf. Multimedia Expo Workshops (ICMEW)*, Jul. 2021, pp. 1–6.

[27] A. F. R. Guarda, N. M. M. Rodrigues, and F. Pereira, "Adaptive deep learning-based point cloud geometry coding," *IEEE J. Sel. Topics Signal Process.*, vol. 15, no. 2, pp. 415–430, Feb. 2021.

[28] D. Meagher, "Geometric modeling using octree encoding," *Comput. Graph. Image Process.*, vol. 19, no. 2, pp. 129–147, Jun. 1982.

[29] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, 2013, vol. 30, p. 3.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 1026–1034.

[31] M. Alzantot, Z. Wang, and M. B. Srivastava, "Deep residual neural networks for audio spoofing detection," 2019, *arXiv:1907.00501*.

[32] D. E. O. Tzamarias, K. Chow, I. Blanes, and J. Serra-Sagrista, "Fast run-length compression of point cloud geometry," *IEEE Trans. Image Process.*, vol. 31, pp. 4490–4501, 2022.

[33] C. Loop, Q. Cai, S. O. Escolano, and P. A. Chou, *Microsoft Voxelized Upper Bodies—A Voxelized Point Cloud Dataset*, document ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) Input document m38673/M72012, 2016.

[34] E. d'Eon, B. Harrison, T. Myers, and P. A. Chou, *8I Voxelized Full Bodies—A Voxelized Point Cloud Dataset*, document ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) Input document WG11M40059/WG1M74006, 2017.

[35] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. Van Gool, "Practical full resolution learned lossless image compression," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 10629–10636.

[36] S. Schwarz, G. Martin-Cocher, D. Flynn, and M. Budagavi, *Common Test Conditions for Point Cloud Compression*, document ISO/IEC JTC1/SC29/WG11 w17766, Ljubljana, Slovenia, 2018.

[37] R. Mekuria, K. Blom, and P. Cesar, "Design, implementation, and evaluation of a point cloud codec for tele-immersive video," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 27, no. 4, pp. 828–842, Apr. 2017.

[38] C. Cao, M. Preda, V. Zakharchenko, E. S. Jang, and T. Zaharia, "Compression of sparse and dense dynamic point clouds—Methods and standards," *Proc. IEEE*, vol. 109, no. 9, pp. 1537–1558, Sep. 2021.

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

[40] D. C. Garcia and R. L. de Queiroz, "Intra-frame context-based octree coding for point-cloud geometry," in *Proc. 25th IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2018, pp. 1807–1811.

[41] S. Milani, "Fast point cloud compression via reversible cellular automata block transform," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2017, pp. 4013–4017.

[42] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach, "Real-time compression of point cloud streams," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2012, pp. 778–785.

[43] D. Lazzarotto, E. Alexiou, and T. Ebrahimi, "On block prediction for learning-based point cloud compression," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2021, pp. 3378–3382.

[44] T. Huang and Y. Liu, "3D point cloud geometry compression on deep learning," in *Proc. 27th ACM Int. Conf. Multimedia*, Oct. 2019, pp. 890–898.

[45] L. Gao, T. Fan, J. Wan, Y. Xu, J. Sun, and Z. Ma, "Point cloud geometry compression via neural graph sampling," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2021, pp. 3373–3377.

[46] H. Roodaki, M. Dehyadegari, and M. N. Bojnordi, "G-Arrays: Geometric arrays for efficient point cloud processing," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Jun. 2021, pp. 1925–1929.

[47] Y. Xu, W. Zhu, Y. Xu, and Z. Li, "Dynamic point cloud geometry compression via patch-wise polynomial fitting," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2019, pp. 2287–2291.

EMRE C. KAYA was born in Edirne, Türkiye, in 1990. He received the B.S. and M.S. degrees from the Department of Electrical and Electronics Engineering, Middle East Technical University, Ankara, Türkiye, in 2015 and 2018, respectively. He is currently pursuing the Ph.D. degree with the Computing Sciences Unit, Tampere University, under the supervision of Prof. I. Tabus. His research interests include point cloud compression, image compression, and visual object detection.

IOAN TABUS (Senior Member, IEEE) received the Ph.D. degree (Hons.) from the Tampere University of Technology, Finland, in 1995.

He held teaching positions with the Department of Control and Computers, Politehnica University of Bucharest, from 1984 to 1995. Since 1996, he has been a Senior Researcher, and since January 2000, he has been a Professor with the Department of Signal Processing, Tampere University of Technology, which was merged into Tampere University, in 2019. He is the coauthor of two books and more than 250 publications in the fields of signal compression, image processing, bioinformatics, and system identification. His research interests include light field image processing, plenoptic image compression, point cloud compression, audio, image, data compression, genomic signal processing, and statistical signal processing. He was a co-recipient of the 1991 Train Vuia Award of Romania, the 2001 NSIP Best Paper Award, the 2004 NORSIG Best Paper Award, the 2016 3DTV Best Paper Award, and the ICIP 2017 Light Field Image Coding Challenge Award. He is an Associate Editor of the IEEE TRANSACTIONS ON IMAGE PROCESSING. He served as an Associate Editor for the IEEE TRANSACTIONS ON SIGNAL PROCESSING and *EURASIP Journal on Advances in Signal Processing*. He has served as a Guest Editor for special issues for the *IEEE Signal Processing Magazine*, *EURASIP Journal on Advances in Signal Processing*, and the IEEE JOURNAL OF SELECTED TOPICS IN SIGNAL PROCESSING. He was the Editor-in-Chief of the *EURASIP Journal on Bioinformatics and Systems Biology*, from 2006 to 2014.