TESE DE DOUTORADO

Adaptive Embedded Coding of 3D Point Clouds Using Hierarchical Transforms and Context Modeling

Victor Fabre Figueiredo

Brasília, 27 de novembro de 2025

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

TESE DE DOUTORADO

Adaptive Embedded Coding of 3D Point Clouds Using Hierarchical Transforms and Context Modeling

Victor Fabre Figueiredo

Tese de Doutorado submetida ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Doutor em Engenharia Elétrica

Banca Examinadora

Prof. Ricardo Lopes de Queiroz, Ph.D., PPGEE / UnB
Orientador

Prof. Eduardo Peixoto Fernandes da Silva, Ph.D., PPGEE / UnB
Examinador Interno

Prof. Eduardo Antônio Barros da Silva, Ph.D., UFRJ
Examinador Externo

Prof. Bruno Zatt, Ph.D., UFPel
Examinador Externo

Prof. Camilo Chang Dorea, Ph.D., CIC / UnB
Examinador Suplente

FICHA CATALOGRÁFICA

FIGUEIREDO, VICTOR FABRE
Adaptive Embedded Coding of 3D Point CloudsUsing Hierarchical Transforms and Context Modeling [Distrito Federal] 2025.
xvi, 86 p., 210 x 297 mm (ENE/FT/UnB, Doutor, Engenharia Elétrica, 2025).
Tese de Doutorado - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. point cloud	2. embedded coding
3. region of interest	4. context modeling
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

FIGUEIREDO, V.F. (2025). *Adaptive Embedded Coding of 3D Point CloudsUsing Hierarchical Transforms and Context Modeling*. Tese de Doutorado, Publicação: PPGEE 215/25, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 86 p.

CESSÃO DE DIREITOS

Victor Fabre Figueiredo
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

To my parents and my fiancée,
Alice, Ricardo and Victoria.

ACKNOWLEDGMENTS

Throughout the writing of this work, I have received a great deal of support and assistance.

I would first like to thank my parents, my brothers and my family for their unconditional support. Without them, I could not have completed this thesis.

In addition, I would like to thank my advisor, Professor Ricardo de Queiroz, whose expertise was invaluable in formulating the research questions and methodology. Also, I would like to thank Dr. Philip A. Chou, who co-advised most of the development of this work. Their insightful feedback pushed me to sharpen my thinking and brought my work to a higher level.

I would like to acknowledge my colleagues at the DIVP group for their collaboration. Particularly, André, Prof. Camilo, Dr. Davi, Prof. Diogo, Dr. Lucas and Dr. Tomás. I would particularly like to thank the University of Brasília (UnB) for providing me with the necessary tools for the development of this work.

Finally, I would like to acknowledge the support of my fiancée, Victoria, who cheered, celebrated, believed and supported me at every step of this work. Not a single page would have been written without her pushing me.

Victor Fabre Figueiredo

Work partially supported by Coordenação de Aperfeiçoamento de Pessoal Superior - Brasil (CAPES) - Finance Code 001

ABSTRACT

Three-dimensional (3D) point clouds have become an essential data representation in virtual, augmented, and mixed reality applications, providing highly detailed and interactive visual experiences. The substantial volume of data required for 3D scenes raises critical challenges for transmission, storage, and real-time rendering. These issues have spurred research into efficient compression strategies that maintain as much fidelity as possible while reducing overall bit-rates.

This thesis addresses these challenges by introducing an adaptive embedded coding framework for 3D point cloud attributes. Central to our approach is the use of hierarchical transforms — specifically the Region-Adaptive Hierarchical Transform (RAHT) — together with Set Partitioning in Hierarchical Trees (SPIHT) to produce a single, progressive bit-stream. By encoding transform coefficients in successive bit-planes, the proposed method allows the encoded data to be truncated at any point, enabling a flexible balance between compression efficiency and reconstruction quality.

To further refine the coding process, we explore multiple enhancements. We incorporate context modeling through classical set partitioning trees and neural networks, such as Multi-Layer Perceptrons (MLPs), which predict the conditional probabilities of coefficients for improved arithmetic coding. We also investigate region-of-interest (ROI) coding to allocate higher fidelity to salient areas of the point cloud. In addition, we propose a preliminary extension of neural-based context modeling in arithmetic coding algorithms for lossless image compression, inspired by recent advances in variational image compression, to highlight its potential applicability to lossless (and eventually lossy) point cloud attribute compression.

Experimental results demonstrate that the proposed framework achieves a competitive rate-distortion performance while offering fully embedded functionality. Moreover, the modular nature of the approach — combining RAHT, SPIHT, neural network-based context modeling, and ROI strategies — makes it readily adaptable to different point cloud formats and use cases. Taken together, these contributions underscore the viability of adaptive embedded coding as a robust and flexible solution to the challenges posed by large-scale 3D point cloud data.

RESUMO

Título: Codificação Progressiva de Nuvens de Pontos 3D Usando Transformadas Hierárquicas e Modelagem de Contexto.

Nuvens de pontos tridimensionais (3D) tornaram-se uma representação fundamental em aplicações de realidade virtual, aumentada e mista, proporcionando experiências visuais altamente detalhadas e interativas. O grande volume de dados necessário para descrever cenas 3D levanta desafios significativos em termos de transmissão, armazenamento e renderização em tempo real. Esses desafios motivam o desenvolvimento de estratégias de compressão mais eficientes, que preservem a maior fidelidade possível ao mesmo tempo em que reduzem a taxa de bits.

Este trabalho aborda tais desafios ao propor um esquema progressivo de codificação *embedded* para os atributos de nuvens de pontos 3D. O cerne da abordagem envolve o uso de transformadas hierárquicas — particularmente a Transformada Hierárquica Adaptativa à Região (RAHT) — combinadas ao *Set Partitioning in Hierarchical Trees* (SPIHT), resultando em um *bit-stream* progressivo único. Ao codificar os coeficientes de transformação em diferentes planos de bits, o método proposto possibilita que a transmissão seja interrompida a qualquer momento, permitindo um equilíbrio flexível entre eficiência de compressão e qualidade de reconstrução.

Para aprimorar ainda mais o processo de codificação, investigamos diversas otimizações. Incorporando modelagem de contexto por meio de árvores de partição hierárquica e redes neurais, como *Multi-Layer Perceptrons* (MLPs), que preveem probabilidades condicionais dos coeficientes para melhorar a etapa de codificação aritmética. Também exploramos a codificação de *regiões de interesse* (ROI) para atribuir maior fidelidade a áreas mais relevantes da nuvem de pontos. Além disso, propomos uma melhoria preliminar de modelagem de contextos usando redes neurais em algoritmos de codificação aritmética para compressão de imagens sem perdas, inspirada em avanços recentes na compressão de imagens, evidenciando seu potencial na compressão sem perdas (e eventualmente com perdas) de atributos de nuvens de pontos.

Os resultados experimentais demonstram que o arcabouço proposto obtém desempenho competitivo em termos de taxa-distorção, ao mesmo tempo em que oferece funcionalidade totalmente *embedded*. Ademais, a natureza modular da abordagem — combinando RAHT, SPIHT, modelagem de contexto baseada em redes neurais e estratégias de ROI — torna-a facilmente adaptável a diferentes formatos de nuvens de pontos e casos de uso. Em conjunto, essas contribuições reforçam a viabilidade da codificação *embedded* adaptativa como uma solução robusta e flexível para os desafios impostos por grandes volumes de dados 3D.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

SYMBOLS

V	The geometry of a point cloud, a list of 3D positions, i.e. occupied voxels.
C	List of colors associated with the occupied voxels from V.

DEFINITIONS

octree	A tree data structure in which each internal node has exactly eight children.
pixel	Picture element.
voxel	Volume element.

ACRONYMS

1D	One-dimension
2D	Two-dimensions
3D	Three-dimensions
4D	Four-dimensions
AR	Augmented reality
bpp	Bits per pixel
bpov	Bits per occupied voxel
CfP	Call for Proposals for Point Cloud Compression
CNN	Convolutional Neural Network
DC	Direct Current, represents coefficients with zero frequency
DCM	Direct Coding Mode
fps	Frames per second
G-PCC	Geometry-based Point Cloud Compression
HDTV	High Definition Television
HEVC	High Efficiency Video Encoding
MPEG	Moving Pictures Expert Group
MR	Mixed reality
MSE	Mean-Squared Error
MVUB	Microsoft Voxelized Upper Body
PCC	Point Cloud Compression
PSNR	Peak Signal-to-Noise Ratio
RAHT	Region-Adaptive Hierarchical Transform
RGB	Red, Blue and Green color space

ROI	Region of Interest
SPIHT	Set Partitioning in Hierarchical Trees
TMC13	Test Model Categories 1 and 3
VAE	Variational Autoencoder
VR	Virtual Reality
YUV	Luma, Chrominance blue, and Chrominance red color space, following the BT.709 HDTV standard

1 INTRODUCTION

"If I have seen further it is by standing
on the shoulders of giants."
- Isaac Newton

In information theory, data compression is the process of encoding information using fewer bits than the original representation. It can be lossy or lossless. Lossless compression reduces the bits used by identifying and exploring statistical redundancy to represent data without losing any information. Lossy compression reduces bits by removing less important information, so there is a corresponding trade-off between preserving information and reducing size. A system that predicts the posterior probabilities of a sequence given its entire history can be used for optimal data compression. In the last few years, neural-network-based methods have further advanced image compression efficiency through sophisticated modeling of pixel prediction errors and of context dependencies [5]–[9], showcasing a close connection between machine learning and compression.

In the last decade, we have seen a great revolution in possible ways to represent the world. Advances in computer graphics created technologies like virtual reality (VR), augmented reality (AR) and mixed reality (MR) [10], [11]. These technologies use three-dimensional models to represent scenes and objects. Nowadays it is possible to produce high-quality real-time 3D captures of natural objects or scenes using a variety of computational cameras. Such cameras range from consumer-grade RGBD cameras, to room-sized multi-viewpoint studios, to vehicular acquisition systems. Point clouds are a representation common to all these capture systems just as image arrays are a representation common to ordinary cameras.

Point clouds are a common representation for the three-dimensional world. Nevertheless, point clouds demand a large amount of resources since a typical point cloud may contain millions of points. A major challenge is to efficiently transmit and/or store high-quality point clouds. Moreover, in order to facilitate inter-operation between production and consumption, compression standards for point clouds are being devised by the Moving Picture Experts Group (MPEG) and the Joint Photographic Experts Group (JPEG) [12], [13]. In 2017, MPEG issued a Call for Proposals for Point Cloud Compression (PCC) [14]. One point cloud can be interpreted as the 3D equivalent of an image and a sequence of point clouds can be interpreted as the 3D equivalent of a video.

There has been a significant body of research devoted to point cloud compression (PCC) [15]. Notably, the Moving Picture Experts Group (MPEG) is actively engaged in the development of standards for PCC [12], [16]–[18]. One of the primary requirements laid out for MPEG's geometry-based PCC (G-PCC) [19] was the incorporation of spatial scalability. This objective was centered on constructing a bit stream characterized by a hierarchical structure, thus facilitating a gradual enhancement from coarse to fine levels.

However, G-PCC only partially meets this requirement, and there is currently no advanced core experiment to fulfill this need within MPEG. This issue was partially addressed [20] by approxi-

mating relations among down-sampled point clouds and truncated inverse Region-Adaptive Hierarchical Transform (RAHT) coefficients [21]. It enables attribute reconstruction at specific octree levels. Kathariya et al. [22] introduced a variable-rate coding method based on tree partitioning, in order to enhance compression efficiency and to enable variable quality point cloud representation. Guarda et al.'s [23] proposed a deep-learning-driven method for scalable encoding of point cloud geometry. The last two works only address the geometry part of the scalability issue.

1.1 PROBLEM STATEMENT

Some embedded compression algorithms have been developed and are extensively studied in a two-dimensional space, such as set partitioning in hierarchical trees (SPIHT) [24]. Although it is clear how to incorporate spatial scalability in image coding, performing it for point cloud compression has not been obvious.

This work proposes a fully embedded coder for point cloud attribute compression that can be used directly into the 3D space of point clouds. The research led to two publications in the IEEE Signal Processing Letters [25], [26], one publication at the MMSP conference [27] and another publication at the SBrT conference [28].

1.2 OBJECTIVES

The objectives of this work are to investigate the main challenges in embedded point cloud compression and to investigate possible methods to enhance arithmetic coding using context modeling. Furthermore, to develop a fully embedded coder for point cloud compression exploring the use of set partitioning in hierarchical trees with RAHT coefficients that can be integrated with arithmetic coding optimization via context modeling or with region of interest coding.

1.3 DERIVED WORK

The research developed in this work resulted in the publication of two letters and two conference papers. In order of publication the papers are:

1. A. L. Souto, V. F. Figueiredo, P. A. Chou and R. L. de Queiroz, "**Set Partitioning in Hierarchical Trees for Point Cloud Attribute Compression,**" in IEEE Signal Processing Letters, vol. 28, pp. 1903-1907, 2021. [25]
2. V. F. Figueiredo, R. L. de Queiroz, P. A. Chou and L. S. Lopes, "**Embedded Coding of Point Cloud Attributes,**" in IEEE Signal Processing Letters, vol. 31, pp. 890-893, 2024. [26]

3. V. F. Figueiredo and R. L. de Queiroz, **"Embedded Bit-Stream Region-of-Interest Coding of Point Cloud Attributes,"** 2024 IEEE 26th International Workshop on Multimedia Signal Processing (MMSP), 2024, pp. 1-6 [27]

4. V. F. Figueiredo and R. L. de Queiroz, **"Codificação Progressiva (Embedded) de Região de Interesse para Atributos de Nuvem de Pontos,"** in XLII Simpósio Brasileiro de Telecomunicações e Processamento de Sinais (SBrT), 2024. [28]

5. V. F. Figueiredo, R. L. de Queiroz, L. S. Lopes and P. A. Chou, **"Context-Adaptive Distributions for Neural-Based Lossless Image Compression,"** submitted to IEEE Signal Processing Letters and waiting for review.

1.4 MANUSCRIPT PRESENTATION

In Chapter 2, a literature review is presented in order to explain the concepts required to produce this work. Then, in Chapter 3 is presented the concept of embedded attribute point cloud coding [25]; Chapter 4 describes the possibility of improving the previously presented coder with a neural network to further encoded the generated bit stream [26]; Chapter 5 introduces the use of region of interest coding for embedded point cloud coding [27], [28]; Chapter 6 presents the work done to improve the use of context and neural-based methods for improving the arithmetic coding of the final bit-stream. Finally, in Chapter 7, general conclusions are given.

2 LITERATURE REVIEW

"Life is like riding a bicycle.
To keep your balance,
you must keep moving."
- Albert Einstein

The image coding technique called Embedded Zerotree Wavelet (EZW) [29] was the first to introduce the use of a fully embedded bit stream to image compression that was competitive in bit-rate performance and was fast in execution. Shapiro's EZW coder is a progressive image compression algorithm that exploits the multi resolution tree structure of a wavelet transform to transmit the most informative coefficients first. Despite its simplicity—and with no training, pre-stored tables, code books, or source priors—EZW delivered performance competitive with state-of-the-art methods on standard images when it was published. Its effectiveness rests on four principles: (i) a discrete wavelet transform (hierarchical subband decomposition); (ii) cross-scale prediction of insignificance via self-similarity, encoded compactly as *zerotrees*; (iii) entropy-coded successive-approximation quantization of significant coefficients; and (iv) universal lossless compression using adaptive arithmetic coding. Together, these elements couple multi resolution sparsity with cross-scale structure to achieve efficient rate–distortion performance with low decoding complexity and natural progressive transmission.

Later on, Set Partitioning in Hierarchical Trees (SPIHT) [24] was introduced as an improvement at encoding the coefficients of a wavelet decomposition of an image into a single embedded bit stream, i.e., any prefix of the bit stream is decodable, with longer prefixes yielding higher-quality reconstructions at higher rates (the full bit stream approaching lossless or near-lossless quality). This intrinsic scalability enables progressive transmission, multicast, graceful degradation, and unequal error protection [30]–[32], in contrast to non-embedded coders that require distinct bit streams for different operating points (note that here "embedding" refers to bit stream scalability, not to continuous-valued embeddings in higher-dimensional spaces as in [33]). Encoders and decoders are said to be embedded if they encode and decode embedded bit streams. SPIHT's efficiency stems from compact significance maps obtained by set partitioning coupled with wavelet zerotrees, which concentrate cross-scale insignificance and permit highly efficient signaling. While the set-partitioning paradigm is natural in wavelet subband image coding, transferring it to point cloud compression is nontrivial. In [33], an embedded representation for point clouds is proposed, but it is not an embedded coder, it is a method to describe the structure of a point cloud that is efficient for completion algorithms.

In this work, our main objective is to introduce a set partitioning in hierarchical trees framework for point cloud attribute compression, enabling well-established point cloud coding techniques to be adapted so that they inherit the fully embedded functionality of SPIHT.

2.1 POINT CLOUDS

With the arising and popularization of three-dimensional digital content, it was necessary to develop methods to represent it that would allow its direct consumption by humans. One of the methods developed to perform such representation is the point cloud. Point clouds, as a 3D data representation, can be used in several applications, including virtual/augmented reality, immersive telepresence, autonomous driving, and cultural heritage archival [12], [34].

A point cloud is a set of points in space represented in a three-dimensional (X, Y, Z) coordinate system. It commonly serves the purpose of representing the outer surface of an object or scene. An example of a 3D captured object in its point cloud format is shown in Figure 2.1.

Figure 2.1: Point Clouds RomanOilLight [35] and Stanford Bunny [36], [37].

Point clouds consist of geometry and all its attributes [38]. The geometric part of a point cloud with N points, can be described by a set V containing the coordinates of all points, such that:

$$V = \{v_1, v_2, \ldots, v_n\} = \left\{ \begin{array}{c} (x_1, y_1, z_1) \\ (x_2, y_2, z_2) \\ \vdots \\ (x_n, y_n, z_n) \end{array} \right\} \tag{2.1}$$

where $n = 1, \ldots, N$ and $v_i = (x_i, y_i, z_i)$ defines the position of the point p_i.

Attributes can be represented in a similar way by a set C where each entry in that set has D attributes per point:

$$C = \{c_1, c_2, \ldots, c_n\} = \left\{ \begin{array}{c} (a_{11}, \ldots, a_{1D}) \\ (a_{21}, \ldots, a_{2D}) \\ \vdots \\ (a_{n1}, \ldots, a_{nD}) \end{array} \right\} \tag{2.2}$$

Commonly, attributes include color components, but may also include transparency, normal vectors, motion vectors, and so forth. Once the geometry is given, the attributes may be thought of as a signal defined on a set of points.

The point in the point cloud is the primitive notion on which geometry is built. It has no length, area or volume, it is used as a unique location in Euclidean space. Thus, for rendering, points are commonly represented as spheres. Employing *voxelized* point clouds (VPC) is more convenient for representing such data.

A *voxel* is the 3D extension of a pixel, while the pixel is a square and the fundamental element of a 2D image, the *voxel* is a cube and is the fundamental element of a three-dimensional object.

We assume that the object represented is contained in a cube of size $L \times L \times L$, where L is a positive integer. We can divide this cube into the three dimensions L times, obtaining L^3 cubes of dimension $1 \times 1 \times 1$, which are by definition *voxels*. The advantage of such approach is that the position of each *voxel* is discrete rather than continuous and the fundamental element is no longer dimensionless.

Different from the 2D image case, in the 3D representation by *voxels* the vast majority of them must be transparent. Also, those *voxels* in the interior of the objects we are representing may also be transparent because normal techniques can not capture information in those regions and/or because that information is not relevant, reducing the number of data required to represent an image. Those *voxels* that are not transparent are referred as occupied *voxels* and those that are transparent are commonly referred as non occupied *voxels*.

2.1.1 OCTREE

In computer science, a tree is a data structure in which its elements, called nodes, are hierarchically related to each other. They are composed of an initial node, called the root, plus its children nodes. Each subsequent node can have one or more children. When a node has no children, it is called a leaf node [39].

An octree is a tree data structure in which each node has exactly eight children (except when it is a leaf node) [40], it is the 3D extension of a 2D quad-tree. They are most often used to partition a three-dimensional space and have been shown to be very efficient when encoding a point cloud [41]. Typically, less than 2% of the *voxels* are actually occupied, such that it is easier to record a voxelized point cloud as a list of occupied *voxels* [42].

To explain the octree scanning process, illustrated in Figure 2.2, we start with a cube with dimensions $L \times L \times L$ where ours *voxels* lay. Then we have a list of *voxels* inside our cube, the cube is then divided in half along one of its dimensions (i.e. x axis). At this stage we say that we traveled one level on this octree. The list of occupied *voxels* is divided in two lists: one for the *voxels* laying in the leftmost half and another list for the rightmost half. We continue our division along another dimension (y axis), dividing in four regions and producing 4 *voxel* lists. Finally, repeating the process for the remaining dimension (z axis), we obtain 8 cubes, or octants, similar to the original one, but with a width of $L/2$. At this stage we say that we have traveled one depth on this octree, where 3 levels are the equivalent of 1 depth.

We can continue with this process, further dividing the space. At depth d we will have cubes of size $L/2^d$. If L is chosen to be a power of 2, at the depth d where $2^d = W$, the resulting cubes will have size $1 \times 1 \times 1$ and will accommodate one single *voxel*. For this reason, for most of voxelized point clouds L is chosen to be a power of 2. The higher the value of L, the more detailed the point cloud can be as it comports more *voxels*.

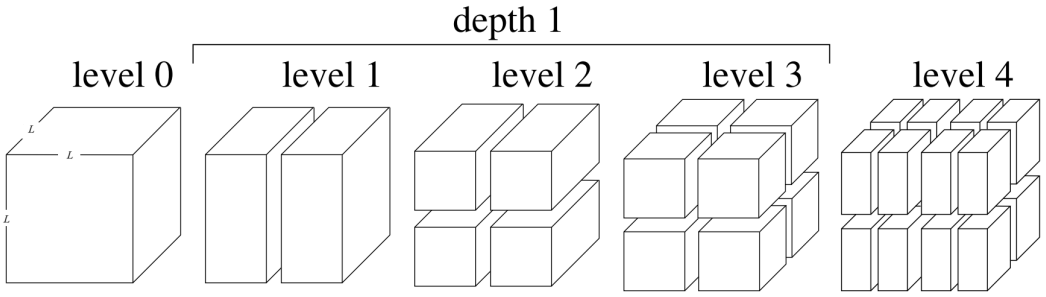


Figure 2.2: Octree scanning of *voxels* [43].

Octree achieves compression by signaling, at each node, which of its eight child octants are occupied, this coding technique eliminates the need to explicitly store the occupancy of every voxel in the 3D space. The same hierarchical representation naturally supports progressive transmission and level-of-detail control: truncating the traversal at a given depth is equivalent to downsampling the point cloud by the corresponding scale factor [44]. In practice, occupancy at each level is conveyed with a binary indicator per octant; because each node partitions space into eight children, these indicators form an 8-bit pattern (one byte) per node per level. Statistical analysis of these occupancy bytes reveals substantial redundancy, which can be exploited by general-purpose entropy coders (e.g., Huffman coding, arithmetic coding, or LZW [45]) to further reduce the bit-rate [46].

2.2 REGION-ADAPTIVE HIERARCHICAL TRANSFORM

MPEG's Geometry-based Point Cloud Compression (G-PCC) [16] standard was designed for compressing static PCs and compresses geometry and attributes separately. In G-PCC, one of the attribute coding methods is based on the region-adaptive hierarchical transform (RAHT) [16], [47]. RAHT is an orthogonal transform that resembles a geometry-adaptive version of the Haar transform [47]. Transform coefficient encoding typically uses arithmetic coding (AC). Since RAHT was developed to be used in real-time telepresence systems, a simpler adaptive run-length Golomb-Rice encoding (RLGR) [48] was originally tested with RAHT. Initially it led to a noticeably inferior performance, compared to the AC coder, while enabling a much simpler implementation. In [38], it was realized that reordering RAHT coefficients could improve the performance of the RLGR-based coder to rival the AC-based one. G-PCC provides a RAHT encoder with a number of improvements in quantization, arithmetic encoding and coefficient prediction.

Most point cloud codecs found in the literature first encode the geometry, and then encode

the attributes conditioned on the geometry. Typical approaches to attribute coding include transform coding using the Graph Fourier Transform (GFT) [49]–[54], the Gaussian Process Transform (GPT, which is the KLT of a Gaussian Process) [55], [56], and the Region-Adaptive Hierarchical Transform (RAHT) [38], [47]. RAHT, unlike the GFT or GPT, does not require an eigen-decomposition, and has been one of the transforms initially adopted into MPEG PCC [12].

The RAHT is a hierarchical orthogonal sub-band transform. It is a variation of the Haar transform that takes the data sparsity into account, it accomplishes this by using adaptive weights to consider different regions with empty or occupied *voxels*.

In the RAHT the weight is set to be 1 to all occupied *voxels* and 0 for void *voxels* initially, then the *voxels* are combined two by two, along each dimension. Let us denote $F^{\ell}_{i,j,k}$ the color of the *voxel* at the position $x = i, y = j, z = k$ at level l and by $w^{l}_{i,j,k}$ its weight. Along the x-dimension, the RAHT is given by:

$$\begin{bmatrix} F^{\ell}_{i,j,k} \\ G^{\ell}_{i,j,k} \end{bmatrix} = \begin{bmatrix} a & b \\ -b & a \end{bmatrix} \begin{bmatrix} F^{\ell+1}_{2i,j,k} \\ F^{\ell+1}_{2i+1,j,k} \end{bmatrix}, \tag{2.3}$$

where

$$a^2 = \frac{w^{\ell+1}_{2i,j,k}}{w^{\ell+1}_{2i,j,k} + w^{\ell+1}_{2i+1,j,k}}, \quad b^2 = \frac{w^{\ell+1}_{2i+1,j,k}}{w^{\ell+1}_{2i,j,k} + w^{\ell+1}_{2i+1,j,k}}. \quad a, b \geq 0. \tag{2.4}$$

and at least one of the *voxels* in the pair is occupied. The inverse transform is given by:

$$\begin{bmatrix} F^{\ell+1}_{2i,j,k} \\ F^{\ell+1}_{2i+1,j,k} \end{bmatrix} = \begin{bmatrix} a & -b \\ b & a \end{bmatrix} \begin{bmatrix} F^{\ell}_{i,j,k} \\ G^{\ell}_{i,j,k} \end{bmatrix}. \tag{2.5}$$

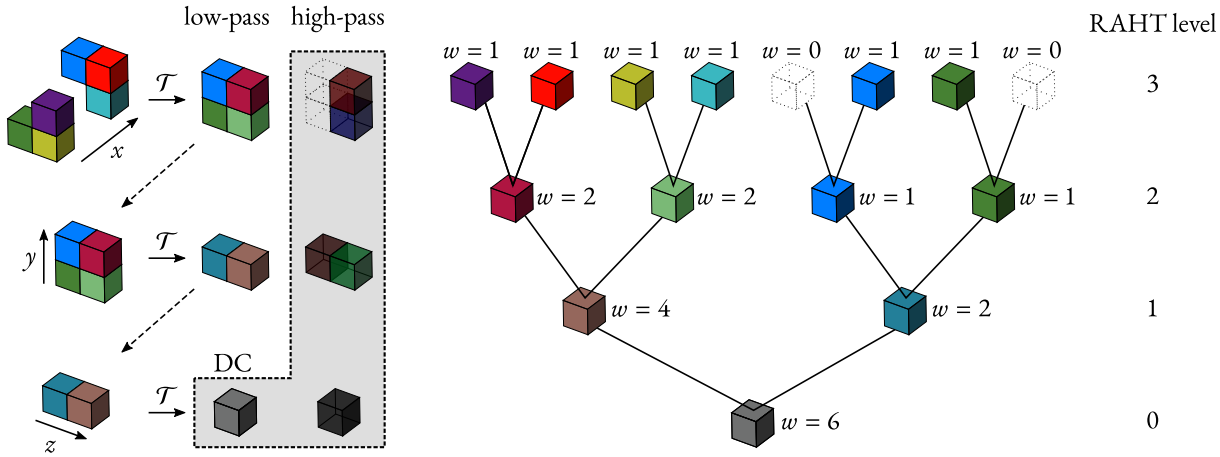


Figure 2.3: RAHT diagram for a $2 \times 2 \times 2$ block. [34]

In RAHT the high-pass coefficient (G) is not subject to further processing along the other dimension. They are sent directly to the next compression steps. The low-pass coefficients (F), on the other hand, are treated as new *voxels*. The process is repeated along each dimension (x, y and z) until reaching the level where only one low-pass coefficient remains, the DC coefficient. When occupied voxels are paired with empty ones, their attributes are directly promoted to the next

level, but their weights do not change. Thus, densely populated regions get more importance than sparser ones in the transform. The DC coefficient is sent to the decoder along with all the high-pass coefficients generated in the process where they are quantized and entropy coded. In Figure 2.3 is shown a diagram for the RAHT implementation for a $2 \times 2 \times 2$ block.

2.3 SET PARTITIONING IN HIERARCHICAL TREES

The SPIHT algorithm [24] is a bit-plane coder that uses a hierarchical set partition to code the bit-plane information. At its heart, the basic SPIHT algorithm is very simple. The encoder is given a step-size Δ, a collection of coefficients $\{c_t : t \in T\}$, and a hierarchical set partition of T. The encoder quantizes each coefficient c_t into an integer $k_t = \text{round}(c_t/\Delta)$, and then transmits the bit-planes of $|k_t|$ along with $\text{sign}(k_t)$ if k_t is non-zero.

The bit-planes are transmitted from the most significant bit-plane $b = b_{\max} = \lfloor \log_2(\max_{t \in T} |k_t|) \rfloor$ to the least significant bit-plane $b = 0$. A coefficient c_t is said to be *significant* in bit-plane b if $\lfloor \log_2(|k_t|) \rfloor \geq b$. Otherwise it is *insignificant* in bit-plane b. As b decreases, more and more coefficients become significant. In each bit-plane b, there is a set of coefficients that were already significant in the previous bit-plane $b + 1$; these coefficients remain significant in bit-plane b. The remaining coefficients may become newly significant in bit-plane b, or they may remain insignificant in bit-plane b. SPIHT transmits information to signal which of these remaining coefficients become newly significant in bit-plane b, and which remain insignificant in bit-plane b. How this information is coded is what distinguishes SPIHT from other bit-plane coders. Specifically, SPIHT uses a hierarchical set partition to code this information.

A hierarchical set partition of T is a partition of T into sets, and a further partition of those sets into sets, and a further partition of those sets into sets, and so on, until there is only a single set of sets T_0 remaining. The hierarchical set partition forms a tree structure. At the leaves of the tree are *singleton* sets, that is, sets $\{t\}$ containing only single elements $t \in T$. At the internal nodes of the tree are *compound* sets, that is, sets containing one or more other sets. Figure 2.4 depicts the hierarchical tree structure for an image.

A singleton set $\{t\}$ in the hierarchical set partition is said to be *significant* (*insignificant*) if its coefficient c_t is significant (insignificant). Furthermore, a compound set in the hierarchical set partition is said to be *significant* if *any* of its member sets are *significant*, or *insignificant* if *all* of its member sets are *insignificant*. This definition is recursive. A set must be either significant or insignificant.

SPIHT can use a single bit to declare that an entire set T' in the hierarchical set partition is insignificant in bit-plane b, needing no further attention in that bit-plane. However, if T' turns out to be significant, then SPIHT must open up the set to consider which of its elements are actually significant. If T' is a singleton set $\{t\}$, then c_t is newly significant. Otherwise, if T' is a compound set $\{T'_1, \ldots, T'_n\}$, then at least one of the sets $T'_1, \ldots, T'_n$ must be significant, and SPIHT must re-

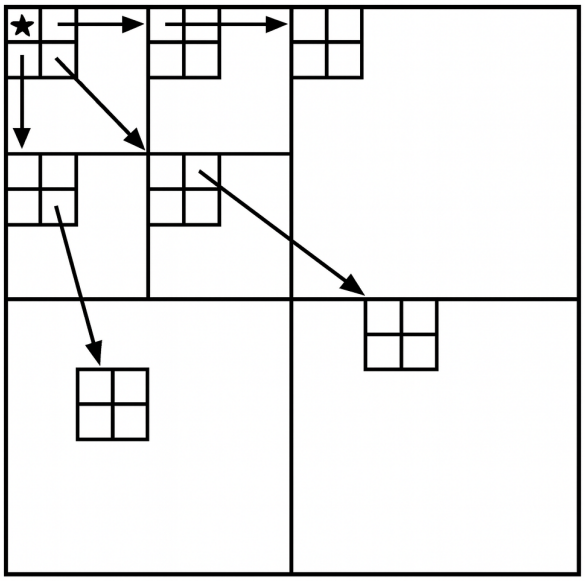


Figure 2.4: Example of the spatial hierarchical tree for an image and its parent-offspring dependencies [24].

cursively use a single bit to signal whether or not each member set is significant. (However, no bit is needed to signal whether or not T'_n is significant if all of its siblings $T'_1, \ldots, T'_{n-1}$ are insignificant.) A version of the SPIHT *encoding* algorithm, analogous to the original algorithm in [24], is shown in Algorithm 1 and in Figure 2.5. The *decoding* algorithm is similar, replacing encoding each bit by decoding each bit, as seen in Algorithm 2. In the algorithm, the following quantities are defined:

- $O(t)$: the nodes that are direct descendants of node t,
- $\mathcal{D}(t)$: the set of nodes that are descendants of node t,
- $\mathcal{L}(t)$: set defined by $\mathcal{L}(t) = \mathcal{D}(t) - O(t)$,
- LIS: List of Insignificant Sets, and
- LSN: List of Significant Nodes.

With this, two different types of compound sets are defined:

- $Type\ A$: $T' = \mathcal{D}(t) = \{\{t_c\} : t_c \in O(t)\} \cup \{\mathcal{L}(t)\}$, and
- $Type\ B$: $T' = \mathcal{L}(t) = \{\mathcal{D}(t_c) : t_c \in O(t)\}$.

After quantizing the coefficients, the SPIHT encoder initializes a List of Significant Nodes (LSN) to the empty set, initializes a List of Significant Sets (LIS) to the list of sets at the top level of the hierarchical set partition, and initializes b to the most significant bit-plane $b = b_{\max}$. Then it processes the bit-planes in sequence through the least significant bit-plane $b = 0$.

The basic SPIHT encoder divides the processing for bit-plane b into a Refinement Pass and a Sorting Pass. Prior to the Refinement Pass, the indices t of any coefficients that might have become

Algorithm 1 SPIHT encoder

Input: $\Delta, \{c_t : t \in T\}, T_0$

/* Initialization */
for $t \in T$ **do** $k_t = \text{round}(c_t/\Delta)$
$LSN \leftarrow$ empty list ▷ List of Significant Nodes
$LIS \leftarrow$ list of member sets in T_0 ▷ List of Insignificant Sets
$b \leftarrow b_{\max} = \lfloor \log_2(\max_{t \in T} |k_t|) \rfloor$ ▷ index of highest bit-plane

/* Refinement Pass */
for each element $t \in LSN$ **do**
    Encode bit b of k_t ▷ refinement bit
end for

/* Sorting Pass */
for each set T' in LIS **do**
    Encode $S_b(T') = \mathbb{1}\{\lfloor \log_2(\max_{t \in T'} |k_t|) \rfloor \geq b\}$ ▷ significance bit
    if $S_b(T') = 1$ **then**
        if T' is a singleton $\{t\}$ **then**
            Encode the sign of k_t ▷ sign bit
            Remove T' from LIS; add t to LSN
        else// T' is a set of sets
            if T' is of type A **then**
                for each singleton set in $O(T')$ **do**
                    Encode $S_b(O(T')) = \mathbb{1}\{\lfloor \log_2(\max_{t \in T'} |k_t|) \rfloor \geq b\}$
                    if $S_b(O(T')) = 1$ **then**
                        Encode the sign of k_t
                        Add t to LSN
                    end if
                end for
                if $\mathcal{L}(T')$ is not empty **then**
                    Move T' to the end of the LIS as a type B
                    Go to Step 2
                else
                    Delete T' from LIS
                end if
            else//T' is of type B ▷ Step 2
                Encode $S_b(\mathcal{L}(T')) = \mathbb{1}\{\lfloor \log_2(\max_{t \in T'} |k_t|) \rfloor \geq b\}$
                Add member sets in T' to LIS as type A
                Delete T' from LIS
            end if
        end if
    end if
end for

/* Iterate */
$b \leftarrow b - 1$
if $b > 0$ **then** go back to Refinement Pass
end if

Output: bit stream

Algorithm 2 SPIHT decoder

Input: bit stream, $b_{\max}$, T_0 ▷ tree structure as in the encoder

/* Initialization */
for all $t \in T$ **do** $\hat{k}_t \leftarrow 0$
$LSN \leftarrow$ empty list ▷ List of Significant Nodes
$LIS \leftarrow$ list of member sets in T_0 ▷ List of Insignificant Sets
$b \leftarrow b_{\max}$ ▷ index of highest bit-plane
/* Refinement Pass */
for each element $t \in LSN$ **do**
    Decode refinement bit $r \in \{0,1\}$ for bit-plane b of $\hat{k}_t$
    if $r = 1$ **then** set $|\hat{k}_t| \leftarrow |\hat{k}_t| + 2^b$ with sign $\text{sign}(\hat{k}_t)$
    end if
end for
/* Sorting Pass */
for each set T' in LIS **do**
    Decode $S_b(T') \in \{0,1\}$ ▷ significance bit
    if $S_b(T') = 1$ **then**
        if T' is a singleton $\{t\}$ **then**
            Decode the sign of $\hat{k}_t$ ▷ sign bit
            Set $|\hat{k}_t| \leftarrow |\hat{k}_t| + 2^b$ and add t to LSN
            Remove T' from LIS
        else// T' is a set of sets
            if T' is of type A **then**
                for each singleton set $\{t\}$ in $O(T')$ **do**
                    Decode $S_b(\{t\}) \in \{0,1\}$ ▷ significance of each offspring
                    if $S_b(\{t\}) = 1$ **then**
                        Decode the sign of $\hat{k}_t$
                        Set $|\hat{k}_t| \leftarrow |\hat{k}_t| + 2^b$ and add t to LSN
                    end if
                end for
                if $\mathcal{L}(T')$ is not empty **then**
                    Move T' to the end of the LIS as a type B
                    Go to Step 2
                else
                    Delete T' from LIS
                end if
            else// T' is of type B ▷ Step 2
                Decode $S_b(\mathcal{L}(T')) \in \{0,1\}$
                if $S_b(\mathcal{L}(T')) = 1$ **then**
                    Add member sets in T' to LIS as type A
                    Delete T' from LIS
                end if
            end if
        end if
    end if
end for
/* Iterate */
$b \leftarrow b - 1$
if $b \geq 0$ **then** go back to Refinement Pass
end if

Output: reconstructed integers $\{\hat{k}_t : t \in T\}$

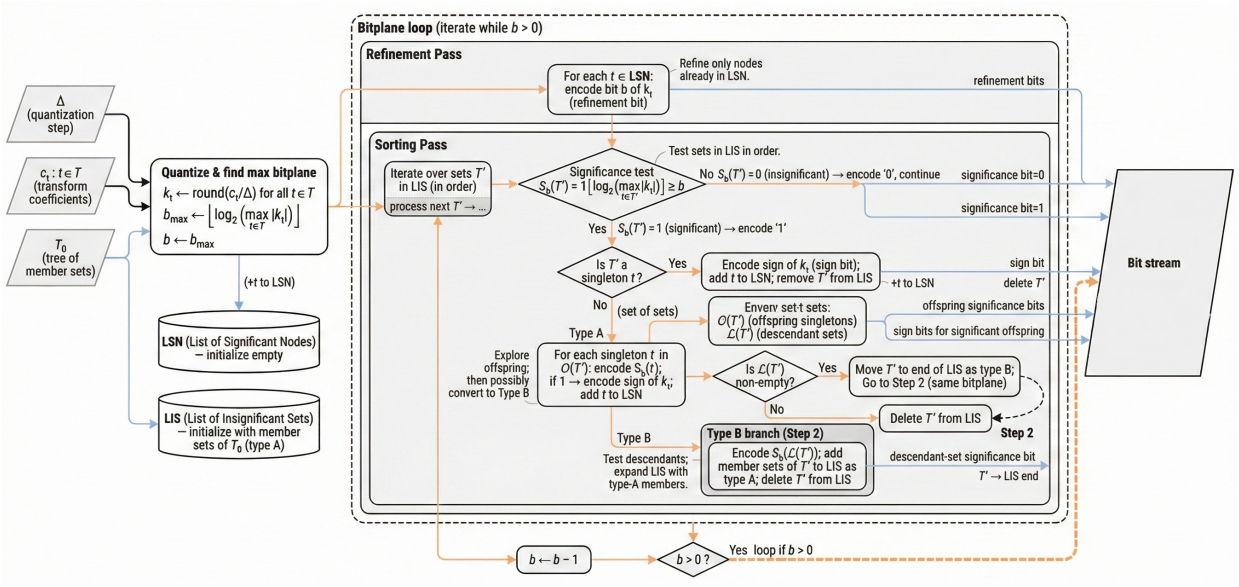


Figure 2.5: Diagram of the SPIHT encoder algorithm described in Algorithm 1.

significant in a previous bit-plane are found in the LSN. Thus in the Refinement Pass, SPIHT refines all such coefficients by encoding bit b of $|k_t|$ for all elements t in the LSN. More elements t will be added to the LSN during the Sorting Pass, when their coefficients c_t become newly significant. The the LSN will be processed anew in the next bit-plane.

In the Sorting Pass, the sets T' found in the LIS are processed sequentially, using a cursor to point to the current set being processed, until there are no more sets in the LIS to process. Some sets T', after being processed, are left in the LIS after the cursor moves to the next set. Because they remain in the LIS, such sets will be re-examined in subsequent bit-planes. Other sets T', after being processed, will be deleted from the LIS after the cursor moves to the next set. Hence they will not be re-examined in subsequent bit-planes. However, such sets will usually result in more sets being added to the end of the LIS. The added sets will eventually be processed in the current bit-plane when the cursor gets to them. Thus the length of the LIS generally increases as its members are processed.

When a set T' in the LIS is processed, a bit $S_b(T')$ is encoded to signal whether T' is significant or not in bit-plane b. If T' is insignificant, then no action is taken; the cursor advances past T' but T' remains in the list for processing in a subsequent bit-plane. However, if T' is significant, then it is opened and its members are examined: If T' is a singleton $\{t\}$, then the sign of k_t is encoded and the element t is put into the LSN for later refinement in a subsequent bit-plane. If T' is a set of sets, then its member sets are added to the end of the LIS for later examination in bit-plane b. In both cases, the cursor advances past T' but T' itself is removed from the LIS so that is is not examined again in a subsequent bit-plane.

For example, suppose T comprises 10 coefficient indices $t = 0, \ldots, 9$. One way that these could

be grouped into a hierarchical set partition T_0 is the following:

$$\begin{aligned} T_0 &= \{\{0\}, T_{11}, T_{12}\} && (2.6)\\ &= \{\{0\}, \{\{1\}, T_{21}\}, \{\{4\}, T_{22}, T_{23}\}\} && (2.7)\\ &= \{\{0\}, \{\{1\}, \{\{2\}, T_{31}\}\}, \{\{4\}, \{\{5\}, T_{32}, T_{33}\}, \{\{8\}, T_{34}\}\}\} && (2.8)\\ &= \{\{0\}, \{\{1\}, \{\{2\}, \{3\}\}\}, \{\{4\}, \{\{5\}, \{6\}, \{7\}\}, \{\{8\}, \{9\}\}\}\}. && (2.9) \end{aligned}$$

Here, T_0 is a set of three sets: $\{0\}$, T_{11}, and T_{12}. The set $\{0\}$ is the singleton set consisting of only the element 0, while T_{11} is a set of two sets ($\{1\}$ and T_{21}) and T_{12} is a set of three sets ($\{4\}$, T_{22}, and T_{23}). The sets T_{21}, T_{22}, and T_{23} are each further defined as sets of sets. Eventually we are left only with singleton sets containing the indices $t = 0, \ldots, 9$, which cannot be further decomposed.

To continue with the example, suppose c_t and Δ are such that $k_t = 9 - t$ (i.e., $k_0 = 9$, $k_1 = 8$, ..., $k_9 = 0$). Thus $t = 0, 1$ become significant at $b = 3$, $t = 2, 3, 4, 5$ become significant at $b = 2$, $t = 6, 7$ become significant at $b = 1$, and $t = 8$ becomes significant at $b = 0$. $t = 9$ never becomes significant.

The basic SPIHT encoder initializes the LSN to the empty list [], and initializes the LIS to $[\{0\}, T_{11}, T_{12}]$. At bit-plane $b = 3$, the LSN is empty, so there is nothing done in the Refinement Pass. In the Sorting Pass, the cursor starts at the first set, $\{0\}$, which is declared to be significant. Then, since it is a singleton, it is unwrapped as the element 0, the sign of k_0 is encoded, and 0 is put in the LSN, which becomes [0]. The cursor is moved to the next set, T_{11}, and $\{0\}$ is removed from the LIS, leaving $[T_{11}, T_{12}]$. Since T_{11}, when recursively enumerated, contains the elements $t = 1, 2, 3$, the first of which is significant in bit-plane $b = 3$, the whole set T_{11} is deemed significant. Hence SPIHT encodes $S_3(T_{11}) = 1$ and opens the set T_{11} by expanding it into its member sets $\{1\}$ and T_{21}. It adds these member sets to the end of LIS, moves the cursor to the next set, T_{21}, and deletes the set T_{11} from the LIS, leaving the LIS equal to $[T_{12}, \{1\}, T_{21}]$. Since T_{12}, when recursively enumerated, contains the elements $t = 4, 5, 6, 7, 8, 9$, none of which is significant in bit-plane $b = 3$, the whole set T_{12} is deemed insignificant. Hence SPIHT encodes $S_3(T_{12}) = 0$, advances the cursor to the next set, $\{1\}$, and leaves T_{12} in the LIS, which is now equal to $[T_{12}, \{1\}, T_{21}]$. Since $\{1\}$, when recursively enumerated, contains the element $t = 1$, which is significant in bit-plane $b = 3$, the set $\{1\}$ is deemed significant, and SPIHT encodes $S_3(\{1\}) = 1$. When SPIHT opens the set and finds that it is a singleton 1, it encodes the sign of k_1, advances the cursor to the next set, T_{21}, removes $\{1\}$ from the LIS, and adds 1 to the LSN, which becomes $[0, 1]$. This leaves the LIS equal to $[T_{12}, T_{21}]$. Since T_{21}, when recursively enumerated, contains $t = 2, 3$, neither of which is significant in bit-plane $b = 3$, the set T_{21} is deemed insignificant. Hence SPIHT encodes $S_3(T_{21}) = 0$, tries to advance the cursor, and leaves T_{21} in the LIS. The LIS remains equal to $[T_{12}, T_{21}]$. Since there are no more set in the LIS to process, bit-plane $b = 3$ is done.

Bit-plane $b = 2$ begins with $LSN = [0, 1]$ and $LIS = [T_{12}, T_{21}]$. In the Refinement Pass, SPIHT encodes bit $b = 2$ of each of $|k_0|$ and $|k_1|$. In the Sorting Pass, the cursor starts again at the beginning of the LIS, pointing to the set T_{12}. The sets are processed as before, ultimately leaving $LSN = [0, 1, 4, 2, 3, 5]$ and $LIS = [T_{23}, \{6\}, \{7\}]$ at the end of bit-plane $b = 2$. SPIHT processes bit-planes $b = 1$ and $b = 0$ in the same manner.

2.4 REGION OF INTEREST

A Region of Interest (ROI) is defined as samples of a data set identified for a particular purpose [57]. The concept of a ROI is commonly used in many application areas, for example, computer vision, medical imaging, optical character recognition (OCR), geographical information systems and others. In computer vision the ROI usually defines the limits of the area under consideration of the image or video.

The region of interest can be specified in a one-dimensional, two-dimensional (see Fig.2.6 for an example), three-dimensional or four-dimensional dataset. Examples of regions of interest of each of these types are given below.

- 1D data set: a frequency interval of a waveform;
- 2D data set: the contour of an object in an image;
- 3D data set: the contour of a surface of an object in a volume;
- 4D data set: the contour of an object during a time interval in a volume.

A ROI is an extra information expressed in a structured form that needs to be encoded. It can usually be encoded as an integral part of the data set as a masking value which tags individual data cells; as a graphic information such as drawing elements; as a set of spatial and/or temporal coordinates.

Regions of interest can be generated manually, semi-automatically, or automatically via software. To distinguish:

- Manual ROI: the user manually defines an area using a keyboard, mouse or other accessory that enables the delimitation;
- Semi-automatic ROI: the software assists the user in drawing or delimits the region based on parameters provided by the user;
- Automatic ROI: the software determines the boundary criteria without user intervention.

Regions of interest are widely used in image compression, such as in the standard *JPEG2000* [58], and in videos [59]. With the detection of regions of interest in point clouds it is possible to use it to preserve the quality of some regions in the compression process.

2.5 NEURAL NETWORK APPROACHES FOR COMPRESSION

Neural networks have become central to a wide range of data compression tasks in recent years, owing to their ability to learn intricate data distributions. In classical approaches, compression

Figure 2.6: Region of Interest highlighted in the image by red rectangle.

pipelines rely on heuristically designed transforms (e.g., Fourier or wavelet transforms) followed by quantization and entropy coding [60]–[63]. In contrast, neural-network-based methods can learn an optimal transform or representation tailored to the underlying data distribution, sometimes yielding better compression performance [5]–[9].

Point cloud compression is a challenging problem due to the sparse nature of 3D data. Neural networks, particularly those capable of handling 3D inputs or learning implicit distributions, have shown promise in significantly reducing bit-rates while preserving the geometric and attribute information of point clouds. Within the broad family of neural networks, Multilayer Perceptrons (MLPs), Convolutional Neural Networks (CNNs) and Transformers have emerged as effective building blocks or prior models for compression systems. These networks can be used both as encoders and as probability estimators to drive more effective adaptive arithmetic coders.

2.5.1 MULTILAYER PERCEPTRONS (MLP)

A Multilayer Perceptron is a fully connected feedforward neural network consisting of multiple fully connected layers of neurons, each with nonlinear activation functions (e.g., ReLU, sigmoid, tanh) [64]. MLPs are considered the fundamental building block of many deep learning architectures and can approximate a wide variety of continuous functions given sufficient depth and width [65]. Their simplicity and universal approximation capabilities make them a flexible choice for numerous tasks, including compression.

Despite the rise of more specialized architectures like CNNs and Transformers, MLPs still play

a role in point cloud compression systems, especially when dealing with:

- **Attribute or feature modeling:** MLPs can model local correlations in point cloud attributes (e.g., color, reflectance, semantic labels) by learning implicit functions that map point coordinates to attributes.
- **Distribution estimation:** In an adaptive arithmetic coding context, an MLP can be trained to predict the probability distribution of the next symbol (or bit), thereby guiding the arithmetic coder toward more efficient encoding [66].

MLPs have the advantage of its simplicity and ability to handle smaller-scale feature learning tasks. However, they typically do not exploit local spatial correlations in structured data as effectively as convolutional networks. When dealing with large-scale data, MLPs may become prohibitively large or under perform if not carefully structured.

2.5.2 CONVOLUTIONAL NEURAL NETWORKS (CNN)

Convolutional Neural Networks apply convolution operations with learnable filters, making them highly efficient at capturing local, spatial correlations. Initially popularized for 2D image processing [67], CNNs have since been adapted to 3D data, including volumetric grids and point clouds (e.g., VoxelNet, PointNet++ [68], [69]).

In compression, CNNs can be used in different stages of the encoder, such as:

- **Transform learning for geometry or color attributes:** CNNs can be used to learn data-driven transforms that map raw point cloud geometry or attributes into a latent space that is more amenable to quantization and entropy coding.
- **Feature extraction for distribution modeling:** In combination with an MLP or a hyperprior network, CNNs can extract salient features that inform the probability distribution models for arithmetic coding. This is especially relevant in hierarchical or multi-scale compression schemes where local neighborhoods of points are processed progressively.
- **Residual or multi-scale architectures:** Residual blocks and multi-scale feature extraction are often employed to better capture both global context (overall shape of the point cloud) and local detail (fine geometry or texture).

CNNs naturally exploit local spatial structure, often achieving higher accuracy or lower distortion for a given bit-rate compared to MLP-based approaches alone.

2.5.3 TRANSFORMERS

Transformers are neural architectures built around *self-attention* mechanisms that adaptively reweight interactions among input tokens [70]. In contrast to convolutions with fixed, local receptive fields, self-attention forms content-adaptive, potentially long-range dependencies, which

is advantageous for data modalities that exhibit irregular sampling and multi-scale structure—as is the case for point clouds. Given query, key, and value projections $\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$, $\mathbf{K} = \mathbf{X}\mathbf{W}_K$, and $\mathbf{V} = \mathbf{X}\mathbf{W}_V$, the scaled dot-product attention for one head is

$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V}, \tag{2.10}$$

where d_k is the key dimension. Multi-head attention applies this operation in parallel across several heads, followed by a linear combination. For point clouds, tokens can represent points, voxels, or patch-level aggregates; positional information (e.g., coordinates, normals) is injected via absolute or relative encodings so that attention remains geometry-aware. Transformers can be integrated into compression systems at multiple stages:

- **Transform learning (analysis/synthesis).** As learned transforms, Transformers map raw attributes (and/or geometry) to compact latents and reconstruct them at the decoder. Their ability to capture non-local correlations can improve latent decorrelation before quantization.
- **Entropy modeling.** Self-attention provides a powerful context model for adaptive arithmetic coding: the network conditions the distribution of a latent symbol on previously coded symbols and spatial neighbors. This can be implemented with *causal masking* (to respect decoding order) and with *relative positional* terms so that the predicted parameters (e.g., mean/scale of Laplacian or Gaussian) reflect local geometry.
- **ROI-aware allocation.** By injecting ROI masks or saliency maps as additional tokens, attention naturally concentrates modeling capacity on important regions, enabling finer control of rate allocation in embedded bit streams.

2.5.4 SCALE HYPERPRIOR NETWORKS

A scale hyperprior model is a deep learning framework popularized for learned image compression by Ballé et al. [5], [6]. The central idea is to learn a latent representation of the data as well as a separate "hyper" encoder that captures the scale (or variance) information of the latent representation. This hyperprior is then used to parameterize the probability distribution (e.g., a Gaussian) of the main latent's elements.

In the context of entropy coding, the scale hyperprior network provides a powerful mechanism for adaptive probability modeling, improving the precision of the distribution estimates used by an arithmetic coder. The scale hyperprior can be used for compression, as follows:

- **Latent representation:** The main network (often a CNN or MLP) encodes the date into a compressed latent representation.
- **Hyper encoder:** A secondary network encodes side information (e.g., local variance or scale parameters) about the latent representation, enabling finer-grained probability modeling.

- **Adaptive arithmetic coding:** Using the hyperprior, the decoder can more accurately predict the likelihood of each latent component, resulting in improved bit allocation during arithmetic coding.

This approach has several advantages such as its scalability, the hyperprior can be designed to be relatively small, preserving compression efficiency, its flexibility, it can be integrated with various backbone architectures, and by conditioning on the hyperprior, the model can assign bits more efficiently where needed.

2.5.5 INTEGRATION WITH AN ADAPTIVE ARITHMETIC CODER

Adaptive arithmetic coding encodes data by progressively refining an interval based on the predicted probability of each symbol. Its efficiency relies on accurate probability estimates for each symbol. In conventional systems, probability models are manually designed, leveraging heuristics about the data's statistics. In neural-network-based systems, the probability estimation is learned directly from data.

An encoder that integrates adaptive arithmetic coding with neural networks could work by first an MLP or CNN outputting parameters of a probability distribution (e.g., mean and variance of a Gaussian), after that another network, like a transformer, could further refines the variance estimates by encoding higher-order statistics. Then for each symbol (or group of symbols) in the latent representation, the distribution parameters are computed in real-time, making it adaptive. Finally, the arithmetic coder uses these parameters to update the coding interval, achieving near-optimal entropy coding.

This synergy between neural networks and adaptive arithmetic coding ensures a data-driven and context-aware compression process, often outperforming traditional methods in terms of rate distortion performance on 3D point clouds [71], [72].

2.6 CLASSICAL APPROACHES FOR LOSSLESS CODING

2.6.1 CALIC: CONTEXT-BASED ADAPTIVE LOSSLESS IMAGE CODING

CALIC (Context-based, Adaptive, Lossless Image Codec) is a predictive, entropy-coded scheme that achieves high lossless compression by coupling a powerful nonlinear predictor with fine-grained context modeling of local image structure. Rather than relying on fixed linear prediction, CALIC conditions both prediction and residual coding on a large number of modeling contexts that capture local gradients and textures, enabling rapid adaptation to spatially varying statistics without pretraining or codebooks. In its original formulation, CALIC emphasizes data modeling over transform design: it estimates the expectation of prediction errors conditioned on context and feeds context-conditioned residuals to an arithmetic coder, delivering state-of-the-art efficiency for

continuous-tone images at modest computational cost [60], [61].

At each pixel, CALIC forms a geometry-aware context from causal neighbors (e.g., directional gradients) and applies a nonlinear, gradient-adjusted prediction (GAP) to produce a locally adaptive estimate. The codec then models the prediction error under that context—capturing both local bias and scale—and maps it to a form well suited for arithmetic coding. Crucially, CALIC's effectiveness stems from (i) a large, carefully quantized set of contexts that reflect edge orientation and activity, and (ii) continuous adaptation of the predictor and residual model to the currently active context, rather than attempting to learn full conditional distributions. This strategy reduces residual entropy across a wide range of images and underpins its strong rate performance [73].

Subsequent work on LOCO-I and JPEG-LS [74], [75] adopted related principles (simple edge-aware prediction, context modeling, and low-complexity entropy coding). Comparative studies [76] report that CALIC maintains a slight rate advantage ($\approx 1-2\%$) on smooth images, whereas LOCO-I excels on sparse-histogram or compound images and was ultimately standardized due to its lower complexity and single-pass design; CALIC has remained a strong benchmark for lossless coding research. These results highlight the enduring value of context-conditioned prediction and residual modeling as building blocks for efficient, standards-compatible lossless image compression.

2.6.2 HIGH EFFICIENCY VIDEO CODING (HEVC)

High Efficiency Video Coding (HEVC) provides a lossless intra mode that preserves the standard's block-based prediction architecture while ensuring exact reconstruction. In the initial specification (HEVC v1), lossless operation is enabled by bypassing transform and quantization—often called "transquant bypass"—and entropy-coding the raw prediction residuals with CABAC. This design minimizes changes to the lossy pipeline and keeps complexity low; however, directly coding spatially correlated residuals can be inefficient, motivating additional tools and refinements introduced in subsequent work and extensions [62].

A key advance specific to lossless intra coding is to improve decorrelation before entropy coding. One influential line of work introduces sample-based mechanisms in place of purely block-based processing. In [77], it was described sample-based angular intra prediction (SAP), which predicts pixels along directional modes at sample granularity, better capturing local edges and textures than block predictors and thus lowering the entropy of the ensuing residuals. In parallel, the HEVC Range Extensions (HEVC v2/REext) standardized residual differential pulse-code modulation (RDPCM): after conventional intra prediction, the residuals are differenced along horizontal or vertical directions (mode-dependent), with syntax signaling governed by simple rules, and CABAC was modestly adapted (e.g., scan/reverse-scan and context tweaks) to the altered residual statistics. Together, these measures significantly reduce the bit-rate of lossless intra coding relative to the first version bypass baseline [78], [79].

Beyond SAP and RDPCM, numerous studies explored complementary refinements that remain conceptually aligned with HEVC's block framework [80]. Examples include cross-residual

and integer-to-integer (i2i) transforms that map integers to integers without dynamic-range growth, allowing transform-domain decorrelation while preserving exact invertibility; these methods report competitive results when integrated with the HEVC reference software. Collectively, such techniques—together with minor CABAC adaptations—illustrate the general strategy for HEVC intra lossless: retain the standard intra-prediction syntax and decoding flow, but enhance pre-coding decorrelation (pixelwise or residualwise) so that CABAC sees lower-entropy symbols, improving rate–compression efficiency without sacrificing the determinism and interoperability required of a standard.

3 SET PARTITIONING IN HIERARCHICAL TREES FOR POINT CLOUD ATTRIBUTE COMPRESSION

"Even the very wise cannot see all ends."
- Gandalf

In this chapter, the proposed method for an embedded attribute encoder for point clouds based on set partitioning in hierarchical trees (SPIHT) [24] is presented.The encoder is used with the region-adaptive hierarchical transform (RAHT), which has been a popular transform for point cloud coding and is included in the standard geometry-based point cloud coder (G-PCC) [12], [34]. The result is an encoder that is efficient, scalable, and embedded. That is, higher compression is achieved by trimming the full bit stream. Despite the inherent scalability of octree-based geometry descriptions, current attribute compression techniques prevent full scalability of compressed point clouds. G-PCC's RAHT coefficient prediction prevents the straightforward incorporation of SPIHT into G-PCC. However, the results over other RAHT-based coders are promising, improving over the original, non-predictive RAHT encoder, while providing the key functionality of being embedded.

Although it is clear how to do set partitioning for wavelet sub-band image coding, performing set partitioning for point cloud compression has not been obvious. In this work, it is explored the use of set partitioning in hierarchical trees in the context of PC attribute compression. SPIHT is an alternative to encoding RAHT coefficients with either arithmetic coder or run-length Golomb-Rice. SPIHT, unlike either alternative, is an embedded coder, where the bit stream is built by progressively increasing the quality and resolution of the PC. It is inherently scalable and allows for progressive transmission. There is a single bit stream, and the amount of compression achieved is determined by how much of the bit stream is kept. Figure 3.1 illustrates the use of an embedded bit stream: as more of the bit stream is used, a better resolution is obtained.

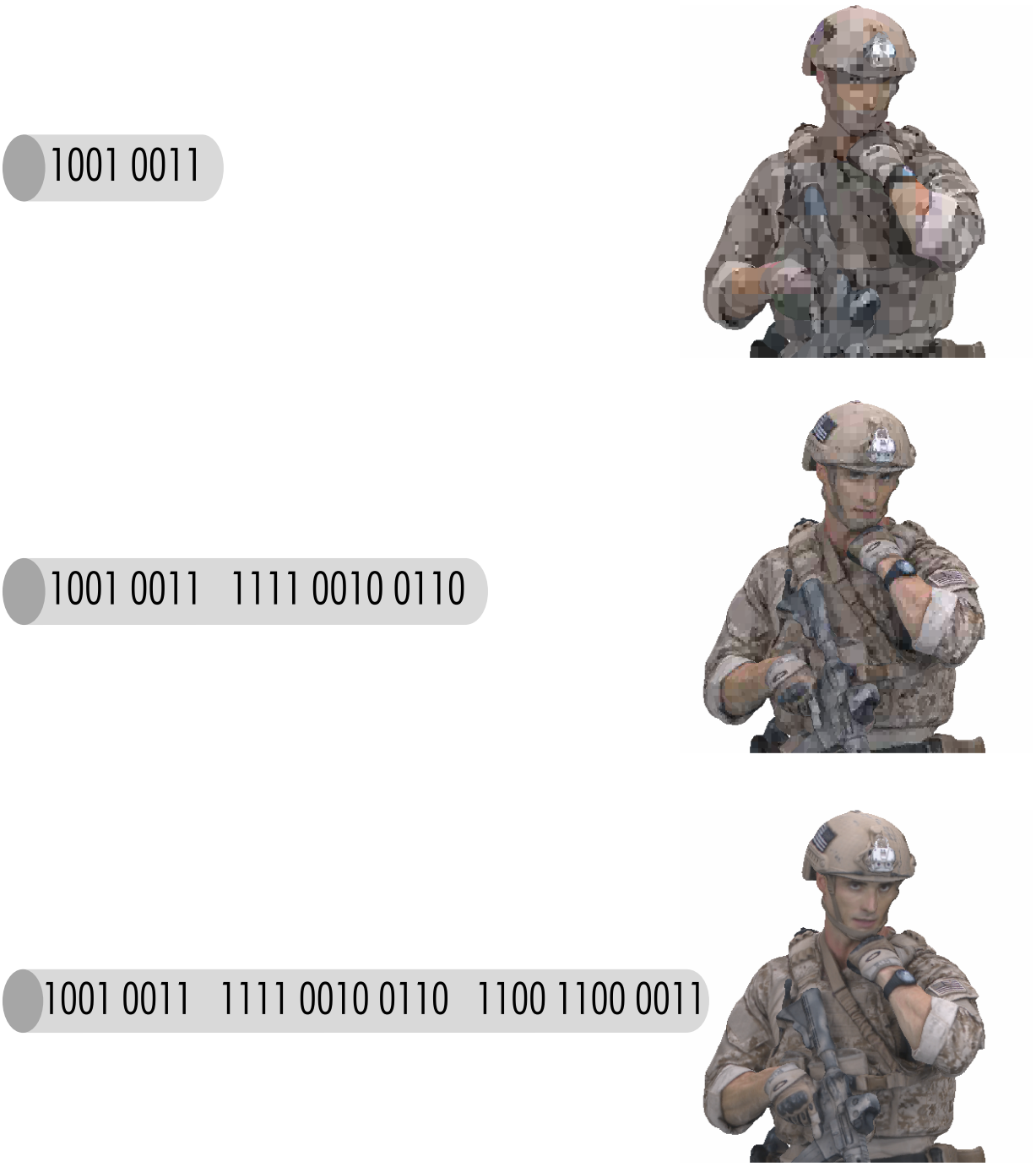


Figure 3.1: Illustration representing the embedded feature of a coder. One bit stream supports variable quality.

3.1 ENCODING AND DECODING OF BIT-PLANES

Let x be a real-valued coefficient to encode, let Δ_0 be an initial (finest) quantization step-size, and let

$$k_0 = \text{round}(x/\Delta_0) \tag{3.1}$$

be the signed quantization index into which x is encoded with a uniform threshold scalar quantizer. In its bit-plane representation, k_0 is represented by a non-negative integer $|k_0|$ and a sign $\text{sgn}(k_0)$, so that

$$k_0 = \text{sgn}(k_0) \cdot |k_0|. \tag{3.2}$$

Suppose the b least significant bits of the bit plane representation of k_0 are discarded, leaving the non-negative integer $\lfloor |k_0|/2^b \rfloor$ and the sign $\text{sgn}(k_0)$ to represent the signed quantization index

$$k_b = \text{sgn}(k_0) \cdot \lfloor |k_0|/2^b \rfloor, \tag{3.3}$$

where $\lfloor \cdot \rfloor$ denotes the *floor* operation. We define the encoding operation as

$$\text{encode}(x, \Delta_0, b) = k_b \tag{3.4}$$

for $b = 0, 1, \ldots$.

It can be shown that (3.4) is identical to the encoding function of a dead-zone quantizer with step-size $\Delta_b = 2^b \Delta_0$ and dead-zone width either $w_0 = \Delta_0$ for $b = 0$ or $w_b = 2\Delta_b$ for $b = 1, 2, \ldots$. Specifically

$$k_b = \text{sgn}(x) \cdot \max\left(0, \left\lfloor \frac{|x| - w_b/2}{\Delta_b} + 1 \right\rfloor\right). \tag{3.5}$$

The reconstruction $\hat{x}_b$ is given by

$$\hat{x}_b = \text{sgn}(k_b) \cdot \left(\frac{w_b}{2} + \Delta_b \cdot (|k_b| - 1 + r_{k_b})\right), \tag{3.6}$$

where $r_{k_b} \in [0, 1]$ is a reconstruction offset as a fraction of the step-size Δ_b. For given reconstruction offsets $\{r_{k_b}\}$, we define the decoding operation as

$$\text{decode}(k_b, \Delta_0, b) = \hat{x}_b \tag{3.7}$$

for $b = 0, 1, \ldots$.

For quantization index k_b, the optimal fractional reconstruction offset r_{k_b} is the value such that $\hat{x}_b = E[X|\text{encode}(X, \Delta_0, b) = k_b]$. Sullivan [81] shows that when X is Laplacian, the optimal fractional offsets are all identical, and satisfy

$$r_{k_b} = \frac{E[X|0 \le X \le \Delta_b]}{\Delta_b} \tag{3.8}$$

for all $k_b \neq 0$. That is, regardless of k_b, the optimal r_{k_b} is the fractional offset, within any bin of width Δ_b not containing the origin, of the centroid of the bin.

It can be shown that if X is Laplacian with mean absolute deviation (MAD) T, i.e., if $X \sim p(x) = \frac{1}{2T}\exp(-\frac{|x|}{T})$, then the expectation in (3.8) is

$$E[X|0 \le X \le \Delta_b] = \frac{\int_0^{\Delta_b} \frac{x}{T} e^{-\frac{x}{T}} dx}{1 - e^{-\Delta_b/T}} = \frac{T - (T + \Delta_b)e^{-\Delta_b/T}}{1 - e^{-\Delta_b/T}} \stackrel{\Delta}{=} \delta_T(\Delta_b). \tag{3.9}$$

Unfortunately, although all coefficients may be approximately Laplacian, they all have different MADs, and hence it is difficult to estimate T for a single coefficient. However, if we observe that a coefficient X lands in bin k_b, i.e., $X \in [k_b\Delta_b, (k_b + 1)\Delta_b]$, then the maximum likelihood estimate of T is approximately $\hat{T} = (k_b + 0.5)\Delta_b$, for $b = 1, 2, \ldots$. Thus we can estimate the optimal r_{k_b} to be

$$r_{k_b} = \frac{\delta_{\hat{T}}(\Delta_b)}{\Delta_b}. \tag{3.10}$$

For $k_b = 1$, this is 0.444852. For larger k_b, this is closer to 0.5, but can nevertheless be evaluated for each k_b using (3.10). It is certainly simpler, and may be almost as good, to set $r_{k_b} = 0.5$ for all k_b, which is the usual choice for decoding a dead-zone encoding.

A dead-zone quantizer with dead-zone width $w = 2\Delta$ for $b = 1, 2, \ldots$ may be close to optimal, and can be better than a quantizer with strictly uniform thresholds (i.e., $w = \Delta$). According to Sullivan [81], the optimal quantizer for a Laplacian source with MAD T has a reproduction at 0, a first positive threshold at t, a first positive reconstruction at $t + \delta_T(\Delta)$, and subsequent thresholds and reproductions respectively at $t+n\Delta$ and $t+n\Delta+\delta_T(\Delta)$, for $n = 1, 2, 3, \ldots$, where $\delta_T(\Delta)$ is defined in (3.9). Negative thresholds and reconstructions mirror the positive thresholds and reconstructions. Thus $2t$ is the dead-zone width and Δ is the width of all other quantization bins (i.e., the step-size). The optimal value of t is given by an implicit formula, but it satisfies $t = z(\Delta - \delta_T(\Delta))$, where z is the so-called "dead-zone ratio". Sullivan shows that the optimal dead-zone ratio is always between 0.95 and 1. Thus, since $\delta_T(\Delta)$ is between 0 and $\Delta/2$, we must have $t = z(\Delta - \delta_T(\Delta)) < z\Delta < \Delta$, and also $t > z\Delta/2 > 0.95\Delta/2$. It turns out that $t > \Delta/2$. Thus the optimal dead-zone for a Laplacian source, having width $2t$, is always larger than the step-size, and smaller than twice the step-size. If you put extra mass at zero, then the optimal dead-zone width can increase arbitrarily. It is not unreasonable to model coefficients as Laplacian plus extra mass at zero. Thus a dead-zone of width twice the step-size may very well be close to optimal.

3.2 SET PARTITIONING IN HIERARCHICAL TREES WITH REGION-ADAPTIVE HIERARCHICAL TRANSFORM

The original SPIHT algorithm [24] and its follow-on work assumed a particular set partition adapted to wavelet image decomposition (see [82, Ch. 10] and its references). However, it is clear from the description in Section 2.3 that SPIHT is more general and can use any hierarchical set partition of T. Certain hierarchical set partitions result in lower numbers of bits encoded than others. The best hierarchical set partitions are those for which the member sets of a set have highly correlated significance. Specifically, if one member of a set is insignificant, then the other members of

the set are also highly likely to be insignificant. This allows the whole set to be declared insignificant with a single bit.

For RAHT decomposition of point cloud attributes, a natural hierarchical set partition is given by the binary tree that defines the RAHT coefficients. This is a natural grouping because if a coefficient c_t at a node t is insignificant, then it likely means that the signal in the block associated with node t is flat. If the signal is flat, then all the RAHT coefficients below node t are also likely to be insignificant. The same correlation properties may not be shared by other hierarchical set partitions, such as those based on run lengths.

The binary tree that defines the RAHT coefficients is a sparse binary tree of depth $3L$ given by the Morton codes [83] of a set of occupied voxels, where the voxels have L bits of precision. For example, if a voxel geometry has 3 bits of precision as $x_0x_1x_2$, $y_0y_1y_2$ and $z_0z_1z_2$, then its Morton code is a 9-bit word $x_0y_0z_0x_1y_1z_1x_2y_2z_2$. The root is at binary level $\ell = 0$ and the leaves are at binary level $\ell = 3L$. Each leaf corresponds to an occupied voxel. Each interior node at binary level ℓ corresponds to an occupied block of $2^{3L-\ell}$ voxels (at least one of which is occupied). The number of occupied voxels in a block is the weight of the block. The occupied block corresponding to an interior node at level ℓ is split along an axis (x, y, or z) into two sub-blocks, at least one of which is occupied. The occupied sub-blocks of a block correspond to child nodes of the block's node. Thus, each internal node has either 1 or 2 children. A child is designated either a left or right child depending on whether its Morton prefix (the path from the root to the node) is even or odd.

Each internal node t of the tree T corresponds to a RAHT coefficient if it has two children. A node is said to be significant at bit-plane b if its associated RAHT coefficient is significant. Each node t that corresponds to a RAHT coefficient further corresponds to a set T_t of sets in a hierarchical set partition. The set T_t of sets always contains the singleton set $\{t\}$ as a member. However, T_t may also contain one or two additional sets as members. To be specific, let left-descendants (resp. right-descendants) of t denote descendants of t in the branch of T rooted at the left (resp. right) child of t. Let t_L (resp. t_R) denote the left-descendant (resp. right-descendant) corresponding to the RAHT coefficient closest to t. Then T_t contains as member sets T_{t_L} and/or T_{t_R}, if they exist. T_{t_L} and T_{t_R} are similarly defined, recursively. That is,

$$T_t = \begin{cases} \{t\} & \text{if } T_t \text{ is a singleton set} \\ \{\{t\}, T_{t_L}, T_{t_R}\} & \text{if } T_t \text{ is a compound set} \end{cases} . \tag{3.11}$$

With this definition of the hierarchical set partition, the SPIHT algorithm can be applied to RAHT coefficients. To illustrate, Fig. 3.2 shows the state of the SPIHT for RAHT algorithm at the beginning of bit-plane b. In the Figure, a gray node t is one that has only one child and therefore does not have an associated RAHT coefficient. A black node is a node whose associated coefficient has already become significant in a previous bit-plane. An isolated white node is a singleton set whose associated coefficient has not become significant yet. A white node that roots a branch denotes a compound set whose descendants have not yet become significant.

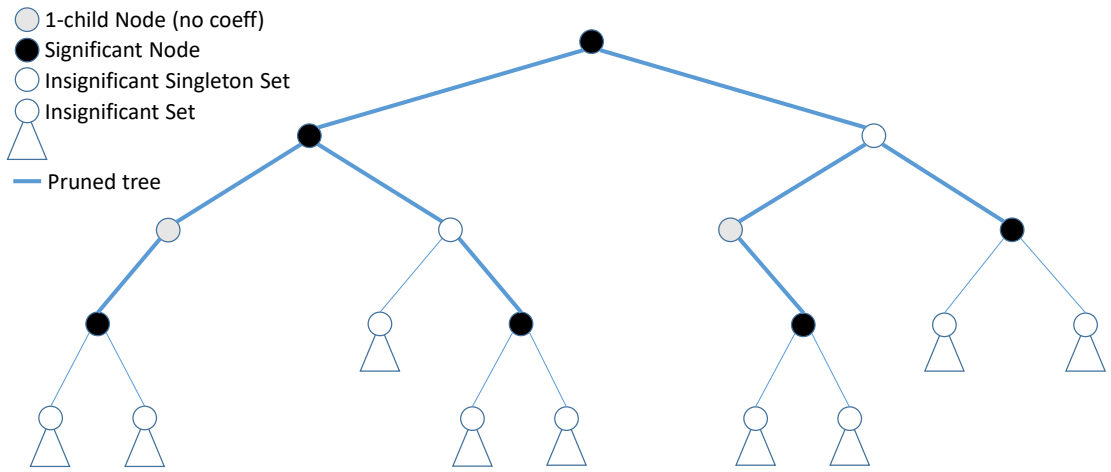


Figure 3.2: Illustration of the SPIHT algorithm for RAHT coefficients.

The lists LSN and LIS are partitioned by level ℓ into segments $LSN(\ell)$ and $LIS(\ell)$, $\ell = 0, \ldots, 3L$. For each bit-plane, the nodes of the tree are visited from level $\ell = 0$ (the root) to $\ell = 3L$ (the leaves). At each level, the nodes are processed, while nodes with no coefficient (1-child nodes) are skipped. Nodes that have already become significant in a previous bit-plane are processed in the refinement pass. Nodes that were not yet significant in a previous bit-plane are processed in the sorting pass. The adapted SPIHT algorithm for RAHT is shown in Algorithms 3 and 4.

3.3 EXPERIMENTAL RESULTS

Simulations were carried out to investigate the performance of SPIHT for RAHT coefficients. In our tests, the first frame of five frontal upper-body sequences from Microsoft Research ("Andrew", "David", "Phil", "Ricardo" and "Sarah") [1] and four full body sequences from 8i Labs ("Longdress", "Loot", "Redandblack" and "Soldier") [2] were used. Figures 3.3 and 3.4 show the projections of each of the point clouds used in our experiments.

Algorithm 3 SPIHT encoder for RAHT

Input: $\Delta, \{c_t : t \in T\}, T_0$

/* Initialization */
for $t \in T$ **do** $k_t = \text{round}(c_t/\Delta)$
for $\ell = 0, \ldots, 3L$ **do** $LSN(\ell) \leftarrow$ empty list ▷ List of Significant Nodes
for $\ell = 1, \ldots, 3L$ **do** $LIS(\ell) \leftarrow$ empty list ▷ List of Insignificant Sets
$LIS(0) \leftarrow$ list of member sets in T_0
$b \leftarrow b_{\max} = \lfloor \log_2(\max_{t \in T} |k_t|) \rfloor$ ▷ index of highest bit-plane
/* Refinement Pass */
for $\ell = 0$ up through $3L - 1$ **do** ▷ level loop
    for each element $t \in LSN$ **do**
        Encode bit b of k_t ▷ refinement bit
    end for
end for
/* Sorting Pass */
for $\ell = 0$ up through $3L - 1$ **do** ▷ level loop
    for each set T' in LIS **do**
        Encode $S_b(T') = \mathbb{1}\{\lfloor \log_2(\max_{t \in T'} |k_t|) \rfloor \geq b\}$ ▷ significance bit
        if $S_b(T') = 1$ **then**
            if T' is a singleton $\{t\}$ **then**
                Encode the sign of k_t ▷ sign bit
                Remove T' from LIS; add t to LSN
            else// T' is a set of sets
                if T' is of type A **then**
                    for each singleton set in $O(T')$ **do**
                        Encode $S_b(O(T')) = \mathbb{1}\{\lfloor \log_2(\max_{t \in T'} |k_t|) \rfloor \geq b\}$
                        if $S_b(O(T')) = 1$ **then**
                            Encode the sign of k_t
                            Add t to LSN
                        end if
                    end for
                    if $\mathcal{L}(T')$ is not empty **then**
                        Move T' to the end of the LIS as a type B
                        Go to Step 2
                    else
                        Delete T' from LIS
                    end if
                else//T' is of type B ▷ Step 2
                    Encode $S_b(\mathcal{L}(T')) = \mathbb{1}\{\lfloor \log_2(\max_{t \in T'} |k_t|) \rfloor \geq b\}$
                    Add member sets in T' to LIS as type A
                    Delete T' from LIS
                end if
            end if
        end if
    end for
end for
/* Iterate */
$b \leftarrow b - 1$
if $b > 0$ **then** go back to Refinement Pass
end if

Output: bit stream

Algorithm 4 SPIHT decoder for RAHT

Input: bit stream, $b_{\max}$, T_0 ▷ tree structure as in the encoder

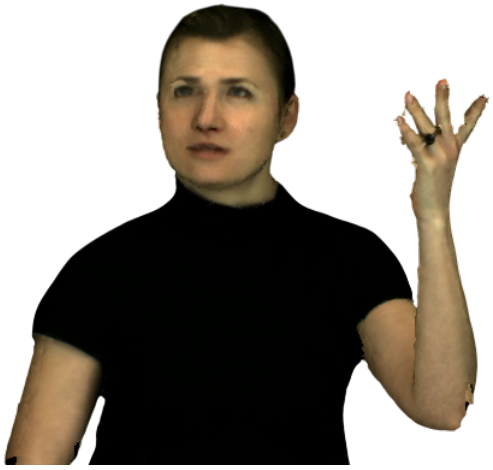
/* Initialization */
for all $t \in T$ **do** $\hat{k}_t \leftarrow 0$
for $\ell = 0, \ldots, 3L$ **do** $LSN(\ell) \leftarrow$ empty list ▷ List of Significant Nodes
for $\ell = 1, \ldots, 3L$ **do** $LIS(\ell) \leftarrow$ empty list ▷ List of Insignificant Sets
$LIS(0) \leftarrow$ list of member sets in T_0
$b \leftarrow b_{\max}$ ▷ index of highest bit-plane
/* Refinement Pass */
for $\ell = 0$ up through $3L-1$ **do** ▷ level loop
    for each element $t \in LSN$ **do**
        Decode refinement bit $r \in \{0,1\}$ for bit-plane b of $\hat{k}_t$
        if $r = 1$ **then** set $|\hat{k}_t| \leftarrow |\hat{k}_t| + 2^b$ with sign $\text{sign}(\hat{k}_t)$
        end if
    end for
end for
/* Sorting Pass */
for $\ell = 0$ up through $3L-1$ **do** ▷ level loop
    for each set T' in LIS **do**
        Decode $S_b(T') \in \{0,1\}$ ▷ significance bit
        if $S_b(T') = 1$ **then**
            if T' is a singleton $\{t\}$ **then**
                Decode the sign of $\hat{k}_t$ ▷ sign bit
                Set $|\hat{k}_t| \leftarrow |\hat{k}_t| + 2^b$, add t to LSN and Remove T' from LIS
            else// T' is a set of sets
                if T' is of type A **then**
                    for each singleton set $\{t\}$ in $O(T')$ **do**
                        Decode $S_b(\{t\}) \in \{0,1\}$ ▷ significance of each offspring
                        if $S_b(\{t\}) = 1$ **then**
                            Decode the sign of $\hat{k}_t$ and Set $|\hat{k}_t| \leftarrow |\hat{k}_t| + 2^b$ and add t to LSN
                        end if
                    end for
                    if $\mathcal{L}(T')$ is not empty **then**
                        Move T' to the end of the LIS as a type B and Go to Step 2
                    else
                        Delete T' from LIS
                    end if
                else// T' is of type B ▷ Step 2
                    Decode $S_b(\mathcal{L}(T')) \in \{0,1\}$
                    if $S_b(\mathcal{L}(T')) = 1$ **then**
                        Add member sets in T' to LIS as type A
                        Delete T' from LIS
                    end if
                end if
            end if
        end if
    end for
end for
/* Iterate */
$b \leftarrow b - 1$
if $b \geq 0$ **then** go back to Refinement Pass
end if

Output: reconstructed integers $\{\hat{k}_t : t \in T\}$

(a) Projection of a frame of point cloud Andrew.

(b) Projection of a frame of point cloud David.

(c) Projection of a frame of point cloud Ricardo.



(d) Projection of a frame of point cloud Sarah.

(e) Projection of a frame of point cloud Phil.

Figure 3.3: Projections of the point clouds used in our tests from upper-body sequences from Microsoft Research [1].

(a) Projection of a frame of point cloud Soldier.

(b) Projection of a frame of point cloud Loot.

(c) Projection of a frame of point cloud Redandblack.

(d) Projection of a frame of point cloud Longdress.

Figure 3.4: Projections of the point clouds used in our tests from full body sequences from 8i Labs [2].

Table 3.1: Average BD PSNR-Y gains and average bit-rate savings of SPIHT relative to RLGR.

Point Cloud	PSNR-Y [dB]	bit-rate [%]
Andrew	0.23	-4.93
David	0.56	-14.69
Ricardo	1.01	-21.72
Sarah	0.70	-14.70
Phil	0.12	-2.63
Soldier	0.21	-5.61
Loot	0.34	-10.47
Redandblack	0.41	-11.66
Longdress	0.30	-6.93
Average	**0.43**	**-10.37**

Figure 3.5: Illustration of projections of the point cloud sequence "Ricardo" decoded using SPIHT, and cutting the bit stream at different bit-planes. From left to right: original (14 bit-planes), 11 bit-planes, 10 bit-planes, 9 bit-planes, and 8 bit-planes.

In Table 3.1, Bjontegaard Delta (BD) [84] PSNR-Y gains and bit-rate savings of SPIHT relative to RLGR for each tested sequence are presented. From Table 3.1, we can see that the use of SPIHT with RAHT can outperform the use of RLGR with RAHT for all tested sequences. The best performances in terms of bit-rate savings and in terms of PSNR-Y are 21.72% and 1.01 dB for "Ricardo". The worst performances in terms of bit-rate savings and in terms of PSNR-Y are 2.63% and 0.12 dB for "Phil". The average bit-rate savings is 10.37% and the average PSNR-Y gains is 0.43 dB.

Figure 3.6 plots the rate-distortion performance of SPIHT relative to RLGR for the first frame of point cloud "Ricardo". The figure also includes the rate-distortion performance of G-PCC with RAHT to give the reader an idea of how much gain G-PCC with RAHT offers over its baseline, RLGR with RAHT, as well as over SPIHT with RAHT. However, one should bear in mind that unlike SPIHT, the G-PCC bit stream is not embedded. The embedded functionality of SPIHT is very desirable, as illustrated in Fig. 3.5, where projections of sequence "Ricardo" are shown for PCs decoded from a single bit stream truncated at different bit-planes.

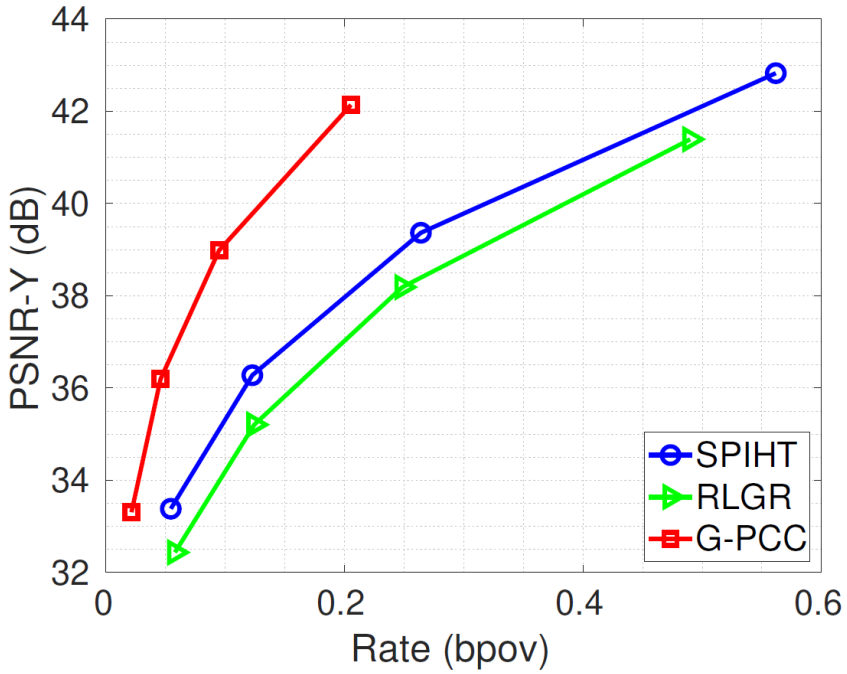


Figure 3.6: Rate-distortion comparison among SPIHT, RLGR and G-PCC for the first frame of PC "Ricardo".

The reasons that G-PCC with RAHT has performance gains over RLGR with RAHT were analyzed [17]. The gains come primarily from the possibility of a transform domain prediction of RAHT [85], [86]. In this approach, the RAHT octree is traversed in a reversed, root-to-leaf (coarse-to-fine) order. A subsequent attribute inter-depth upsampling step is introduced to obtain a local prediction at each level. This prediction is applied to the AC coefficients, so that only the DC terms and the corresponding prediction residual coefficients must be encoded. The attribute prediction itself is computed as a weighted average of neighboring nodes. With this prediction-based formulation of RAHT, reported bitrate reductions reach up to approximately 30% relative to RAHT without prediction [17]. Unfortunately for G-PCC, however, such predictive coding is problematic for the construction of embedded bit streams. Another important mechanism is arithmetic coding with context modeling, including special symbols for jointly coding YUV coefficients that are all zero.

3.4 CONCLUSIONS

This chapter presented an embedded coder for point cloud attributes. It was proposed to encode RAHT coefficients with SPIHT and to use as hierarchical set partitions the binary tree that defines the coefficients. The experimental results show that the use of SPIHT for RAHT can outperform the RLGR-based RAHT, while providing a fully embedded functionality to the codec. This, without use of any entropy coding apart from bit-plane and set partitions. As far as we know, this was the first embedded PC coder. In the next chapter will be presented the further research developed to the use of context modeling for a context-based arithmetic-coded SPIHT.

4 CONTEXT MODELING FOR ARITHMETIC CODING ASSOCIATED WITH SPIHT FOR RAHT

"Science is a way of thinking
much more than it is a body of knowledge."
- Carl Sagan

Arithmetic coding (AC) [87], [88] attains high compression when driven by accurate next symbol probabilities conditioned on past symbols. In practice, binary AC is prevalent—even for m-ary sources—by binarizing symbols and estimating conditional probabilities via frequency counting in look-up tables (LUTs). This LUT paradigm underlies context-adaptive binary arithmetic coding (CABAC) in major image/video standards (JBIG, JBIG-2, JPEG/JPEG2000, H.264/AVC, H.265/HEVC, H.266/VVC), but becomes impractical as context dimensionality grows, due to memory blow-up and context dilution, i.e. many context patterns remain sparsely observed.

Since the G-PCC still outperforms the proposed embedded method, one of the primary goals of this chapter is to present a coding technique that can help close the gap between the two encoders. In this chapter, we present improvement on the developed embedded attribute encoding method for point clouds presented in Chapter 3. We begin by proposing a context-based arithmetic coder for the final bit stream, the context can be based on previously encoded geometry information. The primary idea is to use a non-neural arithmetic coder, but the overhead cost was too high to be compensated by the final gains. We then propose to use a multi-layer perceptron (MLP) to estimate conditional probabilities, in order to drive an arithmetic coder to further compress the embedded data. The result is an encoder that is efficient, scalable, and, best of all, embedded. That is, higher compression is achieved by further trimming the single bit stream.

SPIHT has a different hierarchy of sets when used for RAHT point cloud attribute coefficients compared to when used for wavelet image coefficients, even though the algorithm is the same. For RAHT coefficients, the hierarchy of sets is determined by the tree in which the RAHT coefficients are naturally organized. This tree only depends on the geometry, as described in Chapter 3. The important point to note is that each significance, sign, and refinement bit is associated with some coefficient c_t, which is located at a node t in the tree. Thus, each bit in the SPIHT bit stream may be further compressed with an arithmetic coder (AC) [87] using a context model for node t in the tree.

4.1 CONTEXT MODELING FOR ARITHMETIC CODING IMPROVEMENT

In an attempt to close the gap between our method and the G-PCC coder, we experimented with using arithmetic coding to compress SPIHT's significance, sign, and refinement bits using a logistic probability model driven by previous context. In this section, we present a first context modeling method. However, the side information needed to be conveyed to the encoder was not

worth the final gain in bit-rate. In section 4.2, we describe a second method, which was successfully implemented and has shown considerable gains over the traditional SPIHT for RAHT.

4.1.1 STATISTICS FOR PREDICTING SIGNIFICANCE

Consider the high-pass coefficient c_t at node t of the sparse binary tree. Let t' be the parent of t, let t'' be the grandparent of t, and let t''' be the great grandparent of t. Let $|c_t|$, $|c_t'|$, $|c_t''|$, and $|c_t'''|$ be the magnitudes of the coefficients at those nodes. Let w_{0t}, $w_{0t'}$, $w_{0t''}$, and $w_{0t'''}$ be the weights of the left children of those nodes, and let w_{1t}, $w_{1t'}$, $w_{1t''}$, and $w_{1t'''}$ be the weights of the right children of those nodes. Denote by $w_{ct'}$, $w_{ct''}$, and $w_{ct'''}$ the weights of the children of nodes t', t'', and t''' on the path from the root to node t, and denote by $w_{\bar{c}t'}$, $w_{\bar{c}t''}$, and $w_{\bar{c}t'''}$ the weights of the children of nodes t', t'', and t''' *not* on the path from the root to node t.

Consider the 15 quantities $\log_2(|c_{t'}|)$, $\log_2(|c_{t''}|)$, $\log_2(|c_{t'''}|)$, $\log_2(w_{0t})$, $\log_2(w_{1t})$, $\log_2(w_{0t} + w_{1t})$, $\log_2(w_{ct'})$, $\log_2(w_{\bar{c}t'})$, $\log_2(w_{0t'} + w_{1t'})$, $\log_2(w_{ct''})$, $\log_2(w_{\bar{c}t''})$, $\log_2(w_{0t''} + w_{1t''})$, $\log_2(w_{ct'''})$, $\log_2(w_{\bar{c}t'''})$, and $\log_2(w_{0t'''} + w_{1t'''})$. We hope to observe a linear relationship between $\log_2(|c_t|)$ and some of these quantities.

To quantify these linear relationships, it is performed a regression analysis to find the best linear combination of the above quantities to predict $\log_2(|c_t|)$. To be more specific, model

$$\begin{aligned} \boldsymbol{y} &= [\log_2(|c_t|)] = \begin{bmatrix} 1 & \log_2(|c_{t'}|) & \cdots & \log_2(w_{0t'''} + w_{1t'''}) \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{15} \end{bmatrix} + \boldsymbol{e} \\ &= \boldsymbol{A}\boldsymbol{x} + \boldsymbol{e}, \end{aligned} \tag{4.1}$$

where $\boldsymbol{y}$ is the vector of log-magnitudes for all N nodes t having two children and whose parent, grandparent, and great grandparent each have two children; $\boldsymbol{A}$ is the $N \times 16$ matrix of 16 basis functions evaluated on those nodes t; $\boldsymbol{x}$ is the vector of coefficients of those 16 basis functions; and $\boldsymbol{e}$ is an error vector of length N. The coefficients $\boldsymbol{x}$ that minimize $||\boldsymbol{e}||$ are those for which the error vector $\boldsymbol{e} = \boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}$ is perpendicular to the span of the basis vectors, i.e.

$$e = \boldsymbol{A}^T \boldsymbol{y} - \boldsymbol{A}^T \boldsymbol{A}\boldsymbol{x}, \tag{4.2}$$

or

$$\boldsymbol{x} = (\boldsymbol{A}^T \boldsymbol{A})^{-1} \boldsymbol{A}^T \boldsymbol{y}. \tag{4.3}$$

We can examine the coefficients of $\boldsymbol{x}$ to see which basis functions matter most to predicting $\boldsymbol{y}$. This can also be a way to predict the magnitude of $\log_2(|c_t|)$ from its neighboring quantities.

We can also regress $\log(|c_t^{\max}|)$ on the same quantities, where $c_t^{\max}$ is the coefficient in the set of coefficients rooted at t whose magnitude is maximal, as

$$\boldsymbol{y}^{\max} = \boldsymbol{A}\boldsymbol{x}^{\max} + \boldsymbol{e}^{\max}. \tag{4.4}$$

The regression coefficients $\boldsymbol{x}^{\max}$ can then be used to predict $\log(|c_t^{\max}|)$, and thus predict whether the set is significant or not. Modeling the prediction error as Gaussian distributed with variance $\sigma^2 = ||\boldsymbol{e}^{\max}||^2$, we can produce the probability that the set is significant at a given level, $p = P\{\log(|c_t^{\max}|) \geq n\}$. This probability can be used to drive an entropy coder.

4.1.2 CONTEXT MODELING

As described previously, the SPIHT algorithm encodes all coefficients, bit-plane by bit-plane, into a sequence of bits. Each bit in this encoding stream represents either a refinement bit of a coefficient that had already become significant in a previous bit-plane, a significance bit of a set that was being tested for significance in the current bit-plane, or the sign bit of a coefficient that becomes newly significant in the current bit-plane. However, these bits need not be directly inserted into the bit stream. In order, to improve compression efficiency, they may be further encoded using binary arithmetic entropy coding with a context model. We hypothesize that this context model can largely make up the gain due to the prediction in current G-PCC TMC13 standard with predictive RAHT and the joint coding of YUV coefficients that are all zero. This section describes how such context modeling may be performed.

As we know, the SPIHT for RAHT algorithm iterates over bit-planes, from the most significant bit-plane $b = b_{\max}$ to the least significant bit-plane $b = 0$. Within each bit-plane, the algorithm processes the coefficients from level $\ell = 0$ (the root) to level $\ell = 3L - 1$ (the parents of the leaves). Finally, within each level, the algorithm processes the nodes in Morton order from low to high. Thus, the bits encoded for node t in level ℓ in bit-plane b may use as context any of the bits encoded for any of the nodes t' in levels ℓ' in bit-planes b', where either $b' > b$, or $b' = b$ and $\ell' < \ell$, or $b' = b$ and $\ell' = \ell$ and $t' < t$ in Morton order.

Normally one arrives at the tree at the beginning of bit-plane b by going through the iterations $b' = b_{\max}$ to $b' = b + 1$. However, if one only wishes to output the bits for bit-plane b, the state of the SPIHT for RAHT encoder at the beginning of bit-plane b can be constructed from scratch, only knowing the coefficients $\{c_t : t \in T\}$, by performing the following:

1. Find all significant nodes;
2. Label all nodes on the paths from the root to the significant nodes;
3. Call this the pruned tree;
4. The nodes in the pruned tree that have two children and are not significant nodes are insignificant singleton sets;
5. The branches rooted in nodes directly attached to the pruned tree are insignificant (generally compound) sets.

Within this bit-plane b, the nodes are visited in levels of the binary tree, from the root $\ell = 0$ to the leaves $\ell = 3L - 1$. Further, within each level of the binary tree, the nodes t are visited from

lowest Morton code to highest Morton code. If the node has only one child (and therefore has no coefficient) it is skipped. If the node is not in the pruned tree, or is not among any of the children of the leaves of the pruned tree, they are also skipped. If a node is significant (i.e., has an associated coefficient that became significant at a bit-plane higher than b), then encode its refinement bit in bit-plane b. If a node is an insignificant singleton (i.e., is isolated and has an associated coefficient that has not yet become significant), then test its coefficient for significance, and encode the indicator bit. If the indicator bit is positive, then, in addition, encode the sign of the newly significant coefficient. If the node is the root of an insignificant set (i.e., is the child of a leaf of the pruned tree), then test its branch for significance, and encode the indicator bit. If the indicator bit is positive, then in addition, expand the pruned tree to include the node and test the node for significance. Bits are encoded in at least four different contexts, when:

1. A refinement bit is encoded for each significant node;
2. A significance bit is encoded for each insignificant singleton set;
3. A significance bit is encoded for each insignificant compound set;
4. A sign bit is encoded for each newly significant node.

Thus, the simplest scheme that takes context into account would use a different probability q_1 to encode each bit using an arithmetic coder depending on its context $c = 1, 2, 3, 4$. These probabilities could be adapted or learned. A simple adaptive scheme, to encode a bit with probability $q_1(c, t, \ell, b)$ in context c at node t in level ℓ in bit-plane b, would be to count the number of times $n_0(c, t, \ell, b)$ (resp. $n_1(c, t, \ell, b)$) that a 0 (1) bit was encoded in context c at a node $t' < t$ in level ℓ in bit-plane b, and set

$$q_1(c, t, \ell, b) = \frac{n_1(c, t, \ell, b) + 1}{n_0(c, t, \ell, b) + n_1(c, t, \ell, b) + 2}. \tag{4.5}$$

A simple non-adaptive, but learned context would be to count the number of times $n_0(c, \ell, b)$ ($n_1(c, \ell, b)$) that a 0 (1) bit was encoded in context c at a node in level ℓ in bit-plane b in a large training set, and use

$$q_1(c, \ell, b) = \frac{n_1(c, \ell, b)}{n_0(c, \ell, b) + n_1(c, \ell, b)} \tag{4.6}$$

to encode all bits in context c for all nodes t in level ℓ in bit-plane b.

However, we can do more. We can break each of these four contexts into 27 sub-contexts based on whether, for a given node t, the parent t', grandparent t'', and great-grandparent t''' are either a 1-child node, a significant node, or an insignificant node. For nodes t without all three ancestors, we can assume that the missing ancestors are 1-child nodes. Then, we can use the same adaptive or learned scheme, as before, for each sub-context.

We can also use continuous contexts. Let's say that a node t in level ℓ in bit-plane b represents an insignificant compound set (about to encode a significance bit), and it has a parent t' that is

a significant node, a grandparent t'' that is a 1-child node, and a great-grandparent t''' that is an insignificant node. This is the case with one of the insignificant sets in Figure 3.2. Then, in that sub-context, the parent t' has a coefficient known to $b_{\max} - b + 1$ bits of precision, namely $\hat{k}' = \hat{k}_{t'} = 2^b \lfloor k_{t'}/2^b \rfloor$. The grandparent t'' and great-grandparent t''' can be considered as having coefficients $\hat{k}'' = \hat{k}_{t''} = 0$ and $\hat{k}''' = \hat{k}_{t'''} = 0$. These coefficients can be used in this context c and sub-context s to train a logistic regression model,

$$q_1(c, s, \ell, b, \hat{k}', \hat{k}'', \hat{k}''') = \sigma(a + a' \log_2(|\hat{k}'|) + a'' \log_2(|\hat{k}''|) + a''' \log_2(|\hat{k}'''|)), \tag{4.7}$$

where

$$\sigma(x) = \frac{1}{(1 + e^{-x})} \tag{4.8}$$

is the sigmoid function, and its argument is an affine combination of the logarithms of the known coefficients of the ancestors of t. Logarithms of the weights w_{0t}, w_{1t}, $w_{ct'}$, $w_{\bar{c}t'}$, $w_{ct''}$, $w_{\bar{c}t''}$, $w_{ct'''}$, and $w_{\bar{c}t'''}$ could also be included in the affine combination for possibly better performance, as discussed in section 4.1.1.

Finally, we can include predictions in the context. Suppose that we are about to encode a bit related to node t in level ℓ in bit-plane b. The bit could be a refinement bit, a significance bit, or a sign bit. In any of these contexts, if the value of k_t can be predicted, then the entropy of the bit can be reduced. In predictive RAHT [86], there is a mechanism for predicting the RAHT coefficient at a node t in level ℓ by predicting the average (DC) value of each child block of node t, and computing the RAHT coefficient at node t from these predicted DC values. The predicted DC values are based on the reconstructed DC values of the blocks in level ℓ (which contains t), which in turn are reconstructed from RAHT coefficients at levels 0 through $\ell - 1$. Thus the predictions can be based on information in bit-planes $b_{\max}$ through b. We can include this prediction in the affine combination in (4.7), which may go a long way towards closing the gap with predictive RAHT, in a bit-plane coder.

So far we have been discussing RAHT coefficients as if they are the coefficients of a scalar signal. However, as we know, the point cloud attributes are often vector attributes. Typically, they include at least Y, U, and V color channels. Often, a block is near-uniform in color, meaning that the RAHT coefficients for all three YUV components are near-zero simultaneously. Thus, these coefficients are correlated. One way to deal with their correlation is to construct the hierarchical set partition, so that it would be easy to signal that they are all insignificant together. However, another way would be to include the values of the RAHT coefficients of Y, U, and V — reconstructed up through bit-plane $b - 1$ — in the affine combination in (4.7). This should mostly meet the performance gain in G-PCC due to joint entropy coding of runs of zero YUV coefficients.

4.1.3 TRAINING THE COEFFICIENTS

In this section, we discuss how to train the coefficients a, a', a'', and a''' of the functions 1, $\log_2(|\hat{k}'|)$, $\log_2(|\hat{k}''|)$, and $\log_2(|\hat{k}'''|)$, as in (4.7). It is straightforward to extend the method to

regression on additional quantities such as the weights, predictions and other color components. These coefficients will depend on b, ℓ, c, s.

Let us say we are given a dataset of point clouds. For a given bit-plane b, level ℓ, context c, and sub-context s, let's collect the following training set of $N = n_0 + n_1$ examples:

$$\begin{bmatrix} y_1 & 1 & \log_2(|\hat{k}'_1|) & \log_2(|\hat{k}''_1|) & \log_2(|\hat{k}'''_1|) \\ y_2 & 1 & \log_2(|\hat{k}'_2|) & \log_2(|\hat{k}''_2|) & \log_2(|\hat{k}'''_2|) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y_N & 1 & \log_2(|\hat{k}'_N|) & \log_2(|\hat{k}''_N|) & \log_2(|\hat{k}'''_N|). \end{bmatrix} \tag{4.9}$$

Here, $y_i \in \{0, 1\}$ is the value of the binary symbol being coded in example $i = 1, \ldots, N$. In the training set for bit-plane b, level ℓ, context c, and sub-context s, there will typically be one example for each node t. Thus, using our previous notation, there will be n_0 examples where $y_i = 0$ and n_1 examples where $y_i = 1$.

After the values of a, a', a'', and a''' are trained on this training set, whenever we are in bit-plane b, level ℓ, context c, and sub-context s, and need to encode a binary symbol y at a node t whose parent, grandparent, and great-grandparent coefficients (at bit-planes b and higher) are $\hat{k}'$, $\hat{k}''$, and $\hat{k}'''$, we will be able to use (4.7) to produce a value

$$q_1 = q_1(c, s, \ell, b, \hat{k}', \hat{k}'', \hat{k}''') = \sigma(x), \tag{4.10}$$

where

$$x = a + a' \log_2(|\hat{k}'|) + a'' \log_2(|\hat{k}''|) + a''' \log_2(|\hat{k}'''|). \tag{4.11}$$

The number of bits we will need to encode the binary symbol y using an arithmetic coder is $-\log_2 q_1$ if the binary symbol is $y = 1$, and $-\log_2(1 - q_1)$ if the binary symbol is $y = 0$. Therefore, if we were to encode all of the examples in the training set, the total number of bits would be

$$R(\mathbf{a}) = \sum_{i=1}^{N} \left[-y_i \log_2 \sigma(x_i) - (1 - y_i) \log_2(1 - \sigma(x_i)) \right], \tag{4.12}$$

where

$$x_i = a + a' \log_2(|\hat{k}'_i|) + a'' \log_2(|\hat{k}''_i|) + a''' \log_2(|\hat{k}'''_i|) \tag{4.13}$$

and $\mathbf{a}$ is the vector of coefficients $\mathbf{a} = [a, a', a'', a''']$. The optimal value of $\mathbf{a}$ on this training set is the one that minimizes $R(\mathbf{a})$. To find the optimal value $\mathbf{a}^*$, we start at an initial value $\mathbf{a} = \mathbf{a}^{(0)}$ ($\mathbf{a}^{(0)} = \mathbf{0}$ may be sufficient) and perform gradient descent:

$$\mathbf{a}^{(t+1)} = \mathbf{a}^{(t)} - \mu \nabla_{\mathbf{a}} R(\mathbf{a}^{(t)}), \tag{4.14}$$

for $t = 0, 1, \ldots$, until $\mathbf{a}^{(t)}$ converges to $\mathbf{a}^*$, where μ is the step-size (here 0.0001) and

$$\nabla_{\mathbf{a}} R(\mathbf{a}^{(t)}) = \begin{bmatrix} \frac{\partial R}{\partial a}(\mathbf{a}^{(t)}) \\ \frac{\partial R}{\partial a'}(\mathbf{a}^{(t)}) \\ \frac{\partial R}{\partial a''}(\mathbf{a}^{(t)}) \\ \frac{\partial R}{\partial a'''}(\mathbf{a}^{(t)}) \end{bmatrix} \tag{4.15}$$

is the gradient of R with respect to $\mathbf{a}$ evaluated at the point $\mathbf{a}^{(t)}$.

Calculating the gradient is simple. From the definition $\sigma(x) = 1/(1+e^{-x})$, we get $1-\sigma(x) = e^{-x}/(1+e^{-x})$, and therefore

$$\sigma'(x) = \frac{d}{dx}\sigma(x) = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} = \sigma(x)(1-\sigma(x)). \tag{4.16}$$

From this, we get

$$\frac{d}{dx}\ln\sigma(x) = \frac{\sigma'(x)}{\sigma(x)} = 1-\sigma(x), \tag{4.17}$$

$$\frac{d}{dx}\ln(1-\sigma(x)) = \frac{-\sigma'(x)}{1-\sigma(x)} = -\sigma(x). \tag{4.18}$$

Hence if $x_i^{(t)} = a^{(t)} + a'^{(t)}\log_2(|\hat{k}_i'|) + a''^{(t)}\log_2(|\hat{k}_i''|) + a'''^{(t)}\log_2(|\hat{k}_i'''|)$, then

$$\frac{\partial R}{\partial a}(\mathbf{a}^{(t)}) = \frac{1}{\ln 2}\sum_{i=1}^{N}\left[-y_i(1-\sigma(x_i^{(t)})) + (1-y_i)\sigma(x_i^{(t)})\right] \tag{4.19}$$

$$\frac{\partial R}{\partial a'}(\mathbf{a}^{(t)}) = \frac{1}{\ln 2}\sum_{i=1}^{N}\left[-y_i(1-\sigma(x_i^{(t)})) + (1-y_i)\sigma(x_i^{(t)})\right]\log_2(|\hat{k}_i'|) \tag{4.20}$$

$$\frac{\partial R}{\partial a''}(\mathbf{a}^{(t)}) = \frac{1}{\ln 2}\sum_{i=1}^{N}\left[-y_i(1-\sigma(x_i^{(t)})) + (1-y_i)\sigma(x_i^{(t)})\right]\log_2(|\hat{k}_i''|) \tag{4.21}$$

$$\frac{\partial R}{\partial a'''}(\mathbf{a}^{(t)}) = \frac{1}{\ln 2}\sum_{i=1}^{N}\left[-y_i(1-\sigma(x_i^{(t)})) + (1-y_i)\sigma(x_i^{(t)})\right]\log_2(|\hat{k}_i'''|). \tag{4.22}$$

As a sanity check to see whether this is working, we can set $a' = a'' = a''' = 0$ and only train a. Then a should converge to a^* such that $\sigma(a^*) = p_1$ where

$$p_1 = \frac{n_1}{(n_0+n_1)} = \frac{n_1}{N} \tag{4.23}$$

is the empirical distribution. The reason is that in this case, (4.12) becomes

$$R(a) = \sum_{i=1}^{N}[-y_i\log_2\sigma(a) - (1-y_i)\log_2(1-\sigma(a))] \tag{4.24}$$

$$= -n_1\log_2\sigma(a) - n_0\log_2(1-\sigma(a)) \tag{4.25}$$

$$= N\left[-\frac{n_1}{N}\log_2\sigma(a) - \frac{n_0}{N}\log_2(1-\sigma(a))\right] \tag{4.26}$$

$$= N\left[-p_1\log_2\sigma(a) - p_0\log_2(1-\sigma(a))\right] \tag{4.27}$$

$$= N\left[-p_1\log_2 p_1 - p_0\log_2 p_0 + p_1\log_2\frac{p_1}{q_1} + p_0\log_2\frac{p_0}{q_0}\right] \tag{4.28}$$

$$= N\left[H(p) + D(p||q)\right], \tag{4.29}$$

where $q_1 = \sigma(a)$, $q_0 = 1-\sigma(a)$, $H(p)$ is the empirical entropy, and $D(p||q)$ is the Kullback-Leibler divergence [89]. It is well-known that $H(p)$ is the minimum possible number of bits on average, and $D(p||q) \geq 0$, with equality if and only if $q = p$. Thus to minimize $R(a)$, a must converge to a^* where $\sigma(a^*) = p_1$, at which point $q = p$, $D(p||q) = 0$, and $R(a^*) = NH(p)$.

4.1.4 THEORETICAL GAINS WITH LOGISTIC REGRESSION

In order to improve the compression rate of the first implementation of SPIHT for RAHT, we experimented with using arithmetic coding to compress SPIHT's significance, sign, and refinement bits using a logistic probability model driven by previous context, as described in section 4.1. Based on 4.7, we used a function of the form

$$p = \sigma(a + a' \log|\hat{k}'| + a'' \log|\hat{k}''| + a''' \log|\hat{k}'''|) \tag{4.30}$$

to predict the probability p that the set rooted at a node is newly significant, given the compressed coefficients $\hat{k}'$, $\hat{k}''$, and $\hat{k}'''$ of its parent, grandparent, and great-grandparent. This approach significantly reduced the rate needed to code SPIHT's significance, sign, and refinement bits, as shown in Figures 4.1 and 4.2. Unfortunately, the side information needed to represent the model parameters a, a', a'', a''' did not provide a net benefit for this approach.

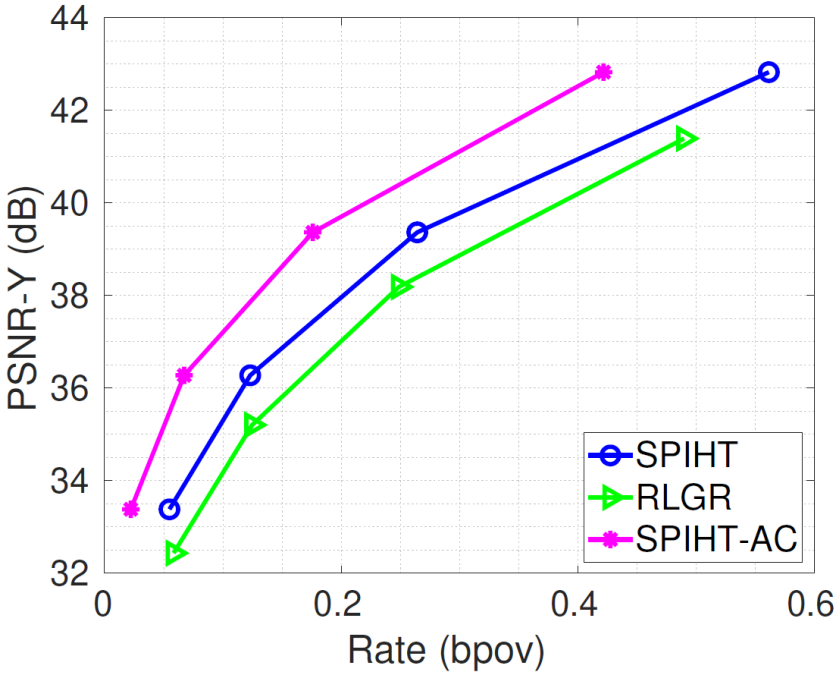


Figure 4.1: Rate-distortion comparison among SPIHT, RLGR and SPIHT with context modeling (SPIHT-AC) for the first frame of PC "Ricardo".

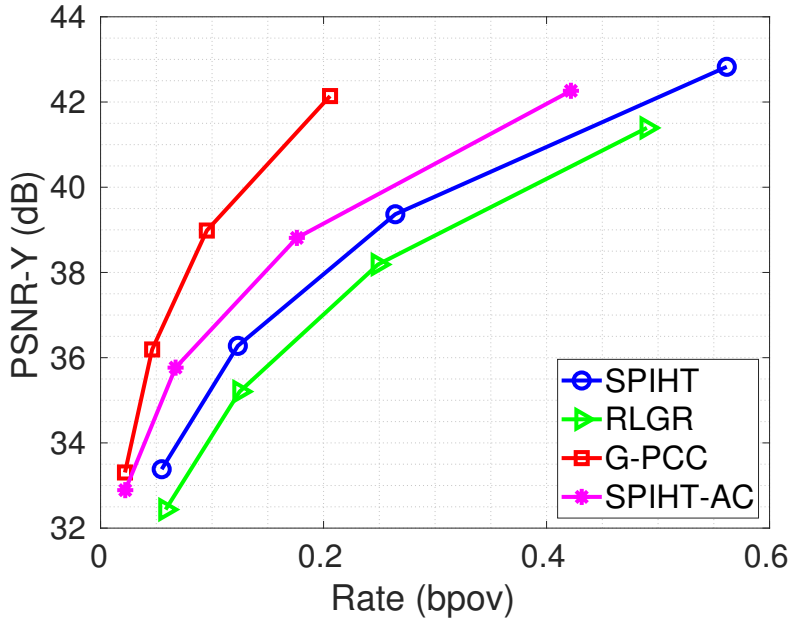


Figure 4.2: Rate-distortion comparison among SPIHT, RLGR, G-PCC and SPIHT with context modeling (SPIHT-AC) for the first frame of PC "Ricardo".

Following on this chapter, we discuss how the use of a multi-layer perceptron helped closing the gap between embedded SPIHT entropy coding and the non-embedded entropy coding used in G-PCC. Out of the scope of this work, but as a suggestion for the readers, another possible direction might be to replace the prediction gain in G-PCC with the transform gain from higher-order RAHT [90], for which it would be more straightforward to perform bit-plane coding.

4.2 PROPOSED CONTEXT MODELING

A compact alternative is to replace LUTs with a small multi-layer perceptron (MLP) that maps a real-valued context vector $\mathbf{X}$ to a Bernoulli parameter $q = f_\theta(\mathbf{X}) \in (0,1)$ via a sigmoid output [91]. This learned estimator generalizes across similar contexts, alleviates sparsity, and removes the need to predefine large discrete state spaces.

Training aligns exactly with compression: the MLP is optimized to minimize the binary cross-entropy

$$R = -\sum_k \left[y_k \log_2 q_k + (1 - y_k) \log_2 (1 - q_k) \right], \qquad q_k = f_\theta(\mathbf{X}_k), \tag{4.31}$$

which equals the expected number of bits spent by an ideal AC on the same sequence. Consequently, gradient-based updates (stochastic or mini-batch) directly reduce rate. The framework naturally extends to m-ary sources by binarization or by predicting distribution parameters for vector-symbol coders. Causality is enforced by constructing $\mathbf{X}$ from previously decoded features only (e.g., bit-plane order, tree level, local occupancy masks, reconstructed neighbors), permitting

seamless use in embedded and progressive pipelines.

From a systems perspective, the MLP is a drop-in replacement for LUTs in context-adaptive binary arithmetic coding: it outputs calibrated probabilities per coded bit, requires orders of magnitude less memory than large LUTs, and remains robust when contexts are high-dimensional or continuous. Computationally, inference scales with the number of network parameters and can be budgeted via shallow architectures, feature selection, and update throttling (e.g., periodic rather than per-symbol backpropagation). In practice, MLP-based context models improve probability calibration and rate–distortion performance in regimes where LUTs underperform due to sparsity, while preserving compatibility with standard arithmetic coding engines and embedded decoding constraints.

As seen in Chapter 3, the SPIHT algorithm iterates over bit-planes, from the most significant bit-plane $b = b_{\max}$ to the least significant bit-plane $b = 0$. Then, within each bit-plane, the coefficients c_t are processed from level $\ell = 0$ (the root) to level $\ell = 3L - 1$ (the parents of the leaves). Finally, within each level, the algorithm processes the nodes t in Morton order from low to high [92]. Thus, the bits encoded at node t, level ℓ, and bit-plane b may use as context any of the bits encoded for any of the nodes $t*$ at levels $\ell*$ and bit-planes $b*$ such that $b* > b$, or $b* = b$ and $\ell* < \ell$, or $b* = b$ and $\ell* = \ell$ and $t* < t$, i.e., all previously processed node bits that are available to the decoder can be used as context.

Let y denote a significance, sign, or refinement bit to be encoded for the RAHT coefficient c_t at node t, level ℓ, and bit-plane b. As context for y, we propose to use a context vector $\mathbf{X}$ comprising four sub-context vectors $(\mathbf{X}_A, \mathbf{X}_B, \mathbf{X}_C, \mathbf{X}_D)$, as follows. Sub-context vector $\mathbf{X}_A$ is a 6-tuple comprising the category (0 for *significance*, 1 for *sign*, and 2 for *refinement*), the bit plane b $(0, \ldots, b_{\max})$, the level ℓ $(0, \ldots, L)$, and the weights w_t, $w_{t'}$, and $w_{t''}$, respectively, for the node t at level ℓ, its parent t' at level $\ell - 1$, and its grandparent t'' at level $\ell - 2$. The *weight* of a node is given by the number of occupied voxels that are descendants of the node in the octree. Sub-context vector $\mathbf{X}_B$ is an 81-tuple of binary variables indicating the occupancy of the voxels in each of the three $3 \times 3 \times 3$ voxel neighborhoods $\mathcal{N}_t$, $\mathcal{N}_{t'}$, and $\mathcal{N}_{t''}$ surrounding nodes t, t', and t'' at levels ℓ, $\ell + 1$, and $\ell + 2$, respectively. Sub-context vector $\mathbf{X}_C$ is an 81-tuple of real-valued coefficients $\hat{c}_{t^*}$ for the occupied voxels, or 0 for the unoccupied voxels, as indicated by $\mathbf{X}_B$. For an occupied voxel, the coefficient $\hat{c}_{t^*}$ at node t^* and level ℓ^* is reconstructed from the available causal information, i.e., down through bit-plane b if $\ell^* < \ell$ or if $t^* < t$ and $\ell^* = \ell$, or otherwise down through bit-plane $b+1$. Sub-context vector $\mathbf{X}_D$ is a similar 81-tuple of reconstructed coefficients, but with coefficients normalized as

$$\tilde{c}_{t^*} = \frac{\hat{c}_{t^*}}{\sqrt{w_{t^*}}}. \tag{4.32}$$

In total, the context vector $\mathbf{X}$ is 330 elements long. We use $\mathbf{X}$ to estimate the conditional probability $p = P(y = 1|\mathbf{X})$ using a function $q = f_\theta(\mathbf{X})$ trained to minimize the cross-entropy (or alternatively the Kullback-Leibler divergence [89]) between the true and predicted probability dis-

tributions [93],

$$R = -\sum_k (y_k \log_2(f_\theta(\mathbf{X}_k)) + (1 - y_k) \log_2(1 - f_\theta(\mathbf{X}_k))), \tag{4.33}$$

which is also the expression for the bit-rate on the dataset using an arithmetic coder [87]driven by context model $q = f_\theta(\mathbf{X})$, this means that minimizing the KL divergence implies in a reduction of the arithmetic coder bit-rate.

Although many neural architectures could be used for the task, we opted for a simple multi-layer perceptron (MLP) with 330 units in the input layer, 1024 units in a first hidden layer, 512 units in a second hidden layer, and one output unit. We use sigmoidal activation in the last (output) layer, and rectified linear unit (ReLU) activation in the hidden layers. The network architecture is illustrated in Figure 4.3.

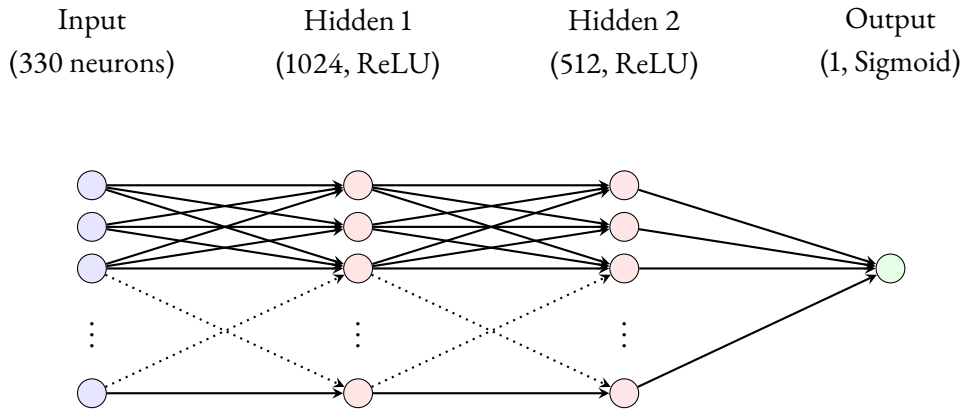


Figure 4.3: Diagram of the fully connected multi-layer perceptron (MLP) architecture used with 330 input units, 1024 units in the first hidden layer (ReLU activation), 512 units in the second hidden layer (ReLU activation), and one output unit (sigmoidal activation).

4.3 EXPERIMENTAL RESULTS

We evaluate our proposed neural-network-based context modeling of SPIHT for RAHT coefficients (C-SPIHT). In our tests, we use five frontal upper-body sequences from Microsoft Research ("Andrew", "David", "Phil", "Ricardo" and "Sarah") [1] and four full body sequences from 8i Labs ("Longdress", "Loot", "Redandblack" and "Soldier") [2]. We use 9-fold cross-validation to evaluate our models, where each of the nine point cloud sequences is held out once. Figures 3.3 and 3.4 show the projections of each of the point clouds used in our experiments.

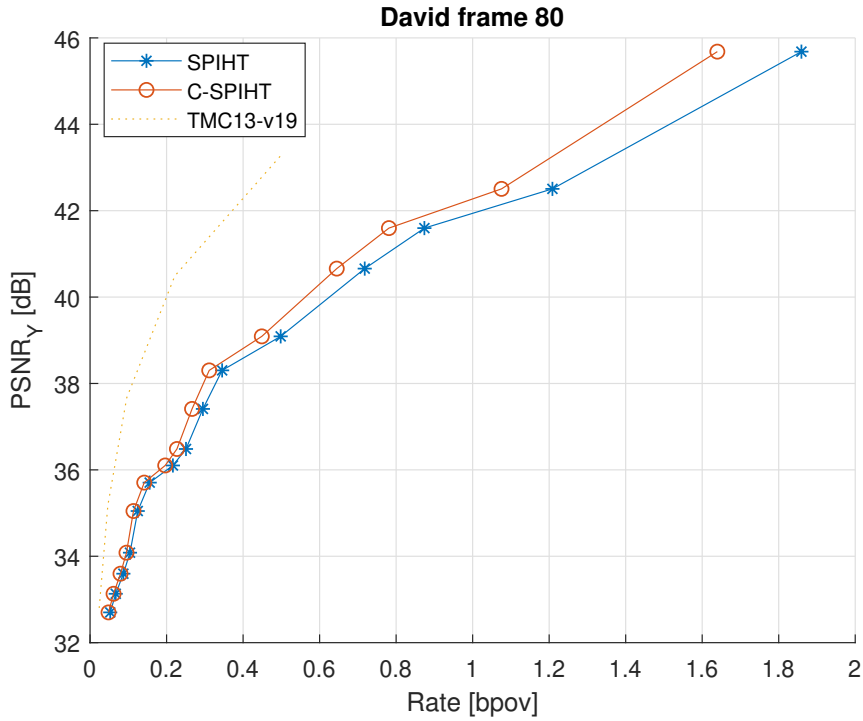


Figure 4.4: Rate-distortion comparison between G-PCC (TMC13-v19), SPIHT and C-SPIHT for the 80^{th} frame of PC "David". The rates used range from bit-plane 5 to bit-plane 12 of a total of 14 bit-planes.

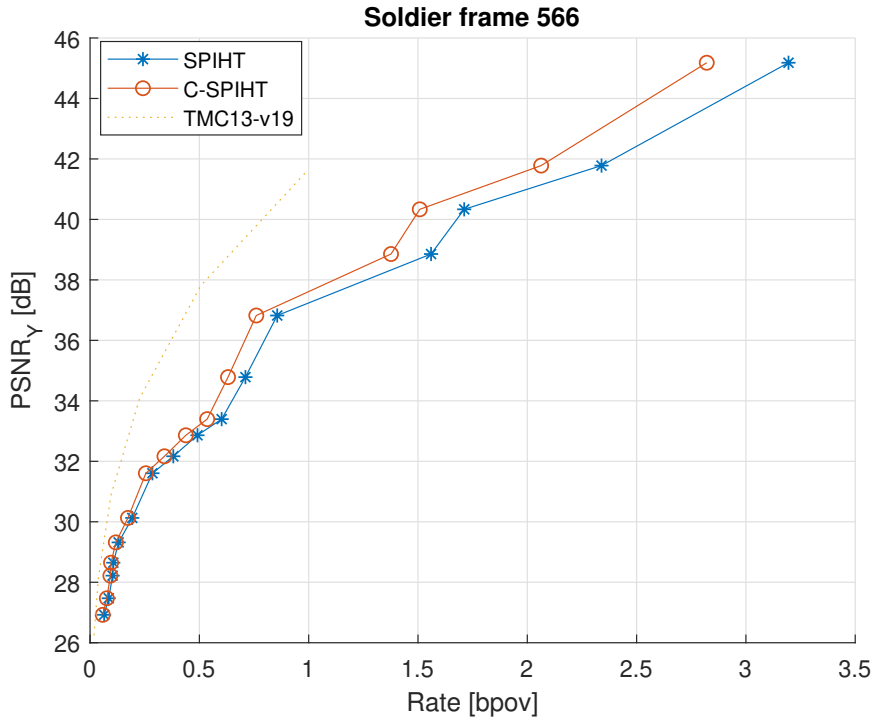


Figure 4.5: Rate-distortion comparison between G-PCC (TMC13-v19), SPIHT and C-SPIHT for the 566^{th} frame of PC "Soldier". The rates used range from bit-plane 5 to bit-plane 12 of a total of 14 bit-planes.

Figures 4.4 and 4.5 are examples of comparisons of the rate-distortion performance of C-SPIHT relative to SPIHT for different point clouds. The figures also include the rate-distortion performance of G-PCC to give the reader an idea of how much gain G-PCC offers over SPIHT and C-SPIHT, at expense of scalability. These are two examples to show C-SPIHT outperforming SPIHT. Table 4.1 presents Bjontegaard Delta (BD) [84] PSNR-Y gains and bit-rate savings of C-SPIHT relative to SPIHT for each tested sequence. From Table 4.1, we can see that the use of context modeling improved the performance of SPIHT for all tested sequences. The best performance in terms of bit-rate savings is 15.98% for "David". The worst performance in terms of bit-rate savings is 9.97% for "Andrew". The average bit-rate savings is 12.97%.

It is interesting to notice that in Figures 4.4 and 4.5, the points used to plot the curves in both SPIHT and C-SPIHT are not only quantization steps but there are also points that represents random stop-points in the bit stream between quantization steps to demonstrated the capability of an embedded bit stream to be decoded at any given point, this intrinsic functionality is desirable in many applications as mentioned before.

Table 4.1: Average BD PSNR-Y gains and average bit-rate savings of C-SPIHT relative to SPIHT.

Point Cloud	PSNR-Y [dB]	bit-rate [%]
Andrew	1.83	-9.97
David	1.39	-15.98
Ricardo	1.40	-13.07
Sarah	1.52	-12.94
Phil	1.74	-13.08
Soldier	1.02	-11.93
Loot	1.08	-12.75
Redandblack	1.27	-13.03
Longdress	1.13	-14.02
Average	**1.37**	**-12.97**

We are also interested in understanding the individual contributions made by each sub-context vector towards the above results. We undertake a process of retraining and inference, experimenting with multiple combinations of the five sub-context vectors that compose $\mathbf{X}$. The results are shown in Table 4.2, where the average performance results for each of the explored combinations are outlined.

The table reveals that for best performance, both $\mathbf{X}_A$ and $\mathbf{X}_B$ are indispensable as well as one of $\mathbf{X}_C, \mathbf{X}_D$. Notably, our findings indicate that the normalization process appears to exert minimal influence on the overall outcome, suggesting a degree of robustness in the model's ability to adapt.

Table 4.2: Average bit-rate savings of C-SPIHT over SPIHT for different context combinations.

Context combination	bit-rate Savings [%]
A	-9.03
$A + B$	-10.78
$A + B + C$	**-12.97**
$A + B + D$	**-12.97**
$B + C + D$	-10.01
$A + B + C + D$	**-12.97**

Non-embedded PCC such as those provided by G-PCC largely outperforms the proposed embedded coders as the comparisons in section 3.3 make clear. We understand that our goal is to continuously improve embedded coders up to a point to make them competitive to non-embedded coders. Hence, our plots and results are relative to regular SPIHT-RAHT coder, while about half that bit-rate can be expected using the most recent version of G-PCC, for example.

Figure 4.6 shows projections of sequence "David" for PCs decoded from a single bit stream truncated at different bit-planes, indicating the rate in bits per occupied voxel (bpov).

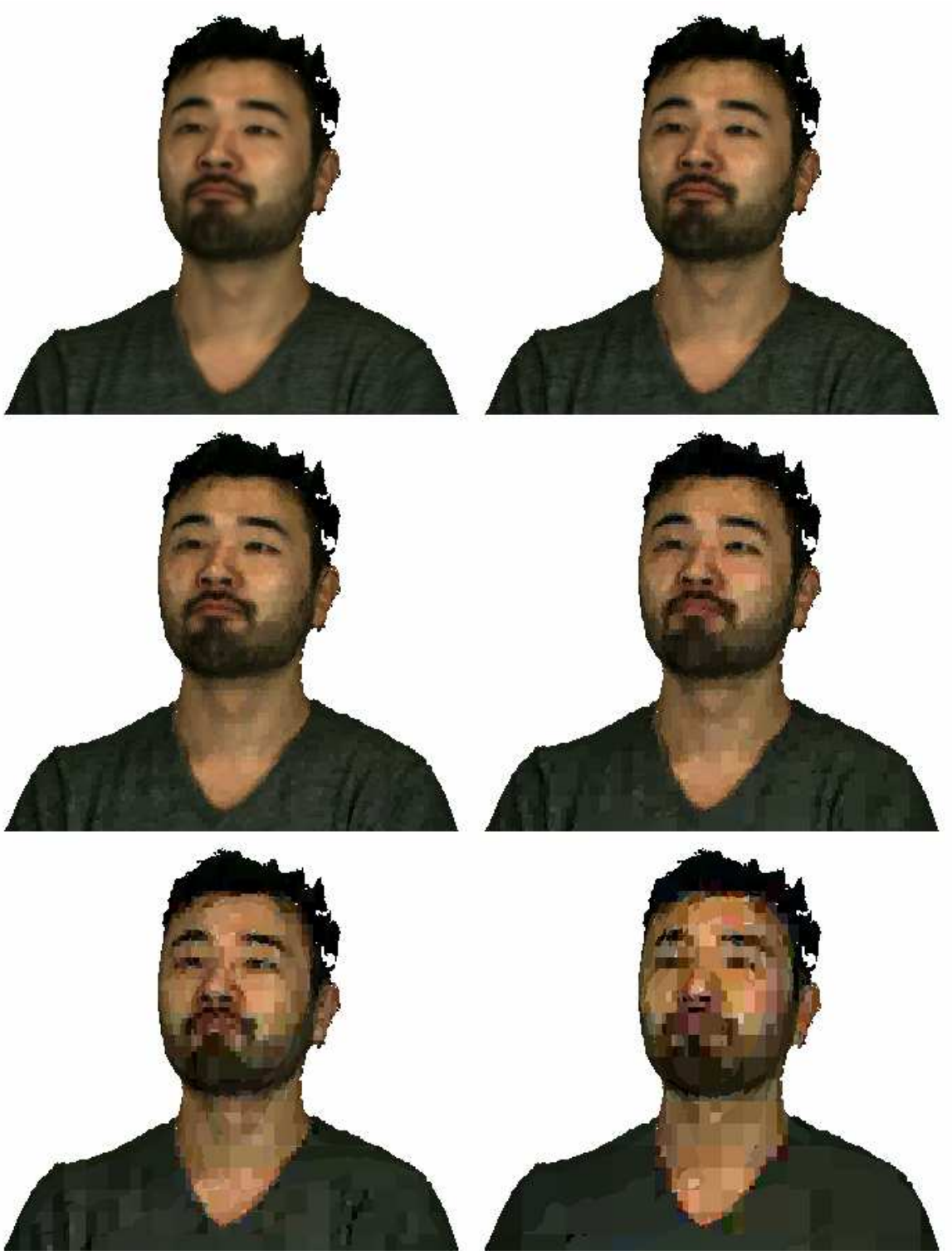


Figure 4.6: Projections of the point cloud frame “David” decoded using C-SPIHT and cutting the bit stream at different bit-planes. From left to right, top to bottom: 7.1761 bpov, 0.2652 bpov, 0.1129 bpov, 0.0480 bpov, 0.0220 bpov and 0.0096 bpov.

4.4 CONCLUSIONS

This chapter presents an improvement to the embedded point cloud attribute coder presented in the previous chapter, which encodes RAHT coefficients using set partitioning in hierarchical trees. We define a context of node ancestors and use a neural network as a context model to generate probabilities to drive an arithmetic encoder, in order to generate the final bit stream. Our experimental results show that the use of C-SPIHT for point clouds improves regular SPIHT, while still providing a fully embedded bit stream. We are aware there is much to do to improve the proposed coder performance to the level of state-of-the-art non-embedded coders. We also understand there is a trade-off for such a functionality. Further improvements suggested are to develop longer RAHT basis functions to see if they can replace coefficient prediction steps that are key to other encoders such as G-PCC, in an effort to bridge the performance gap.

5 EMBEDDED BIT-STREAM REGION-OF-INTEREST CODING OF POINT CLOUD ATTRIBUTES

"In all things of nature there is
something of the marvelous."
- Aristotle

Point clouds may have regions of interest (ROI) with special significance or relevance [94], [95]. These regions can be used to selectively increase fidelity during compression, as the case in image and video compression [59], [96]–[99].

Inspired by [25], [94], [95], we propose a method of combining embedded attribute coding with ROI for point clouds. The proposed approach has the advantage that instead of adapting each codec in a specific way to adjust its fidelity, we advocate using the ROI to modify the distortion measure. There is a simple mapping from the ROI to the distortion measure, which can be quantified (for example using perceptual experiments) independently of any particular embedded codec. As our codec, we choose transform coding with Region-Adaptive Hierarchical Transform (RAHT) [21] because it is automatically optimized for the distortion measure by virtue of its measure-theoretic interpretation [100] and it can be used in an embedded coder. The proposed method is also independent of any region of interest or saliency detection algorithm. Even though there is available literature on embedded ROI coding for images [101], [102] and in ROI coding for point clouds [103], we believe that the work proposed in this chapter was responsible for the publication of the first paper [27] on embedded region of interest attribute coding for point clouds.

This chapter presents a novel approach to embedded Region-of-Interest (ROI) attribute coding for point clouds. The proposed method is an embedded point cloud attribute coder incorporating regions of interest by interleaving bit-streams for ROI and non-ROI regions. This approach brings us some advantages. Firstly, it is independent of any specific ROI detection algorithm. Secondly, no increased complexity is imposed on the decoder. The results demonstrate an improvement in the quality of reconstructed voxels within the ROI, accompanied by some degradation of reconstructed voxels outside the ROI.

5.1 ROI-WEIGHTED DISTORTION MEASURE AND MEASURE-THEORETIC RAHT

In lossy compression, artifacts introduced in some regions can highly influence the subjective perception of quality. The squared error may not correlate well with subjective perception of quality.

Let a generic attribute $\mathbf{X} = \{X_i\}$ be considered, where X_i is the attribute value associated with the i-th *voxel* of a total of N occupied *voxels*, and let $\hat{\mathbf{X}} = \{\hat{X}_i\}$ be its reconstruction. A better way

to account for the relevance is to consider a *weighted squared error*, defined as:

$$d(\mathbf{X}, \hat{\mathbf{X}}) = \sum_i w_i (X_i - \hat{X}_i)^2, \tag{5.1}$$

where w_i are the *weights* associated with each *voxel* and reflects its semantic or perceptual importance.

A region of interest (ROI) can be defined via the weights w_i. A natural choice is to set $w_i = 1$ for all *voxels* outside the ROI and $w_i > 1$ for *voxels* inside the ROI. A codec that minimizes this distortion measure subject to a rate constraint will tend to reproduce X_i as $\hat{X}_i$ with squared error inversely proportional to w_i. It is also possible to define a smooth transition between the ROI border or define multiple ROI with different weights.

The weights can be used to define a measure on the set of *voxels* composing the point cloud. A measure on a set is a systematic way to assign a number to each suitable subset of that set, intuitively interpreted as its size. As each *voxel* has a weight associated with it, we define the measure of a subset S of *voxels* by:

$$\mu(S) = \sum_{i \in S} w_i. \tag{5.2}$$

The measure is non-negative, as all weights are non-negative and the measure of two disjoint subsets is equal to the sum of their measures.

The definition of measure (5.2) induces the definition of the integral,

$$\int f(\mathbf{x}) d\mu(\mathbf{x}) = \sum_i w_i f(\mathbf{x_i}) \tag{5.3}$$

because

$$\frac{d\mu}{d\mathbf{x}} = \sum_i w_i \delta \left(|\mathbf{x} - \mathbf{x_i}|^2 \right) \tag{5.4}$$

since *voxels* are dispersed in discrete positions $\mathbf{x_i}$.

The definition of the integral (5.3) in turn induces the definition of the inner product,

$$\langle f, g \rangle = \int f(\mathbf{x}) g(\mathbf{x}) d\mu(\mathbf{x}) = \sum_i w_i f(\mathbf{x_i}) g(\mathbf{x_i}), \tag{5.5}$$

which in turn induces the definitions of orthogonality, $f \perp g \Leftrightarrow \langle f, g \rangle = 0$, and norm, $||f|| = (\langle f, f \rangle)^{1/2}$. Altogether, these induce a Hilbert space. The weighted squared error given by (5.1) is precisely the squared norm $||f - \hat{f}||^2$ of this Hilbert space, where $f_i = X_i$ and $\hat{f}_i = \hat{X}_i$.

RAHT is region-adaptive to remain orthonormal regardless of the locations of the points. Recently RAHT has been shown to be interpretable as a separable piecewise constant spline wavelet that is orthonormal with respect to the inner product $\langle f, g \rangle$ defined by the weights w_i [100]. Thus, if the weights are set to the weights in the ROI-weighted distortion measure, the transform will remain orthonormal, and moreover uniform scalar quantization of the transform coefficients with

the quantization stepsize set to a constant will minimize the ROI-weighted distortion measure, at least at high rates.

To be specific, let $\mathbb{R}^3$ be uniformly partitioned into cubes of size $2^{-m} \times 2^{-m} \times 2^{-m}$, half-cubes of size $2^{-m} \times 2^{-m} \times 2^{-(m+1)}$, and quarter-cubes of size $2^{-m} \times 2^{-(m+1)} \times 2^{-(m+1)}$, and let $\mathcal{F}_{3m}$, $\mathcal{F}_{3m+1}$, and $\mathcal{F}_{3m+2}$ be the spaces of all functions $f_\ell : \mathbb{R}^3 \to \mathbb{R}$ that are piecewise constant on these blocks, for $\ell = 3m$, $3m+1$, and $3m+2$, respectively. The nested sequence of function spaces $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \cdots \subseteq \mathcal{F}_\ell \subseteq \mathcal{F}_{\ell+1} \subseteq \cdots$ approximates ever more finely (with respect to the norm, i.e., the weighted squared error) the space of piecewise continuous functions.

Now let $B_{\ell,n}$ denote a block at level ℓ indexed by n, let $1_{B_{\ell,n}}(\mathbf{x})$ be its indicator function, and let $w_{\ell,n} = \mu(B_{\ell,n})$ be its measure. Then $\mathcal{F}_\ell$ is spanned by the basis functions

$$\phi_{\ell,n}(\mathbf{x}) = w_{\ell,n}^{-1/2} 1_{B_{\ell,n}(\mathbf{x})}, \tag{5.6}$$

which are orthogonal to each other and are normalized with respect to the inner product and norm induced by the weighted measure. Similarly, let $B_{\ell+1,n_0}$ and $B_{\ell+1,n_1}$ denote the sub-blocks of $B_{\ell,n}$, and let $\mathcal{G}_\ell$ be the orthogonal complement of $\mathcal{F}_\ell$ in $\mathcal{F}_{\ell+1}$. Then $\mathcal{G}_\ell$ is spanned by the basis functions

$$\psi_{\ell,n}(\mathbf{x}) = \frac{-w_{\ell+1,n_0}^{-1} 1_{B_{\ell+1,n_0}}(\mathbf{x}) + w_{\ell+1,n_1}^{-1} 1_{B_{\ell+1,n_1}}(\mathbf{x})}{(w_{\ell+1,n_0}^{-1} + w_{\ell+1,n_1}^{-1})^{-1/2}} \tag{5.7}$$

which are orthogonal to each other and to the functions (5.6), and are normalized, as can be verified by the diligent reader. Thus any function $f_{\ell+1} \in \mathcal{F}_{\ell+1}$ can be written as:

$$f_{\ell+1}(\mathbf{x}) = \sum_n F_{\ell,n} \phi_{\ell,n}(\mathbf{x}) + \sum_n G_{\ell,n} \psi_{\ell,n}(\mathbf{x}), \tag{5.8}$$

where the $F_{\ell,n} = \langle f_{\ell+1}, \phi_{\ell,n} \rangle$ are known as low-pass coefficients and the $G_{\ell,n} = \langle f_{\ell+1}, \psi_{\ell,n} \rangle$ are known as high-pass coefficients. After some algebraic manipulation, (5.6) and (5.7) can be expressed recursively as the "two-scale equations"

$$\phi_{\ell,n}(\mathbf{x}) = a\phi_{\ell+1,n_0} + b\phi_{\ell+1,n_1} \tag{5.9}$$

$$\psi_{\ell,n}(\mathbf{x}) = -b\phi_{\ell+1,n_0} + a\phi_{\ell+1,n_1}, \tag{5.10}$$

where $a = \frac{\sqrt{w_{\ell+1,n_0}}}{\sqrt{w_{\ell,n}}}$ and $b = \frac{\sqrt{w_{\ell+1,n_1}}}{\sqrt{w_{\ell,n}}}$. Substituting these into the definitions of $F_{\ell,n}$ and $G_{\ell,n}$, we obtain

$$\begin{bmatrix} F_{\ell,n} \\ G_{\ell,n} \end{bmatrix} = \begin{bmatrix} a & b \\ -b & a \end{bmatrix} \begin{bmatrix} F_{\ell+1,n_0} \\ F_{\ell+1,n_1} \end{bmatrix}, \tag{5.11}$$

which is a Givens rotation whose angle of rotation depends on the relative weights of the sub-blocks. RAHT applies (5.11) recursively to expand $f_L \in \mathcal{F}_L$ as

$$f_L(\mathbf{x}) = \sum_n F_{0,n} \phi_{0,n}(\mathbf{x}) + \sum_{\ell=0}^{L-1} \sum_n G_{\ell,n} \psi_{\ell,n}(\mathbf{x}), \tag{5.12}$$

where L is chosen large enough so that each cube $B_{L,n}$ contains at most a single point, say $\mathbf{x}_i$ with value $f_i = f(\mathbf{x}_i)$. The number of coefficients is N, i.e., RAHT is critically sampled. (For details,

see [100].) Note that $\phi_{L,n}(\mathbf{x}) = w_i^{-1/2} 1_{B_{L,n}}(\mathbf{x})$, and therefore $F_{L,n} = \langle f, \phi_{L,n} \rangle = w_i^{1/2} f_i$. This generalizes RAHT in [56], for which $w_i = 1$ for all points $i = 1, \ldots, N$.

The RAHT coefficients are uniformly scalar quantized with stepsizes $\Delta(F_{0,n})$ and $\Delta(G_{\ell,n})$, $\ell = 0, \ldots, L-1$, and are entropy coded. Because Givens rotations are orthonormal, norm is preserved. Thus the squared quantization error is

$$\sum_n (F_{0,n} - \hat{F}_{0,n})^2 + \sum_{\ell=0}^{L-1} \sum_n (G_{\ell,n} - \hat{G}_{\ell,n})^2 = \sum_{i=1}^{N} w_i (f_i - \hat{f}_i)^2, \tag{5.13}$$

which is the same as the ROI-weighted distortion measure. Since a constant step-size $\Delta = Q_{step}$ minimizes the squared quantization error subject to an entropy constraint, at least at high rates [104], setting the step-sizes of the RAHT coefficients to a constant also minimizes the ROI-weighted distortion measure desired for ROI coding [94].

In summary, with RAHT, at the encoder, *voxels* in ROI should have initial weights set to $w_i = w$ and initial attributes scaled by $\sqrt{w}$, while *voxels* outside the ROI are initialized with weights set to $w_i = 1$. The decoder should scale back the attributes.

5.2 EMBEDDED REGION OF INTEREST CODING

In point cloud compression there is a trade-off between the number of bits spent to encode the point cloud and the quality of the reconstructed point cloud at the decoder. The higher the quality, the more bits are needed. Regions of interest are supposed to have a higher semantic or perceptual significance than the rest of the point cloud. Therefore, an encoder that prioritizes the quality of those regions, while degrading the others, tends to produce reconstructed point clouds with a better subjective quality, when compared to an encoder that treats all regions equally, for the same number of bits.

In this work, the subject's face in the point cloud is chosen to be the ROI. The proposed method is agnostic to any ROI extraction algorithm. In this work the algorithm used was the one described in [94]. As the human brain tends to be more sensitive to artifacts introduced in the face on reconstructed point clouds, it is believed that prioritizing the subject's face quality during compression may lead to better subjective quality [105], [106].

The decoder must receive information about the ROI location. Assuming there are M occupied voxels in the point cloud, we are required to encode a binary vector $b = [b_0, b_1, \ldots, b_{M-1}]$, representing the inclusion or exclusion of each voxel from the ROI. By sorting the voxels based on their Morton codes [92] to maintain neighborhood relationships, the b_i bits can be employed to construct a differential vector $\overline{b} = \left[\overline{b_0}, \overline{b_1}, \ldots, \overline{b_{M-1}}\right]$ where

$$\overline{b_i} = \begin{cases} b_0 & i = 0 \\ 1 & b_{i-1} \neq b_i, i > 0 \\ 0 & b_{i-1} = b_i, i > 0 \end{cases}. \tag{5.14}$$

The vector $\overline{b}$ exhibits extended sequences of zeros. It is encoded with an algorithm rooted in the run-length Golomb-Rice coder [48], albeit with a modification where solely the run-lengths are encoded using Golomb-Rice. The number of bits spent signaling the ROI is typically not significant compared to the overall encoding bit-rate, an average of 0.003 bits per occupied voxel (bpov) in the dataset used.

In order to achieve a fully embedded bit-stream that prioritizes the region of interest some adaptations needed to be made in the SPIHT for RAHT algorithm described in chapter 3. Here, we are adding to the framework the possibility to use what we call an interleaved bit-stream. That is, for the same point cloud stream, there is a part of the bit-stream that conveys information of the ROI voxels and a part of it that conveys information about the non-ROI voxels. The bits are interwoven in a way that, since the decoder already knows the ROI, it can decode the full bit-stream as an embedded bit-stream. Figure 5.1 illustrates the proposed method.

The ROI is prioritized as the encoder sends a different proportion of ROI and non-ROI bits, in the interlacing level of the encoding. Lets say we have an interlacing level of ζ. An interlacing level of 0 is the same as a regular encoding without any prioritization of the ROI. As we increase ζ the ROI is more prioritized. ζ means that for each non-ROI bit sent to the decoder 2^{ζ} ROI bits are sent.

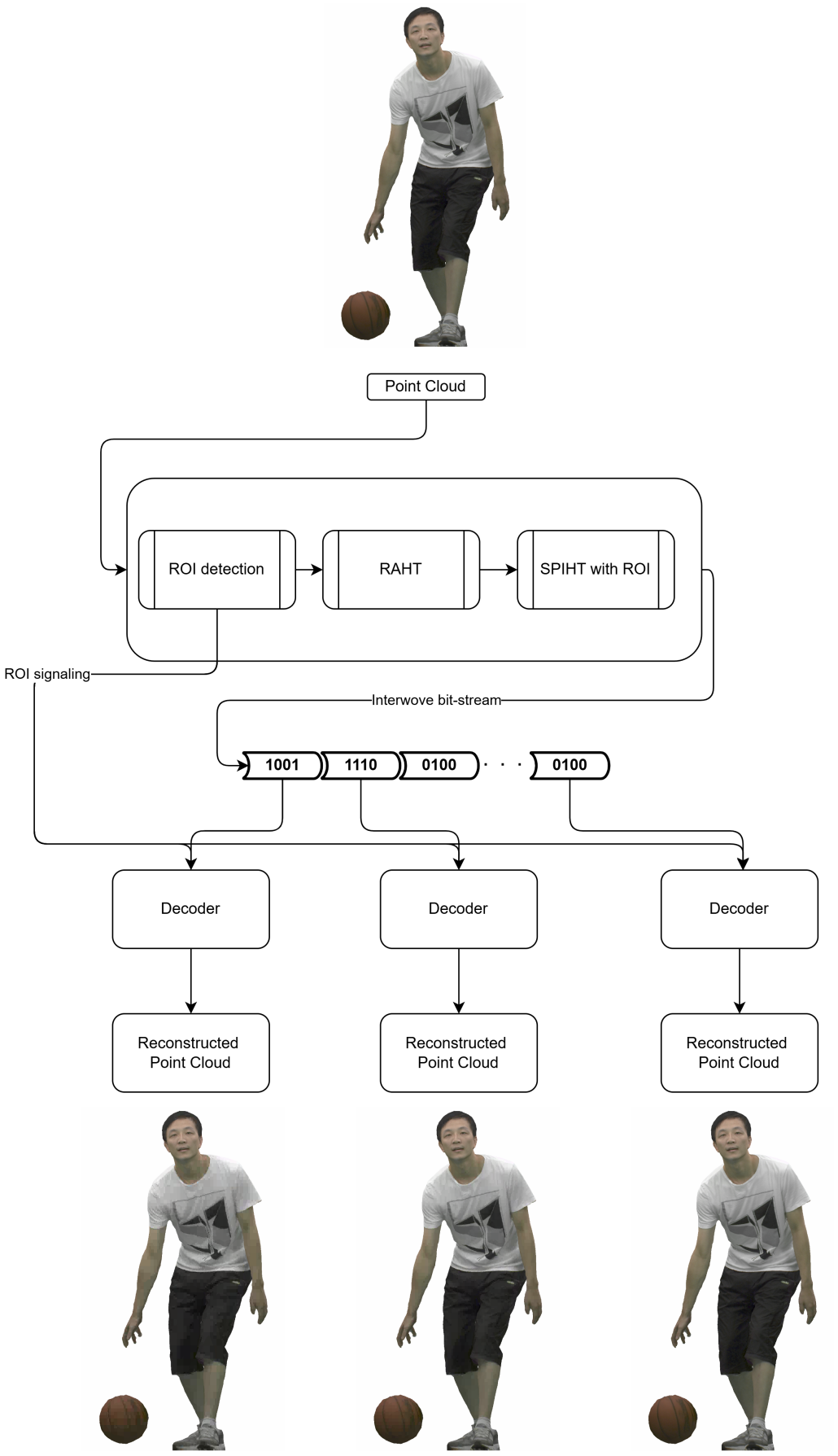


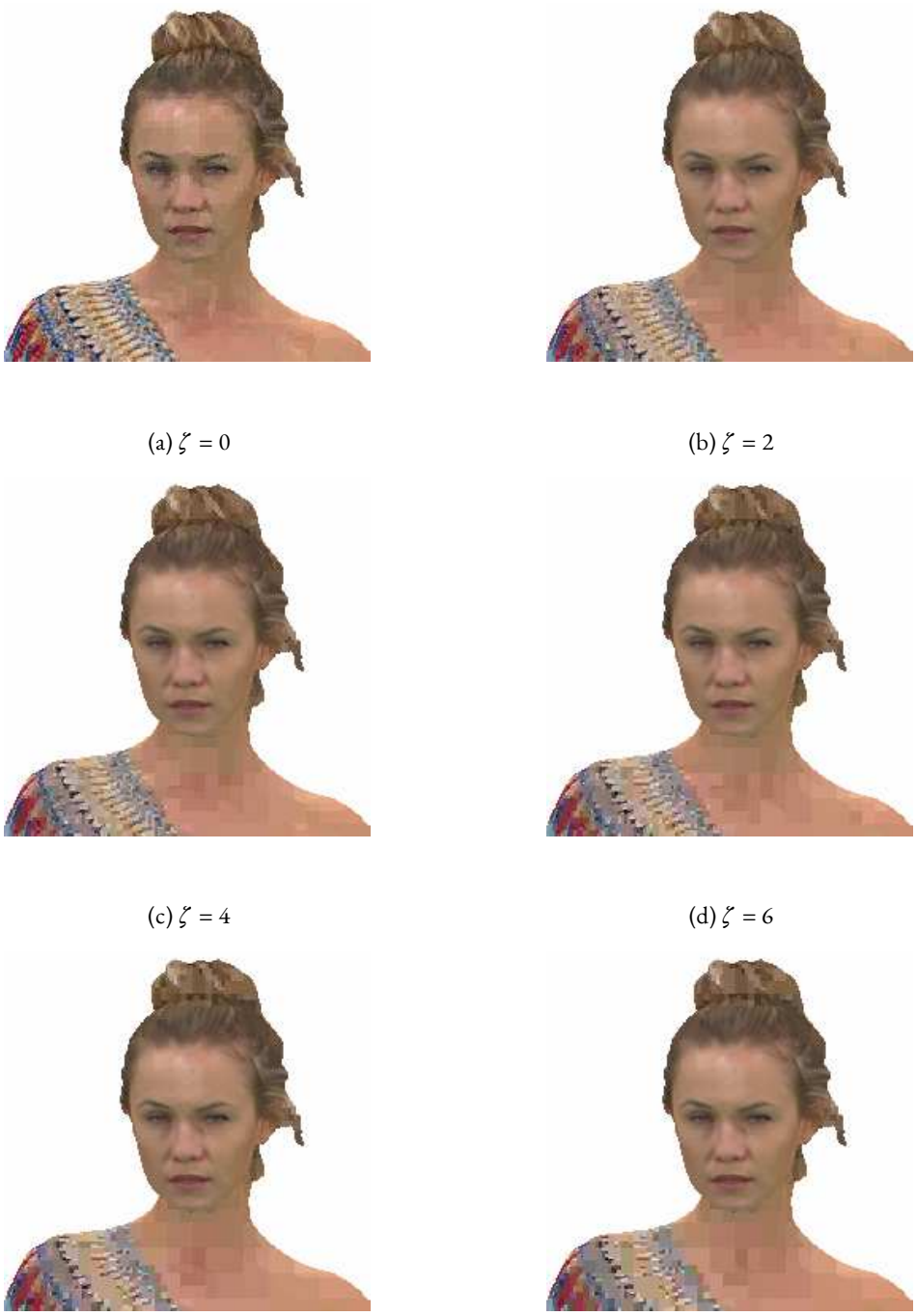
Figure 5.1: Diagram illustrating the proposed algorithm. The interwove bit-stream is composed accordingly to the interlacing level. The ROI signaling is transmitted as described in equation 5.14.

Figures 5.2 and 5.3 illustrate different reconstructions of the point cloud “Longdress" with approximately the same number of bits used by the decoder at different interlacing levels.

(a) $\zeta = 0$ (b) $\zeta = 2$ (c) $\zeta = 4$

(d) $\zeta = 6$ (e) $\zeta = 8$ (f) $\zeta = 10$

Figure 5.2: Projections of the point cloud “Longdress" encoded with different interlacing levels. The number of bits used in the decoding was adjusted so that every reconstruction had approximately 0.91 bpov.



(a) $\zeta = 0$ (b) $\zeta = 2$

(c) $\zeta = 4$ (d) $\zeta = 6$

(e) $\zeta = 8$ (f) $\zeta = 10$

Figure 5.3: Close-up in the ROI of the reconstructed point clouds shown in Fig. 5.2. Note that for $\zeta = 2$ ROI quality might be sufficiently high.

The unique contribution of the interleaved bit-stream in our proposed method lies in its novel approach to prioritizing ROI voxels within point cloud attribute embedded coding. Unlike traditional ROI coding techniques that often require significant modifications to existing codecs, our method seamlessly integrates with embedded attribute coding by interleaving a higher proportion

of ROI bits within the bit-stream. This ensures that ROI regions receive enhanced quality without compromising the overall coding efficiency. By comparing our approach with standard ROI coding methods, such as region-based quantizer adaptations, we demonstrate that our interleaved bit-stream not only simplifies the integration process but also achieves superior perceptual quality in the ROI regions with minimal degradation in non-ROI areas.

5.3 EXPERIMENTAL RESULTS

To test our proposed encoder we used 10 point clouds: Boxer, Longdress, Loot, Redandblack, Soldier, Thaidancer, Basketball player, Dancer, Exercise and Model, all voxelized at depth 10 (i.e. $1024 \times 1024 \times 1024$ voxels) [2]–[4]. Figures 3.4 and 5.4 show the projections of each of the point clouds used in our experiments. The PSNR is computed as a weighted PSNR defined as

$$PSNR(\mathcal{P}, \hat{\mathcal{P}}, \{w_i\}) = 10 \log_{10} \left(\frac{255^2}{E_{noise}} \right), \tag{5.15}$$

where w_i is the weight of the voxel, potentially different for ROI and non-ROI voxels, and the noise energy can be redefined as

$$E_{noise} = \frac{\sum_i w_i \left(Y_i - \hat{Y}_i \right)^2}{\sum_i w_i}. \tag{5.16}$$

$\mathcal{P}$ is the original point cloud, $\hat{\mathcal{P}}$ is the reconstructed point cloud, Y_i is the original value and $\hat{Y}_i$ is the reconstructed value, for the Y channel where the weight of each voxel is $w_{ROI} = 2^{\zeta}$ if it is inside the ROI and 1, otherwise.

(a) Boxer. (b) Thaidancer. (c) Model.

(d) Dancer. (e) Exercise. (f) Basketball player.

Figure 5.4: Projections of the point clouds used in our tests 8i Voxelized Surface Light Field (8iVSLF) Dataset (a)-(b) [3] and Owlii (c)-(f) [4].

Figures 5.5, 5.6, 5.7 and 5.8 show the average rate-distortion curves computed for the 10 tested point clouds for different ζ. We used $\zeta = 0, 2, 4, 6, 8, 10$, resulting in $w_{ROI} = 1, 4, 16, 64, 256, 1024,$

while voxels outside the ROI have weight $w = 1$. In Fig. 5.5 the PSNR is computed only for voxels in the ROI. Bits for the side information are considered. However, when $w = 1$, there is no difference in weight between voxels situated inside and outside the ROI. In this specific scenario, encoding the vector $\bar{b}$ is unnecessary. As ζ increases the encoder favors the ROI bits and we see an increase in PSNR for voxels inside the ROI. The opposite happens for voxels outside the ROI (Fig. 5.6). The weighted PSNR increases drastically for $\zeta > 4$ and slightly improves for lower bit-rates with $0 < \zeta \leq 4$ (Fig. 5.7). The PSNR over all voxels decreases for $\zeta > 0$ (Fig. 5.8).

The proposed method exhibits a limitation concerning the relative size of the ROI compared to the entire point cloud. Tests conducted on the Microsoft Voxelized Upper Bodies dataset [1], where the selected ROIs, on average, constitute more than 30% of the total point cloud, have made this clear. This significant proportion results in a diminished improvement in quality, as the method's effectiveness decreases when the ROI occupies a substantial portion of the point cloud. Consequently, the proposed method does not perform optimally under these conditions.

Subjectively, from Fig. 5.3 the reconstructions with greater interlacing level seem to have a higher quality, since our cognitive vision is more sensitive to artifacts in the face than in rest of the scene. The reconstruction with $\zeta = 2$ seems to have a sufficiently high ROI quality. Essentially, the choice of ζ represents a trade-off and an ideal value may have to be carefully chosen so that the ROI quality is satisfactory and further quality increase may not be worth the further decrease in quality of non-ROI regions.

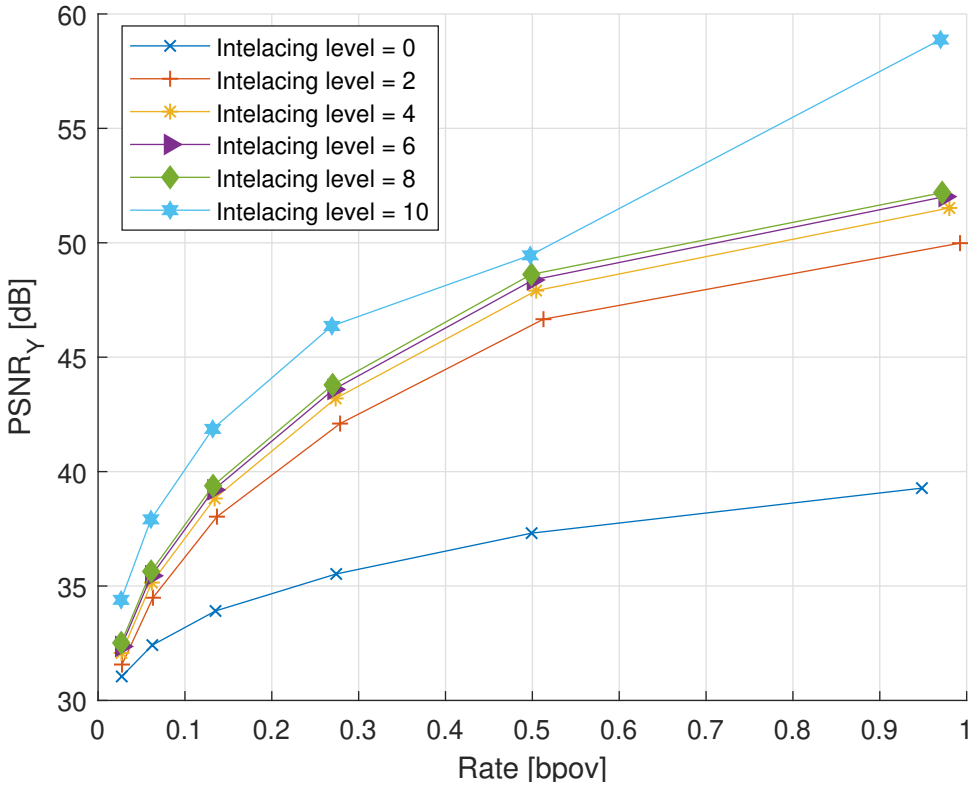


Figure 5.5: Traditional PSNR computed only for voxels inside the ROI. Bits for the side information are considered.

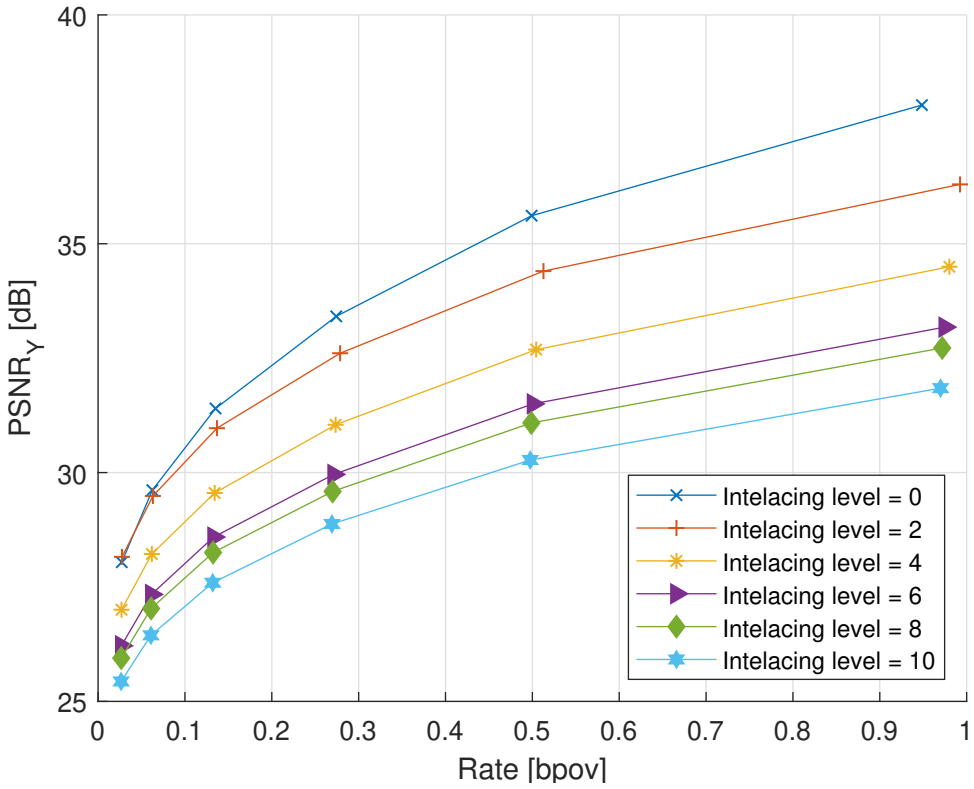


Figure 5.6: Traditional PSNR computed only for voxels outside the ROI. Bits for the side information are considered.

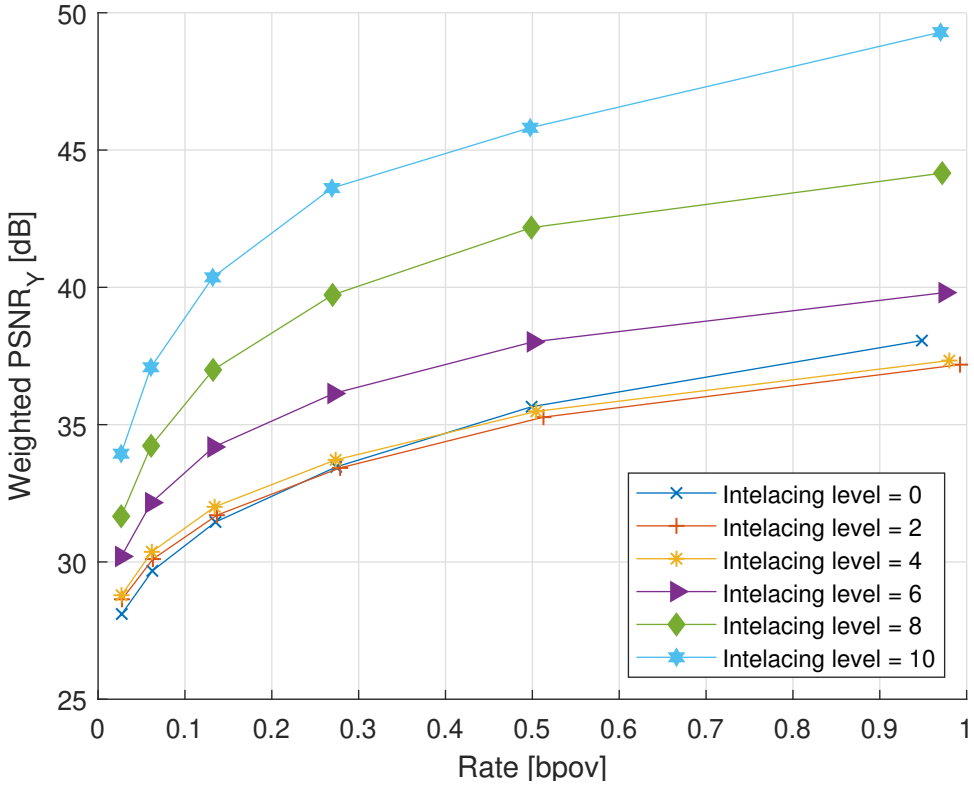


Figure 5.7: Weighted PSNR calculated over all voxels of the point cloud. Bits for the side information are considered.

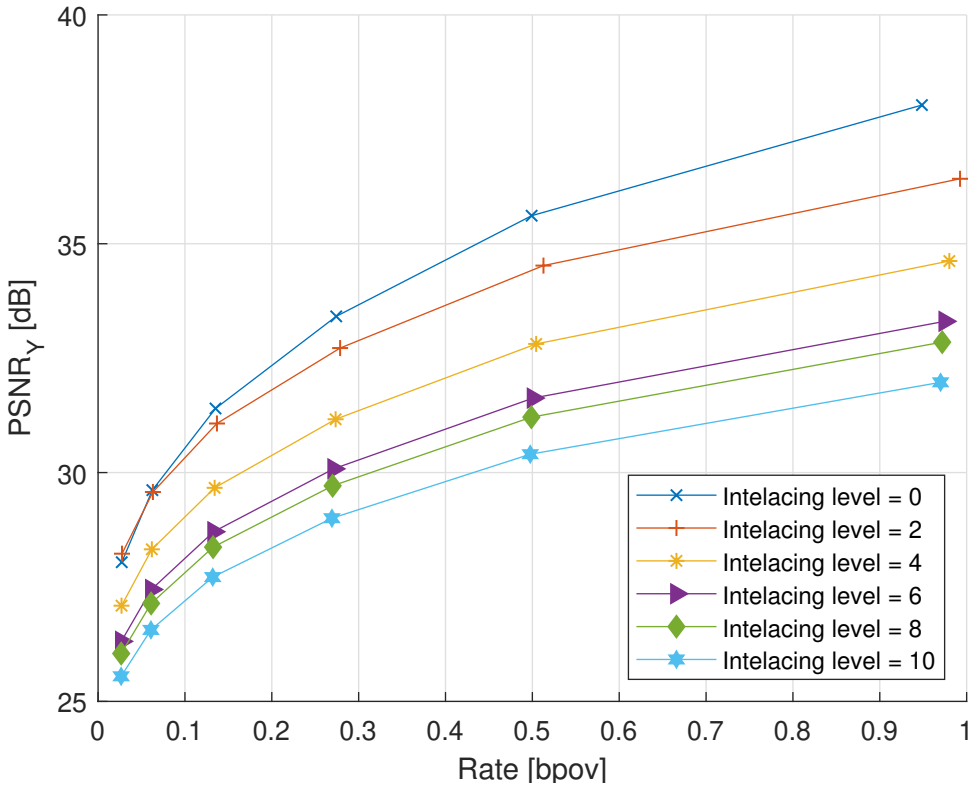


Figure 5.8: Traditional PSNR computed considering all the voxels of the point cloud. Bits for the side information are considered.

5.4 CONCLUSIONS

This chapter introduced an embedded region-of-interest attribute coding for point clouds, through the use of interleaved embedded bit-streams for the ROI and the non-ROI voxels of the point cloud. The proposed interleaving method is agnostic to the embedded point cloud codec or to the region of interest detection algorithm. The results have shown an increase in the quality of the reconstructed voxels inside the region of interest at the expense of some degradation outside the ROI, with little change in encoder or decoder complexity. The choice of the interlacing level must be carefully made, since it represents a trade-off between ROI and non-ROI quality and at that point the further increase in ROI reconstruction quality might not be worth the further decrease in non-ROI quality. We believe our coder may find itself very useful in a number of applications.

6 NEURAL-BASED CONTEXT-ADAPTIVE RESIDUAL MODELING FOR LOSSLESS IMAGE COMPRESSION

"A mind needs books like
a sword needs a whetstone."
- Tyrion Lannister

Lossless image compression remains an important problem. Standard lossless image compression codecs like the H.265/HEVC video codec operating in intra-frame lossless mode [62] (HEVC Intra Lossless) and classical context-based adaptive techniques such as CALIC [60], [61] were significantly outperformed by neural-network-based methods, such as convolutional neural networks (CNNs) and deep autoregressive models, that are effective in capturing complex spatial dependencies [8].

The work presented in [8] is considered the current state-of-the-art for lossless image compression. In it, a lossless pipeline is proposed that couples a classical spatial predictor with a CNN-based residual refiner and a CALIC-style context model feeding a context-based bit-plane (CBP) arithmetic coder. The causal input to the CNN is a wide stencil and the network outputs an 8-bit residual class only used to refine the classical prediction, preserving exact invertibility. After prediction, the integer residual is modeled by a CALIC-inspired context that combines a local binary pattern with a quantized energy term to form an index. Residuals are mean-shifted per context and remapped using the current predictor output to sharpen entropy modeling. The refined error is then encoded by a CBP codec that first predicts the binary length, signals an under-prediction flag when needed, and arithmetic-codes each bit-plane using adaptive-context trees with periodic count halving.

In popular neural-based schemes for lossy image compression [5], [6], [107] there is often a layer to handle both quantization and probability estimation during the training phase. This is sometimes referred to as the Entropy Bottleneck layer [108], [109]. A main innovation in the Entropy Bottleneck layer is the so-called hyperprior [6], [107], which uses side information to adapt the variances of the neural coefficients to different spatial contexts. The hyperprior is a modern take on decades-old solutions to spatial adaptation of the variances for efficient coding such as [110].

As seen in Chapter 4, the use of neural-context-based arithmetic coding can greatly reduce the final bit-rate. This chapter presents the work developed to improve the use of neural-based context modeling in arithmetic coding algorithms for lossless image compression. Here, we propose two neural networks, one for prediction and one similar to a hyperprior network's that estimate error variance, but not requiring side information. Specifically, given a causal context, a prediction network outputs real-valued predictions of the integer pixel to be encoded. The prediction error, or residual, is then quantized and entropy coded. The probability of the quantized residual is estimated by integrating a zero-mean Laplacian density over the quantization bin. The scale of the zero-mean Laplacian density is provided by a second network, given the same causal context, making it adaptive. The probability derived from the density drives an adaptive arithmetic coder. The

networks are jointly trained to minimize the bit-rate of the arithmetic coder. We show that this coding method is equivalent to directly entropy coding the integerized pixel with a shifted Laplacian distribution whose shift and scale are given by the prediction and scale networks. This approach easily extends to other distributions with more parameters.

Our lossless compression approach significantly reduces bit-stream sizes compared to traditional lossless image compression methods [62], [63] and to more recently proposed deep-learning-based ones [8]. Indeed, experimental validation is carried out on the Ultra Video Group (UVG) dataset [111] and results show that our approach consistently outperforms reference methods such as HEVC Intra Lossless, CALIC, and the state-of-the-art [8].

At this stage, the proposed framework is fully developed and evaluated only for lossless image compression. A natural next step is to adapt the same prediction–scale architecture and Laplacian residual modeling to lossy image compression, integrating appropriate quantization and distortion-control mechanisms. Once such a lossy image coder is established, the framework can be further extended to point cloud attribute compression, where the neural-based context modeling and adaptive entropy coding would operate on transformed or directly predicted attribute coefficients, enabling a unified treatment of both 2D images and 3D point cloud attributes.

6.1 PROPOSED CODER

During the development of the lossless coding based on the two different neural networks, we were faced with the possibility of two alternatives regarding the training method. In the first, the networks were sequentially trained and not trained to minimize the same objective. With the second alternative, both networks are trained simultaneously and to minimize the same objective (in this case the mean squared error). The latter presented slightly better results; both methods are described in the following sections along with their results.

6.1.1 ENTROPY BOTTLENECK

In neural lossy compression of images, the entropy bottleneck [108], [109] consists of a neural network layer essential to the inner workings of many autoencoders. During training, it enables the optimization of the encoder parameters by replacing quantization with additive uniform noise. During inference, it performs actual quantization. It is also responsible for learning the prior distribution of the latent representation, which, during inference, feeds an entropy coder to produce the compressed bit-stream and, during training, is used to calculate the rate. The prior distribution is learned from the latent variables using the method described in [6].

We propose an entropy-bottleneck equivalent for predictive lossless coding. That is, we propose a module capable of handling both quantization and probability estimation, to be seamlessly coupled to a simple neural network during both training and inference. It naturally incorporates

conditional probabilities.

6.1.2 SEQUENTIAL TRAINING METHOD

Let x_{ij} denote the pixel in the i-th row and j-th column of a b-bit monochrome image. We associate x_{ij} with its causal context C_{ij} made of the N closest pixels previously encoded. A causal context means that we only account for pixels that have been previously encoded and, thus, are also available to the decoder. C_{ij} only includes the N closest causal pixels in a geometric sense, assuming square pixels. If some of the pixels included by C_{ij} are outside the image boundaries, they are assumed to be a fixed constant value.

In order to use a b-bit pixel with neural networks, its range, from 0 to $(2^b - 1)$, is mapped to a normalized $[0, 1]$ range by dividing by $(2^b - 1)$. The network output (be it a prediction value or a standard deviation) is, then, de-normalized by multiplying it by $(2^b - 1)$.

We train and apply two neural network models: the predictor model and the variance model.

The predictor model takes as input the context C_{ij} and outputs a prediction y_{ij} to the current pixel x_{ij}. It is trained to minimize the mean squared error between its predictions $\{y_{ij}\}$ and the actual pixel values $\{x_{ij}\}$.

The prediction value is rounded to the nearest integer and a residual, or prediction error, is computed as:

$$r_{ij} = x_{ij} - y_{ij}. \tag{6.1}$$

It is r_{ij} that is encoded and transmitted. It ranges from $-(2^b-1)$ to (2^b-1). In order to efficiently encode the residual, one may accurately estimate its symbol probabilities. Figure 6.1 shows the empirical distribution of the quantized residuals for a typical image, showing that the prediction errors indeed follow closely a zero-mean Laplacian distribution [112], [113]. If we assume a zero-mean Laplacian distribution, we still need to estimate its variance. For that, we use the variance model.

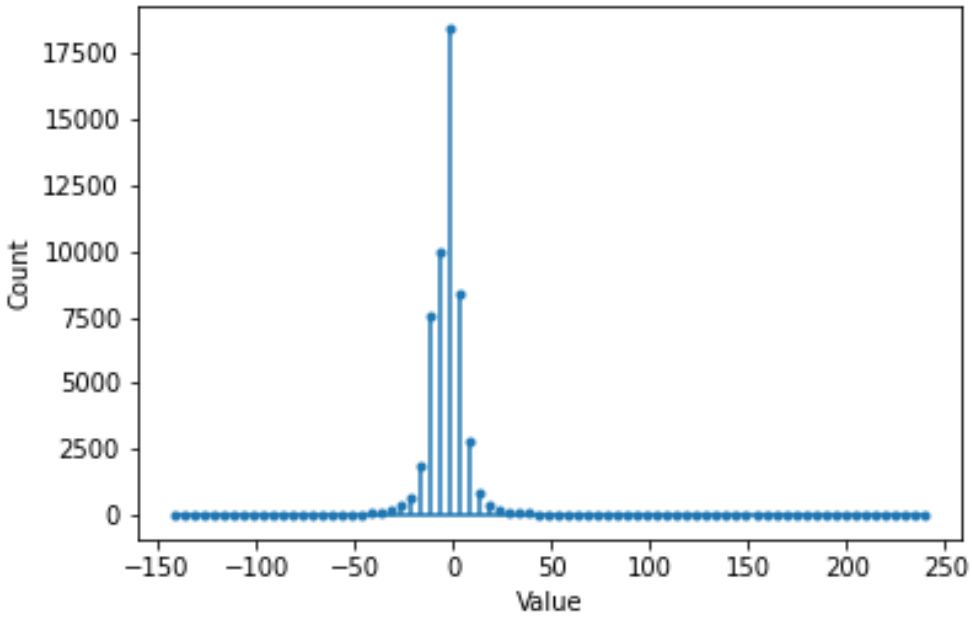


Figure 6.1: Residual distribution for a 512×512 image, with each pixel predicted using a convolutional neural network.

The variance model takes C_{ij} as input and it outputs the variance (standard deviation, σ) of the assumed Laplacian distribution. We train the variance model by minimizing the expected rate of an adaptive arithmetic coder, which derives from the predicted variance, as in [47], [114]. The distribution of the (continuous) residual $\hat{r}$ for a given context is assumed to be Laplacian:

$$p(\hat{r}) = \frac{1}{\sqrt{2}\,\sigma}\,\exp\!\left(-\frac{\sqrt{2}\,|\hat{r}|}{\sigma}\right). \tag{6.2}$$

After quantization with a step size $Q = 1$, the probability of a residual r is

$$p_r = \int_{r-\frac{1}{2}}^{r+\frac{1}{2}} \frac{1}{\sqrt{2}\,\sigma}\,\exp\!\left(-\frac{\sqrt{2}\,|\lambda|}{\sigma}\right) d\lambda, \tag{6.3}$$

yielding code length $-\log_2(p_r)$ bits. Note that r is an integer in $[-(2^b-1), 2^b-1]$. Under the standard high-rate, low-distortion assumption $Q \ll \sqrt{2}\,\sigma$, this simplifies to [47], [114]:

$$-\log_2(p_r) \;\approx\; \frac{1}{\ln 2}\left(\frac{\sqrt{2}\,|r|}{\sigma} \;+\; \ln\!\left(\sqrt{2}\,\sigma\right)\right). \tag{6.4}$$

which is a rough estimate of the bit-rate a perfect arithmetic coder would achieve, and serves as the loss function for training the variance model. What the model aims to minimize is the total bit-rate: $-\sum_{ij}\log_2(p_{r_{ij}})$. Figure 6.2 illustrates the complete algorithm of the proposed method.

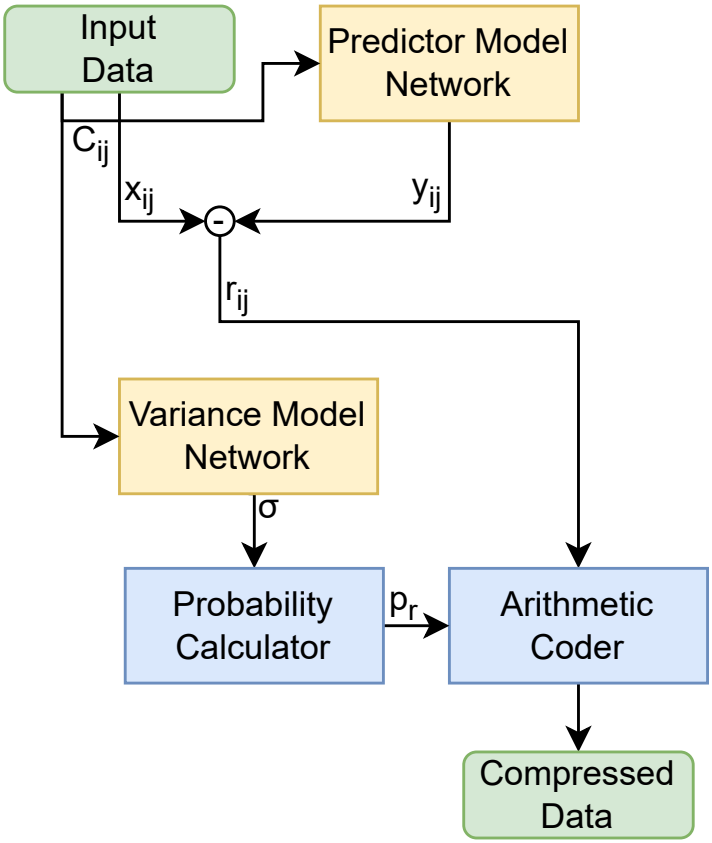


Figure 6.2: Diagram illustrating the proposed coding algorithm, where C_{ij} represents the causal context, x_{ij} represents the pixel to be coded, y_{ij} is the neural prediction of the pixel, r_{ij} the residual, σ is the neural estimated variance and p_r is the probability distribution.

The predictor and the variance models can be trained either sequentially or simultaneously. The only difference is whether the residual that is used to train the variance model is obtained from the final or current state of the predictor network. Either case can lead to convergence. The second option is closer to how the Entropy Bottleneck of [6] operates.

The compression operation, for pixel x_{ij}, is, as follows:

- Use C_{ij} to feed the predictor model network, in order to output y_{ij}.
- Calculate residual $r_{ij} = x_{ij} - y_{ij}$.
- Feed C_{ij} to the variance model network in order to output σ.
- For that σ, calculate the probability distribution p_r for all r in the range.
- Encode r_{ij} using an arithmetic coder and the probability distribution p_r.

In order to decode pixel x_{ij} one should do as follows:

- Feed C_{ij} to the variance model network in order to output σ.
- For that σ, calculate the probability distribution p_r for all r in the range.
- Decode r_{ij} using an arithmetic decoder and the probability distribution p_r.
- Use C_{ij} to feed the predictor model network, in order to output y_{ij}.
- Calculate $x_{ij} = r_{ij} + y_{ij}$.

6.1.3 SIMULTANEOUSLY TRAINING METHOD

We let x_{ij} denote the value of the pixel in the i-th row and j-th column of a b-bit monochrome image. We assume that x_{ij} lies in the range $[0, 2^b)$, but is truncated to an integer $k_{ij} = \lfloor x_{ij} \rfloor$ in the range $\{0, \ldots, 2^{b-1}\}$. The problem is to encode and transmit k_{ij} losslessly for all ij.

We let C_{ij} be the causal context of x_{ij} comprising its N closest causal pixels. We assume the integer pixels in C_{ij} have been previously encoded and are thus available to the decoder prior to decoding k_{ij}.

We let $\mu_{ij} = \mu(C_{ij})$ be a real-valued prediction of x_{ij} based on the context C_{ij}. We implement the function $\mu : C_{ij} \mapsto \mu_{ij}$ with a neural network.

We let r_{ij} be the prediction error, or residual, where

$$r_{ij} = x_{ij} - \mu_{ij}. \tag{6.5}$$

We model the distribution of r_{ij} as a zero-mean Laplacian with density $f(r; 0, \beta_{ij})$, where

$$f(r; 0, \beta) = \frac{1}{2\beta} \exp\left(-\frac{|r|}{\beta}\right) \tag{6.6}$$

and $\beta_{ij} = \beta(C_{ij})$ is *scale* of the zero-mean Laplacian distribution (i.e., β_{ij} is also known as the *mean absolute deviation* or the *standard deviation* (divided by $\sqrt{2}$) of the Laplacian distribution). We also implement the function $\beta : C_{ij} \mapsto \beta_{ij}$ with a neural network. Note that $\beta_{ij} = \sigma_{ij}/\sqrt{2}$ is also proportional to the standard deviation $\sigma_{ij} = \sqrt{E[|r_{ij}|^2]}$ of the distribution.

Equivalently, we model the distribution of x_{ij} as a shifted Laplacian with density $f(x; \mu_{ij}, \beta_{ij})$, where

$$f(x; \mu, \beta) = \frac{1}{2\beta} \exp\left(-\frac{|x - \mu|}{\beta}\right) \tag{6.7}$$

and μ_{ij} is the *mean* of the Laplacian distribution (i.e., μ_{ij} is also known as the *shift* or *center* of the distribution). Thus μ_{ij} may be interpreted either as the prediction of x_{ij} or as the mean of the Laplacian distribution that models the density of x_{ij}. Likewise β_{ij} may be interpreted either as the scale of the zero-mean Laplacian distribution of r_{ij} or as the scale of the mean-μ_{ij} Laplacian distribution of x_{ij}.

Using an arithmetic coder, we transmit $k_{ij} = \lfloor x_{ij} \rfloor$ using $-\log_2 p(k_{ij}; \mu_{ij}, \beta_{ij})$ bits, where $p(k; \mu, \beta)$ is the integral of the density (6.7) over bin k,

$$p(k; \mu, \beta) = \int_k^{k+1} \frac{1}{2\beta} \exp\Big(-\tfrac{|x-\mu|}{\beta}\Big) dx \tag{6.8}$$

$$= F(k+1; \mu, \beta) - F(k; \mu, \beta), \tag{6.9}$$

where $F(x; \mu, \beta)$ is the cumulative distribution function

$$F(x; \mu, \beta) = \int_{-\infty}^{x} f(t; \mu, \beta) dt \tag{6.10}$$

$$= \begin{cases} \frac{1}{2} \exp(-(x-\mu)/\beta) & \text{if } x \geq \mu \\ 1 - \frac{1}{2} \exp((\mu - x)/\beta) & \text{if } x \leq \mu \end{cases}. \tag{6.11}$$

Thus, we have

$$-\log_2 p(k; \mu, \beta) \tag{6.12}$$

$$= -\log_2 (F(k+1; \mu, \beta) - F(k; \mu, \beta)) \tag{6.13}$$

$$\approx \frac{1}{\ln 2} \Big(\tfrac{|k+1/2-\mu|}{\beta} + \ln(2\beta) \Big). \tag{6.14}$$

Our objective is to minimize the total number of bits

$$R = -\sum_{ij} \log_2 p(k_{ij}; \mu(C_{ij}), \beta(C_{ij})). \tag{6.15}$$

We therefore jointly train the prediction network $\mu(C)$ and the scale network $\beta(C)$ to minimize (6.15) over a representative training set. For the purposes of training, one may use the approximation (6.14) if $\beta \gg 1$, but the exact expression (6.13) is preferred. Note that (6.13) is still differentiable almost everywhere (as is ReLU, for example). Thus we may train the networks either simultaneously or alternately using back-propagation and gradient descent through the loss function (6.15). In order to use b-bit pixels with neural networks, we normalize the range of the inputs and outputs from $\{0, \ldots, 2^b - 1\}$ to $[0, 1]$. That is, $\mu(C) = 2^{b-1} \bar{\mu}(C/2^{b-1})$ and $\beta(C) = 2^{b-1} \bar{\beta}(C/2^{b-1})$ and we train $\bar{\mu}$ and $\bar{\beta}$.

Thus, the predictor network $\mu(C)$ and the scale network $\beta(C)$ can be regarded as networks that determine the parameters of the density used to code x_{ij}. From this point of view, the encoder and decoder communicate k_{ij} directly.

In an alternative and equivalent point of view, the encoder and decoder may independently determine the prediction $\mu_{ij} = \mu(C)$, and then communicate the quantized prediction residual

$$\hat{r}_{ij} = k_{ij} - \mu_{ij}. \tag{6.16}$$

Specifically, the encoder may

1. use the predictor network to predict x_{ij} as $\mu_{ij} = \mu(C_{ij})$,

2. subtract μ_{ij} from $k_{ij} = \lfloor x_{ij} \rfloor = \lfloor \mu_{ij} + r_{ij} \rfloor$ to obtain the dithered[1] quantized residual $\hat{r}_{ij} = \lfloor \mu_{ij} + r_{ij} \rfloor - \mu_{ij}$,
3. transmit $\hat{r}_{ij}$ using $-\log_2 p(\hat{r}_{ij}; 0, \beta_{ij})$ bits, where

$$p(\hat{r}; 0, \beta) = F(\hat{r} + 1; 0, \beta) - F(\hat{r}; 0, \beta) \tag{6.17}$$

and $\beta_{ij} = \beta(C_{ij})$. Then, the decoder may

1. again use the predictor network to predict x_{ij} as $\mu_{ij} = \mu(C_{ij})$
2. decode $\hat{r}_{ij}$ using the integral of the zero-mean Laplacian density (6.17), where the scale parameter is determined by the scale network $\beta_{ij} = \beta(C_{ij})$,
3. recover $k_{ij} = \mu_{ij} + \hat{r}_{ij}$.

These two approaches are equivalent because they use equivalent codes produced by identical probabilities:

$$\begin{aligned} p(\hat{r}; 0, \beta) &= \int_{\hat{r}}^{\hat{r}+1} \frac{1}{2\beta} \exp\left(-\tfrac{|r|}{\beta}\right) dr && (6.18)\\ &= \int_{\hat{r}+\mu}^{\hat{r}+\mu+1} \frac{1}{2\beta} \exp\left(-\tfrac{|x-\mu|}{\beta}\right) dx && (6.19)\\ &= \int_{k}^{k+1} \frac{1}{2\beta} \exp\left(-\tfrac{|x-\mu|}{\beta}\right) dx && (6.20)\\ &= p(k; \mu, \beta). && (6.21) \end{aligned}$$

Dithering the quantization of r_{ij}, so that $\hat{r} = k - \mu$ (whereas $r = x - \mu$), is what makes these exact. Note that $\hat{r}$ is not generally an integer. Figure 6.3 illustrates the proposed equivalent coding systems. Our approach may be easily extended to distributions with more parameters, for example the generalized Gaussian distribution.

[1] A *dithered* quantizer is one that adds a value before quantization and subtracts it afterwards, effectively shifting the quantization bin.

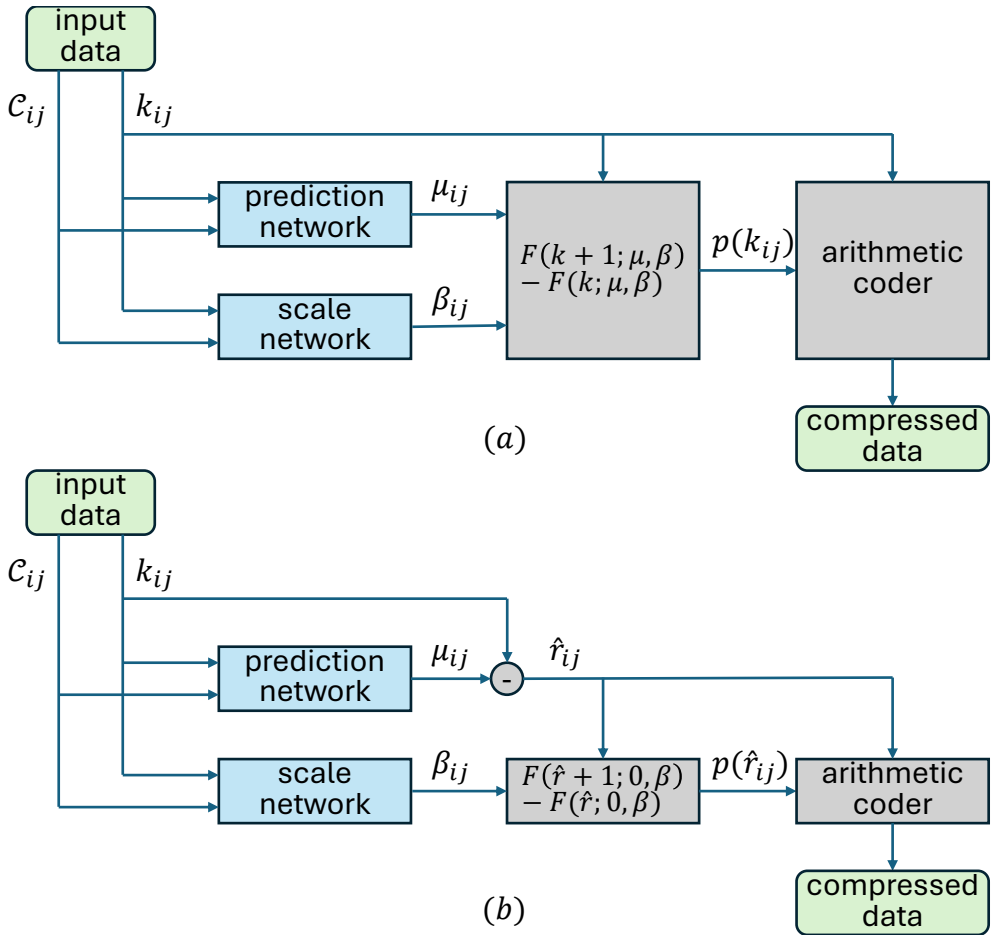


Figure 6.3: Diagram illustrating the proposed equivalent coding systems, where (a) prediction and scale networks output parameters of Laplacian(μ_{ij}, β_{ij}) distribution, which drives the arithmetic coder to code the pixel value k_{ij} directly, and (b) prediction network outputs prediction μ_{ij}, and scale network produces parameter of Laplacian$(0, \beta_{ij})$ distribution, which drives the arithmetic coder to code the prediction residual $\hat{r}_{ij}$.

Where C_{ij} represents the causal context, k_{ij} represents the integer value of the pixel to be coded, μ_{ij} is the neural prediction of the pixel, $\hat{r}_{ij}$ the prediction residual, β_{ij} is the neural estimate of the scale, and $p(k_{ij})$ and $p(\hat{r}_{ij})$ are the estimated probabilities of k_{ij} and $\hat{r}_{ij}$, respectively.

6.1.4 NETWORK ARCHITECTURES

In both training methods described before, we implemented both the predictor and the variance models using three different neural network architectures: a lightweight Convolutional Neural Network (CNN), a Multi-Layer Perceptron (MLP), and a lightweight Transformer network. For all network architectures, the prediction and scale networks were alternately trained.

The CNN baseline maps a single-channel context patch of spatial size $(N+2) \times (N+2)$ to a scalar output. It consists of two 3×3 convolutional layers with padding 1, increasing the number of channels from 1 to 16 and then to 32, each followed by a ReLU nonlinearity. The resulting feature map is flattened and passed through a fully connected layer of size $32(N+2)^2 \rightarrow 64$ with ReLU,

followed by a final linear layer of size $64 \to 1$. Including biases, this architecture contains

$$160 + 4640 + \left(32(N+2)^2 \cdot 64 + 64\right) + 65 = 2048(N+2)^2 + 4929 \tag{6.22}$$

trainable parameters, which yields 299 841 parameters for $N = 10$.

The MLP architecture operates on an N-dimensional causal context vector and consists of three affine layers with widths $N \to 64N \to 32N \to 1$, with ReLU activations after the first two layers and a final sigmoid nonlinearity. Including biases, the total number of trainable parameters is

$$(N \cdot 64N + 64N) + (64N \cdot 32N + 32N) + (32N \cdot 1 + 1) = 2112N^2 + 128N + 1, \tag{6.23}$$

which corresponds to 212 481 parameters for $N = 10$.

The Transformer-based network maps a sequence of N scalar pixel values, quantized to a 256-level grayscale alphabet, to a scalar output in $(0, 1)$ that parameterizes the modeled distribution. Each input token is embedded into a 20-dimensional space via the sum of learned token embeddings (of size 256×20) and positional embeddings (of size $N \times 20$). The embedded sequence is processed by a single transformer block comprising a 2-head self-attention layer with model dimension 20 and a position-wise feed-forward module with dimensions $20 \to 40 \to 20$, with both sub-layers equipped with residual connections, layer normalization, and dropout with probability 0.13. The output sequence is then averaged over the temporal dimension and fed to a classifier head implemented as a three-layer perceptron with dimensions $20 \to 20 \to 20 \to 1$, LeakyReLU nonlinearities, dropout, and a final sigmoid activation. For this configuration, the total number of trainable parameters is

$$20N + 9521, \tag{6.24}$$

i.e., 10 521 parameters for $N = 50$.

The CNN and MLP architectures thus have comparable capacity for the same N, while the Transformer variant is deliberately kept lightweight. Additionally, all networks use a causal context of up to $N = 50$ pixels. Each architecture is instantiated twice, one instance for pixel prediction and another for scale estimation. Both networks are separate, but share their input, the causal context.

6.2 EXPERIMENTAL RESULTS

In our tests, we used the Ultra Video Group (UVG) dataset [111], which is made of 16 versatile 4K (3840 x 2160) 8-bit video sequences. For our training, we used 9 sequences from the dataset, while the other 7 are used for testing. The training sequences were CityAlley, FlowerFocus, FlowerKids, FlowerPan, Lips, RaceNight, RiverBank, SunBath and Twilight. Such a dataset was meant to coincide with that used in [8].

All experiments were run on a single NVIDIA GPU (GTX 3080) and the memory footprint of our models is sufficiently small to allow training on most modern GPUs. They are intentionally kept shallow and light to approximate real-time performance for high-resolution images.

Even though we are using video sequences to run our tests, there is no inter-frame prediction of any kind. All the tests were run on the luminance channel. The experimental evaluation compares our coder performance to: (*i*) the HEVC Intra Lossless coder [62] within the x265 implementation, operating with the "veryslow" lossless-mode preset, and only using intra prediction; (*ii*) FLIF [63]; (*iii*) CALIC [61]; and (*iv*) CBPNNv under the SLOW profile [8].

Table 6.1: Bit-rate results for the method described in section 6.1.3. Bit-rate savings (in %) against HEVC Intra Lossless also indicated.

	State-of-the-art codecs				**Proposed method**		
Video Sequence	**HEVC Intra Lossless**	**FLIF**	**CALIC**	**CBPNNv**	**CNN**	MLP	Transformer
Beauty	3.76	3.45 (8.14%)	3.42 (8.91%)	3.40 (9.61%)	2.22 (41.06%)	**2.08 (44.56%)**	3.39 (9.81%)
Bosphorus	3.12	2.48 (20.65%)	2.43 (22.15%)	2.38 (23.73%)	2.90 (7.08%)	**2.37 (23.91%)**	2.92 (6.45%)
HoneyBee	3.57	3.01 (15.62%)	2.97 (16.69%)	2.94 (17.44%)	2.70 (24.25%)	**2.59 (27.48%)**	2.94 (17.63%)
Jockey	3.18	2.76 (13.10%)	2.75 (13.39%)	2.71 (14.80%)	2.51 (20.87%)	**2.38 (25.15%)**	2.73 (13.94%)
ReadySteadyGo	3.47	2.75 (20.70%)	2.68 (22.72%)	2.62 (24.62%)	2.85 (17.77%)	**2.33 (32.82%)**	2.63 (24.24%)
ShakeNDry	4.16	3.23 (22.31%)	3.15 (24.28%)	3.15 (24.25%)	2.67 (35.80%)	**2.65 (36.22%)**	3.07 (26.19%)
YachtRide	3.41	2.65 (22.19%)	2.56 (24.89%)	**2.51 (26.53%)**	3.14 (7.99%)	2.59 (23.94%)	3.18 (6.88%)
Average bpp	*3.52*	*2.90 (17.53%)*	*2.85 (19.00%)*	*2.81 (20.14%)*	*2.71 (22.12%)*	**2.43 (30.58%)**	2.98 (15.02%)

Table 6.2: Bit-rate results for the method described in section 6.1.2. Bit-rate savings (in %) against HEVC Intra Lossless also indicated.

	State-of-the-art codecs				**Proposed method**		
Video Sequence	**HEVC Intra Lossless**	**FLIF**	**CALIC**	**CBPNNv**	**CNN**	MLP	Transformer
Beauty	3.76	3.45 (8.14%)	3.42 (8.91%)	3.40 (9.61%)	2.23 (9.61%)	**2.09 (44.29%)**	3.41 (9.38%)
Bosphorus	3.12	2.48 (20.65%)	2.43 (22.15%)	**2.38 (23.73%)**	2.97 (23.73%)	2.43 (22.02%)	2.99 (4.13%)
HoneyBee	3.57	3.01 (15.62%)	2.97 (16.69%)	2.94 (17.44%)	2.72 (17.44%)	**2.61 (26.92%)**	2.96 (16.99%)
Jockey	3.18	2.76 (13.10%)	2.75 (13.39%)	2.71 (14.80%)	2.51 (14.80%)	**2.38 (24.15%)**	2.73 (13.94%)
ReadySteadyGo	3.47	2.75 (20.70%)	2.68 (22.72%)	2.62 (24.62%)	2.88 (24.62%)	**2.35 (32.27%)**	2.65 (23.62%)
ShakeNDry	4.16	3.23 (22.31%)	3.15 (24.28%)	3.15 (24.25%)	2.67 (24.25%)	**2.65 (36.13%)**	3.07 (26.08%)
YachtRide	3.41	2.65 (22.19%)	2.56 (24.89%)	**2.51 (26.53%)**	3.32 (26.53%)	2.74 (19.65%)	3.36 (1.63%)
Average bpp	*3.52*	*2.90 (17.53%)*	*2.85 (19.00%)*	*2.81 (20.14%)*	*2.76 (20.81%)*	**2.46 (29.49%)**	3.02 (13.68%)

Tables 6.1 and 6.2 show the results, where we can compare the proposed method against the above-mentioned state-of-the-art methods. Along with the rate, we also quote the percentage of bit-rate savings compared to the HEVC Intra Lossless method. The improvement is denoted Δ_{CR} and it is calculated as:

$$\Delta_{CR} = 1 - \frac{bpp_{method}}{bpp_{HEVC}} \tag{6.25}$$

Note that all the three proposed architectures outperformed the HEVC Intra Lossless, in either of the training methods. The MLP architecture, however, is the one that yields the best results. The proposed method using the MLP architecture with the simultaneously training method outperforms the HEVC Intra Lossless on average by 30.58%. It also outperforms the CBPNNv method on average by 13.71%. Hence, we believe that the method proposed using MLP is the current state-of-the-art in lossless image compression.

As expected, the second training method performed slightly better than the first one. This is because it is more efficient if both networks are trained to minimize the same objective, in this case

the bit-rate, instead of the prediction network being trained to minimize the MSE and the scale network trained to minimize the bit-rate.

6.3 CONCLUSIONS

This work presents a novel approach to lossless image coding by introducing a neural-network-based context modeling framework that uses both the mean and scale of the prediction error to improve the arithmetic coding stage of the final bit-stream. Its design avoids side information while retaining the benefits of learned, spatially varying scale.

We show that our approach is equivalent to using neural networks to select the parameters of a probability distribution over the input symbol for each context. As such, our approach extends easily to probability distributions with multiple parameters.

We also presented a comparison between different network architectures to determine the best suited architecture for a shallow and light neural network to facilitate real-time or near-real-time performance. Experimental results indicate that the proposed method outperforms the most recent state-of-the-art method [8] on their tested dataset.

We plan to further investigate the use of residual distribution estimation to improve the arithmetic coding stage in lossy image compression. We may also extend to other media such as videos and point clouds, as well as to distributions with multiple parameters.

7 CONCLUSION

"Veni, vidi, vici."

- Caius Julius Caesar

In this thesis, we have proposed and investigated several methods for efficient encoding of point cloud attributes, emphasizing the importance of embedded (progressive) bit-streams and adaptive context modeling. These approaches aim to address the growing need to transmit and store large-scale three-dimensional data, which has become increasingly common in applications such as virtual, augmented, and mixed reality systems. Below, we summarize the main contributions of this thesis, discuss their limitations, and outline potential avenues for future research.

7.1 SUMMARY OF CONTRIBUTIONS

1. Embedded Attribute Coding of Point Clouds (Chapters 3, 4, and 5)

    - **SPIHT for RAHT Coefficients:** We introduced the first fully embedded point cloud coder by adapting Set Partitioning in Hierarchical Trees (SPIHT) to encode the Region-Adaptive Hierarchical Transform (RAHT) coefficients. This approach provides a single bit-stream that can be truncated at any point to trade off bit-rate for quality, thus enabling true scalability. Experimental results showed that SPIHT-based encoding can outperform the conventional Run-Length Golomb-Rice (RLGR) method while offering embedded functionality.
    - **MLP-based Context Modeling for SPIHT:** To further improve the SPIHT-based encoder, we employed a Multi-Layer Perceptron (MLP) to model conditional probabilities of the RAHT coefficients. By exploiting local context (e.g., ancestor nodes in the SPIHT tree), this neural network guides an adaptive arithmetic coder, resulting in higher compression efficiency. Our experiments demonstrated consistent improvements over the baseline SPIHT approach.
    - **Region-of-Interest Coding:** We extended the embedded approach to support region-of-interest (ROI) compression, wherein certain spatial regions of the point cloud are assigned higher fidelity during encoding. The proposed method interleaves bits associated with ROI and non-ROI voxels in a single embedded stream, allowing seamless control of quality allocation. Results highlighted a marked improvement in ROI reconstruction fidelity, although at the expense of slightly reduced quality outside the ROI.

2. Entropy-Bottleneck Layer for Context and Neural-Based Lossless Compression (Chapter 6)

    - **Context-Conditioned Entropy Modeling:** We introduce a context modeling framework that predicts both the mean and the variance of pixel intensities and uses these statistics as a scale hyperprior to parameterize the latent distribution. By supplying the

arithmetic coder with more accurate, context-conditioned probabilities, the method reduces the entropy of the residual signal and improves coding efficiency without requiring pre-stored codebooks or source-specific training. On the evaluated dataset, the proposed context-based lossless image coding scheme achieves average bit-rate reductions over a recent state-of-the-art baseline [8]. While ablation studies are possible, the modular design suggests clear extensions: (i) to lossy operating points by relaxing the distortion metric, and (ii) to lossless point cloud *attribute* compression by conditioning mean/variance estimates on geometric or voxelized contexts, thereby retaining the benefits of context-aware entropy modeling in 3D settings.

7.2 LIMITATIONS

Despite demonstrating promising results, the methods presented in this thesis share a few common limitations:

- **Performance Gap in Non-Embedded Settings:** Although the proposed approaches achieve embedded (progressive) encoding, they do not yet match the performance of state-of-the-art non-embedded algorithms that often rely on heavy prediction tools.
- **Neural Network Complexity:** The adoption of MLPs or more complex architectures (e.g., convolutional neural networks) introduces a computational overhead, which might be non-trivial for large-scale point clouds or real-time applications.
- **ROI vs. Global Quality Trade-off:** While ROI-based methods successfully allocate higher quality to regions of interest, they may degrade non-ROI regions. Determining the optimal interleaving or weighting remains an open challenge that depends on application-specific requirements.
- **Context Modeling in a 3D Context:** The context-based approach was demonstrated primarily on image data. Extending it to three-dimensional (or temporal) attributes demands careful design of the context space and may require specialized network architectures.

7.3 FUTURE WORK

Several promising directions emerge for extending this research:

- **Advanced Network Architectures:** Incorporating more sophisticated neural networks (e.g., graph neural networks, transformers for 3D data) could capture richer geometric relationships in point clouds, potentially leading to higher compression gains.

- **Efficient Hardware Implementations:** Future work could focus on optimizing the proposed methods for energy-efficient hardware or parallelizable structures. This is especially relevant given the size and throughput demands of modern 3D capture systems.
- **Refining ROI Strategies:** Enhanced ROI detection algorithms or adaptive ROI shape encoding schemes might yield more flexible and nuanced quality allocation across point cloud regions.
- **Neural-Context-Based PCC:** Building on the success of context modeling in 2D image compression, the integration of neural-context-based networks into a full 3D pipeline (for both geometry and attribute coding) remains a crucial next step.
- **Combining Embedded and Non-Embedded Features:** A hybrid solution where specific layers (bit-planes) are embedded while others use more aggressive prediction could strike a balance between maximum compression efficiency and partial scalability.

7.4 FINAL REMARKS

The rapid growth in immersive applications, combined with emerging consumer-grade 3D capture devices, continues to fuel the demand for robust and scalable point cloud compression solutions. The contributions in this thesis—ranging from SPIHT-based embedded coding to neural network-driven context modeling—demonstrate that embedded compression of point cloud attributes is both feasible and beneficial. By providing a single bit-stream that can be truncated at will, these methods address scenarios where bandwidth or storage constraints vary dynamically over time.

Although further improvements are needed to rival the performance of non-embedded specialized schemes, the embedded approaches discussed here emphasize flexibility and scalability, which are often decisive factors in real-world systems. Taken together, this work paves the way for future research in hybrid or purely neural compression strategies, including hyperprior-based methods that promise improved adaptation to point cloud data. Ultimately, these advances bring us closer to truly efficient and adaptive point cloud communication, enabling a broader range of interactive and immersive applications in virtual, augmented, and mixed realities.

REFERENCES

[1] C. Loop, Q. Cai, S. Escolano, and P. Chou, "Microsoft Voxelized Upper Bodies - A Voxelized Point Cloud Dataset," ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG), input document m38673/M72012, May 2016.

[2] 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), Geneva, input document m40059/M74006, January 2017.

[3] M. Krivokuća, P. A. Chou, and P. Savill, "8i voxelized surface light field (8iVSLF) dataset," ISO/IEC JTC1/SC29/WG11 MPEG, input document m42914, Jul. 2018.

[4] Y. Xu, Y. Lu, , and Z. Wen, "Owlii Dynamic human mesh sequence dataset," ISO/IEC JTC1/SC29/WG11 MPEG, Macau, input document m41658, Oct. 2017.

[5] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimized image compression," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[6] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational image compression with a hyperprior," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[7] D. Minnen, J. Ballé, and G. D. Toderici, "Joint autoregressive and hierarchical priors for learned image compression," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/file/53edebc543333dfbf7c5933af792c9c4-Paper.pdf

[8] I. Schiopu and A. Munteanu, "Deep-learning-based lossless image coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 7, pp. 1829–1842, 2020.

[9] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto, "Learned image compression with discretized gaussian mixture likelihoods and attention modules," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 7936–7945.

[10] P. Milgram and H. Colquhoun, "A taxonomy of real and virtual world display integration," Jan. 2001.

[11] P. Milgram, "A taxonomy of (real and virtual world) display and control interactions," in *Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology*, ser. VRST '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 10. [Online]. Available: https://doi.org/10.1145/1643928.1643932

[12] S. Schwarz, M. Preda, V. Baroncini, M. Budagavi, P. Cesar, P. A. Chou, R. A. Cohen, M. Krivokuća, S. Lasserre, Z. Li, J. Llach, K. Mammou, R. Mekuria, O. Nakagami, E. Siahaan, A. Tabatabai, A. M. Tourapis, and V. Zakharchenko, "Emerging MPEG Standards for Point Cloud Compression," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 1, pp. 133–148, 2019.

[13] MPEG, "MPEG121 version of MPEG standardisation roadmap," ISO/IEC JTC1/SC29/WG11 MPEG, output document N17332, ISO/IEC MPEG JTC1/SC29/WG11, Tech. Rep. N17332, January 2017.

[14] 3DG, "MPEG 3DG and Requirements: Call for Proposals for Point Cloud Compression v2," ISO/IEC MPEG JTC1/SC29/WG11, Hobart, AU, Approved WG 11 document N16763, April 2017.

[15] C. Cao, M. Preda, and T. Zaharia, "3d point cloud compression: A survey," in *The 24th International Conference on 3D Web Technology*, ser. Web3D '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1–9. [Online]. Available: https://doi.org/10.1145/3329714.3338130

[16] 3DG, "G-PCC Codec Description v12," ISO/IEC MPEG JTC1/SC29/WG11, Geneva, CH, Approved WG 11 document N18891, October 2020.

[17] D. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, and A. Tabatabai, "An overview of ongoing point cloud compression standardization activities: video-based (V-PCC) and geometry-based (G-PCC)," *APSIPA Transactions on Signal and Information Processing*, vol. 9, p. e13, 2020.

[18] H. Liu, H. Yuan, Q. Liu, J. Hou, and J. Liu, "A comprehensive study and comparison of core technologies for mpeg 3-d point cloud compression," *IEEE Transactions on Broadcasting*, vol. 66, no. 3, pp. 701–717, 2020.

[19] 3DG, "PCC Requirements," ISO/IEC MPEG JTC1/SC29/WG11, Gwangju, KR, Approved WG 11 document W17353, Jan. 2018.

[20] D. C. Garcia, A. Souto, G. Sandri, T. Borges, and R. De Queiroz, "Point cloud reconstruction from truncated geometry-based streams," *Journal of Communication and Information Systems*, vol. 38, pp. 77–84, Jul. 2023.

[21] 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, aug 2016.

[22] B. Kathariya, L. Li, Z. Li, J. Alvarez, and J. Chen, "Scalable point cloud geometry coding with binary tree embedded quadtree," in *2018 IEEE International Conference on Multimedia and Expo (ICME)*, 2018, pp. 1–6.

[23] A. F. R. Guarda, N. M. M. Rodrigues, and F. Pereira, "Point cloud geometry scalable coding with a single end-to-end deep learning model," in *2020 IEEE International Conference on Image Processing (ICIP)*, 2020, pp. 3354–3358.

[24] A. Said and W. A. Pearlman, "A new, fast, and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 243–250, 1996.

[25] A. L. Souto, V. F. Figueiredo, P. A. Chou, and R. L. de Queiroz, "Set partitioning in hierarchical trees for point cloud attribute compression," *IEEE Signal Processing Letters*, vol. 28, pp. 1903–1907, 2021.

[26] V. F. Figueiredo, R. L. de Queiroz, P. A. Chou, and L. S. Lopes, "Embedded coding of point cloud attributes," *IEEE Signal Processing Letters*, vol. 31, pp. 890–893, 2024.

[27] V. F. Figueiredo and R. L. de Queiroz, "Embedded bit-stream region-of-interest coding of point cloud attributes," in *2024 IEEE 26th International Workshop on Multimedia Signal Processing (MMSP)*, 2024, pp. 1–6.

[28] ——, "Codificação progressiva (embedded) de região de interesse para atributos de nuvem de pontos," in *XLII Simpósio Brasileiro de Telecomunicações e Processamento de Sinais (SBrT)*, October 2024.

[29] J. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, pp. 3445–3462, 1993.

[30] V. Ruiz, J. Fernández, and I. García Fernandez, "Image compression for progressive transmission," *The Nineteenth IASTED International Conference on Applied Informatics: Advances in Computer Applications*, 01 2001.

[31] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven layered multicast," in *Conference Proceedings on Applications, Technologies, Architectures, and Protocols for Computer Communications*, ser. SIGCOMM '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 117–130. [Online]. Available: https://doi.org/10.1145/248156.248168

[32] A. Mohr, E. Riskin, and R. Ladner, "Unequal loss protection: Graceful degradation of image quality over packet erasure channels through forward error correction," *Selected Areas in Communications, IEEE Journal on*, vol. 18, pp. 819 – 828, 07 2000.

[33] L. P. Tchapmi, V. Kosaraju, H. Rezatofighi, I. Reid, and S. Savarese, "Topnet: Structural point cloud decoder," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 383–392.

[34] D. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, and A. Tabatabai, "An Overview of Ongoing Point Cloud Compression Standardization Activities: Video-based

(V-PCC) and Geometry-based (G-PCC)," *APSIPA Transactions on Signal and Information Processing*, vol. 9, 2020.

[35] M. Zuffo, "University of São Paulo point cloud dataset." [Online]. Available: http://uspaulopc.di.ubi.pt/

[36] M. Levoy. (2014) The Stanford 3D Scanning Repository. Stanford University Computer Graphics Laboratory. Accessed: 15-10-2020. [Online]. Available: http://graphics.stanford.edu/data/3Dscanrep/

[37] G. Turk and M. Levoy, "Zippered Polygon Meshes from Range Images," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques - SIGGRAPH '94*. ACM Press, 1994.

[38] G. Sandri, R. L. de Queiroz, and P. A. Chou, "Comments on 'Compression of 3D Point Clouds Using a Region-Adaptive Hierarchical Transform'," *ArXiv e-prints, arXiv: 1805.09146v1 [eess.IV]*, May 2018.

[39] Y. LANGSAM, M. AUGENSTEIN, and A. M. TENENBAUM, *Data Structures using C and C++*. New Jersey: Prentice Hall, 1996.

[40] D. Meagher, "Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3D objects by computer," Oct. 1980.

[41] C. Loop, C. Zhang, and Z. Zhang, "Real-time high-resolution sparse voxelization with application to image-based modeling," in *Proceedings of the 5th High-Performance Graphics Conference*, ser. HPG '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 73–79. [Online]. Available: https://doi.org/10.1145/2492045.2492053

[42] R. L. de Queiroz, D. C. Garcia, P. A. Chou, and D. A. Florencio, "Distance-based Probability Model for Octree Coding," *IEEE Signal Processing Letters*, vol. 25, 2018.

[43] G. L. Sandri, "Compression of Point Cloud Attributes," Ph.D. dissertation, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília–DF, Brazil, 2019.

[44] D. Meagher, "Geometric modeling using octree encoding," *Computer Graphics and Image Processing*, vol. 19, no. 2, pp. 129–147, 1982. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0146664X82901046

[45] K. Sayood, *Introduction to Data Compression*, 5th ed., ser. The Morgan Kaufmann Series in Multimedia Information and Systems. Morgan Kaufmann, an imprint of Elsevier, 2017.

[46] D. C. Garcia and R. L. de Queiroz, "Context-based Octree Coding for Point-Cloud Video," in *IEEE International Conference on Image Processing (ICIP)*, 2017, pp. 1412–1416.

[47] 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, August 2016.

[48] H. Malvar, "Adaptive Rrun-Length/Golomb-Rice Encoding of Quantized Generalized Gaussian Sources with Unknown Statistics," in *Data Compression Conference*, April 2006, pp. 23 – 32.

[49] C. Zhang, D. Florêncio, and C. Loop, "Point Cloud Attribute Compression with Graph Transform," in *IEEE International Conference on Image Processing (ICIP)*, October 2014, pp. 2066–2070.

[50] D. Thanou, P. A. Chou, and P. Frossard, "Graph-based motion estimation and compensation for dynamic 3D point cloud compression," in *Proceedings of the IEEE Int'l Conf. Image Processing (ICIP)*, Sept 2015.

[51] D. Thanou, P. A. Chou, and P. Frossard, "Graph-Based Compression of Dynamic 3D Point Cloud Sequences," *IEEE Transactions on Image Processing*, vol. 25, no. 4, pp. 1765–1778, 2016.

[52] E. Pavez and P. A. Chou, "Dynamic polygon cloud compression," in *IEEE Int'l Conf. Acoustics, Speech and Signal Processing (ICASSP)*, March 2017.

[53] E. Pavez, P. A. Chou, R. L. de Queiroz, and A. Ortega, "Dynamic polygon cloud compression," *CoRR*, vol. abs/1610.00402, 2016.

[54] R. A. Cohen, D. Tian, and A. Vetro, "Attribute compression for sparse point clouds using graph transforms," in *IEEE Int'l Conf. Image Processing (ICIP)*, Sept 2016.

[55] P. A. Chou and R. L. de Queiroz, "Gaussian process transforms," in *Proceedings of the IEEE Int'l Conf. Image Processing (ICIP)*, Sept 2016.

[56] R. L. de Queiroz and P. A. Chou, "Transform coding for point clouds using a Gaussian process model," *IEEE Trans. Image Processing*, vol. 26, no. 8, Aug. 2017.

[57] R. Brinkmann, *The Art and Science of Digital Compositing*. Morgan Kaufmann Publishers Inc., 1999, pp. 184–185.

[58] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The jpeg 2000 still image compression standard," *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 36–58, 2001.

[59] H. Hadizadeh and I. V. Bajić, "Saliency-aware video compression," *IEEE Transactions on Image Processing*, vol. 23, no. 1, pp. 19–33, 2014.

[60] X. Wu and N. Memon, "Calic-a context based adaptive lossless image codec," in *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, vol. 4, 1996, pp. 1890–1893 vol. 4.

[61] ——, "Context-based, adaptive, lossless image coding," *IEEE Transactions on Communications*, vol. 45, no. 4, pp. 437–444, 1997.

[62] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[63] J. Sneyers and P. Wuille, "Flif: Free lossless image format based on maniac compression," in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 66–70.

[64] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, Dec. 1989. [Online]. Available: https://doi.org/10.1007/BF02551274

[65] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[66] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[67] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[68] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 652–660.

[69] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 5099–5108.

[70] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6000–6010.

[71] H. He, R. Gao, F. Yan, and S. Ji, "PVCAE: Point-voxel convolutional autoencoder for point cloud compression," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 3, pp. 1161–1174, 2022.

[72] T. Huang, S. Chen, S. Schwarz, and G. Sullivan, "Neural approaches for geometry and attribute compression of dynamic point clouds," *IEEE Transactions on Multimedia*, 2022.

[73] X. Wu, P. Chou, and X. Xue, "Minimum conditional entropy context quantization," in *2000 IEEE International Symposium on Information Theory (Cat. No.00CH37060)*, 2000, pp. 43–.

[74] M. Weinberger, G. Seroussi, and G. Sapiro, "From logo-i to the jpeg-ls standard," in *Proceedings 1999 International Conference on Image Processing (Cat. 99CH36348)*, vol. 4, 1999, pp. 68–72 vol.4.

[75] ——, "The loco-i lossless image compression algorithm: principles and standardization into jpeg-ls," *IEEE Transactions on Image Processing*, vol. 9, no. 8, pp. 1309–1324, 2000.

[76] D. Clunie, "Lossless compression of grayscale medical images - effectiveness of traditional and state of the art approaches," *Proceedings of SPIE*, vol. 3980, 02 2000.

[77] M. Zhou, W. Gao, M. Jiang, and H. Yu, "Hevc lossless coding and improvements," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1839–1843, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:25741668

[78] F. Kamisli, "Lossless compression in hevc with integer-to-integer transforms," in *2016 IEEE 18th International Workshop on Multimedia Signal Processing (MMSP)*, 2016, pp. 1–6.

[79] ——, "Lossless intra coding in hevc with integer-to-integer dst," in *2016 24th European Signal Processing Conference (EUSIPCO)*, 2016, pp. 2440–2444.

[80] S.-W. Hong, J. Kwak, and Y.-L. Lee, "Cross residual transform for lossless intra-coding for hevc," *Signal Processing: Image Communication*, vol. 28, p. 1335–1341, 11 2013.

[81] G. J. Sullivan, "Efficient scalar quantization of exponential and Laplacian random variables," *IEEE Trans. Information Theory*, vol. 42, no. 5, Sep. 1996.

[82] W. A. Pearlman and A. Said, *Digital Signal Compression: Principles and Practice*. Cambridge University Press, 2011.

[83] G. M. Morton, "A computer oriented geodetic data base; and a new technique in file sequencing," IBM, Ottawa, Canada, Technical Report, 1966.

[84] G. Bjontegaard, "Calculation of average PSNR differences between RD curves," in *ITU-T VCEG Meeting*, no. VCEG-M33, Austin, Texas, USA, April 2001.

[85] S. Lasserre and D. Flynn, "On an Improvement of RAHT to Exploit Attribute Correlation," ISO/IEC MPEG JTC1/SC29/WG11, Geneva, CH, Tech. Rep. m47378, March 2019.

[86] ——, "G-PCC CE13.18 report on upsampled transform domain prediction in RAHT," ISO/IEC MPEG JTC1/SC29/WG11, Gothenburg, Sweden, Input document m49380, July 2019.

[87] J. Rissanen and G. G. Langdon, "Arithmetic coding," *IBM Journal of Research and Development*, vol. 23, no. 2, pp. 149–162, 1979.

[88] P. Howard and J. Vitter, "Arithmetic coding for data compression," *Proceedings of the IEEE*, vol. 82, no. 6, pp. 857–865, 1994.

[89] S. Kullback and R. A. Leibler, "On Information and Sufficiency," *The Annals of Mathematical Statistics*, vol. 22, no. 1, pp. 79 – 86, 1951. [Online]. Available: https://doi.org/10.1214/aoms/1177729694

[90] P. A. Chou, M. Koroteev, and M. Krivokuća, "A volumetric approach to point cloud compression—part i: Attribute compression," *IEEE Transactions on Image Processing*, vol. 29, pp. 2203–2216, 2020.

[91] L. S. Lopes, P. A. Chou, and R. L. de Queiroz, "Adaptive context modeling for arithmetic coding using perceptrons," *IEEE Signal Processing Letters*, vol. 29, pp. 2382–2386, 2022.

[92] G. M. Morton, "A computer oriented geodetic data base; and a new technique in file sequencing," IBM, Ottawa, Canada, Technical Report, 1966.

[93] J. Schmidhuber and S. Heil, "Sequential neural text compression," *IEEE Transactions on Neural Networks*, vol. 7, no. 1, pp. 142–146, 1996.

[94] G. Sandri, V. F. Figueiredo, P. A. Chou, and R. de Queiroz, "Point cloud compression incorporating region of interest coding," in *2019 IEEE International Conference on Image Processing (ICIP)*, 2019, pp. 4370–4374.

[95] V. F. Figueiredo, G. L. Sandri, R. L. de Queiroz, and P. A. Chou, "Saliency maps for point clouds," in *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*, 2020, pp. 1–5.

[96] C. Christopoulos, A. Skodras, and T. Ebrahimi, "The jpeg2000 still image coding system: an overview," *IEEE Transactions on Consumer Electronics*, vol. 46, no. 4, pp. 1103–1127, 2000.

[97] L. Itti, "Automatic foveation for video compression using a neurobiological model of visual attention," *IEEE Transactions on Image Processing*, vol. 13, no. 10, pp. 1304–1318, 2004.

[98] Y. Liu, Z. G. Li, and Y. C. Soh, "Region-of-interest based resource allocation for conversational video communication of h.264/avc," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 1, pp. 134–139, 2008.

[99] D. Yee, S. Soltaninejad, D. Hazarika, G. Mbuyi, R. Barnwal, and A. Basu, "Medical image compression based on region of interest using better portable graphics (bpg)," in *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2017, pp. 216–221.

[100] P. A. Chou, M. Koroteev, and M. Krivokuća, "A volumetric approach to point cloud compression—part i: Attribute compression," *IEEE Transactions on Image Processing*, vol. 29, pp. 2203–2216, 2020.

[101] E. Kang, H. Choi, and S. Ko, "Progressive region of interest coding using an improved embedded zerotree wavelet coding," in *Proceedings of IEEE. IEEE Region 10 Conference. TENCON 99. 'Multimedia Technology for Asia-Pacific Information Infrastructure' (Cat. No.99CH37030)*, vol. 1, 1999, pp. 609–612 vol.1.

[102] M. Subedar, L. Karam, and G. Abousleman, "An embedded scaling-based arbitrary shape region-of-interest coding method for jpeg2000," in *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, 2004, pp. iii–681.

[103] F. Song, G. Li, X. Yang, W. Gao, and S. Liu, "Block-adaptive point cloud attribute coding with region-aware optimized transform," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 33, no. 8, pp. 4294–4308, 2023.

[104] R. Gray and D. Neuhoff, "Quantization," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2325–2383, October 1998.

[105] E. Alexiou, I. Viola, T. M. Borges, T. A. Fonseca, R. L. de Queiroz, and T. Ebrahimi, "A comprehensive study of the rate-distortion performance in mpeg point cloud compression," *APSIPA Transactions on Signal and Information Processing*, vol. 8, p. e27, 2019.

[106] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[107] J. Ballé, P. A. Chou, D. Minnen, S. Singh, N. Johnston, E. Agustsson, S. J. Hwang, and G. Toderici, "Nonlinear transform coding," *IEEE Journal of Selected Topics in Signal Processing*, vol. 15, no. 2, pp. 339–353, 2021.

[108] J. Ballé. Entropy bottleneck layer. [Online]. Available: https://github.com/tensorflow/compression/blob/v1.3/docs/entropy_bottleneck.md

[109] J. Bégaint, F. Racapé, S. Feltman, and A. Pushparaja, "Compressai: a pytorch library and evaluation platform for end-to-end compression research," *arXiv preprint arXiv:2011.03029*, 2020.

[110] S. LoPresto, K. Ramchandran, and M. Orchard, "Image coding based on mixture modeling of wavelet coefficients and a fast estimation-quantization framework," in *Proceedings DCC '97. Data Compression Conference*, 1997, pp. 221–230.

[111] A. Mercat, M. Viitanen, and J. Vanne, "Uvg dataset: 50/120fps 4k sequences for video codec analysis and development," in *Proceedings of the 11th ACM Multimedia Systems Conference*, ser. MMSys '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 297–302. [Online]. Available: https://doi.org/10.1145/3339825.3394937

[112] J. B. O'Neal, "Predictive quantizing systems (differential pulse code modulation) for the transmission of television signals," *The Bell System Technical Journal*, vol. 45, no. 5, pp. 689–721, 1966.

[113] G. Langdon and A. Zandi, "Characterizing prediction error distributions for lossless image compression," in *Conference Record of The Thirtieth Asilomar Conference on Signals, Systems and Computers*, vol. 1, 1996, pp. 573–576 vol.1.

[114] 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, Aug 2017.