TÉCNICO LISBOA

Learning-Based Point Cloud Geometry Coding: Integrating Sampling Tools for Increased RD Performance

Manuel Maria de Ornelas Marques Ruivo

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors

Prof. Fernando Manuel Bernardo Pereira

Dr. André Filipe Rodrigues Guarda

Examination Committee

Chairperson: Prof. José Eduardo Charters Ribeiro da Cunha Sanguino

Supervisor: Prof. Fernando Manuel Bernardo Pereira

Member of the Committee: Prof. Nuno Miguel Morais Rodrigues

November 2022

Declaration of Originality

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

First and foremost, I have to thank to the entire DeepPCR team for a year full of intensive brainstormings and productive meetings. I'm especially grateful to my supervisors, Prof. Fernando Pereira and Dr. André Guarda, for providing this great challenge, and for being always available to share their knowledge and advice. It was a true pleasure working with you both. A special acknowledgment to Instituto de Telecomunicações for the conditions provided to develop this work.

This M.Sc. Thesis marks the conclusion of an important stage of my education. This would not be possible without:

My family and friends, who provided relentless support throughout these crucial years of my life.

My parents Rita and Manuel José, my sister Marta, and my girlfriend Marta, who were always so caring and supportive.

My godfather Ricardo, who gave me such valuable advice.

My life-long partners in crime, Maria Bau, Mariana Almeida, and Guilherme Semião, to whom I will always be indebted for growing by my side and believing in me.

My sister- and brothers-in-arms, Adelaide Ambrósio, David Souto, Gonçalo Tomás, and Miguel Rodrigues, who fought by my side such tough battles, and prevailed, making this long ride that much more enjoyable and fun.

What a journey this has been, more highs and lows than we ever expected… but together we did it!

To all of you, my most profound OBRIGADO.

Lastly, I would like to dedicate this work to the two people who taught me the most, my beloved grandparents Ricardo Marques. To my dear grandmother, Maria Josefina, for being the best role model I could have asked for, and for showing me what a pure heart is; and to my esteemed grandfather, Carlos Norberto, for his unbelievable storytelling skills that make anyone time-travel, and for passing on to me his legacy as a sailor and engineer. The lessons you both taught me will forever live in me.

Gosto de gostar de ti.

Resumo

As Nuvens de Pontos são um dos modelos de representação visual tridimensional mais versáteis da atualidade, uma vez que são capazes de dar ao utilizador os seis graus de liberdade necessários para criar experiências verdadeiramente imersivas. Contudo, dado que as nuvens de pontos precisam de um elevado número de pontos para conseguir criar a ilusão de superfícies contínuas e detalhadas, o uso destes modelos só é possível com a utilização de codificação eficiente, processo este que é responsável por reduzir a capacidade necessária para armazenar ou transmitir nuvens de pontos com um dado nível de qualidade.

Na última década foram propostos diversos codificadores de nuvens de pontos, que utilizam diferentes abordagens, entre eles as duas normas MPEG que permitem a codificação de nuvens de pontos estáticas e dinâmicas. Recentemente, dado o sucesso obtido em áreas como a visão computacional e a codificação de imagem, começaram-se a investigar também abordagens baseadas em aprendizagem profunda para a codificação de nuvens de pontos. O desempenho deste tipo de soluções foi de tal forma competitivo que dois dos mais relevantes grupos de normalização de codificação de informação multimédia, o MPEG e o JPEG, decidiram iniciar o desenvolvimento de normas de codificação de nuvens de pontos usando esta nova abordagem técnica.

O objetivo desta Tese de Mestrado é o desenvolvimento de uma solução de codificação de nuvens de pontos baseada em aprendizagem profunda que beneficie do uso de técnicas de amostragem para melhoria do seu desempenho de compressão. Para alcançar este objetivo, esta tese começa por rever alguns codificadores de geometria das nuvens de pontos e algumas das técnicas de amostragem mais relevantes. Posteriormente, é apresentada a solução de codificação com ferramentas de amostragem integradas, nomeadamente super-resolução. Além disso, a solução de codificação é complementada com a adição de um mecanismo de controlo de débito com vista a minimizar a complexidade da operação de otimização dos respectivos parâmetros de codificação. Finalmente, apresentam-se as conclusões relativas ao trabalho desenvolvido e perspectivas de trabalho futuro.

A solução de codificação desenvolvida nesta tese foi proposta e adoptada pelo JPEG como o Modelo de Verificação inicial para o desenvolvimento da norma JPEG Pleno: Codificação de Nuvens de Pontos baseada em Aprendizagem.

Palavras-Chave: nuvens de pontos, codificação, aprendizagem profunda, amostragem, super-resolução.

Abstract

Point Clouds represent one of the most versatile three-dimensional visual representation models available nowadays, as they can provide to the user the six degrees of freedom required for a truly immersive experience. However, since point clouds require large point sets of points to create the illusion of real surfaces, the usage of this representation model is only possible with the assistance of efficient point cloud coding solutions, a process that aims at reducing the storage/rate necessary to store/transmit a point cloud at a target quality.

In the last decade, several point cloud coding solutions have been proposed using distinct approaches. Among the most relevant are the two MPEG standards addressing static and dynamic point clouds. More recently, motivated by the success obtained in computer vision tasks and image coding, deep learning approaches started to be considered also for point cloud coding. The performance of these deep learning-based coding solutions has been so competitive that both MPEG and JPEG, the most important multimedia coding standardization groups, have decided to develop point cloud coding standards adopting this type of approach.

The goal of this M.Sc. Thesis is to propose a sampling-based approach to improve the rate-distortion performance of deep learning-based point cloud geometry coding. To accomplish this goal, this thesis starts by reviewing some point cloud coding solutions and sampling tools. Afterwards, a deep learning-based point cloud coding solution with integrated sampling tools is proposed. Additionally, this coding solution is extended with a rate control mechanism to minimize the optimization complexity associated to the selection of appropriate coding parameters. Finally, the conclusions for the developed work are presented as well as directions for future work.

The coding solution designed in this thesis has been proposed and adopted by JPEG as the initial Verification Model for the development of the JPEG Pleno Learning-based Point Clouds Coding standard.

Keywords: point cloud, coding, deep learning, sampling, super-resolution.

Table of Contents

List of Figures

List of Tables

List of Acronyms

2D	Two Dimensional
3D	Three Dimensional
ACC	Accuracy
ADL-PCC	Adaptive Deep Learning Point Cloud Coding
AE	Autoencoder
API	Auxiliar Patch Information
AR-GCN	Point Cloud Super-Resolution with Adversarial Residual Graph Networks
ARC	Automatic Rate Control
AVC	Advanced Video Coding
BCE	Binary Cross-Entropy
BD	Bjontegaard-Delta
bpp	bits per point
CCE	Categorical Cross Entropy
CD	Chamfer Distance
CNN	Convolutional Neural Network
CTC	Common Test Conditions
CTTC	Common Train and Test Conditions
dB	Decibel
DeepPCR	Deep Learning-based Point Cloud Representation
DL	Deep Learning
DLBER	DL-based Block Encoding and Reconstruction
DoF	Degree of Freedom
EMD	Earth Mover Distance
EUVIP	European Workshop on Visual Information Processing
FCT	*Fundação para a Ciência e Tecnologia*
FL	Focal Loss
FPS	Farthest Point Sampling
G-PCC	Geometry-based Point Cloud Coding
GT	Ground Truth
HEVC	High Efficiency Video Coding
HR	High Resolution
IRB	Inception Residual Block
IRN	Inception Residual Network

JPEG	Joint Photographic Experts Group
LR	Low Resolution
LoD	Level of Detail
LT	Lifting Transform
Meta-PU	An Arbitrary-Scale Upsampling Network for Point Cloud
ML	Machine Learning
M-PCGC	Multiscale Point Cloud Geometry Coding
MPEG	Moving Picture Experts Group
MPU	Multi-step Progressive Up-sampling Network
NN	Neural Network
OM	Occupancy Map
PC	Point Cloud
PCC	Point Cloud Coding
PCGC-ASR	Point Cloud Geometry Coding with Adaptive Super-Resolution
PCGP-DL	Point Cloud Geometry Prediction Across Spatial Scale using Deep Learning
PSNR	Peak Signal-to-Noise Ratio
PU-GAN	Point Cloud Up-sampling Adversarial Network
Q	Quantizer
RAHT	Region-Adaptive Hierarchical Transform
RD	Rate-Distortion
ReLU	Rectified Linear Unit
RGB	Red, Green and Blue
RGC	Residual Graph Convolution
SCD	Sparse Convolutional Downscaling
SCNN	Sparse Convolutional Neural Network
SCU	Sparse Convolutional Upscaling
SR	Super-Resolution
TRI	Target Rate Infringement
V-PCC	Video-based Point Cloud Coding
VAE	Variational Autoencoder
VM	Verification Model
ΔOR	Delta Optimal Rate
ΔTR	Delta Target Rate

1 Introduction

This chapter introduces the context and motivation behind the topic of this M.Sc. Thesis, entitled Learning-Based Point Cloud Coding: Integrating Sampling Tools for Increased RD Performance. Afterward, the main objectives for the work will be defined alongside the overall structure of this thesis and some achievements.

1.1 Context and Motivation

Multimedia applications have been evolving toward providing users with more immersive and realistic experiences. Considering the real world, it is natural to see an evolution from two-dimensional (2D) visual representations, such as images and video, to tri-dimensional (3D) representations, which can provide users with more interactive, immersive, and mesmerizing experiences.

The user's perception of visual immersion is deeply conditioned by the Degrees of Freedom (DoF) offered in a visual experience [1]. Among the six possible DoF, there are three translational and three rotational degrees. The larger the number of DoFs available to the user, the more immersive and faithful to reality the experience will be. A popular way to model all the light available for the users' eyes is the so-called *plenoptic function* [1] which is a 7D representation of light able to express the light at every single point in the 3D space, for every direction, for each wavelength and along time.

There are three main types of 3D representation models for the plenoptic function, capable of expressing the light information needed to offer 6-DoF experiences, namely light fields, meshes, and *Point Clouds* (PCs). These models can be grouped into two categories, notably image-based representation models, such as light fields, which use multiple 2D views/perspectives [1], and geometry-based representation models, such as meshes and PCs, which represent 3D objects/scenes directly in the 3D space. The key difference between PCs and meshes lies in the fact that meshes add to the PCs connectivity between the 3D points over the object's surface. In this thesis, the focus will be on PCs since they allow representing and processing 3D objects directly in the 3D space, facilitating user interaction and navigation, while offering lower computational complexity than meshes since connectivity data does not exist (although it may always be created).

In practice, a PC is defined as an unordered collection of 3D points located on the objects' surfaces. The PC points are represented by their 3D cartesian coordinates (x, y, z), usually referred to as the *PC geometry*. However, to offer more realistic and immersive experiences, PCs may have additional information associated with the points' geometry, e.g. regarding color/texture, normal vectors, and reflectance; these are the so-called *PC attributes*, where color assumes a central role.

Regarding time variance, PCs can be classified as static, if representing an object/scene with a single

shot invariant in time, which is the 3D equivalent of an image; or dynamic, if evolving in time, similarly to video. Furthermore, PCs can also be classified in terms of their *density,* ranging from dense to sparse. Dense PCs are characterized by a large number of points, close to each other, usually having a fairly regular distribution, thus creating smooth surfaces and objects/scenes. Sparse PCs are defined by having large distances between points, thus being more prone to non-uniformities, such as holes/missing parts or noisy outliers.

PCs are a very versatile 3D representation model, hence being used in a multitude of application domains, such as autonomous driving, immersive personal communications, augmented and virtual reality, educational and medical applications, and cultural heritage. An example of the latter is shown in Figure 1.

Figure 1: An example of a PC, named *Guanyin* [2].

As it is possible to observe in Figure 1, the illusion of real surfaces is provided by the high-density point set. Therefore, a high quality of experience requires a rather large set of points to represent a single static PC, sometimes in the order of millions of points, thus originating very large amounts of data to be stored and/or transmitted. Consequently, PC Coding (PCC) with significant compression levels is a must to reduce the large data to more manageable sizes, to bring PC-based applications to practical deployment.

With this purpose in mind, the Moving Pictures Expert Group (MPEG) has already developed two PC coding standards, the so-called Geometry-based Point Cloud Compression (G-PCC) and Video-based Point Cloud Compression (V-PCC), for static and dynamic PCs, respectively. These coding solutions will be reviewed later in this thesis.

As for the Joint Photographic Experts Group (JPEG), a Call for Evidence on PCC was launched in July 2020 [2]. Surprisingly, the only response to this JPEG Call for Evidence was the Adaptive Deep Learning (DL)-based PCC (ADL-PCC) proposed by Guarda *et. al.* in [3]. This PC coding solution was the result of the growing adoption of DL-based tools in the multimedia world, which had started with great performances for computer vision tasks, such as recognition and classification [4], and reached also competitive performance for image coding over conventional image coding standards [5]. The

promising results obtained with ADL-PCC, including over the MPEG G-PCC standard, led JPEG to launch a project especially targeting DL-based PC coding, which had a final Call for Proposals in January 2022 [6].

1.2 Objective and Structure

The key objectives of this M.Sc. Thesis are:

- Provide an overview of the main PC geometry coding solutions in the literature, namely the existing MPEG standards and the most relevant DL-based PC coding solutions;
- Introduce the main sampling concepts and review the most relevant DL-based PC sampling tools available in the literature;
- Propose an improved sampling-based approach for a DL-based PC geometry coding solution, such as ADL-PCC [3], aiming at improving its Rate-Distortion (RD) performance;
- Extend the proposed PC geometry coding solution with an automatic rate control mechanism:
- Assess the RD performance of the proposed PC coding solutions using representative test conditions to derive meaningful conclusions about the competitiveness of the proposed solutions.

To accomplish its objectives, this thesis is structured as follows:

- **Chapter 2** – Briefly reviews the state-of-the-art MPEG PCC standards, covering both the geometry-based and video-based PC coding solutions.
- **Chapter 3** – Reviews two of the most relevant DL-based PC geometry coding solutions in the literature.
- **Chapter 4** – Introduces the key PC sampling concepts and reviews the state-of-the-art on DL-based PC sampling tools.
- **Chapter 5** – Proposes a new DL-based PC geometry coding solution, based on ADL-PCC, integrating sampling tools for improved content-adaptive compression efficiency.
- **Chapter 6** – Extends the proposed DL-based PC geometry coding solution with a rate control mechanism, thus reducing the optimization process complexity.
- **Chapter 7** – Presents the conclusions for the developed work alongside some notes on future work to improve the proposed solutions.

The proposed PC geometry coding solution was part of the JPEG submission made by the research group where this thesis was developed; this solution was then selected as the best coding solution to become the geometry coding solution of the JPEG Pleno Point Cloud Coding Verification Model which is the starting point of the standard specification. Moreover, this PC geometry coding solution together with other contributions from the same research group also led to the submission of two additional papers. The three submissions are referenced as:

- A. F. R. Guarda, N. M. M. Rodrigues, M. Ruivo, L. Coelho, A. Seleem, F. Pereira, “IT/IST/IPLeiria Response to the Call for Proposals on JPEG Pleno Point Cloud Coding”, Doc. WG1m96005, JPEG Online Meeting, July 2022.
- M. Ruivo, A. F. R. Guarda, F. Pereira, “Double-Deep Learning-Based Point Cloud Coding with Adaptive Super-Resolution”, accepted in the 10th European Workshop on Visual Information

Processing (EUVIP), Lisbon, Portugal, September 2022.

- A. F. R. Guarda, M. Ruivo, L. Coelho, A. Seleem, N. M. M. Rodrigues, F. Pereira, “Deep Learning-Based Point Cloud Coding and Super-Resolution: a Joint Geometry and Color Approach”, submitted to IEEE Transactions on Multimedia, October 2022.

All the work was developed in the context of the project “Deep Learning-based Point Cloud Representation” with reference PTDC/EEI-COM/1125/2021, funded by Fundação para a Ciência e Tecnologia (FCT).

2 The MPEG Point Cloud Coding Standards in a Nutshell

This chapter will give a brief overview of the state-of-the-art MPEG standards for Point Cloud Coding (PCC), which are the only international coding standard solutions available so far for this type of visual data. Since there are several, relevant types of PCs, notably static, and dynamic PCs, MPEG has specified two standards, Geometry-Based Point Cloud Coding (G-PCC) and Video-Based Point Cloud Coding (V-PCC) to address the specific requirements associated with the use cases where each type of PC is relevant [7]. For each MPEG PCC standard, this section will look at its objective, technical approach, architecture and walkthrough, and performance.

2.1 Geometry-Based Point Cloud Coding

The following sub-sections briefly describe the G-PCC standard.

2.1.1 Objective and Technical Approach

The G-PCC standard offers an efficient coding solution for both PC geometry and attributes of static PCs, considering both lossy and lossless compression. Regarding geometry, the G-PCC standard offers a volumetric coding approach as it makes the most out of the PC 3D representation by adopting an octree to structure the voxelized PCs. The octree can be fully created up to the PC's original precision/resolution, thus offering lossless coding, or be pruned to a lower precision/resolution, offering an octree-based lossy coding solution; the lossy octree coding mode may also be combined with the so-called *Trisoup*, which defines a triangle-based surface approximation for the leaf octants. Regarding the attributes, the G-PCC standard offers two coding modes, notably the Region-Adaptive Hierarchical Transform (RAHT) exploiting the octree layered structure and the Lift Transform (LT) leveraging on the neighbors' correlation to save rate.

2.1.2 Architecture and Walkthrough

The complex task of PC coding may be divided into two parts, acknowledging that PCs hold information regarding geometry and attributes. Even though these two data components are coded separately in G-PCC, the two coding processes are not fully independent as the attributes to be coded must fit in the (lossy) decoded geometry that will be available at the decoder. This requires a step between the two coding processes, so-called *recoloring*, where the original attributes, e.g. color/texture, are mapped/transferred at the encoder to the decoded geometry, e.g. each decoded point gets the texture of the closest point in the original PC.

The G-PCC geometry and attributes encoder architectures are shown in Figure 2. Due to the dependence mentioned previously, the geometry coding process takes precedence; thus being

addressed first.

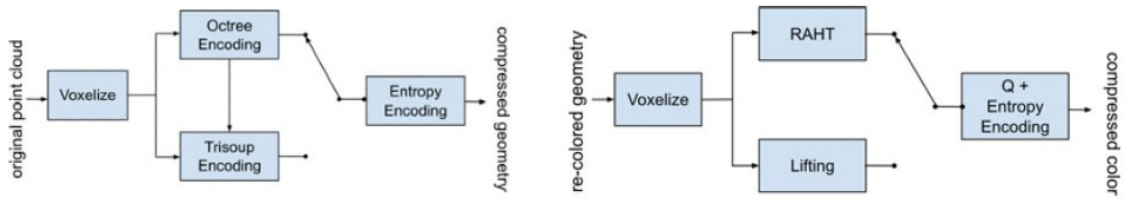


Figure 2: The G-PCC geometry (on the left) and attributes encoder (on the right) architecture [8].

The G-PCC geometry encoding process is composed of a few key steps, notably depending on if lossy or lossless compression is targeted:

1. **Voxelization** – Division of the 3D space into an evenly spaced grid of voxels and consecutive translation of the PC points to these voxels [9]. This is achieved by performing a many-to-one mapping technique where the input PC floating points are converted into integers with a fixed and pre-determined precision, i.e. bits per coordinate.

2. **Octree Encoding (Lossless or Lossy) –** Recursive process for splitting the PC into smaller blocks, structuring the points into a regular data structure – an octree – that will be filled with the occupancy information of every voxel. To do so, the PC bounding box is divided into eight sub-blocks, so-called *octants*, and their occupancy is checked – if filled, the block will be represented by a '1' in the octree and a new recursive octant-division step occurs for that sub-block; if empty, the block will be represented by a '0' and the recursive process ends for that block. When the octree reaches the PC's original precision, this process offers lossless coding; if the octree is pruned before, the coding process is lossy.

3. **Trisoup Encoding (Lossy)** – Given the potentially large number of points in an octree, the full PC representation is not always possible or required due to rate limitations. Instead, a pruned octree is used, where the octree leaf nodes no longer represent voxels but rather larger blocks. While the (lossy) coding process may stop here, it is alternatively possible to estimate a surface for the octree leaf nodes using a triangle-based reconstruction, trying to mitigate the quality loss, especially when severe pruning is applied, e.g. for lower rates [7], [10]. The object surface is recreated using a set of triangles defined by additional metadata that must be encoded along with the pruned octree.

4. **Entropy Encoding** – A coding technique that exploits the statistical correlation in the octree symbol representation to save rate in a lossless way [7].

If lossy geometry coding is applied, an intermediate step is necessary before encoding the attributes since the decoded geometry is not the original PC geometry for which the attributes are available. For that reason, a recoloring process is performed to make the attribute information match the decoded geometry which is also available at the encoder; for example, each decoded point gets the attributes of the closest point in the original PC. The result of this operation corresponds to the input of the attribute encoder, available on the right side of Figure 2. The attribute encoder is constituted by the following four modules:

1. **Voxelization** – Similar to the geometry voxelization process described above.

2. **RAHT (Lossy)** – Haar-inspired hierarchical transform [11] that exploits the octree structure to obtain an optimal attribute encoding. Using a bottom-to-top approach, the transform traversals the octree and computes a value for each node by transforming the upconverted sum of all descendants' attributes. The up-conversion is necessary as not all PCs are equally dense; thus, a weighted ponderation is necessary to make the algorithm density adaptive. This transform is used as a prediction to the transformed sum of attributes at the same level, hence generating a high-pass residual that needs to be quantized and entropically encoded.

3. **Lifting Transform (Lossless or Lossy)** – Euclidian distance-based prediction scheme that relies on different Levels of Detail (LoD) for efficient attribute encoding. The coding process starts by grouping the points into different refinement levels, allowing to recursively predict the attribute values of a LoD from the k-nearest-neighbors on the level below. When compared to the original, these predictions may have an offset, so-called *residue*. The residues produced at each recursive step and the lowest LoD must be entropically encoded, to be used by the decoder. If lossy coding is desired, the lifting transform offers a weighted relationship between the various LoDs, as well as an adaptive quantization strategy [7] before entropy coding.

4. **Quantization & Entropy Encoding** – By default, uniform quantization is performed [12] and the results are entropically encoded to generate the bitstream to be sent to the decoder.

2.1.3 Performance Assessment

Once seen how the G-PCC codec is designed, it is relevant to see how it performs. Several tests were conducted by Liu et al. [13], under the MPEG Common Test Conditions (CTC) [14], considering the point-to-point PSNR D1 quality metric for geometry and the Y-PSNR quality metric for the color, taken as the single attribute. The codec was tested using both lossless and lossy compression for geometry and color.

Starting with geometry, it has been shown that the octree coding outperforms the Trisoup coding mode for sparse PCs. On the contrary, the Trisoup mode proved to be extremely efficient for lower rates, while maintaining a considerable high quality. Regarding the attributes, the lifting transform is preferred for dense PCs, whereas the RAHT showed very efficient compression performance for sparse PCs.

2.2 Video-Based Point Cloud Coding

The following subsections briefly describe the V-PCC standard.

2.2.1 Objective and Technical Approach

V-PCC is the state-of-the-art coding solution for dynamic PCs, allowing both lossy and lossless compression for geometry and attributes.

V-PCC adopts a projection-based coding approach, which converts a 3D PC into 2D attributes, e.g.

color, and depth maps to exploit the full compression potential of very efficient state-of-the-art 2D video codecs to code both the geometry and attributes. Additionally, vital metadata for the 3D decoding process, like the Occupancy Maps (OM) and Auxiliar Patch Information (API), is created, coded, and multiplexed with the video coding-based data stream.

2.2.2 Architecture and Walkthrough

The overall V-PCC encoder architecture is shown in Figure 3.

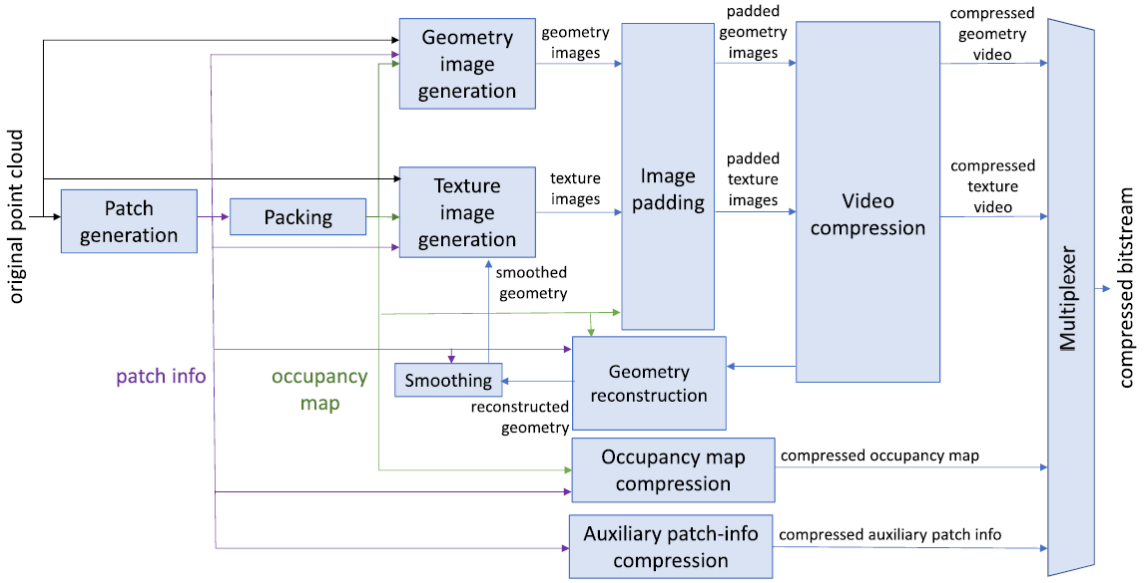


Figure 3: Overall V-PCC encoder architecture [12].

The V-PCC coding of the original PC proceeds as follows:

1. **Patch Generation** – A projection-based approach is adopted to segment the PC into 3D regions to be subsequentially projected into the 2D space, generating the so-called *2D patches* [15]. For this process, the normal at each point is computed, using a set of neighboring points. Each point is then projected into one of the six projection planes from the PC bounding box, notably the one maximizing the dot product between the plane's normal and the point's normal, thus allowing to obtain orthogonal projections reducing the self-occlusions and little distortion [15]. In this 2D projection process, two types of 2D maps are created: the attributes map, notably texture/color maps, which stores the attribute's values, e.g. Red, Green, and Blue (RGB) values for color maps, associated with each (x, y) map position and the depth map which stores the depth associated to that 3D position using as reference the projection plane, in practice the z coordinate. Since 3D objects may be complex, it is important to note that multiple points may be mapped into the same (pixel) position in the 2D maps; to address this issue, V-PCC allows the use of up to 16 layers to store overlapping points [15]; however, only two layers – the far and near layer - are typically used.

2. **Packing** – The projected 2D patches are positioned into a 2D grid, with some pre-defined resolution, using a packing procedure that defines the position of each patch inside the full 2D map. Since there are pixels that correspond to real 3D points and other pixels that do not, the OM is filled with '1' and '0' to define which 2D positions correspond to 'filled' 3D positions. The packing

procedure is an iterative process that is responsible for positioning the projected patches in the 2D grid while minimizing the unused space, to reduce the coding rate; since there are multiple PC frames in a dynamic PC, the packing process needs to lead to temporally coherent maps, i.e. having similar patches in similar locations throughout time, to allow the following video encoder to obtain better RD performance by exploiting the temporal correlation. During packing, rotations and translations may be applied to the patches to increase their fitting likelihood. If packing is not possible, e.g. due to the patch size, the 2D grid increases its resolution, by doubling its height [7] and restarting the iterative packing process. When every patch is packed in the grid, the 2D map is trimmed to remove unused space.

3. **Geometry Image Generation** – The geometry-packed patches correspond to the depth map, as they hold information regarding the distance from each 3D point to the projected surface [7]. Therefore, the geometry can be represented by a monochromatic image where the luminance value of a pixel represents the depth of a point.

4. **Texture Image Generation** – The texture-packed patches contain information regarding the color of the projected surfaces; thus, a colored image is obtained. While coding could happen in the original RGB format, a conversion to the YUV 4:2:0 color space is typically carried out as the chrominance sub-sampling allows to achieve greater compression efficiency.

5. **Image Padding** – The empty spaces between the packed patches in the 2D maps have sharp, high-frequency transitions which are not good for efficient coding. In this context, padding aims at filling the empty spaces, through a process known as geometry dilation [7], to create a smoother image for both the texture and depth maps, thus leading to more efficient video compression.

6. **Video Compression** – At this stage, the texture and depth maps are, in practice, two video sequences, which may be coded with any video codec, notably the most efficient video coding standards. While V-PCC does not impose any specific video codec to be used, H.264/AVC and H.265/HEVC are the most commonly used options [12], for both lossy and lossless video compression.

7. **Auxiliary Patch Information (API) Compression –** The API represents all the patch metadata necessary to reconstruct the PC from the geometry and attribute maps, e.g. the index of each patch's projection plane (as there are 6 in total) [12]. This data must be arithmetically encoded and multiplexed with the video streams to be used by the decoder.

8. **Occupancy Map (OM) Compression –** The OM is a binary map that signals the blocks with valid 3D projected points, i.e. the 2D map positions which correspond to 'filled' 3D positions. To save rate, the OM resolution may be changed depending on the correspondence between an OM and a 2D map pixel; in practice, one OM pixel corresponds to a $B \times B$ block of pixels in the 2D maps, where $B = 1$ and $B = 4$ offer lossless and lossy OM compression, respectively [12].

9. **Geometry Reconstruction and Smoothing** – Since the reconstructed geometry may differ from

the original geometry due to the lossy compression, the geometry needs to be reconstructed at the encoder to perform the recoloring process where the attribute information, e.g. color, is mapped into the reconstructed geometry when generating the 2D texture map [15]. Additionally, smoothing techniques are applied to remove discontinuities caused by compression artifacts [12].

10. **Multiplexer –** This module is responsible for merging the four sub-streams – geometry, attributes, OM, API – into a single bitstream to be stored or transmitted over a network.

2.2.3 Performance Assessment

As for G-PCC, several tests were conducted by Liu et al.[13] to assess the V-PCC RD performance under the MPEG CTC [14], considering again the PSNR D1 quality metric for geometry and the Y-PSNR quality metric for color. The V-PCC codec was tested for both geometry and color/texture, with lossless and lossy compression. The RD performance comparison between G-PCC and V-PCC is shown in Figure 4.

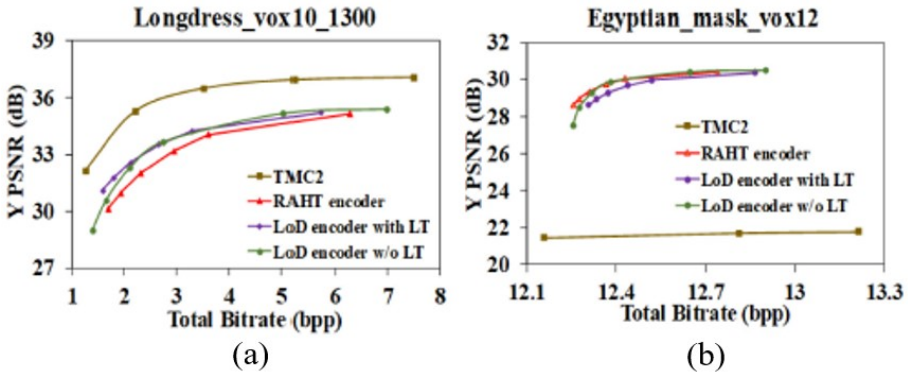


Figure 4: RD performance comparison between V-PCC (TMC2 label) and three different G-PCC configurations (RAHT and LoD labels), when using lossy compression for two PC: (a) *Longdress* and (b) *Egyptian Mask* [13].

For lossy compression, the best-performing coding solution will depend on the PC characteristics. For denser PCs, like the *Longdress_vox10_1300* PC, V-PCC outperforms all G-PCC coding configurations, especially for lower rates, as seen in Figure 4a; on the contrary, for sparser or noisier PC, like the *Egyptian_mask_vox12* PC, G-PCC becomes more competitive and may be the best fit, as shown in Figure 4b. For lossless compression, V-PCC is outperformed by G-PCC, as it allows obtaining a lower rate regardless of the PC type.

3 Deep Learning-based Point Cloud Geometry Coding: Main Solutions

The competitive performance of Deep Learning (DL) tools in computer vision tasks and image coding has made them a promising solution also for PCC. This chapter will review two of the most relevant DL-based PCC solutions in the literature, namely the Adaptive DL-based Point Cloud Coding (ADL-PCC) and Multiscale Point Cloud Geometry Coding (M-PCGC). For each of these codecs, this chapter will address its objective and technical approach, architecture and walkthrough, training process, and performance assessment.

3.1 Adaptive Deep Learning-based Point Cloud Coding

The following sub-sections briefly review the ADL-PCC coding solution proposed by Guarda et al. in 2020 [3] following other papers published before. This PCC solution was the single one proposed to JPEG following its Call for Evidence in 2020 and has led to the change of the JPEG Pleno PCC scope towards learning-based coding.

3.1.1 Objective and Technical Approach

The ADL-PCC solution offers a novel, lossy DL-based coding solution for static PC geometry. As for DL-based image codecs, ADL-PCC uses *Convolutional Neural Networks* (CNNs) to extract features from the input PC, creating a latent representation to be entropically encoded and sent to the decoder. Due to the unorganized structure of PCs, the full PC is divided into 3D binary blocks, where each voxel is filled with a ‘1’ or ‘0’ depending on being full or empty, respectively. Given the PCs' different characteristics, notably in terms of density, an adaptive coding approach is required to improve the compression efficiency, thus leading to the selection of one from multiple CNN models available to code each PC block. These CNN models differ in the parametrization of the loss function used during training, optimizing each model to a different block density.

3.1.2 Architecture and Walkthrough

The ADL-PCC encoder and decoder architectures are shown in Figure 5. It is possible to observe that ADL-PCC has a symmetric encoding and decoding process, i.e. every step performed to obtain the PC compressed bitstream is then reversed to reconstruct the full PC at the decoder. For this reason, the following ADL-PCC walkthrough will focus on the encoding process:

1. **PC Block Partitioning** – Similarly to the voxelization process used in MPEG G-PCC, ADL-PCC uses a binary, voxel-based data structure to represent the PC in 3D space [3]. The unorganized PC

points are mapped into a regular grid and binary 3D blocks are formed, where occupied voxels are represented by a '1' and empty voxels by a '0'. The creation of this regular block-based structure is vital to allow a DL-based coding approach [16] where each PC block is coded independently.

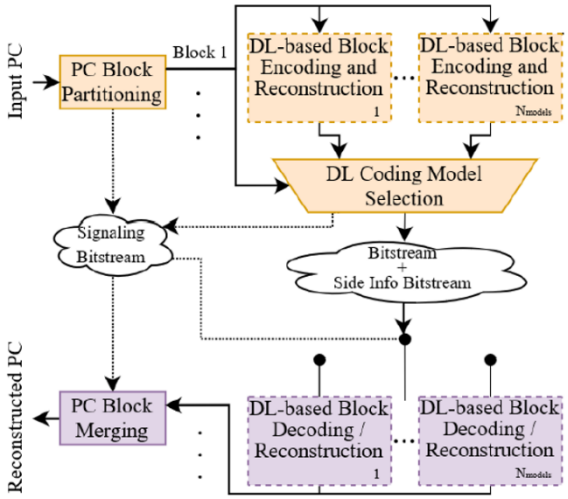


Figure 5: ADL-PCC encoder (in orange) and decoder (in purple) architecture [3].

2. **DL-based Block Encoding and Reconstruction (DLBER)** – This is the core module of the ADL-PCC coding solution and thus deserves a deeper description. The DLBER's architecture can be seen in Figure 6. The presence of multiple DLBER modules at the encoder offers adaptive capabilities to this coding solution as multiple DL models are checked in parallel to identify the most efficient for each PC block.

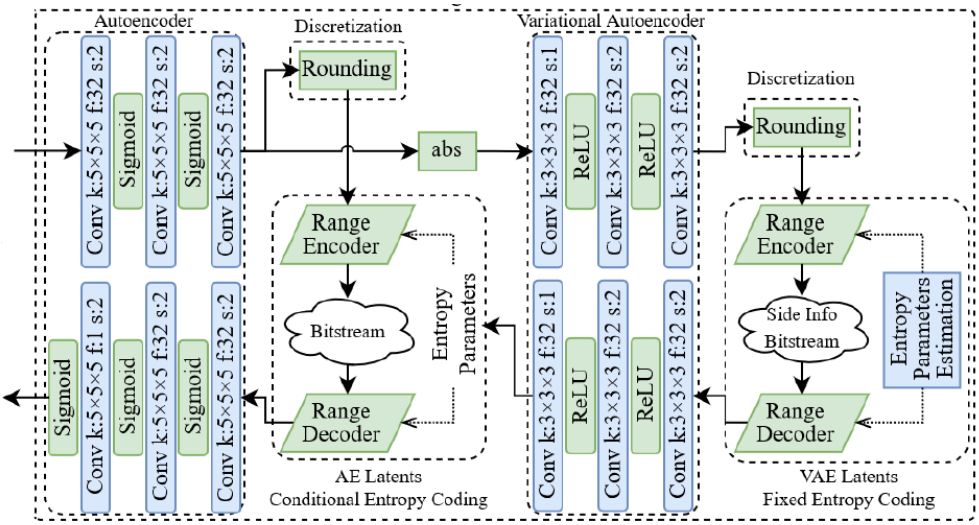


Figure 6: DLBER coding model architecture [3].

As can be seen in Figure 6, the DLBER module has a complex architecture with multiple elements that must be introduced:

- **Autoencoder (AE)** – Considers an input binary 3D block with a specific size, e.g. $64 \times 64 \times 64$, and applies a previously learned non-linear transform. This transform is performed by a 3-layer CNN, each layer applying 32 filters of size $5 \times 5 \times 5$, a sigmoid activation function, and a down-

sampling factor (or stride) of 2. The output of this encoder part of the AE is the so-called *latent representation*, in this case with an $8 \times 8 \times 8 \times 32$ dimension.

- **Discretization** – The latent representation obtained by the AE must be discretized to be entropically encoded. To do so, a simple rounding to integer is performed thus implying that no rate control is performed at this stage.
- **AE Latents Conditional Entropy Coding** – For entropy coding, each discretized value of the latent representation is modeled by a Gaussian function. The distribution characteristic parameters such as the standard deviation are dynamically estimated by a variational autoencoder to better adapt to the block latent statistics, thus resulting in a more efficient latent compression.
- **Variational Autoencoder (VAE)** – This module performs a hyper-transform to extract the parameters for the adaptive entropy coding. To do so, a 3-layer CNN is used, where each layer uses 32 filters of size $3\times 3 \times 3$ and a ReLU activation function [5]. The last two layers also apply a down-sampling factor of 2, thus producing a VAE latent representation of $2 \times 2 \times 2 \times 32$ dimension, which defines the latent statistical characterization used for entropy coding.
- **VAE Latents Fixed Entropy Coding** – Similarly to the AE, the VAE latents need to be discretized and entropically encoded. In this case, a fixed entropy coding is performed using an optimized model learned during the end-to-end training phase. The output entropy-coded latent representation is sent to the decoder through a side bitstream. The additional bitrate associated with the VAE latents statistical characterization is compensated by the higher compression efficiency of the AE conditional entropy coding. These trade-offs are optimized during the end-to-end training process.

3. **DL Coding Model Selection** – As multiple DL models are simultaneously used to code each PC block, some appropriate performance assessment must be carried out to find which model offers the optimal compression performance, thus adapting the codec behavior to each block's content. To do so, a distortion metric like the point-to-point distance (D1) is used to evaluate the reconstruction quality of every block with each model. The selected model for each block is the one leading to the lower RD cost, i.e. lowest distortion and rate. The result of this model selection process is signaled to the decoder using an adaptative arithmetically coded [17] signaling bitstream, occupying at most 0.5% of the total rate.

3.1.3 Training Process

Neural Networks (NN) rely on the training phase to learn how to perform a task. To do so, every neural network needs a representative training set – the input data that allow the NN to learn its target and thus what is relevant and what is not – and a loss function to minimize – the NN goal during training process.

The ADL-PCC solution used 13 PCs from the MPEG PC dataset as training data. These training PCs were partitioned into blocks and processed to fit the CNN requirements regarding input data, such as block size and precision, according to the MPEG PCC CTC [18].

Regarding the loss function, it can be generically represented as:

$$\text{Loss Function} = \text{Distortion} + \lambda \times \text{Coding Rate} \quad (1)$$

This expression shows the adoption of a RD trade-off, thus implying that the best quality is targeted while acknowledging that quality has always a rate cost. The Lagrangian multiplier, λ, allows to define the weight between the distortion/quality and the rate in the optimization performed during the training process; for example, a lower λ implies the rate cost is lower and thus a model targeting higher rates (and qualities) is trained; naturally, the opposite happens with higher λ. For ADL-PCC, the adopted block distortion metric is the so-called Focal Loss (FL), a variation of the Binary Cross Entropy; the FL is used to compute the binary classification error of each voxel reminding that each original voxel has associated a value of '0' or '1'. The FL is computed as:

$$FL(u,v)\begin{cases}-\alpha(1-v)^{\gamma}\cdot\log(v), & u=1\\ -(1-\alpha)v^{\gamma}\cdot\log(1-v), & u=0\end{cases} \quad (2)$$

where u is the original classification ('0' for empty and '1' for occupied voxels), v is the reconstructed voxel occupation probability (between 0 and 1), γ is a parameter expressing the relevance of voxels hard to classify (through experimental testing the value of $\gamma = 2$ was considered optimal by Guarda et al. [3]), and α is a parameter expressing the imbalance between 0's and 1's (there are typically many more 0s), which varies between 0 and 1. Since no optimal α value exists for all types of blocks, as they may have different densities, α is the hyperparameter that must be varied to obtain the multiple DL models that make ADL-PCC truly adaptive to the content being coded. Since changing α during testing time is not possible, it is required to train multiple CNNs for different α values to provide the desired adaptivity.

For ADL-PCC, 6000 input blocks were used to train each of the 25 CNNs – one for each possible RD point associated with a pair of λ and α values. Over 106 epochs were used for training with the Adam Algorithm [19], a learning rate of $10^{-4,}$ and mini-batches of 8 blocks. The learning rate is a tuning parameter in an optimization algorithm that defines the step size used to minimize the loss function; the batch size is the total number of training samples present in a single batch, where mini-batch is a subset of the entire dataset when this cannot be passed into the algorithm all at once.

3.1.4 Performance Assessment

To have a representative performance assessment of ADL-PCC, it is vital to have a new dataset for the testing phase unseen during the training process. If this requirement is not met, the testing results would be biased, and consequently, not representative of the performance that this coding solution would have for any other PC. For this reason, new PCs from the MPEG PCC and JPEG Pleno PC datasets were selected for testing. In [3], Guarda et al. conducted several tests using the selected test set, under the MPEG CTC [20] and considering the PSNR D1 quality metric. The anchor codecs for benchmarking are the G-PCC Trisoup and G-PCC Lossless standards.

Figure 7 shows a RD performance comparison between the various DL models associated to different α values and the G-PCC benchmarks, for two rather different test PCs: a very sparse PC, *Bumbameuboi*, and a very dense PC, *Queen*. The rate is measured in Bits Per Point (BPP) where a

point corresponds to a full voxel.

It is relevant noting that the various DL models are associated to different α values and thus they don't offer adaptability capabilities by themselves; this is the reason to label them as DL-PCC (and not ADL-PCC) where the A (from adaptive) is dropped. Moreover, the rate associated to the G-PCC Lossless codec is signaled (vertical line) to indicate that all DL-PCC points on the right side of the G-PCC Lossless rate can be ignored since they offer lossy quality at a rate superior to the G-PCC rate where original quality is obtained.

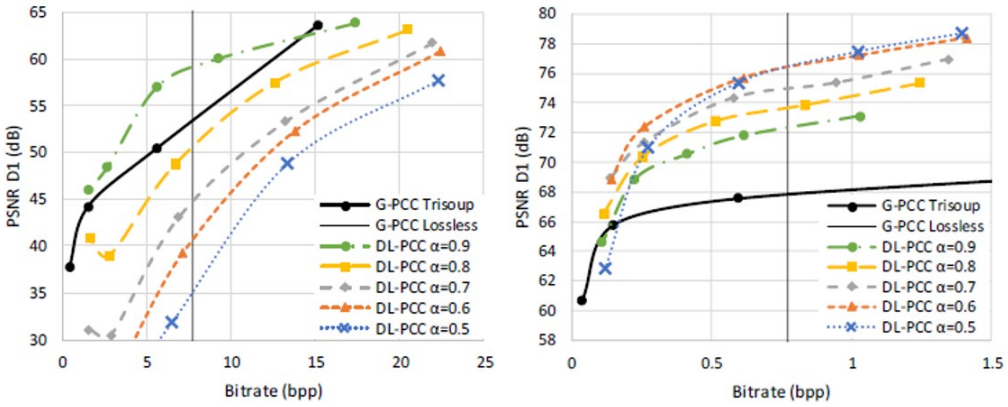


Figure 7: RD performance comparison between DL-PCC (with a single model) for different α values and G-PCC Trisoup and Lossless, for two test PC: (a) Bumbameuboi and (b) Queen [3].

The results in Figure 7 show that, as expected, the sparsity of the PC determines which α value is best and should be chosen by the encoder to achieve optimal RD performance. On one hand, Figure 7a shows that for the *Bumbameuboi* sparser PC, a higher α (in this case 0.9) leads to better performance, clearly outperforming G-PCC Trisoup. On the other hand, Figure 7b shows that a lower α value suits better denser PCs like *Queen*, even though all DL models available outperform G-PCC for most of the rates. Additionally, it can be noted that the best DL coding model (best α value) may also vary with rate, especially for the lower ones. Having these results in mind, it is possible to conclude that, as expected, the α value has a large impact on the RD performance and the adaptation capabilities associated with the DL model selection process are vital to be able to encode any type of PC efficiently.

In a second experiment, Guarda et al. [3] assessed the RD performance of the adaptive coding solution, i.e. ADL-PCC, when using N different models simultaneously, with N varying from 2 to 5, again in comparison with G-PCC Trisoup. Considering the same test PCs as previously, the RD performance results are shown in Figure 8.

As expected, the RD performance results obtained show that ADL-PCC outperforms the G-PCC Trisoup benchmarking solution for every rate achievable by ADL-PCC. Regarding the number of models, the simplest ADL-PCC configuration with adaptation capabilities, i.e. with only two models – one for denser PC/blocks ($\alpha = 0.5$) and another for sparser PC/blocks ($\alpha = 0.9$) – fails to outperform G-PCC Trisoup for denser PCs at lower rates; however, this does not happen for $N = 4$ where 2

additional models are available ($\alpha = 0.6$ and 0.8). As ADL-PCC with 4 and 5 models show similar quality, i.e. same PSNR D1 levels, for all rates, it is proved that ADL-PCC with four models, i.e. with $\alpha = \{0.5, 0.6, 0.8, 0.9\}$, is the best ADL-PCC coding solution, as it also has the lower training time and encoding complexity since one less model is trained and checked at coding time.

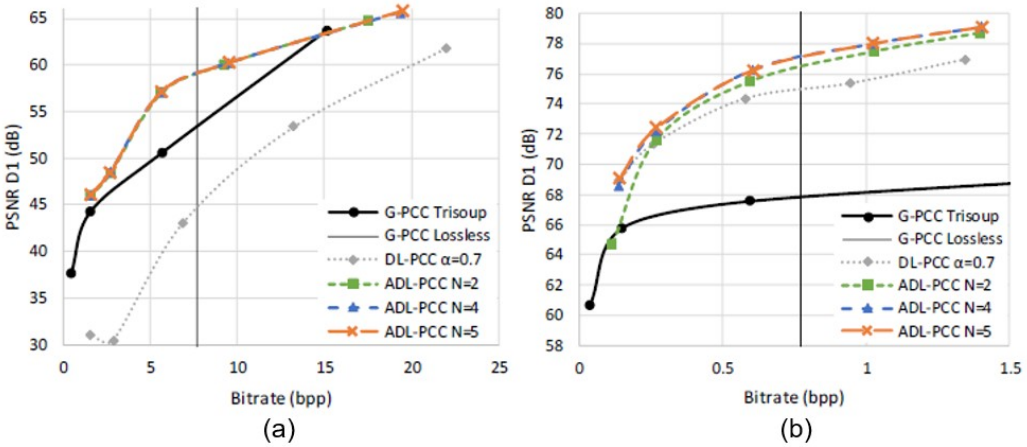


Figure 8: RD performance comparison between ADL-PCC (for N models) and G-PCC Trisoup and Lossless for two PCs: (a) *Bumbameuboi* and (b) *Que*en [3].

3.2 Multiscale Point Cloud Geometry Coding

This section intends to briefly review the M-PCGC coding solution proposed by Wang et al. during the 2021 Data Compression Conference [21]. The same authors had previously proposed another PC geometry coding solution, the so-called Learned-PCGC [3] that will be used for comparison in the Performance Assessment sub-section below.

3.2.1 Objective and Technical Approach

The M-PCGC solution offers a lossy, multiscale and efficient DL-based PC geometry coding solution. In alternative to the dense convolutions adopted by Guarda et al. in the ADL-PCC solution [3] and applied to the 3D binary blocks, this coding solution adopts sparse convolutions where the PC/block is represented using a sparse tensor, i.e. list of coordinates and feature pairs that correspond only to the filled voxels of the PC, thus reducing the complexity of all required computations, as they are not applied over empty voxels. Furthermore, Wang et al. [21] propose a new NN architecture for PC geometry coding using really deep and efficient networks, the so-called Inception-Residual Networks (IRNs) for feature extraction. Additionally, the multiscale resampling adopted in this solution allows to efficiently perform a progressive coarse-to-fine refinement of the reconstructed PC.

3.2.2 Architecture and Walkthrough

One of the main M-PCGC novelties is the representation of the input PC as a sparse tensor, represented by a set of coordinates, C, holding the geometric information of the PC's filled voxels, and a set of features, F, associated to each non-empty geometry coordinates, storing information related to the probability of each non-empty position being occupied. This means the input PC/block is represented as a list of (C, F) pairs where C are the coordinates of the filled voxels and F is all 1s

array, even though this might not be the case after being processed by the sparse convolutions.

The overall M-PCGC architecture is shown in Figure 9. The input (C, F) tensor is then pre-processed and encoded as follows:

1. **Pre-Processing** – This initial pre-processing module performs a scaling task, notably down-sampling the input PC. This operation is performed in a rather straightforward way, multiplying the point's coordinates by a sampling factor smaller than one, thus creating a denser PC that can be more efficiently compressed. A more detailed description of this process is given by Wang et al. in [22].

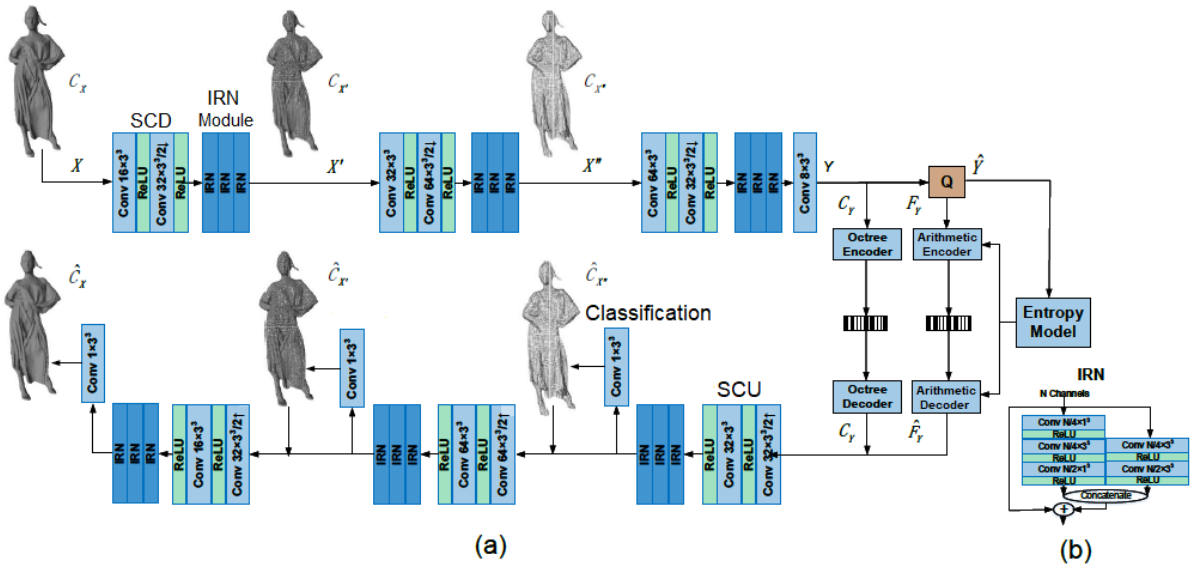


Figure 9: (a) M-PCGC overall architecture; (b) IRN layer architecture (based on [21]).

2. **Sparse Convolution Downscaling (SCD)** – After being pre-processed, the (C, F) tensor is transformed (downscaled) with a set of convolutional layers to obtain a smaller dimensionality representation. For this coding solution, a *Sparse CNN* (SCNN) is used with a stride size of 2, thus producing an output that is half the resolution/scale of the input. The traditional (dense) CNNs are just one type of SCNNs, as only N-dimensional hypercubical kernels are considered for CNNs, whereas SCNNs may use kernels with any N-dimensional shape, thus having more freedom to define the relationship between the input points [23]. This module is applied three times to produce three different scales of the input PC, hence creating the multiscale representation. A three-time repetition was chosen by Wang et al. in [21], as it lead to the optimal complexity/performance trade-off during experimental testing.

3. **Inception-Residual Network (IRN) Module** – After each downscaling step, a three-layer IRN module is used to estimate the PC's voxel occupancy probability distribution for the new scale, which is stored as F it in the downscaled tensor. When compared to the NNs used by other DL-based approaches, namely ADL-PCC, this module provides a deeper, hence more powerful, network with acceptable training times and low computational requirements, due to the residual approach, since a faster and lighter training process is involved [23]. Each IRN module is composed of three IRN layers, which architecture is shown in Figure 9b, and deeply examined in

[22] and [24].

4. **Octree Encoder** – After obtaining the smallest scale representation for the block/PC, the downscaled geometry (C_Y in Figure 9) is losslessly encoded using G-PCC Octree. Since this lossless compressed geometry corresponds to a small percentage of the overall rate, lossy geometry compression was not considered also because it would lead to a larger distortion with no impactful rate savings, according to Wang et al. [21].

5. **Quantizer (Q)** – In opposition to the geometric occupancy, C_Y, the features (F_Y) are lossy encoded and thus a quantization process is applied to improve the overall compression efficiency. This quantization is achieved by a simple rounding to integer, meaning that quantization is not used to perform rate control and obtain various RD points.

6. **Arithmetic Encoder and Entropy Model** – To exploit the statistical redundancy, the quantized features are arithmetically encoded using a statistical model proposed by Ballé et al. in [5].

After encoding, the compressed bitstream can be stored or sent to the decoder. Since the decoding process mirrors the encoding process, the decoder applies the Sparse Convolution Upscaling (SCU) and IRN modules to recreate the list of (C, F) pairs, for each of the three PC scales. Since F contains the probabilities of each voxel being occupied and not the occupancy itself, a key module is introduced in the decoder after each upscaling step:

7. **Classification & Reconstruction** – The F feature values expressing the probability that the non-empty voxels (at each scale) are effectively filled, are recovered. These feature values are sorted, and the top-k voxels (those with the highest probability of being filled) are considered occupied, where *k* is chosen accordingly to the PC density, while the other voxels are discarded/pruned. The results of this classification will be used to perform a new upscaling step. This process is repeated three times, for three different scales, thus allowing to have a gradual, progressive improvement of the PC geometry at a low complexity cost. Afterward, a post-processing (up-sampling) module must be used to revert the initial pre-processing (down-sampling) module, allowing the decoded PC to recover the original PC precision.

3.2.3 Training Process

For M-PCGC, as for ADL-PCC, there are two vital elements during training. The first is the training set, in this case, the ShapeNet dataset [25], and the second is the loss function to be minimized, in this case, the Lagrangian RD cost, presented in equation (1) (available in Section 3.1.3).

For M-PCGC, the adopted distortion metric is the so-called Binary Cross Entropy (BCE), defined as follows:

$$L_{BCE} = \frac{1}{N} \sum_{i} -(x_i \log(p_i) + (1 - x_i) \log(1 - p_i)) \tag{3}$$

where x_i is the original voxel i classification/occupation (0 or 1) and p_i is the estimated probability of the voxel i being occupied, stored in F. Given the target multiscale representation, the overall distortion is obtained with the multiscale BCE loss, i.e. the average of BCE loss for each scale, so that

for all of them good quality is targeted.

According to the experiments conducted by Wang et al. [21], $\lambda \in [0.25,10]$ was used to cover a wide rate range. In total, 32 000 batches were used during training, considering mini-batches of 8 and using the Adam Optimizer [19] with learning rates varying between 8×10^{-4} and 2×10^{-5}. During the training phase, the network considered a block size of $128 \times 128 \times 128$. As for the testing, no information is directly provided, although the full PC is most likely used.

3.2.4 Performance Assessment

As usual, the performance assessment must be done with a set of PCs different from those used during training and thus unknown to the coding models. Therefore, 10 test PCs were selected from multiple PC datasets, e.g. 8i Voxelized Full Bodies [26], for the M-PCGC testing phase [21]. For a fair and meaningful performance assessment, the test conditions followed as much as possible the so-called MPEG CTC [20]. Once again, the PSNR D1 quality metric was used to assess the quality, and the MPEG standards, G-PCC (Octree and Trisoup modes) and V-PCC, were used as references.

Regarding the M-PCGC RD performance, some of the results obtained by Wang et al. [21] are shown in Figure 10 considering the aforementioned MPEG standard benchmarks, as well as the so-called Learned-PCGC solution [22] proposed by the same authors of M-PCGC, which still uses a block-based volumetric approach but instead of sparse convolutions it uses a traditional CNN-based (with dense convolutions) AE to compress a single PC scale, as well as, a VAE to encode the latents entropically.

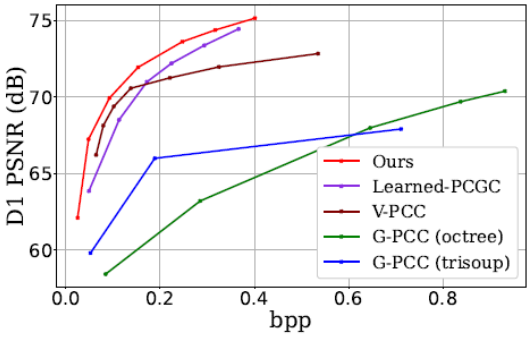


Figure 10: RD performance comparison between multiple PC geometry coding solutions for the Longdress PC [21].

The results in Figure 10 show that M-PCGC outperforms all the other benchmarking solutions, for any bitrate, thus indicating that M-PCGC is a very promising PC geometry coding solution. Unfortunately, since only dense PCs, like *Longdress*, were used for testing this conclusion only refers to this type of PCs.

Furthermore, the runtime comparison performed by Wang et al. [21] shows that for the same hardware, M-PCGC has the lowest average encoding time among the coding solutions reported in Figure 10 and the lowest average decoding time among the DL-based solutions considered, i.e. Learned-PCGC and M-PCGC. These results highlight the computational efficiency of the multiscale approach, the power of the IRN, and the importance of exploiting the PC's sparsity nature.

Regarding the initial spatial down-sampling applied during the pre-processing stage to obtain denser PCs for coding, multiple sampling factors may be used and assessed since there is no obvious best value. Figure 11 [22] shows the RD performance, for the PSNR D1 metric, for two very distinct PCs when various scaling factors are used.

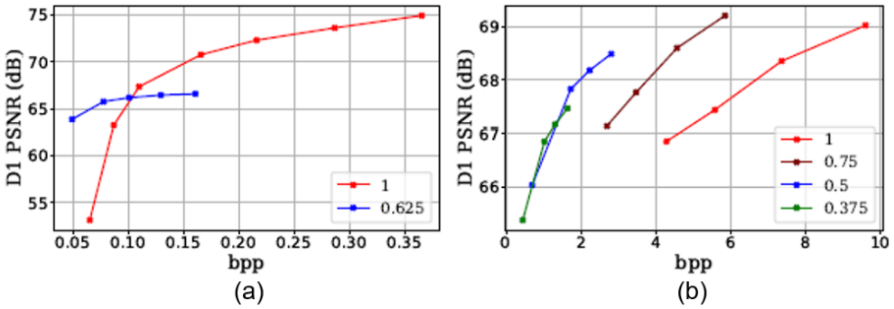


Figure 11: Learned-PCGC RD performance comparison for various spatial scaling factors (s), for two PCs: (a) denser *Longdress* and (b) sparser *Shiva*. [22]

The results in Figure 11 show the impact of the spatial resolution in the final RD performance and allow concluding that:

- For denser PCs, the down-sampling is extremely beneficial for lower rates since the information lost with this operation, causing higher levels of distortion, is well compensated by the amount of rate saved due to the smaller amount of information to code, thus representing an advantageous trade-off. However, this is no longer true for higher rates, as the reconstructed PC quality is limited by the pre-processing down-sampling [22].
- For sparser PCs, down-sampling the PC proves to be an important step for any rate, as down-sampled PCs, and consequently denser, are more efficiently compressed, i.e. with smaller BPP.

In summary, it can be observed that the optimal sampling factor varies with the rate and the PC characteristics, notably density. Furthermore, the design of a mechanism to automatically define the most appropriate sampling factor, for each PC at a given rate, would be extremely beneficial from the compression point of view.

4 PC Sampling in a Coding Context: Key Concepts and Relevant Tools

It is always difficult for a coding solution to excel under every scenario but ideally, it should deliver high performance for the widest set of conditions possible. In this work, it is relevant to identify and overcome the main struggles faced by DL-based PC geometry coding solutions, notably those relying on a volumetric PC representation, in particular the main weaknesses of the ADL-PCC solution proposed by Guarda et al. in [3], since this will be the starting point for the solution proposed in Chapter 5.

Unlike traditional 2D image and video, PCs are irregular in structure and density. In consequence, some PCs, namely the sparser ones, are harder to code efficiently, particularly when considering a coding solution like ADL-PCC that relies on a volumetric PC representation. This issue is especially critical for the lower bitrates where quality may significantly drop or the minimum attained rate is not as low as desirable [3]. Consequently, the idea of densifying the PC prior to its encoding is a promising one, similar to what is performed by the M-PCGC coding solution reviewed in Section 3.2. However, if some PC densification is obtained through some form of down-sampling performed at the encoder, an up-sampling counterpart performed at the decoder becomes a must to recover the original PC precision and density. In this context, this chapter will define some key concepts and processing pipelines and will review some relevant PC sampling-related tools in the quest to improve the ADL-PCC RD performance, especially for lower rates and sparser PCs.

4.1 Key Concepts and Alternative Processing Pipelines

First, it is important to define the terminology to be used from now on regarding PC sampling. To start, *PC sampling* is here defined as any type of operation/tool which samples or resamples the PC, notably regarding its precision/resolution, i.e. grid size, and/or point distribution (set of points). There are at least five relevant PC sampling tools which are covered in this work, notably:

1. **Down-sampling:**

a. ***Grid Down-sampling*** – Reduces the PC resolution by increasing the voxel size, thus decreasing the total number of voxels in the 3D grid. By adapting the original points to the new grid of lower resolution, a denser PC is produced. This operation may also be referred to as *precision down-sampling* since the precision defines the size and number of voxels along each axis in the grid.

b. ***Set Down-sampling*** – Reduces the number of points in the PC set, thus generating a sparser PC, according to some relevant criterion/criteria. As a result, a PC comprised by a subset of the points

in the original PC is created, while maintaining the same resolution.

2. **Up-sampling:**

a. ***Grid Up-sampling*** – Increases the PC resolution by reducing the voxel size, thus increasing the total number of voxels in the 3D grid; the obtained PC may be more or less sparse depending on the process used to fill the new, smaller voxels. For the same reason as before, this operation may also be called *precision up-sampling*.

b. ***Set Up-sampling*** – Increases the number of points in the PC set, thus creating a denser PC which point set contains the full original set and more. This operation can also be referred to as *Super-Resolution (SR)*.

3. **Resampling** – Redefinition of the PC set of points, for the relevant grid/precision, according to some relevant criterion/criteria. The output is a PC that, despite having the same resolution, is constituted by a set of points that may have, or not, the same cardinality and/or the same positioning as the original depending on the resampling purposes.

To help visualize these concepts, Figure 12 includes some examples that show the difference between the PC sampling operations above defined.

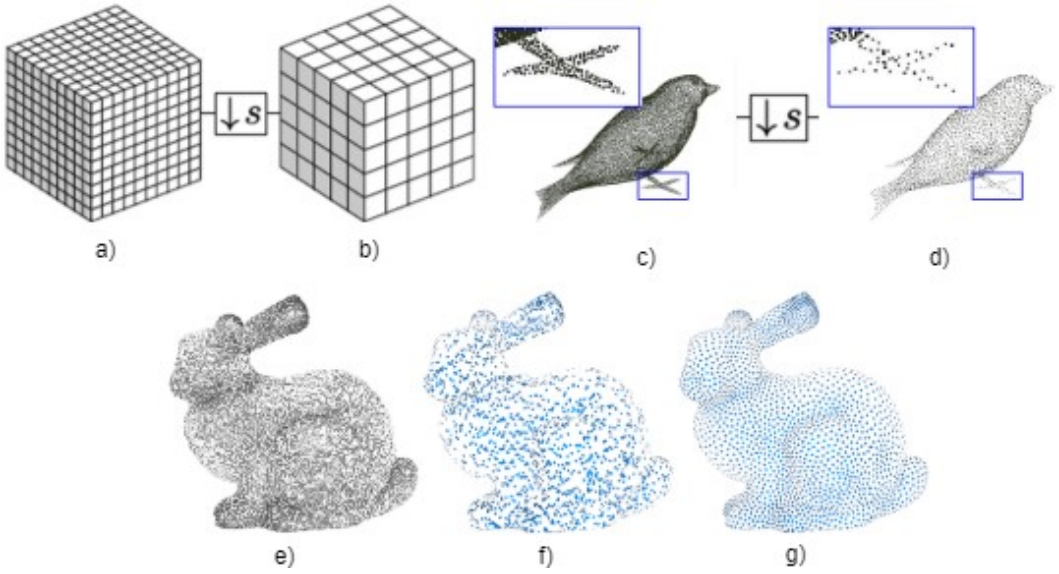


Figure 12: Examples of grid down-sampling (a, b), set down-sampling (c, d), and resampling (e-g) based on [27], [28] and [29].

Considering the examples in Figure 12 a) and b), it is possible to observe a grid/precision down-sampling by a factor 2, i.e. the output grid in b) is half the resolution of the input grid in a). For instance, if the initial grid has 10-bit precision per coordinate, i.e. 2^{10} voxels along each axis, then the output grid would have a precision of 9-bit precision per coordinate, meaning 2^9 voxels along each axis. Figure 12 c) and d) show a PC set down-sampling by a factor 4; therefore, if the input PC has 8192 points, the output PC will have 2048 points. Finally, Figure 12 e) to g) show a two-step example of PC resampling, the first reducing the number of points and the second redistributing them to create

a more uniform PC, while keeping the original precision/resolution.

At this stage, it is critical to remind that PC sampling is here considered in a PC geometry coding context, and thus it is important to discuss how the tools above may be used to boost the ADL-PCC RD performance, overcoming its weaknesses. By considering these additional PC sampling tools, the extended ADL-PCC coding pipeline would become as presented in Figure 13, where all types of PC sampling tools are considered.

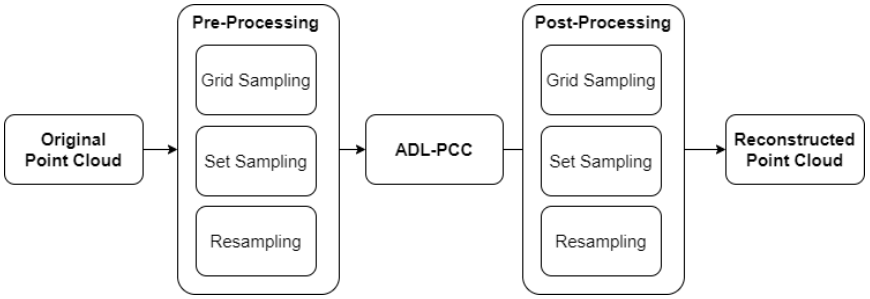


Figure 13: The extended ADL-PCC coding pipeline.

The main goal of the PC pre-processing module is to offer the core ADL-PCC encoder a PC with characteristics, notably density, that allow achieving better RD performance and eventually reaching lower rates than before. Hence, at the encoder side, the focus is clearly on reducing the grid resolution to densify the PC. While this implies some information loss, it is important to highlight that this happens in a lossy coding context where some quality loss regarding the original PC will inevitably happen. In this context, the following pre-processing tool configurations, at the encoder side, may be relevant:

- Basic grid down-sampling
- Basic grid down-sampling + Resampling
- Grid down-sampling
- Grid down-sampling + Resampling

Eventhough there are, in theory, more tool configurations, they do not help to achieve the desired goal, hence not being listed above. In this context, ‘basic grid down-sampling’ corresponds to a simple merging of multiple voxels and setting the merged voxel to ‘filled’ if at least one of the merging voxels is ‘filled’. In the list above, the solutions not labeled as ‘basic’ should perform in a more sophisticated way, naturally coming with a complexity cost.

Looking at the decoder side, the post-processing module mainly targets at recovering the quality lost with the initial down-sampling and eventually the lossy encoder process itself. First, to recover the initial PC resolution/precision, some grid up-sampling, more or less sophisticated, needs to be included. Finally, to maximize the quality of the reconstructed PC, some additional set up-sampling and/or resampling may be considered, notably if the previous grid up-sampling is not sophisticated enough and some additional PC densification is required. These processes largely depend on the target geometry quality metric which needs to be considered when selecting the post-processing tools configuration, e.g. PSNR D1 is very sensitive to a large reduction of the number of points regarding

the original PC. In this context, the following post-processing tool configurations may be relevant at the decoder side:

- Basic grid up-sampling
- Basic grid up-sampling + Resampling
- Basic grid up-sampling + Set up-sampling
- Basic grid up-sampling + Set up-sampling + Resampling
- Grid up-sampling
- Grid up-sampling + Resampling
- Grid up-sampling + Set up-sampling
- Grid up-sampling + Set up-sampling + Resampling

Once again, 'basic grid up-sampling' corresponds to a simple division of each voxel into multiple smaller voxels and setting to 'filled' only one of these voxels, thus leading to a rather sparse PC. In the list above, the solutions not labeled as 'basic' should have a more sophisticated approach to reach better quality, eventually at the cost of some additional complexity.

Following the analysis above, the next section will review some of the most interesting (non-basic) PC sampling-related solutions in the literature, always having a geometry coding context in mind.

4.2 Relevant Sampling Tools in the Literature

Point Cloud sampling has been an area of intense research in recent years, especially PC set up-sampling solutions has they have been proven to be a powerful tool for PC classification. This sub-section will review some relevant tools available in the literature that may be instrumental in a coding context, thus complementing the PC codec in a more sophisticated PC representation pipeline. Having this in mind, the Point Cloud Geometry Prediction Across Spatial Scale using Deep Learning (PCGP-DL) tool proposed by Akhtar et al. in [30] will be first reviewed, followed by the Meta-PU: An Arbitrary-Scale Up-sampling Network for Point Cloud proposed by Ye et al. in [31].

4.2.1 Point Cloud Geometry Prediction Across Spatial Scale using Deep Learning

This sub-section will review the PCGP-DL PC grid up-sampling tool proposed by Akhtar *et al.* in [30] and published at the 2020 IEEE International Conference on Visual Communications and Image Processing.

4.2.1.1 Objective and Technical Approach

The PCGP-DL tool offers a novel and promising DL-based approach for PC geometry grid up-sampling. This approach was designed as a post-processing tool after a PC decoder with the purpose of increasing the decoded PC's resolution on a coding pipeline which reduces the PC resolution for or during the coding process. In this context, this tool accepts as input a Low-Resolution (LR) and voxelized PC to output a High-Resolution (HR) reconstructed PC where the term resolution refers to the grid precision; in practice, there are larger voxels at the input and smaller voxels at the output.

Similar to the M-PCGC coding solution reviewed before, this grid up-sampling tool relies on sparse tensors and convolutions for efficient PC representation and processing, respectively. It starts by performing a simple octree-like up-sampling, e.g. dividing the PC voxels into smaller sub-voxels, and then predicts their occupancy by using a deep neural network with three IRN blocks that perform multi-resolution feature extraction from the input PC. Note that this is not a 'basic' grid up-sampling solution in the sense that the sub-voxels are filled in a more sophisticated way, targeting at maximizing the quality of the reconstructed PC regarding the original PC.

4.2.1.2 Architecture and Walkthrough

As seen in Section 3.2, adopting sparse tensors and convolutions for PC representation and processing allows designing more efficient tools that only process the filled voxels, discarding completely the empty ones, unlike in a traditional volumetric approach. The PCGP-DL tool's overall architecture is presented in Figure 14.

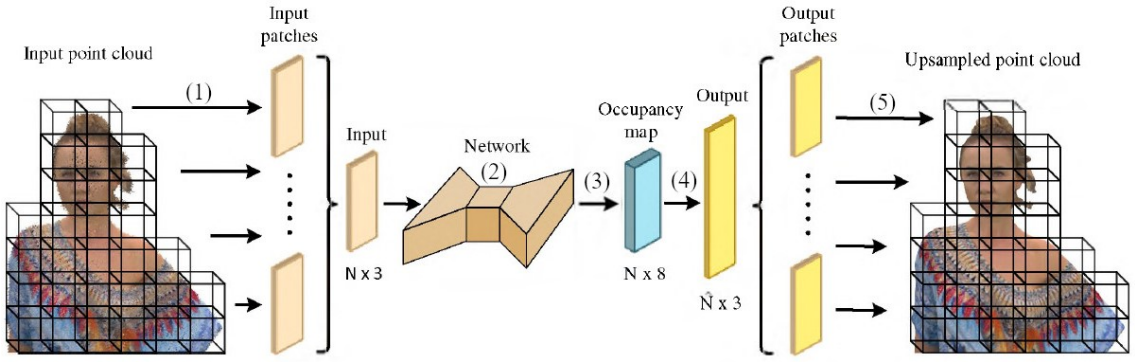


Figure 14: Overall PCGP-DL up-sampling pipeline for a sampling factor of 2 [30]. The numbers represent operations such as (1) PC Partitioning, (2) Occupancy Prediction, (3) Binarization, (4) Resolution Transformation, and (5) PC Merging.

Using the pipeline in Figure 14, the PC grid up-sampling is performed as follows:

1. **PC Partitioning** – The input PC is divided into 3D (cubic) blocks for easier handling. Each of these blocks is fed to the DL network to perform the grid up-sampling. It is important to note that each occupied voxel is represented by its geometric (x, y, z) coordinates, therefore, a block is represented by a set of $N \times 3$ elements, where N is the number of occupied voxels in the block. From now on, every operation is performed block-wise.

2. **Occupancy Prediction** – The sub-voxels occupancy is predicted by a U-shaped network; this name derives from its network architecture as shown in Figure 15. Please note the notation used, where the network's building Blocks will be represented with a capital B and will be followed by a number, whereas the PC partitioned blocks will not. The partitioned PC is fed into the network block by block to produce a set of occupancy probabilities for each sub-voxel. To do so, the network uses two fundamental steps:

- **Contracting Path** – The first half of the U-shaped network is responsible for multi-resolution feature extraction. During this process, multiple 'Block 1s' are used, each composed by a sparse

convolutional layer followed by three Inception Residual Blocks (IRB). The convolutional layer is responsible for down-sampling the input while the IRB units, like the IRNs used in Section 3.2, are responsible for feature extraction. Each convolution layer prior to the IRB units is followed by a ReLU activation function and a batch normalization step for improved training stability and speed [32].

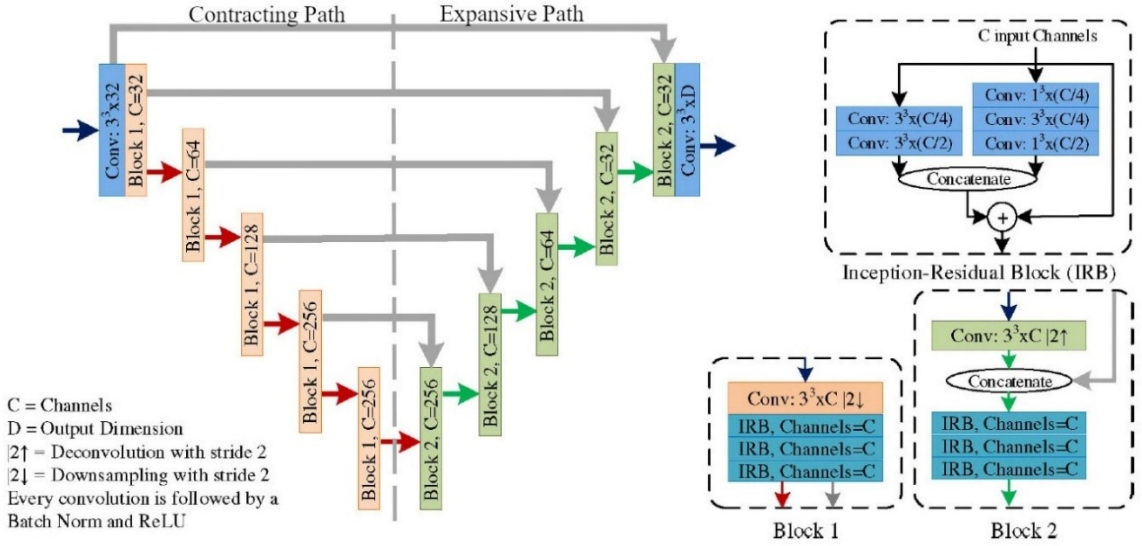


Figure 15: PCGP-DL network architecture and building Blocks [25]. On the left, the U-shaped network with a contracting path followed by an expansive path; on the right, the composition of the various block types.

- **Expansive Path** – This path is the second half of the U-shaped network and is responsible for feature expansion, thus creating a symmetric process to the contracting path. It relies on an iterative use of 'Block 2's, which are symmetric to 'Block 1' apart from the concatenation between the upsampled output and the HR features. At the final stage of the expansive path, a convolution is used to convert the output of the last 'Block 2' into a set of occupancy probability/prediction for every sub-voxel. The number of output sub-voxels is determined by the key tool to control input parameter, the so-called sampling factor (s), that determines the ratio between the size of the voxels in the LR and HR PCs. For instance, for an HR PC with half the voxel size in each direction of the LR PC, an sampling factor of 2 is used, leading to $s^3 = 8$ HR sub-voxels for each LR voxel.

3. **Binarization** – Based on the occupancy prediction for each sub-voxel, the final occupancy map for each HR block is created, i.e. a binary map with the occupancy value ('0' or '1') for each sub-voxel. To do so, each prediction probability is compared to a threshold value (typically 0.5), and all voxels with probability greater than this value are set to '1' (filled/occupied), while the remaining ones are set to '0' (empty). The occupancy map is represented by a set of $N \times s^3$ elements where N represents the initial number of filled voxels and s^3 is the number of sub-voxels per LR voxel.

4. **Resolution Transformation** – Once the occupancy map is defined, the occupied sub-voxels are used to create the output HR block, represented by a set of $\hat{N} \times s^3$ elements where $\hat{N}$ is the new

number of occupied voxels, therefore $N \leq \widehat{N} \leq N \times s^3$, and 3 corresponds to each of the (x, y, z) coordinates.

5. **PC Merging** - The final step is responsible for merging the HR blocks into a single HR PC output.

4.2.1.3 Training Process

As always for DL-based supervised processing solutions, it is critical to describe the training process, notably the training dataset and the loss function. Regarding the former, the *Longdress* and *Loot* PCs were selected from the 8i voxelized full bodies dataset [26]. Regarding the loss function, the BCE was adopted to evaluate the distortion between the output HR PC and the original HR PC, i.e., the so-called Ground Truth (GT). This loss function has already been defined in detail in Section 3.2.3.

For the training process, input blocks with size $128 \times 128 \times 128$ were adopted; different models are trained for different sampling factors, notably 4/3, 2, and 4. Note that during the octree up-sampling stage, 8 sub-voxels were used for the first two up-sampling factors, and 64 sub-voxels were used for the latter. The training process lasted for 4096 epochs, using an Adam optimizer [19] with a fixed learning rate of 10^{-3} and mini batches of 32 blocks.

4.2.1.4 Performance Assessment

For performance assessment, three new PCs were used, from the MPEG dataset, namely the *Redandblack*, *Soldier,* and *Queen* PCs. For quality assessment, the PSNR D1geometry quality metric was used in the context of several experiments following the MPEG CTC [14]. Regarding the input parameters, such as the block size and up-sampling factor, the same values adopted for training were used.

For benchmarking/comparison purposes, the authors defined a basic grid up-sampling technique (as defined in the previous sub-section) as reference to evaluate the performance of the PCGP-DL. The performance results are presented in Table 1.

Table 1: Average PSNR D1 geometry quality metric assessment [1].

Sampling Factor	'Basic' Grid Up-sampling (dB)	PCGP-DL Grid Up-sampling (dB)	Difference (dB)
4/3	64.6646	73.8630	9.1984
2	63.2080	72.0758	8.8678
4	58.0077	65.1890	7.1813

The results in Table 1 show a major quality improvement when considering PCGP-DL over the 'basic' up-sampling technique. As expected, it is also possible to observe that the prediction task becomes harder for larger up-sampling factors, hence leading to a smaller but still very impressive quality improvement. Naturally, these quality gains very much depend on the adopted objective quality metric.

Finally, it is necessary to recall that PCGP-DL is a post-processing tool. If used as the final step in a coding pipeline, the associated quality improvement comes at no cost, rate-wise. Having this in mind,

the potential of the PCGP-DL up-sampling tool in a coding context is clear as will be later demonstrated in this thesis.

4.2.2 Meta-PU: An Arbitrary-Scale Up-sampling Network for Point Cloud

This sub-section will review the Meta-PU set up-sampling tool proposed by Ye et al. in [31], inspired by the SR technique for 2D images proposed by Hu *et al.* [33], and published at the 2021 IEEE Transactions on Visualization and Computer Graphics.

4.2.2.1 Objective and Technical Approach

The Meta-PU tool offers a promising Meta-Learning PC set up-sampling solution. Meta-Learning usually refers to a learning to learn process, where a Machine Learning (ML) algorithm learns from the output/prediction of another ML algorithm. The Meta-PU tool sets apart from other solutions in the literature by allowing any random (non-integer) sampling factor to be used with a single trained model; this set up-sampling factor defines how many more points must have the output PC regarding the input PC. Meta-PU is a post-processing tool focused on producing an output PC denser than the input PC, without changing the PC resolution/precision. Meta-PU achieves this goal by using a CNN to extract a set of features from the input PC and adjusting the feature processing network even after training, thanks to the use of a meta-subnetwork, i.e. a network which output will help the main network to adjust accordingly to the desired set up-sampling factor/scale. Subsequently, the features are converted into a set of new points that are added to the input PC, producing an output denser than the target set upsampled PC. Finally, a set down-sampling technique, so-called Farthest Point Sampling (FPS) strategy, is used to achieve the target PC density, defined by the user's selected set up-sampling factor.

4.2.2.2 Architecture and Walkthrough

Unlike all the solutions and tools previously studied, Meta-PU does not apply a regular (3D) grid to the PC, therefore is not based on voxels. Instead, an alternative representation approach is used where each PC point is jointly considered with its k-nearest neighbors, hence referred as *the centroid and its neighbors*. For this reason, most of the Meta-PU operations are point-wise (and not block-wise) operations. The overall Meta-PU architecture is shown in the Figure 16.

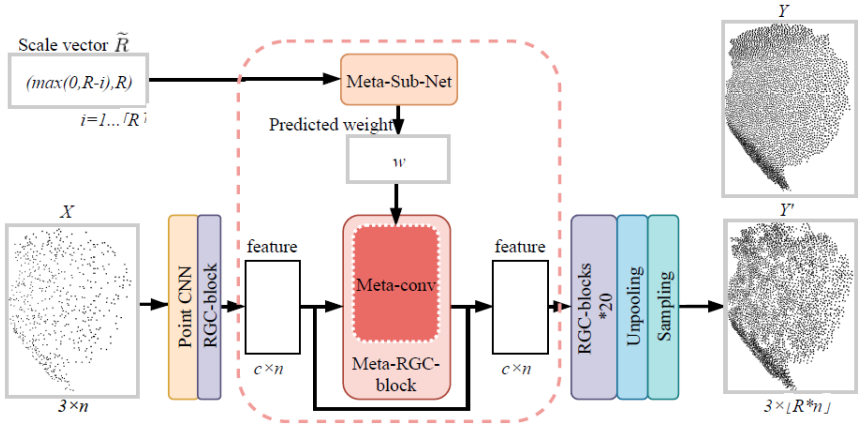


Figure 16: Overall Meta-PU architecture [31]. X is a sparse input PC, Y is the ground truth dense PC,

only available during training, and Y' is the Meta-PU output, the upsampled PC.

The Meta-PU tool has two inputs: X, a sparse PC with n points defined by their (x, y, z) coordinates, thus being represented by a set of $n \times 3$ elements; and the target up-sampling factor, R, later used to create the Scale Vector ($\tilde{R}$), see details below. Theoretically, R may assume any value; however, due to GPU limitations, an upper limit is imposed, R_{max} = 16, thus constraining R to assume any value (integer or not) between 1 and 16.

The PC set up-sampling process occurs as follows:

1. **Point CNN** – A three-layer CNN is used to extract spatial feature from the input PC points. This network is responsible for applying multiple point-wise convolutions to every centroid and associated neighbors to extract a $k \times c$ set of features for each of the n points in the PC. These features are then grouped by a max-pooling layer, thus creating a singular $n \times c$ output tensor of features.

2. **Residual Graph Convolutional (RGC) Block** – Similarly to the Point CNN network, the RGC blocks are responsible for feature extraction from each centroid and set of neighbors. The RCG block includes multiple graph convolutions layers and residual skip-connections, as shown in its architecture in Figure 17 .

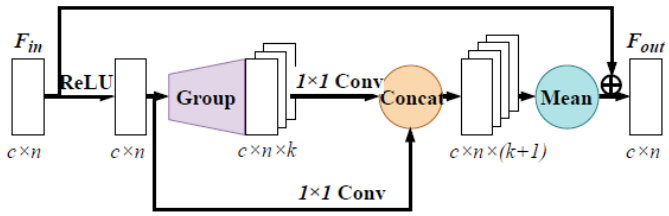


Figure 17: RGC block architecture [31].

The RGC block receives the $n \times c$ feature tensor from the Point CNN and is responsible for performing the following convolutions:

$$f_{out}^{p} = w_0 * f_{in}^{p} + w_1 * \sum_{q \in N(p)} f_{in}^{q}, \forall p \in X \quad (4)$$

where f_{in}^{p}, f_{out}^{p} are the RGC block input and output feature tensors, respectively, considering the point *p* as the centroid, and f_{in}^{q} is the feature tensor considering the neighbor *q* as centroid, therefore considering $N(p)$ as the set of neighbors of *p*. Additionally, the weights w_0, w_1 are learned during training and $*$ is the point-wise convolution. The residual skip-connection is introduced to address the vanishing gradient problem and improve convergence speed. In total, 22 of these RGC blocks are used in the Meta-PU network. The output of this block is a feature tensor with the same dimensions as the input.

3. **Meta-RGC Block** – Among all the RGC blocks, the Meta-RGC block is special as it is responsible for providing flexibility to deal with arbitrary-scale up-sampling factors using a single model. The architecture of this block is shown in Figure 18.

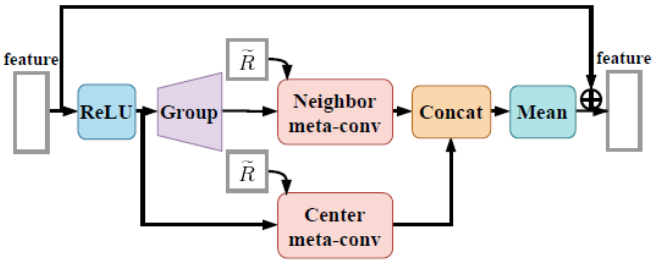


Figure 18: Meta-RGC block architecture [31].

The Meta-RGC applies a variant of the RGC-block's convolutions, the so-called meta-convolutions, described as follows:

$$f_{out}^{p} = \varphi(\tilde{R};\theta_0) * f_{in}^{p} + \varphi(\tilde{R};\theta_1) * \sum_{q \in N(p)} f_{in}^{q} \ , \forall\, p \in X \tag{5}$$

In contrast to the previous RGC block, the convolutional weights $\varphi(.)$ are now dynamically estimated using the meta-subnetwork described below. Through these dynamic weights, the number and position of the points in the output PC may vary with *R*, thus serving different *R* with a single model, providing a more uniform point distribution.

4. **Meta-subnetwork** – The major Meta-PU technical novelty is largely associated to this module/network which is responsible for computing the convolutional weights of the second RGC block, accordingly to the target up-sampling factor. The architecture of this network is shown in Figure 19.

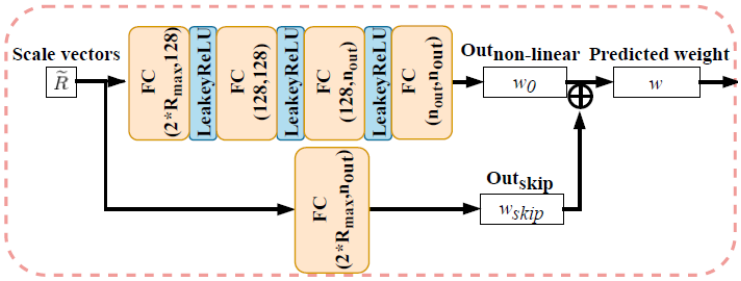


Figure 19: Meta-sub-network architecture [31], where the FC blocks correspond to fully connected layers and Leaky ReLU is a variant of the ReLU activation function [34].

As input, the Meta-sub-network takes the so-called *scale vector* $(\tilde{R})$, which is obtained by pre-processing the input up-sampling factor, R. The pre-processing of R, inspired in [33], consists simply in listing a set of $2R_{max}$ up-sampling factor pairs, following the format (R', R) where R' is $\max(0, R - i)$, $i = 1 \ldots [R]$. Keep in mind that R may assume any value (integer or not), and so does R'. Each entry in the scale vector will be used to make the network consider multiple sub-scales (R') to achieve the optimal result, i.e. optimal meta-convolutional weights for the target up-sampling factor R.

5. **Unpooling** – The unpooling layer is responsible for reshaping the features extracted and adding points to the input PC. This layer appears after the last RGC-block which is similar in architecture to the RGC block described above, although producing an output tensor of dimensions $(R_{max} \times$

$3) \times n$. The unpooling operation reshapes this tensor into a size $(R_{max} \times n) \times 3$. Furthermore, a skip connection between points is added at this stage with the consequence that the output tensor includes the input PC plus the newly creates set of points, thus yielding a denser PC with $\lfloor R_{max} \times n \rfloor$ points. The floor operator is necessary to ensure the number of points in the PC is, naturally, an integer number.

6. **Sampling** – After obtaining a PC that is possibly denser than desired, it is necessary to apply a set down-sampling technique to obtain the target number of points. The sampling block adopts the so-called Farthest Point Sampling (FPS) tool [35], responsible for selecting $N = \lfloor \mathrm{R} \times n \rfloor$ points from the denser PC with $\lfloor R_{max} \times n \rfloor$ points. The sampling criteria is to choose the set of points that maximizes the Euclidean distance between them contributing to a more uniform output point set [31]. In addition to this uniformity improvement, the benefit of using an up-and-down sampling strategy comes from making the network learn from a wider range of up-sampling factors, as proved in the Sub-section 4.2.2.4 .

4.2.2.3 Training Process

For training, 40 from the 60 meshes in the VisionAir Repository [36] were uniformly sampled using Poisson Disk Sampling to obtain the dense GT PCs (Y) with N points each, which is relevant only during the training process. Each of these PCs was further sampled using a non-uniform sampling technique to obtain the sparser input PCs (X) with n points each to be set upsampled. Additionally, each of these PCs was partitioned into 100 patches, for easier handling.

A novel loss function was proposed by Ye *et al.* [31] to encourage new points to lie on the object's surfaces and be uniformly distributed as follows:

$$\mathcal{L} = \lambda_{rec}\mathcal{L}_{rec} + \lambda_{uni}\mathcal{L}_{uni} + \lambda_{rep}\mathcal{L}_{rep} \quad (6)$$

where the parameters λ are the weights of each loss component in the joint loss function. The three components are defined as follows:

1. **Reconstruction Loss** $(\mathcal{L}_{rec})$ – $\mathcal{L}_{rec}$ is responsible for assuring that new points lie on the object surfaces; it is based on the unbiased Sinkhorn divergences proposed by Feydy *et. al.* in [37] and elaborated in [31]. To do so, the difference between the upsampled result $(\hat{Y})$ and the GT PC (Y) is measured.

2. **Uniform Loss** $(\mathcal{L}_{\mathrm{uni}})$ – $\mathcal{L}_{\mathrm{uni}}$ is a two-part loss function responsible for promoting uniform point distributions, both locally and globally [31]. To do so, a chi-squared model is adopted to evaluate both the point density and minimum distance deviation from their expected values; check [38] for a more precise definition.

3. **Repulsion Loss** $(\mathcal{L}_{rep})$ – $\mathcal{L}_{rep}$ has been proposed by Yu *et. al.* in [39] and serves to guarantee that the generated points are not too close to the input points in X, hence incentivizing uniformity.

The Meta-PU tool was trained using a variable scale training strategy, i.e. for each of the 60 training epochs, the factor R is randomly selected from a set of sampling factors (from 1.1 to 16, in increments

of 0.1). This strategy guarantees that Meta-PU achieves good performance during testing, regardless the target up-sampling factor. The Meta-PU model was trained using the Adam algorithm [19] as loss optimizer; and learning rates of 10^{-3} and 10^{-4} were used for the meta-subnetwork and the Point CNN and RCG blocks, respectively. Additionally, the loss function weights, $\lambda_{rec}, \lambda_{uni}, \lambda_{rep}$, were set as $1, 10^{-3}$ and $5 \cdot 10^{-3}$, respectively.

4.2.2.4 Performance Assessment

The same VisionAir dataset was used as test dataset, now using the 20 meshes not included in the training set. The meshes were sampled in the way as for training. Multiple tests were conducted by Ye *et. al.* in [31], and multiple performance metrics were adopted for an exhaustive assessment of the Meta-PU performance. Among the metrics used, the most relevant in the context of this thesis are the two point-to-point distance metrics, i.e. the *Chamfer Distance* (CD) and the *Earth Mover Distance* (EMD). These metrics are defined as follows:

$$d_{CD}(PC_1, PC_2) = \sum_{x \in PC_1} \min_{y \in PC_2} \|x - y\|_2^2 + \sum_{y \in PC_2} \min_{x \in PC_1} \|x - y\|_2^2 \quad (7)$$

$$d_{EMD}(PC_1, PC_2) = \min_{\phi: PC_1 \to PC_2} \sum_{x \in PC_1} \|x - \phi(x)\|_2 \quad (8)$$

where $PC_1 \; and \; PC_2$ are two distinct PCs and $\phi(x)$ is a bijective function that transforms one PC set into another, i.e. the GT PC into the output upsampled PC. For both metrics, the lower the distance d, the better is the output quality regarding the GT PC.

The following state-of-the-art set up-sampling tools were used for benchmarking purposes:

- **PC SR with Adversarial Residual Graph Networks (AR-GCN)** – A single-factor set up-sampling tool proposed by Wu et al. [40] which introduces residual connections into graph convolutions and skip connections between the input and output.
- **PC Up-sampling Adversarial Network (PU-GAN)** – A single-factor set up-sampling tool proposed by Li et al. [38] based on a generative adversarial network for feature extraction and expansion.
- **Multi-step Progressive Up-sampling Network (MPU)** – A multi-factor set up-sampling tool proposed by Yifan et al. [41] which obtains a denser PC by recursively applying the same up-sampling factor, hence only requiring one trained model.

A comparison between the single-scale and multi-scale up-sampling solutions was made by performing multiple tests using various up-sampling factors, namely $R = \{2,4,6,9,16\}$. The results are presented in Table 2. It is important to note that to obtain these results, five PU-GAN models and three AR-GCN models were trained whereas for MPU and Meta-PU only one model was required.

Table 2: Comparison between two single-factor up-sampling tools (AR-GCN, PU-GAN), a multi-factor up-sampling tool (MPU) and Meta-PU for various scale factors. For each column, the best result is highlighted in bold. On the right, the average inference time (for every factor) is presented.

R	**2x**		**4x**		**6x**		**9x**		**16x**		**Average Time (s)**
Tools	**CD**	**EMD**	**CD**	**EMD**	**CD**	**EMD**	**CD**	**EMD**	**CD**	**EMD**	
AR-GCN	-	-	0.009	0.018	-	-	**0.008**	0.022	0.009	0.023	10.28
PU-GAN	0.016	0.009	0.010	0.016	0.012	**0.013**	0.009	**0.009**	0.009	**0.022**	10.06
MPU	**0.010**	0.013	0.009	0.012	-	-	-	-	**0.008**	0.023	143.61
Meta-PU	**0.010**	**0.005**	**0.008**	**0.008**	**0.008**	0.014	**0.008**	0.016	**0.008**	0.023	**0.79**

The results show that Meta-PU outperforms the other up-sampling tools for most of the up-sampling factors, *R*. This proves that the adopted variable-scale training strategy is beneficial not only for additional flexibility but also for performance improvement due to the training involving several up-sampling factors at once. More tests were conducted in [31] to help prove this. Furthermore, the average Meta-PU inference time is drastically lower than the other state-of-the-art tools. Once again, a more detailed description of how these times were obtained is available in [31] .

The results obtained come to prove the power and flexibility of the Meta-PU up-sampling tool, thus being a promising addition to the ADL-PCC coding pipeline.

5 Proposing a Point Cloud Geometry Coding Solution with Adaptive Super-Resolution

In this chapter, a new PC geometry coding solution with content-adaptive super-resolution is proposed. This solution was submitted to the JPEG Pleno Call for Proposals on DL-based Point Cloud Coding issued in January 2022 [6] [42] and was finally selected as the best performing, thus corresponding to the geometry component of the first JPEG Pleno PCC Verification Model (VM).

5.1 Objective and Technical Approach

The proposed PC geometry coding solution offers an efficient content-adaptive coding solution by integrating a super-resolution module, notably allowing to code PCs with a large range of densities and reaching lower rates in a lossy compression context.

The PC geometry, represented as 3D binary blocks, is coded using a 3D CNN capable of producing a compact latent representation. The PC diversity, namely in density, poses a great challenge to DL-based coding solutions, namely for ADL-PCC [3], the starting point of the proposed solution. The introduction of grid and set sampling operations, notably down-/up-sampling and super-resolution (SR), are considered to mitigate the effects of such PC heterogeneity on the final compression performance. While the down-sampling operation ensures the high density of the block being coded at a lower resolution, the 3D convolutional U-net used for super-resolution is responsible for densifying the decoded high-resolution blocks, increasing the overall reconstruction quality at no rate cost. From this point on, this solution will be referred to as Point Cloud Geometry Coding with Adaptive Super-Resolution and labelled as PCGC-ASR.

5.2 Architecture and Walkthrough

The PCGC-ASR's architecture is presented in Figure 20.

Each input PC is processed according to the following step-by-step walkthrough:

1. **PC Block Partitioning –** The input voxelized PC is divided into disjoint 3D binary blocks ('1' and '0' corresponding to filled and empty voxels, respectively) of a fixed target size, which are independently coded; the size of these blocks defines the random-access granularity.

2. **Basic Block Down-sampling –** Depending on the PC characteristics, each block may be down-sampled to a lower grid precision/resolution; this is typically advantageous when the PC is sparse, as it increases the density of the blocks to be encoded. Note that this process may be lossy, as

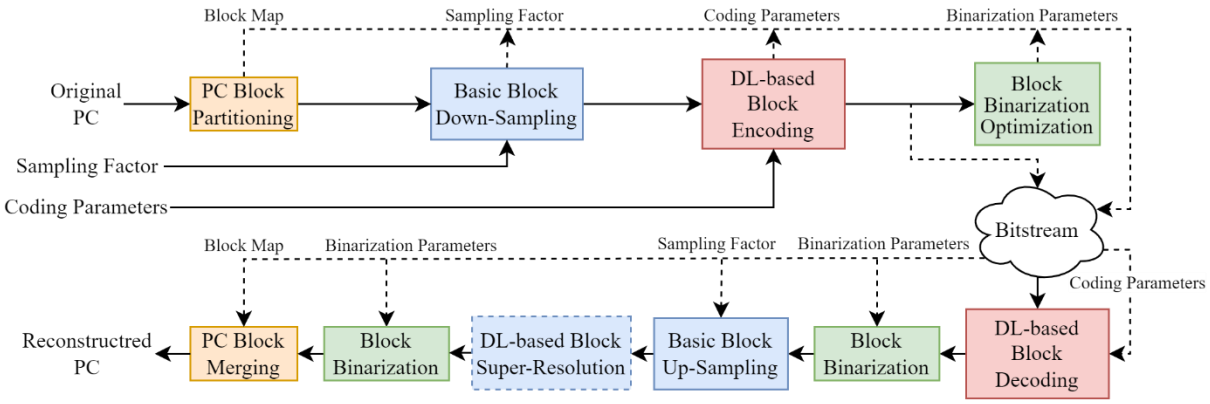


Figure 20: PCGC-ASR overall architecture, where yellow is used for block partitioning/merging, blue for sampling, red for coding, and green for binarization modules.

close standing points in high resolution may be merged into a single point in the lower resolution, thus losing information.

3. **DL-based Block Encoding** – Each block is encoded with an end-to-end DL coding model. This process can be compared to a typical transform coding approach; however, in this case, a convolutional autoencoder (AE) with Inception-Residual [24] Blocks (IRB) is used to learn a non-linear transform. The transform generates a set of coefficients, usually referred to as the *latent representation,* which are then explicitly quantized, by dividing them by a real-valued quantization step followed by rounding, and, finally, entropy coded. An adaptive entropy coding model is learned via a variational autoencoder (VAE) [43].

4. **Block Binarization Optimization** – For additional adaptability to different PC densities, this module optimizes the binarization process that will be applied at the decoder to each single voxel. Since the output of the DL coding model, at the decoder, does not immediately correspond to a binary block but rather to the probabilities of voxels being occupied, this optimization process produces a binarization parameter k, corresponding to the number of filled voxels that must be decoded for optimal reconstruction quality, according to some selected geometry quality metric. Naturally, k needs to be included in the coded bitstream.

5. **DL-based Block Decoding** – Blocks are decoded using the decoder counterpart of the DL-based block encoder mentioned before. Once again, this will produce a set of decoded probabilities corresponding to the likelihood of voxel occupancy.

6. **Block Binarization** – The voxels' decoded probabilities are binarized to reconstruct the PC coordinates using the binarization parameter value resulting from the block binarization optimization performed at the encoder.

7. **Basic Block Up-sampling** – When considering down-sampling at the encoder, each block must be up-sampled back to the original grid resolution/precision. The block up-sampling operation is very simple and does not recover any lost information with the down-sampling operation. Since the

number of points remains unaltered and the voxels become smaller, this module will output a sparser block.

8. **DL-based Block Super-Resolution [Optional]** – After up-sampling, the block can go through an advanced set up-sampling (or super-resolution) operation to mitigate the impact of the down-sampling losses in the reconstruction quality, at no rate cost. Like the DL-based decoder, this module generates a set of occupancy probabilities.

9. **Block Binarization** – After super-resolution, each block voxel must be once again binarized according to the binarization parameters.

10. **PC Block Merging** – The decoded blocks are merged to reconstruct the full PC.

5.3 Main Tools

This section aims at providing a more detailed presentation of the main tools that build the PCGC-ASR codec, including their impact on the final coding solution.

5.3.1 Basic Block Down/Up-Sampling

The basic block down and up-sampling modules are two key elements of the proposed coding solution as they allow to make the overall solution more flexible and versatile. It may be useful to remind here the definition of grid down-sampling, provided in Section 4.1, namely as the lossy operation in which the grid's resolution is decreased by a sampling factor, allowing to increase the voxel size and, consequently, the block density. In this context, these modules hold a dual purpose in the PCGC-ASR architecture:

1. **Sparser PC Coding** – As seen in Section 4, DL-based coding solutions that rely on a volumetric approach, such as ADL-PCC, often have uneven RD performance depending on the density of the PC being coded, namely struggling to produce competitive results for sparser PCs. The integration of the down-sampling module at the encoder will allow to densify every block before being coded, and consequently, enhance the compression performance of the DL model for originally sparser PCs.

2. **Lower Rates Coding** – When considering a DL coding approach, it is common to use multiple models to achieve a range of target rates. Nevertheless, training DL-based coding models for very low rate is often a difficult task due to the appearance of coding artifacts which deteriorate severely the reconstruction quality. However, considering down-sampling prior to coding will allow to reduce the block's resolution, thus reducing the amount of information and, consequently, requiring less rate to be coded. This simple approach allows achieving a wider range of rates while maintaining decent quality levels even at lower rates.

Regarding the block down-sampling decoder counterpart, the block up-sampling module is included at the decoder to assure the reconstructed PC has the same resolution as the input PC. This is an inherent requirement of the proposed coding solution since the main goal is to obtain the closest

possible representation of the original PC at the decoder/user side, notably having the same resolution.

5.3.2 Deep Learning-based Block Encoder/Decoder

Given the coding context of the proposed solution, it is vital to present how the coding itself is performed. The used DL coding model, shown in Figure 21a, is an evolution of the end-to-end DL model used in ADL-PCC [3].

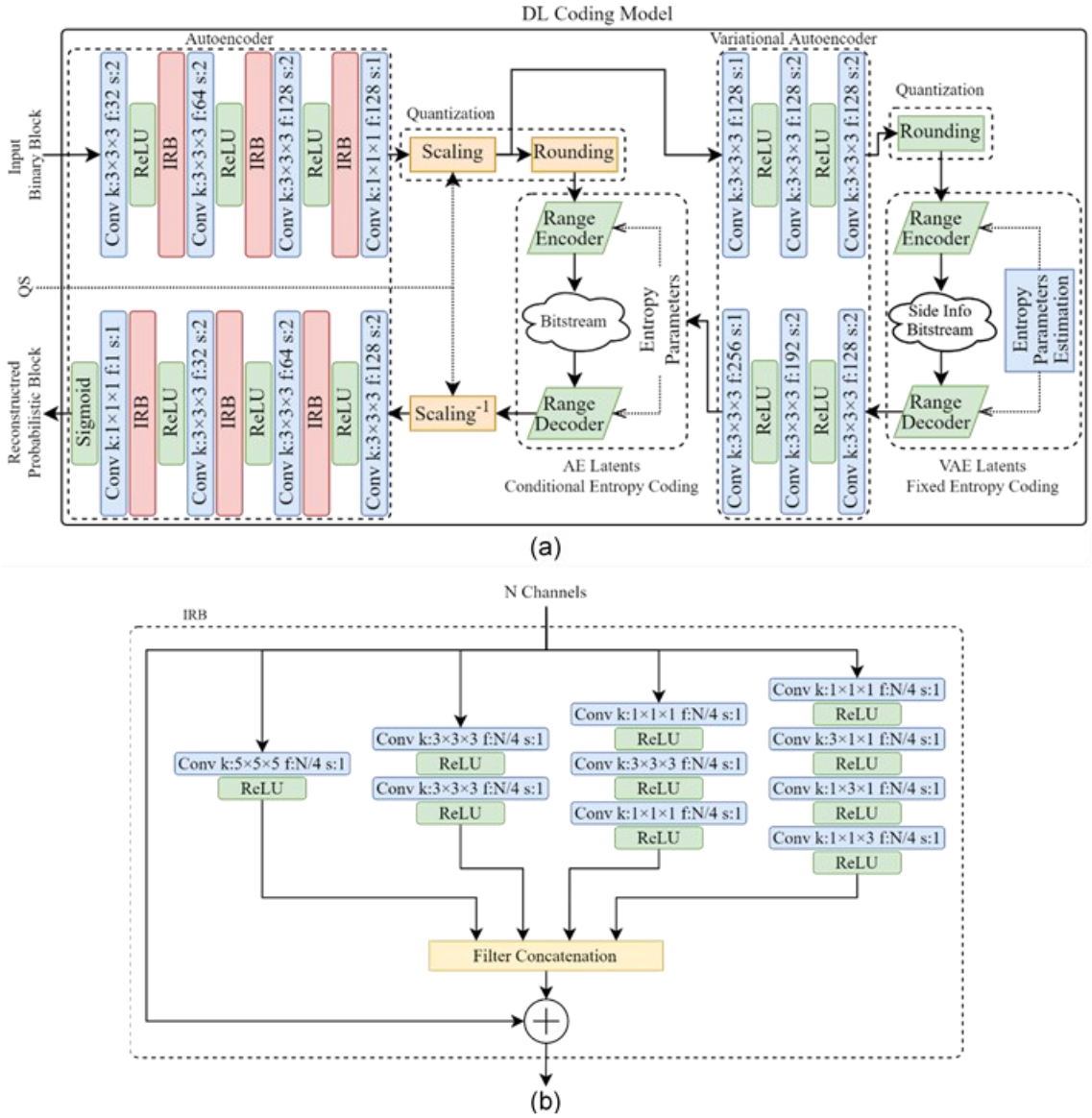


Figure 21: Overview on the PCGC-ASR DL coding model. (a) Overall DL coding architecture; (b) Detailed architecture of each Inception Residual Block (IRB).

At the encoder side, the input 3D binary block is processed by the following modules:

1. **Autoencoder (AE)** – The convolutional AE transforms the input binary 3D block into a latent representation of lower dimensionality. This latent representation consists of multiple feature maps,

which number depends on the adopted number of filters for the convolutional layers, in this case 128. The AE consists on a combination of 3D convolutional layers and Inception Residual Blocks (IRB). The IRB is inspired in the Inception-ResNet [24], a popular neural network used for diverse image processing tasks; it contains several convolutional layers in parallel with different filter support sizes, which allow to extract different types of features from varying neighboring scopes (from $5 \times 5 \times 5$ to $1 \times 1 \times 1$); in addition, a residual skip connection allows to propagate the features along the network, which also facilitates the training of deeper models. The number of filters starts from 32 in the first layer, and progressively increases to 128 at the final encoder layer, resulting in a rich latent representation. In summary, the AE contains a total of 2 842 016 trainable parameters, with 1 208 432 at the encoder side, and 1 633 584 at the decoder side.

2. **Quantization** - The AE latent representation is explicitly quantized before entropy coding. Considering a given quantization step, which can be any positive real value, the latents are first scaled by the quantization step, and then rounded to the closest integer. This explicit quantization approach allows to fine tune the target rate at coding time while using a single trained DL coding model. At training time, an implicit quantization approach is considered (i.e., a unitary quantization step), and the rounding is replaced by a differentiable approximation (necessary for backpropagation), which consists in adding uniform noise to simulate the quantization error [44].

3. **AE Latents Conditional Entropy Coding** – A conditional entropy bottleneck layer, from the Tensorflow compression library [5], is used to entropy code the block latent representation. This entropy coding approach uses a Gaussian mixture model conditioned on a hyperprior. During training, this layer estimates the entropy of the latent representation according to the entropy coding model, which is used for the RD optimization process. At coding time, a range encoder is used to create the block coded bitstream.

4. **Variational Autoencoder (VAE)** – A VAE is used to capture possible structure information still present in the block latent representation, which is then used as a hyperprior for the conditional entropy bottleneck. This way, the entropy coding model parameters can be more accurately estimated and adapted to each coded block. The mean-scale hyperprior as proposed in [43] was adopted. In this process, the VAE generates its own latent representation, which must also be coded and included in the bitstream as additional side information to the decoder, so that the entropy coding model parameters can be replicated at the decoder. The VAE has a similar but simpler design to the AE, with only 3 convolutional layers and no IRBs, at both the encoder and decoder. The VAE contains a total of 3 760 960 trainable parameters.

5. **VAE Latents Fixed Entropy Coding** – Similar to the conditional entropy coding, this module entropy codes the VAE latent representation. However, it uses a fixed entropy coding model for all blocks, which is learned during training, instead of an adaptive one, dynamically adapted to the block during inference/testing. As all the components of the end-to-end DL coding model are jointly trained, the additional side information rate is compensated by reducing the rate associated with the latents, thus optimizing the overall RD performance.

When compared to the starting point ADL-PCC architecture, the newly designed PCGC-ASR codec presents additional convolutional layers, notably with the introduction of the IRBs, allowing for a more effective feature extraction, and thus a more compact latent representation. Furthermore, the use of quantization and a more powerful VAE provide a finer rate control and compression efficiency, respectively. In total, the proposed DL coding model has 10 times more trainable parameters than the ADL-PCC model used. Note, however, that the proposed coding solution only needs one DL coding model, instead of the many used by ADL-PCC, to achieve a single RD trade-off.

5.3.3 Deep Learning-based Block Super-Resolution

As seen previously, performing block down-sampling prior to coding can have multiple benefits, such as achieving lower rates and boost the compression performance for sparser PCs. However, the information loss caused by this module can substantially penalize the reconstruction quality. Therefore, a SR post-processing tool is considered to densify the decoded block and mitigate the impact of such losses. The proposed DL-based SR module is based on Akhtar et. al. in [30] and is shown in Figure 22.

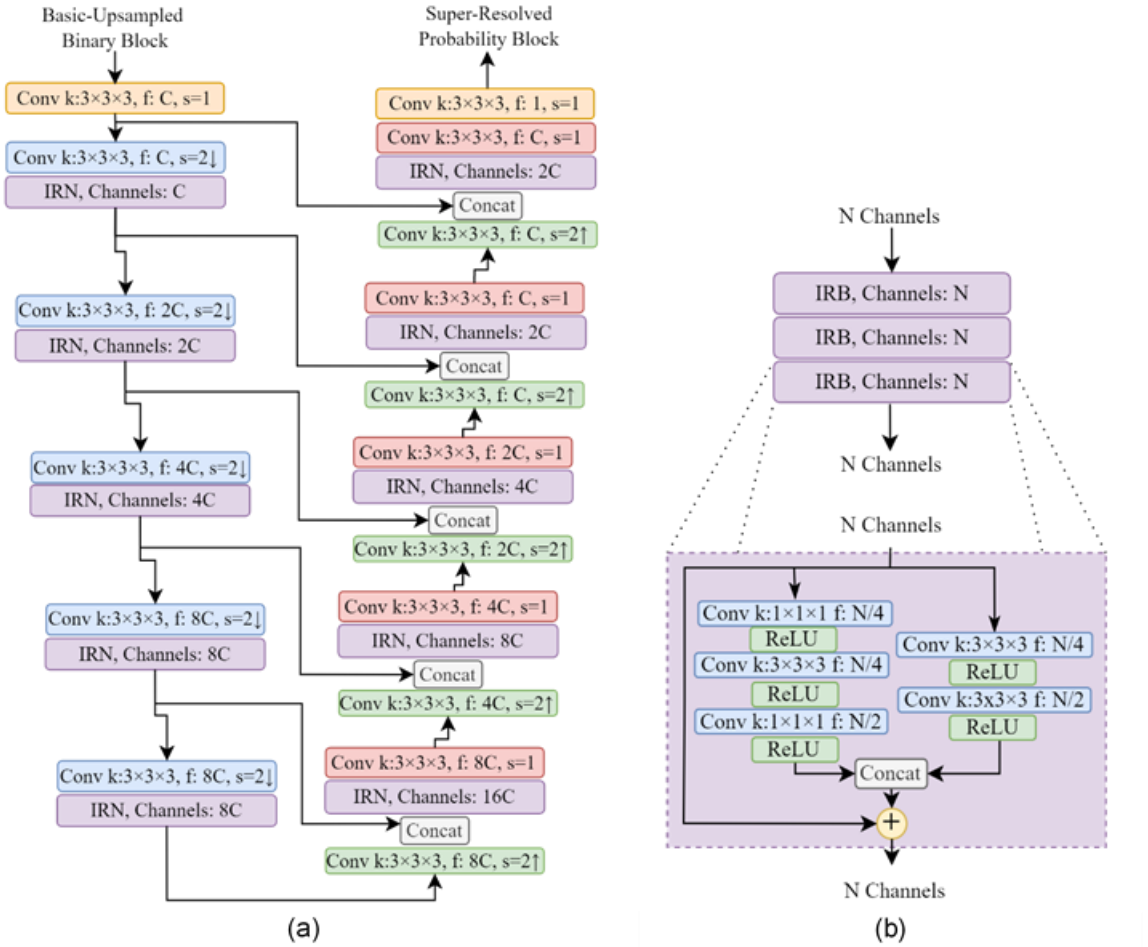


Figure 22: Proposed Super-Resolution module. (a) Overall DL Super-Resolution model architecture. (b) Inception Residual Network (IRN) detailed architecture.

The SR architecture consists on a 3D CNN shaped as a U-net [45], as shown in Figure 22b. The full architecture can be divided into two main processing stages:

1. **Contracting Path** – The descending path of the U-net performs multiscale feature extraction. Similarly to the AE in the DL coding model described in Section 5.3.2, it combines 3D down-sampling convolutional layers with IRB layers. However, the IRB layers are now stacked three at a time, thus creating a Inceptual Residual Network (IRN). On one hand, when compared to the IRBs used in the coding model, this are much simpler and lighter, with fewer and smaller filter supports (maximum of $3 \times 3 \times 3$). On the other hand, the DL SR model uses a much deeper network, with many more convolutional and IRBs layers. Furthermore, the number of channels/filter per convolutional layer grows progressively over the layers, what also contributes for a deeper network.

2. **Expanding Path** – The ascending path is nearly symmetrical to the contracting path, now successively up-sampling the features and aggregating them with the multiscale features extracted in the contracting path. By considering the features obtained at different scales, this path is able to accurately predict the occupation of the voxels, thus densifying the block, and consequently, mitigating the of losses originated by the down-sampling process performed at encoder.

Naturally, the DL-based Block Super-Resolution module can offer significant RD performance gains, especially for originally dense and uniform PCs. However, this is not always the case, depending on the PC characteristics, namely sparsity, heterogeneous point distribution or noise, and reconstruction quality of the DL-based codec (whether it contains many coding artifacts or not). For this reason, the SR module is an optional post-processing module. In total, the DL SR model has 7 288 893 trainable parameters.

5.3.4 Binarization Optimization

Both the DL coding and SR models produce an occupation probability for each voxel, thus requiring a binarization process during inference/testing time, in which the voxels' occupancy is determined based on a threshold. This threshold may either be manually fixed or could be adaptively selected. In this sub-section, the focus will be on the latter since this makes the performance less dependent on the training conditions of the DL coding model as shown by the obtained performance results.

As such, the so-called *optimized Top-k binarization* approach was adopted for selecting the occupied voxels, notably selecting as points the k voxels with the largest probabilities, with k being defined as:

$$k = N_{input} \times \beta, \tag{9}$$

where N_{input} is the number of input points of the original block, and β a factor optimized at the encoder by determining its value which leads to the best reconstructed quality, assessed by one of two metrics: PSNR D1 or PSNR D2.

To balance the computational complexity and RD performance, and since binarization must happen

twice, i.e. both after the DL-based decoder and super-resolution modules, two binarization optimization approaches are proposed:

1. **Double Binarization Optimization** - Two distinct β values are optimized for coding binarization and SR binarization. This requires SR optimization during the encoding process which increases considerably the encoding complexity.

2. **Single Binarization Optimization** - The optimization process is performed only once for the coding model and the same β value is used for SR, thus at no increased complexity cost. Note that while achieving good RD performance, notably for dense PCs, this approach is not optimal.

Naturally, only one or two binarization parameters, i.e. number of reconstructed points (k), need to be included in the bitstream to be available to the binarization modules at the decoder.

For the final performance assessment, a PSNR D1 double binarization optimization process will be considered, as it leads to the optimal RD performance.

5.4 Training Process

This section describes the training process of the DL coding and SR models. Despite sharing some similarities, it is important to note that these models were not trained jointly, hence there is no global end-to-end training. Both training processes followed the training conditions described in the JPEG Pleno CTTC [46].

5.4.1 Training Datasets

For the training dataset, 24 static PCs were selected from the JPEG Pleno CTTC [46] training dataset. This dataset is used to make the network learn to fulfil its goal. However, DL networks tend to become biased towards the training data when overtrained, also known as overfitting, hence losing generalization capabilities, i.e. achieving poorer performance over unseen data. To solve this problem a second dataset, the so-called validation dataset, must be involved in the training process. This dataset was composed of four new PCs, from the JPEG Pleno CTTC [46] training data, used to verify when overfitting starts to occur and thus terminate the training process. This overfitting protection mechanism is usually referred to as Early Stopping. Note that this dataset will never affect the training process directly.

A brief summary of the composition of these datasets is shown in Table 3.

DL Coding Model

To train the DL coding model, both training and validation datasets were used. Firstly, each PC was down-sampled according to its sparsity, ensuring the overall density of the dataset. This is an important pre-processing step as it mimics the behavior of the basic down-sampling module in the proposed pipeline, allowing the network to learn how to code dense PCs, as intended. Secondly, given the block-level scope of the coding operation, each of the training and validation PCs was divided into blocks of fixed size, in this case $64 \times 64 \times 64$. In total, 35 861 and 3 822 blocks were used during training and validation, respectively. Note that blocks with less than 500 points were not considered as

Table 3: PC composition of the training and validation datasets.

Dataset	Point Cloud	Frame	Original Precision	Original Points	Training Precision	Training Points
Training	*Loot*	1200	10	805 285	10	805 285
	Redandblack	1550	10	757 691	10	757 691
	Soldier	690	10	1 089 091	10	1 089 091
	Thaidancer	Viewdep	12	3 130 215	11	1 007 956
	Andrew10	1	10	1 276 312	10	1 276 312
	David10	1	10	1 492 780	10	1 492 780
	Sarah10	1	10	1 355 867	10	1 355 867
	The20sMaria	600	-	10 383 094	11	3 681 165
	UlliWegner	1400	-	879 709	10	537 042
	Basketball_player	200	11	2 925 514	11	2 925 514
	Exercise	1	11	2 391 718	11	2 391 718
	Model	1	11	2 458 429	11	2 458 429
	Mitch	1	11	2 289 640	11	2 289 640
	ThomasSenic	170	11	2 277 443	11	2 277 443
	Football	1365600	11	1 021 107	11	1 021 107
	Façade 15	-	14	8 907 880	12	6 834 258
	Façade 64	-	14	19 702 134	12	12 755 151
	Egyptian_mask	-	12	272 684	9	269 739
	Head_00039	-	12	13 903 516	12	13 903 516
	Shiva_00035	-	12	1 009 132	10	901 081
	ULB_Unicorn	-	13	1 995 189	10	1 588 315
	Landscape_00014	-	14	71 948 094	12	15 270 319
	Stanford_Area_2	-	16	47 062 002	12	19 848 824
	Stanford_Area_4	-	16	43 399 204	12	24 236 048
Validation	*Boxer*	viewdep	12	3 493 085	11	30 56 129
	Dancer	1	11	2 592 758	11	2 592 758
	Façade 09	-	12	1 596 085	11	1 560 834
	Frog_00067	-	12	3 614 251	11	3 321 097

they hold too little information and may ‘poison’ the training process.

DL Super-Resolution Model

The dataset pre-processing for the DL SR model was very similar to the DL coding model, with the exception of the block size. In this case, the block size is defined by the sampling factor being used. In total, 2 SR models were trained, notably for sampling factor 2 with block size of $64 \times 64 \times 64$, and sampling factor 4 with a block size of $128 \times 128 \times 128$. After partitioned and down-sampled, each PC block was then basic up-sampled back to the original resolution. Once again, the idea is to recreate the conditions in which this DL SR model will be used during inference/testing. There is no coding involved in the training process of the SR model since the idea is for it to learn how to mitigate the basic down-sampling losses. Some preliminary tests have shown this approach leads to better RD performance than trying to learn both down-sampling losses and coding artifacts together as it would happen if coding was involved in its training process.

5.4.2 Loss Function

This sub-section addresses the loss function, a fundamental component of the training process.

DL Coding Model

For the DL coding model, the same loss function used in the ADL-PCC training was considered. This

loss function represents a RD trade-off, where the coding rates are estimated by the entropy of the AE and VAE latent representations, and the distortion is measured by the FL, described in Section . It is also important to recall the impact of the Lagrangian multiplier, λ, leading to different RD trade-off levels. In summary, the loss function can be represented as shown in (*10*).

$$Loss\ Function = FL(\alpha, \gamma = 2) + \lambda \times Coding\ Rate \quad (10)$$

In ADL-PCC, the α parameter assumed several different values, thus creating multiple DL coding models to provide a content-adaptive behavior to the codec. However, since a new approach to deal with different densities is proposed for PCGC-ASR, only a single α, $\alpha = 0.7$, is needed.

As for ADL-PCC, distinct models must be trained to achieve different RD trade-offs, what is accomplished by varying the λ parameter in (2). A total of 6 models were trained, using λ = 0.00025, 0.0005, 0.001, 0.0025, 0.005, and 0.01.

DL Super-Resolution Model

The DL SR model does not require a RD-based loss function since it is not a coding tool, thus not consuming any rate. For that reason, the loss function used exclusively the FL with $\alpha = 0.7$, exactly as used to train the PCGC-ASR DL coding model. Note that this loss function is measured using the original blocks as reference and not the down and up-sampled blocks that are available at the DL SR model's input.

5.4.3 Training Strategy and Hyperparameters

DL Coding Model

The six PCGC-ASR DL coding models were trained following a sequential training strategy. In this case, the first model to be trained is the one with the easiest task, i.e. with the higher rate and lower distortion trade-off, obtained for the lowest λ parameter. Once its training is terminated by the early stopping algorithm described in Section 5.4.1, this DL coding model is stored. At this stage, the loss function is updated to match the next RD trade-off (updated λ) and the weights of the previous trained model are loaded as starting point for the new model. This process is repeated until the sixth model (for the lowest rate) is trained. The performed sequential training allows to significantly speed up the overall training process and even to obtain some RD performance gains.

Each of the DL coding model contains 6 602 976 parameters that were trained using the Adam algorithm [19] with a learning rate of 10^{-4} and minibatches of 16 blocks. As for the early stopping algorithm, a patience of 5 epochs was considered. This means that the training would be interrupted if the model's validation loss did not decrease over 5 consecutive epochs.

DL Super-Resolution Model

Each of the two DL SR models contains 7 288 893 parameters that were trained independently using the Adam algorithm [19] with a learning rate of 10^{-4} and minibatches of 8 and 1 blocks for sampling factors of 2 and 4, respectively. Once again, an early stopping patience of 5 epochs was considered.

5.5 Performance Assessment

This section reports the PCGC-ASR performance for representative test material under meaningful test conditions.

5.5.1 Test Material and Conditions

For the objective performance assessment, the 25 PCs used by JPEG to test the codecs submitted as response to the Final Call for Proposals on JPEG Pleno PCC [6] were used. A summary of the PCs in the test dataset is shown in Table 4. In addition, a rendering of these PCs is shown in Figure 23.

Table 4: Test PCs' summary. The sparsity values correspond to the average 20-nearest neighbor's distance.

PC Name	Precision	Number of Points	Sparsity
RWT2	10	2 308 312	1.610369
RWT34	10	2 212 435	1.614183
RWT53	10	1 563 866	1.60725
RWT70	10	1 871 158	1.60316
RWT120	10	1 880 897	1.614311
RWT130	10	3 150 249	1.602903
RWT134	10	2 350 417	1.61968
RWT136	10	835 264	1.571434
RWT144	10	4 143 911	1.719278
RWT152	10	3 402 890	1.683609
RWT246	10	3 486 192	1.663971
RWT305	10	1 859 311	1.621199
RWT374	10	1 658 288	1.60332
RWT395	10	655 412	1.545843
RWT430	10	3 839 604	1.716197
RWT462	10	813 808	1.570685
RWT473	10	2 790 723	1.634813
RWT501	10	2 404 242	1.614124
RWT503b	10	291 955	1.549672
RWT529	10	663 367	1.600387
goat_skull	10	764 536	2.059569
kinfuscene0043	10	238 739	3.734644
kinfuscene0069	10	308 150	4.700099
PC_09	10	661 892	2.435324
RuaDeCoimbra_vox10	10	492 052	1.692315

(a) *RWT2*

(b) *RWT34*

(c) *RWT53*

(d) *RWT7*

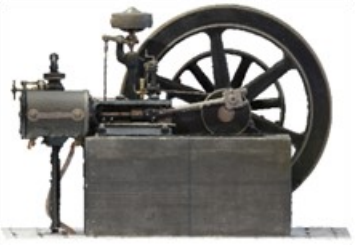
(e) *RWT120*

(f) *RWT130*

(g) *RWT134*

(h) *RWT136*

(i) *RWT144*



(j) *RWT152*

(k) *RWT246*

(l) *RWT305*

(m) *RWT374*

Figure 23: Example rendering for the test PCs.

Furthermore, the test conditions described in the JPEG Pleno PCC CTTC [46] were adopted, namely the desired target rates for PC geometry coding. In total, four target coding rates were considered, namely 1.5, 0.5, 0.15, 0.05 bpp with a 10% error margin, thus creating four target rate ranges.

5.5.2 Performance Metrics

To assess the RD performance of the proposed PCGC-ASR geometry codec, two objective geometry quality metrics were considered: a point-to-point metric (PSNR D1) and a point-to-plane metric (PSNR D2) following the recommendations in the JPEG Pleno PCC CTTC [46]. Furthermore, for a better comparison over the entire test dataset, the Bjontegaard-Deltas (BD) for rate and both PSNR D1 and PSNR D2 will also be considered.

5.5.3 PCGC-ASR Coding Configurations and RD Performance

As seen in Section 5.4, the DL-based coding solutions are trained for specific target RD trade-offs. Therefore, achieving a specific bitrate at testing time without the possibility of training new models may be challenging. As such, the proposed geometry coding solution considers a set of input parameters that allow to have a more flexible coding configuration at testing time. These parameters are presented below, ordered by their rate control granularity, in this case in a coarser-to-finer fashion:

- **Sampling factor**: This coarser rate control parameter allows achieving higher rates when using a unitary sampling factor and progressively lower ones when considering sampling factors of 2 and 4.
- **Super-Resolution**: Associated to the sampling factor, it switches on or off the optional SR module. Despite not directly affecting the coding rate, SR may significantly improve the reconstruction quality obtained, when considering a sampling factor larger than 1, at no rate cost, thus improving the RD performance of such sampling factors.
- **DL coding model**: Another coarse rate control parameter. As mentioned before, there are six DL coding models available, each trained for a different RD trade-off (distinct λ values), spanning a wide range of rates. The achieved rate decreases with a growing λ value, and vice-versa.
- **Block size**: A finer rate control parameter that defines the size of the 3D blocks. The block size may be $256 \times 256 \times 256$, $128 \times 128 \times 128$ or $64 \times 64 \times 64$. Note, however, that not all block sizes are available for all sampling factors due to memory limitations.
- **Quantization step**: A very fine rate control parameter that is applied to the AE latents quantization and assumes any positive value larger or equal to 1. A growing quantization step will progressively reduce the used coding rate. This parameter is useful to adjust RD points which are slightly above the target; by default its value is 1.

Naturally, there are multiple possible combinations of these parameters, each representing a distinct RD trade-off. For each PC, these configurations, i.e. RD points, can be represented in a single RD plot to help visualizing which configurations achieve the desired target rates. An example for a test PC, *RWT501*, is shown in Figure 24.

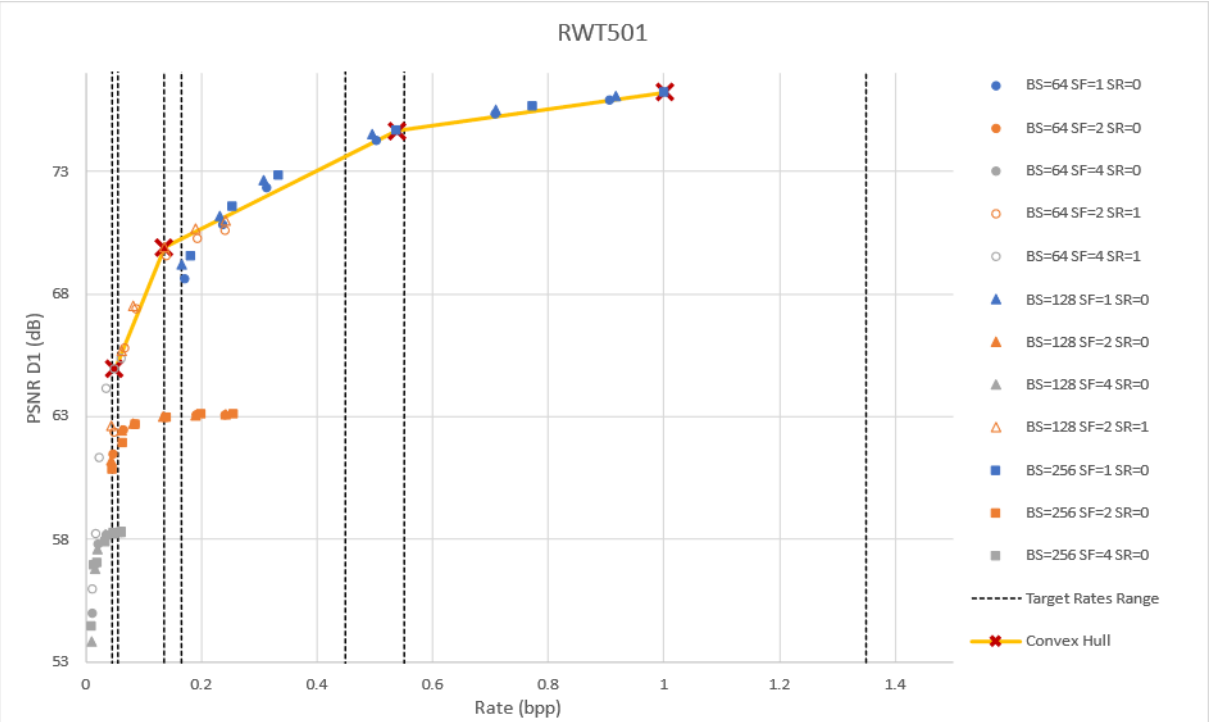
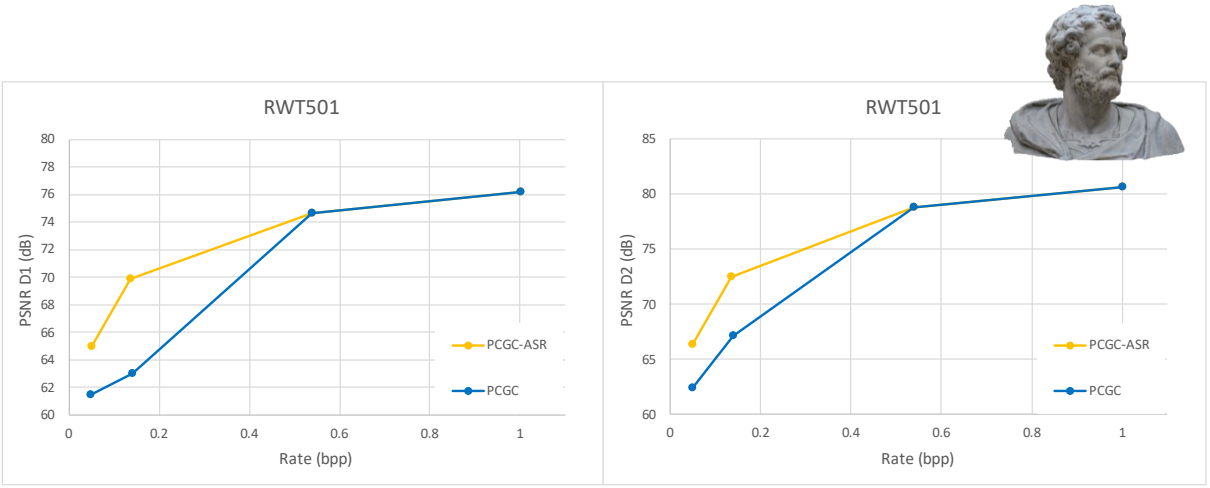


Figure 24: Example of an RD plot containing multiple coding configurations for the *RWT501* PC. Each input parameter is represented by its initials, except the unitary quantization step that is omitted.

Regarding the RD plot example in Figure 24, all the RD points consider a unitary QS, since this was enough to meet the selected target rates, thus larger QS were not considered. Note, however, this may not be the case for all PCs, hence still being important to have such option available. In Figure 24, the RD points that satisfy the target rate conditions are marked with a red cross. The full set of optimal RD points may be connected using a single convex hull curve (in yellow) which depicts the overall RD performance of the proposed PCGC-ASR codec for this PC. Furthermore, it is important to note that, for many dense PCs like *RWT501* the highest target rate of 1.5 bpp is never reached. While it would be possible to train a model for a smaller λ (that could achieve 1.5 bpp for denser PCs), this would bring a negligible quality gain as the quality is already near-lossless.

Consider now Figure 25, where the RD performance of two codec configurations is show for the same PC, *RWT501*. The plotted RD curves, labeled as PCGC-ASR and PCGC, correspond to the convex hull curves obtained when considering and not considering the SR model, respectively. Note that, for the latter, the suffix ASR is dropped since no adaptive SR is performed.

As it can be seen in Figure 25, since *RWT501* is a rather dense PC, the SR model is very impactful on the final RD performance when targeting lower coding rates, and therefore using a sampling factor larger than 1. As previously mentioned, this module does not affect the coding rate; however, it improves drastically the reconstruction quality, reaching up to 7dB and 4.5 dB gains for PSNR D1 and PSNR D2, respectively. Naturally, the solution labelled as PCGC-ASR (in yellow) will be the one considered during the RD performance benchmarking presented in the next sub-section.



(a) (b)

Figure 25*:* RD performance of the proposed PCGC-ASR coding model with and without Adaptive SR (ASR) for the *RWT501* PC, considering (a) PSNR D1 and (b) PSNR D2 as evaluation metric.

5.5.4 Benchmarking RD Performance

The benchmarking solutions must be the relevant PC coding solutions that establish a meaningful and fair reference for the coding solution under examination, in this case, PCGC-ASR. For that reason, the MPEG PCC standards, reviewed in Chapter 2, are considered, naturally here considering only the geometry component. Since the proposed coding solution targets lossy compression of static PCs, the MPEG PCC standards must do the same, thus a V-PCC Intra configuration is used in addition to G-PCC Octree. Additionally, since the proposed PCGC-ASR is an evolution of the ADL-PCC solution, reviewed in Section 3.1, it is natural to also consider it as a benchmarking solution.

The RD performance of the PCGC-ASR codec for the test dataset, measured as BD-Rate and BD-PSNR, is shown in Table *5*, having as reference each benchmarking solution. Additionally, the RD performance for two test PCs, *RWT462* (a denser PC) and *kinfuscene0063* (a sparser PC)*,* are shown in Figure 26. Note that the BD results obtained with ADL-PCC as a reference must be taken with a *grain of salt* since only part of the RD curves will be considered, due to the non-overlapping segments (between ADL-PCC and PCGC-ASR curves) being discarded while computing the BD-Rate/PSNR.

The RD Performance results can be summarized as follows:

- **PCGC-ASR vs G-PCC Octree** – The proposed PCGC-ASR solution clearly outperforms G-PCC Octree for all test PCs. The PCGC-ASR codec achieves average rate savings of 81.42% and 73.23% and average BD-PSNR gains of 7.43 dB and 6.53 dB, for PSNR-D1 and PSNR-D2 metrics, respectively. Regarding the RD performance shown in Figure 26, PCGC-ASR outperforms G-PCC Octree for all target rates. A similar conclusion can be made regarding PC density, however, the gains obtained with of PCGC-ASR over G-PCC Octree decrease when going from a denser to a sparser PC. This is expected as G-PCC Octree is more efficient for sparser PC coding.
- **PCGC-ASR vs V-PCC Intra** – The proposed PCGC-ASR solution outperforms V-PCC Intra for all test PCs, except for *RWT134*, *RWT246* and *RWT430*. As shown in Figure 26, these three PCs

Table 5: BD-Rate and BD-PSNR for the proposed PCGC-ASR solution for the entire test dataset using the benchmarking coding solutions as reference.

Reference	G-PCC Octree				V-PCC Intra				ADL-PCC			
PC Name	PSNR D1		PSNR D2		PSNR D1		PSNR D2		PSNR D1		PSNR D2	
	BD-Rate (%)	BD-PSNR (dB)	BD-Rate (%)	BD-PSNR (dB)	BD-Rate (%)	BD-PSNR (dB)	BD-Rate (%)	BD-PSNR (dB)	BD-Rate (%)	BD-PSNR (dB)	BD-Rate (%)	BD-PSNR (dB)
RWT2	-84.2	7.7	-77.2	6.9	-30.2	1.8	-28.8	2.0	-47.6	2.5	-49.2	3.1
RWT34	-89.0	9.3	-81.2	8.4	-40.5	1.9	-32.0	1.9	-55.6	2.9	-52.4	3.1
RWT53	-86.7	8.6	-78.4	7.3	-38.0	1.7	-31.0	1.7	-50.4	2.7	-48.6	3.1
RWT70	-82.5	7.3	-73.8	6.0	-55.2	4.4	-49.5	4.4	-44.0	2.1	-40.5	2.2
RWT120	-85.0	8.1	-74.8	6.7	-8.8	2.6	5.5	2.3	-40.2	2.0	-33.9	1.9
RWT130	-79.8	6.5	-62.9	4.8	-51.7	3.4	-48.5	4.4	-47.2	2.3	-48.5	2.8
RWT134	-93.3	9.8	-89.2	9.2	0.7	0.0	11.6	-0.4	-59.7	2.6	-55.6	2.8
RWT136	-89.8	10.1	-83.8	9.0	-42.9	2.0	-34.1	1.9	-47.3	2.6	-45.2	2.9
RWT144	-91.6	8.4	-88.3	8.3	-26.7	1.6	-21.3	1.6	-67.2	2.4	-61.2	2.3
RWT152	-78.4	6.2	-67.7	4.9	-66.9	4.2	-63.9	4.3	-49.1	2.5	-41.0	2.4
RWT246	-93.0	8.8	-88.7	8.3	27.5	-0.8	50.7	-1.3	-63.2	2.4	-56.9	2.4
RWT305	-83.0	7.4	-68.3	5.4	-28.1	1.2	-25.2	1.6	-58.9	3.6	-58.8	4.2
RWT374	-89.1	9.3	-81.3	8.2	-44.5	3.9	-21.3	3.6	-48.4	2.4	-45.1	2.7
RWT395	-83.5	8.5	-71.5	7.1	-62.6	6.6	-48.9	6.0	-45.0	2.6	-45.2	3.0
RWT430	-90.4	7.2	-81.8	6.5	31.8	-1.0	187.7	-3.7	-62.4	2.2	-48.9	2.1
RWT462	-90.7	10.3	-85.8	9.5	-45.5	2.3	-41.7	2.6	-47.8	2.6	-47.0	3.0
RWT473	-82.1	7.3	-70.8	6.2	-25.6	3.2	0.3	2.2	-47.8	2.4	-44.6	2.5
RWT501	-85.8	8.1	-75.4	6.9	-38.1	3.9	-9.4	2.8	-46.4	2.1	-43.5	2.3
RWT503b	-76.9	7.7	-59.5	5.9	-36.2	3.4	-33.2	3.9	-35.5	2.1	-33.1	2.2
RWT529	-89.0	10.1	-81.6	8.7	-44.9	4.1	-24.6	4.1	-52.6	2.9	-51.7	3.5
goat_skull	-79.1	5.2	-76.6	5.7	-58.1	2.4	-62.0	4.0	-55.3	1.4	-56.4	2.4
kinfuscene0043	-49.3	3.0	-42.9	2.8	-29.8	1.0	-31.0	1.0	-65.6	3.6	-33.7	1.2
kinfuscene0069	-57.4	3.7	-52.7	3.9	-42.7	1.1	-51.1	2.5	-86.7	9.2	-69.7	6.1
PC_09	-52.7	2.3	-50.5	2.1	-50.7	1.2	-77.9	3.6	-64.2	2.1	-51.1	1.5
RuaDeCoimbra_vox10	-73.1	4.7	-66.1	4.5	-46.1	3.3	-64.2	6.0	-40.9	1.5	-47.0	2.4
Average	-81.4	7.4	-73.2	6.5	-34.2	2.4	-21.8	2.5	-53.2	2.7	-48.3	2.7

represent objects with simpler shapes, smooth/planar surfaces, and without obstructions, hence making V-PCC Intra coding very efficient, and harder to outperform. Nevertheless, PCGC-ASR achieves average rate savings of 34.16% and 21.75% and average BD-PSNR gains of 2.38 dB and 2.53 dB, for PSNR-D1 and PSNR-D2 metrics, respectively. Considering the plots in Figure 26, it is clear that the proposed PCGC-ASR also outperforms V-PCC Intra regardless of target rate and PC density. When coding either PCs, V-PCC presents competitive performance, being the benchmarking closer the proposed solution, despite failing to achieve the lowest target rate for the sparser PC.

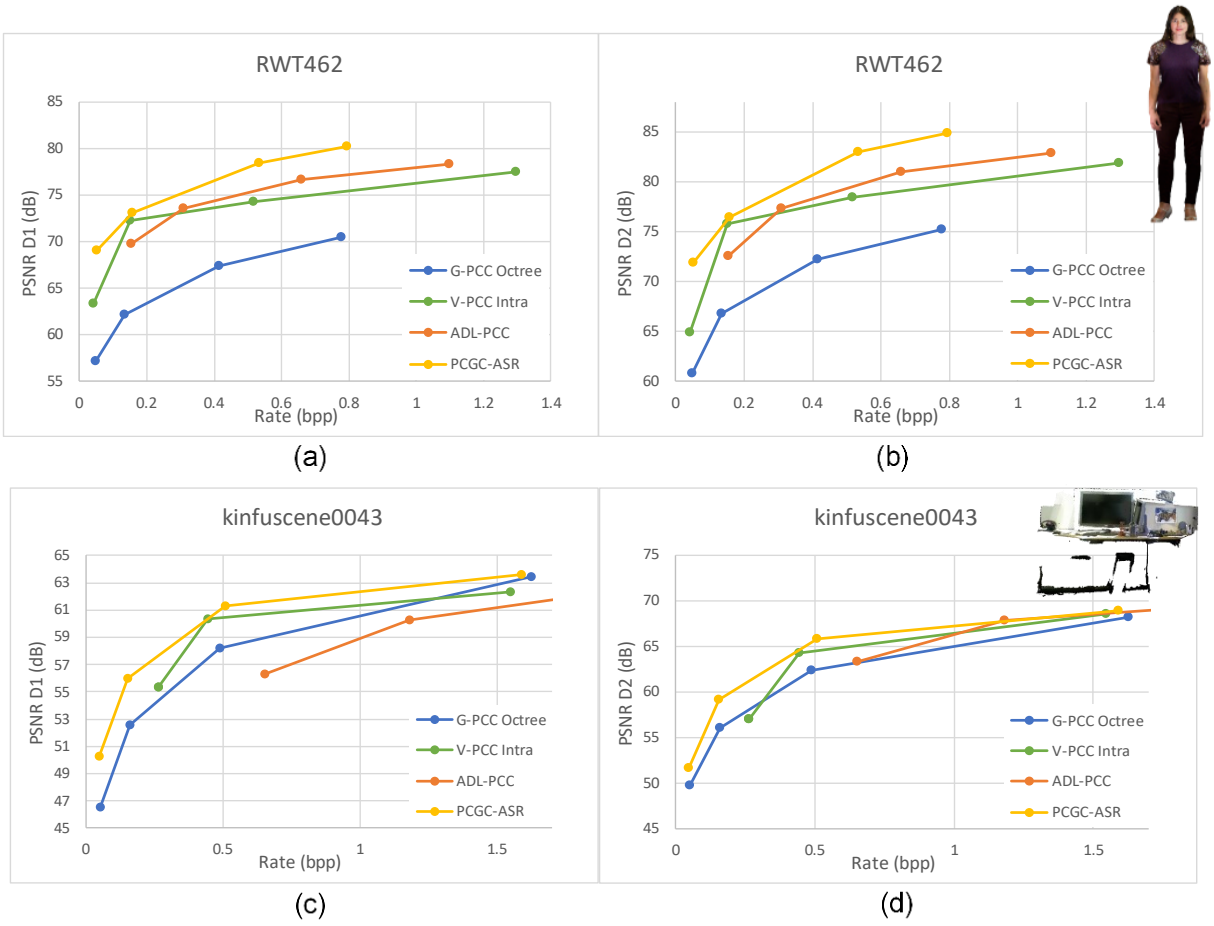


Figure 26: RD Performance of PCGC-ASR and benchmarking solutions for *RWT462* considering PSNR D1(a) and PSNR D2 (b); for *kinfuscene0043* considering PSNR D1 (c) and PSNR D2 (d).

Note that outperforming this metric is a hard, hence remarkable achievement as this codec relies on very powerful and efficient 2D video codecs, such as VVC, that have been developed and refined for many decades.

- **PCGC-ASR vs ADL-PCC** - As expected, the proposed PCGC-ASR also outperforms ADL-PCC for all test PCs, achieving average rate savings of 53.2% and 48.3% and average BD-PSNR gains of 2.7 dB for both PSNR-D1 and PSNR-D2 metrics. Note how these average values stand in between the results for G-PCC Octree and V-PCC Intra as it was already the case when assessing the ADL-PCC performance in Section 3.1.4. Regarding the RD performance shown in Figure 26, in particular for a denser PC like *RWT462*, ADL-PCC does not meet the target rates due to the lack of flexibility, as it only holds a single rate control parameter, the DL coding model, unlike PCGC-ASR. Additionally, it is visible that both PCGC-ASR and ADL-PCC curves have similar shape, however, the former is shifted up, i.e. higher-quality coding, due to the existence of richer and more meaningful latents leading to less coding artifacts, and shifted to the left, i.e. using less rate, thanks to the use of sampling and SR. When coding a sparser PC like *kinfuscene0043*, ADL-PCC fails to achieve coding rates lower than 0.65 bpp whereas PCGC-ASR can go to 0.05 bpp while maintain top-notch quality, thanks to its adaptive SR approach.

In summary, the obtained RD performance results show the efficiency and flexibility of the proposed PCGC-ASR coding solution. The G-PCC Octree benchmark is outperformed for all test PCs. However, this is not always the case for V-PCC Intra, as a minority of the test PCs hold very specific characteristics that benefit the performance of this MPEG standard. Nevertheless, V-PCC Intra is outperformed by PCGC-ASR for most of the PCs in the test dataset, which is a remarkable achievement given how V-PCC relies on 2D video codecs that have decades of development and fine-tuning. Finally, PCGC-ASR clearly outperforms ADL-PCC regardless of the PC density and target rate. This comes to prove the benefit of using the newly designed and improved DL coding model, as well as the importance of the adaptive down and up-sampling with SR approach in achieving lower rates and improving the compression performance for sparser PCs.

6 Proposing an Automatic Parameter Selection Mechanism for Point Cloud Geometry Coding Rate Control

In this chapter, a novel automatic parameter selection mechanism is proposed to perform rate control for the PC geometry coding solution proposed in Chapter 5.

6.1 Objective and Technical Approach

The proposed PC geometry coding solution, labeled PCGC-ASR, relies on four user-controlled input parameters, notably block size, sampling factor, coding model, and quantization step, to define how the PC should be coded, and thus to control the desired RD trade-off. The selection process of these coding parameters, when aiming to achieve a certain target rate, may be a tedious and computationally intensive task, especially when considering a brute-force approach, like in Section 5.5.3. In this chapter, an automatic parameter selection mechanism is proposed, allowing to perform rate control for the PCGC-ASR coding solution. Among the four input parameters, the sampling factor and coding model are the two most impactful as they lead to large rate variations, as seen in Section 5.5.3, and thus being the ones considered in the proposed rate control mechanism; whereas the values of the remaining parameters, which provide a finer rate tuning, have been fixed to reduce the problem's complexity, notably using a unitary quantization step and a block size of $64 \times 64 \times 64$.

To better understand the context where this mechanism will be used, three distinct types of rates must be defined:

- **Target Rate –** Commonly used in rate control, the Target Rate is defined by the user, establishing the maximum acceptable coding rate; while going above this rate may have serious negative consequences, a margin of 10% is allowed in the proposed mechanism.
- **Optimal Rate** – The rate below the Target Rate (with the 10% upper margin) which maximizes the RD performance; this rate is obtained by varying the above considered coding parameters.
- **Actual Rate** – The rate achieved by the codec when adopting the automatically selected coding parameters; ideally, the Actual Rate would match the Optimal Rate.

The goal of the proposed rate control mechanism would be the selection of the optimal sampling factor and DL coding model to code the input PC with the desired Target Rate. Given the coding parameters' discrete nature, this task can be understood and addressed as a *classification* problem, where the goal is to predict the optimal class/label corresponding to a unique sampling factor and coding model pair. In total, there are 18 classes corresponding to the 18 possible configurations between the three sampling factors and the six DL coding models defined in Section 3.1.3. Under this approach, the rate

control mechanism corresponds to a DL-based classifier that takes as input a set of statistical features, representing meaningful PC characteristics, to predict the optimal class/label and, consequently, select the optimal sampling factor and coding model configuration to code the input PC, considering the user-defined Target Rate.

For better understanding of this chapter, it is useful to immediately define the four main phases involved in the selection and assessment of the final rate control classification model:

1. **Training** - The best classifier's architecture is unknown à priori, therefore in this stage, a massive training process was conducted, training a total of 1500 distinct architectures/models instead of a single hand-designed architecture.
2. **Validation** - Afterwards, a performance-based ranking of the trained classifier models is obtained using a validation dataset; at this stage, all models are discarded with the exception of the top-3 best performing in terms of classification accuracy, which will be considered in the next step, the model selection.
3. **Selection** - This phase chooses the best architecture for the final rate control mechanism; to do so, the three previously identified models are assessed using a new dataset and a set of meaningful performance metrics. Naturally, the best performing in these conditions will be the selected model.
4. **Assessment** - Lastly, the selected model goes through a final performance assessment using a largely accepted dataset that will determine how well it generalizes, hence showing the overall performance of the proposed rate control mechanism.

More details on these four stages will be provided from Sections 6.3 to 6.6.

6.2 Architecture and Walkthrough

The architecture of the PCGC-ASR codec extended with the Automatic Rate Control (ARC) Parameter Selection mechanism is shown in Figure 27.

This chapter's contribution to the architecture shown in Figure 27a is exclusively the addition of the ARC mechanism, thus its architecture (Figure 27b) must be carefully presented. As input, the ARC receives the original PC, as well as the Target Rate. These inputs are processed sequentially through the following modules:

1. **Multi-Resolution Statistical Feature Extraction** – Responsible for extracting meaningful statistical features from the original PC, at multiple resolutions, to be used by the DL-based classifier. Firstly, a grid down-sampling operation is performed using the three sampling factors available in PCGC-ASR, i.e. sampling factors 1, 2, and 4. Afterwards, for every PC resolution, the following set of statistical features is extracted:
    - **Number of PC Points** – Provides information regarding the overall PC size and, since considered at multiple resolutions, shows how the PC changes with down-sampling, namely how much information is lost during this process.
    - **Average and Standard Deviation of Point Distance** – Provide information regarding the overall PC density (average) and uniformity (standard deviation). Additionally, when compared

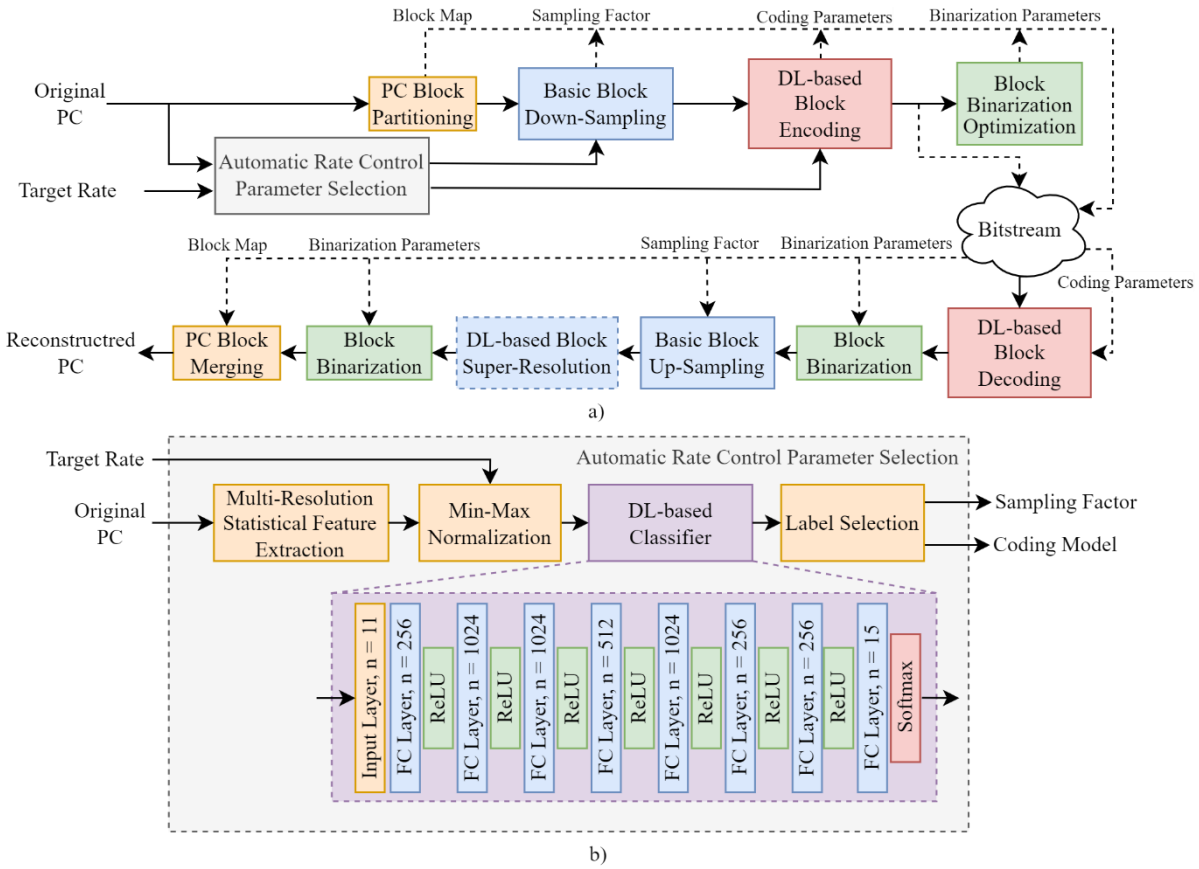


Figure 27: (a) Overall architecture of the proposed PCGC-ASR codec extended with ARC;

(b) ARC's detailed architecture and DL-based Classifier model's architecture.

at different resolutions, they evaluate the down-sampling effects on the PC density. These features are computed, for each point in the PC, using the 5 nearest neighbor's average distance, at each resolution.

- **Average and Standard Deviation of Point Distance** – Provide information regarding the overall PC density (average) and uniformity (standard deviation). Additionally, when compared at different resolutions, they evaluate the down-sampling effects on the PC density. These features are computed, for each point in the PC, using the 5 nearest neighbor's average distance, at each resolution.
- **Number of Blocks** – Provides additional information regarding the PC size. This is the only extracted feature that is resolution-invariant, given how the block size changes proportionally with the resolution, i.e. when the resolution halves, so does the block size.

2. **Min-Max Normalization** – The previously described PC features have, naturally, very different dynamic ranges; if using them directly, such characteristic would destabilize the DL-based classifier's behavior. For this reason, a Min-Max Normalization is performed, thus ensuring that all features have a similar dynamic range. The normalization is obtained with:

$$x_{norm} = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{11}$$

where x are the input features' values, $\min(x)$ and $\max(x)$ the minimum and maximum values that each feature in x assume in the training dataset, respectively, and x_{norm} the normalized output values, ranging from 0 to 1. Since the normalization fitting, i.e. estimating the minimum and maximum of each feature, is only performed over the training dataset, it is possible to have x_{norm} values larger than 1 in the test dataset. Nevertheless, the dynamic ranges will be much closer after normalization, thus fulfilling its goal.

3. **DL-based Classifier** – Responsible for predicting the probability of each label corresponding to the possible parameter configurations; naturally, the recommended coding configuration corresponds to the label/configuration with the highest probability. To do so, a FC neural network is used, as shown in Figure 27b). Such network topology was adopted since it is capable of solving complex non-linear problems while being flexible to distinct types and volumes of input data.

   The classifier network's architecture, see Figure 27b, can be summarized as follows:

   - **Input Layer** – Prepares the normalized input PC features and Target Rate to be processed by the classifier's network. The number of units in the input layer is 11 to match the number of PC features extracted – three resolution-variant features for each of the three resolutions and a single resolution-invariant feature – plus the Target Rate.
   - **Output FC Layer** – Restrains the network to the number of labels needed to solve the classification problem. In this case, given the three sampling factors and six coding models available in PCGC-ASR, there are 18 unique pairs. However, this number was reduced to 15 after exhaustive testing of the codec has shown that the remaining three labels were never used (more information in Table 6). Additionally, the softmax activation function is used to convert the numerical features obtained with the last layer to output probabilities.
   - **Hidden FC Layers** – Hold the majority of the network's computational complexity and are responsible for processing the inputs, i.e. the PC statistical features and Target Rate. Unlike the input and output layers, the hidden layers have no design restriction, notably on the number of hidden layers, units per layer, and activation function. Given the high number of designs possible by playing with such parameters, the architecture design was carried out with the assistance of an exhaustive training mechanism, called *KerasTuner* [47], which allows to significantly speed up the network's design process. The exhaustive training process randomly selected and trained 1500 network configurations by changing the above mentioned hyperparameters. For more information on this training process, see Section 6.3.

   As briefly mentioned in Section 6, the 1500 trained models will go through a performance assessment process in which three model will be selected. Later on, these three models will go through a more rigorous performance assessments to ensure the selection of the optimal DL-based classifier model for the proposed ARC mechanism. In this case, the architecture of the best classifier model, containing a total of 2 697 999 parameters distributed for 8 FC layers, is presented in Figure 27b.

4. **Label Selection** – Responsible for picking the highest predicted probability, i.e. performing a top-1 probability binarization, thus selecting the label corresponding to the optimal coding parameter configuration pair. However, given the rate control nature of the classification task at hand, it is necessary to translate the predicted label into the pair of sampling factor and coding model values that should be provided to the PCGC-ASR codec; this conversion is summarized in Table 6. Note that after probability binarization, each label is represented by a 15-bit word with a single non-zero bit; this is the so-called *one-hot representation*. Simultaneously, and simplifying the conversion process, each label can also be represented by the index of the non-zero bit (i_{nz}) in the one-hot representation word, where $i_{nz} = 0$ and $i_{nz} = 14$ represent the least and most significant bit of the 15-bit word, respectively.

Table 6: Correspondence between the classifier label's index (i_{nz}) and the coding parameters pairs.

i_{nz}	Sampling Factor	Coding Model (λ value)
[0, 5]	1	[0.00025,0.0005,0.001,0.0025,0.005,0.01]
[6, 11]	2	[0.00025,0.0005,0.001,0.0025,0.005,0.01]
[12, 14]	4	[0.001,0.0025,0.005]

For easier understanding, consider the following example of probabilities predicted by the DL-based classifier:

$$\{0.01, 0.01, 0.01, 0.02, 0.06, 0.01, 0.05, 0.75\ , 0.02, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01\}$$

After the top-1 probability binarization is performed, the set of probabilities above becomes:

$$\{0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0\}$$

which is a one-hot representation word that corresponds to a unique parameter configuration pair. Given the position of its non-zero bit, $i_{nz} = 7$, and using Table 6, it is possible to conclude that the automatically selected coding parameters were: a sampling factor of 2 and the DL coding model for $\lambda = 0.005$. These parameters can now be provided to PCGC-ASR to code the original PC exactly as previously described.

6.3 DL-based Classifier Model Training Process

Given the DL-based approach for the ARC classifier, it is indispensable to clearly define its training process. The main goal of this training process was for the DL-based classifier to learn the relation between the PC features and the PC coding performance to determine the best PCGC-ASR parameter configuration for the specified PC and Target Rate.

6.3.1 Training Conditions

All training conditions established in the JPEG CTTC [46] were followed. Most importantly, all the defined Target Rates were considered with the exception of the lowest one since the PC quality is poor and thus not that relevant for practical use. Therefore, the Target Rates of 1.5, 0.5, and 0.15 bpp

were considered.

6.3.2 Training Dataset

As training data, the DL-based classifier requires a set of input PC features, described in Section 6.2, alongside the Target Rate, and the corresponding optimal parameter configuration class/label to be used as ground truth for training.

Training PCs

A set of 130 PCs were selected from the ShapeNet dataset [25], containing thousands of sampled meshes obtained in the conditions defined by Lazzarotto *et al.* in [48]. Due to the used mesh creation and PC sampling processes, these PCs are rather similar in density and uniformity. This lack of diversity is an undesirable characteristic if the proposed rate control mechanism is to be used for rather diverse conditions; hence a number of PC down-sampling operations was performed to create more sparsity and heterogeneity among the training PCs. For each PC, a set down-sampling factor was randomly selected from 1, 2, 4, and 8, allowing to obtain PCs with between 100% and 12.5% of the original number of points. This operation makes PCGC-ASR use a vaster set of coding parameter configurations, namely all the 15 coding combinations mentioned in Section 6.2., to meet the required Target Rates for all PCs.

These PCs were used for statistical feature extraction, following the procedure described in Section 6.2. Moreover, they were also used for the acquisition of the optimal labels as described below.

Optimal Label as Ground Truth

The optimal coding configuration classes/labels for the PCs above set the ground truth for the training process, i.e. provide a reference for the DL-based classifier predicted labels.

To obtain the optimal labels, each training PC needs to be coded 15 times, once for each possible parameter configuration, thus creating 15 RD trade-offs/points and labels. The optimal label is the one corresponding to the coding parameter pair that can be provided to the codec to obtain the RD point that maximizes the PC geometry quality metric, in this case PSNR D1, while being below the Target Rate. In reality, a 10% error margin was considered, thus allowing for the Optimal Rate to be at most 110% of the Target Rate. For a better understanding, see Figure 28 where a couple of examples are provided. Considering the Target Rate marked in grey, the optimal RD point is the one to the left of the red dashed line (Target Rate with the 10% upper margin) that maximizes the PSNR D1 metric; naturally, another quality metric may be selected. On the left-side example, it can be seen that the selected optimal RD point (marked with the red cross) has a larger absolute error, in rate, than the RD point slightly to the right of the red dashed line, when using the Target Rate as reference. Nevertheless, the one marked with the cross is considered the Optimal Rate as it is preferable to have a service with possibly slightly lower quality over service interruption, as it may happen in a real-life transmission scenario when the Target Rate plus margin is surpassed. On the right-side example, the blue dot marked with a red cross is slightly to the left of the red dashed line hence still being considered optimal, since within the 10% upper margin.

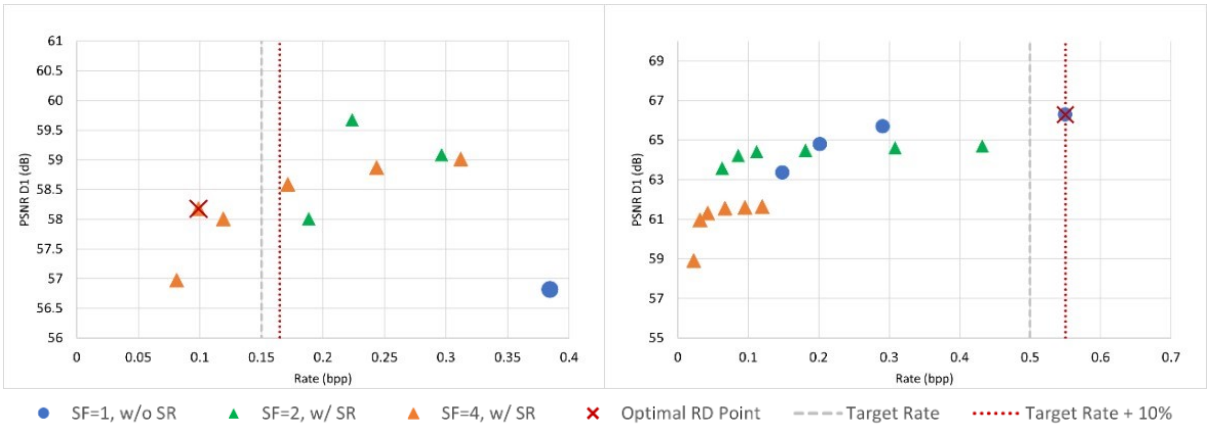


Figure 28: Examples for the selection of the optimal RD points (the red crosses) for a given Target Rate (vertical grey line) and its 10% upper margin (vertical red line).

Once determined the optimal RD point, the coding parameters (sampling factor and DL coding model pair) that lead to such RD trade-off are converted into the optimal label using Table 6. This optimal label will be the one included in the training dataset and used in the loss function.

In summary, each data sample consists of 11 classifier's inputs, i.e. the Target Rate and 10 PC statistical features, and one classifier's output, the optimal parameter configuration. In total, 390 data samples were considered for training, obtained with the 130 training PCs, each coded with three Target Rates, as previously mentioned.

6.3.3 Loss Function

The problem at hand is a multi-class classification problem, in this case selecting the optimal among the 15 possible coding parameter configuration labels. For this reason, the loss function adopted was the Categorical Cross Entropy (CCE) which is defined as follows:

$$L_{CCE} = -\sum_{i} y_i \cdot \log \hat{y}_i \tag{12}$$

where y_i is the one-hot word of the optimal label, and $\hat{y}_i$ the predicted probabilities' array for a given data sample i. This loss is inversely proportional to the estimated probability for the optimal label, thus decreasing when $\hat{y}_i$ gets closer to y_i . Keep in mind that the goal is to predict a label that corresponds to the ground truth, optimal label associated with the coding configuration for the Optimal Rate.

6.3.4 Training Parameters and Hyperparameters

For the networks' training, the Adam Optimizer [19] was used with a learning rate of 5×10^{-5} and a batch size of 8 data samples. As usual, to avoid overfitting to the training data, an early stopping algorithm with patience of 100 epochs was used, hence interrupting the training process if the loss on the validation dataset does not decrease for 100 consecutive epochs. More on the validation process in the next sub-section.

Due to the use of the massive training process as design method, the output of this stage is a set of 1500 trained models with different architectures, namely distinct number of hidden layers, units per

hidden layer and activation functions. Naturally, these models have to be assessed to select the final adopted model.

6.4 DL-based Classifier Model Validation Process

The validation process has two main purposes: overfitting prevention, as described in Section 6.3.4, and ranking the 1500 DL-based classifier models based on their preliminary performance.

6.4.1 Validation Conditions

The ranking process used the same three Target Rates as for training, i.e. 1.5, 0.5, and 0.15 bpp.

6.4.2 Validation Dataset

The validation dataset is formed by 10 new PCs selected from the ShapeNet mesh dataset [25]. These PCs were pre-processed and used for PC feature extraction and optimal label selection as described in Section 6.3.1, thus creating a dataset with 30 data samples. The validation dataset shares the layout with the training dataset, hence each data sample has 11 classifier's inputs and one output.

6.4.3 Assessment Metric

Since this is only a first (and not the final) selection, the performance assessment metric adopted to rank the 1500 trained classifier models relies on a single evaluation metric, the overall prediction accuracy that can be computed as:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions} \tag{13}$$

This is a popular used metric in many classification tasks as it provides a good expression of the classifier model capabilities, especially when considering a traditional classification problem, e.g. image classification, where all that matters is if the DL-based classifier model predicts the labels correctly. In this case, given the coding environment in which the proposed ARC is considered, this metric is less important but still relevant for a first reduction of the number of classifier models, i.e. from 1500 to 3.

6.4.4 Top-3 Ranked DL-based Classifier Models

After computing the validation accuracy for each of the 1500 trained classifier models, they are organized in descending accuracy order. The first 3 classifier models, i.e. the three models offering the best accuracy performance, are kept whereas all others are discarded; these models are referred from now on as candidate models.

The architecture of these three candidate models differs on the number of hidden layers and units per hidden layer while sharing the same activation function between layers (ReLU), as shown in Table 7.

Table 7: DL-based classifier candidate models main characteristics.

Classifier Model Name	Number of Hidden Layers	Units per Hidden Layer	Total Number of Parameters
Candidate Model 1	8	1024,256,256,256,265,1024,1024,256	2 051 087
Candidate Model 2	7	256,1024,1024,512,1024,256,256	2 697 999
Candidate Model 3	8	256,512,1024,1024,1024,512,1024,512	4 341 775

The three candidate models will be those considered for the model selection process described in the next sub-section. Note that the candidate models are still ordered by their accuracy on the validation dataset, i.e. Candidate Models 1 and 3 were the most and third most accurate, respectively.

6.5 DL-based Classifier Model Selection Process

This sub-section describes the DL-based classifier model selection process adopted to choose the final model between the three candidates identified during the validation process.

6.5.1 Selection Conditions

For the classifier models' final selection, the same conditions used during training were adopted, namely the three Target Rates, i.e. 1.5, 0.5, and 0.15 bpp.

6.5.2 Selection Dataset

For the model selection dataset, 10 new PCs from the same ShapeNet mesh dataset [25] were selected. As described in Section 6.3.1, these PCs were pre-processed and used for PC feature extraction and optimal label selection, thus creating a dataset with 30 data samples. Note that each data sample in the model selection dataset shares the layout used in training, i.e. 11 classifier's inputs and the optimal label to be predicted.

6.5.3 Assessment Metrics

In a traditional classification problem, e.g. image classification, the final goal is to predict the correct labels; thus, when assessing its performance, the predicted label can be either right or wrong. However, when considering the coding environment in which the ARC mechanism is integrated, that is not enough since it becomes fundamental to understand by how much the classifier misses the optimal label, e.g. in terms of actual rate, and what is the impact of such mispredictions in the final RD performance. Naturally, there is no single assessment metric capable of providing such information all at once, hence the following metrics have been selected for a complete ARC performance assessment:

- **Accuracy (Acc)**: provides the ratio between the number of correctly predicted labels and the total number of predictions, i.e. the percentage of predicted labels that are the same as the optimal ones (computed as in (13)).

- **Delta Target and Delta Optimal Rates (ΔTR, ΔOR)**: Measure by how much, on average, the selected RD point missed the Target and Optimal Rates, in percentage, respectively; these metrics are computed for each data sample as follows:

$$\Delta X_{Rate}(\%) = \frac{|\, X_{Rate} - Actual_{Rate} \,|}{X_{Rate}} \times 100, \tag{14}$$

where X_{Rate} can be either the Target or Optimal Rate (in bpp), depending on the desired metric. Naturally, their average value can be considered for dataset-wide assessment, as performed on the next sub-section.

- **Target Rate Infringement (TRI)**: Measures by how much the Actual Rate surpasses the Target Rate (within 10% upper margin), in percentage, due to label misprediction; this metric is computed for each data samples as follows:

$$TRI(\%) = \frac{Target_{\,Rate} - Actual_{Rate}}{Target_{Rate}} \times 100,\ Actual_{Rate} > 1.1 \times Target_{\,Rate} \tag{15}$$

Naturally, this metric's average value can be considered for dataset-wide assessment, as performed on the next sub-section. This is a relevant metric since in a real-life scenario where transmission restraints are imposed, using a rate larger than the Target Rate may have rather negative consequences, e.g. service interruption or buffering.

- **Bjontegaard-Delta Rate and PSNR (BD-Rate, BD-PSNR):** Measure the rate gain/loss, in percentage, for the same quality (BD-Rate) and quality gain/loss, in dB, for the same rate (BD-PSNR) between two RD curves obtained with the codec using the ARC predicted coding parameter configuration and the optimal one. Note that the average BD-Rate/PSNR metrics are estimated considering the positive and negative values, thus being possible to obtain a near-zero average based on positive and negative values balancing out.

By combining these evaluation metrics, it is possible to have a more solid and meaningful ARC performance assessment.

6.5.4 Selecting the Final DL-based Classifier

The candidate models chosen in Section 6.4 must go through a more detailed performance assessment to help determine which classifier model should be used in the proposed ARC solution. The model selection performance assessment results are shown in Table 8.

Table 8: Candidate models' performance; for each column/metric, the bold demarks the best value.

Classifier Model	Acc (%)	ΔTR (%)	ΔOR (%)	TRI (%)	BD-Rate (%)	BD-PSNR (dB)
Candidate Model 1	66.67	**21.490**	17.927	12.355	2.80067	-0.03064
Candidate Model 2	**70.00**	21.633	**14.793**	**10.636**	**0.98921**	**-0.02508**
Candidate Model 3	63.33	22.981	17.496	19.687	5.73548	-0.03224

The results in Table 8 can be summarized as follows:

- **Importance of Model Selection –** As previously mentioned, Candidate Model 1 was the most

accurate for the validation dataset. However, since this dataset was also used to prevent overfitting, thus conditioning indirectly the training process, the validation assessment may not be totally unbiased. When observing the results in Table 8, Candidate Model 1 is no longer the best performing, hence showing the importance of performing the model selection on a new dataset. Furthermore, the use of multiple assessment metrics allows for a more careful comparison between classifier models. This was more difficult to do while performing the model ranking with the validation dataset due to the large number of classifier models (1500).

- **Best Classifier Candidate Model –** The performance results show that Candidate Model 2 is the best performing for multiple assessment metrics. The higher overall accuracy is a good indicator as correctly predicting more labels should also lead to a benefit on the other metrics, e.g. it holds the lower Target Rate infringement which is naturally desirable. Additionally, the BD metrics also point in the same direction, showing how little impact the mispredicted labels have on the final RD performance.

In summary, Candidate Model 2 was selected as the final classifier model to be integrated in the proposed ARC solution; for this reason, its detailed architecture is shown in Figure 27b.

6.6 DL-based Classifier Model Performance Assessment

This sub-section describes the performance assessment process and its results for the final ARC classifier model, notably using a more challenging and externally defined PC dataset.

6.6.1 Test Conditions

The test conditions defined for the final performance assessment must be representative of real-life conditions, namely the Target Rates. For that reason, a larger set of Target Rates were considered during this performance assessment, notably considering the three Target Rates used for training, alongside three new ones, defined within the training rate range of the DL-based classifier model, namely 1.0, 0.6, and 0.25 bpp. By doing so, the results obtained will show how well the DL-based classifier model generalizes to unseen rate conditions.

6.6.2 Test Dataset

For the final test dataset, the 25 test PCs used during the assessment of the proposals submitted as a response to the JPEG Call for Proposals on Learning-based Point Cloud Coding were adopted, thus forming a large and diverse dataset. As described in Section 6.3.1, these PCs were pre-processed and used for PC feature extraction and optimal label selection. Considering the six test Target Rates previously mentioned, 150 test data samples were generated, each holding 11 classifier's inputs, i.e. Target Rate plus 10 PC features, and the ground truth/optimal label.

6.6.3 Assessment Metrics

The performance assessment metrics considered in the final performance assessment were the same used for the DL-based classifier model selection, see Section 6.5.3.

6.6.4 Performance Assessment

The final performance assessment results are shown in Table 9 for the two sets of Target Rates described in Section 6.6.1.

Table 9: Performance of the proposed ARC mechanism on the test dataset.

Target Rates (bpp)	Acc (%)	ΔTR (%)	ΔOR (%)	TRI (%)	BD-Rate (%)	BD-PSNR (dB)
1.5, 0.5, 0.15	66.67	26.116	12.284	20.115	4.32651	-0.13477
1.0, 0.6, 0.25	65.33	18.809	13.975	5.813	-0.38020	-0.00673

The results in Table 9 allow deriving the following conclusions:

- **Performance for the training Target Rates (in orange) –** As expected, there is a reduction in the overall performance when compared to the performance results obtained with the model selection dataset. This is due to the usage of the new dataset, which is larger and includes more heterogeneous PCs. Nevertheless, the performance drop is small, and the model generalizes fairly well. While the overall accuracy achieved may be considered low when compared to the typical 90% desired in many classification tasks, the BD metrics show that the approximately 30% mispredicted labels have a small impact on the final RD performance, which is desirable for the proposed rate control mechanism.
- **Performance for the unseen Target Rates (in blue) –** When assessing the performance for the new and unseen Target Rates, the results show a steady performance, rather similar to the performance obtained with the training Target Rates. This comes to prove that the classifier model generalizes well regardless of the Target Rates being used during training or not. Additionally, the BD metrics results highlight the volatility of a non-absolute average, as near-zero values are obtained by averaging several positive and negative BD-Rate/PSNR that balance out each other.

Regarding complexity, notably number of coding (and decoding) operations, the ARC mechanism allows to reduce the number of coding operations in 93.3% as coding only needs to be performed once with the automatically selected parameter configuration instead of with all the 15 possible parameter configurations, as performed in the brute-force optimization approach considered in Section 5.5.3. Naturally, this has a large implication in terms of running time, as the 5000 seconds required to perform coding optimization with the brute force approach, hence coding and decoding the PC 15 times, are reduced to 300 seconds, the average running time of encoding and decoding a PC once. These average running times were obtained in the same conditions, namely hardware, described by Guarda *et al.* in [42].

In summary, the proposed rate control mechanism, ARC, allows obtaining similar RD compression performance as with the brute-force optimization, eventually with a rather RD performance small penalty, while using only 6.7% of the computational complexity, hence offering a remarkable trade-off.

6.7 Ablation Studies

This section aims at studying the importance of the PC statistical features in the proposed automatic rate control mechanism. To do so, an ablation study was conducted by analyzing the performance impact of removing specific statistical features from the DL-based classifier's input. The following ablation studies were considered:

- **Single Resolution** – Removing all information concerning resolutions lower than the original, i.e. all statistical information obtained from the down-sampled input PCs by a factor of 2 and 4. Consequently, only 4 out of the 10 input PC features were considered, notably number of points and blocks, average, and standard deviation of the original PC point distance.
- **No Density Information** – Removing all information with respect to PC density and uniformity, i.e. the average and standard deviation of the point distances. Thus, only 4 input features remained, notably: number of blocks, and the number of points for the three resolutions.
- **No Number of Points** – Removing the number of PC points at each resolution, thus providing less information regarding the overall PC size and PC's behavior to down-sampling; the classifier considers now 7 out of the 10 input PC features.
- **No Number of Blocks** – Removing the number of blocks, thus reducing the information available regarding PC size. The other 9 statistical features are provided to the classifier.

For these ablation studies, the architecture of the selected model, shown in Figure 27, was slightly modified to adjust the input layer of the DL-based classifier to the new number of inputs, and retrained from scratch following exactly the procedure described in Section 6.3. Since the architecture was fixed, no massive training was necessary; however, given the non-deterministic behavior of the training process, each model was trained 3 times, producing three ablation models for each condition above. Just as before, a model selection process was conducted, following the terms described in Section 6.5, and the best classifier model was selected, naturally followed by a performance assessment under the conditions defined in Section 6.6. The summary of the results obtained is shown in Table 10.

Table 10: Ablation models' performance; the bold signals the best value for each column/metric.

Ablation Models	Classifier Inputs	Acc (%)	ΔTR (%)	ΔOR (%)	TRI (%)	BD-Rate (%)	BD-PSNR (dB)
Proposed ARC Model	11	**66.67**	**26.116**	**12.284**	**20.115**	4.32651	-0.13477
Single Resolution	5	34.67	37.365	35.292	24.901	2.43324	-0.11707
No Density Information	5	44.00	35.235	31.246	26.768	2.18857	-0.10192
No Number of Points	8	64.00	36.255	22.919	47.375	**1.85013**	-0.07873
No Number of Blocks	10	**66.67**	33.942	22.102	38.881	1.98884	**-0.06007**

The results in Table 10 allow deriving the following conclusions:

- **Single Resolution** - The model trained for a single and original resolution failed to predict all the labels that used a sampling factor larger than one (hence achieving approximately 30% accuracy), since no information about other resolutions was provided. This comes to prove the relevance of multi-resolution features in the proposed rate control mechanism.
- **No Density Information** – As for the single resolution, the model trained without the density-related features shows a very poor performance, regardless of the metric being considered, thus proving that these are indispensable features.
- **No Number of Points and Blocks** – The impact of these features on the final solution is more discrete yet noticeable. When compared to the reference, i.e. the proposed ARC model, the impact on accuracy is small if any. However, it is in the delta rate and Target Rate infringement metrics that the importance of these two features is seen, namely in the latter where a reduction of about 20% is achieved when using them. It is important to remember how surpassing the Target Rate can have profound negative consequences, thus reducing the amount of rate by which the target is surpassed is desirable.

In summary, the statistical features carefully hand-picked to assist the DL-based classifier in the proposed rate control solution play a vital role in providing indispensable information about the PC being coded, namely multi-resolution PC size, density, and uniformity.

6.8 Extending the DL-based Classifier Model with Coding Rate Features

The proposed rate control mechanism has room for improvement, namely when a lower Target Rate is required, since most of the classification errors happen for the lower rates. One possible explanation could be the absence of information that the DL-based classifier has regarding the behavior of the PCGC-ASR while coding each specific PC as they may be very different, namely in sparsity. With this in mind, this section studies the possibility to extending the number of input features used by the DL-based classifier, notably with two additional features informing on the minimum and maximum coding rates achieved while coding the specific input PC with the PCGC-ASR codec. The idea is to provide to the classifier with information regarding the codec's achievable rate range for that given PC, since different PCs achieve very different maximum and minimum rates due to their statistical differences. Intuitively, the addition of these coding rates may provide the DL-based classifier with information regarding the 'relation' between each specific PC and the PCGC-ASR codec that is not available in the previous set of PC features.

To obtain the minimum and maximum coding rates, the PC has to be coded twice using the configurations that are known to produce these rates, notably using a unitary sampling factor and the lowest lambda coding model for the maximum rate, and a sampling factor of 4 and the highest lambda coding model for the minimum rate. After these two preliminary coding and decoding operations, the minimum and maximum rates are provided to the DL-based classifier alongside with the remaining 10 PC statistical features, defined in Section 6.2, and the Target Rate.

The addition of these two input features to the FC neural network/model demands additional

complexity to help processing them. Thus, a new and more complex DL-based classifier was selected from a new set of automatically trained networks. The training and model selection processes were performed as described in Sections 6.3 to 6.5.

Note, that there is a considerable difference in the extended solution regarding the normalization process. In Section 6.2 this normalization is performed independently to each and every input feature, thus ensuring that they all achieve the 0 to 1 range. However, if that same operation was performed over the minimum, maximum and target coding rates, they would fall out of proportion, hence losing meaningful information, as they are expected to have different minimum and maximum values within the same dataset. Consequently, the simplest solution is to not consider these three input coding rates (minimum, maximum and target) in the normalization process at all. This is only acceptable as the unnormalized coding rates assume small values near one (in bpp) like the normalized features, thus not disturbing the network stability.

The final performance results obtained with the selected extended DL-based classifier model are presented in Table *11* for the training and new Target Rates. Additionally, to ensure that no harm is caused by the non-normalized rates, the results for a fully normalized model, i.e. with the coding rates normalized, are also shown.

Table 11: Performance for the selected extended models for two sets of Target Rates, marked with distinct colors. The orange/blue bold signals the best result for each metric and Target Rates set.

Classifier Model Name	Target Rates (bpp)	Acc (%)	ΔTR (%)	ΔOR (%)	TRI (%)	BD-Rate (%)	BD-PSNR (dB)
Proposed ARC Model (No Rates)	1.5, 0.5, 0.15	66.67	26.116	12.284	20.115	4.32651	-0.13477
	1.0, 0.6, 0.25	65.33	18.809	13.975	5.813	-0.38020	-0.00673
Non-Normalized Rates Extended Model	1.5, 0.5, 0.15	**72.00**	24.972	**10.854**	**15.587**	**1.64247**	**-0.03868**
	1.0, 0.6, 0.25	**68.00**	16.862	**11.131**	**4.278**	-1.12663	**0.01083**
Normalized Rates Extended Model	1.5, 0.5, 0.15	65.33	**23.799**	13.021	19.383	6.23727	-0.20232
	1.0, 0.6, 0.25	66.67	**15.860**	14.289	11.613	**-2.22247**	-0.07910

The results in Table 11 can be summarized as follows:

- **No Coding Rates vs Non-Normalized Coding Rates** – The addition of the non-normalized minimum and maximum coding rates as input features to the DL-based classifier provided a noticeable performance improvement. The increment in accuracy performance also translates into an improvement in delta rates and lower Target Rate infringement, both very desirable. Furthermore, the BD metrics show a small improvement as well. The outcome is the same when testing for the unseen Target Rates (in blue), thus showing, once again, how well the classifier

model generalizes to the new Target Rates.

- **Non-Normalized vs Normalized Coding Rates** – The rate normalization process annuls the gains produced by the addition of the minimum and maximum coding rates. This is expected, as previously explained, since the proportion of the rates is not kept, thus essentially rendering meaningless the potential information provided by these additional coding rates.

The results in Table 11 are interesting as they show some performance improvement over the reference classifier model. However, it is still necessary to check if the improvement comes from the lower rates where the ARC mechanism struggles the most. Having that in mind, an accuracy breakdown by Target Rate is shown in Table 12.

Table 12: Accuracy breakdown by Target Rate. The bold signals the best value for each Target Rate.

Classifier Model Name	**Per Rate Accuracy (%)**			
Target Rates (bpp)	1.5	0.5	0.15	Average
Proposed ARC (No Rates)	**92.0**	**84.0**	24.0	66.67
Non-Normalized Rates Extended	**92.0**	80.0	**44.0**	**72.00**
Normalized Rates Extended	**92.0**	80.0	24.0	65.33

The results in Table 12 show that the 6% average accuracy increase obtained with the non-normalized rates (over the reference, i.e. without the rates) comes entirely from the lower Target Rates, i.e. considering a Target Rate of 0.15 bpp, where a 20% accuracy improvement is visible. Once again, for the normalized rates extended model the gains fade out, thus showing the importance of keeping the three rates (minimum, maximum, and target) in proportion.

In summary, the results obtained with the performance assessment of the selected extended models show an improvement over the proposed ARC model. This improvement is proven to come from the lower rates in which the proposed model struggled the most. These gains were obtained with the non-normalized rates extended classifier model, hence showing the importance of keeping the minimum and maximum coding rates and the Target Rate in the same proportion, as expected.

Despite the interesting concept of including the achievable coding rate range for the classifier model to learn and use for a more accurate classification prediction, the heavy burden of having to code twice the PC before using the ARC mechanism cannot be ignored. In the complexity domain, the extended rates classifier model requires coding each PC three times, thus increasing the number of coding (and decoding) operations by 200% over the proposed ARC mechanism.

In summary, the improvement of 6% in the overall accuracy over the proposed ARC model, shown in Table 11 and Table 12, comes at a 200% increase in complexity, which does not seem to represent an advantageous trade-off. For that reason, the extended rates model was not considered in the final rate control mechanism.

7 Summary and Future Work

This chapter summarizes the main contributions of this thesis, namely focusing on the obtained achievements and derived conclusions. Furthermore, some ideas for possible future research work are also offered.

7.1 Summary

The main goal of this M.Sc. Thesis was to propose an advanced DL-based PC coding pipeline with integrated sampling tools for improved RD performance. Naturally, this objective was defined not only with the main fragilities of ADL-PCC in mind, as this was the starting point for the proposed solution, but also the promising results obtained by other DL-based coding solutions already using sampling-based approaches to improve the overall coding performance and flexibility.

The proposed static PC geometry coding solution, labeled PCGC-ASR, adopted some key changes over ADL-PCC, notably no longer considering multiple DL coding models in parallel in favor of an adaptive super-resolution approach for a more effective content-adaptive behavior. The DL coding model architecture was improved by using deeper and more powerful CNNs allowing to boost the compression performance.

Regarding the sampling tools, PCGC-ASR relies on a lossy basic grid down-sampling operation to reduce the PC block's resolution and sparsity prior to coding, as previous works have shown this approach to be beneficial for compression performance. Naturally, a decoder counterpart had to be considered to restore the original resolution, i.e. basic grid up-sampling, however, not enough to compensate the losses on the down-sampling operation. For that reason, a second post-processing tool has been proposed using a DL-based super-resolution approach. This second deep CNN model, so-called DL SR model, has to learn to extract meaningful PC features and use them to predict how densify the PC surfaces, thus helping to mitigate the down-sampling losses and their effect on the RD performance.

The proposed PCGC-ASR solution underwent a rigorous performance assessment that has clearly shown the benefits of integrating sampling tools in the coding pipeline, namely the DL SR model considered for sampling factors larger than one, which brought very significant quality gains at not rate cost, thus allowing to achieve more competitive performances when coding sparser PCs and trying to achieve lower coding rates. When compared to the benchmarking solutions, PCGC-ASR clearly outperformed G-PCC Octree and ADL-PCC for all the test PCs considered, achieving average rate savings of 81.4% and 53.2%, respectively, measured as BD-Rate for the PSNR D1 metric.

Furthermore, PCGC-ASR was also able to outperform V-PCC Intra for 22 out of the 25 JPEG test PCs, with the exceptions being PCs with rather specific characteristics that directly benefited from the V-PCC Intra coding solution projection-based approach. Nevertheless, an average rate saving of 34.2% was obtained, once again, measured as BD-Rate for the PSNR D1 metric. Note that while outperforming G-PCC and ADL-PCC was expected, doing it for V-PCC Intra is a considerable achievement as this solution relies of very powerful 2D video codecs that have been developed for many decades now.

Additionally, this coding solution, extended to perform a joint PC geometry and color coding, was submitted as response to the JPEG Pleno Call for Proposals on Learning-based Point Cloud Coding. After assessment by the JPEG experts, this solution was selected as Verification Model, thus becoming the starting point for the JPEG Pleno DL-based PC coding standard. Nevertheless, it is important noting that the proposed DL-based joint coding model is outperformed by the V-PCC Intra standard for the color-only and joint metrics, thus showing that DL-based PC color coding (joint or standalone) is still in early stages of development, with large room for improvement.

One of the main PCGC-ASR advantages is the coding flexibility at testing time offered by a set of coding parameters that have been defined to provide coarse-to-fine rate control. Naturally, this requires some optimization process to be conducted to determine what is the optimal parameters configuration to code a given PC for a specific target rate. An alternative to the brute-force approach corresponding to simply testing all parameters configurations has been proposed in the form of an automatic rate control (ARC) parameter selection mechanism. This tool is responsible for extracting a set of meaningful multiscale PC statistical features and use a DL-based classifier to predict the optimal parameter configuration for the target PCGC-ASR rate.

Naturally, the mispredictions of the DL-based classifier could have an adverse impact on the PCGC-ASR RD performance since the best parameter configuration would not be selected, it has been shown that the impact of the mispredictions on the RD performance is small. As a consequence, the proposed rate control mechanism offers an advantageous trade-off as it allows to reduce the optimization complexity by 93.3% when compared to the alternative brute-force approach.

In summary, the conducted studies have shown that the integration of sampling tools in the DL-based PC coding pipeline, such as the adaptive super-resolution considered in PCGC-ASR, improves significantly the RD coding performance, and allows to address a larger rate range while coding a larger variety of PCs, namely in density. Despite the promising results, there is still room for improvement as this is a fairly novel and embryonic research field, especially when compared to other multimedia areas such as image and video coding. The next sub-section provides some ideas for future next steps that may follow the work developed in this thesis.

7.2 Future Work

The proposed PC geometry coding solution relies on a volumetric-based PC representation approach that explicitly considers and codes both the empty and occupied voxels in 3D blocks. Considering that PCs have a lot of free space, this will inevitably limit the achievable computational efficiency. A

possible solution would be a sparse PC representation, that only stores information regarding the occupied voxels, alongside sparse convolutions, that would allow to reduce the required computational resources for coding but also improve its RD performance, namely due to the possibility of efficiently coding an entire PC at once, without block partitioning. Another interesting study regarding the DL coding and SR models would be the use of more complex network architectures, notably including transformers and attention mechanisms, that have shown promising results in varied multimedia domains like image and video coding, and more recently, PCs as well.

Regarding the sampling tools, some upgrades could also be considered. On one hand, the proposed DL SR model requires training multiple models for distinct sampling factor, hence not being able to cover a wide range of sampling factors. Consequently, the idea of having a single DL SR model for all sampling factors is an interesting one, e.g. using a meta-learning approach, i.e. use an auxiliar network to adapt the main DL SR model's behavior to a specific sampling factor without model retraining, as it would improve the overall flexibility of the coding solution. On the other hand, the option of integrating resampling tools has not yet been explored, in particular at the decoder side. The use of such sampling tools could be considered to adapt the coding solution to meet special requirements that may exist in some human- or machine-related applications, e.g. sharper edges for computer vision tasks.

As previously mentioned, the color results obtained for the extended DL-based joint geometry and color coding fall short of the expectations. Naturally, there are multiple possible directions of development, such as, improving the loss function considered during the training process or considering color dilation techniques.

Lastly, regarding the rate control mechanism, and the DL classifier in particular, some improvements could also be considered, like the extension of the rate control mechanism to consider all the available PCGC-ASR coding parameters, thus providing a finer control over the coding rate. Moreover, considering more complex network architectures could further mitigate the losses caused by the automatic rate control, hence improving its overall usability in real-life scenarios.

References

[1] F. Pereira et al., “Point Cloud Coding: A Privileged View Driven by a Classification Taxonomy,” Signal Process. Image Commun., vol. 85, p. 115862, Jul. 2020.

[2] “Final Call for Evidence on JPEG Pleno Point Cloud Coding.” Doc. ISO/IEC JTC 1/SC29/WG1 N88014, Online Meeting, Jul. 2020.

[3] A. F. R. Guarda et al., “Adaptive Deep Learning-Based Point Cloud Geometry Coding,” IEEE J. Sel. Top. Signal Process., vol. 15, no. 2, pp. 415–430, Feb. 2021.

[4] A. Voulodimos et al., “Deep Learning for Computer Vision: A Brief Review,” Comput Intell Neurosci., vol. 2018, Feb. 2018.

[5] J. Ballé et al., “Variational Image Compression with a Scale Hyperprior,” Int. Conf. Learning Representations, Vancouver, Canada, Apr. 2018.

[6] “Final Call for Proposals on JPEG Pleno Point Cloud Coding.” Doc. ISO/IEC JTC 1/SC29/WG1 N100097, Online Meeting, Jan. 2022.

[7] D. Graziosi et al., “An Overview of Ongoing Point Cloud Compression Standardization Activities: Video-based (V-PCC) and Geometry-based (G-PCC),” APSIPA Trans Signal Inf Process, vol. 9, pp. 1–17, Mar. 2020.

[8] E. Alexiou et al., “A Comprehensive Study of the Rate-Distortion Performance in MPEG Point Cloud Compression,” APSIPA Trans. Signal Inf. Process., vol. 8, pp. 1–27, Jul. 2019.

[9] Y. Zhou et al., “End-to-End Multi-View Fusion for 3D Object Detection in LiDAR Point Clouds,” Massachusetts Institute of Technology, MA, USA, May 2020, pp. 923–932.

[10] J. P. C. Prazeres, “Static Point Clouds Compression Efficiency of MPEG Point Clouds Coding Standards”, M.Sc. Thesis, Universidade da Beira Interior (UBI), Covilhã, Portugal, Jul. 2020.

[11] R. L. de Queiroz et al., “Compression of 3D Point Clouds Using a Region-Adaptive Hierarchical Transform,” IEEE Trans. Image Process., vol. 25, no. 1–8, pp. 3947–3956, Aug. 2016.

[12] S. Schwarz et al., “Emerging MPEG Standards for Point Cloud Compression,” IEEE J. Emerg. Sel. Top. Circuits Syst., vol. 9, no. 1, pp. 133–148, Mar. 2019.

[13] H. Liu et al., “A Comprehensive Study and Comparison of Core Technologies for MPEG 3-D Point Cloud Compression,” IEEE Trans. Broadcast., vol. 66, no. 3, pp. 701–717, Sep. 2020.

[14] S.Schwarz et al., “Common Test Conditions for Point Cloud Compression.” Doc. ISO/IEC JTC1/SC29/WG11 (MPEG), Doc. N18175, Marrakech, Morocco, Jan. 2019.

[15] E. S. Jang et al., “Video-Based Point-Cloud-Compression Standard in MPEG: From Evidence Collection to Committee Draft [Standards in a Nutshell],” IEEE Signal Process. Mag., vol. 36, no. 3, pp. 118–123, May 2019.

[16] D. Tang et al., “Deep Implicit Volume Compression,” IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Seattle, WA, USA, Jun. 2020.

[17] I. H. Witten et al., “Arithmetic Coding for Data Compression,” Commun. ACM, vol. 30, no. 6, pp. 520–540, Jun. 1987.

[18] S.Schwarz et al., “Common Test Conditions for Point Cloud Compression,” ISO/IEC JTC1/SC29/WG11 MPEG, Doc. N18175, Marrakech, Morocco, Jan. 2019.,

[19] D. P. Kingma et al., “Adam: a Method for Stochastic Optimization,” Int. Conf. Learning Representations, San Diego, CA, USA, May 2015.

[20] “Common Test Conditions for Point Cloud Compression.” ISO/IEC JTC1/SC29/WG11 MPEG, Doc. N19084, Brussels, Belgium, Jan. 2020.

[21] J. Wang et al., “Multiscale Point Cloud Geometry Compression,” Data Compression Conf. (DCC), Cliff Lodge, UT, USA, Mar. 2021.

[22] J. Wang et al., “Lossy Point Cloud Geometry Compression via End-to-End Learning,” IEEE Trans. Circuits Syst. Video Technol., pp. 1–15, Jan. 2021.

[23] C. Choy et al., “4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks,” 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, Jun. 2019.

[24] C. Szegedy et al., “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning,” Proceedings of the AAAI Conf. on AI, San Francisco, CA, USA, Feb. 2017.

[25] A. X. Chang et al., “Shapenet: An Information-Rich 3D Model Repository,” ArXiv Prepr. ArXiv151203012, Dec. 2015.

[26] M. Krivokuća et al., “8i Voxelized Surface Light Field (8iVSLF) Dataset.” Doc. ISO/IEC JTC1/SC29 WG11 (MPEG), Doc. m42914, Ljubljana, July 2018.

[27] R. de Queiroz et al., “Fractional Super-Resolution of Voxelized Point Clouds,” IEEE Trans. Image Process. Submitt., Jul. 2021.

[28] G. Qian et al., “PU-GCN: Point Cloud Upsampling using Graph Convolutional Networks,” IEEE/CVF Conf. on Computer Vision Pattern Recognition (CVPR), Nashville, TN, USA, Mar. 2021.

[29] Z. Chen et al., “Point Cloud Resampling using Centroidal Voronoi Tessellation Methods,” Comput.-Aided Des., vol. 102, pp. 12–21, Sep. 2018.

[30] A. Akhtar et al., “Point Cloud Geometry Prediction Across Spatial Scale using Deep Learning,” IEEE Int. Conf. on Vis. Commun. and Image Process. (VCIP), Hong Kong, China, Dec. 2020.

[31] S. Ye et al., “Meta-PU: An Arbitrary-Scale Upsampling Network for Point Cloud,” IEEE Trans Vis Comput Graph. TVCG Accept., Feb. 2021.

[32] S. Loffe et al., “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” 201 Int. Conference on Machine Learning (ICML), Lille, France, Mar. 2015.

[33] X. Hu et al., “Meta-SR: A Magnification-Arbitrary Network for Super-Resolution,” IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, Jun. 2019.

[34] A. L. Maas et al., “Rectifier Nonlinearities Improve Neural Network Acoustic Models,” 30th International Conference on Machine Learning (ICML), Atlanta, GA, USA, May 2013.

[35] C. R. Qi et al., “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space,” 31st Conference on Neural Information Processing Systems (NeurIPS), Long Beach, CA, USA, Jun. 2017.

[36] “VisionAir Digital Shape Repository.” http://visionair.ge.imati.cnr.it/ontologies/shapes/

[37] J. Feydy et al., “Interpolating between Optimal Transport and MMD using Sinkhorn Divergences,” 22nd International Conference on Artificial Intelligence and Statistics (AISTATS), Okinawa, Japan, Oct. 2018.

[38] R. Li et al., “PU-GAN: a Point Cloud Upsampling Adversarial Network,” Int. Conf. on Comput. Vision (ICCV), Seoul, South Korea, Jul. 2019.

[39] L. Yu et al., “PU-Net: Point Cloud Upsampling Network,” IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, Mar. 2018.

[40] H. Wu et al., “Point Cloud Super Resolution with Adversarial Residual Graph Networks,” 31st Br. Mach. Vis. Conf BMVC, vol. Manchester, UK, Aug. 2019.

[41] W. Yifan et al., “Patch-based Progressive 3D Point Set Upsampling,” IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, Mar. 2019.

[42] A. F. R. Guarda et al., “IT/IST/IPLeiria Response to the Call for Proposals on JPEG Pleno Point Cloud Coding.” arXiv, Aug. 2022.

[43] D. Minnen et al., “Joint Autoregressive and Hierarchical Priors for Learned Image Compression.” 32nd Conf. on Neural Info. Proc. Sys., Sep. 2018.

[44] A. F. R. Guarda et al., “Deep Learning-Based Point Cloud Geometry Coding: RD Control Through

Implicit and Explicit Quantization," 2020 IEEE International Conference on Multimedia Expo Workshops (ICMEW), London, United Kingdom, Jul. 2020.

[45] O. Ronneberger et al., "U-Net: Convolutional Networks for Biomedical Image Segmentation," Int. Conf. on Med. Image Comput. and Computer-Assisted Intervention (MICCAI), Munich, Germany, Oct. 2015.

[46] "Common Training and Test Conditions for Point Cloud Compression." Doc. ISO/IEC JTC 1/SC29/WG1 N100112, Online Meeting, Jan. 2022.

[47] O'Malley et al., "KerasTuner." 2019. [Online]. Available: https://github.com/keras-team/keras-tuner

[48] D. Lazzarotto et al., "Sampling color and geometry point clouds from ShapeNet dataset." arXiv, Jan. 18, 2022.