UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

POINT CLOUD DATA COMPRESSION

ASHWIN KUMAR GURURAJAN

Thesis supervisor: JOAN SERRA SAGRISTA (Universitat Autonoma de Barcelona)

Tutor: ALFREDO VELLIDO ALCACENA (Department of Computer Science)

Degree: Master Degree in Innovation and Research in Informatics (Advanced Computing)

Thesis report

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

28/06/2023

Abstract

The rapid growth in the popularity of Augmented Reality (AR), Virtual Reality (VR), and Mixed Reality (MR) experiences has resulted in an exponential surge of three-dimensional data. Point clouds have emerged as a commonly employed representation for capturing and visualizing three-dimensional data in these environments. Consequently, there has been a substantial research effort dedicated to developing efficient compression algorithms for point cloud data. This Master's thesis aims to investigate the current state-of-the-art lossless point cloud geometry compression techniques, explore some of these techniques in more detail and then propose improvements and/or extensions to enhance them and provide directions for future work on this topic.

Table of Contents

Cover 1

Abstract 2

Table of Contents 3

List of figures 5

List of tables 6

1 Introduction 7
- 1.1 Objectives . . . 8
- 1.2 Use Cases . . . 8
  - 1.2.1 Virtual, augmented and mixed reality . . . 8
  - 1.2.2 Automatic navigation systems . . . 8
  - 1.2.3 3D viewpoint . . . 9
  - 1.2.4 Remote sensing and geographical information systems . . . 10
  - 1.2.5 Design, Manufacturing, and Printing . . . 10
  - 1.2.6 Medical Imaging . . . 11
  - 1.2.7 Construction and Manufacturing . . . 11
  - 1.2.8 Consumer and Retail . . . 12
  - 1.2.9 Cultural heritage . . . 12
- 1.3 Background . . . 12
- 1.4 Structure of the Document . . . 14

2 State of the art 15
- 2.1 Regular Point Clouds . . . 15
- 2.2 LiDAR Point Clouds . . . 17
- 2.3 Deep Learning . . . 18

3 Detailed Examination of Advanced Techniques for PCC 20
- 3.1 MPEG . . . 20
  - 3.1.1 V-PCC . . . 20
  - 3.1.2 G-PCC . . . 21
- 3.2 SparsePCGC . . . 23
- 3.3 FRL . . . 25

4 FRL Revisited 29
- 4.1 Statistics . . . 29
- 4.2 Preprocessing . . . 33
- 4.3 Performance Improvements . . . 34
- 4.4 Extensions . . . 36
  - 4.4.1 Axis ordering . . . 36
  - 4.4.2 Supervoxels . . . 36
- 4.5 LiDAR Support . . . 37
  - 4.5.1 Context-adaptive binary arithmetic coding (CABAC) . . . 38

5 Experimental Results **43**
5.1 Datasets 43
5.2 Results 48
5.3 Analysis of results 51

6 Conclusion and future work **57**

References **59**

List of figures

1 Example of a point cloud . . . 7
2 Example of a point cloud with attributes . . . 7
3 Example of VR streaming use case . . . 9
4 Illustrations of large-scale 3D mapping data. . . . 9
5 Free viewpoint sport capture as a point cloud . . . 10
6 Example of the wide area scanning use case . . . 10
7 Example of design, manufacturing and 3D printing use case . . . 11
8 Cultural heritage in the 3D Cloud project . . . 12
9 Voxelization process [17] . . . 14
10 Occupied vs non-occupied voxels . . . 14
11 Simplified 2D view of an octree construction. . . . 15
12 Left: Recursive subdivision of a cube into octants. Right: The corresponding octree. . . . 16
13 Depiction of a point cloud along with different options to limit encoded voxels. 16
14 A taxonomy of Learning-based Point Cloud Compression. . . . 18
15 3D Patch projection and respective occupancy map, geometry, and attribute 2D images, (a) 3D patch, (b) 3D Patch Occupancy Map, (c) 3D Patch Geometry Image, (d) 3D Patch Texture Image. . . . 20
16 Example of patch packing. . . . 21
17 G-PCC reference encoder. . . . 22
18 Initial steps in octree construction. . . . 22
19 Trisoup point derivation. . . . 23
20 Various Representation Models of Voxelized PCG. . . . 23
21 SparsePCGC. . . . 24
22 FRL Compression Scheme. . . . 25
23 Example of runs in a slice. . . . 25
24 3D Context maps. . . . 26
25 Context dictionary. . . . 26
26 Rate Comparison in Bits per Occupied Voxel (bpov). . . . 27
27 Encoding speed vs bitrate. . . . 28
28 Point clouds in MVUB dataset. . . . 43
29 Point clouds in 8iVFB dataset. . . . 43
30 Point clouds in Owlii dataset. . . . 44
31 Point clouds in Private dataset. . . . 44
32 3D Modalities S3DIS Dataset . . . 45
33 Data Processing of S3DIS Dataset. . . . 45
34 Velodyne Dataset. . . . 46
35 Shapenet Dataset. . . . 46
36 Axis ordering. . . . 50
37 Speed Sorting. . . . 54

List of tables

1 Datasets 47
2 FRL C implementation time(s) for different parts 48
3 FRL C++ implementation time(s) for different parts 49
4 Bitrates (bpov) frl existing vs new 2d supervoxels 50
5 Bitrates (bpov) frl existing vs new 3d supervoxels 51
6 Sparse Point Cloud Bitrates with 3D supervoxels. 51
7 LiDAR Data Bit Rate (Bpov). 51
8 Statistics for a subset of data 52

1 Introduction

Point clouds have gained prominence as the preferred digital representation for three-dimensional objects, thanks to recent advancements in 3D acquisition technologies, which have favoured point clouds over polygonal meshes. A point cloud is essentially a huge collection of tiny individual points plotted in 3D space. The points may represent a 3D shape or object. In Figure 1 we present a visualisation of point cloud data.

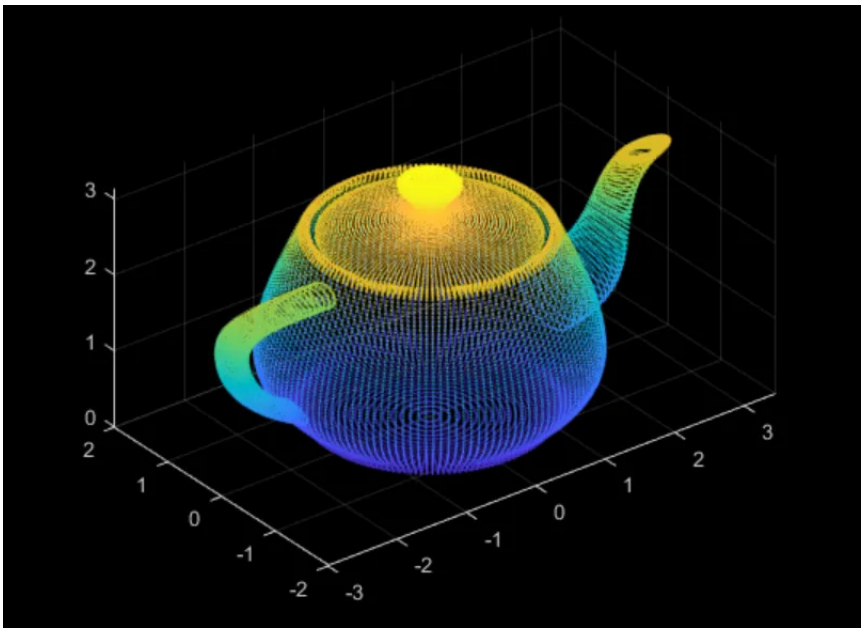


Figure 1. Example of a point cloud [1].

Each point position has its set of Cartesian coordinates (X, Y, Z) which indicates its location (geometry data). They normally also have some additional attribute data (for example colour, intensity, time, etc) [2]. Figure 2 is an example of a point cloud with 2 attributes which are its colour and luminance information.

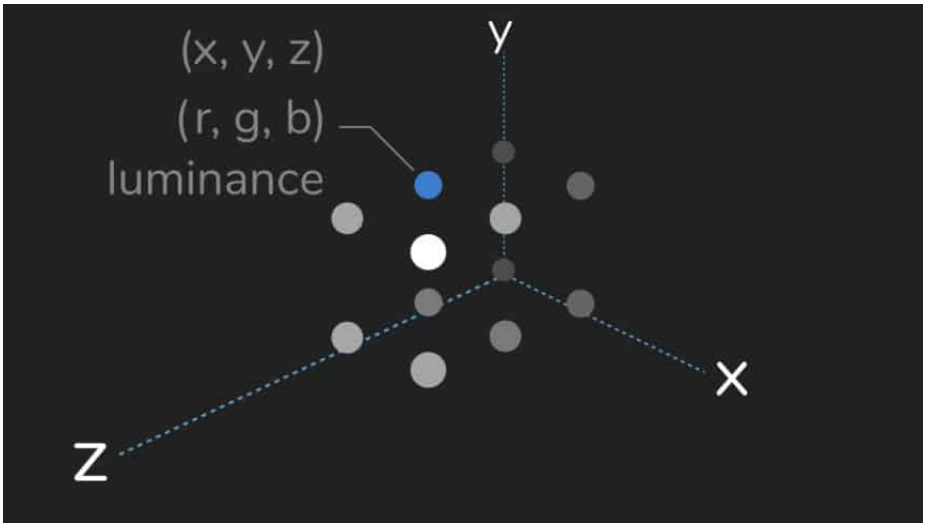


Figure 2. Example of a point cloud with attributes [3].

1.1 Objectives

This subsection gives an overview of the general and specific motivations and objectives for this thesis.

- Explore and better understand what point cloud data is and how it is useful.
- Understand the importance of compression specifically for point clouds
- Investigate the state-of-the-art techniques that enable lossless point cloud geometry compression.
- Select a few of the state-of-the-art techniques and explore them in more detail. Find the advantages and disadvantages for each of those techniques along with recommended usage scenarios.
- Propose improvements or enhancements to one of the techniques to improve its compression ratio or encoding speed.
- Provide guidance for future work on this topic.

1.2 Use Cases

In this subsection, we discuss the various use cases for point cloud data. Point clouds are very important as they are one of the foundational pillars in representing the 3D digital world despite irregular topology among discrete points. State-of-the-art sensors can capture point clouds from various platforms (satellite, aerial, UAV, vehicle-borne, backpack, handheld and static terrestrial), perspectives (nadir, oblique and side-view), spectrums (multispectral), and granularity (point density and completeness). This section presents a detailed overview of the various use cases [4] categorized based on specific industries, and in some cases explain the need for compression.

1.2.1 Virtual, augmented and mixed reality

Point clouds play a crucial role in supporting the visualization of 3D content within virtual, augmented, and mixed reality (VR/AR/MR) environments, as well as in computer graphics and gaming scenarios. These technologies find applications in government, commercial, and military sectors for modelling, training, situational awareness, and planning purposes. Figure 3 provides an illustrative workflow for this use case, showcasing the need to render the point cloud in real time with varying viewpoints, scales, and resolutions during gaming or VR experiences.

1.2.2 Automatic navigation systems

The development of autonomous vehicles heavily relies on point cloud data. Point clouds play a vital role in capturing the 3D shape and intricate details of the surrounding environment for autonomous vehicles.

Devices equipped with localization capabilities, depth sensors, and colour measurement capabilities enable the creation of large-scale 3D maps for both indoor and outdoor environments. Localization can be achieved through GPS, inertial measurement units (IMU), cameras, or a combination of these devices. Depth measurements can be obtained using time-of-flight, radar, or laser scanning systems. Commercially available mapping systems, such as the high-end mobile mapping system depicted, exemplify the various forms in which these systems exist. Additionally, lightweight mobile platforms mounted on drones or mobile phones/tablets are being developed.

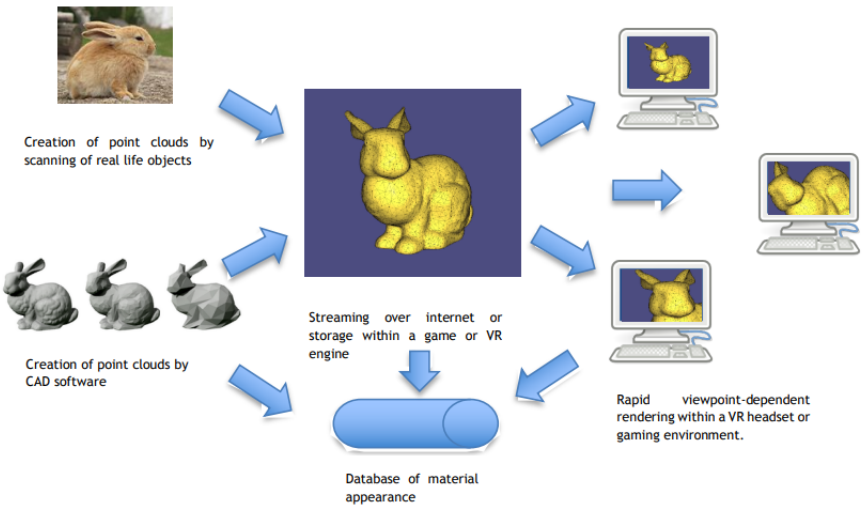


Figure 3. Example of VR streaming use case [4].

Regardless of the specific platform used for measurement acquisition, a 3D map can be generated by combining depth measurements,featuring a high-density laser-scanned point cloud, with camera images. This fusion of point cloud data and camera images to produce a 3D map. These maps can be enriched with road markings, lane information, and road signs, enabling autonomous vehicle navigation in urban environments.

The objective of this use case is to analyze the point cloud data to identify objects of interest, such as pedestrians and obstacles, without requiring human intervention. Self-driving cars of the future utilize sensors (LIDAR) to collect point cloud information. This data is essential for refining machine learning and deep learning algorithms, thereby advancing ADAS (Advanced Driver Assistance Systems) technology and ultimately leading to full automation. Figure 4 shows 3D mapping point cloud data.

Figure 4. Illustrations of large-scale 3D mapping data [5].

1.2.3 3D viewpoint

To enable free viewpoint playback and interaction with sports events, such as basketball or baseball, on mobile devices and TV, it is crucial to have capabilities of compression and a file format standard that allows for the streaming and storage of point cloud data in an interoperable manner. In addition to point cloud data, the full scene may also incorporate textures and/or video elements. Figure 5 shows viewpoint based sports capture as a point cloud.

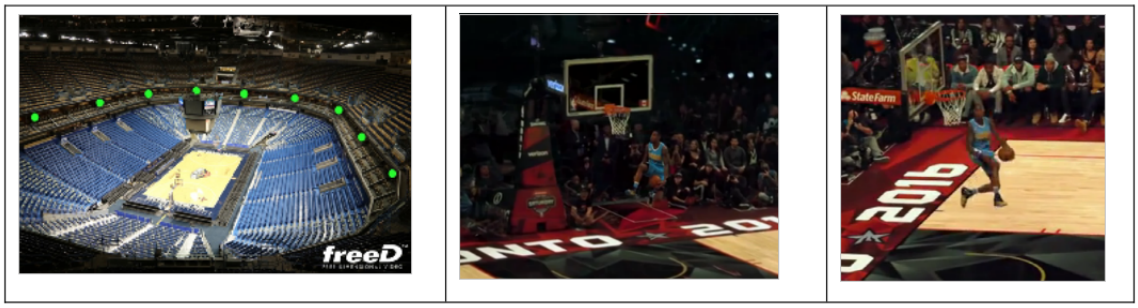


Figure 5. Free viewpoint sport capture as a point cloud [5].

1.2.4 Remote sensing and geographical information systems

Point clouds are utilized to capture the 3D shape and intricate details of objects that exceed 20 meters in size, i.e. wide area scanning. This application involves various imaging modalities like LiDAR (Light Detection and Ranging), airborne SAR (Synthetic Aperture Radar), radar, or multi-view stereo to collect data. Figure 6 exemplifies a point cloud that is characteristic of this use case. The data exhibits high resolution but does not encode the material appearance of the captured scene. However, the massive size of these point clouds poses challenges for efficient handling within existing geographic information and communication technology (geo-ICT) infrastructures. To address this issue, point cloud compression techniques can play a vital role in enhancing point cloud data management systems (PCDMS).

Figure 6. Example of the wide area scanning use case [4].

1.2.5 Design, Manufacturing, and Printing

Point clouds are extensively used in the design, manufacturing, and 3D printing industries to facilitate the production of 3D content. These point clouds are artificially created and serve as a crucial foundation for manufacturing processes, whether it involves 3D printing or traditional manufacturing methods. Figure 7 illustrates an example workflow for this scenario.

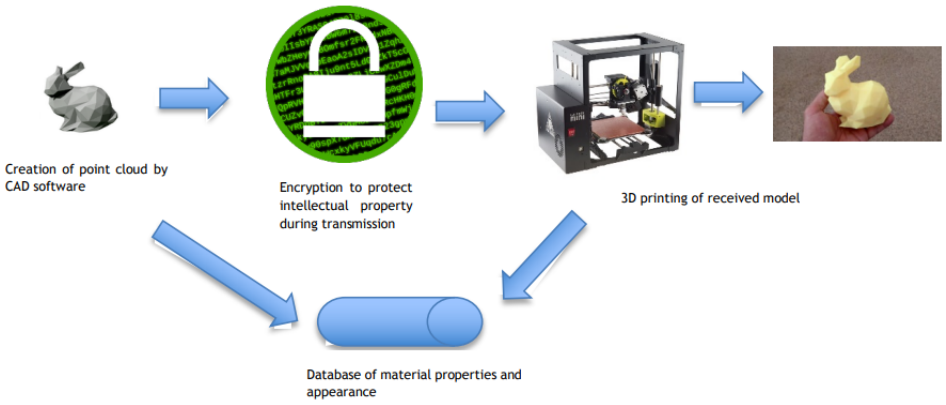


Figure 7. Example of design, manufacturing and printing use case [4].

1.2.6 Medical Imaging

Point clouds find significant application in 3D medical imaging, where they are generated by scanning the internal or external anatomy of human and animal subjects. This use case serves various purposes such as research, medical record-keeping, diagnosis, and treatment planning. They are also used in the development of medical prosthetics.

1.2.7 Construction and Manufacturing

Point clouds generated from 3D scanning are used to detect defects, damage, shape deviations, and cracks in construction materials and industrial parts. Preserving fine details and sharp edges through lossless coding is crucial for accurate defect detection. Multiple point clouds may be stored to handle object internal structures and occlusions. Compressed domain 3D processing and computer vision operations assist in automating defect detection.

In addition, point clouds have emerged as valuable tools for safety inspections during construction projects. By creating point clouds at key milestones, builders can ensure precise adherence to the architect's blueprint. These records serve as a reference for potential issues that may arise with the building in the future.

Point clouds are employed to visualize architecture and infrastructure in construction sites, allowing the overlay of planned elements on existing ones. In the realm of education and entertainment, real-world locations like the Taj Mahal or the Sagrada Familia can be accurately captured and stored as point cloud data. By utilizing a render farm, an incredibly detailed digital model can be created, offering opportunities for special effects in feature films or facilitating virtual tours for online visitors [6].

Point clouds also play a crucial role in the preservation and restoration of historical monuments. Rather than allowing these landmarks to fade away over time, point clouds can be employed to record them for future generations or aid in restoration efforts. For example, in the restoration of Notre-Dame de Paris after the devastating fire in 2019, a billion-data-point point cloud was consulted.

3D scanning of confined or hazardous areas aims to detect defects, damage, shape distortions, or cracks. Safety inspections during construction projects are another emerging application for point clouds. By creating point clouds at key milestones, builders can ensure

precise adherence to the architect's blueprint. These records serve as a reference for potential issues that may arise with the building in the future.

1.2.8 Consumer and Retail

Point clouds are utilized to capture the 3D shape and intricate details of small objects, particularly for online shopping experiences. These point clouds often exhibit complex structures and high resolutions, emphasizing material appearance. Retail websites and mobile applications frequently render these point clouds. Handling object internal structures and occlusions often necessitates the use of multiple point clouds. These point clouds exhibit complex structures and high resolutions, including attributes such as glossiness or transparency.

1.2.9 Cultural heritage

Point clouds are employed to capture the 3D shape and intricate details of objects like jewellery, pottery, bones, fossils and other small to large-size artefacts. Cultural heritage point clouds serve the dual purpose of enabling detailed analysis and visualization on websites or mobile devices for public access. Figure 8 shows a culturally significant object scanned as a point cloud.

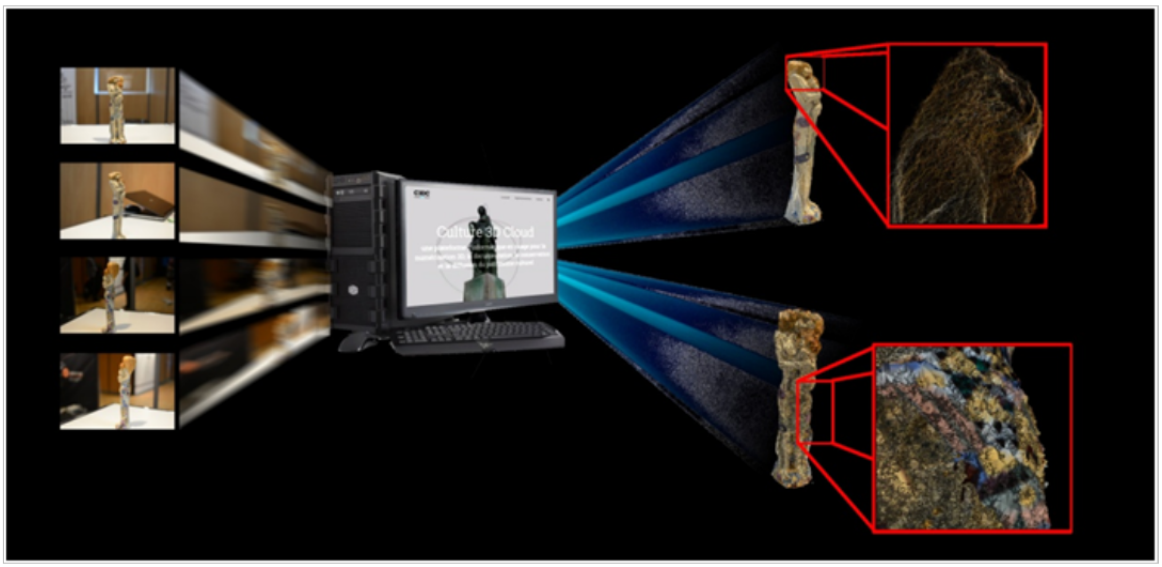


Figure 8. Cultural heritage in the 3D Cloud project [5].

Attesting to these use cases we've seen a huge number of papers exploring the application of point clouds in these domains [7–16].

1.3 Background

Point clouds in their raw format require a huge amount of memory for storage or bandwidth for transmission. For example, a typical dynamic point cloud used for entertainment purposes usually contains about 1 million points per frame, which at 30 frames per second amounts to a total bandwidth of 3.6 Gbps if left uncompressed. Furthermore, the emergence of higher resolution point cloud capture technology imposes, in turn, even a higher requirement on

the size of point clouds. In order to make point clouds usable, compression is necessary.

The data contained within a point cloud can be divided into two distinct components: geometry data, which encompasses the three-dimensional coordinates of each point, and attribute data, which includes additional information associated with each point, such as colour, normals, or reflectivity. Consequently, point cloud compression schemes are categorized into two main groups, namely geometry compression algorithms and attribute compression algorithms, depending on the specific focus of the compression approach. At the same time, certain techniques can handle both of them simultaneously. In this thesis, the focus is on the Geometry compression of point cloud data. As in other compression techniques, point cloud geometry compression can be lossless, lossy, or even near-lossless. We primarily focus on lossless schemes.

Point clouds are acquired through various techniques, including camera arrays, LiDAR sensing, and RGBD cameras. The choice of capturing technique is crucial as it imparts specific characteristics to the resulting point clouds, which can be effectively exploited during compression. Key considerations for point cloud compression algorithms include the geometry resolution and the number of points, with the surface sampling density being determined by their ratio.

Camera arrays typically yield dense, regularly sampled point clouds that lend themselves well to 2D projections. RGBD cameras capture point clouds that can be represented as 2D images with depth and colour information. These point clouds exhibit density, and regular sampling, and are susceptible to occlusions. On the other hand, spinning LiDAR sensors, commonly used in autonomous driving applications, generate point clouds that can be approximated as 2D images with depth and reflectance attributes. While these LiDAR point clouds are sparse and unevenly sampled in euclidean space, they demonstrate increased density and more uniform sampling in spherical space, which approximates the sensing mechanism.

Following the acquisition of point cloud data, a crucial preprocessing step involves quantizing the geometry information through conversion into integer coordinates. This process, commonly known as voxelization, entails representing each point in three-dimensional space as a cubic cell called a voxel, positioned within a regular 3D cubic grid. Voxelization allows for differentiation between occupied voxels, which correspond to points in the point cloud, and unoccupied voxels, representing empty space.

It's important to note that voxelization can be either lossy or lossless, depending on the chosen voxel size parameter. When the voxel size aligns with the decimal precision of the point cloud coordinates, voxelization results in lossless compression. For example, if the point coordinates are in the format 3.53, 4.89, 5.00, voxelizing with a voxel size of 0.01 would preserve the original geometry information without any loss. However, if a larger voxel size, such as 1, is used, the voxelization process becomes lossy as it introduces quantization errors and information loss. Figure 9 shows a visualization of a point cloud and the same point cloud voxelized with voxel size = 1.

Now the biggest concern after voxelization is that in the voxel domain, the number of empty voxels is much higher than the number of occupied voxels. If we encode the voxels as is we're wasting too many bits to encode empty voxels and we would instead prefer to only encode the occupied voxels. We'll take a deeper look at how we can go from this step to a more compressed format in the rest of this thesis. As such both the JPEG [18] and the

(a) Original Point Cloud **(b)** Voxelized with voxel size = 1

Figure 9. Voxelization process [17]

MPEG [19] groups have ongoing standardization processes involving point cloud data.

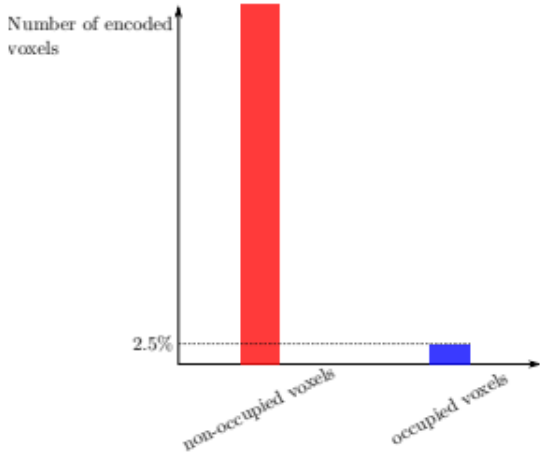


Figure 10. Occupied vs non-occupied voxels

It is worth noting that in the case of point clouds, it is possible to achieve improved quality with lower bitrates. For instance, coding sparse samplings of surfaces tends to be more computationally demanding compared to dense samplings. Sparse samplings may also lead to inferior renderings and pose safety concerns, such as occupancy false negatives (areas that are not sampled) in the context of autonomous driving. Figure 10 shows the percentage of non-occupied voxels vs occupied voxels. It shows the amount of compression gains to be had if only occupied voxels are compressed.

1.4 Structure of the Document

We take a look at the current state-of-the-art techniques in Section 2. Then in Section 3 we take a look at some of these techniques in more detail. In Section4 several improvements and extensions to the most promising technique are discussed and implemented and the corresponding results are shared. While in Section 5 we show the results for the different improvements and extensions and analyse these results. Section 6 provides the conclusions and proposals for possible future work.

2 State of the art

In our analysis of point cloud compression, we distinguish between the compression of geometry and attributes and mainly focus on geometry compression. Furthermore, we specifically address approaches that are tailored to LiDAR point clouds, as these datasets often exhibit unique structures that require specialized treatment [20].

2.1 Regular Point Clouds

Existing lossless point cloud geometry compression designs can be categorized based on how a point cloud is represented during the encoding process. These representations serve as the foundation for the encoding process while aiming to minimize the number of encoded non-occupied voxels.

The most common representation used in compression schemes is octree decomposition. In this approach, the cubic voxel space is recursively divided into eight equally-sized cubes called octants [21]. During each iteration, only octants containing at least one occupied voxel are further subdivided into child octants, and their occupancy status is indicated using a single bit. This hierarchical tree structure enables a priori compression of the point cloud geometry. However, the performance of the algorithm is significantly improved by incorporating transforms, contexts, and entropy encoders. Figure 11 shows a view of an octree construction in 2 dimensions (2D).

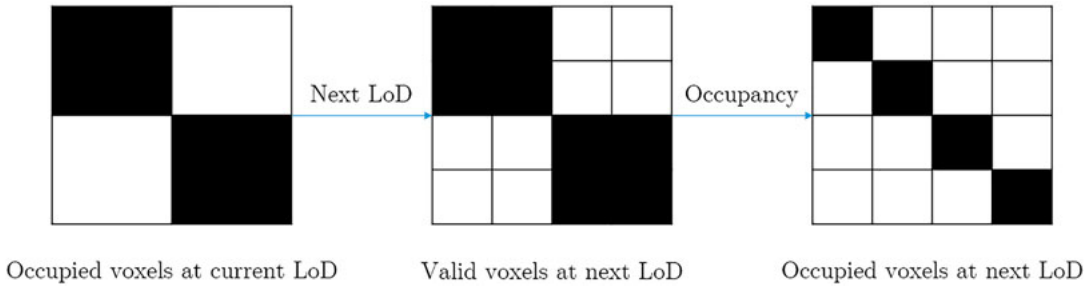


Figure 11. Simplified 2D view of an octree construction [20].

Intra-frame methods often combine octrees with context-based arithmetic or LZW encoders [22]. Transform-based methods, on the other hand, use reversible transformations such as context-adaptive binary 3D Cellular Automata [23–25] on octets. For dynamic point cloud scenes, inter-frame-based designs perform operations on octrees of adjacent frames [26, 27] or sort the octree based on previously encoded frames to optimize context utilization [28]. In some approaches, the entropy encoder utilizes a large number of contexts, and even the context frequency tables are compressed using an octree representation [22, 29, 30]. Figure 12 shows the same octree decomposition in 3D.

Apart from octree decomposition, there are other point cloud representations, such as the recent G-Arrays [32], known for their reduced memory requirements and fast lookup speeds. In contrast to the TMC13 (also called G-PCC which stands for Geometery-Point Cloud Compression) lossless geometry compression variant of MPEG, which also relies on octree representation, the standardization working group has introduced a lossy alternative. This alternative involves projecting the point cloud onto 2D surfaces and utilizing standard 2D video encoders [33, 34]. Projection-based compression algorithms typically represent point cloud geometry through a series of 2D projections using occupancy and depth maps.

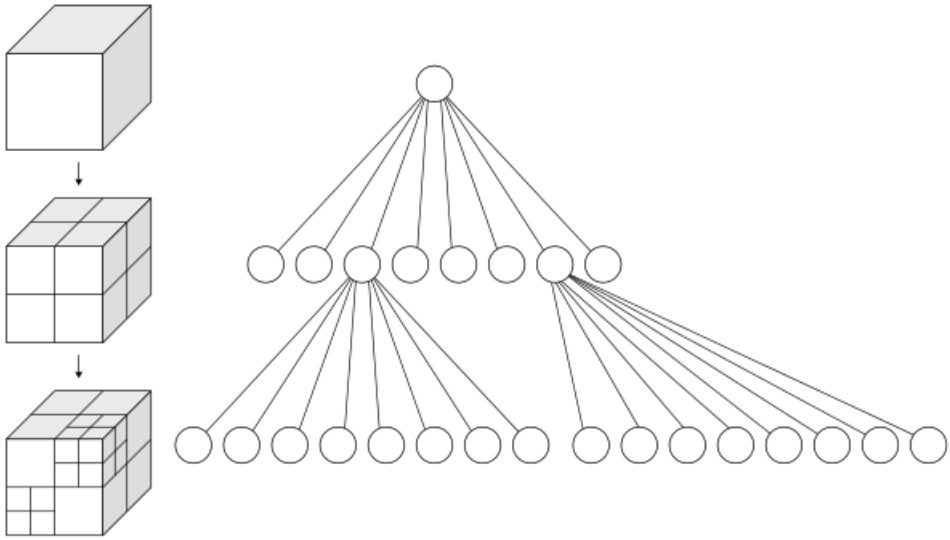


Figure 12. Left: Recursive subdivision of a cube into octants. Right: The corresponding octree [31].

There are both lossy [35–37] and lossless methods [38] available for this approach.

There are alternative approaches that directly operate on the voxel domain, working with the coordinates of occupied voxels. One such approach, as described in [39], involves encoding the residual differences between these voxel coordinates. The advantage of voxel-domain encoders is their ability to bypass complex preprocessing procedures typically associated with other representation methods, such as octree construction or the calculation of various projection maps.

A specific subset of voxel domain compression methods utilizes the occupancy matrix to represent the structure of the point cloud. The entire cubic voxel space, consisting of N^3 voxels, is modeled as a three-dimensional sparse boolean matrix of size N×N×N. The elements of this matrix hold a value of 1 for occupied voxels. Each 2D slice of the occupancy matrix, perpendicular to the x, y, or z axis, is represented by a N×N boolean array, referred to as an image.

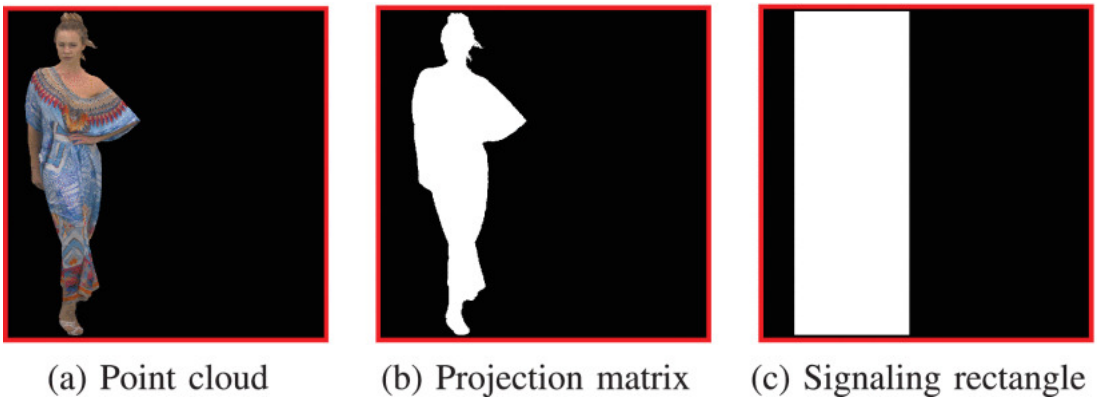


Figure 13. Depiction of a point cloud along with different options to limit encoded voxels [40].

In [41], the authors employ this representation to compress the occupancy matrix of a point cloud through a boolean decomposition, followed by a dyadic decomposition and the

utilization of 3D contexts [42]. These techniques yield impressive results. To further enhance the compression performance, an inter-frame variant of the method was introduced [43], and the context selection process was refined [44, 45]. Another improvement to the intra-frame design was presented in [46]. This enhancement involves eliminating the dyadic decomposition and utilizing an N×N projection matrix, which is calculated by performing a boolean OR operation across all images of the occupancy matrix that are perpendicular to the x, y, or z axis. The resulting binary projection matrix, illustrated in Figure 13, represents the shading of the point cloud on the plane perpendicular to the chosen axis (x, y, or z). By transmitting this binary matrix to the decoder, one can effectively limit the number of encoded voxels, as the absence of occupied voxels can be inferred from precise 2D coordinates across all images. Figure 13 shows some techniques that can be used to limit encoded voxels which several techniques use as a part of their pipeline.

2.2 LiDAR Point Clouds

While the previous section focuses on general point cloud geometry compression, this section focuses on techniques primarily aimed at LiDAR point cloud compression.

LiDAR point clouds exhibit specific characteristics due to their sparse nature in 3D space and non-uniform density. These structures can be leveraged for compression, as the point clouds often reside on lower-dimensional manifolds within 3D space.

When dealing with LiDAR point clouds, they can be represented as sequences, and specialized codecs like LASzip [47] are designed to treat them as such. These codecs employ metadata related to the LiDAR scanning mechanism [48] to achieve lossless, order-preserving compression. In the case of spinning LiDAR sensors, the authors [49] propose a separation of the point cloud into the ground, object, and noise layers. The ground points, forming circular patterns and comprising the majority, can be efficiently compressed. Object points, on the other hand, are typically sparsely distributed clusters.

Transforming LiDAR point clouds into depth images is another approach, approximating the sensor mechanism using quantized spherical angular coordinates. While this transformation is often lossy, newer sensors provide lossless [50] depth images directly. The acceptability of the distortion introduced by this transformation depends on the application. The authors in [51] employ predictive coding in hybrid coordinate systems (spherical, cylindrical, and 3D space) to compress range images, utilizing mode selection for prediction. Similarly, [52] segment the range image into ground and objects, followed by predictive and residual coding.

In some cases, LiDAR point cloud packets can be compressed directly. These packets have a natural 2D arrangement with laser index and rotation, making them more suitable for low-latency scenarios compared to full frames. Image compression techniques can be applied to compress such packets [53]. Additionally, [54] propose the use of Simultaneous Localization And Mapping (SLAM) to predict and compress dynamic LiDAR point clouds.

In the three-dimensional space, the structure of the LiDAR sensor can serve as a prior for octree coding. Parameters like the number of lasers, their positions, angles, and angular resolutions per frame can be encoded and utilized to enhance compression. When the acquisition process is known, only valid voxels or those deemed plausible according to the acquisition model need to be encoded. This approach, known as the angular coding mode in G-PCC [55], has been implemented to improve compression efficiency.

2.3 Deep Learning

In Deep Learning based point cloud compression, voxel-based approaches are typically better suited for dense point clouds, while point-based approaches are more suitable for sparse point clouds. The complexity of voxel-based approaches is influenced by the dimension of the voxel grid (precision), whereas the complexity of point-based approaches depends on the number of points in the cloud.

Considering prior information is also crucial. The sensor itself often provides the primary source of structure in point clouds. For instance, spinning LiDAR sensors can be approximated using a spherical coordinate system, point clouds captured with camera arrays can be represented as multiple 2D images, and RGBD cameras produce point clouds that can be stored on a single 2D image, among other examples. Leveraging this prior information can lead to significant compression improvements, but it necessitates specific processing tailored to each type of prior. Figure 14 shows a general taxonomy of deep learning based point cloud compression techniques.

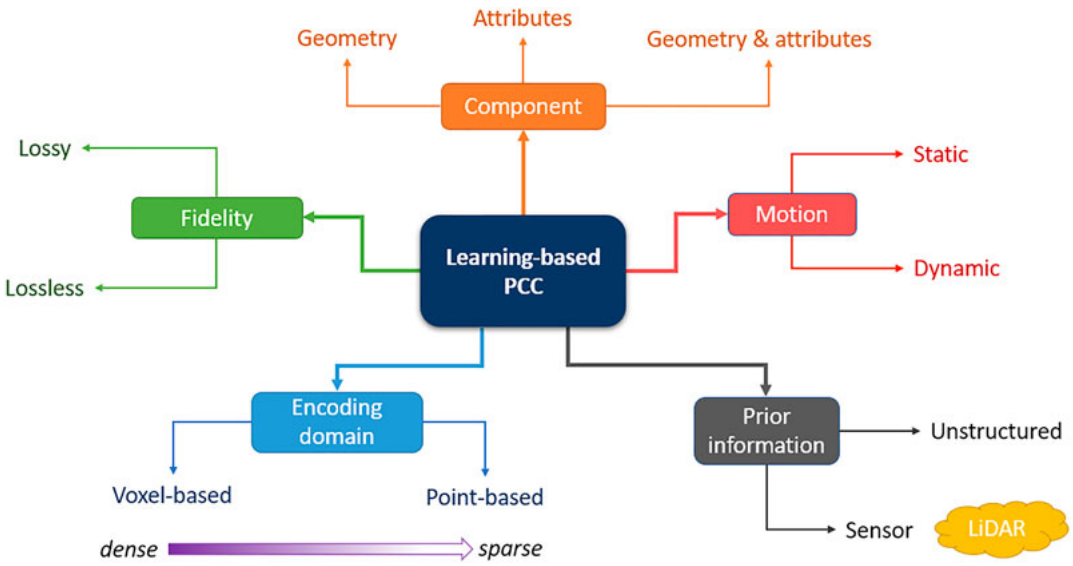


Figure 14. A taxonomy of Learning-based Point Cloud Compression [20].

Recent advancements in point cloud geometry compression have witnessed the utilization of Deep Neural Networks (DNNs), which have achieved remarkable compression ratios. For lossless designs, researchers have adopted a hybrid representation approach, combining octree-based voxel processing with rendering of voxel sub-volumes using occupancy matrices [38, 56–58]. However, it is worth noting that the complexity associated with these approaches results in longer encoding times.

While these hybrid and occupancy-based methods offer competitive compression ratios, they suffer from increased memory requirements and significant computational costs. The sequential nature of processing one voxel at a time, along with encoding a large number of non-occupied voxels, severely hampers the compression speed. For instance, in occupancy-based methods like [46], it is common for only 2.3% of the processed and encoded voxels to be occupied. Similarly, hybrid-based methods such as [58] demonstrate that the percentage of occupied voxels in the initial level of processed sub-volumes is below

5%.

A new generation of deep learning methods has emerged that aims to leverage the sparsity inherent in point clouds through various approaches. One such method, called SparsePCGC (Sparse Point Cloud Geometry Compression) [59], employs a Sparse Convolution-based Neural Network (SparseCNN) that combines sparse convolutions and voxel sampling to effectively capture and embed spatial correlations. Building upon this foundation, the researchers developed the SparseCNN-based Occupancy Probability Approximation (SOPA) model, which can estimate the occupancy probability in either a single-stage manner, relying solely on cross-scale correlations, or in a multi-stage manner, utilizing correlations among same-scale neighbours at different stages. By integrating these techniques, they have achieved state-of-the-art performance among deep learning methods, excelling in both compression ratio and speed.

3 Detailed Examination of Advanced Techniques for PCC

In this section, we examine a subset of the state-of-the-art techniques seen above in much more detail.

3.1 MPEG

As we've seen already the MPEG team has been working on standardisation efforts related to point cloud compression.

3.1.1 V-PCC

2D video compression has proven to be a highly successful technology, widely utilized due to the extensive adoption of video coding standards. To leverage this technology, point cloud compression (PCC) methods convert the 3D point cloud data into a 2D representation, which can then be encoded using 2D video encoders. The MPEG group also has significant experience in video-based coding techniques and thus being able to transfer and leverage this expertise was a huge plus. Figure 15 shows the 3D patch projection in action.

The proposal that served as the foundation for the V-PCC test model introduced the concept of generating 3D surface segments, referred to as 3D patches, by dividing the point cloud into connected regions [60].

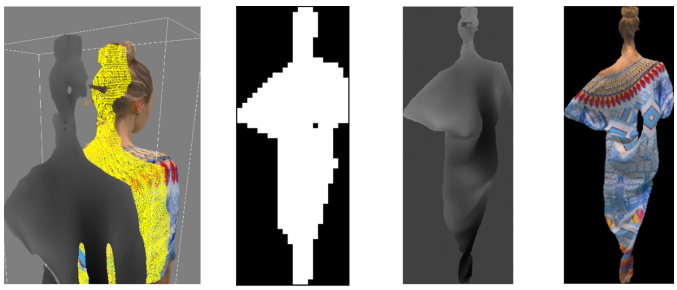


Figure 15. 3D Patch projection and respective occupancy map, geometry, and attribute 2D images, (a) 3D patch, (b) 3D Patch Occupancy Map, (c) 3D Patch Geometry Image, (d) 3D Patch Texture Image [60].

Each 3D patch was then independently projected into a 2D patch. This approach effectively addressed projection challenges, such as self-occlusions and hidden surfaces, and provided a feasible solution for the point cloud conversion task. Furthermore, orthographic projections were utilized to avoid resampling issues and enable lossless compression.

The process of projecting a 3D patch onto a 2D patch can be likened to using a virtual orthographic camera to capture a specific portion of the point cloud. It can be visualized as having multiple virtual cameras capturing different parts of the point cloud and combining those camera images to form a mosaic, essentially an image encompassing the collection of projected 2D patches as seen in Figure 16. This procedure yields a set of metadata associated with each patch projection, analogous to describing each virtual camera capture.

For each patch, there can be up to three associated images:

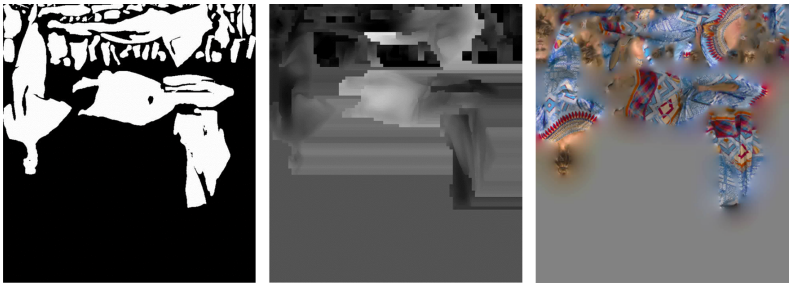


Figure 16. Example of patch packing [60].

- An occupancy map, a binary image indicating whether a pixel corresponds to a valid 3D projected point
- A geometry-image containing depth information, representing the distance between the point's location and the projection plane
- One or more attribute images, such as the texture (color) of each point, represented by a color image with R, G, and B components, or a material attribute represented by a monochrome image

In the case of V-PCC, the geometry image, in particular, resembles a depth map since it records the distance between the 3D point and the projection plane. Furthermore, these mappings were exclusively used with 3D meshes rather than point clouds.

3.1.2 G-PCC

The V-PCC coding approach revolves around projecting the 3D point cloud onto a 2D representation, while G-PCC takes a different approach by directly encoding the content in 3D space. G-PCC achieves this by employing data structures, such as an octree, to describe the point locations in 3D space. An exhaustive explanation of these data structures will be provided in the subsequent section. Unlike V-PCC, G-PCC does not make any assumptions regarding the coordinate representation of the input point cloud. Instead, the points are internally represented using integer-based values derived from floating-point representations. This conversion process is conceptually akin to voxelization, involving scaling, translation, and rounding operations.

Another important aspect of G-PCC is the utilization of tiles and slices to enable parallel coding functionality. In G-PCC, a slice refers to a collection of points encompassing both geometry and attributes, which can be encoded and decoded independently. On the other hand, a tile is a grouping of slices that includes bounding box information. Tiles can overlap with each other, allowing the decoder to selectively decode specific slices to reconstruct a partial area of the point cloud.

One limitation of the current G-PCC standard is its exclusive reliance on intra-prediction, meaning it does not incorporate any temporal prediction tools. However, upcoming versions of the standard are exploring techniques based on point cloud motion estimation and inter-prediction to address this limitation.

Figure 17 illustrates a block diagram of the key components in the G-PCC reference encoder, also known as TMC13. The encoder operates on the principle of separately encoding geometry and attributes, with attribute coding relying on decoded geometry. Consequently, the initial phase involves coding the point cloud positions, which are often represented as floating-point values in a world coordinate system. This process begins with

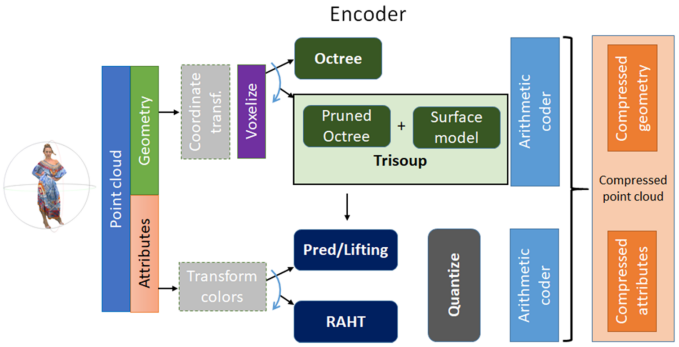


Figure 17. G-PCC reference encoder [60].

a coordinate transformation, followed by voxelization. Subsequently, geometry analysis takes place using either the octree or trisoup scheme. The resulting structure is then subjected to arithmetic encoding.

The voxelized point cloud is efficiently represented using an octree structure in a lossless manner. Let's consider a quantized volume of D × D × D voxels to contain the point cloud. Initially, the volume is divided into eight sub-cubes, each with dimensions of D/2 × D/2 × D/2 voxels. This recursive process is repeated for each occupied sub-cube until D reaches 1. It is worth noting that typically only around 1% of voxel positions are occupied, making octrees an effective choice for representing point cloud geometry. The octets generated during this process represent the occupancy state of an octree node and are compressed using an entropy coder that considers the correlation with neighbouring octets. Figure 18 shows the initial steps in octree construction.

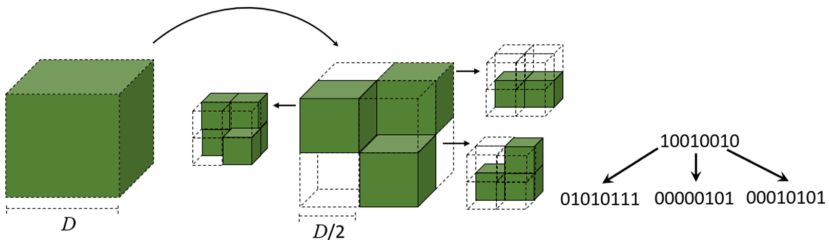


Figure 18. Initial steps in octree construction [60].

Another approach is to represent the geometry using a pruned octree, which is created by constructing the tree from the root down to a certain level. In this representation, the occupied sub-blocks that are larger than a voxel are represented by the tree's leaves. To approximate the object's surface, a set of triangles is used. Since there is no connectivity information between these triangles, this technique is referred to as "triangle soup" or "trisoup" as seen in Figure 19. Although this method is entirely optional.

Within G-PCC, three distinct methods are available for attribute coding:

- RAHT(Region-adaptive Hierarchical Transform)
- Predicting Transform
- Lifting Transform

The underlying principle of RAHT involves utilizing attribute values from a lower level of the octree to predict values at the next level. The Predicting Transform employs an interpolation-based hierarchical nearest-neighbor prediction scheme. Additionally, the Lifting Transform builds upon the Predicting Transform by incorporating an additional update

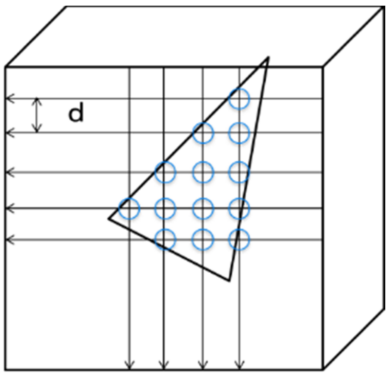


Figure 19. Trisoup point derivation [60].

or lifting step.

G-PCC is considered to be the current baseline to beat in terms of compression speed and efficiency.

3.2 SparsePCGC

They start from a voxelized representation. Figure 20 illustrates the volumetric representation of a PCG using a densely-sampled uniform voxel grid known as the "uniform voxel" representation. Binary occupancy status is used to indicate whether a voxel's geometric position is positively occupied or not. Positively-Occupied Voxels (POVs), depicted as solid grey in the figure, are labelled with "1" (e.g., POV(1)), while Non-Occupied Voxels (NOVs) within white boxes are labelled with "0" (e.g., NOV(0)). These POVs are obtained through voxelization, converting the raw "points" of the original PCG.

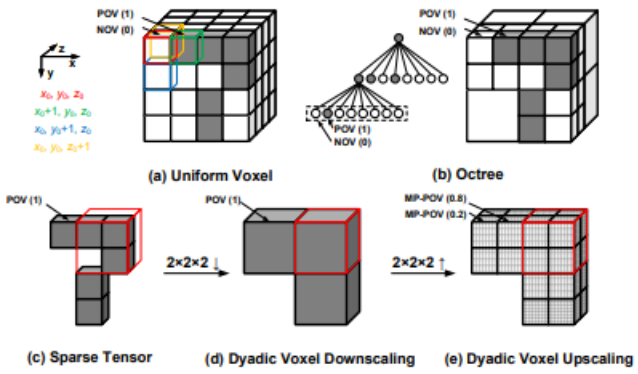


Figure 20. Various Representation Models of Voxelized PCG [59].

Given the non-uniform distribution of POVs in an input PCG, they propose the utilization of a Sparse Tensor to effectively exploit the sparsity of POVs. With the sparse tensor, only the geometric coordinates and applicable attributes are stored for Most-Probable POVs (MP-POVs). MP-POVs are generated by upscaling POVs from lower scales via voxel upscaling, as shown in Figure 20, to establish a scale-wise sparse tensor within a multiscale representation framework.

The original PCG is progressively downscaled into multi-resolution PCGs and correspondingly upscaled for hierarchical reconstruction, as depicted in Figure 21. Dyadic voxel sampling is employed for upscaling or downscaling the related sparse tensor, involving a stride of 2 along each axis (x, y, and z-axis in a Cartesian coordinate system). The process of merging eight connected sub-voxels within a 2 × 2 × 2 voxelized box into a single voxel constitutes the downscaling step during encoding. If any of the sub-voxels is a POV, the merged voxel is considered a POV.

In the encoding phase, full knowledge of each POV is available at each scale. On the other hand, during decoding, a specific voxel is subdivided into eight sub-voxels. If this voxel is a POV, its eight children are identified as MP-POVs since it is not possible to determine which sub-voxel is positively occupied immediately after upscaling. The collection of MP-POVs forms a superset of POVs.

The proposed SparsePCGC method encodes the occupancy of MP-POVs at each scale into a compressed bitstream and subsequently decodes the binary bits to reconstruct POVs at the same scale. The accuracy of the occupancy probability approximation for each MP-POV is crucial for the performance of SparsePCGC, regardless of whether lossy or lossless compression is employed. Figure 21 shows the different steps in the entire pipeline.

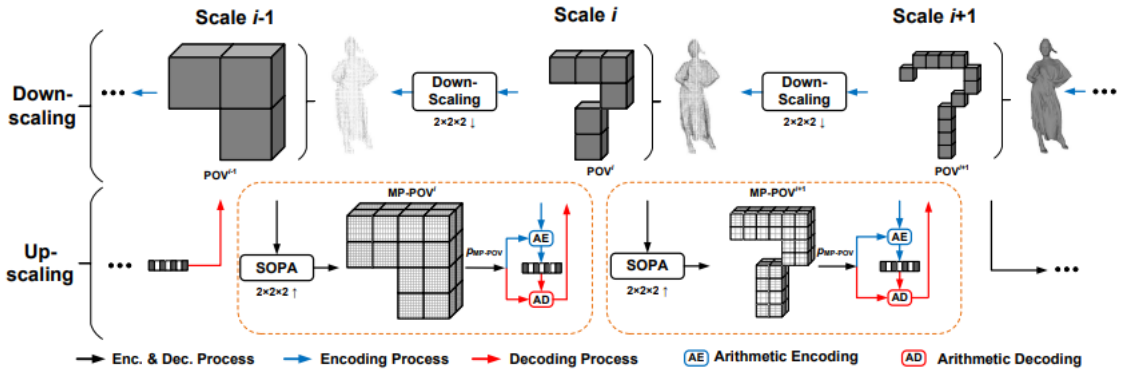


Figure 21. SparsePCGC [59].

Within the multiscale representation model, cross-scale context modelling is conducted by utilizing decoded POVs (including occupancy status and feature attributes, if applicable) from the preceding lower scale to approximate the occupancy probability of each MP-POV at the current scale. It is important to note that cross-scale context modelling is limited to two consecutive scales only. This enables the training of models using samples from two consecutive scales and their subsequent application for inference across all scales. This strategy of sharing models across scales reduces the overall model size significantly.

The proposed cross-scale context modelling is achieved through SparseCNN-based Occupancy Probability Approximation (SOPA). A single-stage SOPA can approximate the occupancy probabilities of eight upscaled MP-POVs simultaneously. Alternatively, multi-stage SOPA can leverage neighbourhood correlation to progressively approximate occupancy probabilities.

With these techniques, they are achieving very high compression ratios while at the same time maintaining decent encoding speed which was a pain point for previous deep learning-based compression techniques.

3.3 FRL

The FRL (Fast Run-Length) compression scheme directly competes with TMC13 as a non-deep learning lossless geometry point cloud compression scheme. The simplified version of the proposed single-pass compression scheme in Figure 22 highlights the four main processes along with their corresponding inputs and outputs, which are,

- Run extraction
- Context assignment
- Probability computation
- Arithmetic Encoder

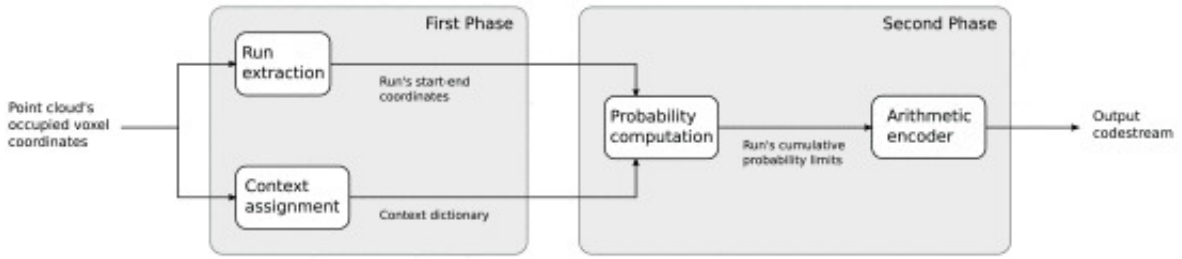


Figure 22. FRL Compression Scheme [40].

The initial phase involves two processes: run extraction and context assignment. These processes take the coordinates of the occupied voxels in the point cloud as input. Before the run extraction process is started a global signalling rectangle is computed, which is a signalling rectangle defined along the slice axis within which all points in all slices are contained. If a 3D point cloud is converted into a series of images, each image is a slice. The run-extraction process then scans the geometry information of the point cloud and identifies runs of voxels within a single 2D plane of the voxel space (slice). One crucial element that contributes to the accelerated speed of FRL is the formation of symbols based on runs of voxels, rather than individual voxels, significantly lowering the computational overhead.

Each run consists of a sequence of non-occupied voxels that follow the end of the previous run and a single occupied voxel. In other words, the run terminates as soon as the first occupied voxel is encountered while following a raster-scan order. The process concludes by providing the voxel coordinates corresponding to the starting and ending points of each run. In Figure 23 squares correspond to voxels, with the ones marked with a dot being occupied. The signalling rectangle is traced with a bold line. Each run is marked with a different colour.

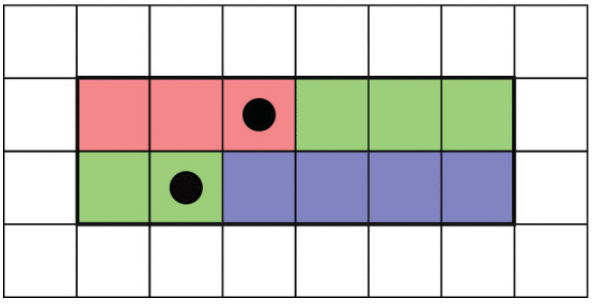


Figure 23. Example of runs in a slice [40].

To improve the prediction accuracy of voxel occupancy in a run the neighbouring voxels' statuses are taken into account by means of contexts [42]. Specifically, 3D contexts are employed, consisting of 5 voxels located on the current encoded slice and 9 voxels located on the previous slice, totalling 14 voxels. Figure 24 illustrates the context map used in the proposed compression scheme, indicating the spatial arrangement of the voxels that form the context in relation to the target voxel under examination. Each context is associated with a unique integer value determined by a 14-bit bitmask that is constructed based on the occupancy status of its constituent voxels. The presence of an occupied voxel at position i in the context map contributes a value of 2^i to the integer representation of the context. As a result, the proposed scheme utilizes 2^{14} distinct contexts, with integer representations ranging from 0 for the empty context composed solely of non-occupied voxels, to $2^{14}-1$. In the Figure 24 2D coordinates of v and the 9th context coincide. If the non-occupied voxels are noted in blue and the occupied in red, v's context's integer representation is $2^2+2^7=132$.

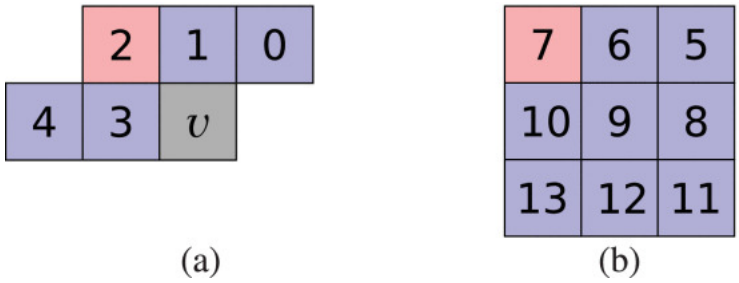


Figure 24. 3D Context maps [40].

The following example Figure 25 illustrates the computation of the context dictionary. Occupied voxels are represented by dotted squares and are identified by their coordinates, while shaded squares represent voxels associated with a non-empty context. This simplified illustration focuses on context map positions 1 and 3.

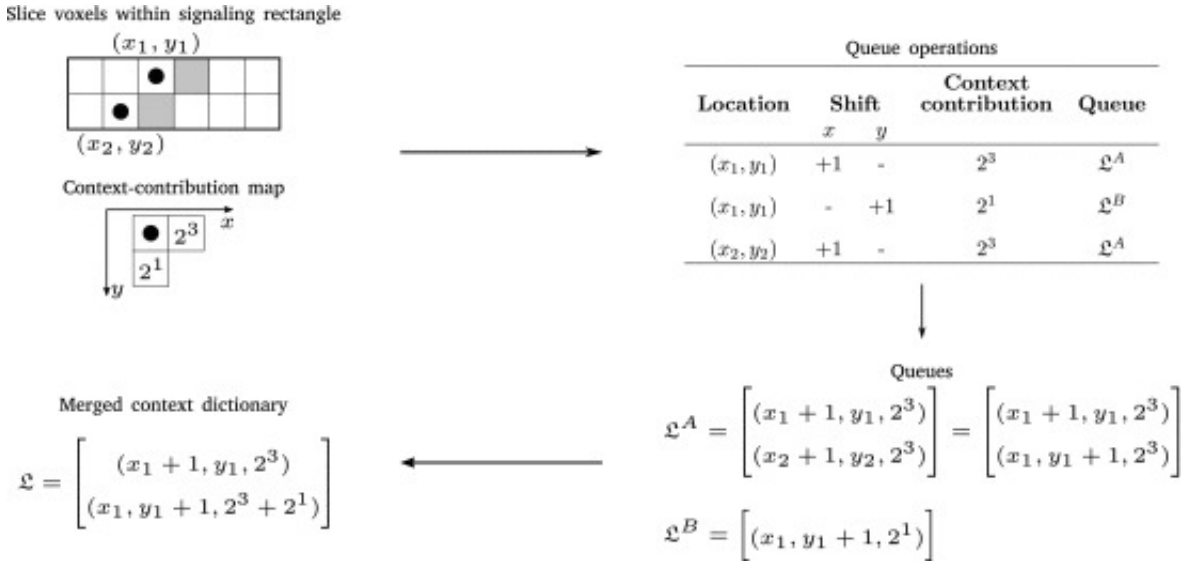


Figure 25. Context dictionary [40].

The second phase consists of the probability computation and arithmetic encoder processes. Before encoding the runs with the arithmetic encoder, it is necessary to assign upper and lower cumulative probability limits to each symbol. This is accomplished in the

probability computation process, which utilizes the outputs from the first phase. By employing various numerical optimizations, the calculation of these limits is expedited, reducing their complexity to the scale of occupied voxels, regardless of the number of non-occupied voxels being transmitted. The occurrence probability of each run is determined through an independent joint probability calculation that takes into account the occupancy status of its constituent voxels. Additionally, the likelihood of a voxel's status is adaptively estimated using conditional probabilities and contexts. The number of occurrences of each specific context with respect to occupied or non-occupied voxels is tracked and updated to facilitate this estimation process. The Context Adaptive Binary Arithmetic Encoder (CABAC) used here performs very well in terms of compression ratios as well as speed.

The precedent drawbacks of hybrid and occupancy-type methods are addressed by their proposed lossless intra-frame geometry compression scheme while maintaining state-of-the-art compression performance. Only the point coordinates are used in their discussed method, which allows them to encode the point cloud directly on the voxel domain without rendering the voxel space, contributing to its memory-efficient design. Furthermore, grouping the encoded voxels in runs drastically decreases the encoding time. The introduction of several improvements accelerates key calculations, lowering the encoder's computational complexity to the order of occupied voxels, regardless of the number of encoded non-occupied voxels. Lastly, the employment of 3D contexts and an adaptive arithmetic encoder yields significant compression results. Figure 26 shows the bitrate of FRL against other very competitive methods.

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

Figure 26. Rate Comparison in Bits per Occupied Voxel (bpov) [40].

FRL outperforms TMC13 standard in dense point clouds with compression gains of up to 15%. Although deep learning methods report better compression ratios they can't be deployed in a lot of point cloud use cases wherein compression speed is of paramount importance. The following Figure 27 of the relative encoding speed of FRL against TMC13 as baseline vs bitrate clearly demonstrates it.

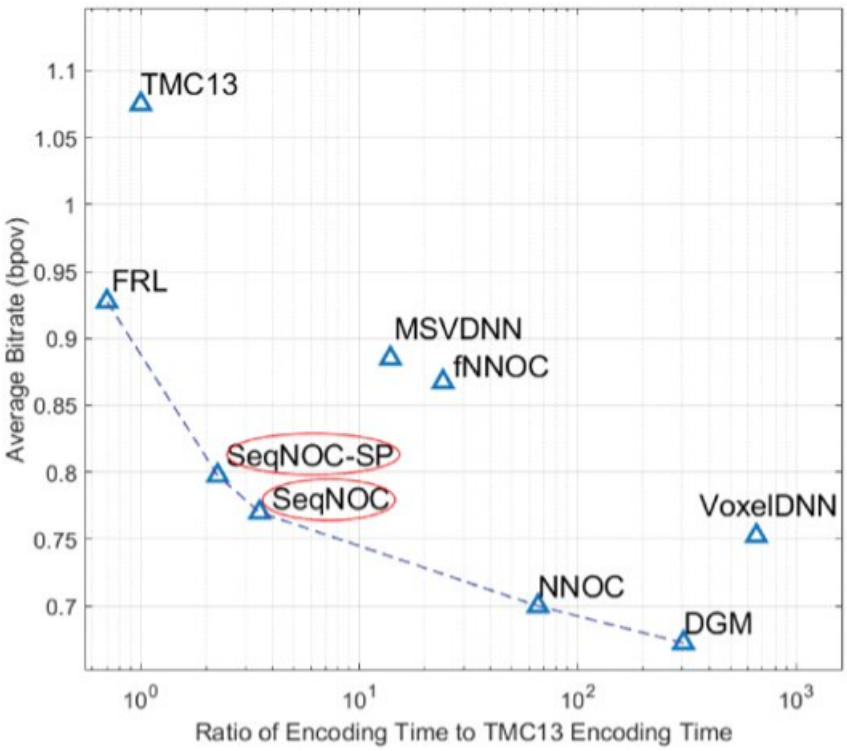


Figure 27. Encoding speed vs bitrate [61].

4 FRL Revisited

After thoroughly studying the state-of-the-art algorithms for point cloud compression and exploring several of them in detail, FRL seemed to be a very promising technique. It demonstrated exceptional compression ratios and exhibited the fastest encoding speeds among the algorithms investigated. In this chapter, we take a deeper look at FRL and propose several improvements and extensions to augment its performance in the various stages of the compression pipeline.

The primary objective of the proposed improvements and extensions is to address two fundamental aspects: compression ratios and compression speed. By focusing on these areas, the aim is to strike a balance between achieving higher compression ratios to reduce storage requirements and enhancing the compression speed for efficient encoding and decoding processes.

The proposed enhancements and extensions are thoroughly evaluated through extensive experiments and comparative evaluations. Performance metrics such as compression ratios, encoding speeds, and decoding speeds are measured and compared against the original FRL technique and other state-of-the-art compression algorithms. These results are presented in Section 5.2 and these evaluations provide valuable insights into the effectiveness of the proposed changes and their impact on the overall compression performance.

4.1 Statistics

Similar to how image properties play a huge role in tuning image compression techniques to exploit those properties, a comprehensive understanding of the properties specific to point clouds is crucial for achieving compression gains in point cloud data. An extensive analysis of point cloud properties was conducted in this research. A diverse dataset of point cloud files was examined, and a comprehensive set of statistics was computed for each file.

The computation of these statistics serves multiple purposes. Firstly, it allows for the quantification and validation of various hypotheses regarding point cloud characteristics and behaviours. Moreover, to our knowledge, this is also one of the first studies to thoroughly analyse various statistics in point clouds with such detail. In the following section, a brief introduction to the terminology used is introduced, followed by the statistics computed and an explanation of each statistic.

A preprocessed point cloud can be represented as a set of N points

$$P = \{(x_n, y_n, z_n) \in N_0^3\}_{n=1,\dots,N}$$

and, for each axis,

$$X = \{x_n\}_{n=1,\dots,N}, Y = \{y_n\}_{n=1,\dots,N}, Z = \{z_n\}_{n=1,\dots,N}$$

We can also define mathematically the i-th slice across the x axis, for $i = 1, \dots, \max X$ as

$$S_x^{(i)} = \{(x_n, y_n, z_n) \in P : x_n = i\}_{n=1,\dots,N}$$

the j-th slice across the y axis, for $j = 1, \dots, \max Y$ as

$$S_y^{(j)} = \{(x_n, y_n, z_n) \in P : y_n = j\}_{n=1,\dots,N}$$

and the k-th slice across the z axis, for $k = 1, \ldots, \max Z$ as

$$S_z^{(k)} = \{(x_n, y_n, z_n) \in P : z_n = k\}_{n=1,\ldots,N}.$$

Ranges:

For each axis, its range is from 0 to its maximum value. Here we use 0 for the minimum value but that is due to how we preprocess the point cloud, which is explained in the next section. For the sake of simplicity, we only mention the maximum value in the range as the minimum value is always 0. That is:

- Range x:
$$\max X$$
- Range y:
$$\max Y$$
- Range z:
$$\max Z$$

Bit depth:

The bit depth is a unique value which represents the minimum number of bits needed to represent each value of the point cloud. Mathematically:

- Bit depth:
$$\lfloor 1 + \log_2 \max\{\max X, \max Y, \max Z\} \rfloor$$

Note the bit depth only computes the minimum bits required for one axis of a point, so to compute the bit depth for the entire point this per-axis bit depth value is multiplied by 3 for the 3 axes.

Proportions of empty slices:

The proportion of empty slices, for each axis, is the number of slices across this axis with no points divided by the total number of slices. Or, in other words, one minus the proportion of slices with, at least, one point.

- Prop. empty sl. x:
$$1 - \frac{|\{S_x^{(i)} : |S_x^{(i)}| > 0\}|}{\max X}$$
- Prop. empty sl. y:
$$1 - \frac{|\{S_y^{(j)} : |S_y^{(j)}| > 0\}|}{\max Y}$$
- Prop. empty sl. z:
$$1 - \frac{|\{S_z^{(k)} : |S_z^{(k)}| > 0\}|}{\max Z},$$

where $|A| = \text{card} A = \# A$ is the cardinality of the set A.

Number of occupied voxels

The number of occupied voxels in a slice is the number of voxels that represent a point of the point cloud in this slice.

- **Ocup. voxels x** for each slice $i = 1, \ldots, \max X$ across the x axis:

$$|S_x^{(i)}|$$

- **Ocup. voxels y** for each slice $j = 1, \ldots, \max Y$ across the y axis:

$$|S_y^{(j)}|$$

- **Ocup. voxels z** for each slice $k = 1, \ldots, \max Z$ across the z axis:

$$|S_z^{(k)}|$$

Instead of having all the values of occupied voxels from each axis and slice, there is the min, max, mean and the median of every slice for each axis.

Signalling rectangles

The signalling rectangle of one slice is the rectangle with the minimum area that contains all the points of that slice.

Each signalling rectangle can be represented by two points, the top right point and the bottom left point. The top right point for the i-th slice across the x axis can be denoted as $(y_{tr}^{(i)}, z_{tr}^{(i)})$, and the bottom left as $(y_{bl}^{(i)}, z_{bl}^{(i)})$; $(x_{tr}^{(j)}, z_{tr}^{(j)})$ and $(x_{bl}^{(j)}, z_{bl}^{(j)})$ for the y axis, and $(x_{tr}^{(k)}, y_{tr}^{(k)})$ and $(x_{bl}^{(k)}, y_{bl}^{(k)})$ for the z axis.

The calculated values are:

- **(Bottom hor. x, Bottom vert. x)** for each slice $i = 1, \ldots, \max X$ across the x axis:

$$(y_{bl}^{(i)}, z_{bl}^{(i)})$$

- **(Bottom hor. y, Bottom vert. y)** for each slice $j = 1, \ldots, \max Y$ across the y axis:

$$(x_{bl}^{(j)}, z_{bl}^{(j)})$$

- **(Bottom hor. z, Bottom vert. z)** for each slice $k = 1, \ldots, \max Z$ across the z axis:

$$(x_{bl}^{(k)}, y_{bl}^{(k)})$$

- **(Top hor. x, Top vert. x)** for each slice $i = 1, \ldots, \max X$ across the x axis:

$$(y_{tr}^{(i)}, z_{tr}^{(i)})$$

- **(Top hor. y, Top vert. y)** for each slice $j = 1, \ldots, \max Y$ across the y axis:

$$(x_{tr}^{(j)}, z_{tr}^{(j)})$$

- **(Top hor. z, Top vert. z)** for each slice $k = 1, \ldots, \max Z$ across the z axis:

$$(x_{tr}^{(k)}, y_{tr}^{(k)})$$

Instead of having all the values of the components of the signalling rectangle points from each axis and slice, there is the min, max, mean and the median of each of them. Note this is slightly different from the global signalling rectangle previously discussed as here we find the signalling rectangle per slice and not for the entire point cloud.

Binary entropies

The binary entropy of N bits $b_1, b_2, \ldots, b_N$ is defined as

$$H_b(p) = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{1-p}$$

where

$$p = \frac{1}{n} \sum_{n=1}^{N} b_n$$

and it gets its maximum value when $p = \frac{1}{2}$, that is, when the number of 0 bits is the same as the number of 1 bits.

In our case, in a slice, we consider that a voxel has a value 0 when it is empty and 1 when it represents a point. So, for example, the binary entropy of the first slice across the x-axis will be $H_b(p_1)$, where p_1 will be the proportion of occupied voxels in this slice.

So, mathematically:

- **Entropy x** for each slice $i = 1, \ldots, \max X$ across the x axis:

$$H_b(p_i), p_i = \frac{|S_x^{(i)}|}{\max Y \max Z}$$

- **Entropy y** for each slice $j = 1, \ldots, \max Y$ across the y axis:

$$H_b(p_j), p_j = \frac{|S_y^{(j)}|}{\max X \max Z}$$

- **Entropy z** for each slice $k = 1, \ldots, \max Z$ across the z axis:

$$H_b(p_k), p_k = \frac{|S_z^{(k)}|}{\max X \max Y}$$

The entropy of each slice across each axis is also calculated but considering only the voxels of the signalling rectangle, so the probabilities are different: they are calculated by dividing by the width multiplied by the height of each signalling rectangle. The width of the rectangle of the i-th slice can be noted as $w_x^{(i)}$, and the height as $h_x^{(i)}$, and the same for the y and z axis.

So, the second entropies are:

- **SR entropy x** for each slice $i = 1, \ldots, \max X$ across the x axis:

$$H_b(p_i), p_i = \frac{|S_x^{(i)}|}{w_x^{(i)} h_x^{(i)}}$$

- **SR entropy y** for each slice $j = 1, \ldots, \max Y$ across the y axis:

$$H_b(p_j), p_j = \frac{|S_y^{(j)}|}{w_y^{(j)} h_y^{(j)}}$$

- **SR entropy z** for each slice $k = 1, \ldots, \max Z$ across the z axis:

$$H_b(p_k), p_k = \frac{|S_z^{(k)}|}{w_z^{(k)} h_z^{(k)}}$$

All of these statistics are computed for all the point clouds across all datasets.

4.2 Preprocessing

The initial preprocessing step in the FRL technique involves voxelization, which is essential for converting the original point cloud data into a voxel representation. Voxelization is a process of partitioning the point cloud space into small cubic regions called voxels. During this step, an important consideration arises depending on the nature of the original point cloud data.

For regular point clouds that do not contain floating-point values, they are voxelized using a voxel size of 1, in which case the resulting voxelization is lossless. This is because the voxelization directly represents the presence or absence of points within each voxel without any loss of information. This scenario is typical with regular point clouds, as each point in the point cloud usually has integer coordinates.

However, when working with LiDAR point clouds, a different situation arises. LiDAR point clouds often have floating-point values for the geometry coordinates, introducing a challenge for voxelization. If such point clouds are voxelized using a voxel size of 1 without any additional considerations, vital information can be lost due to the truncation of floating-point values. One simple solution to this problem is to read the floating-point values as they are and perform computations using floating-point arithmetic. However, this approach is not commonly preferred in point cloud compression techniques, as many algorithms require input points to be represented as integers. Therefore, an alternative strategy is employed to ensure lossless compression while preserving the integer representation requirement.

To overcome this challenge, a conversion process is applied to transform the floating-point values into integer values. This is achieved by multiplying each floating-point coordinate by a precision factor, represented as $10^{precision}$. The precision factor is determined based on the maximum number of values in the mantissa for the given dataset. By multiplying the coordinates, the floating-point values are effectively scaled up to become integers, enabling lossless compression without sacrificing the integer representation requirement.

After the conversion of floating-point values into integers by multiplying them with the precision factor, an additional step is required to rescale each dimension separately. This rescaling process aims to transform the integer values to a standardized range from 0 to the new maximum range. Without this rescaling, the integer values would retain their original magnitude, potentially resulting in a wide range of values that could adversely affect compression performance.

Each dimension needs to be rescaled independently as the original point cloud may have different ranges of values along different axes, which is often the case. For instance, in a three-dimensional point cloud, the X, Y, and Z coordinates may span different ranges.

The purpose of reducing the range of values is twofold. First, it helps to normalize the data, making it more amenable to various compression techniques that are sensitive to the magnitude of values. Secondly, reducing the range of values helps to mitigate potential numerical issues that may arise during the compression process. Large integer values can often lead to overflow problems when performing arithmetic operations.

Once the point cloud has been transformed into integer values, it can be voxelized using a voxel size of 1. This approach also allows us to encode in the near-lossless mode by modifying the scaling factor, although this is not further explored in this thesis.

Furthermore, it is worth noting that some point clouds may contain repeated geometric values. In such cases, only one instance of each unique geometric value is retained. Finally, the original min range along with the number of digits of precision is saved, which is then added to the header of the encoded stream to allow for seamless reconstruction of the original point cloud after decompression.

4.3 Performance Improvements

FRL's original code was implemented in C. A deliberate decision was made to undertake a comprehensive rewrite of the codebase to Modern C++. This has resulted in a multitude of benefits and advantages.

One of the key benefits of migrating to Modern C++ was the embrace of the OOPS. By leveraging classes, encapsulation, and inheritance, the rewritten code is much more structural and modular.

Another noteworthy benefit was the integration of stronger type-checking features of Modern C++. The introduction of static typing and compile-time type checking provides increased safety and reliability to the code. This helped us catch potential errors and inconsistencies at compile time, minimizing the risk of runtime errors and improving overall code quality. This along with the utilization of modern language features and syntax enhancements has significantly enhanced code readability. Concepts such as lambda functions, range-based loops, smart pointers, and modern template metaprogramming techniques were used. One of the advantages of smart pointers is, that they ensure due to RAII (Resource acquisition is initialization), that the actual object is deleted. When using a raw pointer, we need to have a delete for every possible exit point, and still, an exception will lead to a memory leak. Smart pointers will also free the memory if an exception occurs. The original C code relied heavily on manual pointers and switching to smart pointers in the C++ implementation helped avoid many possible issues.

The adoption of Modern C++ also allowed us to take advantage of the enhanced standard library. With a wide range of robust and efficient data structures, algorithms, and utilities, the rewritten code benefits from improved performance and productivity. Overall, the decision to rewrite the existing codebase from C to Modern C++ has proven to be highly beneficial.

After the code rewrites, dedicated efforts were undertaken to improve the performance of the encoder further. It is important to note that FRL already had the fastest encoder among all the state-of-the-art methods, implying even a small performance improvement in this phase would widen the gap even further compared to the other techniques.

To be able to improve the various components of the encoding pipeline separately the total encoding time was divided into several distinct parts. These smaller parts try to accomplish very different tasks. The following are the key parts,

- Load time: Load time refers to the duration taken to read the initial ply (point cloud) file. This phase involves accessing the input point cloud data from storage and converting it into a format internally that's usable for further processing of the point cloud. The load time can vary depending on factors such as file size, file format (ascii or binary point clouds) and the efficiency of the file reading mechanism. Efficient handling of the load time is crucial to ensure a smooth transition from file input to subsequent

processing stages.

- Sort time: Sort time represents the time required to sort the ply file based on the slice axis ordering. This ordering allows us to organize the point cloud data in a specific manner that facilitates the generation of two-dimensional (2D) slices. These slices serve as the foundation for subsequent compression operations. Sorting the point cloud data along a chosen axis enables efficient access and processing of the data during the encoding process. The sort time directly impacts the performance and speed of generating the necessary 2D slices.
- Encoding time: The encoding time corresponds to the actual duration dedicated to the encoding process. This phase involves the application of the FRL technique along with the arithmetic encoder to generate the final compressed bitstream and to save this bitstream. The encoding time plays a crucial role in determining the overall compression speed achieved by the technique.

This breakdown into various components provided valuable insights into the overall efficiency of the encoding process. It also helped us to identify potential bottlenecks and address them specifically to enhance the overall speed and effectiveness of FRL.

The ply file loader, the one directly contributing to the loading time, underwent a complete overhaul, incorporating several enhancements aimed at accelerating the file reading process. To expedite the parsing speed, a significant improvement was achieved by adopting the approach of pre-loading the entire file into memory and encapsulating it within a stream. This technique proved to be advantageous, particularly for files that are smaller than 1GB in size. By loading the entire file upfront, the parser gained the advantage of continuous and efficient access to the data eliminating the need for repeated disk access during parsing, resulting in faster parsing times.

Additionally, another optimization was introduced by providing a list size hint. This feature allows the parser to anticipate the size of lists within the file, such as the points, and allocate appropriate memory in advance. By preallocating memory based on this hint, unnecessary reallocations and memory resizing operations are avoided, leading to improved efficiency during the parsing process.

Furthermore, the revamped ply file loader prioritized the direct conversion of read data into a usable format for further processing. This approach eliminated the need for intermediate data representations, such as temporary structures or additional conversions. By directly converting the read data into a suitable format, the loader reduced memory consumption and processing overhead, resulting in streamlined file reading and enhanced performance.

To further enhance the speed of the point cloud sorting process, particular attention was given to implementing an optimized sorting algorithm for reordering the point cloud data for the required axis ordering. A custom sorting implementation leveraging the flexibility of custom comparator functors in C++. By defining custom comparator functors, the sorting algorithm could be tailored to the specific requirements of the point cloud data. These functors allowed for precise control over the comparison logic used during the sorting process.

During the implementation, it was observed that the order of comparisons within the comparator functors had a noticeable impact on the sorting speed. By strategically arranging the comparisons in the functor, a moderate speed improvement was achieved. This optimization was achieved by considering the likelihood of early termination based on

the comparison order.

4.4 Extensions

Once the above performance improvements to FRL were completed the next focus was on attempting to improve the compression ratio i.e. lower the bitrates. For this, two new techniques were investigated. The first one being an improved axis ordering pipeline and the second being Supervoxelization. We explore them in more detail in this section.

4.4.1 Axis ordering

In the original FRL technique to construct the runs, the voxel space is initially structured into 2D planes, referred to as slices, which are oriented perpendicular to a predefined axis. It is important to note that these slices are not rendered as matrices and do not represent a form of point cloud rendering in the proposed method. For the sake of simplicity, the paper assumes that the z-axis has been chosen as the predefined axis. As a result, the 2D coordinates of any voxel within a slice can be conveniently described using the x and y axes. In this subsection, we delve deeper into this ordering.

In the original FRL paper, the author used an empirically derived axis ordering strategy, which turned out to be "z,x,y" for most of the point clouds. This approach relied on prior knowledge and specific properties of the data to determine an optimal ordering that would result in efficient compression. However, in scenarios where complete information about the data properties or parameters is not available, pre-computing the ordering becomes impractical, and employing arbitrary orderings often leads to suboptimal compression ratios.

To address this limitation, a new automatic axis ordering pipeline has been developed. This pipeline leverages an analysis of the properties exhibited by the point clouds when reading them i.e. on the fly to determine an effective ordering scheme. Through extensive examination, it was observed that FRL achieves the best results when the axis is ordered in descending order relative to the number of occupied voxels.

To illustrate this concept, let's consider a point cloud with Occupied voxels x = 500, Occupied voxels y = 300, and Occupied voxels z = 100 (note we've already seen what these metrics mean in the statistics section 4.1. In this case, ordering the axes in the x, y, and z sequence i.e in the decreasing order of Occupied voxels yields the most competitive bitrates. By organizing the axes in this manner, the resulting compression process can effectively exploit the distribution and occupancy patterns within the point cloud data, leading to improved compression performance.

The introduction of this new automatic axis ordering pipeline allows FRL to adapt to diverse point cloud datasets without requiring prior knowledge or manual intervention to determine the optimal ordering. This approach enhances the versatility and applicability of FRL to various scenarios, ensuring that it can deliver efficient compression even in situations where specific details about the data are unknown or subject to variation.

4.4.2 Supervoxels

A novel approach was explored, which involved grouping the original voxels into supervoxels and performing compression in the supervoxel domain. In this technique, 2D supervoxels

of size NxN were utilized, and a supervoxel was considered occupied if at least one of its constituent voxels was occupied. In the context of traditional 3D segmentation, supervoxels typically refer to groups of voxels that exhibit similarity in both data attributes and spatial proximity. They are intended to be connected and localized entities, distinct from clusters. However, the supervoxels discussed in this context differ from the traditional concept. Here, the supervoxels employed are of uniform dimensions, deviating from the notion of variable-sized supervoxels in traditional segmentation methods.

However, solely considering occupied supervoxel information does not provide sufficient data for lossless reconstruction of the original voxel locations during decompression. To address this, two additional data files, namely "points" and "counts," were introduced. The "counts" file encoded the number of occupied voxels within each supervoxel. For instance, if a supervoxel of size 2x2 contained 4 voxels and 2 of them were occupied, the "counts" value would be set to 2. On the other hand, the "points" file stored the grid coordinates of these 2 occupied voxels within the supervoxel. By employing this supervoxeltechnique, the resulting point cloud comprised solely of supervoxels exhibited significantly higher density compared to the initial point cloud, leading to a substantially reduced bitrate. The counts and points files could be encoded with other compression techniques and appended to the compressed bitstream, enabling lossless reconstruction of the original point cloud during decoding.

After initially considering 2D supervoxels, the technique was further expanded into 3D supervoxels, introducing an additional dimension to capture spatial coherence within the data. In contrast to the 2D supervoxelization technique, which utilizes squares in a 2D space, the 3D supervoxelization technique employs 3D cubes as the basic units for grouping voxels. One of the main reasons was to make the sparse datasets denser to aid in better compression with FRL. By extending the supervoxelization process to three dimensions, the 3D supervoxelization technique aims to exploit the volumetric structure and spatial coherence present in the data. Instead of considering only neighbouring voxels in a 2D plane, this technique allows for the aggregation of neighbouring voxels across the entire volume.

4.5 LiDAR Support

LiDAR point cloud data exhibits distinct characteristics compared to regular point cloud data, necessitating specific modifications in compression techniques to support LiDAR point cloud compression. This section discusses the changes implemented in FRL to accommodate LiDAR point cloud data compression.

Despite preprocessing the point clouds to reduce their range and represent all points as integers, applying FRL directly to the preprocessed LiDAR point clouds did not yield satisfactory results. This is attributed to the inherent nature of LiDAR sensors, which provide increased precision compared to regular point clouds. Consequently, representing these higher precision point clouds requires a significantly larger bit depth. Additionally, LiDAR-generated point clouds tend to be extremely sparse.

In some cases, the preprocessed LiDAR point clouds necessitate up to 31 bits per axis, resulting in a total of 93 bits for a 3D point representation. This represents a substantial increase compared to the previous dataset, where the maximum bit depth was 14 bits per axis or 42 bits for a 3D point. The considerable increase in bit depth posed significant challenges for the existing codebase, which was not originally designed to handle such

large bit depths. As a result, overflow errors and other issues arose during the encoding process.

Detecting and addressing errors in the codebase during the debugging process proved to be extremely challenging due to the complex nature of the issues and the inability to manually identify and track them effectively. To aid in the debugging efforts, various additional tools were evaluated. Among them, Valgrind and Google sanitizers emerged as the most popular options. Ultimately, the decision was made to utilize Google sanitizers due to their minimal performance impact compared to Valgrind, which tended to significantly slow down the debug process.

Google sanitizers comprise a comprehensive set of dynamic code analysis tools designed to detect common bugs and issues in the code. These tools include:

- Thread Sanitizer: This tool detects data races, thread leaks, and deadlocks, helping identify potential concurrency issues.
- Address Sanitizer: Address Sanitizer is employed to identify buffer overflows and dangling pointer dereferences, which are common sources of memory corruption bugs.
- Leak Sanitizer: This tool, which is part of Address Sanitizer, is used to detect memory leaks, ensuring efficient memory management.
- Undefined Behavior Sanitizer: By detecting integer overflows and floating-point overflow issues, this sanitizer helps identify undefined behavior scenarios in the code.
- Memory Sanitizer: Memory Sanitizer is employed to identify uninitialized memory reads, helping catch potential bugs related to accessing uninitialized memory regions.

By utilizing Google address sanitizers, a total of 14 bugs were identified and addressed within the FRL algorithm. These bugs ranged from issues related to handling 31x31-bit number multiplications to various other fixes required to enhance the overall functionality and reliability of the algorithm. Each bug was thoroughly fixed and verified to ensure the correctness and stability of the modified FRL implementation.

4.5.1 Context-adaptive binary arithmetic coding (CABAC)

Context-adaptive binary arithmetic coding (CABAC) is a technique used in lossless data compression to encode binary symbols based on their contextual information. It is a form of entropy coding that utilizes the statistics of previously encoded symbols to adaptively adjust the encoding process. The core idea behind CABAC is to exploit the dependencies and correlations between neighbouring symbols to improve compression efficiency. Instead of encoding each symbol independently, CABAC takes into account the surrounding context (in our case 3D context) and adapts its encoding scheme accordingly [62].

The encoding process in CABAC involves maintaining a set of probability models, which estimate the likelihood of a particular symbol based on the observed context. These probability models are initially set to default values but are updated and refined as the encoding progresses. The updating mechanism allows CABAC to learn and adapt to the specific characteristics and patterns present in the data.

During encoding, CABAC divides the data stream into smaller units, such as slices, and assigns a probability model to each unit based on the current context. The probability model is used to determine the range or interval in which the symbol will be encoded. The

size of the interval reflects the probability assigned to the symbol. A more probable symbol will have a larger range, while a less probable symbol will have a smaller range.

To perform the actual encoding, CABAC employs binary arithmetic coding techniques. The binary arithmetic coder maintains a fractional interval on the number line, which is divided into subintervals corresponding to the probability ranges of the symbols. The encoder then progressively narrows down the interval based on the observed symbols, until a unique subinterval representing the encoded symbol is identified. This subinterval is output as the compressed representation of the symbol. Decoding in CABAC follows a similar process but in reverse.

After implementing the necessary fixes from the previous section, the subsequent step was to utilize Context-Adaptive Binary Arithmetic Coding (CABAC) for encoding the bitstream. However, the original CABAC implementation used in FRL encountered scaling issues when applied. To overcome these challenges and accommodate the broader range of input values found in LiDAR datasets, a comprehensive examination of arithmetic encoding and the associated scaling issues was undertaken. This analysis resulted in the identification of necessary modifications to properly handle the expanded range of input values from the LiDAR datasets. In the following discussion, we will provide a concise overview of the encoding process in an arithmetic encoder and delve into the encountered scaling issues, as well as the corresponding solutions that were implemented.

The encoder comprises a function and static variables that store the current state:

- The variable $mLow$ holds the current lower bound and is initially set to 0.
- The variable $mHigh$ holds the current upper bound and is initialized with $0x7FFFFFFF$, which is the maximum value that can be represented in 31 bits.
- The variable $mStep$ represents a step size used in the encoding process. It may not be static in the encoder, but the decoder relies on this property. To prevent overflows, only 31 of the 32 bits are used. Further details on this will be discussed later.

The function is declared as follows:

```
void Encoder(unsigned int low_count, unsigned int high_count, unsigned int total);
```

The encoder receives the cumulative probabilities of the current symbol a_i and the next lower symbol a_{i1}, which are calculated by the model. Based on these probabilities, the encoder computes new upper and lower bounds. Initially, the interval from $mLow$ to $mHigh$ is divided into a total number of steps, resulting in a step size:

```
mStep = (mHigh - mLow + 1) / total;
```

The addition of 1 to the difference between $mHigh$ and $mLow$ is necessary because $mHigh$ represents the open upper bound. In analogy to the decimal system, it is similar to an interval from 0 to 99.9, where the upper bound is stored as 99. The inclusion of the fractional places makes the interval larger by 1 compared to 99−0 = 99.

The limitation to 31 bits is due to the initialization of $mHigh$ with the maximum possible value. If 32 bits were used, the calculation $mHigh - mLow + 1$ could result in an overflow, leading to exceptions or encoding errors that may corrupt the file.

The update of the upper bound is performed as follows:

```
mHigh = mLow + mStep * high_count - 1;
```

And the lower bound is updated as:

```
mLow = mLow + mStep * low_count;
```

Both calculations depend on the previous value of $mLow$, so overwriting it should be the last step. As the interval is open-ended, the value of $mHigh$ needs to be decreased by one to properly reflect this condition.

When the presented encoding methods are applied to multiple symbols, a new challenge arises: the values of $mLow$ and $mHigh$ progressively converge until they coincide, rendering further encoding impossible. However, a straightforward solution can be derived from the following observation: In the event that $mLow$ and $mHigh$ reside within the same half of the number range (either < 0x40000000 or ≥ 0x40000000), it is guaranteed that they will never exit this range due to subsequent symbols shrinking the interval. Hence, the information about the half becomes irrelevant for subsequent steps and can be stored and disregarded.

Based on the provided implementation, when the most significant bits (MSB) of $mLow$ and $mHigh$ match, we can store them in the output sequence and shift them out, employing a technique known as E1- or E2-scaling:

```
while ((mHigh < g_Half) || (mLow >= g_Half)) {
    if (mHigh < g_Half) // E1
    {
        SetBit(0);
        mLow = mLow * 2;
        mHigh = mHigh * 2 + 1;
    }
    else if (mLow >= g_Half) // E2
    {
        SetBit(1);
        mLow = 2 * (mLow - g_Half);
        mHigh = 2 * (mHigh - g_Half) + 1;
    }
}
```

Here, g_Half represents the global constant 0x40000000, denoting the center of the range. Multiplying by 2 expands the interval, while adding 1 establishes the upper bound as an open interval is being dealt with. This is equivalent to the more intuitive approach of adding 1 before and subtracting 1 after calculations involving an open bound. The function $SetBit()$ appends a bit to the output sequence, while its complementary function in the decoder is referred to as $GetBit()$. Both functions operate sequentially, functioning as a first-in, first-out (FIFO) queue. Seeking within the encoded sequence is neither feasible nor necessary since the algorithm itself operates sequentially.

E3 scaling, introduced alongside E1 and E2 scaling, enhances the encoding process by addressing a specific scenario where E1 and E2 scaling alone falls short. When $mLow$ and $mHigh$ converge towards the center of the interval, remaining within their respective halves, the interval becomes too small to enable further encoding. This situation is exemplified by the extreme case of $mLow$ reaching 0x3FFFFFFF and $mHigh$ reaching 0x40000000, where

they differ in every bit except for the overflow bit, yet further encoding becomes impossible.

To overcome this limitation, E3 scaling comes into play. Once $mLow$ exits the lowest quarter (with the maximum value of the first quarter being $g_FirstQuarter$), and $mHigh$ exits the highest (fourth) quarter (with the maximum value of the third quarter being $g_ThirdQuarter$), the total range becomes less than half of the original range. This condition remains unchanged due to the ongoing shrinking process. While it is initially uncertain which half will contain the result, the subsequent E1 or E2 scaling operation reveals the values that could have been stored earlier if such foresight were possible. This may seem peculiar, but it is the principle behind E3 scaling: the interval is enlarged similarly to E1 or E2 scaling, but instead of storing a bit in the output sequence, the fact that an E3 scaling occurred is remembered using the auxiliary variable $mScale$:

```
while ((g_FirstQuarter <= mLow) && (mHigh < g_ThirdQuarter)) {
mScale++;
mLow = 2 * (mLow - g_FirstQuarter);
mHigh = 2 * (mHigh - g_FirstQuarter) + 1;
}
```

During the subsequent E1 or E2 scaling, the correct bit for each E3 scaling is added to the output sequence. Employing E3 scaling followed by E1 scaling implies that the interval could have fit within the range between $g_FirstQuarter$ and g_Half. This is analogous to performing E1 scaling followed by E2 scaling. Similarly, the E3-E2 sequence can be interpreted, and the same principle applies to iterated E3 scalings. As a result, E3 scalings are stored after the next E1/E2 scaling, utilizing the inverse bit of that scaling:

```
while ((mHigh < g_Half) || (mLow >= g_Half)) {
if (mHigh < g_Half) // E1
{
SetBit(0);
mLow = mLow * 2;
mHigh = mHigh * 2 + 1;
for (; mScale > 0; mScale--)// E3
SetBit(1);
}
else if (mLow >= g_Half) // E2
{
SetBit(1);
mLow = 2 * (mLow - g_Half);
mHigh = 2 * (mHigh - g_Half) + 1;
// E3
for (; mScale > 0; mScale--)
SetBit(0);
}
}
```

A substantial portion of this reference implementation draws heavily from [63]. The CABAC utilized in FRL is built upon this reference arithmetic encoder implementation, incorporating context adaptive probability estimation and additional functionalities. However, the scaling techniques implemented in FRL's CABAC were not originally designed to handle the exceedingly large LiDAR data. Therefore, considerable effort was dedicated to enhancing

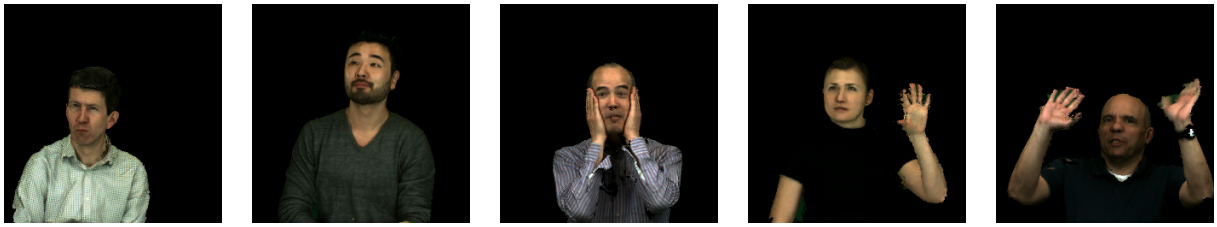
the encoder's upper limit to 64 bits and ensuring support for all existing features, while maintaining proper scaling without compromising performance. As a result, a new and enhanced CABAC was developed, capable of effectively handling a wide range of point clouds.

5 Experimental Results

5.1 Datasets

The original point cloud datasets used for conducting the experiments were sourced from the MPEG PCC Content Database, which comprises various datasets combined to establish a standard for point clouds. Let's take a comprehensive look at each of the datasets used:

Microsoft Voxelized Upper Bodies (MVUB): The MVUB dataset consists of five subjects named Andrew, David, Phil, Ricardo, and Sara. The upper bodies of these subjects were captured using four frontal RGBD cameras at a frame rate of 30 frames per second. Each subject's capture lasted between 7 to 10 seconds. The dataset provides two spatial resolutions for each sequence: a 512x512x512 voxel cube (depth 9) and a 1024x1024x1024 voxel cube (depth 10). Only the voxels near the surface of the subjects are occupied, and the voxel attributes include the red, green, and blue components of the surface color. Voxels at depth 9 have an approximate size of 1.5mm cubed, while voxels at depth 10 are approximately 0.75mm cubed. Figure 28 shows a visualization of these point clouds. This dataset can be accessed at [64].



(a) Andrew9 **(b)** David9 **(c)** Phil9 **(d)** Sarah9 **(e)** Rucardo9

Figure 28. Point clouds in MVUB dataset.

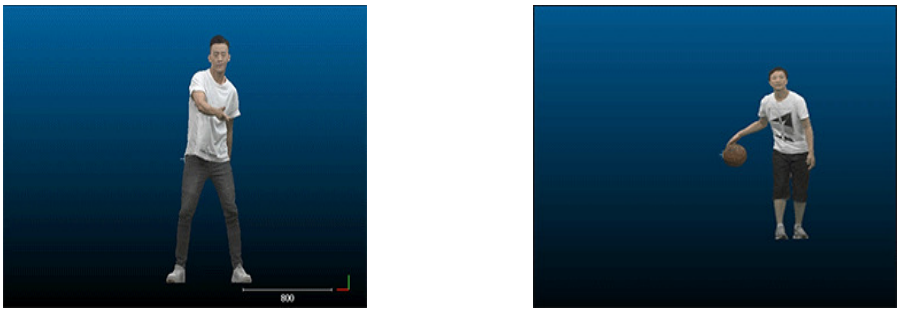
8i Voxelized Full Bodies (8iVFB v2): The 8iVFB v2 dataset includes four sequences: longdress, loot, redandblack, and soldier. Each sequence captures the full body of a human subject using 42 RGB cameras arranged in 14 clusters. The capture duration for each sequence is 10 seconds, with a frame rate of 30 frames per second. The dataset provides a single spatial resolution of a 1024x1024x1024 voxel cube (depth 10). Only the voxels near the surface of the subjects are occupied, and the voxel attributes include the red, green, and blue components of the surface colour. The size of a voxel at depth 10 corresponds to approximately 1.75mm per voxel on a side, based on the height of the subject. Figure 29 shows a visualization of these point clouds. This dataset can be accessed at [65].

(a) Longdress **(b)** Loot **(c)** Red and black **(d)** Soldier

Figure 29. Point clouds in 8iVFB dataset.

Owlii Dynamic Human Textured Mesh Sequence : The Owlii dataset comprises four sequences: basketball, dancer, exercise, and model. Each sequence has a frame rate of 30 frames per second and spans a duration of 20 seconds. The sequences consist of around 40,000

triangles, and texture maps are provided at a resolution of 2048x2048. Figure 30 shows a visualization of these point clouds. This dataset can be accessed at [66].



(a) Dancer **(b)** Basketball

Figure 30. Point clouds in Owlii dataset.

Private MPEG Dataset : MPEG also has another research dataset (not publically available) consisting of two sequences: Egyptian and Shiva both with bit depths of 12.Figure 31 shows a visualization of these point clouds. This dataset is available by request from the MPEG group.

(a) Shiva **(b)** Egyptian

Figure 31. Point clouds in Private dataset.

The datasets obtained from the MPEG PCC Content Database were carefully chosen due to their diverse characteristics, making them an invaluable benchmark for evaluating and establishing standards in the field of point clouds. The intention was to select datasets that were representative of the most commonly encountered point cloud data, with the aim of achieving good compression ratios. The expectation was that achieving good compression results on these datasets would also translate to effective compression in real-world scenarios.

However, it's important to note that the MPEG dataset exclusively consists of regular point clouds and does not include any LiDAR data. Recognizing the need to assess the performance of compression algorithms specifically for LiDAR data, an effort was made to incorporate a comprehensive set of LiDAR datasets into our test suite. These additional datasets were selected to provide a representative sample of LiDAR data and ensure a comprehensive evaluation. The newly added LiDAR datasets in our test suite are as follows:

S3DIS (Stanford 3D Indoor Scene Dataset) : This dataset comprises six large-scale indoor areas, encompassing a total of 271 rooms. Each point in the scene point cloud is annotated with one of thirteen semantic categories, allowing for detailed analysis and evaluation

of compression techniques for indoor LiDAR data. The dataset is preprocessed using our method and preprocessed data is in a spatial resolution of up to a 16384x16384x16384 voxel cube (depth 14). Figure 33 shows the data processing steps. The RGB output of the scanner is registered on the 3D modalities (each yellow marker represents one scan). These are then processed from the RGB and 3D data to make depth, surface normals, and 2D semantic (projected from 3D semantics) images for each scan. The processed equirectangular images are sampled to make new regular images. This dataset can be accessed at [67].

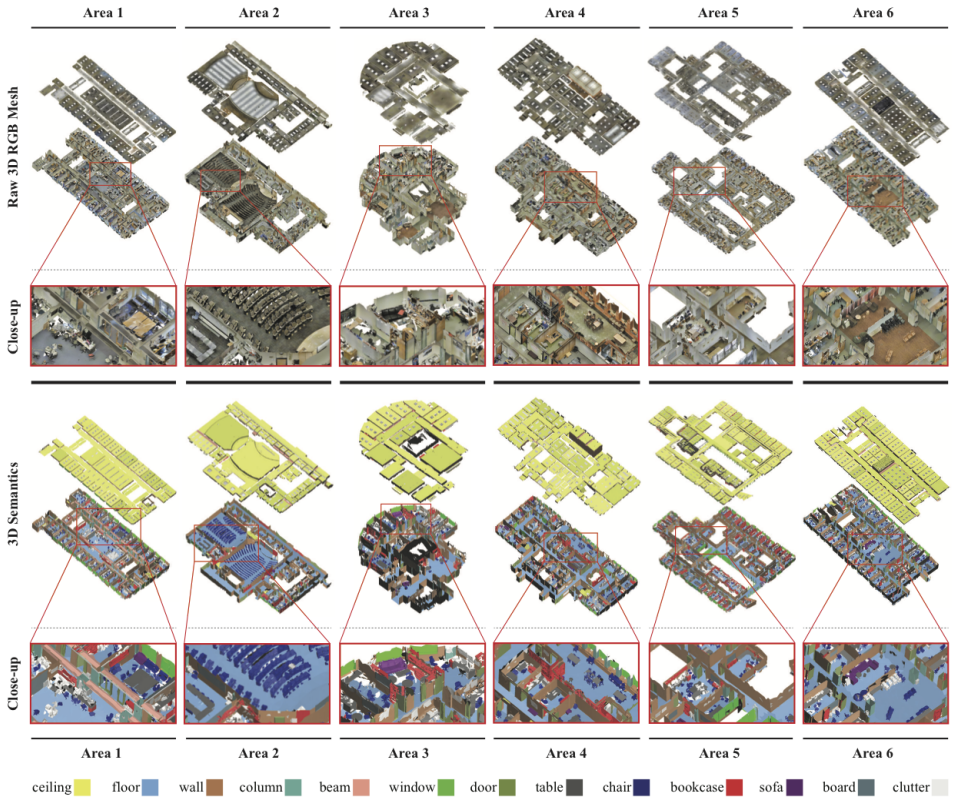


Figure 32. 3D Modalities S3DIS Dataset [67]

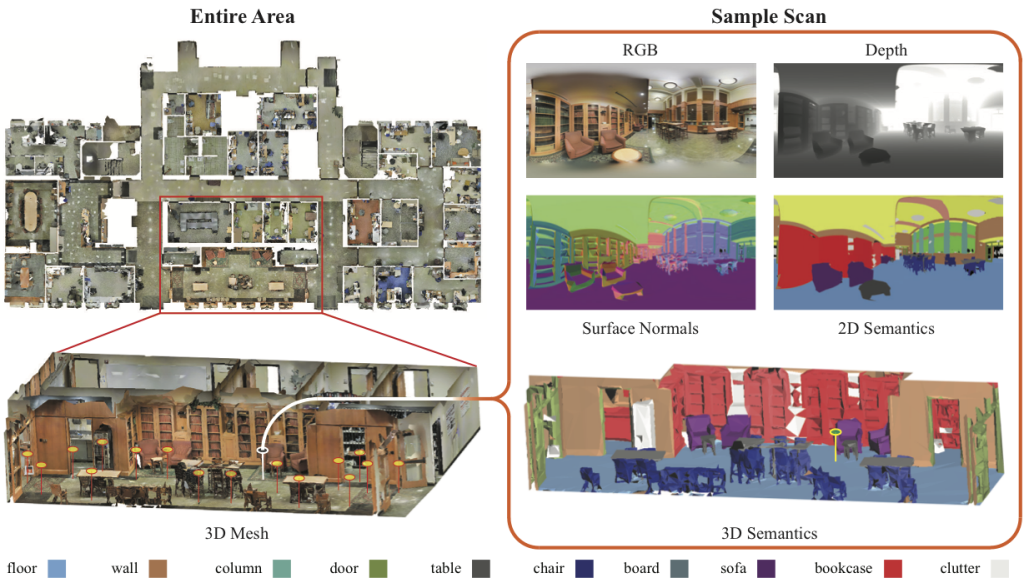


Figure 33. Data Processing of S3DIS Dataset [67]

Velodyne : A subset of the KITTI dataset specifically focused on autonomous vehicles. The Velodyne dataset contains LiDAR data captured using Velodyne sensors, offering the opportunity to evaluate compression performance in the context of autonomous driving scenarios. The dataset is preprocessed using our method and preprocessed data is in a spatial resolution of a 262144x262144x262144 voxel cube (depth 18). Figure 34 shows a visualization of these point clouds. This dataset can be accessed at [68].

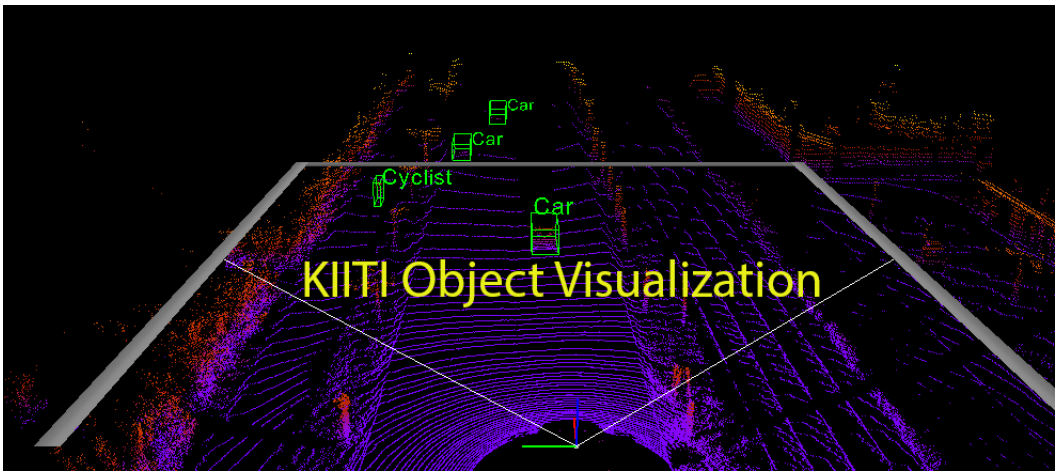


Figure 34. Velodyne Dataset [69]

Shapenet : This dataset combines the Trimble 3D Warehouse and Yobi3D datasets, providing a collection of 3D CAD models representing sixteen different types of objects. The inclusion of Shapenet enables the evaluation of compression algorithms on LiDAR data representing various objects and structures. The dataset is preprocessed using our method and preprocessed data is in a spatial resolution of a 131072x131072x131072 voxel cube (depth 17). Figure 35 shows a visualization of these point clouds. This dataset can be accessed at [70].

Figure 35. Shapenet Dataset [70]

By including these new LiDAR datasets in our test suite, we aim to enhance the

comprehensiveness of our evaluations and provide insights into the performance of compression algorithms on a wider range of LiDAR data. This expansion allows us to explore the challenges and opportunities specific to LiDAR data compression, further advancing the field and facilitating more efficient handling of LiDAR point clouds. Table 1 provides an overview of the different point cloud sequences,the average number of voxels in each sequence which is the same as the average number of points in every point cloud sequence and the percentage of occupied voxels i.e. the (average number of voxels) / (overall spatial resolution of the voxel cube).

File Name	Avg. no. of voxels	% of occupied voxels
andrew9	284255	2.12E-03
david9	349173	2.60E-03
phil9	333609	2.49E-03
ricardo9	226226	1.69E-03
sarah9	259690	1.93E-03
MVUB : Average	**290590**	**2.17E-03**
longdress	834315	7.77E-04
loot	793821	7.39E-04
redandblack	727071	6.77E-04
soldier	1075299	1.00E-03
8iVFB v2: Average	**857627**	**7.99E-04**
basketball	2925514	3.41E-04
dancer	2592758	3.02E-04
Owlii : Average	**2759136**	**3.21E-04**
Egyptian	272684	3.97E-06
Shiva	1009132	1.47E-05
Research : Average	**640908**	**9.33E-06**
S3DIS	131866	3.00E-08
Shapenet	2583	1.15E-12
Velodyne	127886	7.10E-12
LiDAR: Average	**87445**	**1.00E-08**

Table 1. Datasets

5.2 Results

In this section, we present the comprehensive results of the performance improvements and extensions made to the original FRL algorithm. The enhancements and optimizations implemented in the algorithm are evaluated and analyzed to assess their impact on various aspects of the system's performance.

Furthermore, the results are accompanied by a thorough discussion in the subsequent section (5.3), where we delve into the implications and significance of the findings. We analyze the observed performance improvements, identify any limitations or trade-offs introduced by the modifications, and explore potential areas for further enhancements.

In order to evaluate the performance of the original FRL algorithm in terms of compression speed, a comprehensive benchmarking process was conducted on all the datasets. The focus of this benchmarking analysis was to measure the speed of the different stages of the FRL compression pipeline.

File Name	Load	Sort	Encoding	Total encoding	Decoding
andrew9	0.06826	0.02810	0.09262	0.18899	0.24676
david9	0.08306	0.03373	0.11332	0.23012	0.41599
phil9	0.08287	0.03336	0.11171	0.22794	0.36539
ricardo9	0.05837	0.02360	0.07900	0.16097	0.20569
sarah9	0.06216	0.02567	0.08603	0.17386	0.20975
Average	**0.07095**	**0.02889**	**0.09654**	**0.19637**	**0.28872**
longdress	0.20036	0.08313	0.25269	0.53617	2.19099
loot	0.19467	0.07938	0.24615	0.52020	2.34351
redandblack	0.18198	0.07520	0.23478	0.49196	2.07407
soldier	0.25408	0.10819	0.34095	0.70323	3.01562
Average	**0.20777**	**0.08648**	**0.26864**	**0.56289**	**2.40605**
basketball	0.29867	0.29359	0.88169	1.47395	13.30540
dancer	0.27015	0.25160	0.89233	1.41408	14.90080
Average	**0.28441**	**0.27260**	**0.88701**	**1.44401**	**14.10310**
Egyptian	0.15448	0.03166	0.22041	0.40654	190.14400
Shiva	0.51955	0.12856	0.72823	1.37634	95.26360
Average	**0.33701**	**0.08011**	**0.47432**	**0.89144**	**142.70380**
Global Average	**0.22504**	**0.11702**	**0.43163**	**0.77368**	**39.87542**
Total (sum)	**2.42866**	**1.19577**	**4.27990**	**7.90434**	**324.68158**

Table 2. FRL C implementation time(s) for different parts

The original FRL algorithm was applied to each dataset, and the time taken to complete each stage was recorded. In order to achieve reliable and consistent results, this process was repeated 100 times for each dataset. By conducting multiple runs, we aimed to capture the variations in execution time that may arise due to different factors such as system load, data characteristics, and algorithmic complexity. This allowed us to account for any outliers or anomalous results and ensure that the average values obtained were representative of the algorithm's performance. This data is presented in Table 2

After obtaining the baseline performance using the original FRL algorithm, the migration to C++ was undertaken to further improve the performance. The performance improvements discussed in Section 4.3 were implemented during this migration process. These

File Name	Load	Sort	Encoding	Total encoding	Decoding
andrew9	0.02155	0.01988	0.08998	0.13141	0.24636
david9	0.02767	0.02503	0.10998	0.16268	0.41565
phil9	0.02653	0.02417	0.10561	0.15631	0.35478
ricardo9	0.01840	0.01533	0.07351	0.10724	0.19446
sarah9	0.01960	0.01768	0.08396	0.12124	0.20885
Average	**0.02275**	**0.02042**	**0.09261**	**0.13578**	**0.28402**
Speedup %	**67.92909**	**29.33428**	**4.06915**	**30.85760**	**1.62697**
longdress	0.06630	0.06404	0.24485	0.37519	2.19182
loot	0.06165	0.06045	0.23440	0.35650	2.30292
redandblack	0.05715	0.05559	0.22101	0.33375	2.02556
soldier	0.08193	0.08233	0.33387	0.49813	3.01785
Average	**0.06676**	**0.06560**	**0.25853**	**0.39089**	**2.38454**
Speedup %	**67.86890**	**24.14071**	**3.76212**	**30.55578**	**0.89408**
basketball	0.08651	0.20530	0.87660	1.16841	13.34470
dancer	0.09280	0.18706	0.90210	1.18195	14.82550
Average	**0.08966**	**0.19618**	**0.88935**	**1.17518**	**14.08510**
Speedup %	**68.47679**	**28.03225**	**-0.26353**	**18.61697**	**0.12763**
Egyptian	0.03878	0.01913	0.20409	0.26201	178.65400
Shiva	0.13887	0.07851	0.73031	0.94769	95.54180
Average	**0.08882**	**0.04882**	**0.46720**	**0.60485**	**137.09790**
Speedup %	**73.64393**	**39.05328**	**1.50047**	**32.14928**	**3.92835**
Global Average	**0.06700**	**0.08276**	**0.42692**	**0.57668**	**38.46289**
Speedup %	**70.22789**	**29.27985**	**1.08973**	**25.46323**	**3.54235**
Total (sum)	**0.73776**	**0.85449**	**4.21027**	**5.80253**	**313.32424**
Speedup %	**69.62281**	**28.54049**	**1.62682**	**26.59059**	**3.49799**

Table 3. FRL C++ implementation time(s) for different parts

improvements aimed to optimize the algorithm and enhance its efficiency. The performance improvements in the point cloud compression system can be primarily attributed to two key factors: enhancements in the point cloud loading pipeline and improvements in the sorting algorithm for reordering the point cloud based on the required axis ordering.

Table 3 presents the new time results achieved after applying these performance improvements. The benchmarking process followed the same methodology as the original FRL technique, with the encoding and decoding stages recorded and averaged over 100 runs. It shows the improvements over the original implementation in percentages.

We now show the results from the extensions discussed in section 4.4 using the axis ordering technique. The axis ordering technique is employed to optimize the performance of the FRL implementation further in terms of final bitrate. By carefully arranging the order in which the data is accessed and processed, the axis ordering technique aims to maximise the compression performance. Figure 36 shows the compression rates in bits per occupied voxel (bpov) as reported by FRL in the different axis orders used along with the average occupancy of voxels across the 3 dimensions in a subset of the data.

	(X,Y,Z)	(X,Z,Y)	(Y,X,Z)	(Y,Z,X)	(Z,X,Y)	(Z,Y,X)	Ocup. voxels X	Ocup. voxels Y	Ocup. voxels Z
Wall_1	15.17	16.11	13.49	13.53	15.87	14.62	8.443	308.524	12.003
Wall_2	13.73	15.04	11.99	11.57	14.17	12.29	22.832	1742.947	53.275
Floor_1	12.74	12.24	14.26	14.90	13.44	14.78	832.491	11.668	16.889
Chair_1	16.15	16.19	16.28	16.20	16.83	16.88	19.270	18.987	15.176
Chair_2	15.77	15.63	16.36	16.11	16.43	16.29	14.374	13.979	8.591
Ceiling_1	14.90	12.98	14.87	12.56	12.77	12.09	138.762	64.708	9772.232
Board_1	12.91	14.184	11.99	11.57	14.17	12.29	23.287	1489.561	33.732
Column_1	12.91	13.44	13.77	15.38	15.44	16.28	88.513	30.823	10.772
Beam_1	13.29	12.09	15.11	16.12	12.80	14.97	83.476	11.827	32.127
Table_1	15.12	14.56	14.90	13.76	13.45	12.80	8.3504	18.312	139.371
Bookcase_1	14.27	15.04	13.66	15.17	15.59	15.49	14.263	27.137	17.363
Bookcase_2	17.01	16.40	17.13	16.73	16.38	16.44	20.966	9.402	9.129
Clutter_1	14.73	14.28	14.51	14.30	14.18	14.08	8.974	11.213	24.373

Figure 36. Axis ordering.

Name		2	3	4	7	15	31
	FRL	New	New	New	New	New	New
longdress	0.945	3.372	5.215	5.147	7.072		
loot	0.941	3.387	5.233	5.163	7.080		
redandblack	0.989	3.406	5.244	5.170	7.083		
soldier	0.910	3.374	5.226	5.157	7.076		
andrew9	0.962	3.378	5.218	5.149	7.074		
david9	0.929	3.367	5.213	5.146	7.074		
phil10	0.940	3.365	5.207	5.139	7.067		
phil9	0.964	3.378	5.217	5.147	7.072		
ricardo10	0.950	3.371	5.213	5.145	7.071		
ricardo9	0.952	3.375	5.216	5.147	7.073		
sarah9	0.948	3.371	5.213	5.145	7.071		
Egyptian	18.064	19.057	19.878	18.842	18.298	16.555	15.634
Shiva	15.779	16.415	16.662	15.337	14.817	13.733	13.671
basketball	8.316	9.870	10.925	10.232	10.938	11.377	12.340
dancer	10.801	12.051	12.838	11.934	12.231	12.163	12.784

Table 4. Bitrates (bpov) frl existing vs new 2d supervoxels

We now present the results from the second extension to the FRL algorithm, which involves the utilization of supervoxels of different sizes. The purpose of this extension is to explore the impact of supervoxel size on the final bitrates achieved by the compression algorithm. We initially present results for the 2D supervoxels followed by results from 3D supervoxels later. In this experiment, the point cloud data is partitioned into 2D supervoxels, where each supervoxel represents a NxN grid region of voxels within the point cloud. The size of the supervoxels varies, allowing us to investigate the relationship between supervoxel size and the resulting compression performance. The 2D supervoxel sizes range from smaller sizes, such as 3x3 voxels, to larger sizes, such as 31x31 voxels. The reported bitrates also take into account the bitrates necessary to encode the auxiliary files points and counts. Table 4 reports the bitrates obtained using the original FRL algorithm vs the 2D supervoxels of different sizes.

Table 5 shows the results of the benchmark conducted for sparse point clouds using the MPEG Private Dataset, this time applying the 3D supervoxel technique.

Table 6 shows the bitrates on these sparse point clouds with TMC13, original FRL

Name		2	3	4	7	15	31
	FRL	New	New	New	New	New	New
Egyptian	18.064	18.932	20.069	18.188	15.487	14.134	16.251
Shiva	15.779	15.953	16.227	14.293	13.328	13.675	16.148

Table 5. Bitrates (bpov) frl existing vs new 3d supervoxels

implementation and the 3D supervoxel FRL implementation.

Name	TMC13	FRL	FRL_Extensions
Egyptianmask	11.78	18.20	14.13
Shiva	9.68	15.11	13.32
Average	10.73	16	13.72

Table 6. Sparse Point Cloud Bitrates with 3D supervoxels.

Table 7 presents the results of the benchmark comparing four compression methods: Gzip-9, SparsePCGC, FRL, and TMC13. Note : SparsePCGC results are still being produced and might have to be updated. These methods were evaluated using the new LiDAR datasets, which contain point cloud data captured by LiDAR sensors in various environments and scenarios. Gzip-9 is included in the benchmark as a general-purpose compression method widely used in various domains. It serves as a baseline for comparison against the more specialized point cloud compression techniques. The benchmark evaluates the performance of these four compression methods in terms of the achieved bit rates, measured in bits per occupied voxel (Bpov). Lower bit rates indicate higher compression efficiency, as fewer bits are required to represent each occupied voxel in the point cloud.

	Shapenet	Velodyne	S3DIS
GZIP-9	63.01	55.76	28.49
SparsePCGC	34.47	17.97	11.39
FRL	36.7	27.81	14.6
TMC13	33.1	20.51	11.24

Table 7. LiDAR Data Bit Rate (Bpov).

Table 8 shows a subset of the generated statistics for a subset of point clouds. Although all the statistics mentioned in the previous section 4.1 were computed for all the point clouds.

5.3 Analysis of results

In this section, we embark on a detailed analysis of the results presented above. Our objective is to gain a deeper understanding of the performance of the different methods and draw meaningful conclusions from the benchmarking outcomes.

The statistics obtained from the analysis of the three point clouds in Table 8 provide valuable insights into the characteristics of LiDAR data and shed light on potential avenues for improving compression techniques. Let's expand on these points:

- Range Reduction through Preprocessing: The statistics indicate that the preprocessing step, likely involving scaling, effectively reduces the range of values in the LiDAR data.

File name	000217.ply	000571.ply	000775.ply
Range x	158469.00	158896.00	156458.00
Range y	129937.00	152269.00	92296.00
Range z	30562.00	20077.00	24518.00
Bit depth	18.00	18.00	18.00
Prop. empty sl. x	0.74	0.70	0.74
Prop. empty sl. y	0.75	0.69	0.67
Prop. empty sl. z	0.86	0.72	0.82
Ocup. voxels x (mean)	0.81	0.80	0.82
Ocup. voxels y (mean)	0.98	0.84	1.39
Ocup. voxels z (mean)	4.18	6.37	5.21
Bottom hor. x (mean)	64439.90	65453.49	66767.26
Bottom hor. y (mean)	26267.79	15670.75	20238.81
Bottom hor. z (mean)	75127.55	76926.91	75625.22
Bottom vert. x (mean)	26968.32	16132.54	20962.10
Bottom vert. y (mean)	70986.54	72951.58	69731.79
Bottom vert. z (mean)	26329.88	15691.62	20275.79
Top hor. x (mean)	84929.86	84692.65	84382.90
Top hor. y (mean)	26758.41	16095.50	20650.10
Top hor. z (mean)	51755.52	49278.67	55792.24
Top vert. x (mean)	54332.80	47824.76	57258.53
Top vert. y (mean)	94883.73	96711.77	98591.60
Top vert. z (mean)	85104.87	89721.50	80004.17
Entropy x (mean)	0.00	0.00	0.00
Entropy y (mean)	0.00	0.00	0.00
Entropy z (mean)	0.00	0.00	0.00
SR entropy x (mean)	0.00	0.00	0.00
SR entropy y (mean)	0.00	0.00	0.00
SR entropy z (mean)	0.00	0.00	0.00

Table 8. Statistics for a subset of data

This range reduction is beneficial for compression, as it reduces the overall magnitude of the data and potentially enhances compression efficiency.

- Higher Bit Depth Requirement: Despite the range reduction achieved through preprocessing, the LiDAR data still requires a higher bit depth compared to regular data, as evidenced by the consistent value of 18 across the three point clouds. This observation suggests that LiDAR data contains intricate details or requires more precise representation, which limits the compression gains achievable through traditional compression methods.
- Significant Proportion of Empty Slices: Across each dimension, a considerable number of slices within the scaled range are found to be empty. This finding indicates that a substantial portion of the LiDAR point cloud represents empty space. This observation presents an opportunity for improving compression ratios by exploring alternative encoding methods that can represent empty slices more efficiently. By avoiding the use of a fixed 1-bit representation for empty slices and employing more advanced encoding techniques, such as array of binary trees (ABT) compression gains can be achieved.
- Reconsidering Signaling Rectangle Strategies: The analysis suggests that the global signalling rectangle, a technique used in FRL, might not be the most effective approach.

Instead, utilizing signalling rectangles for groups of slices or exploring other adaptive signalling strategies could be more suitable. Further research and experimentation are necessary to test and validate the effectiveness of alternative signalling rectangle techniques in improving compression performance.

- Sparse Data: The "Occupied Voxels (mean)" statistics provide insights into the sparsity of the LiDAR data. In the x or y dimensions, there are fewer than 1 voxel per slice across the range in two out of the three point clouds. The maximum value of 6.37 across the z dimension for the "000571.ply" point cloud also indicates a relatively sparse distribution of points. This sparsity highlights the potential for employing machine learning techniques to enhance the density of the point cloud. By leveraging such techniques to encode the information and make the point cloud denser, followed by applying FRL to the densified point cloud, significant improvements in compression ratios could be achieved.
- Supervoxel Technique Limitations: The analysis suggests that the supervoxel technique might not be viable for LiDAR point clouds. While using supervoxels can potentially make the point cloud denser, it requires the supervoxels to be large, particularly in the case of LiDAR extremely large, leading to a significant increase in additional side information. This increase in side information negates any potential compression gains, making the supervoxel technique impractical for LiDAR point clouds.

Table 3 clearly shows the speedup achieved using the custom modification in percentages over the previous implementation. It provides valuable insights into the speedup achieved by transitioning from the old C implementation to the new C++ implementation as well as the effect of the different optimization techniques attempted for the different subtasks within the encoding pipeline.

The ply file loader underwent a comprehensive revamp, incorporating several enhancements that significantly improved its performance. As a result of these improvements, we see an impressive overall speedup of 70%. This speedup can be explained with two specific optimizations that played a crucial role:

- Pre-loading and Streaming: For ply files with a size smaller than 1GB, a new approach was implemented where the entire file is pre-loaded upfront and wrapped into a stream. This technique proved to be a net win for parsing speed, resulting in a remarkable improvement of approximately 40%. By loading the entire file into memory at the beginning, the parsing process can be performed more efficiently, avoiding the need to read and process the file in smaller chunks, which introduces overhead. This optimization capitalizes on the benefits of reading and processing the entire file as a contiguous stream.
- List Size Hint: Another enhancement introduced in the ply file loader was the provision of a list size hint. By providing information about the expected size of the list data structures in the ply file, a significant performance improvement of 2 times was achieved. The list size hint allows the loader to allocate the appropriate amount of memory in advance, preventing the need for frequent resizing and reallocation of list structures during parsing. This optimization reduces memory operations and improves the overall efficiency of the parsing process.

By combining these two optimizations, the ply file loader was able to achieve the speedup. The revamped loader provides more efficient and streamlined file parsing,

enabling faster data access and subsequent processing operations.

The custom sorting implementation using C++ custom comparator functors is around 28% faster than the previous implementation. Figure 37 clearly indicates a 1.4x speed improvement with the new sorting method over the original method in synthetic data and the real-world performance is very similar.

These two speedups combined resulted in an overall total encoding time speedup of 26% along with marginal improvements in the actual encoding time and decoding time with the move to the improved C++ implementation. This is a significant compression speed improvement over the original method, especially considering that currently, FRL is the fastest point cloud encoding technique roughly 1.8x faster than the MPEG standard.

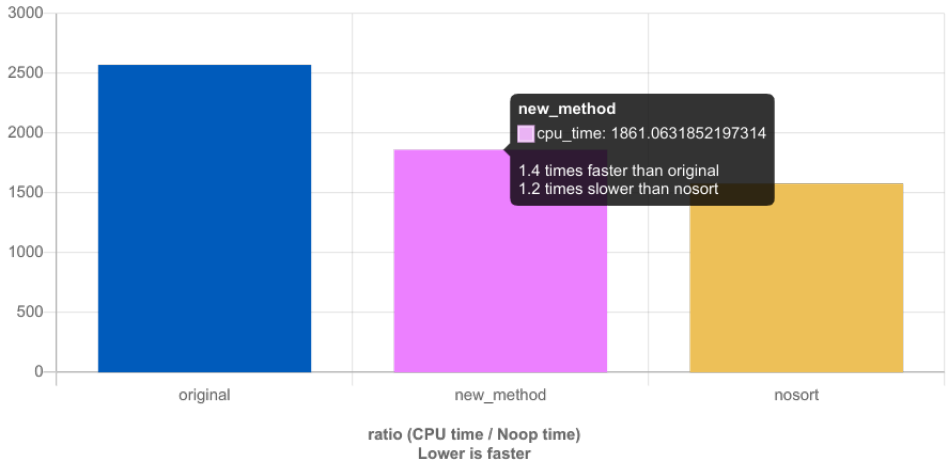


Figure 37. Sorting speed.

The analysis of Figure 36 reveals the effectiveness of the new automatic axis ordering technique, which generates significant compression gains of up to 3 bits per occupied voxel. This gain represents a substantial improvement in compression efficiency. For instance, let's consider the object "Wall_1" and order it based on decreasing occupied voxels. The resulting axis order is y, z, x. By referring to the corresponding bitrates produced by the different axis orderings, we can observe that the y, z, x ordering achieves one of the lowest bitrates, specifically 13.53.

However, it is worth noting that the axis ordering y, x, z appears to yield slightly better results in this particular case, with a bitrate of 13.49. This observation is consistent with similar behaviour observed in several other point clouds where the ratio between the occupied voxels in the last two dimensions is high. This finding suggests that there may be a correlation between the ordering of the last two axes and improved compression ratios. A possible future work could be on testing this phenomenon across a wider range of datasets to see if we could arrive at a relatively simple formula to know in which cases we need to swap the last 2 axes for better compression ratios.

Table 4 presents the results obtained from comparing the original FRL implementation with the 2D supervoxelization technique using different supervoxel dimensions (ranging from 2x2 up to 31x31) on the entire MPEG dataset. The purpose of this comparison was to examine the tradeoff between encoding denser point clouds achieved through supervoxelization and the additional bitrate required to encode the auxiliary data. The findings reveal an

interesting property of this tradeoff.

First, it is important to note that the dense point clouds, as expected, did not show any bitrate improvements when employing the 2D supervoxelization technique. In fact, these point clouds exhibited a significant increase in reported bitrates. This behavior can be attributed to the high entropy observed in the "counts" and "points" auxiliary files associated with the dense point clouds. In such cases, it is more efficient to directly apply the FRL algorithm to achieve the best compression ratios.

However, for sparse point clouds with bit depths exceeding 12, such as the Egyptian and Shiva datasets, the 2D supervoxels of size 31x31 demonstrated notable improvements of up to 2.4 bits per occupied voxel. This improvement signifies the effectiveness of enhancing the density of sparse point clouds through the supervoxelization technique, followed by applying the FRL algorithm. By making the sparse point clouds relatively denser, the compression ratios can be significantly enhanced.

The results highlight the importance of considering the characteristics of the point cloud data when selecting the appropriate compression techniques. Dense point clouds benefit more from direct application of FRL, while sparse point clouds with higher bit depths can achieve improved compression ratios by employing supervoxelization and subsequently applying FRL.

Based on the results from 2d supervoxelization one would expect 3d supervoxelization to perform even better in these sparse point clouds. Table 5 shows these results. And the 3D supervoxels of size 15x15x15 perform the best for the Egyptian dataset with a bitrate of 14.134 which is an improvement of 3.93 bits per occupied voxel much higher than the improvement with the 2D supervoxels. Similarly, Shiva reports the lowest bitrates of 13.328 bits per occupied voxel on 3D super voxels of size 7x7x7.

When comparing this implementation against the original FRL and TMC13 in Table 6 we can see that even with bitrate improvements from the 3D supervoxels method with FRL we are not able to beat the performance of TMC13 for sparse point clouds. Although the gap in bitrates between FRL and TMC13 has almost halved which is still a very good result.

From the thorough analysis, we can clearly see a trend. For dense point clouds 2D Supervoxel do not perform well while for sparse point clouds 2D Supervoxel sizes around 15 are the best. Similarly for 3D supervoxels the sweet spot lies in either 7x7x7 or 15x15x15 supervoxels. It looks like there is a difficult tradeoff between performance and super voxel size: e.g when the necessary side information is low the supervoxels are not large enough to provide useful contexts, whereas when they are large enough to compress the point clouds significantly each occupied voxel requires a lot of side information to determine its location in a supervoxel (due to the large size of the supervoxels).

Table 7 shows 4 compression methods tested against the new LiDAR datasets. Note: SparsePCGC results are still being produced and might have to be updated. As expected TMC13 the current MPEG standard has the best bitrates i.e. compression ratios in Shapenet and S3DIS which is 2 out of the 3 datasets. The SparsePCGC method closely follows TMC13 in bitrates and outperforms TMC13 in the Velodyne dataset (on which it was trained). Such a close performance was not initially expected and the results were surprising. If SparsePCGC is trained on an extensive all-composing LiDAR dataset, we could expect further bitrate improvements. FRL has slightly higher bitrates compared to these techniques. Note these

are sparse point clouds, the one area where FRL needs improvements. GZIP-9 as expected produces the worst bitrates of all, almost double than all other techniques.

6 Conclusion and future work

In conclusion, this thesis successfully achieved its objectives of gaining a comprehensive understanding of point cloud representation, the different use cases for point cloud data, investigating state-of-the-art compression techniques, and conducting a detailed analysis of geometric compression methods applied to point cloud data. We saw the different methods by which researchers try to improve the compression methods. We also noticed how important it was to not generate intermediate representations in terms of compression speeds although in some cases this intermediate representation was the main reason for the improvement in compression ratios.

Through this research, the most promising techniques were identified and thoroughly studied to understand the advantages and disadvantages of each method to learn if any techniques from one method could be borrowed to improve compression ratios or compression speed to the other. These techniques were also evaluated on a diverse set of point cloud data, generating valuable compression results and performance measurements. The dataset benchmarked against is a custom-curated superset of the original MPEG PCC Content dataset with the addition of selected LiDAR data.

A new general-purpose preprocessing pipeline was created for point cloud data which further aided in increasing compression ratios by helping to rescale the values in different axes. The entire codebase was migrated from C to C++ and innovative performance improvements were proposed for improving the C++ code further leading to an overall speed improvement of 26% compared to the original FRL implementation in C. Furthermore several extensions for the FRL technique were proposed, and these extensions were implemented in C++ and rigorously tested. This led to interesting results wherein the axis ordering technique provided a bitrate improvement of up to 3 bits per occupied voxel, and this technique was added without a significant overhead to the point cloud loading pipeline thus allowing for dynamic switching to the optimal axis ordering. The supervoxel technique was also theorized and tested leading to bitrate gains of upto 3.93 bits per occupied voxel in sparse point clouds with the 3D supervoxel variant and a bit rate improvement of up to 2.4 bits per occupied voxel in the 2D supervoxel variant. These enhancements were rigorously validated through experiments and comparative evaluations, demonstrating their effectiveness and potential for practical application.

We saw how the TMC13 standard was a very competitive technique with a nice tradeoff between compression ratios and compression speeds and how FRL was a very strong contender as an alternative. We've now improved both the compression speed and compression ratio of FRL making it even more competitive and a viable alternative to the TMC13 standard for lossless geometry point cloud compression. We also noticed that in general deep learning-based compression methods are very competitive with traditional techniques in terms of compression ratios although their speed is an important restricting factor in several use cases.

The findings and insights obtained from this research contribute to the field of point cloud compression, providing valuable knowledge and recommendations for future advancements in the area. The comprehensive report documents the entire research process, including methodology, experimental setup, results analysis, and conclusions.

The thesis touched upon multiple different aspects of Geometry point cloud compression, but not all ideas were fully explored. The next section aims to propose research directions for researchers interested in investigating further and some recommendations for them.

- Use a more representative dataset if aiming at general-purpose point cloud compression.
- Contribute to the establishment of benchmarks for evaluating point cloud compression algorithms. This includes creating benchmark datasets representative of various application domains (can start based on our dataset), and defining evaluation metrics that capture important aspects of point cloud quality.
- Focus on context selection techniques for dense point cloud geometry compression.
- Work on semi-adaptive neural network approaches for point cloud geometry.
- Focus on developing scalable algorithms that maintain high compression ratios while reducing computational requirements and memory usage. This includes exploring parallel processing techniques, distributed compression frameworks, and efficient storage formats.
- Work on developing real-time compression techniques that can effectively compress (near lossless or lossy) and decompress point cloud data in a streaming manner. This involves optimizing algorithms for low-latency processing, considering hardware acceleration, and developing efficient compression architectures.

References

[1] *Read 3-D point cloud from PLY or PCD file - MATLAB pcread - MathWorks España*. URL: https://es.mathworks.com/help/vision/ref/pcread.html.

[2] *Point cloud - Wikipedia*. 2023. URL: https://en.wikipedia.org/wiki/Point_cloud.

[3] *What Are Point Clouds, And How Are They Used? | dronegenuity*. URL: https://www.dronegenuity.com/point-clouds/.

[4] URL: https://ds.jpeg.org/documents/jpegpleno/wg1n100096-094-REQ-JPEG_Pleno_Point_Cloud_Use_Cases_and_Requirements_v1.6.pdf.

[5] *Use cases for Point Cloud Compression | MPEG*. URL: https://mpeg.chiariglione.org/standards/mpeg-i/point-cloud-compression/use-cases-point-cloud-compression.

[6] *What is a Point Cloud? - GIGABYTE Global*. URL: https://www.gigabyte.com/Glossary/point-cloud.

[7] *ARPointCloud | Apple Developer Documentation*. URL: https://developer.apple.com/documentation/arkit/arpointcloud.

[8] Yue Wang, Shusheng Zhang, Bile Wan, Weiping He, and Xiaoliang Bai. "Point cloud and visual feature-based tracking method for an augmented reality-aided mechanical assembly system". In: *International Journal of Advanced Manufacturing Technology* 99 (9-12 Dec. 2018), pp. 2341–2352. ISSN: 14333015. DOI: 10.1007/S00170-018-2575-8/METRICS. URL: https://link.springer.com/article/10.1007/s00170-018-2575-8.

[9] Augusto Luis Ballardini, Simone Fontana, Daniele Cattaneo, Matteo Matteucci, and Domenico Giorgio Sorrenti. "Vehicle Localization Using 3D Building Models and Point Cloud Matching". In: *Sensors 2021, Vol. 21, Page 5356* 21 (16 Aug. 2021), p. 5356. ISSN: 1424-8220. DOI: 10.3390/S21165356. URL: https://www.mdpi.com/1424-8220/21/16/5356/htm%20https://www.mdpi.com/1424-8220/21/16/5356.

[10] K. Kohira and H. Masuda. "POINT-CLOUD COMPRESSION FOR VEHICLE-BASED MOBILE MAPPING SYSTEMS USING PORTABLE NETWORK GRAPHICS". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-2-W4 (2W4 Sept. 2017), pp. 99–106. ISSN: 2194-9042. DOI: 10.5194/ISPRS-ANNALS-IV-2-W4-99-2017. URL: https://doi.org/10.5194/isprs-annals-IV-2-W4-99-2017.

[11] Xiaohu Lin, Fuhong Wang, Bisheng Yang, and Wanwei Zhang. "Autonomous Vehicle Localization with Prior Visual Point Cloud Map Constraints in GNSS-Challenged Environments". In: *Remote Sensing 2021, Vol. 13, Page 506* 13 (3 Jan. 2021), p. 506. ISSN: 2072-4292. DOI: 10.3390/RS13030506. URL: https://www.mdpi.com/2072-4292/13/3/506/htm%20https://www.mdpi.com/2072-4292/13/3/506.

[12] Balázs Nagy, 863x, and Csaba Benedek. "Real-time point cloud alignment for vehicle localization in a high resolution 3D map". In: (). URL: http://mplab.sztaki.hu/geocomp/.

[13] Dejing Ni, A. Y.C. Nee, S. K. Ong, Huijun Li, Chengcheng Zhu, and Aiguo Song. "Point cloud augmented virtual reality environment with haptic constraints for teleoperation". In: *https://doi.org/10.1177/0142331217739953* 40 (15 Feb. 2018), pp. 4091–4104. ISSN: 14770369. DOI: 10.1177/0142331217739953. URL: https://journals.sagepub.com/doi/10.1177/0142331217739953.

[14] *Pointclouds - Vrex*. URL: %5CUrl%7Bhttps://www.vrex.no/pointclouds/?utm_term=point%20cloud%20vr&utm_campaign=Pointclouds+US&utm_source=adwords&utm_medium=ppc&hsa_acc=3645430097&hsa_cam=16724933464&hsa_grp=132361248382&hsa_ad=590462967808&hsa_src=g&hsa_tgt=kwd-386850568506&hsa_kw=point%20cloud%20vr&hsa_mt=p&hsa_net=adwords&hsa_ver=3&gad=1&gclid=CjwKCAjwm4ukBhAuEiwA0zQxk6Umq9LyfeDchnQa_BCOFHytCmikXk2vq11gfN4IilJNwtYWsziNqRoC0lgQAvD_BwE%20%7D.

[15] *Pointcloud | VR Sketch*. URL: https://vrsketch.eu/docs-pointcloud.html.

[16] Tashi, A. M.M.Sharif Ullah, Michiko Watanabe, and Akihiko Kubo. "Analytical Point-Cloud Based Geometric Modeling for Additive Manufacturing and Its Application to Cultural Heritage Preservation". In: *Applied Sciences 2018, Vol. 8, Page 656* 8 (5 Apr. 2018), p. 656. ISSN: 2076-3417. DOI: 10.3390/APP8050656. URL: https://www.mdpi.com/2076-3417/8/5/656/htm%20https://www.mdpi.com/2076-3417/8/5/656.

[17] *Voxelization — Open3D latest (664eff5) documentation*. URL: http://www.open3d.org/docs/latest/tutorial/Advanced/voxelization.html.

[18] Touradj Ebrahimi, Siegfried Foessel, Fernando Pereira, and Peter Schelkens. "JPEG Pleno: Toward an Efficient Representation of Visual Reality". In: *IEEE Multimedia* 23 (4 Oct. 2016), pp. 14–20. ISSN: 19410166. DOI: 10.1109/MMUL.2016.64.

[19] Li Cui, Rufael Mekuria, Marius Preda, and Euee S. Jang. "Point-Cloud Compression: Moving Picture Experts Group's New Standard in 2020". In: *IEEE Consumer Electronics Magazine* 8 (4 July 2019), pp. 17–21. ISSN: 21622256. DOI: 10.1109/MCE.2019.2905483.

[20] Maurice Quach, Jiahao Pang, Dong Tian, Giuseppe Valenzise, and Frederic Dufaux. "Survey on Deep Learning-Based Point Cloud Compression". In: *Frontiers in Signal Processing* 2 (Feb. 2022), p. 846972. ISSN: 2673-8198. DOI: 10.3389/FRSIP.2022.846972.

[21] Donald Meagher. "Geometric modeling using octree encoding". In: *Computer Graphics and Image Processing* 19 (2 June 1982), pp. 129–147. ISSN: 0146-664X. DOI: 10.1016/0146-664X(82)90104-6.

[22] Diogo C. Garcia and Ricardo L. De Queiroz. "Intra-frame context-based octree coding for point-cloud geometry". In: *Proceedings - International Conference on Image Processing, ICIP* (Aug. 2018), pp. 1807–1811. ISSN: 15224880. DOI: 10.1109/ICIP.2018.8451802.

[23] S. Milani. "Fast point cloud compression via reversible cellular automata block transform". In: *Proceedings - International Conference on Image Processing, ICIP* 2017-September (Feb. 2018), pp. 4013–4017. ISSN: 15224880. DOI: 10.1109/ICIP.2017.8297036.

[24] Simone Milani, Enrico Polo, and Simone Limuti. "A Transform Coding Strategy for Dynamic Point Clouds". In: *IEEE Transactions on Image Processing* 29 (2020), pp. 8213–8225. ISSN: 19410042. DOI: 10.1109/TIP.2020.3011811.

[25] Simone Limuti, Enrico Polo, and Simone Milani. "A Transform Coding Strategy for Voxelized Dynamic Point Clouds". In: *Proceedings - International Conference on Image Processing, ICIP* (Aug. 2018), pp. 2954–2958. ISSN: 15224880. DOI: 10.1109/ICIP.2018.8451254.

[26] Ricardo L. De Queiroz, Diogo C. Garcia, Philip A. Chou, and Dinei A. Florencio. "Distance-based probability model for octree coding". In: *IEEE Signal Processing Letters* 25 (6 June 2018), pp. 739–742. ISSN: 10709908. DOI: 10.1109/LSP.2018.2823701.

[27] Julius Kammerl, Nico Blodow, Radu Bogdan Rusu, Suat Gedikli, Michael Beetz, and Eckehard Steinbach. "Real-time compression of point cloud streams". In: *Proceedings - IEEE International Conference on Robotics and Automation* (2012), pp. 778–785. ISSN: 10504729. DOI: 10.1109/ICRA.2012.6224647.

[28] Diogo C. Garcia and Ricardo L. De Queiroz. "Context-based octree coding for point-cloud video". In: *Proceedings - International Conference on Image Processing, ICIP* 2017-September (Feb. 2018), pp. 1412–1416. ISSN: 15224880. DOI: 10.1109/ICIP.2017.8296514.

[29] Diogo C. Garcia, Tiago A. Fonseca, and Ricardo L. De Queiroz. "Example-Based Super-Resolution for Point-Cloud Video". In: *Proceedings - International Conference on Image Processing, ICIP* (Aug. 2018), pp. 2959–2963. ISSN: 15224880. DOI: 10.1109/ICIP.2018.8451043.

[30] Diogo C. Garcia, Tiago A. Fonseca, Renan U. Ferreira, and Ricardo L. De Queiroz. "Geometry Coding for Dynamic Voxelized Point Clouds Using Octrees and Multiple Contexts". In: *IEEE Transactions on Image Processing* 29 (2020), pp. 313–322. ISSN: 19410042. DOI: 10.1109/TIP.2019.2931466.

[31] *Octree - Wikipedia*. URL: https://en.wikipedia.org/wiki/Octree.

[32] Hoda Roodaki, Masoud Dehyadegari, and Mahdi Nazm Bojnordi. "G-arrays: Geometric arrays for efficient point cloud processing". In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings* 2021-June (2021), pp. 1925–1929. ISSN: 15206149. DOI: 10.1109/ICASSP39728.2021.9413902.

[33] Sebastian Schwarz, Marius Preda, Vittorio Baroncini, Madhukar Budagavi, Pablo Cesar, Philip A. Chou, Robert A. Cohen, Maja Krivokuca, Sebastien Lasserre, Zhu Li, Joan Llach, Khaled Mammou, Rufael Mekuria, Ohji Nakagami, Ernestasia Siahaan, Ali Tabatabai, Alexis M. Tourapis, and Vladyslav Zakharchenko. "Emerging MPEG Standards for Point

Cloud Compression". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9 (1 Mar. 2019), pp. 133–148. ISSN: 21563357. DOI: 10.1109/JETCAS.2018.2885981.

[34] Euee S. Jang, Marius Preda, Khaled Mammou, Alexis M. Tourapis, Jungsun Kim, Danillo B. Graziosi, Sungryeul Rhyu, and Madhukar Budagavi. "Video-Based Point-Cloud-Compression Standard in MPEG: From Evidence Collection to Committee Draft [Standards in a Nutshell]". In: *IEEE Signal Processing Magazine* 36 (3 May 2019), pp. 118–123. ISSN: 15580792. DOI: 10.1109/MSP.2019.2900721.

[35] Ioan Tabus, Emre Can Kaya, and Sebastian Schwarz. "Successive Refinement of Bounding Volumes for Point Cloud Coding". In: *IEEE 22nd International Workshop on Multimedia Signal Processing, MMSP 2020* (Sept. 2020). DOI: 10.1109/MMSP48831.2020.9287106.

[36] Yingzhan Xu, Wenjie Zhu, Yiling Xu, and Zhu Li. "Dynamic Point Cloud Geometry Compression via Patch-wise Polynomial Fitting". In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings* 2019-May (May 2019), pp. 2287–2291. ISSN: 15206149. DOI: 10.1109/ICASSP.2019.8682413.

[37] Jian Xiong, Hao Gao, Miaohui Wang, Hongliang Li, King Ngi Ngan, and Weisi Lin. "Advanced Geometry Surface Coding for Dynamic Point Cloud Compression". In: (Mar. 2021). URL: https://arxiv.org/abs/2103.06549v1.

[38] Emre Can Kaya and Ioan Tabus. "Neural Network Modeling of Probabilities for Coding the Octree Representation of Point Clouds". In: *IEEE 23rd International Workshop on Multimedia Signal Processing, MMSP 2021* (2021). DOI: 10.1109/MMSP53017.2021.9733658.

[39] Wenjie Zhu, Yiling Xu, Li Li, and Zhu Li. "Lossless point cloud geometry compression via binary tree partition and intra prediction". In: *2017 IEEE 19th International Workshop on Multimedia Signal Processing, MMSP 2017* 2017-January (Nov. 2017), pp. 1–6. DOI: 10.1109/MMSP.2017.8122226.

[40] Dion E.O. Tzamarias, Kevin Chow, Ian Blanes, and Joan Serra-Sagrista. "Fast Run-Length Compression of Point Cloud Geometry". In: *IEEE Transactions on Image Processing* 31 (2022), pp. 4490–4501. ISSN: 19410042. DOI: 10.1109/TIP.2022.3185541.

[41] Rodrigo Rosario and Eduardo Peixoto. "Intra-Frame Compression of Point Cloud Geometry using Boolean Decomposition". In: *2019 IEEE International Conference on Visual Communications and Image Processing, VCIP 2019* (Dec. 2019). DOI: 10.1109/VCIP47243.2019.8965783.

[42] Eduardo Peixoto. "Intra-Frame Compression of Point Cloud Geometry Using Dyadic Decomposition". In: *IEEE Signal Processing Letters* 27 (2020), pp. 246–250. ISSN: 15582361. DOI: 10.1109/LSP.2020.2965322.

[43] Eduardo Peixoto, Edil Medeiros, and Evaristo Ramalho. "Silhouette 4d: An Inter-Frame Lossless Geometry Coder of Dynamic Voxelized Point Clouds". In: *Proceedings - International Conference on Image Processing, ICIP* 2020-October (Oct. 2020), pp. 2691–2695. ISSN: 15224880. DOI: 10.1109/ICIP40778.2020.9190648.

[44] Evaristo Ramalho and Eduardo Peixoto. "Context Selection for Lossless Compression of Bi-Level Images". In: ().

[45] Evaristo Ramalho, Eduardo Peixoto, and Edil Medeiros. "Silhouette 4D With Context Selection: Lossless Geometry Compression of Dynamic Point Clouds". In: *IEEE Signal Processing Letters* 28 (Aug. 2021), pp. 1660–1664. ISSN: 1070-9908. DOI: 10.1109/LSP.2021.3102525.

[46] Dion E.O. Tzamarias, Kevin Chow, Ian Blanes, and Joan Serra-Sagrista. "Compression of point cloud geometry through a single projection". In: *Data Compression Conference Proceedings* 2021-March (Mar. 2021), pp. 63–72. ISSN: 10680314. DOI: 10.1109/DCC50243.2021.00014.

[47] Martin Isenburg. "LASzip". In: *Photogrammetric Engineering and Remote Sensing* 79 (2 2013), pp. 209–217. ISSN: 00991112. DOI: 10.14358/PERS.79.2.209.

[48] "LAS Specification 1.4-R15 Release Information". In: (2002).

[49] *IEEE Xplore Full-Text PDF:* URL: https://ieeexplore-ieee-org.are.uab.cat/stamp/stamp.jsp?arnumber=9632532.

[50] *Lidar as a camera - digital lidar's implications for computer vision | Ouster*. URL: https://ouster.com/blog/the-camera-is-in-the-lidar/.

[51] *IEEE Xplore Full-Text PDF:* URL: https://ieeexplore-ieee-org.are.uab.cat/stamp/stamp.jsp?arnumber=6955830.

[52] Xuebin Sun, Han Ma, Yuxiang Sun, and Ming Liu. "A Novel Point Cloud Compression Algorithm Based on Clustering". In: *IEEE Robotics and Automation Letters* 4 (2 Apr. 2019), pp. 2132–2139. ISSN: 23773766. DOI: 10.1109/LRA.2019.2900747.

[53] Chenxi Tu, Eijiro Takeuchi, Chiyomi Miyajima, and Kazuya Takeda. "Compressing continuous point cloud data using image compression methods". In: *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC* (Dec. 2016), pp. 1712–1719. DOI: 10.1109/ITSC.2016.7795789.

[54] Chenxi Tu, Eijiro Takeuchi, Chiyomi Miyajima, and Kazuya Takeda. "Continuous point cloud data compression using SLAM based prediction". In: *IEEE Intelligent Vehicles Symposium, Proceedings* (July 2017), pp. 1744–1751. DOI: 10.1109/IVS.2017.7995959.

[55] URL: https://mpeg-pcc.org/wp-content/uploads/2021/09/N00151_G-PCC_codec_description_v12_d0.docx.

[56] Dat Thanh Nguyen, Maurice Quach, Giuseppe Valenzise, and Pierre Duhamel. "Multiscale deep context modeling for lossless point cloud geometry compression". In: *2021 IEEE International Conference on Multimedia and Expo Workshops, ICMEW 2021* (2021). DOI: 10.1109/ICMEW53276.2021.9455990.

[57] Dat Thanh Nguyen, Maurice Quach, Giuseppe Valenzise, and Pierre Duhamel. "Learning-based lossless compression of 3D point cloud geometry". In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings* 2021-June (2021), pp. 4220–4224. ISSN: 15206149. DOI: 10.1109/ICASSP39728.2021.9414763.

[58] Dat Thanh Nguyen, Maurice Quach, Giuseppe Valenzise, and Pierre Duhamel. "Lossless Coding of Point Cloud Geometry Using a Deep Generative Model". In: *IEEE Transactions on Circuits and Systems for Video Technology* 31 (12 Dec. 2021), pp. 4617–4629. ISSN: 15582205. DOI: 10.1109/TCSVT.2021.3100279.

[59] Jianqiang Wang, Dandan Ding, Zhu Li, Xiaoxing Feng, Chuntong Cao, and Zhan Ma. "Sparse Tensor-based Multiscale Representation for Point Cloud Geometry Compression". In: ().

[60] D. Graziosi, O. Nak agami, S. Kuma, A. Zaghetto, T. Suzuki, and A. Tabatabai. "An overview of ongoing point cloud compression standardization activities: video-based (V-PCC) and geometry-based (G-PCC)". In: *APSIPA Transactions on Signal and Information Processing* 9 (Feb. 2020), e13. ISSN: 2048-7703. DOI: 10.1017/ATSIP.2020.12. URL: https://www.cambridge.org/core/journals/apsipa-transactions-on-signal-and-information-processing/article/an-overview-of-ongoing-point-cloud-compression-standardization-activities-videobased-vpcc-and-geometrybased-gpcc/56FCAF660DD44348BCB1BCA9B5EC56CF.

[61] Emre Can Kaya and Ioan Tabus. "Lossless Compression of Point Cloud Sequences Using Sequence Optimized CNN Models". In: (2022).

[62] *Context-adaptive binary arithmetic coding - Wikipedia*. URL: https://en.wikipedia.org/wiki/Context-adaptive_binary_arithmetic_coding.

[63] Eric Bodden and Joachim Kneis. "Arithmetic Coding revealed 1 A guided tour from theory to praxis". In: (2007).

[64] *JPEG Pleno Database: Microsoft Voxelized Upper Bodies - A Voxelized Point Cloud Dataset*. URL: https://plenodb.jpeg.org/pc/microsoft.

[65] *JPEG Pleno Database: 8i Voxelized Full Bodies (8iVFB v2) - A Dynamic Voxelized Point Cloud Dataset*. URL: https://plenodb.jpeg.org/pc/8ilabs/.

[66] *Owlii Dynamic Human Textured Mesh Sequence Dataset – MPEG Point Cloud Compression*. URL: https://mpeg-pcc.org/index.php/pcc-content-database/owlii-dynamic-human-textured-mesh-sequence-dataset/.

[67] *Large Scale Parsing*. URL: http://buildingparser.stanford.edu/dataset.html.

[68] *The KITTI Vision Benchmark Suite*. URL: https://www.cvlibs.net/datasets/kitti/.

[69] *Object Visualization*. URL: https://github.com/kuixu/kitti_object_vis.

[70] *ShapeNet*. URL: https://shapenet.org/.