Visualization Research Center (VISUS)

University of Stuttgart
Allmandring 19
D–70569 Stuttgart

Master Thesis

Point cloud and Particle data compression techniques

Niranjan Ravi

Course of Study:	Information Technology
Examiner:	Prof. Dr. Thomas Ertl
Supervisors:	Dr. Bernd Meese, Dr. Guido Reina, Patrick Gralka, M.Sc.
Commenced:	November 10, 2022
Completed:	May 10, 2023

Abstract

The contemporary need for heightened processing speed and storage capacity has necessitated the implementation of data compression in various applications. This study encompasses a diverse array of applications, varying in scale, that need the implementation of efficient compression techniques. At present, there is no universally preferred compression technique that can outperform others across all data types. This is due to the fact that certain compression methods are more effective in compressing specific applications than others. Point cloud data finds widespread usage in diverse domains such as computer vision, robotics, and virtual as well as augmented reality. The dense nature of point cloud data presents difficulties with respect to storage, transmission, and computation. In a similar way, particle data usually contains significant amounts of particles that have been produced through simulations, experiments, or observations. The magnitude of particle data and the computational resources necessary to handle and examine such datasets can pose a formidable obstacle. To date, there has been no direct comparative analysis of compression methodologies applied to particle data and point cloud data. This study represents the initial attempt to compare these two distinct categories. The primary objective of this study is to test different compression techniques belonging to the particle and point cloud worlds and establish a standardized metric for evaluating the effectiveness of those compression methodologies. An integrated tool has been developed in this work that incorporates various compression techniques to evaluate the appropriateness of each technique for particle data and point cloud data. The assessment of compression techniques involves the consideration of particle error metrics and point cloud error metrics. Evidence from experiments in this work demonstrates that particle compressors exhibit superior performance across both tested data categories, while point cloud compressors demonstrate superior performance solely for point cloud data. Also, it reveals that the particle error metrics exhibit stringent boundaries, which are deemed necessary for the type of data they are intended to analyze. In contrast, the point cloud error metrics display more relaxed boundaries.

Achnowledgment

I would like to start by expressing my sincere gratitude to my thesis supervisors, **Dr. Bernd Meese** from Fraunhofer IPA, **Dr. Guido Reina**, and **Patrick Gralka** from the Visualisierungsinstitut der Universität Stuttgart (VISUS), for their continuous support and motivation. My thesis, which explored the fascinating topics of point cloud compression and particle data compression, was influenced by their knowledge of the subject. My report's completion was facilitated by their exceptional supervision. Working with them was a pleasure, and I will always be appreciative of all of their wise advice throughout the writing of my thesis.

Second, I would like to thank **Prof. Dr. Thomas Ertl** from the bottom of my heart for allowing me to complete my master's thesis in the challenging and fascinating field in his elite group. I also want to express my gratitude to the employees and other associates at the VISUS for their assistance throughout my work.

Finally, I would like to thank my parents **Ravi Natarajan** and **Saraswathi Aathinarayanan**, my sisters **Divya Booppathy** and **Anitha Vigneswaran**, my friends, and other people for their unwavering support and encouragement during my years of study as well as throughout the research and writing of my thesis. Without their help, this task could not have been completed.

Thank you.

Contents

List of Figures

List of Tables

Acronyms

API	Application Programming Interface. 24
CPU	Central Processing Unit. 15
CR	Compression Ratio. 41
DCM	Direct Coding Mode. 21
DCT	Direct Cosine Transform. 17
DPCC	Deep Point Cloud Compression. 29
GPCC	Geometric Point Cloud Compression. 19
GPU	Graphical Processing Unit. 15
HPC	High Process Computing. 15
MPEG	Moving Pictures Experts Group. 27
MSE	Mean Square Error. 43
NRMSE	Normalized Root Mean Square Error. 8
PCA	Principal Component Analysis. 17
PCC	Point Cloud Compression. 9
PSNR	Peak Signal to Noise Ratio. 19
RD	Rate Distortion. 29
RMSE	Root Mean Square Error. 27
SZ	Squeeze. 18
VPCC	Video Point Cloud Compression. 19

1 Introduction

Current high-performance computing trends indicate an exponential growth in the number of cores and a corresponding drop in memory bandwidth per core. Similar bandwidth constraints have already been identified for Input/Output operations, inter-node communications, and communication between Central Processing Unit (CPU) and Graphical Processing Unit (GPU) memory [24]. This pattern shows that the quantity of data transportation will influence the performance of computing calculations. Moreover, because huge data sets are frequently created remotely, such as on shared clusters or in the cloud, the expense of transporting the computing results for visual interpretation, quantitative analysis, and historical storage can be significant[24]. With an increase in computational power, a new problem in visualization is created. Here, the visualization and simulation must contend for the same memory and bandwidth resources, further demanding the hardware. Getting rid of redundant data, for example, by utilizing data compression, is one way to ease this bottleneck in data transfer. If data compression can be completed rapidly enough to feed the computation-starved cores, employing otherwise wasted compute cycles to compress the data makes sense[24].

For High Process Computing (HPC) applications, it is essential to substantially reduce the data size that must be dumped during the execution with relatively low computation cost and the necessary compression error bounds to save disk space, improve throughput, and increase post-processing performance. Lossless compression suffers from a poor compression ratio when dealing with exceptionally large amounts of data. Current research based on production scientific simulations demonstrates that error-bounded lossy compression algorithms are regarded as a suitable trade-off solution [1, 40]. The uniqueness of a particular feature is deemed essential in achieving optimal compression for a given application. These features may include fast support for random-access decompression, precise compression rate granularity, asymmetry (more immediate decompression than compression), limited error, support for arbitrary dimensionality, ease of parallelization, topological robustness, and so on. These factors combine to make multidimensional compression a vast and challenging problem for which no one-size-fits-all solution exists [2]. Hence in this work, different lossy and lossless compressors are analyzed to compress particle datasets and point-cloud datasets with the aim of finding a best-performing metric that can be used to measure the performance of compressors of both categories.

The data that we focus on in this work are point cloud data and particle data. Point cloud data is mostly used in different applications such as virtual reality, augmented reality, autonomous driving, and so on. It is an efficient way of representing 3D geometry. It contains the representation of the surface area of the object in 3D. In contrast, the particle data is mostly from HPC applications such as Climate-simulation, Hydrodynamics, Particle simulations, and so on. It contains information about the particles that are generated in any of these applications. The particles in the particle data are a representation of volumes. They are very large in size compared to the point cloud data in most cases. The two types of applications differ widely in different aspects; hence, different compression methods must be applied to achieve efficient compression. Various compression techniques have been analyzed based on different metrics, and two compressors are shortlisted from each category based on certain criteria. From the particle compressor category, SZ [7] and ZFP [24] are chosen, and from the point cloud category, Octattention [10] and PCC_GEO_CNN_V2 [33] are chosen.

The primary objective of this study is to examine the efficacy of compression techniques for particle data and point cloud data and assess their performance based on diverse quality metrics. This work involves the development of a comprehensive tool that integrates various compressor techniques. This tool facilitates the evaluation of these techniques on diverse datasets and enables the acquisition of corresponding results. It is feasible to figure out the techniques that exhibit superior performance in comparison to other methodologies, predicated on the outcomes derived from the conducted experiments. The secondary objective of this study is to examine and identify a singular metric that can serve as an indicator of the compressors' efficiency for both data categories.

1.1 Point cloud data and Particle data

1.1.1 Point cloud data

A three-dimensional object or environment can be represented through a collection of points in space. Every individual point is distinguished by its placement within the Cartesian coordinate system, which is represented by the coordinates (x, y, z). This is called a point cloud. Additionally, the points may possess supplementary attributes, such as color or hue. Point clouds are commonly obtained by utilizing a range of sensors, such as LIDAR, PIN hole cameras, RGB-D cameras, or photogrammetry. The interdependence among the positions of points in point clouds is noteworthy owing to their direct acquisition from the object or environment, leading to a substantial correlation between them. Point clouds are well-suited for various tasks, including object detection, segmentation, and registration, which involve identifying and localizing objects or object parts within a given scene. A prevalent technique employed in the processing of point cloud data involves the utilization of voxelized or grid-based representations. This involves the process of dividing the space into small cubes or cells and then assigning the points to the cell that is closest to them. It involves transforming the point cloud data into a three-dimensional grid or an image-like format, which can subsequently undergo analysis through traditional methods based on image processing o.

1.1.2 Particle data

In contrast, the representation of a fluid or gas system as a collection of particles is commonly referred to as particle data. Each particle in this representation possesses distinct properties, including position, velocity, and potential mass or charge. The utilization of particle data is prevalent in the realm of fluid dynamics simulations, with the objective of representing the dynamics of a fluid or gas throughout a given period. In the realm of particle data, the inter-particle relationship is comparatively weaker than that of point clouds due to the dynamic nature of particles, which are not stationary in space but rather exhibit movement and mutual interaction over time. The processing of particle data is more intricate compared to point clouds due to the interdependence of particle positions at a particular time step on their preceding positions and velocities, as well as the forces and interactions among them.

Although point cloud data and particle data possess different structural and foundational characteristics, they do exhibit certain similarities with respect to their processing and analysis. The techniques employed for data processing and analysis depend upon the attributes of the data. As an illustration, it is commonly observed that point clouds exhibit high density and stability, whereas particle data displays low density and variability. The implication of this statement is that distinct techniques may be necessary to analyze the two categories of data.

1.2 Lossy compression and Lossless compression

Lossy and lossless compression are two types of data compression techniques. Lossless compression refers to a compression technique that permits full restoration of the original data following compression and decompression. This indicates that no data is lost during compression, and the restored file is identical to the original. It works by removing redundant data and more efficient encoding of the remaining data. In lossless compression, redundancies and patterns within the data are identified and removed without losing any information. The compressed data is then stored or transmitted using fewer bits than the original data. It is widely used in applications where data accuracy is critical. The compression ratio achieved through lossless compression depends on the nature of the data. The error bound value is minimum in the case of lossless compression. But the compression ratio that can be achieved using this method is too low, especially for scientific applications.

To achieve a better compression ratio, lossy compression is usually preferred over this. Scientific data often consists of large volumes of data generated from simulations, experiments, or observations. Lossy compression methods may be used to eliminate unnecessary and insignificant material from the original stream. This is accomplished via the use of mathematical algorithms that examine data and find places that may be compressed more aggressively. In the case of scientific data reflecting a simulation, for example, the algorithm may discover intermediate stages or minor details that have a reduced influence on the ultimate output. These sections may be compressed more aggressively to get larger compression ratios without hurting the overall results much. Wavelet-based compression [42, 44, 52], Direct Cosine Transform (DCT) [4, 29, 53], and Principal Component Analysis (PCA) [8, 13] are some of the basic examples of lossy compression methods used in scientific data or HPC. These strategies make use of the data's mathematical structure to uncover duplicates and extraneous material that may be deleted. To guarantee that the compressed data retains the precision necessary for scientific analysis or simulation, the compression method and amount of compression must be carefully chosen. In this work, the main focus is on different lossy compression techniques. Several lossy compression algorithms have been proposed in recent times for particle data and point cloud data.

1.3 Compression methods for particle data

Various techniques for data compression have been effectively employed in the domain of image and video processing applications. However, it should be noted that particle disciplines demand the quantification of errors for variables of significance. Therefore, these methodologies cannot be directly employed in such areas of research [3]. Developing a lossy compression algorithm that exhibits a substantial compression ratio for particle data compression use cases while simultaneously ensuring a constrained error rate can pose a challenge. For instance, ZFP is a lossy compression method that is based on orthogonal block transform and embedded coding [24]. An orthogonal block transform is a technique used to transform data into a set of frequency coefficients, which represent the energy or power of the corresponding data at different frequency bands. Embedded coding works by encoding the frequency coefficients in a hierarchical fashion, with the most important coefficients encoded first. The ZFP compressors use these methodologies in order to achieve high compression and decompression rates. However, it is incapable of generating a compression ratio that is significantly high while sticking to a predetermined error threshold. The ISABELA compressor converts multi-dimensional snapshot data of simulations into a sequence of sorted data before compressing it using an interpolation technique called B-spline interpolation. [23]. It is a type of mathematical interpolation technique that is used to construct a smooth curve that passes through a set of given data points. However, this process loses the location information of the data points, and it has to use an additional index array to record the original position of each point. As a result, ISABELA [23] suffers from low compression ratios, mainly when working with a large number of data points. Another compression method for particle data, known as Squeeze (SZ) [7], offers better results in terms of compression ratio than ZFP. It leverages multiple curve-fitting models to encode the data stream. It supports both absolute error bound and relative error bound. The essential concept of this method is to evaluate each data point to determine if it can be approximated by a best-fit curve fitting model and replace it with a two-bit code representing the model type if the approximation is within a user-specified error bound. Also, a GPU version of this method exists, called CuSZ [46]. The main operations are parallelized, leveraging the GPUs and achieving comparatively higher compression and decompression rates [46]. A unique tensor decomposition-based compression technique, TTHRESH, is aimed at storage and visualization applications, with the primary goal being data compression at high compression ratios [2]. It works based on the Tucker decomposition model and higher-order singular value decomposition (HOSVD) procedure to construct the orthogonal Tucker factors. Tucker decomposition is a technique used to decompose a high-dimensional tensor into a smaller core tensor and a set of factor matrices. TTHRESH is mainly focused on visualization applications. Hence it cannot be trusted for the precision and accuracy of data. It is evident that each compression technique for the particle data is constrained to a specific application or use case. Hence, in this work, since the main focus is on accuracy and precision, the compressors that perform best in this category are chosen.

1.3.1 Frameworks

Apart from these methods, there exist several frameworks to test the quality of different compression techniques. One such framework is called Z-checker [43], designed to assess the quality of lossy compression techniques used in scientific research. Its goal is to evaluate the compressed data's viability for scientific analysis while also offering insight into the strengths and limitations of various compression techniques. The Z-checker framework is made up of three parts: the Z-checker program, the Z-compressor software, and the Z-checker database. The Z-checker program is used to assess the quality of compressed data by examining parameters including compression ratio, error rate, and compression speed. The Z-compressor program compresses scientific data using several techniques. They have tested two compression techniques, SZ and ZFP, by integrating them into their framework. The evaluation metrics provided by the Z-checker framework include compression ratio, Peak Signal to Noise Ratio (PSNR), maximum error, mean error, and compression time. These metrics help users to understand the trade-offs between compression quality and computational efficiency of different compressors. By providing a standardized evaluation process and a comprehensive database of compression results, the Z-checker framework enables users to make informed decisions about which compression algorithm to use for their specific scientific data. It is the first tool that is designed to assess the lossy compression for scientific data sets [43]. Another framework, called FRaZ [47], is a fixed-ratio lossy compression framework respecting the user-specified error constraints. It is mainly designed to accurately determine the appropriate error settings for different lossy compressors based on target compression ratios. It is tested for different compressors like SZ and ZFP by integrating them into their framework. However, it should be noted that, unlike these frameworks, this particular work facilitates the incorporation of point cloud compressors also.

1.4 Compression methods for point cloud data

When it comes to point cloud data compression, there are primarily two categories to consider: geometry compression and attribute compression. In this case, the main concern is with the geometric compression of point cloud data. Research on PCC methods has established two standards, namely Geometric Point Cloud Compression (GPCC) [16] and Video Point Cloud Compression (VPCC) [20]. A recent survey by Cao et al. [5] presented an in-depth exploration of the existing approaches in this field. An updated and comprehensive survey is now available [6], providing a more current review of the topic. Furthermore, Quach et al.[30] gave a broad overview of point cloud compression, focusing on learning techniques. As point cloud data has recently dominated the digital landscape, developing and implementing efficient compression techniques is crucial to handle the ever-increasing daily volume of data. The point cloud compression methods are classified into traditional methods and learning-based methods. Figure 1.1 represents the different categories or basis for classification of the learning-based point cloud compression methods.

A voxel is a fundamental unit of three-dimensional space that represents a small volume element, often in the shape of a cube, within a larger three-dimensional space. There exist several point cloud compression methods that are based on voxel data structures [10, 28, 35]. Geometric patterns can be naturally preserved in the voxel form, as opposed to an octree. But the negative impact of voxel-based networks is that it is susceptible to variations in density and may not work with sparse point clouds. To find a solution to the density variation issue, the octree technique directly processes the octree occupancy code. The octree format was proposed in the early 1980s as a tool for the geometric modeling of arbitrary 3D objects [38]. Nevertheless, it was not applied in a point cloud compression method until 2006 [21]. Since then, various techniques for point cloud compression have used octree representation [11, 12, 18, 22, 36, 39]. An inter-frame octree-based method, presented by De Queiroz et al. [34], constructs the context by calculating voxel distances to occupied voxels in a reference frame. Learning-based compression often aims to capture common

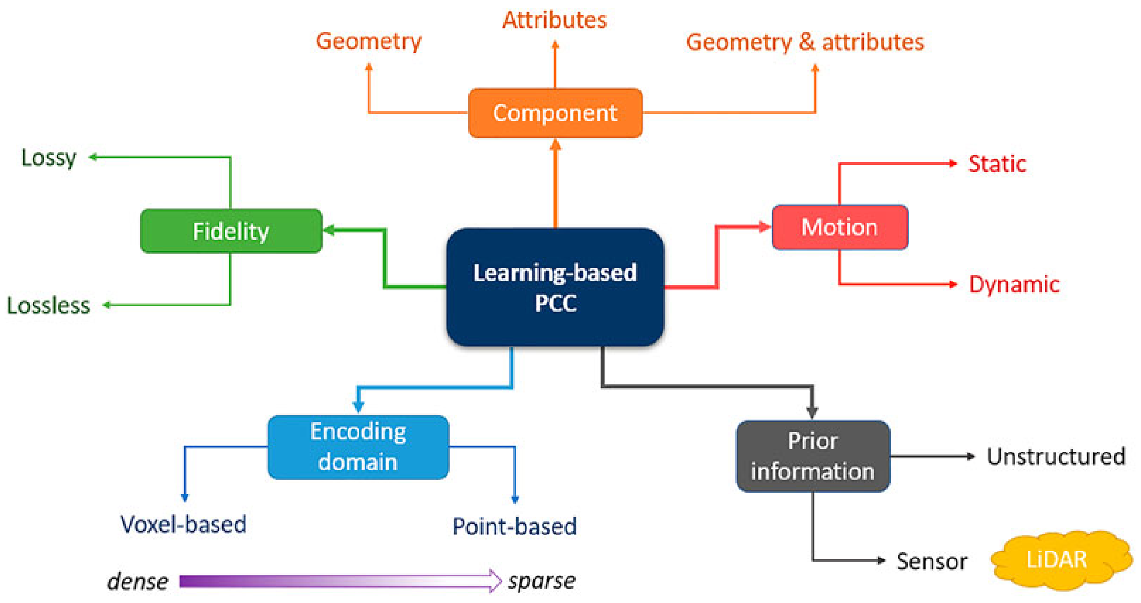


Figure 1.1: Learning-based PCC classification[31]

scene properties [39]. Recently, deep neural autoencoder networks such as those described in [19], [18], and [33] have been developed and effectively applied to compress 3D point cloud data in various domains. These autoencoder networks use unsupervised learning to automatically extract and learn the most relevant features of the input data, resulting in a more efficient compression of the 3D point cloud data. By leveraging the power of deep neural networks, these techniques have successfully achieved high compression ratios while preserving the essential information in the data. OctAttention generates and utilizes large-scale contexts that include information from different parts of nodes of the tree structure, such as ancestral parts and sibling parts of the current node [10]. This method employs parallelism while encoding multiple nodes. Deep compression for dense point clouds [16] is a technique that learns common patterns that arise through local feature descriptors and uses them to compress and reconstruct the point cloud data. Another method that uses deep learning to adapt approximation models to alleviate the shortcomings of octree structure is mentioned in [33]. It improved from using shallow networks to compress the point clouds to deep networks to compress larger point clouds. A trade-off between memory usage and coding performance is established [33]. Apart from compression algorithms that are tailored to specific applications, there exist general-purpose compression algorithms for point clouds that are intended to perform compression effectively across a broad spectrum of applications. In this work, two such compressors that are presumed to work for comparatively larger datasets are chosen to be integrated into the tool.

1.4.1 Voxels

A voxel is a three-dimensional element that is cube-shaped and represents a specific value or attribute of a point in space. To clarify, a voxel can be defined as a three-dimensional pixel that possesses a spatial position and encompasses data pertaining to the entities that exist at that particular location. A voxel has the capacity to encapsulate various physical or virtual attributes, such as color, density, or other properties. In order to represent a dataset that is voxel-based, it is possible to organize the set of voxels into a three-dimensional grid, where each individual voxel is situated at a distinct x, y, and z coordinate within the spatial domain. Subsequently, the arrangement of volumetric pixels can be depicted as a three-dimensional visual representation or sequence of images for the purpose of presenting the fundamental data. The act of transforming a 3D geometric model or point cloud into a representation based on voxels is known as voxelization. To clarify, the process of voxelization involves dividing the spatial area occupied by the three-dimensional model into a uniform grid composed of tiny cubes, with each cube denoting a voxel [50]. The selection of algorithms for performing the task is dependent upon the characteristics of the input data and the targeted output resolution.

1.4.2 Octree coding

The voxelized point cloud is represented using an octree structure [27]. Consider the point cloud to be contained in a quantized volume of $(D)x(D)x(D)$ voxels. As shown in Fig.1.2, the volume is initially divided vertically and horizontally into eight sub-cubes with dimensions $(D/_2)x(D/_2)x(D/_2)$ voxels. This procedure is repeated recursively for each occupied subcube until D equals 1. Notably, only 1 of voxel positions are occupied on average [37], which makes octrees a very convenient way to depict the geometry of a point cloud [17]. During each phase of decomposition, an assessment is made to identify the units that are currently occupied and those that are unoccupied.

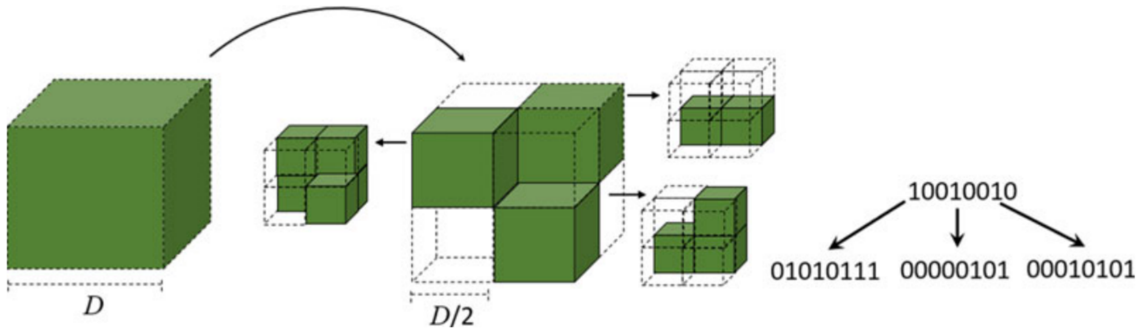


Figure 1.2: Octree visualization [17]

Occupied blocks are designated with a 1, while vacant blocks are marked with a 0. These octets represent the occupancy state of an octree node in a single-byte word and are compressed by an entropy coder that considers their correlation with adjacent octets [17]. For the coding of isolated points, since there are no other points within the volume to correlate with, Direct Coding Mode (DCM) is introduced [25] as an alternative to entropy coding the octets. In DCM, the coordinates of the points are encoded explicitly without compression. DCM mode is inferred from neighboring nodes to avoid signaling DCM mode for every node in the tree [17].

These are some of the general information regarding particle compression and point cloud compression. From here on, this work is structured as follows:

Chapter 2 – Methods: presents the review of the literature and analysis related to different compression techniques.

Chapter 3 – Implementation describes the tool that is designed to test different compressors, different sorting techniques, renderings, and datasets.

Chapter 4 – Evaluation summarizes the metrics that are used to evaluate different compressors, results generated using different compression techniques, discussion, and future work.

Chapter 5 – Conclusion provides the conclusion for this work.

2 Methods

In this part, some of the fundamental data compression techniques are reviewed in detail. These include compressors from both particle and point cloud categories. These principles are important for comprehending the compressor methods used in the integrated tool.

2.1 Particle compressors

Particle data compressors operate by spotting patterns in the data that can be used to cut down on the amount of information required to accurately represent the data. This is accomplished by more effectively encoding the data to make use of its underlying structure and correlations. In this work, two compressors from this category are shortlisted based on the below analysis to integrate into the tool and test the performance of these compressors for the datasets used in this experiment.

2.1.1 SZ

One of the best-performing compressors, SZ, is designed to compress data that are generated by high-performance computing (HPC) applications. Large particle datasets created by simulations or experiments in HPC systems can be compressed using the lossy compression algorithm SZ. The technique first linearizes the multi-dimensional snapshot data and then uses the prediction error to further compress the data using a wavelet transform. The experiments specifically demonstrate that even when employing the most straightforward Z-order scanning technique to linearize the multi-dimensional data, the time cost is twice as lengthy as that of the compression without the linearization step due to several expensive multiplication operations [7]. As a result, it is suggested to build the 1-D data sequence for compression using the data array's inherent memory sequence.

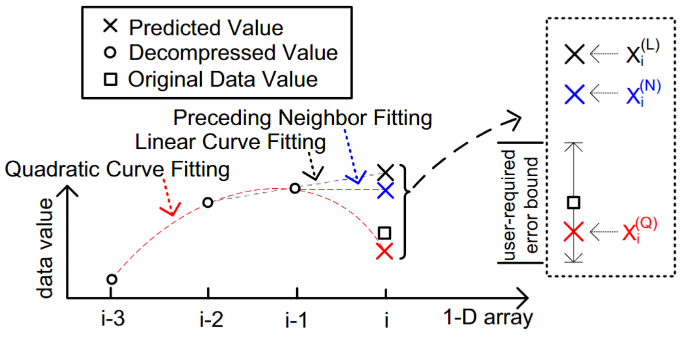


Figure 2.1: Illustration of Fitting Models used in SZ[7]

This technique involves the utilization of curve-fitting models, namely preceding neighbor fitting, linear-curve fitting, and quadratic-curve fitting, as illustrated in Figure 2.1. The objective is to select optimal curve-fitting models that can effectively accommodate or forecast the data points with precision while also adhering to the user-defined error margins. Subsequently, the code pertaining to the curve-fitting model is employed to substitute the data that can be anticipated with a considerable degree of precision. Moreover, the utilization of binary representation analysis is employed to effectively execute lossy compression on unforeseen data that cannot be approximated by curve-fitting models [7]. The primary benefit of this conversion methodology is a noteworthy decrease in conversion expenses and enhanced preservation of the locality. The findings indicate that SZ exhibits potential as a data compression method for high-performance computing (HPC) applications that necessitate precise data and high compression ratios. Based on its performance and capability to compress different types of data, it has been shortlisted to integrate into the tool as one of the particle compressors. It is implemented using the C programming language and also has Application Programming Interface (API)s for other programming languages like C++, Python, and so on. The integration of this compressor technique is fairly easier as the library is well-maintained, and a proper manual with a set of instructions is provided.

2.1.2 CuSZ

The GPU version of SZ is called CuSZ [46], which uses GPU to perform compression and decompression of data. The main reason to use GPU to compress data is to increase compression and decompression throughput. This is the first ever error-bounded lossy compressor designed to run on GPU to use the memory bandwidth and parallelism provided by the GPUs. The amount of parallelism that has been applied in this compressor at each level of compression and decompression is shown in Figure 2.2. Several customizations have been proposed in this method, like the dual-quantization scheme and efficient Huffman coding, which are different from the original method. These customization steps over the original method are shown in Figure 2.2.

compression	sequential	coarse-grained	fine-grained	atomic
DUAL-QUANTIZATION			●	
histogram			●	●
build Huffman tree	●			
canonize codebook	●		●	●
Huffman encode (fix-length)			●	
deflate (fix- to variable-length)		●		
decompression				
inflate (Huffman decode)		●		
reversed DUAL-QUANTIZATION		●		

Figure 2.2: Amount of parallelism applied in CuSZ's compression and decompression at each step[46]

All these improvements result in an enormous increase in compression throughputs compared to when run in CPU while retaining the same quality in decompressed output and improving compression ratio by around 3.5 times [46]. This method has been tested to integrate into the tool in this work. Unfortunately, there were memory issues and GPU architecture issues that were still unaddressed at that point in time by the respective authors. It worked fine for the datasets that are tested in their paper [46], but it did not work for the datasets used in this experiment.

2.1.3 ZFP

The ZFP method entails breaking the data into fixed-size blocks and using an iterative compression methodology that uses quantization, encoding, and entropy coding techniques to reduce each block to a defined number of bits. The resultant compressed data is saved in an array format that allows for quick on-demand decompression as needed. It is a compression technique with a fixed rate that allows for arbitrary reading and writing of d-dimensional floating-point arrays in small blocks of 4^d values [24]. It is one of the methods where the user can mention the exact number of bits to allocate for each array. It is more suitable for 2D and 3D data and does not support higher-dimensional data.

The main steps that this compression technique follows are that it first aligns the floating-point values in a block to a typical exponent, followed by conversion to fixed-point representation and application of orthogonal block transform, after which the system orders the transform coefficients and encodes the bit planes individually [24]. The quantized data is encoded with a mix of delta encoding and Huffman coding to take advantage of surrounding value correlation and eliminate redundancy. Entropy coding with an arithmetic coding technique is used to further compress the encoded data. To aid efficient decompression, the compressed data is stored in an array format that includes information. It is swift as the operations involved primarily are integer addition and shifting. The compressor yielded a high compression ratio of 16 times or more for visualization and analysis applications, and they could often apply it without a significant loss of data quality. Effectiveness is not compromised in these applications when using compressed data [24].

As a result, high compression ratios are achieved by leveraging the correlation between nearby values in the input data and employing effective entropy coding algorithms. The method is also intended to have minimal decompression overhead, which implies that the compressed data may be effectively decompressed on the fly without needing unnecessary processing. Also, because of the method's ability to achieve constant-rate compression, the compressed data always contains a fixed number of bits per value (even user-specific bits), regardless of the distribution of the original data. This can be used in applications that need a certain amount of storage or a set pace of data transfer. Since the method is lossless, the compressed data may be precisely restored to the original input data [24] which makes it highly applicable for storing and retrieving large datasets.

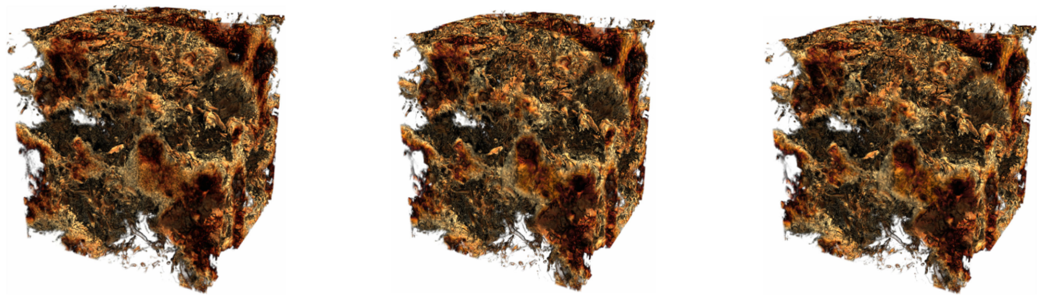
The implementation of this compression technique is carried out using the C++ programming language. The API is available in various programming languages, including C and Python. This compressor exhibited encouraging performance for the particle data classification, representing one of the earliest instances of such success. This method can be utilized to compress floating-point arrays that possess less than three dimensions. This compressor is capable of supporting both particle data and point cloud data, as they can both be represented in the form of floating-point arrays. Presently, it is widely recognized for its rapid compression and decompression rates. Comparable to SZ, ZFP also possesses a meticulously curated repository of resources, complete with precise guidelines for its application in the specific use case required for this study. Therefore, it has been incorporated into the tool developed in this study as the second approach within the particle compressors category.

2.1.4 CuZFP

The CuZFP refers to a variant of the ZFP compressor [24] that is designed to operate on a GPU architecture while relying exclusively on integer addition and shifting operations, much like its CPU-based counterpart. The developers exercised their discretion to utilize the GPU for the purpose of enhancing the execution speed, which led to a noteworthy escalation in the throughput of the compression and decompression of floating-point arrays. However, the compression ratios attained were comparable to those obtained when executed on CPUs, as no logical aspect of the execution on GPUs was altered. This compression technique is added in the same ZFP library and requires only minimal modifications to run it. It works for the datasets that are used in this experiment. The option to execute CuZFP along with normal execution of ZFP is integrated into the tool in this work. Still, it is not included in the test cases in this experiment as it would not be a fair comparison between the other particle compressors.

2.1.5 TTHRESH

It is a method that prioritizes visualization and user exploration of the decompressed data. Hence higher error rates are acceptable in this case. As a result, this compressor achieves high compression ratios. This is designed for Cartesian grid data that consists of three or more dimensions. It uses the tensor decomposition and Tucker model for higher-order compression and dimensionality reduction in the field of graphics and visualization [2]. Despite the relatively slower compression and decompression speeds of this method, it makes up for it by addressing the concerns related to storage and visualization. Figure 2.3 shows the renderings of the original file and the decompressed files that are obtained using this method. In these renderings, there are barely any differences between the original file and the decompressed files that are visible to the naked eye.



(a) 300:1 compression (1.71MB) **(b)** 10:1 compression (51.2MB) **(c)** Original (512MB)

Figure 2.3: Renderings of data using TTHRESH compressor[2]

This novel approach is not implemented as a library, but it is implemented as a standard executable as an open-source C++ implementation. Similar to SZ and ZFP, even this method has options to set the target Root Mean Square Error (RMSE) and PSNR values required in the compression. When tested this method with the datasets used in this experiment, it gave segmentation errors when performing the Tucker decomposition operation on the tensors. The input data was not acceptable by this compressor as its distribution was sparse. It required more dense distributions of data that are 3-dimensional or higher than that. Since the datasets in this experiment do not exhibit these characteristics, this method looks less appealing to be integrated into the tool for further explorations.

2.2 Point cloud compressors

The point cloud compressors are designed to reduce the size of a given set of points in three-dimensional space while retaining the maximum possible level of accuracy in representing the original points. It is aimed at the compression of geometry or attributes of the object point clouds. In this work, two compressors from this category are shortlisted based on the below analysis to integrate into the tool and test the performance of these compressors for the datasets that are used in this experiment.

2.2.1 MPEG-PCC

The Moving Pictures Experts Group (MPEG) standards group developed a point cloud compression technique based on geometry called Geometric Point Cloud Compression GPCC. It is a geometry-based method that uses a pyramidal arrangement of cubes to encode points in three-dimensional space. It is particularly effective for point sets that are uniformly distributed, outperforming other state-of-the-art methods. However, there may be better choices for sparse distributions [41]. The GPCC offers a native 3D representation and the potential for further improvements that have not yet been fully explored. The G-PCC coding scheme employs three attribute coding techniques, namely Region Adaptive Hierarchical Transform (RAHT) coding, interpolation-based hierarchical nearest-neighbor prediction (Predicting Transform), and interpolation-based hierarchical nearest-neighbor prediction with an update/lifting step (Lifting Transform) [17]. The MPEG-PCC group has created standardized metrics that are used to assess the quality of the point cloud compressors. It is used as a standard metric for evaluation in GPCC. This method is investigated with the objective of determining the intended use of the standard point cloud error metrics in a point cloud compression system. As a result of its standardization, numerous other point cloud compressors have implemented it, leading to its incorporation as the point cloud error metric in this study. More about these metrics are discussed in Chapter 4.

2.2.2 OctAttention

It is one of the learning-based methods that is used for compressing point cloud data. Since voxels are often regarded as ineffective for representing sparse point clouds, this compression technique uses octrees instead of voxels. This unique technique proposes a tree-structured attention mechanism that efficiently eliminates geometric redundancy [10] to simulate the interdependency of nodes within a large-scale context. Figure 2.4 shows the architecture of the Octattention compressor. The numerical value present within the node in Figure 2.4 signifies the corresponding occupancy code in the decimal system. The initial step involves the encoding of the point cloud into an octree structure, wherein each node of the octree is distinguished by its level, octant, and occupancy code. These three characteristics are integrated individually. An instance of constructing a context window (red) with a length of N=8 is presented as an illustration in Figure 2.4. The utilization of three levels of predecessors within the context window is denoted by the color green, where the height of the context window is K=4. The context in the window is first used to encode a node (blue), and then it is fed into a masked context attention module (right), which is then used to model the occupancy code distribution by a multi-layer perceptron (MLP). The serialized occupancy code is then encoded using the predicted distribution by the arithmetic encoder into the final compressed bit stream [10]. The sole stage that results in compression degradation is the one where the quantization error is ascertained. However, it is imperative to increase the depth of the octree in order to achieve the desired level of precision.

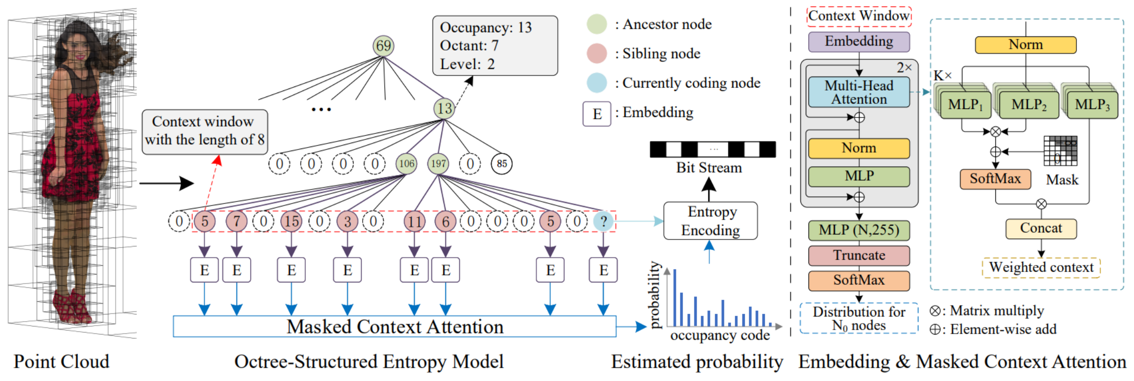


Figure 2.4: System overview of OctAttention [10]

The source code of this compression tool was made publicly available in the git repository as an open-source library. In contrast to alternative point cloud compressors, the training and implementation of the compression models utilized in this study were comparatively straightforward. The absence of errors during the execution process may be regarded as a contributing factor to this result. It is implemented using the Pytorch library in Python. The instruction manual provides a detailed set of instructions for the operation of this compressor. The compressor was utilized to perform compression and decompression procedures on a sample dataset from both point cloud and particle data during the preliminary examination. The learning-based approach of this method presents a potentially intriguing opportunity for further investigation and in-depth analysis, particularly when compared to the fact that the particle compressors selected for integration into the tool are not learning-based. Hence it is considered to be included as one of the point cloud compressors of the tool developed in this experiment.

2.2.3 Improved Deep Point Cloud Geometry Compression

It is an improved version of the model Deep Point Cloud Compression (DPCC) [32] as it has a memory usage problem which is a limitation for compression and decompression of large point clouds. The main focus of this compression technique was to compress the geometry of static point clouds in a lossy manner. It is based on the auto-encoder architecture and octree structuring. By encoding approximations over a coarse octree, it is possible to mitigate the drawbacks of the octree structure for lossy compression [33]. Various factors such as entropy modeling, deeper transforms, sequential training, optimal threshold, and changing balancing weight in focal loss are tweaked to improve the performance of the DPCC model [33]. From the paper [33], the authors train the compression models for each Rate Distortion (RD) trade-off with a corresponding value in their study. This method enables the customization of transforms and entropy models, which improves RD the performance. Unfortunately, training one model for each trade-off is time-consuming when utilizing this independent-training approach. Therefore, a novel sequential training technique to solve this problem is offered to accelerate training and improve RD performance. The basic idea is to utilize previously learned neural network weights as the foundation for future neural networks. Initially, a low-distortion, high-bitrate model is trained, and then it uses its learned weights for each subsequent model. The different trade-offs are learned in decreasing order, with the bitrate continuously dropping while the growth in distortion is minimized. This compressor is referred as PCC_GEO_CNN_V2 or PCC_GEO in this work.

The compressor has been developed specifically to compress voxelized input data. Voxelization is the technique of partitioning the spatial domain occupied by the 3D model into a regular grid consisting of small cubes, where each cube represents a voxel. The process of voxelization involves the creation of voxels in accordance with the required number of samples. In general, the voxelization process has the potential to lead to data loss. However, even if it may not have the same degree of detail, it is done so as to have the benefit of speed and efficiency while processing 3-D data. In contrast to point cloud data, which captures the surface area of a scanned object, particle data provides information on individual particles in three dimensions. The source code utilized in this study is publicly accessible on the Git repository. The authors' research has demonstrated this approach's capability to effectively compress large data sets. It is implemented in the recent version of Python (3.6) using the TensorFlow library. The instructions for executing the code can be found in the repository. No issues were encountered during the setup process. Therefore, this approach is regarded as the secondary point cloud compression technique that can be incorporated into the tool for more comprehensive examination.

2.2.4 Deep Compression for Dense Point Cloud Maps

This technique is based on the deep convolutional autoencoder architecture similar to the previous method. The primary goal of this technique is to compress the data in a lossy way, which means some information will be lost in the compression process. The process begins with the encoder compressing the input data into a more generalized representation, called an embedding or code. This compressed format is then used as the input for the decoder, which attempts to reconstruct the original data. The network is trained using a technique called backpropagation, where the reconstructed output is compared to the original point cloud data. The structure of the encoder, decoder, and backpropagation of this compressor technique is shown in Figure 2.5. This type of lossy compressor is usually preferred to integrate into the tool for further analysis and test it with datasets from the point cloud and particle categories. But there is a file missing in the git repository, which is mainly used by this technique. Despite the existence of a git repository containing the code, the absence of a particular file hinders the ability to replicate the author's work. Also, the git repository has not been maintained for the past few years. Therefore, this cannot be considered to integrate into the tool.

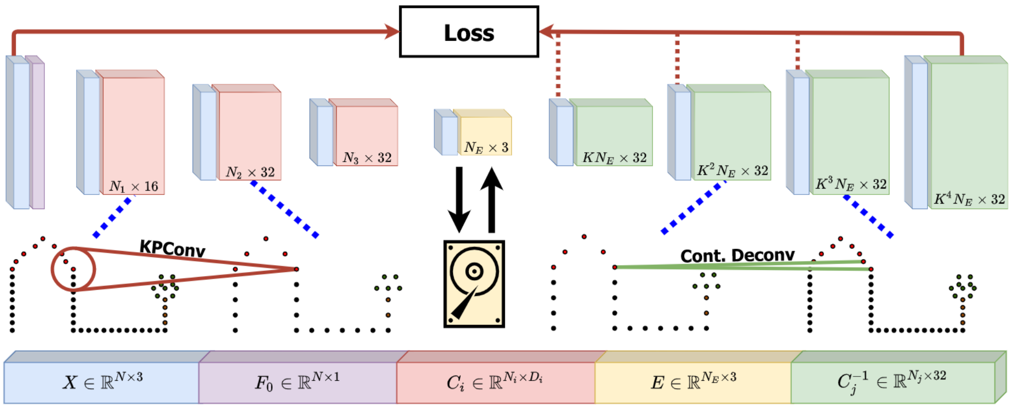

Loss
$N_1 \times 16$
$N_2 \times 32$
$N_3 \times 32$
$N_E \times 3$
$KN_E \times 32$
$K^2N_E \times 32$
$K^3N_E \times 32$
$K^4N_E \times 32$
KPConv
Cont. Deconv
$X \in \mathbb{R}^{N\times 3}$
$F_0 \in \mathbb{R}^{N\times 1}$
$C_i \in \mathbb{R}^{N_i\times D_i}$
$E \in \mathbb{R}^{N_E\times 3}$
$C_j^{-1} \in \mathbb{R}^{N_j\times 32}$

Figure 2.5: System overview of Deep Compression for Dense Point Cloud Maps [48]

2.2.5 Deep Autoencoder based Lossy geometric compression

This method is also based on autoencoder architecture for lossy point cloud compression. The architecture of this method is provided in Figure 2.6. It differs from other point cloud compressors because it takes the point cloud data directly as input rather than a voxel grid or collection of images. The architecture comprises an encoder based on PointNet, a quantizer that is uniform, a block for estimating entropy, and a module for nonlinear synthesis transformation [51].

Figure 2.6 shows the overall flow of this compression technique. The initial stage of this compression methodology includes the utilization of sampling layers, which execute a downsampling procedure on the input point clouds, resulting in a point cloud that exhibits a noticeable point density. Subsequently, the reduced point set is fed into a codec based on an autoencoder architecture, which consists of an encoder that produces a compressed representation from an unstructured point set supplied as input to the quantizer. The reconstructed point cloud is generated by the decoder using the quantized representation that is transmitted from the quantizer [51].

From an architectural standpoint, this compression technique appears to hold promise. However, its implementation is solely intended to provide support for the ShapeNet point cloud dataset. Therefore, this compression methodology is unsuitable for the use case in this study and cannot be deemed as a viable option for further examination.

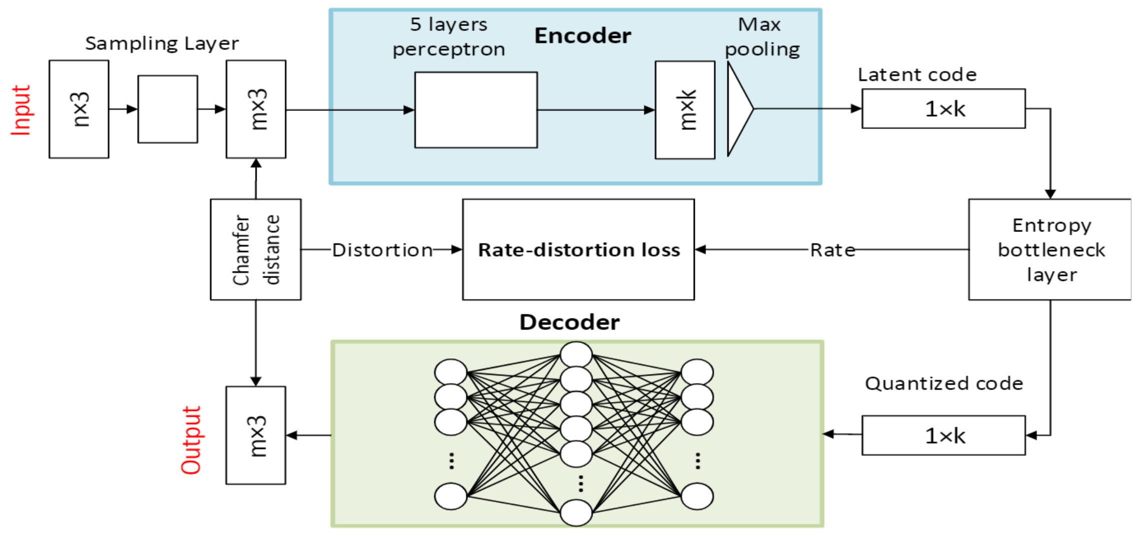


Figure 2.6: System overview of Deep Autoencoder based lossy geometric compression [51]

2.3 Analysis overview

The above analysis reveals that individual compressors are tailored to compress data for distinct purposes. A universal solution for all types of data does not exist. Consequently, there is a necessity to create a tool that can incorporate these compressors and assess their efficacy on diverse datasets. Table 2.1 provides an overview of compressors that have been analyzed in this work.

Compressor	Dataset Type	Dataset size tested in their work	Type of CPU/GPU	Neural networks usage	Availability of code	Programming Language
SZ [7]	Particle data	1.5TB	Argonne FUSION cluster server - 16cores and 64GB memory	No	Yes	C
ZFP [24]	Particle data	3.6GB	Single core of an iMac with 3.4 GHz Intel Core i7 processors and 32 GB of 1600 MHz DDR3 RAM	No	Yes	C++

Continued on next page

Table 2.1 – *Continued from previous page*

Compressor	**Dataset Type**	**Dataset size tested in their work**	**Type of CPU/GPU**	**Neural networks usage**	**Availability of code**	**Programming Language**
CuSZ [46]	Particle data	<6.3GB	NVIDIA V100 GPU	No	Yes - with GPU memory issues	C++, CUDA
TTHRESH [2]	Particle data	512MB	4-core Intel i7-4810MQ with 2.80 GHz and 4 GB RAM	No	Yes - Not able to run for the datasets used in this experiment	C++
OctAttention [10]	Point cloud data	<300MB	Xeon E5-2637 CPU NVIDIA TITAN Xp GPU (12GB memory)	Yes	Yes	Python, pytorch
Deep Autoencoder-based Lossy Geometry Compression [51]	Point cloud data	1-3MB	i7-8700 CPU and a GTX1070 GPU (with 8G memory)	Yes	Yes - missing file, hindering the execution of code	Python2.7
Deep compression for Dense point clouds [48]	Point cloud data	2-5MB	GeForce RTX 2080 SUPER and with an IntelCPU with3.5 GHz	Yes	Yes - with several external dependencies and unmaintained git repository	Python, Pytorch, Open3d
Improved Deep Point Cloud Geometry Compression [33]	Point cloud data	15-20MB	Nvidia GeForce GTX 1080Ti	Yes	Yes	Python3.6, Tensorflow

Table 2.1: Overview of analysis of different compressors

3 Implementation

3.1 Tool Design

The overview of tool design is shown in Figure 3.1.

3.1.1 Compression technique selection criteria

The primary objective underlying the development of this tool is to conduct a comparative analysis of the efficiency of various compressor variants across diverse datasets. To accomplish this task, it is essential to design the tool in a flexible manner that can effectively manage diverse compression methodologies for a wide range of datasets. The primary factor for evaluating the suitability of incorporating the compressor technique into the tool is its operational simplicity. In order to achieve this, it is crucial that the source code for the compressor is made readily available for unrestricted use by the general public. Each of the techniques listed in the table is accompanied by its corresponding source code, which can be found in the respective GitHub repositories. Multiple challenges were encountered during the execution of the code aimed at replicating the compressor techniques employed in the original work. Several factors can impede the optimal functioning of a compressor, including hardware limitations, software restrictions, coding errors, inadequate documentation, memory constraints, and other similar issues. Certain compressors were specifically engineered to solely perform compression and decompression on a predetermined group of datasets. It is evident that certain git repositories have not received adequate maintenance in recent years, thereby indicating a lack of sufficient support for the associated compressor in the event of inquiries or complications. Following an analysis of various compressors, as outlined in Table 2.1, four compressors have been selected for integration into the tool. Two compressors designed for compressing particle data (SZ and ZFP) and two compressors designed for compressing point cloud data (Octattention and PCC_GEO) were selected.

3.1.2 Setup

To achieve file compression and decompression, a distinct set of instructions must be supplied to each compressor. Prior to executing the point cloud compressors Octattention and PCC_GEO, it is necessary to activate the corresponding environments. It is essential that these environments encompass all necessary libraries essential for their execution. The particle compressors SZ and ZFP have minimal setup requirements due to their implementation in the C programming language. Comprehensive instruction manuals on effectively performing the initial setup and executing this work without encountering any difficulties are available in the GitHub repository.

3.1.3 Input types

The data that is passed as input to the compressors are either particle or point cloud data. The most common file formats that are passed as inputs in this work are the *mmpld* file format for particle data and the *ply* file format for point cloud data. For particle data compressors like SZ [7] and ZFP [24], the input data is converted into a *binary* format that contains floating point arrays in single-precision without any header data. The tool must be adapted to accept double-precision floating points and skip headers, if present, in future work. The data is first extracted from a *mmpld* file, which is a form of representation of particle data developed to enable visualization in the MegaMol tool [14]. Data is stored in little-endian *binary* format in the *mmpld* files. This extracted data is then written into a *binary* file [14] and passed as input to the compressors. If the input is not in *binary* format or not a *mmpld* file, then it is first converted into the *binary* file format and then provided to the particle compressors. If the particle data has to be compressed using a point cloud compressor, it is first converted into a *ply* file format containing floating-point arrays of x, y, and z coordinates and then provided as input to the point cloud compressors. If the input is point cloud data in *ply* file format, it is then directly fed to the point cloud compressors without any conversion. For the PCC_GEO point cloud compressor, the input files that are passed should be voxelized. Since this work is aimed at creating a tool to compress data using different compressor techniques, the tool is designed in such a way as to adapt these file conversions and adaptations accordingly. The tool handles the file conversions irrespective of the input file format with specific extensions depending on the compressor type that is used. These formats include *mmpld, bin, ply, pcd, stl, xyz, obj,* and *off*. So, irrespective of the file format that is provided as input, the tool generates the necessary file format that is needed for the selected compressor. Also, the inputs can be provided either as a directory or a specific input file.

3.1.4 Compression and Decompression

The tool is designed in a dynamic manner to execute different types of compression techniques. Different compressors require different sets of instructions to execute them. Some values are set before the execution of the commands, and some are provided during the execution. The desired compressor that is needed for the current execution is provided as input to the tool, along with the parameters that are needed for the compressor. Based on that, the tool executes the corresponding compressor. It compresses and decompresses the files and saves them in the corresponding destination folders.

3.1.5 Generating Error metrics

In order to evaluate the compression results, the files are fed to the program that calculates the error metrics. The tool has integrated two different error metrics. One belongs to the particle data, and the other one belongs to the point cloud data. If these metrics are needed, then the respective flag is passed in the input command. These error metrics are generated, and the results are documented in a *csv* file. The error metrics that are generated after compression and decompression are essential factors in evaluating the performance of the compressors. The integrated tool generates several error metrics, including the mean squared error, peak signal-to-noise ratio, normalized root mean square error, and so on. These error metrics can be used to compare the performance of different compressors on different types of data. By using these error metrics, the performance of the compressors is evaluated from different perspectives, providing a comprehensive view of the quality of the compression.

The order of points is an essential factor in the case of particle data. To establish the correspondence, the decompressed data is sorted and arranged in a manner similar to the original dataset. Since learning-based methods, such as Octattention and PCC_GEO_CNN_V2, are used in this compression tool, the decompressed data contain points that are generated in a random order irrespective of the order of points in the original file. But the particle compressors, SZ and ZFP, that are used in the tool maintain the order of particles in the decompressed file. Having a correspondence between the original file and decompressed file is important for the particle error metrics, whereas it is not important in the case of point cloud error metrics.

Given the diverse categories of compressors utilized for particle and point cloud data, it is essential to establish a standardized basis for evaluating the efficiency of these compression methods. Three distinct sorting techniques were evaluated based on comprehensive research on various sorting methods. The application of these sorting methodologies results in the reorganization of the sequence of points in the decompressed file in accordance with the sequence of points in the original file. Various sorting techniques such as Argsort utilizing pandas, KD search tree, and minimum difference computation via MATLAB were evaluated in this work.

3.1.6 Advantages

Some of the significant shortcomings in the compressors, when executed separately, are addressed by this integrated tool. They are the command line inputs for the compressors. The inputs that are provided to the compressors are lengthy and create confusion for the user while trying to run it. However, this tool has integrated almost every detail that is needed as input to the compressor. For example, to run the particle compressors, the dataset length and dimension have to be provided as inputs. This is a problem every time as the length of the data provided should be known. But this tool calculates the length of the data and provides it as an input to the compressor directly. So this way, a large number of files can be passed to compress and decompress in a single execution. Another advantage of this tool is that it accepts a wide variety of input data types. Most of the file type conversions are in-built and taken care of by the tool. Even if there are any errors during execution, the tool provides the native commands to run the program, which helps us to debug the problem and find a solution for it. Furthermore, the tool is highly flexible and customizable, enabling users to configure various parameters and settings according to their specific needs and requirements. This allows users to fine-tune the compressor to achieve optimal results for their particular use case.

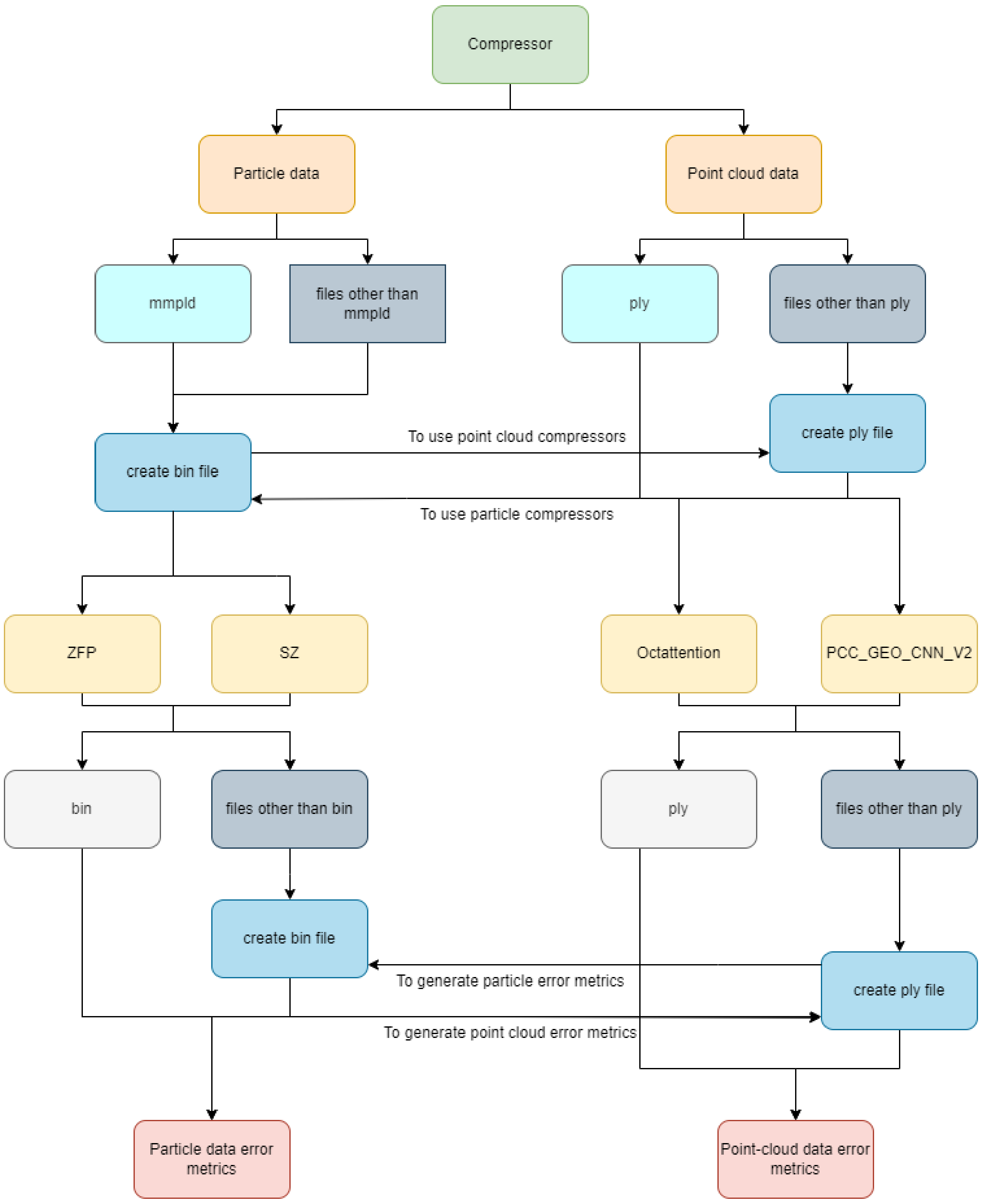


Figure 3.1: System overview of integrated compressor tool

Overall, the tool offers significant advantages over compressors executed individually, enabling users to streamline their data compression and decompression workflows and achieve improved efficiency and productivity. By simplifying the process and providing a user-friendly interface, the tool makes it easier for users to leverage the benefits of data compression and manage their data effectively.

3.2 Sorting techniques

There is a clear deviation in the order of points in the decompressed files of the point cloud compressors. It will not create fair comparison metrics if the points are not ordered according to the sequence of the points in the input file. To create this ordering, three different sorting techniques are tested, and the best one is chosen to establish the correspondence. The three different techniques that are chosen are Argsort using pandas, KD search tree, and Minimum difference calculation using MATLAB. These techniques are used to rearrange the order of points in the decompressed file to match the order of points in the original file. This is important in order to compare the quality of compression among the different compressors. By rearranging the order of points in the decompressed file, we can compare the decompressed file to the original file and determine the level of accuracy and fidelity of each compressor. Overall, the sorting techniques used in this study are useful for measuring the quality of compression among the different compressors on the same ground. The time taken for the above-mentioned three sorting algorithms has been mentioned in Table 4.21. Based on the results obtained in Figure 4.7, and Table 4.20, the Argsort technique has out-performed the other methods in several parameters in the particle error metrics. Hence the decompressed files obtained through Octattention and PCC_GEO compressors are sorted using this method, and the corresponding metrics are used as a comparison for the compressors.

3.2.1 Argsort using Pandas library

In this study, we used the Argsort technique to sort the points in the decompressed file based on their original order in the original file. An example of the argsort operation is shown in Figure 3.2. This technique is implemented using the Pandas library in Python. The Argsort technique using pandas is a data manipulation technique that sorts the data based on the values in a specific column along with its current index position. As a first step, the array of points of the original data is used to create a pandas dataframe. Then these values are sorted based on ascending order along with their index positions. Then the decompressed points are sorted in ascending order and added to the dataframe. The dataframe is then reordered once again based on the original index positions of the original data. It is fast and efficient, and it works well even for datasets with a large number of points. The pseudocode for this algorithm is provided in Chapter 5.

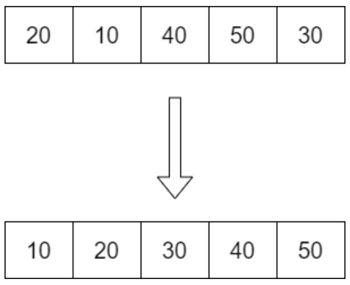


Figure 3.2: Example of Argsort

3.2.2 KD-Search Tree

The KD search tree is a particular kind of binary search tree that is especially efficient for searching high-dimensional data in spaces with many dimensions. A sequence of axis-aligned hyperplanes is used to divide the data space in this modification of the conventional binary search tree. These trees are efficiently built and searched using the KD-Search Tree algorithm. An example of a KD-Search tree is given in Figure 3.3.

A KD-Search Tree is built by first dividing the data at a particular axis. Any dimension in the data space may be used for this, although it is usually best to choose the one having the widest range of values. The next step is to choose a pivot point on that axis that separates the data set into two halves. The left subtree contains all data points with values less than or equal to the pivot, whereas the right subtree contains all data points with values higher than the pivot. Then, recursively go through this procedure for each of the two subtrees, switching the axis at each level. The KD-Search Tree is searched for a query point by comparing it with the pivot point in the current node starting at the tree's root. We recursively search the left subtree if the query point is less than or equal to the pivot along the current axis. If not, the appropriate subtree is searched. This procedure is repeated until a leaf node is reached, which either has a data point that perfectly matches the query point or has a data point that is the data set's closest match to the query point.

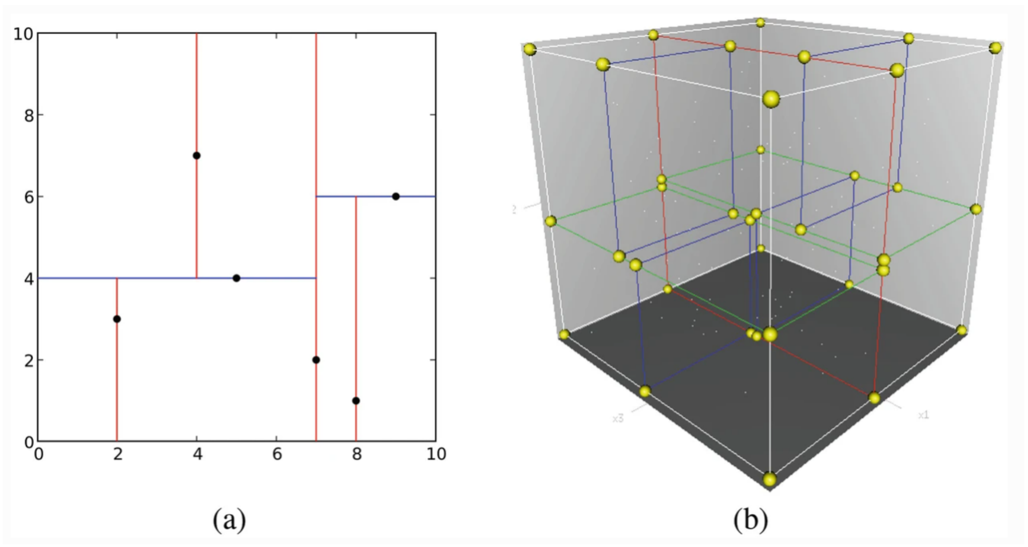


Figure 3.3: Example of 2D and 3D KD-Search Trees[26]

The KD search tree technique is used in this study to sort the points in the decompressed file based on their closest proximity to the original points in the original file. This technique is useful for datasets with a large number of points and works well for sorting points that are close to each other in the original file.

3.2.3 Minimum difference calculation using MATLAB

The minimum difference calculation technique using MATLAB is a mathematical technique used to calculate the minimum difference between two sets of points. In this study, we used this technique to calculate the minimum difference between the points in the original file and the points in the decompressed file. The technique works by calculating the Euclidean distance between each point in the original file and its nearest point in the decompressed file. This technique is useful for datasets with a large number of points and works well for sorting points that are not close to each other in the original file. The pseudocode for this algorithm is provided in Chapter 5.

3.3 Renderings

The rendered images shown in Figure 4.4 and Figure 4.5 provide an effective way to compare the quality of compression among the different compressors. The renderings were created using the Cloud Compare software. The Airplane data contains a large number of points, and it is impressive to see how well the decompressed results match the original file. This indicates that the compression algorithms used in this study are highly effective in maintaining the quality of the point cloud data. However, the results for the particle data, Nozzle, are not as consistent. While the particle compressors performed well in maintaining the quality of the data, the point cloud compressors showed a significant loss of data in the decompressed results. This can be seen in the differences between the original and decompressed images in Figure 4.5.

3.4 Datasets

The choice of datasets for the evaluation is crucial as it can significantly impact the performance and effectiveness of the algorithms. It is important to select appropriate datasets that are representative of real-world scenarios and can provide meaningful insights and results. Two different categories of datasets are used in this work.

In our evaluation, the datasets mentioned in Table 3.1 are used for the experiments. The particle datasets such as Fluid, Nozzle, Riemann [15] and Nozzle [9] are used. The point cloud datasets are from ModelNet40 [49]. The choice of point cloud datasets from ModelNet40 is appropriate as it is a widely used benchmark dataset for evaluating 3D object recognition and classification algorithms. It contains models of objects of 40 different categories, and the models from Airplane, Car, and Chair are chosen from that. By choosing models from three different categories, the evaluation can be more comprehensive and diverse, enabling the analysis of the compressors in a wide manner. Also, the Stanford Bunny dataset, from the Stanford 3D Scanning Repository, which is the most commonly used point cloud data in any point cloud data analysis, is chosen in this experiment. One of the point cloud compressors (PCC_GEO_CNN_V2) only accepts input data that are voxelized. Hence, these datasets are voxelized in such a way that the total number of points in the raw dataset is retained so that there is not much loss of information. The usage of a variety of dataset types in the assessment aids in determining the robustness and generalizability of the compression methods.

Because particle data and point cloud data are fundamentally different in nature, algorithms that perform well on both types of datasets are more versatile and adaptable to different applications.

Type	**Dataset**	**Shape**	**Type**	**Size (in MB)**
Particle	Fluid	29999997x3	FP32	340
	Nozzle	1550333x3	FP32	18.6
	Laser	199885091x3	FP32	2400
	Riemann	306112864x3	FP32	3700
Point cloud	Airplane	93453x3	FP32	1.1
	Stanford Bunny	185253x3	FP32	2.2
	Car	265270x3	FP32	3.2
	Chair	191163x3	FP32	2.3

Table 3.1: Desciption of datasets used

4 Evaluation

Several error metrics have been used to analyze the results of different compressors with the goal of finding out the best error metric that can be used to evaluate both particle and point cloud data compressors. In this section, the different quality metrics that are used to analyze the compressors in the tool are explained. It is followed by briefings on results obtained for different experiments, along with renderings and sorting techniques.

4.1 Quality Metrics

4.1.1 Particle data error metrics

In this category, the error was evaluated based on the results of six different metrics. They are: (i).Maximum absolute error (ii).Maximum relative error (iii).Maximum point-wise relative error (iv).PSNR (v).NRMSE (vi).Pearson correlation coefficient (vii).Compression Ratio (CR)

4.1.1.1 Maximum absolute error

It denotes the maximum difference value between the point in the decompressed file Vo and the corresponding point in the original file Vi. The maximum absolute error value is given by

$$Maximum\ Absolute\ Error = max(Vo - Vi), \forall Vo \tag{4.1}$$

where $o = 0, 1, ...n$ and n is the total number of points in the decompressed file.

4.1.1.2 Maximum relative error

It denotes the maximum difference value between the point in the decompressed file Vo and the corresponding point in the original file Vi, divided by the difference between the maximum and minimum of all points in the original file. The maximum relative error value is given by

$$Maximum\ Relative\ Error = max\left(\frac{Vo - Vi}{max(Vi) - min(Vi)}\right), \forall Vo \tag{4.2}$$

where $o = 0, 1, ...n$ and n is the total number of points in the decompressed file.

4.1.1.3 Maximum point-wise relative error

It denotes the maximum difference value between the point in the decompressed file Vo and the corresponding point in the original file Vi, divided by Vi. It is given by

$$Maximum\ Pointwise\ Relative\ Error = max\left(\frac{Vo - Vi}{Vi}\right), \forall Vo \tag{4.3}$$

where $o = 0, 1, ...n$ and n is the total number of points in the decompressed file.

4.1.1.4 PSNR

This metric compares the decompressed file and the original file based on the RMSE relative to the peak size of the signal. It is given by

$$Peak\ Signal\ to\ Noise\ Ratio(PSNR) = 20 * \log_{10}\left(\frac{max(Vi) - min(Vi)}{RMSE}\right) \tag{4.4}$$

$$Root\ Mean\ Square\ Error\ (RMSE) = \sqrt{\frac{1}{|V|} * \sum_{i=1}^{|V|} |Vi - Vo|^2} \tag{4.5}$$

where $i = 0, 1, ...n$ and n is the total number of points in the original file.

4.1.1.5 NRMSE

Due to the diversity of variables, the root mean square error is normalized by dividing the difference between the maximum and minimum values of the points of the original dataset. It is given by

$$Normalized\ Root\ Mean\ Square\ Error\ (NRMSE) = \frac{RMSE}{max(Vi) - min(Vi)} \tag{4.6}$$

where $i = 0, 1, ...n$ and n is the total number of points in the original file.

4.1.1.6 Pearson correlation coefficient

The Pearson correlation coefficient is a statistical measure employed to evaluate the intensity and direction of a linear relationship between two variables. The variables in question could be the x and y coordinates. The Pearson correlation coefficient, denoted by ρ, varies from -1 to 1, where -1 represents a perfect negative correlation, 0 represents no correlation, and 1 represents a perfect positive correlation. A positive correlation indicates that as one variable increases, the other also tends to increase. In contrast, a negative correlation indicates that as one variable increases, the other tends to decrease. The Pearson correlation coefficient is given by

$$Pearson\ Correlation\ Coefficient\ (\rho) = \frac{Cov(Vi, Vo)}{\sigma_i * \sigma_o} \tag{4.7}$$

$$Cov(Vi, Vo) = \frac{\sum(Vi - \hat{Vi})(Vo - \hat{Vo})}{n} \tag{4.8}$$

where $i = 0, 1, ...n$ and n is the total number of points in the original file, σ_i and σ_o are the standard deviations of points of the original and decompressed files, $\hat{Vi}$ and $\hat{Vo}$ are the mean value of the points of the original and decompressed files.

4.1.1.7 Compression Ratio

The compression ratio is a measurement of the quantity of compression applied to a data set. It is typically expressed as a ratio of the original data size to the compressed data size. The compression ratio quantifies the degree of compression obtained by a compression algorithm and can be used to compare the efficacy of various compression techniques. It is given by

$$Compression\ Ratio = \frac{Size\ of\ original\ file}{Size\ of\ compressed\ file} \tag{4.9}$$

It is essential to note that the compression ratio is only sometimes the most accurate indicator of an algorithm's efficiency. Sometimes, a compression algorithm may accomplish a high compression ratio but at the expense of substantially more prolonged compression and decompression times or degraded data quality. When evaluating the efficacy of a compression algorithm, the compression ratio should be considered alongside other factors mentioned above.

4.1.2 Point cloud data error metrics

In this category, the error was evaluated based on the results of the D1 metric as per the MPEG standards [16] for point cloud geometry compression. The D1 metric is a point-to-point distance metric and is defined as the average squared distance between each point in the first point cloud, say A, and its nearest neighbor in the second one, say B, as shown in Figure 4.1. It compares the original data with the decoded data and provides a numerical value [45]. For each point V_i in file A, a corresponding point $\hat{Vi}$ in file B is identified based on the nearest neighbor.

It is given by

$$Mean\ Square\ Error\ (MSE),\ e_{A,B} = \frac{1}{|V|}\sum_{i=1}^{|V|} ||E(V_i, \hat{V_i})||_2^2 \tag{4.10}$$

where E is the error vector between the points V_i and $\hat{V_i}$. Since it was difficult to interpret and understand the values from Mean Square Error (MSE), it is converted to a PSNR value which would normalize the metrics with respect to a peak value p. It is given by

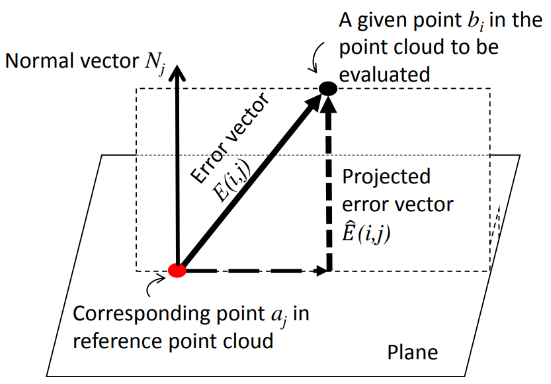

A given point b_i in the point cloud to be evaluated
Normal vector N_j
Error vector $E(i,j)$
Projected error vector $\hat{E}(i,j)$
Corresponding point a_j in reference point cloud
Plane

Figure 4.1: Point-to-point (D1) error metric [45]

$$PSNR_{(A,B)} = 10 * log_{10}\left(\frac{p^2}{e_{A,B}}\right) \tag{4.11}$$

It is obtained in three different variations: (I). MSE 1, PSNR 1 (ii). MSE 2, PSNR 2 (iii). MSE F , PSNR F

4.1.2.1 MSE 1, PSNR 1

This value is obtained by considering the original file as A and the decompressed file as B.

4.1.2.2 MSE 2, PSNR 2

This value is obtained by considering the original file as B and the decompressed file as A.

4.1.2.3 MSE F, PSNR F

This value is the maximum of MSE 1 and MSE 2 and the minimum of PSNR 1 and PSNR 2

4.1.3 Compression/Decompression times

The duration of compression and decompression operations performed by a specific compressor is a significant factor in evaluating its efficacy. Each compression technique possesses a distinct priority with regard to the acceleration of compression and decompression processes. The variance is contingent upon the magnitude of the data that is utilized as input for the compression algorithms. Typically, as evidenced by several studies [7, 10, 24, 33], a higher compression ratio achieved by a compressor is associated with a proportional increase in the time required for compression and decompression, resulting in a significant execution time. The optimal compressor in this metric is characterized by a shorter compression and decompression duration, coupled with a high compression ratio value and minimal data loss during the decompression process.

4.2 Results

The experiments are performed in the CPU Intel Xeon Gold 5122 (x2) 8c/16t @ 3,6 GHz processor with RAM 384GB and GPU NVIDIA Tesla V100-PCIE-32GB and also using NVIDIA GeForce RTX 2060 6 GB. The compression and decompression results that are generated for the particle and point cloud datasets using different compressors are provided in this section. It is essential for the particle error metrics to have the same order of points in the decompressed file as that of the original file. But the outputs of the point cloud compressors are reordered, as they are learning-based models. Without reordering, there were huge margins of error between the particle compressors and point cloud compressors in terms of particle error metrics. Since that would not make a fair comparison, the decompressed files of the point cloud compressors are reordered accordingly using the best-performing sorting algorithm. Based on the analysis in Table 4.20, the Argsort sorting method using the Pandas library is chosen and applied to the decompressed output of the point cloud compressors.

The particle compressors and point cloud compressors exhibit a significant variance in error margin with respect to particle error metrics, even after applying the sorting algorithm. For the sorting algorithm and particle error metrics calculation to be applied, as stated already, it is necessary that the number of points in both the decompressed array and the original array is identical. In the context of the Octattention compressor, it is observed that the decompressed array possesses a smaller number of points compared to the original array. As a result, it is necessary to slice the original array to match the length of the decompressed array. This particular step may have resulted in sporadic outliers within the particle error metrics pertaining to the Octattention compressor. Conversely, it has been observed that the PCC_GEO compressor generates a greater number of points in the array after decompression compared to the original array. The present scenario involves slicing the decompressed array to match the length of the original array, followed by the computation of particle error metrics. The impact of this step on the point cloud data was negligible, as the particle error metrics for PCC_GEO exhibited slightly superior performance compared to Octattention. But it had a significant influence on the particle data, as the PCC_GEO's particle error metrics were much greater than those of the Octattention.

This clarifies certain anomalies observed in the particle error metrics pertaining to the point cloud compressors. The particle error metrics are computed by evaluating individual points from the decompressed file against their corresponding points in the original file based on their respective index positions. The point cloud error metrics involve the simultaneous consideration of all the points, which represent the (x,y,z) coordinates. In MSE1, the original file is kept as point cloud A, and the decompressed file is kept as point cloud B and vice versa to calculate MSE2. The nearest neighbors are calculated for these points before computing the MSE values. This results in shallow MSE values and high PSNR values in the point cloud error metrics. The reason for this is that, with respect to correlation, the association among points in the point cloud data is usually considerably more robust than the association among particles in particle data. Hence, it is highly probable that utilizing a single point for the purpose of computing errors may lead to increased deviations in particle error metrics for the point cloud data.

4.2.1 Point Cloud data

The results indicate that particle compressors exhibit superior performance in terms of both point cloud error metrics and particle error metrics when applied to point cloud data. This comes with the downside of low compression ratios for the particle compressors SZ and ZFP, in comparison to the high compression ratios of the point cloud compressors Octattention and PCC_GEO.

4.2.1.1 Airplane

In the context of the Airplane dataset, it was observed that the compression ratios achieved by the particle compressors, namely SZ and ZFP, were relatively low, 2.57 and 2.01, respectively. However, the point cloud compressors, Octattention and PCC_GEO, demonstrated significantly higher compression ratios, 81.04 and 254.18, respectively. The SZ compressor produced a 0.5 increase in compression ratio compared to that of ZFP for nearly the exact value of NRMSE. The metrics values are mostly dominated by the ZFP compressor for this dataset in Table 4.1 and Table 4.2, with a compromise in the compression ratio. The Octattention compressor's maximum pointwise relative error is notably high, which may be associated with the requisite slicing procedure utilized to sort the decompressed file and produce the particle error metrics.

Compressor	MSE1	MSE1 PSNR	MSE2	MSE2 PSNR	MSEF	MSEF PSNR
SZ	0.02125	81.69514	0.02125	81.69514	0.02125	81.69514
ZFP	0.01735	82.57489	0.01735	82.57489	0.01735	82.57489
Octattention	0.25330	70.93242	0.19253	72.12365	0.25330	70.93242
PCC_GEO	0.23075	71.33736	0.26574	70.72409	0.26574	70.72409

Table 4.1: Point cloud error metrics - Airplane

Compressor	Avg abs error	Max abs error	Max Relative Error	Max PW Relative Error	PSNR	NRMSE	Pearson Correlation coefficient	CR
SZ	0.06778	0.25378	0.00050	0.00050	75.66604	0.00016	1.00000	2.57
ZFP	0.05947	0.25000	0.00049	0.14844	76.54580	0.00015	1.00000	2.01
Octattention	10.20236	82.43484	0.25420	48.24E+4	26.23252	0.04879	0.97673	171.86
PCC_GEO	3.39695	33.0000	0.06458	1.0000	38.56017	0.01180	0.99914	254.18

Table 4.2: Particle error metrics - Airplane

4.2.1.2 Car

For the Car point cloud, the highest compression ratio was achieved using Octattention (457.14), and the highest PSNR was achieved using SZ (84.80632 / 78.77723 in the point cloud and particle error metrics, respectively) in Table 4.3 and Table 4.4. Both the particle compressors resulted in low average absolute errors (less than 0.06), and the point cloud compressors, Octattention and PCC_GEO, resulted in high error values in Table 4.4 as stated already. The SZ compressor yielded a significantly low NRMSE value of 0.00012 for this dataset, thereby leading to a notable increase in PSNR and a 0.6 rise in compression ratio in comparison to ZFP.

Compressor	**MSE1**	**MSE1 PSNR**	**MSE2**	**MSE2 PSNR**	**MSEF**	**MSEF PSNR**
SZ	0.01038	84.80632	0.01038	84.80632	0.01038	84.80632
ZFP	0.01746	82.54778	0.01746	82.54778	0.01746	82.54778
Octattention	0.26483	70.73908	0.12781	73.90300	0.26483	70.73908
PCC_GEO	0.38204	69.14758	0.43289	68.60494	0.43289	68.60494

Table 4.3: Point cloud error metrics - Car

Compressor	**Avg abs error**	**Max abs error**	**Max Relative Error**	**Max PW Relative Error**	**PSNR**	**NRMSE**	**Pearson Correlation coefficient**	**CR**
SZ	0.04048	0.25314	0.00050	0.00050	78.7772	0.00012	1.00000	2.49
ZFP	0.05969	0.25000	0.00049	0.14844	76.5187	0.00015	1.00000	1.92
Octattention	0.70301	3.14572	0.40538	7.40577	17.65647	0.13097	0.87555	457.14
PCC_GEO	10.75119	45.00000	0.08806	2.00000	30.99174	0.02821	0.99797	262.30

Table 4.4: Particle error metrics - Car

4.2.1.3 Chair

For the Chair data, the highest compression ratio was achieved using Octattention (242.11). SZ achieved the highest PSNR values once again in both metrics in Table 4.5 and Table 4.6. While both the particle compressors result in a comparatively low average absolute error (less than 0.06), the point cloud compressors, Octattention and PCC_GEO, resulted in high error values in Table 4.6 as expected. The observed high maximum pointwise relative error for the Octattention compressor could be attributed to the necessary slicing procedure for sorting the decompressed file and generating the particle error metrics.

Compressor	**MSE1**	**MSE1 PSNR**	**MSE2**	**MSE2 PSNR**	**MSEF**	**MSEF PSNR**
SZ	0.01517	83.15882	0.01517	83.15882	0.01517	83.15882
ZFP	0.01748	82.54447	0.01748	82.54447	0.01748	82.54447
Octattention	0.25032	70.98381	0.24972	70.99411	0.25032	70.98381
PCC_GEO	0.19038	72.17247	0.20487	71.85393	0.20487	71.85393

Table 4.5: Point cloud error metrics - Chair

Compressor	Avg abs error	Max abs error	Max Relative Error	Max PW Relative Error	PSNR	NRMSE	Pearson Correlation coefficient	CR
SZ	0.05235	0.25455	0.00050	0.00050	77.12973	0.00014	1.00000	2.30
ZFP	0.05973	0.25000	0.00049	0.25000	76.51538	0.00015	1.00000	1.92
Octattention	5.46234	18.12195	0.00449	19.91E+4	55.50159	0.00168	0.99999	242.11
PCC_GEO	3.84525	15.00000	0.02935	1.00000	39.50627	0.01058	0.99960	5.71

Table 4.6: Particle error metrics - Chair

4.2.1.4 Stanford Bunny

For the Stanford Bunny point cloud, the highest compression ratio was achieved using Octattention (192.98). However, the highest PSNR was achieved using SZ (83.43148 / 77.40239 in the point cloud and particle error metrics, respectively) in Table 4.7 and Table 4.8. While both particle compressors result in a comparatively low average absolute error (less than 0.06), the point cloud compressors Octattention and PCC_GEO once again yielded high errors in Table 4.8.

Compressor	MSE1	MSE1 PSNR	MSE2	MSE2 PSNR	MSEF	MSEF PSNR
SZ	0.01425	83.43148	0.01425	83.43148	0.01425	83.43148
ZFP	0.01749	82.54087	0.01749	82.54087	0.01749	82.54087
Octattention	0.25211	70.95285	0.19112	72.15571	0.25211	70.95285
PCC_GEO	0.20321	71.88930	0.21601	71.62396	0.21601	71.62396

Table 4.7: Point cloud error metrics - Stanford Bunny

Compressor	Avg abs error	Max abs error	Max Relative Error	Max PW Relative Error	PSNR	NRMSE	Pearson Correlation coefficient	CR
SZ	0.05174	0.25302	0.00050	0.00050	77.40239	0.00013	1.00000	2.46
ZFP	0.05977	0.25000	0.00049	0.16016	76.51178	0.00015	1.00000	1.93
Octattention	5.82912	12.33915	0.04380	19.46E+3	32.52498	0.02365	0.99569	192.98
PCC_GEO	1.25962	6.00000	0.01174	0.50000	49.25540	0.00345	0.99996	188.03

Table 4.8: Particle error metrics - Stanford Bunny

4.2.1.5 Compression and Decompression times for point cloud datasets

The compression and decompression times presented in Table 4.9 for point cloud data indicate that the particle data compressors exhibit the most speedy performance. In terms of compression times, the ZFP compressor exhibits a slight advantage over SZ. Octattention outperforms PCC_GEO in terms of compression times for most of the datasets, as shown in Figure 4.2. The decompression time of PCC_GEO is comparatively superior to that of Octattention. Octattention exhibits better performance than PCC_GEO when considering the total time period of both compression and decompression processes, with the exception of the Chair dataset.

Dataset	**Compressor**	**Compression Time (in sec)**	**Decompression Time (in sec)**
Airplane	sz	0.06162	0.06250
	zfp	0.04309	0.04655
	octattention	12.51126	32.49205
	pcc_geo	114.61089	22.32992
Car	sz	0.09168	0.11881
	zfp	0.06261	0.07779
	octattention	11.27894	10.75662
	pcc_geo	228.36266	28.70321
Chair	sz	0.09092	0.11483
	zfp	0.06788	0.06607
	octattention	261.30471	1545.45663
	pcc_geo	170.25710	28.15363
Stanford Bunny	sz	0.07737	0.08627
	zfp	0.05287	0.06268
	octattention	19.96328	65.93031
	pcc_geo	202.14645	32.87222

Table 4.9: Compression and Decompression Times - Point cloud data

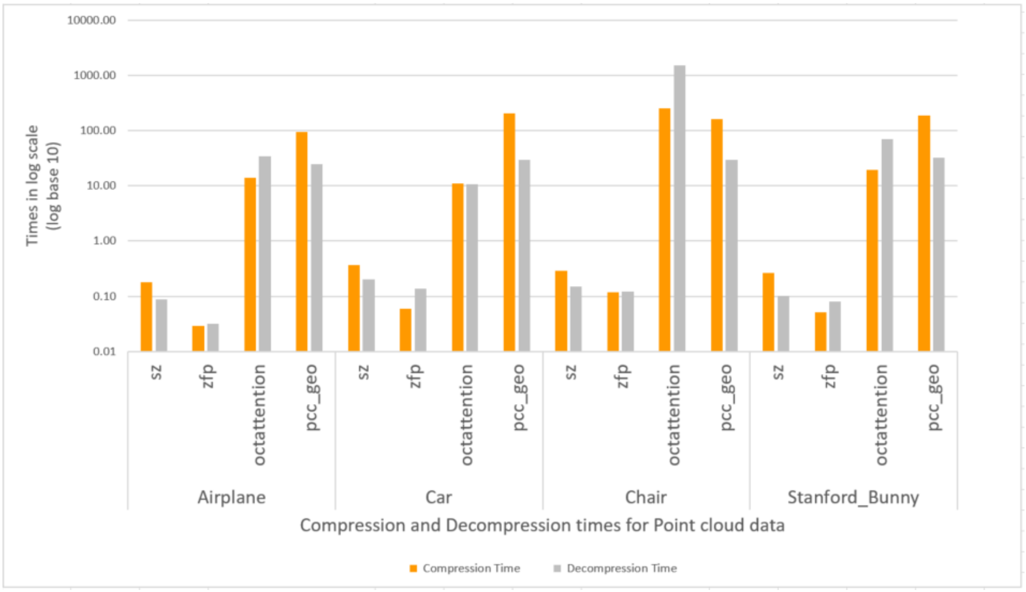


Figure 4.2: Compression and Decompression Times - Point cloud data

4.2.2 Particle data

For the particle data, the performances of the particle compressors are better in both point cloud error metrics and particle error metrics. This comes with the drawback of low compression ratios for them. The size of the input file for the Octattention compressor should be less than 300MB as per its author [10]. When tested initially for the Fluid dataset (360MB), it was able to compress and decompress but took exponentially high compression and decompression times. So in this work, it was tested for the Fluid and Nozzle datasets. But not able to run it for larger datasets like Laser and Riemann. Even though there is no explicit size limitation for the PCC_GEO compressor, when tested for the particle datasets, Riemann and Laser, the compressor was running for a long time without any improvements. For the other two particle datasets, Fluid and Nozzle, the performance of the PCC_GEO compressor is not the best. As the PCC_GEO compressor produces more points in the decompressed file than the original file, it cannot be directly used for calculating the particle error metrics and applying the sorting algorithm, as the total number of points in both files must be the same for these two steps. So, the decompressed file is sliced to the same number of points in the original file, and then the sorting technique is applied before calculating the particle error metrics.

4.2.2.1 Fluid

For the Fluid data, the highest compression ratio was achieved using PCC_GEO (64.29). However, the highest PSNR was achieved using SZ (74.52318) in Table 4.11 and using Octattention (70.98880) in Table 4.10. This high value of PSNR for the particle compressors in particle error metrics is a direct cause of very low values of NRMSE (1.9E-4 and 3.5E-4 for SZ and ZFP, respectively) in Table 4.11. The error values for the Octattention and PCC_GEO are high in the particle error metrics even after sorting the decompressed files.

Compressor	MSE1	MSE1 PSNR	MSE2	MSE2 PSNR	MSEF	MSEF PSNR
SZ	0.34840	69.54798	0.17114	72.63516	0.34840	69.54798
ZFP	0.42772	68.65710	0.32618	69.83409	0.42772	68.65710
Octattention	0.25003	70.98880	0.23033	71.34527	0.25003	70.98880
PCC_GEO	2.89722	60.34891	5.48212	57.57924	5.48212	57.57924

Table 4.10: Point cloud error metrics - Fluid

Compressor	Avg abs error	Max abs error	Max Relative Error	Max PW Relative Error	PSNR	NRMSE	Pearson Correlation coefficient	CR
SZ	0.20718	1.81763	0.00100	0.00100	74.52318	0.00019	1.00000	6.02
ZFP	0.50598	3.53776	0.00194	5.522E+4	69.04273	0.00035	1.00000	3.14
Octattention	20.13453	241.1558	0.13207	1.00352	32.85151	0.02277	0.99702	51.43
PCC_GEO	528.4666	3072.000	3.00293	1.00000	0.45166	0.94932	0.78359	64.29

Table 4.11: Particle error metrics - Fluid

4.2.2.2 Nozzle

For the Nozzle data, the highest compression ratio was achieved using Octattention (35.59). The NRMSE values are almost the same for both particle compressors (1.1E-4 for SZ and ZFP, respectively). There is little variation in the PSNR values (78.98459 / 79.02946 for SZ and ZFP, respectively) in Table 4.13. The PSNR values of ZFP are slightly higher than SZ in both metrics (87.29506 / 79.02946 for point cloud error metrics and particle error metrics, respectively) in Table 4.12 and Table 4.13. While both particle compressors result in a comparatively low average absolute error (less than 0.03), the point cloud compressor Octattention and PCC_GEO produce higher error values in the particle error metrics even after applying the sorting algorithm.

Compressor	MSE1	MSE1 PSNR	MSE2	MSE2 PSNR	MSEF	MSEF PSNR
SZ	0.00591	87.25018	0.00591	87.25018	0.00591	87.25018
ZFP	0.00585	87.29506	0.00585	87.29506	0.00585	87.29506
Octattention	0.25026	70.98480	0.23309	71.29350	0.25026	70.98480
PCC_GEO	6.54309	56.81089	17.30187	52.58780	17.30187	52.58780

Table 4.12: Point cloud error metrics - Nozzle

Compressor	Avg abs error	Max abs error	Max Relative Error	Max PW Relative Error	PSNR	NRMSE	Pearson Correlation coefficient	CR
SZ	0.03467	0.19586	0.00050	0.00050	78.98459	0.00011	1.0000	4.66
ZFP	0.03477	0.21243	0.00054	35.93922	79.02946	0.00011	1.0000	2.29
Octattention	1.02552	4.99034	0.01263	0.99530	48.39891	0.00381	0.99982	35.59
PCC_GEO	213.2935	1024.000	1.000978	1.00000	11.07014	0.27957	0.91157	6.64

Table 4.13: Particle error metrics - Nozzle

4.2.2.3 Laser

The point cloud compressors were not able to compress this dataset since it was very large (around 2.4GB). The Octattention compressor cannot be run for this dataset because of memory limitations. It occupied the entire RAM of 377GB while trying to compress this dataset and crashed eventually. So, the metrics cannot be generated using both the point cloud compressors for this dataset. Hence, for the Laser data, the highest compression ratio was achieved using SZ (5.75) in Table 4.15. The NRMSE values were almost the same for both particle compressors (1.1E-4 for SZ and ZFP) in the particle error metrics. However, the PSNR values of ZFP are slightly higher than SZ in both metrics (63.94795 / 79.55520 for point cloud error metrics and particle error metrics, respectively). Thus, it can be inferred from the data presented in Table 4.14 and Table 4.15 that there is no significant difference in the precision of the two compressors. With regard to the compression ratio, SZ demonstrates the highest level of performance as a compressor for this dataset.

Compressor	MSE1	MSE1 PSNR	MSE2	MSE2 PSNR	MSEF	MSEF PSNR
SZ	1.54177	63.08853	1.18500	64.23154	1.54177	63.08853
ZFP	1.26496	63.94795	1.15366	64.34793	1.26496	63.94795

Table 4.14: Point cloud error metrics - Laser

Compressor	Avg abs error	Max abs error	Max Relative Error	Max PW Relative Error	PSNR	NRMSE	Pearson Correlation coefficient	CR
SZ	0.41442	3.13086	0.00050	0.00050	78.88121	0.00011	1.00	5.75
ZFP	0.52274	3.75519	0.00060	5.313E+5	79.55520	0.00011	1.00	2.59

Table 4.15: Particle error metrics - Laser

4.2.2.4 Riemann

The Riemann dataset, which was approximately 3.7GB in size, proved to be too large for the point cloud compressors to compress effectively. Consequently, the generation of error metrics for this dataset using the point cloud compressors was not feasible. The utilization of SZ resulted in the attainment of the maximum compression ratio (7.79). The normalized root mean square error (NRMSE) values for the two particle compressors, SZ and ZFP, are 1.7E-4 and 1.9E-4, respectively. Regarding the PSNR value, it can be observed that SZ (75.16564) exhibits a slightly higher value than ZFP in Table 4.17, whereas ZFP (67.60569) presents a slightly higher PSNR value than SZ in Table 4.16. Therefore, it is evident from Table 4.17 and Table 4.17 that the accuracy of the two compressors is not significantly different. In relation to the compression ratio, SZ exhibits the most superior performance as a compressor for this particular dataset.

Compressor	MSE1	MSE1 PSNR	MSE2	MSE2 PSNR	MSEF	MSEF PSNR
SZ	0.86238	65.61174	0.42651	68.66944	0.86238	65.61174
ZFP	0.54488	67.60569	0.45059	68.43089	0.54488	67.60569

Table 4.16: Point cloud error metrics - Riemann

Compressor	Avg abs error	Max abs error	Max Relative Error	Max PW Relative Error	PSNR	NRMSE	Pearson Correlation coefficient	CR
SZ	0.41344	3.39014	0.00099	0.00099	75.16564	0.00017	1.00	7.79
ZFP	0.50239	3.64331	0.00107	1.798E+6	74.50244	0.00019	1.00	2.83

Table 4.17: Particle error metrics - Riemann

4.2.2.5 Compression and Decompression times for particle datasets

Based on the compression and decompression times in Table 4.18, the particle data compressors are clearly the fastest without any doubt, as shown in Figure 4.3. ZFP demonstrates superior performance in terms of compression and decompression times, clearly outpacing other methods. In contrast, PCC_GEO and Octattention's compression and decompression times significantly surpass the particle compressors by several orders of magnitude. Thus, it exhibits lower efficiency compared to the particle compressors in this category.

Dataset	Compressor	Compression Time (in sec)	Decompression Time (in sec)
Fluid	sz	3.75466	5.40148
	zfp	3.65877	5.31007
	octattention	6.44228	27503.87835
	pcc_geo	57820.46577	292.97576
Nozzle	sz	0.23120	0.30484
	zfp	0.23961	0.32285
	octattention	281.67406	1436.65434
	pcc_geo	36096.49541	397.22808
Laser	sz	29.05339	38.37580
	zfp	28.07034	41.26340
Riemann	sz	38.28582	57.23078
	zfp	37.79854	56.95882

Table 4.18: Overview of analysis of different compressor

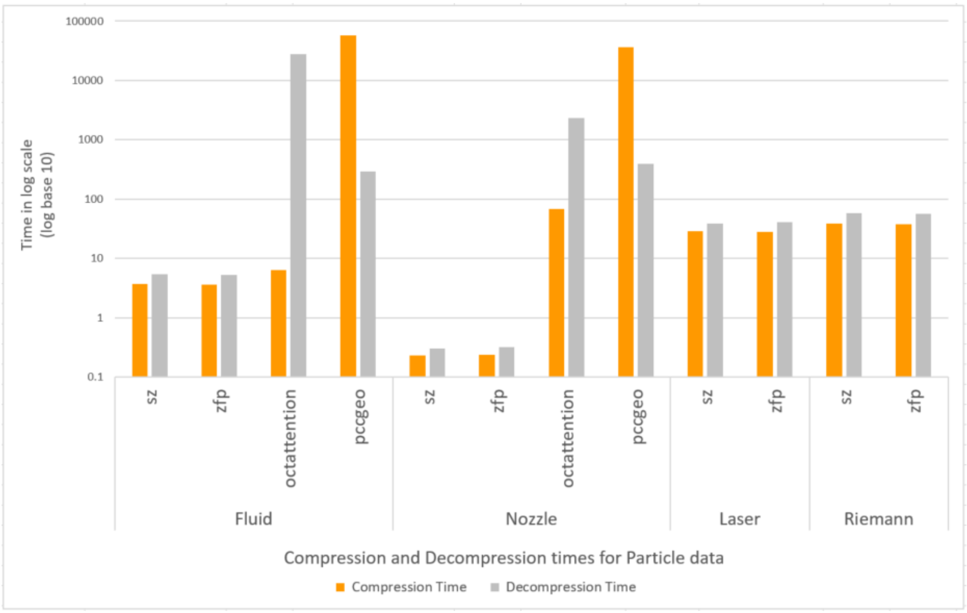


Figure 4.3: Compression and Decompression Times - Particle data

4.2.3 Similar range of NRMSE

The normalized root mean square error (NRMSE) can be regarded as a metric for evaluating the precision of the compression method. The SZ and ZFP compressors offer options for compressing the dataset in a manner that can achieve the desired level of NRMSE. A comparative analysis could potentially yield valuable insights regarding the compressor's efficacy with respect to particle metrics and point cloud metrics. The PSNR values of the particle compressors SZ and ZFP for the decompressed data in Table 4.19a are lower than those of the PCC_GEO compressor, despite having a similar range of NRMSE values in Table 4.19b. The comparative evaluation of compressors based on point cloud metrics reveals that PCC_GEO exhibits superior performance in comparison to the other two compressors. The rendered decompressed files are shown in Figure 4.4b and Figure 4.4d. This observation is attributed to the fact that the data being tested in Table 4.19b is a point cloud. For the particle data used in this study, high precision is an important factor. So it does not make sense to compress them with higher NRMSE values for this kind of comparison between the compressors.

Compressor	**MSE1**	**MSE1 PSNR**	**MSE2**	**MSE2 PSNR**	**MSEF**	**MSEF PSNR**
SZ	5.96954	57.20931	22.03440	51.53771	22.03440	51.53771
ZFP	1.31142	63.79132	15.79825	52.98263	15.79825	52.98263
PCC_GEO	0.23075	71.33736	0.26574	70.72409	0.26574	70.72409

(a) Point cloud error metrics

Compressor	**Avg abs error**	**Max abs error**	**Max Relative Error**	**Max PW Relative Error**	**PSNR**	**NRMSE**	**Pearson Correlation coefficient**	**CR**
SZ	5.17221	10.3438	0.01955	10.05084	38.94686	0.01129	0.99849	7.63
ZFP	4.39566	25.7500	0.04806	11.5000	39.64702	0.01041	0.99874	4.02
PCC_GEO	3.39695	33.000	0.06458	1.0000	38.56017	0.01180	0.99914	254.18

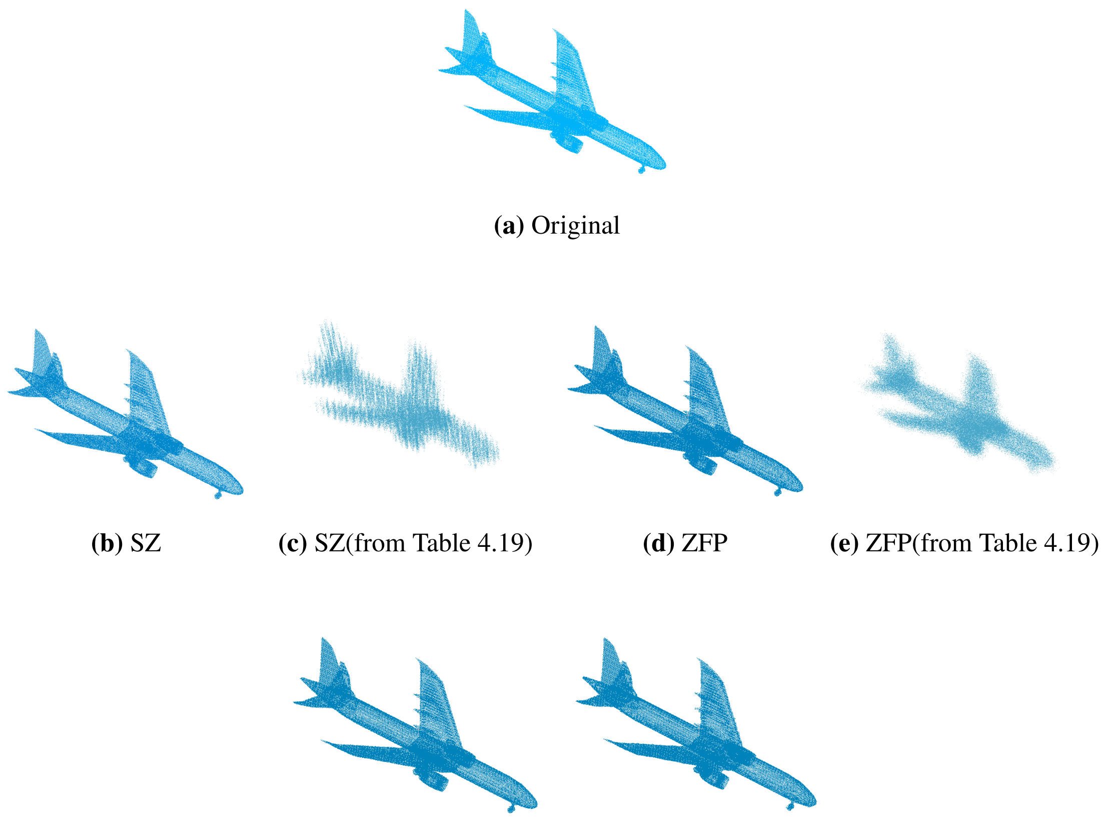
(b) Particle error metrics

Table 4.19: Airplane dataset: Test for performance in similar NRMSE range

4.2.4 Renderings of decompressed files

4.2.4.1 Airplane

Based on the renderings of the Airplane decompressed files of all four compressors in Figure 4.4, it is evident that there is not much difference between the original (Figure 4.4a) and the decompressed files (Figure 4.4b, Figure 4.4d, Figure 4.4f, Figure 4.4g). Hence, it can be inferred from these renderings that all four compressors have performed well for the point cloud data in the case of visualization. Figure 4.4c and Figure 4.4e represent the decompressed files of SZ and ZFP compressor from Table 4.19.



(a) Original

(b) SZ **(c)** SZ(from Table 4.19) **(d)** ZFP **(e)** ZFP(from Table 4.19)

(f) Octattention **(g)** PCC_GEO_CNN_V2

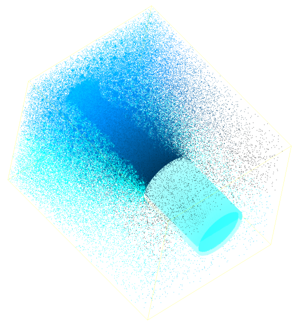
Figure 4.4: Renderings of Airplane data using different compressors

4.2.4.2 Nozzle

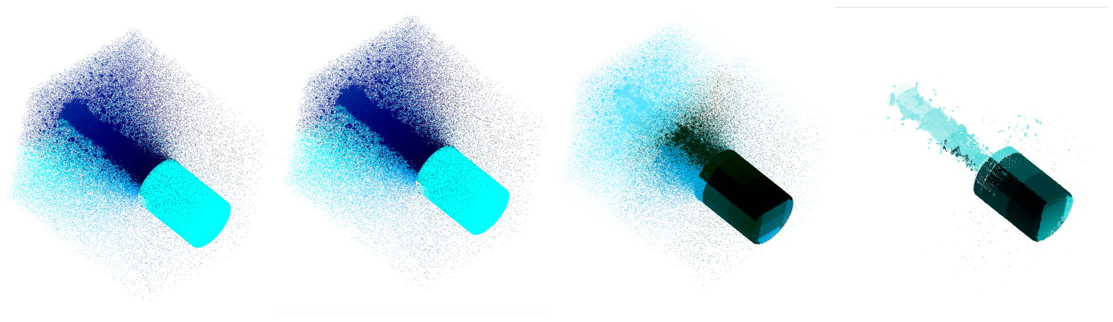
The analysis of the Nozzle decompressed files of all four compressors depicted in Figure 4.5 reveals that there is a minimal disparity between the original data (Figure 4.5a) and the decompressed files of SZ and ZFP compressors (Figure 4.5b and Figure 4.5c). However, a significant level of variance is observed in the uncompressed files generated by the Octattention and PCC_GEO compression algorithms (Figure 4.5d and Figure 4.5e). Consequently, it may be deduced from these depictions that SZ and ZFP compressors exhibited satisfactory performance for the particle data. In contrast, the point cloud compressors demonstrated less than optimal performance for the particle data in the context of visualization.

4.2.5 Sorting decompressed files

For optimal values in the particle error metrics, it is important that the sequence of points in the decompressed file is similar to that of the original file. The preservation of the point sequence in the decompressed file is ensured by the particle compressors. Conversely, point cloud compressors do not preserve the sequence of points in the resultant decompressed file. Various sorting techniques are examined to determine the optimal one that aligns with the use case, as the current approach



(a) Original



(b) SZ **(c)** ZFP **(d)** Octattention **(e)** PCC_GEO_CNN_V2

Figure 4.5: Renderings of Nozzle data using different compressors

lacks fair comparison in particle error metrics. Figure 4.6 provides insights into the performance of three different sorting techniques when applied to the Nozzle dataset. The results for other datasets that are tested to find the best sorting algorithm are provided in Chapter 5. The graph has the absolute error values on the x-axis and the number of occurrences on the y-axis. The red region represents 90% of the cumulative sum of absolute error values, and the blue region represents the remaining 10% of the values. These two regions are separated by the green dotted line. The Octattention decompressed file of the Nozzle particle data is considered as a reference to analyze the different sorting techniques. It is evident from Figure 4.6a that there exists no correspondence between the original and the decompressed file before applying any sorting algorithm. After applying the KD Search tree sorting method, the particle error metrics improved only to a minimum extent. The method using MATLAB to find the iterative minimum difference between points of original and decompressed files reduced the absolute error values enormously, with just one or two peaks above 100, with the maximum absolute error difference being 192.99670 in Table 4.20. The occurrence of such high values is rare enough that the red region is barely noticeable in Figure 4.6c. This method yields an average absolute error of 1.76873. The primary limitation of this approach is the duration required for sorting substantially large data sets. It increases exponentially based on the file size. The implementation of the Argsort technique in Python involves the utilization of the Pandas library. The reduction of the maximum absolute error value to below 5 in Table 4.20 is observed, with a higher frequency of instances in the blue region indicating minimal absolute error disparities between the original and the decompressed file. This method exhibits a comparatively lower average absolute error value (1.02552) in relation to the other methods.

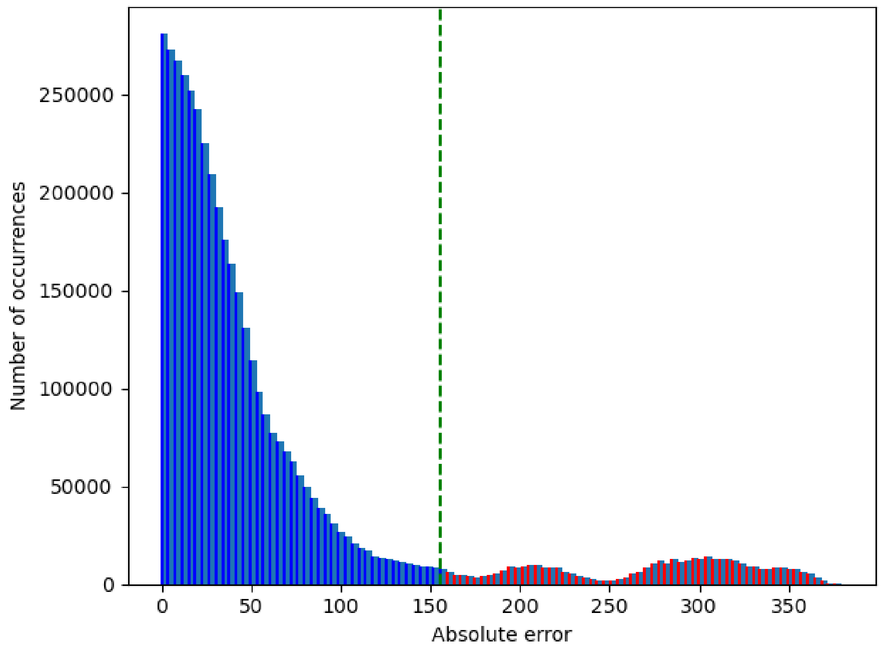
Sorting Method	Avg abs error	Max abs error	Max Relative Error	Max PW Relative Error	PSNR	NRMSE	Pearson Correlation coefficient
Without any sorting	61.0997	379.479	0.96071	15.69E+4	11.9403	0.25292	0.03
KDSearch	40.0187	379.829	0.96281	12.38E+4	14.7720	0.18256	0.49
Element-wise minimum difference calculation using MATLAB	1.76873	192.997	0.48860	1.11495	30.5855	0.02956	0.98
Argsort using Pandas	1.02552	4.99034	0.01263	0.99530	48.3989	0.00380	0.99

Table 4.20: Nozzle dataset: Octattention output - sorted decompressed files

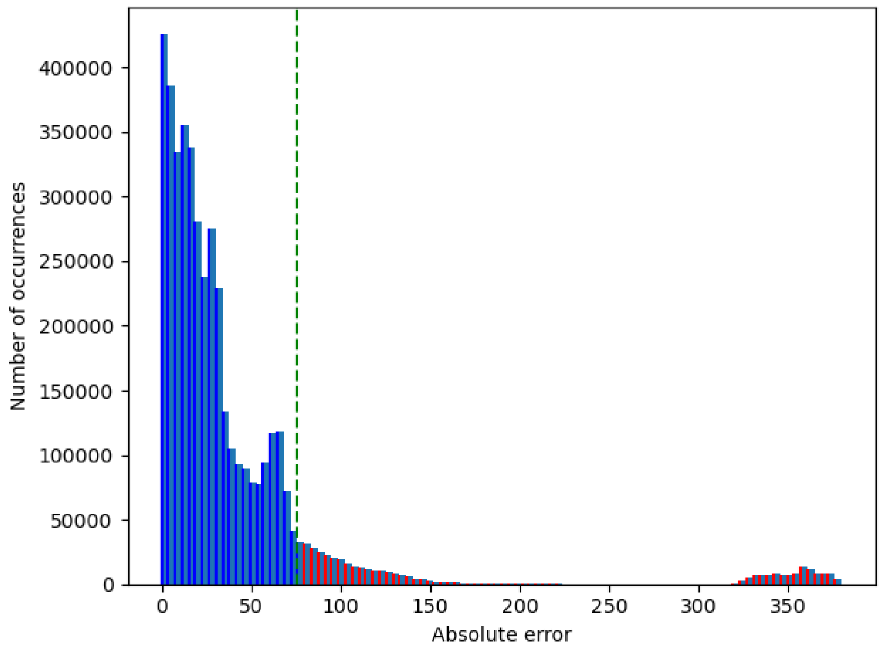
Method	Time taken (in sec)			
	Airplane	Car	Nozzle	Fluid
KD Search tree	0.31021	1.33518	112.88663	22687.12458
Element-wise minimum difference calculation using MATLAB	25.72142	216.55580	5362.72143	NA
Argsort using Pandas	0.35619	1.15498	7.23031	193.29431

Table 4.21: Time taken for sorting datasets

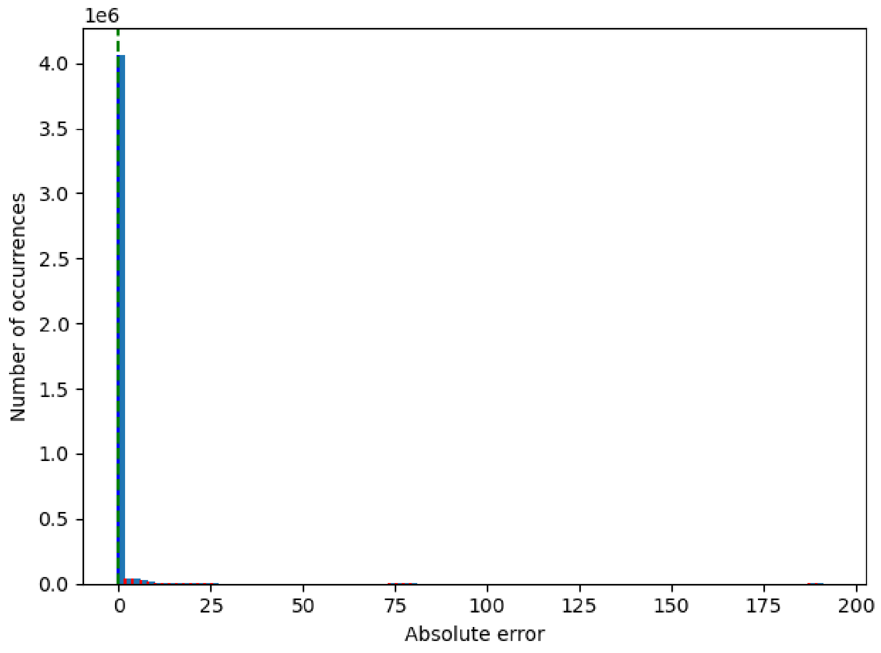
Table 4.21 contains the time taken for three different sorting techniques. It shows that the choice of sorting method can have a significant impact on the time taken to solve the problem of establishing correspondence between original and decompressed files. It has been executed across a spectrum of file sizes, encompassing both small and large magnitudes. It includes the following datasets - airplane, car, nozzle, and fluid. The KD Search tree sorting algorithm exhibited a significantly lower computational time in comparison to the MATLAB-based approach. However, it remains elevated in comparison to the Argsort technique. The element-wise minimum difference computation using MATLAB took the longest time to sort for all datasets. It was not possible to find the time taken for the fluid dataset in this approach, as it took longer than 5 days. Therefore, it is evident that this methodology is solely applicable to datasets with shorter array lengths. According to the data presented in Table 4.20, Table 4.20 and Figure 4.7, the Argsort sorting algorithm exhibits superior performance compared to the other algorithms that were tested in this study. Therefore, this method is adopted in the experiments for organizing the decompressed files generated by point cloud compressors to ensure fair comparisons in the particle error metrics.



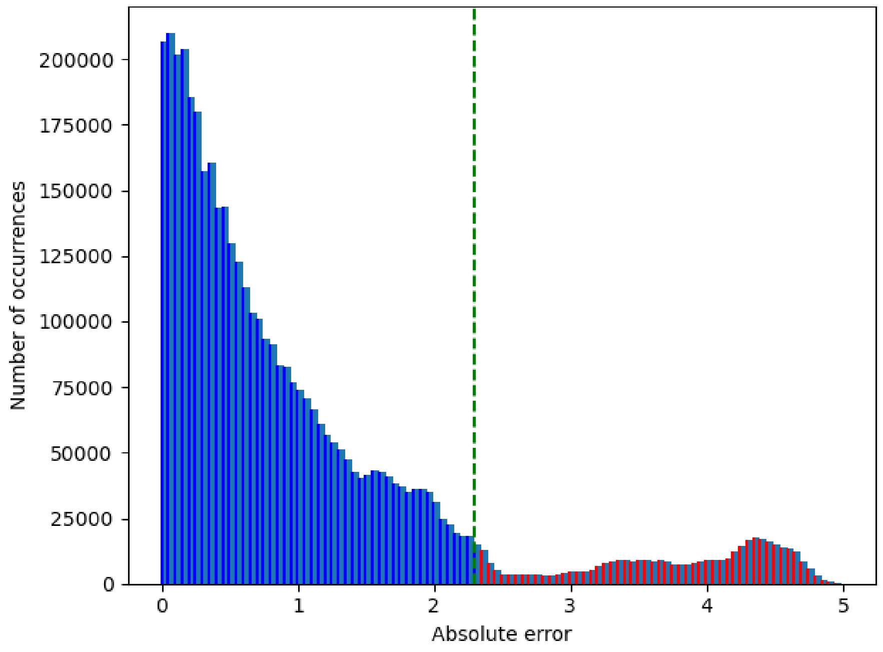
(a) Before applying sorting technique



(b) Using KD Search tree



(c) Element-wise minimum difference using MATLAB



(d) Argsort using Pandas

Figure 4.6: Absolute error difference for Nozzle data: Octattention decompressed file

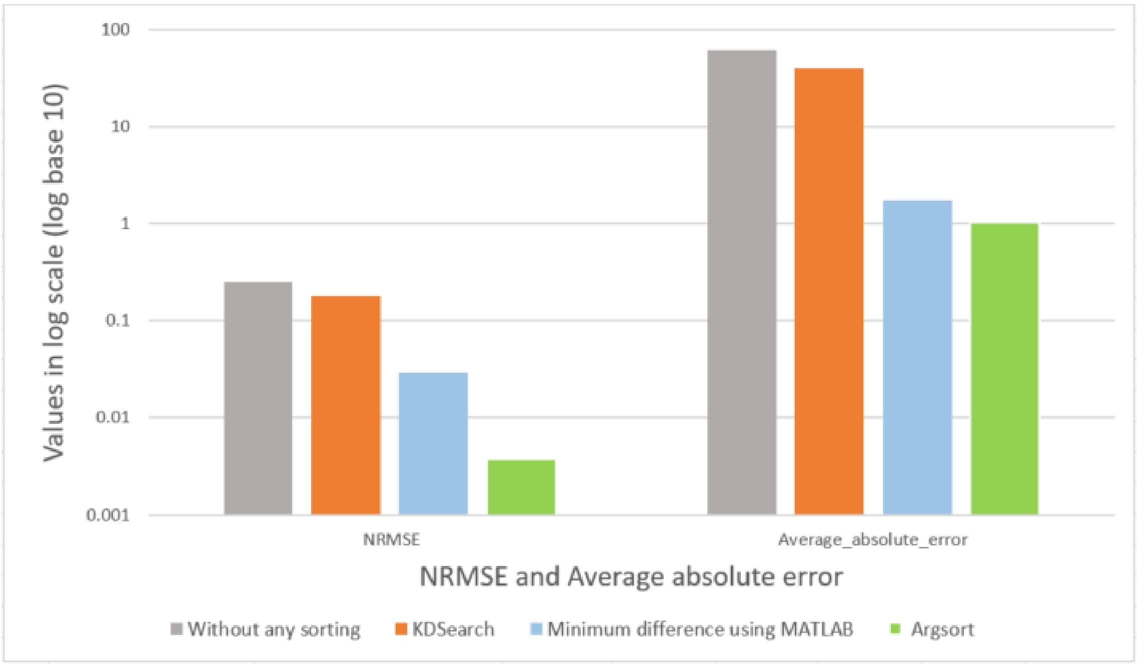


Figure 4.7: NRMSE and Average absolute error values of sorted results of Nozzle - Octattention decompressed file

4.3 Discussions

The loss of data in the decompressed outputs of the point cloud compressors used in this experiment, PCC_GEO, and Octattention, is because they are designed to work with regular grids and do not handle irregularly distributed data well. Therefore, the decompressed output of these compressors results in a loss of details and irregularities in the particle data. Overall, the rendered images provide a useful visual representation of the quality of the compressed data produced by each of the four compressors in Figure 4.4 and Figure 4.5. Based on the results obtained in the previous section, the following points are inferred:

- Point cloud compressors:
  - Results in better performance in terms of point cloud error metrics.
  - Yields high errors in particle error metrics, compared to the particle compressors, even after applying a sorting algorithm.
  - It is slow compared to particle compressors but produces high compression ratios.
  - Requires more memory while processing the compression of large data files.
  - Both the point cloud compressors chosen do not have many options to tweak the compression metrics.
  - Designed specifically for compressing point cloud data.
- Particle compressors:

- Results in better performance in terms of particle error metrics and also point cloud error metrics.
- It is faster and compresses all types of data but produces low compression ratios for the datasets tested.
- Requires fairly less memory compared to point cloud compressors for compressing large data files.
- More configuration settings are present in both the particle compressors for tweaking the compression metrics.
- Designed for compressing floating-point arrays. Mostly, any type of data that can be represented in the form of floating-point arrays can be compressed using SZ and ZFP.

4.3.1 Octattention

The Octattention compressor has a limitation on the size of data that can be fed to it as input. For point cloud datasets and particle datasets, the Octattention compressor produced better results in terms of point cloud metrics. But the results of particle error metrics for this compression technique are high compared to the particle compressors. This is because, in terms of correlation, the relationship between points in point cloud data is typically much stronger than the relationship between particles in particle data. The error metrics are devised accordingly to measure the characteristics of such data in corresponding categories. As a result, high errors are produced in the particle error metrics for the point cloud compressors. Hence regarding precision, the Octattention compressor exhibited adequate outcomes for the point cloud data. However, its effectiveness was suboptimal for the particle data. This is clearly observed in Figure 4.10, Figure 4.11, Figure 4.12, and Figure 4.13. It produced the highest compression ratios for most of the datasets tested in this experiment, as shown in Figure 4.9 and Figure 4.8.

4.3.2 PCC_GEO

The PCC_GEO compressor needs the input data to be voxelized. The compressor yielded a high compression ratio value compared to other particle compressors that are tested in this work. The particle error metrics for this compressor are also bad similar to the other point cloud compressor. Although PCC_GEO does not have a specified size constraint, its performance was evaluated on particle datasets, revealing that the compressor exhibited prolonged execution times without any observable enhancements for particularly sizable datasets, such as Riemann and Laser. The performance of the PCC_GEO compressor was suboptimal for the Fluid and Nozzle particle datasets. It can be clearly observed in Figure 4.5. This can also be noticed from the results in Table 4.10 and Table 4.12. For the point cloud data, the compression ratio of the PCC_GEO compressor was almost in the same range as that of the Octattention compressor as shown in Figure 4.8, except for the Chair dataset.

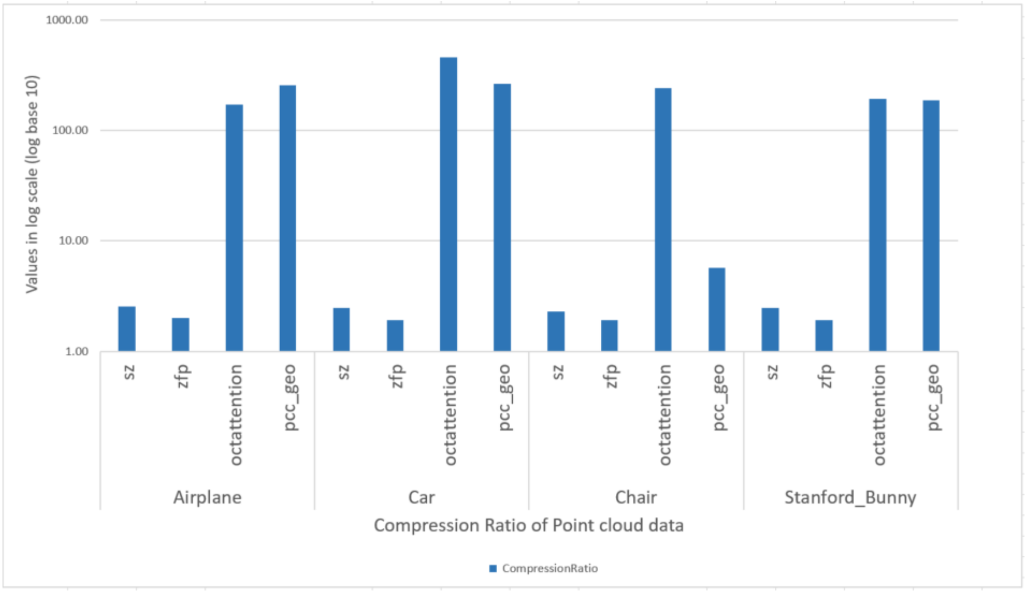


Figure 4.8: Compression Ratio - Point cloud data

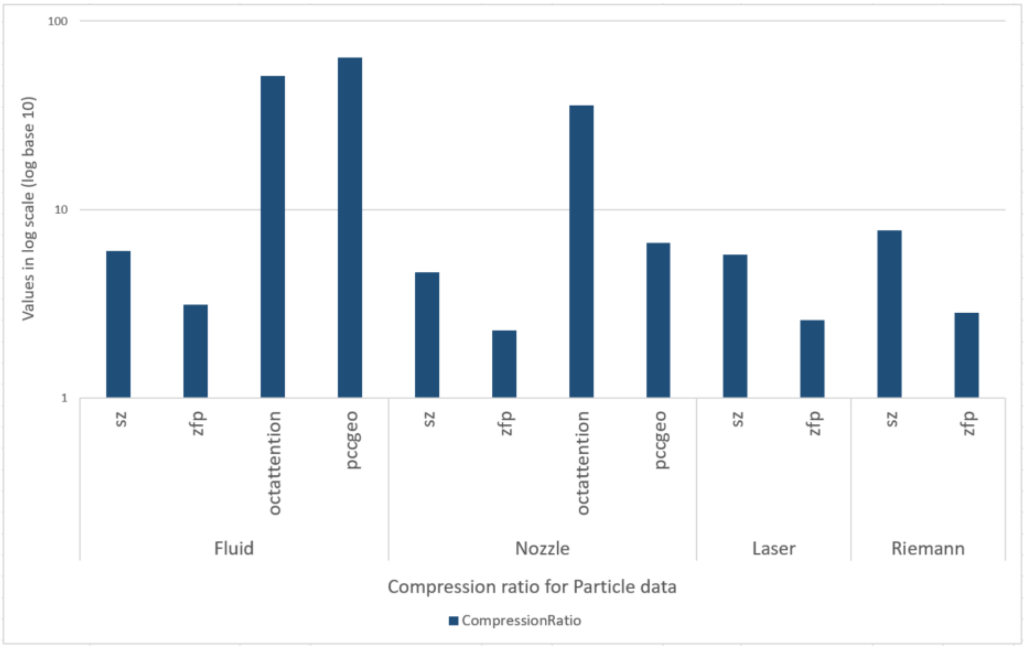


Figure 4.9: Compression Ratio - Particle data

4.3.3 SZ

The SZ compressor was able to perform compression and decompression for both particle and point cloud datasets. It achieved very low NRMSE values in most cases compared to any other compressor used in the tool, as shown in Figure 4.11 and Figure 4.10. But the drawback is the amount of compression ratio that can be achieved using this compressor for the datasets tested in this experiment, without much loss of data, is very minimum, as shown in Figure 4.9 and Figure 4.8. Still, the compression ratio produced by this compressor for the same range of NRMSE value (around 1E-4) is more than double the compression ratio achieved by the ZFP compressor. It is far less than the compression ratio of the point cloud compressors, Octattention and PCC_GEO. While the SZ compressor may not achieve the highest compression ratios compared to other compressors, as shown in Figure 4.9 and Figure 4.8, it is still a helpful compression technique for scenarios where data fidelity is paramount, and compression ratios can be sacrificed to maintain high accuracy.

4.3.4 ZFP

The ZFP compressor was also able to compress and decompress both particle data and point cloud data. It performs the best in terms of compression and decompression speeds as shown in Figure 4.3 and Figure 4.2. For most datasets, the amount of time taken for compression and decompression is the least using this compressor, making it one of the best in this category. The NRMSE values were comparatively in the same range as that of the SZ compressor and, in some cases, slightly higher than that as shown in Figure 4.11 and Figure 4.10. However, the main drawback of the ZFP compressor is the low compression ratio (Figure 4.9 and Figure 4.8), which is less than that of SZ, Octattention, and PCC_GEO. This means that the compressed data takes up more storage space compared to the other compressors, which can be a significant issue for very large datasets. Nonetheless, the ZFP compressor remains a popular choice for applications that require fast data access.

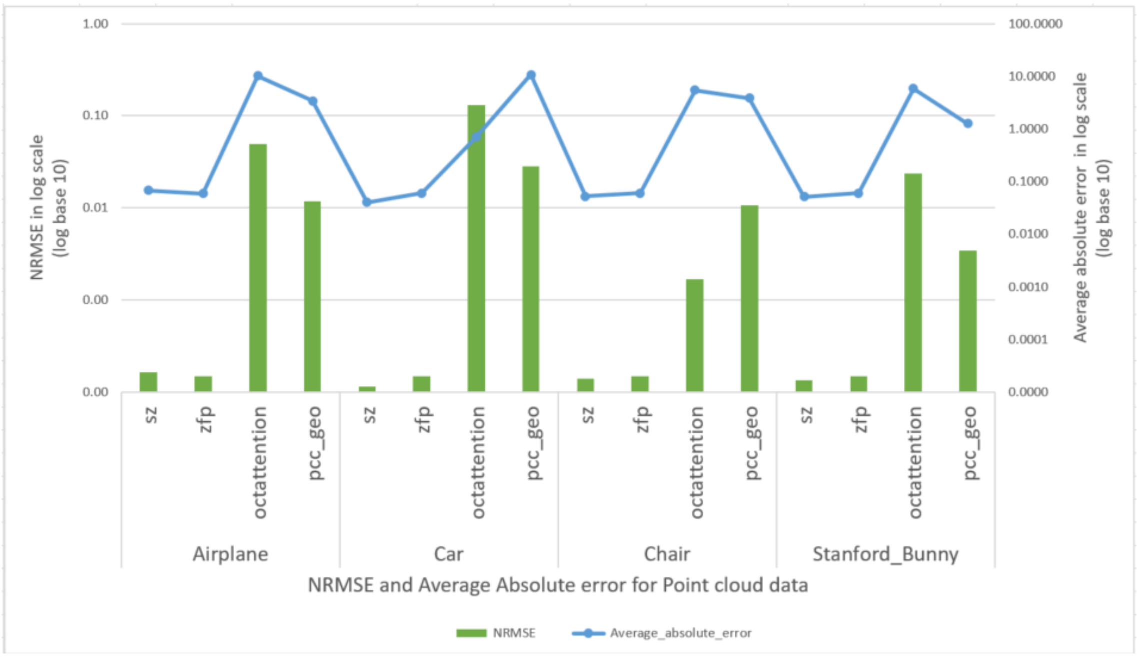


Figure 4.10: NRMSE and Average Absolute error - Point cloud data

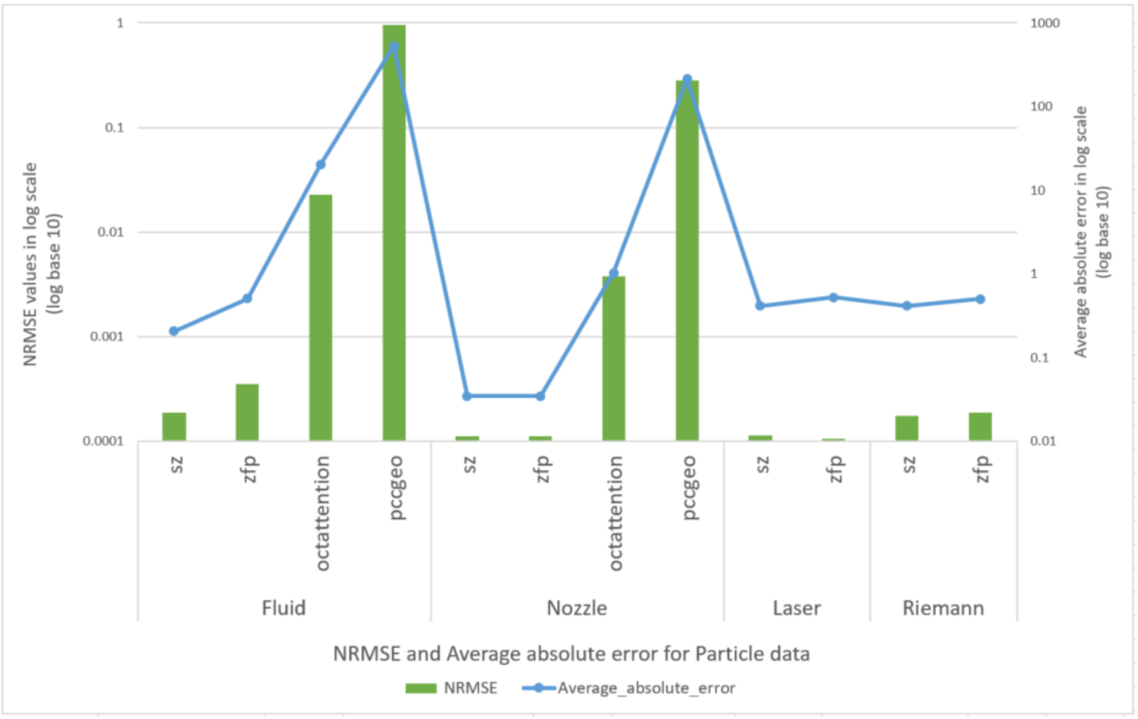


Figure 4.11: NRMSE and Average Absolute error - Particle data

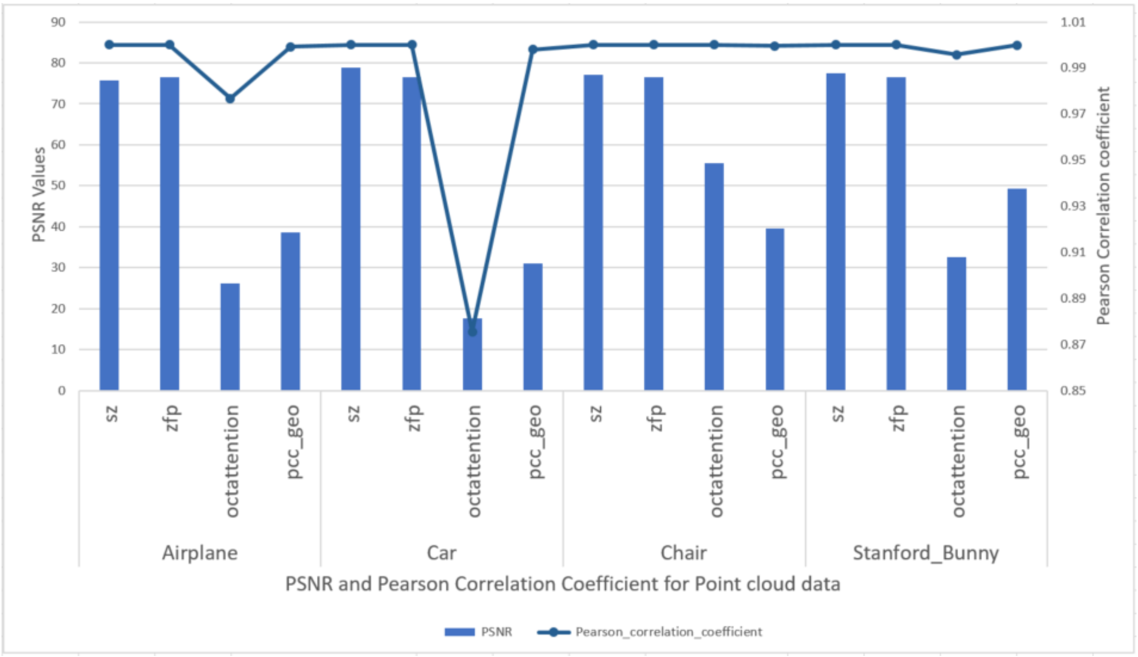


Figure 4.12: PSNR and Pearson Correlation coefficient - Point cloud data

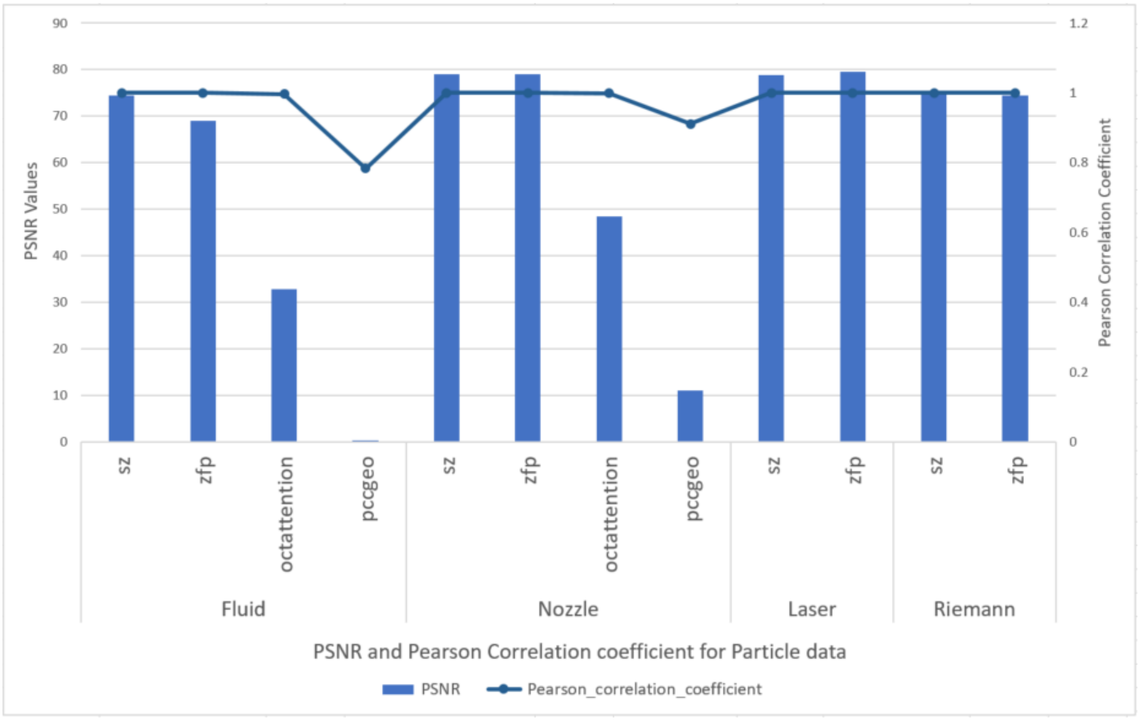


Figure 4.13: PSNR and Pearson Correlation coefficient - Particle data

4.3.5 Summary

Based on the outcomes, it is clear that the point cloud compressors Octattention and PCC_GEO perform well for all datasets smaller than 500MB in the point cloud metrics category. However, they struggle to compress and decompress larger particle datasets. The particle compressors (SZ and ZFP) can compress both categories of data efficiently. It is evident from the results for all datasets in terms of both particle error metrics and point cloud error metrics. However, the compression ratio that can be achieved using these two particle compressors is limited. At the same time, the point cloud compressors tested in the tool yielded very high compression ratios. In the realm of particle compressors, ZFP exhibited superior performance compared to SZ with respect to compression and decompression speeds across a majority of scenarios. In numerous cases, the SZ compressor exhibited a compression ratio that was superior to that of ZFP while maintaining nearly identical NRMSE values. So it is proved that ZFP can be used for applications requiring high compression and decompression speeds, whereas SZ is used when high data accuracy is essential.

Even though producing comparatively good results in the point cloud error metrics, the point cloud compressors failed to produce better results in the particle error metrics. This is because the particle metrics take one point at a time in the given order. In contrast, the point cloud metrics calculation is done by taking all the points at a time, and the nearest neighbors are found for those points, and then the error values are computed between them. It is designed in such a way because of the strong correlation of the position of points in the point cloud data. The correlation of the position of particles in the particle data is not as important as the correlation of the position of points in the point cloud data.

It is understood that the point cloud compressors used in this work are designed only to compress point cloud data, and the particle compressors used in this work are designed to compress data of both categories. The particle error metrics work only for the particle data, and the point cloud error metrics work for both categories of data. The compression of point clouds can be judged as good or bad only using point cloud metrics. Since the particle datasets are much more complicated and challenging than the point cloud datasets, the compression cannot be justified as good, only based on the naive point cloud metrics. It needs the particle error metrics to assess its quality.

4.4 Future work

To the best of my understanding, there is currently no available direct comparison between compression techniques utilized for particle datasets and those utilized for point cloud datasets. The tool devised in this study represents the initial attempt to perform a direct comparison between the two distinct compression categories. In addition, it is feasible to incorporate a graphical user interface (GUI) into the integrated compressor tool, thereby facilitating the evaluation of the compressor methodology tailored to the particular use case. The potential for further development of the integrated compressor tool exists with the incorporation of additional compressor techniques. Moreover, there is also potential for additional enhancements to the current system, such as offering recommendations for the compression of particular applications based on the compression metrics derived from prior files. This could involve presenting users with insights regarding the various metrics. This facilitates the user's decision-making process regarding the compression technique that would produce the intended outcomes for their specific application.

5 Conclusion

The present study offers a comprehensive analytical framework for evaluating various compression algorithms belonging to different categories and proposes potential avenues for enhancing their performance. While it is widely acknowledged that there is no single compression technique that can outperform all others across all types of data, the present study has undertaken research aimed at investigating a potential universal metric that could be employed to assess the effectiveness of different compression methods for varying categories of datasets. A novel tool has been developed to examine and compare diverse compression methodologies and assess their effectiveness by utilizing the particle error metrics and point cloud error metrics. The findings suggest that ZFP is the optimal compressor among the tested compressors in the tool for all categories of input data if the objective is to attain high compression and decompression speeds. Conversely, if the objective is to attain a superior level of precision, then the recommended compressor of choice would be SZ. The Octattention compressor is the preferred option when the objective is to attain a high compression ratio. Nevertheless, it is more appropriate solely for the point cloud data. Based on the results, no single metric can best prove the effectiveness of the particle and point cloud compression techniques. It needs multiple metrics to assess the quality of the compression. The only metric that is common in both worlds is the compression ratio. But a compression technique cannot be judged solely based on the compression ratio alone.

Bibliography

[1] A. H. Baker, H. Xu, J. M. Dennis, M. N. Levy, D. Nychka, S. A. Mickelson, J. Edwards, M. Vertenstein, A. Wegener. “A Methodology for Evaluating the Impact of Data Compression on Climate Simulation Data”. In: HPDC ’14. Vancouver, BC, Canada: Association for Computing Machinery, 2014, pp. 203–214. ISBN: 9781450327497. DOI: 10.1145/2600212.2600217. URL: https://doi.org/10.1145/2600212.2600217 (cit. on p. 15).

[2] R. Ballester-Ripoll, P. Lindstrom, R. Pajarola. “TTHRESH: Tensor Compression for Multidimensional Visual Data”. In: *IEEE Transactions on Visualization and Computer Graphics* 26.9 (Sept. 2020), pp. 2891–2903. DOI: 10.1109/tvcg.2019.2904063. URL: https://doi.org/10.1109%2Ftvcg.2019.2904063 (cit. on pp. 15, 18, 26, 32).

[3] T. Banerjee, J. Choi, J. Lee, Q. Gong, J. Chen, S. Klasky, A. Rangarajan, S. Ranka. *Scalable Hybrid Learning Techniques for Scientific Data Compression*. 2022. arXiv: 2212.10733 [cs.LG] (cit. on p. 18).

[4] E. A. Baran, A. Kuzu, S. Bogosyan, M. Gokasan, A. Sabanovic. “Comparative Analysis of a Selected DCT-Based Compression Scheme for Haptic Data Transmission”. In: *IEEE Transactions on Industrial Informatics* 12.3 (2016), pp. 1146–1155. DOI: 10.1109/TII.2016.2555982 (cit. on p. 17).

[5] C. Cao, M. Preda, T. B. Zaharia. “3D Point Cloud Compression: A Survey”. In: *The 24th International Conference on 3D Web Technology* (2019) (cit. on p. 19).

[6] C. Cao, M. Preda, V. Zakharchenko, E. S. Jang, T. Zaharia. “Compression of Sparse and Dense Dynamic Point Clouds—Methods and Standards”. In: *Proceedings of the IEEE* 109.9 (2021), pp. 1537–1558. DOI: 10.1109/JPROC.2021.3085957 (cit. on p. 19).

[7] S. Di, F. Cappello. “Fast Error-Bounded Lossy HPC Data Compression with SZ”. In: *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2016, pp. 730–739. DOI: 10.1109/IPDPS.2016.11 (cit. on pp. 16, 18, 23, 24, 31, 34, 44).

[8] Y. Ding, R. Xie, Y. Zou, J. Guo. “NMR Data Compression Method Based on Principal Component Analysis”. In: *Applied Magnetic Resonance* 47 (Mar. 2016). DOI: 10.1007/s00723-015-0750-8 (cit. on p. 17).

[9] E. Eisfeld, H.-R. Trebin, J. Roth. “A wide-range modeling approach for the thermal conductivity and dielectric function of solid and liquid aluminum”. In: *The European Physical Journal Special Topics* 227 (2019), pp. 1575–1590 (cit. on p. 39).

[10] C. Fu, G. Li, R. Song, W. Gao, S. Liu. “OctAttention: Octree-Based Large-Scale Contexts Model for Point Cloud Compression”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.1 (June 2022), pp. 625–633. DOI: 10.1609/aaai.v36i1.19942. URL: https://ojs.aaai.org/index.php/AAAI/article/view/19942 (cit. on pp. 16, 19, 20, 27, 28, 32, 44, 50).

[11] D. C. Garcia, T. A. Fonseca, R. U. Ferreira, R. 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. DOI: 10.1109/TIP.2019.2931466 (cit. on p. 19).

[12] D. C. Garcia, R. L. de Queiroz. "Context-based octree coding for point-cloud video". In: *2017 IEEE International Conference on Image Processing (ICIP)*. 2017, pp. 1412–1416. DOI: 10.1109/ICIP.2017.8296514 (cit. on p. 19).

[13] F. L. Gewers, G. R. Ferreira, H. F. D. Arruda, F. N. Silva, C. H. Comin, D. R. Amancio, L. D. F. Costa. "Principal Component Analysis: A Natural Approach to Data Exploration". In: 54.4 (May 2021). ISSN: 0360-0300. DOI: 10.1145/3447755. URL: https://doi.org/10.1145/3447755 (cit. on p. 17).

[14] P. Gralka, M. Becher, M. Braun, F. Frieß, C. Müller, T. Rau, K. Schatz, C. Schulz, M. Krone, G. Reina, T. Ertl. "MegaMol – a comprehensive prototyping framework for visualizations". In: *The European Physical Journal Special Topics* 227.14 (Mar. 2019), pp. 1817–1829. ISSN: 1951-6401. DOI: 10.1140/epjst/e2019-800167-5 (cit. on p. 34).

[15] P. Gralka, I. Wald, S. Geringer, G. Reina, T. Ertl. "Spatial Partitioning Strategies for Memory-Efficient Ray Tracing of Particles". In: *2020 IEEE 10th Symposium on Large Data Analysis and Visualization (LDAV)*. 2020, pp. 42–52. DOI: 10.1109/LDAV51489.2020.00012 (cit. on p. 39).

[16] D. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, 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 (2020), e13. DOI: 10.1017/ATSIP.2020.12 (cit. on pp. 19, 20, 43).

[17] D. B. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, A. J. 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 (2020) (cit. on pp. 21, 27).

[18] Y. Hu, W. Yang, Z. Ma, J. Liu. *Learning End-to-End Lossy Image Compression: A Benchmark*. 2021. arXiv: 2002.03711 [eess.IV] (cit. on pp. 19, 20).

[19] T. Huang, Y. Liu. "3D Point Cloud Geometry Compression on Deep Learning". In: *Proceedings of the 27th ACM International Conference on Multimedia*. MM '19. Nice, France: Association for Computing Machinery, 2019, pp. 890–898. ISBN: 9781450368896. DOI: 10.1145/3343031.3351061. URL: https://doi.org/10.1145/3343031.3351061 (cit. on p. 20).

[20] E. S. Jang, M. Preda, K. Mammou, A. M. Tourapis, J. Kim, D. B. Graziosi, S. Rhyu, M. 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 (2019), pp. 118–123. DOI: 10.1109/MSP.2019.2900721 (cit. on p. 19).

[21] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, E. Steinbach. "Real-time compression of point cloud streams". In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 778–785. DOI: 10.1109/ICRA.2012.6224647 (cit. on p. 19).

[22] M. Krivokuća, P. A. Chou, M. Koroteev. "A Volumetric Approach to Point Cloud Compression–Part II: Geometry Compression". In: *IEEE Transactions on Image Processing* 29 (2020), pp. 2217–2229. DOI: 10.1109/TIP.2019.2957853 (cit. on p. 19).

[23] S. Lakshminarasimhan, N. Shah, S. Ethier, S.-H. Ku, C. S. Chang, S. Klasky, R. Latham, R. Ross, N. F. Samatova. “ISABELA for effective in situ compression of scientific data: ISABELA FOR EFFECTIVE IN-SITU REDUCTION OF SPATIO-TEMPORAL DATA”. In: *Concurrency and Computation. Practice and Experience* 25.4 (July 2012). ISSN: 1532-0626. DOI: 10.1002/cpe.2887. URL: https://www.osti.gov/biblio/1564924 (cit. on p. 18).

[24] P. Lindstrom. “Fixed-Rate Compressed Floating-Point Arrays”. In: *IEEE Transactions on Visualization and Computer Graphics* 20 (Aug. 2014). DOI: 10.1109/TVCG.2014.2346458 (cit. on pp. 15, 16, 18, 25, 26, 31, 34, 44).

[25] H. Liu, H. Yuan, Q. Liu, J. Hou, J. Liu. *A Comprehensive Study and Comparison of Core Technologies for MPEG 3D Point Cloud Compression*. Dec. 2019 (cit. on p. 21).

[26] S. Liu, M. Zhang, P. Kadam, C.-C. J. Kuo. “Traditional Point Cloud Analysis”. In: *3D Point Cloud Analysis: Traditional, Deep Learning, and Explainable Machine Learning Methods*. Cham: Springer International Publishing, 2021, pp. 15–52. ISBN: 978-3-030-89180-0. DOI: 10.1007/978-3-030-89180-0_2. URL: https://doi.org/10.1007/978-3-030-89180-0_2 (cit. on p. 38).

[27] D. Meagher. “Geometric Modeling Using Octree-Encoding”. In: *Computer Graphics and Image Processing* 19 (June 1982), pp. 129–147. DOI: 10.1016/0146-664X(82)90104-6 (cit. on p. 21).

[28] D. T. Nguyen, M. Quach, G. Valenzise, P. Duhamel. “Multiscale deep context modeling for lossless point cloud geometry compression”. In: *2021 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*. 2021, pp. 1–6. DOI: 10.1109/ICMEW53276.2021.9455990 (cit. on p. 19).

[29] A. Pandey, B. Singh Saini, B. Singh, N. Sood. “Quality controlled ECG data compression based on 2D discrete cosine coefficient filtering and iterative JPEG2000 encoding”. In: *Measurement* 152 (2020), p. 107252. ISSN: 0263-2241. DOI: https://doi.org/10.1016/j.measurement.2019.107252. URL: https://www.sciencedirect.com/science/article/pii/S0263224119311169 (cit. on p. 17).

[30] M. Quach, J. Pang, T. Dong, G. Valenzise, F. Dufaux. “Survey on Deep Learning-based Point Cloud Compression”. In: *Frontiers in Signal Processing* 2 (2022). DOI: 10.3389/frsip.2022.846972. URL: https://hal.science/hal-03579360 (cit. on p. 19).

[31] M. Quach, J. Pang, D. Tian, G. Valenzise, F. Dufaux. “Survey on Deep Learning-Based Point Cloud Compression”. In: *Frontiers in Signal Processing* 2 (Feb. 2022). DOI: 10.3389/frsip.2022.846972 (cit. on p. 20).

[32] M. Quach, G. Valenzise, F. Dufaux. “Learning Convolutional Transforms for Lossy Point Cloud Geometry Compression”. In: *CoRR* abs/1903.08548 (2019). arXiv: 1903.08548. URL: http://arxiv.org/abs/1903.08548 (cit. on p. 29).

[33] M. Quach, G. Valenzise, F. Dufaux. *Improved Deep Point Cloud Geometry Compression*. 2020. arXiv: 2006.09043 [cs.CV] (cit. on pp. 16, 20, 29, 32, 44).

[34] Z. Que, G. Lu, D. Xu. *VoxelContext-Net: An Octree based Framework for Point Cloud Compression*. 2021. arXiv: 2105.02158 [cs.CV] (cit. on p. 19).

[35] Z. Que, G. Lu, D. Xu. “VoxelContext-Net: An Octree based Framework for Point Cloud Compression”. In: *CoRR* abs/2105.02158 (2021). arXiv: 2105.02158. URL: https://arxiv.org/abs/2105.02158 (cit. on p. 19).

[36] R. L. de Queiroz, P. A. Chou. "Compression of 3D Point Clouds Using a Region-Adaptive Hierarchical Transform". In: *IEEE Transactions on Image Processing* 25.8 (2016), pp. 3947–3956. DOI: 10.1109/TIP.2016.2575005 (cit. on p. 19).

[37] R. L. de Queiroz, P. A. Chou. "Compression of 3D Point Clouds Using a Region-Adaptive Hierarchical Transform". In: *IEEE Transactions on Image Processing* 25.8 (2016), pp. 3947–3956. DOI: 10.1109/TIP.2016.2575005 (cit. on p. 21).

[38] R. L. de Queiroz, D. C. Garcia, P. A. Chou, D. A. Florencio. "Distance-Based Probability Model for Octree Coding". In: *IEEE Signal Processing Letters* 25.6 (2018), pp. 739–742. DOI: 10.1109/LSP.2018.2823701 (cit. on p. 19).

[39] M. Ruhnke, B. Steder, G. Grisetti, W. Burgard. "Unsupervised learning of compact 3D models based on the detection of recurrent structures". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2010, pp. 2137–2142. DOI: 10.1109/IROS.2010.5649730 (cit. on pp. 19, 20).

[40] N. Sasaki, K. Sato, T. Endo, S. Matsuoka. "Exploration of Lossy Compression for Application-Level Checkpoint/Restart". In: *Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium*. IPDPS '15. USA: IEEE Computer Society, 2015, pp. 914–922. ISBN: 9781479986491. DOI: 10.1109/IPDPS.2015.67. URL: https://doi.org/10.1109/IPDPS.2015.67 (cit. on p. 15).

[41] S. Schwarz, M. Preda, V. Baroncini, M. Budagavi, P. Cesar, P. A. Chou, R. A. Cohen, M. Krivokuća, S. Lasserre, Z. Li, J. Llach, K. Mammou, R. Mekuria, O. Nakagami, E. Siahaan, A. Tabatabai, A. M. Tourapis, V. Zakharchenko. "Emerging MPEG Standards for Point Cloud Compression". In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9.1 (2019), pp. 133–148. DOI: 10.1109/JETCAS.2018.2885981 (cit. on p. 27).

[42] A. L. Souto, V. F. Figueiredo, P. A. Chou, R. L. de Queiroz. "Set Partitioning in Hierarchical Trees for Point Cloud Attribute Compression". In: *IEEE Signal Processing Letters* 28 (2021), pp. 1903–1907. DOI: 10.1109/LSP.2021.3112335 (cit. on p. 17).

[43] D. Tao, S. Di, H. Guo, Z. Chen, F. Cappello. "Z-checker: A Framework for Assessing Lossy Compression of Scientific Data". In: *CoRR* abs/1707.09320 (2017). arXiv: 1707.09320. URL: http://arxiv.org/abs/1707.09320 (cit. on pp. 18, 19).

[44] D. Thanou, P. A. Chou, P. Frossard. "Graph-Based Compression of Dynamic 3D Point Cloud Sequences". In: *IEEE Transactions on Image Processing* 25.4 (2016), pp. 1765–1778. DOI: 10.1109/TIP.2016.2529506 (cit. on p. 17).

[45] D. Tian, H. Ochimizu, C. Feng, R. Cohen, A. Vetro. "Geometric distortion metrics for point cloud compression". In: *2017 IEEE International Conference on Image Processing (ICIP)*. 2017, pp. 3460–3464. DOI: 10.1109/ICIP.2017.8296925 (cit. on pp. 43, 44).

[46] J. Tian, S. Di, K. Zhao, C. Rivera, M. H. Fulp, R. Underwood, S. Jin, X. Liang, J. Calhoun, D. Tao, F. Cappello. "cuSZ: An Efficient GPU-Based Error-Bounded Lossy Compression Framework for Scientific Data". In: *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*. PACT '20. Virtual Event, GA, USA: Association for Computing Machinery, 2020, pp. 3–15. ISBN: 9781450380751. DOI: 10.1145/3410463.3414624. URL: https://doi.org/10.1145/3410463.3414624 (cit. on pp. 18, 24, 25, 32).

[47] R. Underwood, S. Di, J. C. Calhoun, F. Cappello. “FRaZ: A Generic High-Fidelity Fixed-Ratio Lossy Compression Framework for Scientific Floating-point Data”. In: *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2020, pp. 567–577. DOI: 10.1109/IPDPS47924.2020.00065 (cit. on p. 19).

[48] L. Wiesmann, A. Milioto, X. Chen, C. Stachniss, J. Behley. “Deep Compression for Dense Point Cloud Maps”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 2060–2067. DOI: 10.1109/LRA.2021.3059633 (cit. on pp. 30, 32).

[49] Z. Wu, S. Song, A. Khosla, X. Tang, J. Xiao. “3D ShapeNets for 2.5D Object Recognition and Next-Best-View Prediction”. In: *CoRR* abs/1406.5670 (2014). arXiv: 1406.5670. URL: http://arxiv.org/abs/1406.5670 (cit. on p. 39).

[50] Y. Xu, X. Tong, U. Stilla. “Voxel-based representation of 3D point clouds: Methods, applications, and its potential use in the construction industry”. In: *Automation in Construction* 126 (2021), p. 103675. ISSN: 0926-5805. DOI: https://doi.org/10.1016/j.autcon.2021.103675. URL: https://www.sciencedirect.com/science/article/pii/S0926580521001266 (cit. on p. 21).

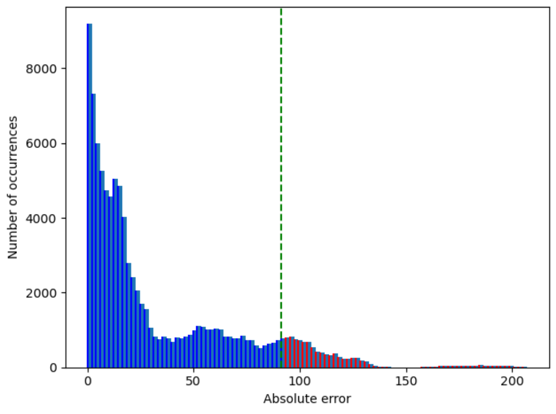
[51] W. Yan, Y. shao, S. Liu, T. H. Li, Z. Li, G. Li. *Deep AutoEncoder-based Lossy Geometry Compression for Point Clouds*. 2019. arXiv: 1905.03691 [cs.CV] (cit. on pp. 30–32).

[52] S. Zhang, W. Zhang, F. Yang, J. Huo. “A 3D Haar Wavelet Transform for Point Cloud Attribute Compression Based on Local Surface Analysis”. In: *2019 Picture Coding Symposium (PCS)*. 2019, pp. 1–5. DOI: 10.1109/PCS48520.2019.8954557 (cit. on p. 17).

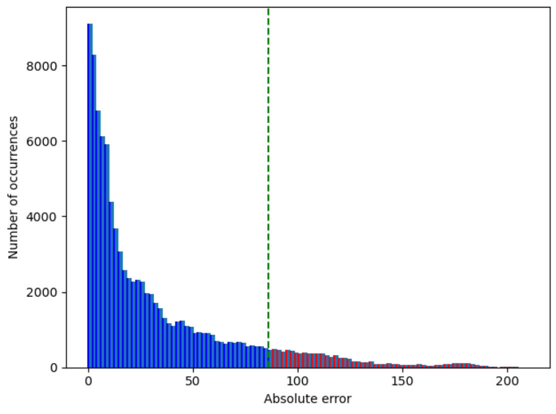
[53] X. Zhang, W. Wan, X. An. “Clustering and DCT Based Color Point Cloud Compression”. In: *J. Signal Process. Syst.* 86.1 (Jan. 2017), pp. 41–49. ISSN: 1939-8018. DOI: 10.1007/s11265-015-1095-0. URL: https://doi.org/10.1007/s11265-015-1095-0 (cit. on p. 17).

A Appendix

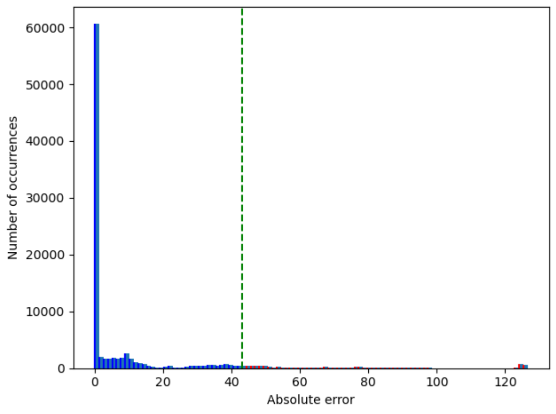
A.1 Results of sorting techniques on different datasets



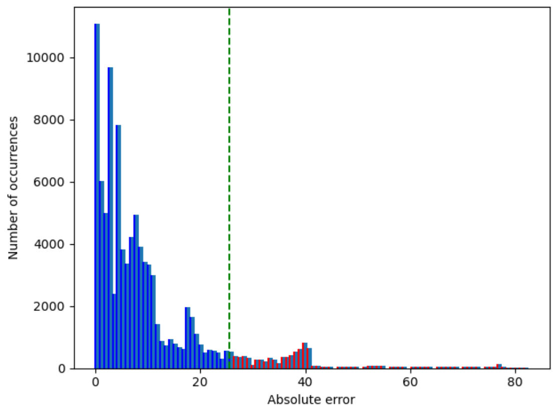
(a) Before applying sorting technique



(b) Using KD Search tree

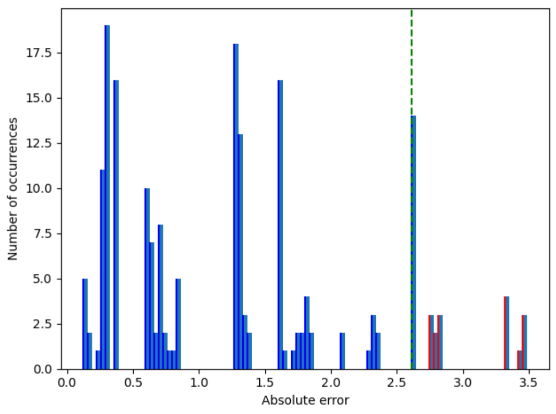


(c) Element-wise minimum difference using MATLAB

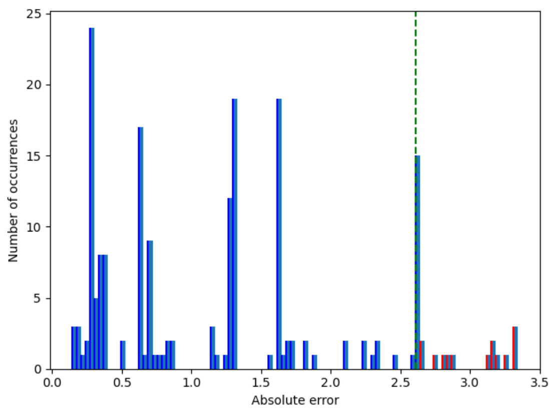


(d) Argsort using Pandas

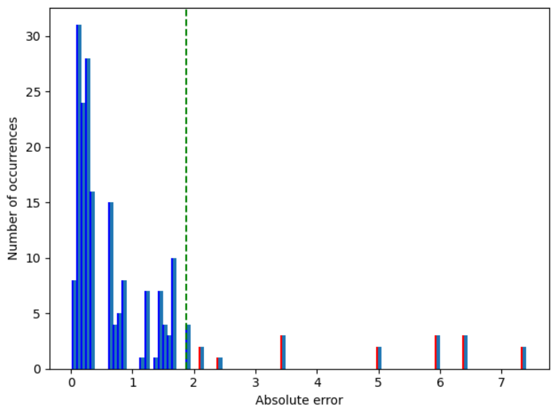
Figure A.1: Absolute error difference for Airplane data: Octattention decompressed file



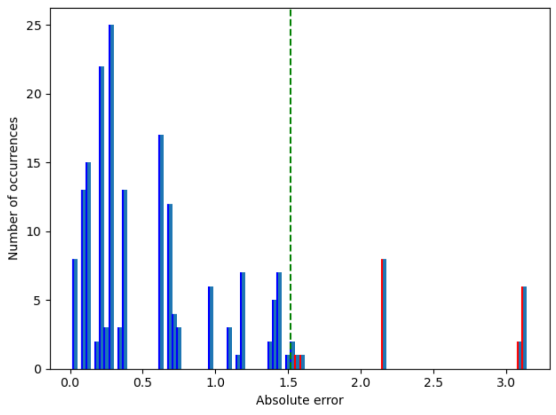
(a) Before applying sorting technique



(b) Using KD Search tree

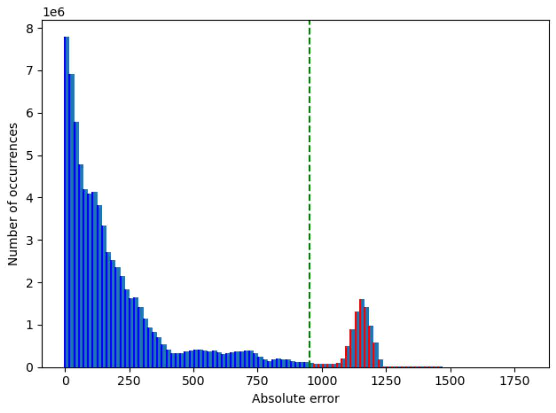


(c) Element-wise minimum difference using MATLAB

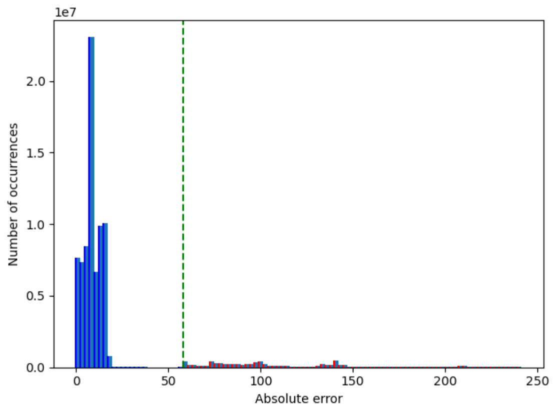


(d) Argsort using Pandas

Figure A.2: Absolute error difference for Car data: Octattention decompressed file



(a) Before applying sorting technique



(b) Argsort using Pandas

Figure A.3: Absolute error difference for Fluid data: Octattention decompressed file

A.2 Pseudocode for Sorting Algorithm

Algorithm A.1 Argsort using Pandas library in Python

```
Require: $testset\_array$: Contains points of the original file
Require: $dataset\_array$: Contains points of the decompressed file
1: $df = pd.DataFrame(data = 'test' : testset\_array[: len(dataset\_array)])$ ← DataFrame with columns 'test'
2: $df.sort\_values(by =' test', inplace = True)$ ← sort $df$ by 'test' along with it's index positions
3: $df['data'] = sorted(dataset\_array)$ ← Contains sorted values from $dataset\_array$
4: $df.sort\_index(inplace = True)$ ← Sorts the $dataframe$ based on original index positions of the $testset\_array$
5: $df['data']$ ← Contains the sorted $dataset\_array$
```

Algorithm A.2 Minimum difference calculation using MATLAB

```
$'fopen()'$ ← Original and decompressed binary files are opened
$'fread()'$ ← Data in binary format is read and returned as a numeric arrays ($testset\_array$ and $dataset\_array$ )
if $len(testset\_array) \geq len(dataset\_array)$ then
    $testset\_array$ ← $testset\_array(1 : len(dataset\_array))$ //Truncates testset_array to the length of dataset_array
else
    $dataset\_array$ ← $dataset\_array(1 : len(testset\_array))$ //Truncates dataset_array to the length of testset_array
end if
$temp$ ← values from $dataset\_array$ are copied
$n$ ← number of elements in $testset\_array$
$tic$ ← Starts a timer
$y$ ← row vector of $n$ zeros
for $iteration = 1, 2, \ldots n$ do
    Find the index of the closest value in $temp$ using $min(abs(temp - testset\_array(k)))$
    Store the index in the corresponding element of $y$
end for
$toc$ ← Stops the timer and display time
$'fwrite()'$ ← Writes $dataset\_array$ to binary file
```

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted hard copies.

place, date, signature