Point Cloud Compression Using Sorted Delta Encoding and Huffman Coding

Ricardo Corral-Corral

August 2025

Abstract

This paper presents a novel method for compressing 2D point cloud data by leveraging independent sorting, delta encoding, and theoretical Huffman coding. Inspired by a humorous Stack Exchange post, we transform the original data into a sorted representation, compute differences (deltas), and apply entropy encoding to achieve high compression ratios. Experimental results on synthetic datasets with up to 10 million points demonstrate compression ratios exceeding 7.99:1 (87.5% compression), with an average of 0.01 bits per delta. The methodology is implemented in Elixir as the 'CloudDelta' library, available at `https://github.com/doctorcorral/cloud_delta/` and `https://hex.pm/packages/cloud_delta`.

1 Introduction

Point cloud data, prevalent in robotics, computer vision, and scientific simulations, poses storage and transmission challenges due to its high dimensionality. Traditional compression methods (e.g., G-PCC) typically achieve 2:1 to 4:1 ratios, but synthetic datasets with structured patterns offer opportunities for higher efficiency. This work builds on a Stack Exchange post [1] that jokingly suggested sorting data to "improve" regression, adapting it into a lossless compression technique. We sort X and Y coordinates independently, compute deltas, and use theoretical Huffman coding to exploit redundancy, achieving significant compression on synthetic data.

2 Methodology

2.1 Data Generation

We generate synthetic 2D point clouds with n points, where X is uniformly distributed over $[0, 2)$ and Y follows patterns such as $Y = X^2 + \text{noise}$, $Y = \sin(\pi X) + \text{noise}$, $Y = 2X + \text{noise}$, or random Y, with noise drawn from a normal distribution ($\mu = 0$, $\sigma = 0.5$). These patterns introduce exploitable redundancy.

2.2 Compression Algorithm

The compression pipeline consists of four key steps, transforming the original point cloud data into a compact representation:

Step 1 Independent Sorting: Sort the X and Y coordinates independently, producing sorted arrays X_{sorted} and Y_{sorted}. This step generates inverse permutation indices P_X^{-1} and P_Y^{-1} to track the original order for reconstruction.

Step 2 Delta Encoding: Compute the first differences of the sorted arrays, $\Delta X = \text{diff}(X_{\text{sorted}})$ and $\Delta Y = \text{diff}(Y_{\text{sorted}})$. Prepend the initial values $X[0]$ and $Y[0]$ to the delta sequences for lossless reconstruction.

Step 3 Huffman Encoding: Construct a theoretical Huffman tree from the frequency map of all deltas (ΔX and ΔY) using a priority queue. Variable-length codes are assigned based on frequency, optimizing for redundancy in sorted data (fully implemented in memory, with hybrid binary storage).

Step 4 Storage: Store the compressed data as a sequence comprising the initial values (64 bits total, 32 bits each for $X[0]$ and $Y[0]$), the permutation indices (8 bits per index), and the delta sequences in a hybrid binary format. Details are available in the ‘CloudDelta‘ repository.

Reconstruction reverses the process: decode the deltas, reconstruct X_{sorted} and Y_{sorted} via cumulative sums, and apply the inverse permutations to recover the original (X, Y) pairs, ensuring lossless compression.

2.3 Implementation

The method is implemented in Elixir as the ‘CloudDelta‘ library, utilizing the Nx library for numerical computations. The source code, documentation, and package are available at `https://github.com/doctorcorral/cloud_delta/` and `https://hex.pm/packages/cloud_delta`.

3 Results

Experiments were conducted on synthetic datasets with $n = 100, 1000, 10000, 100000, 1000000, 10000000$ across squared, sine, linear, and random patterns. Key metrics are summarized in Table 1.

The compression ratio increases with n, peaking at 7.99:1 for linear and random data at $n = 10,000,000$, with bits per delta dropping to 0.0-0.01. This reflects growing redundancy in deltas as dataset size scales.

Pattern	Size	Compression Ratio	% Compression	Avg Bits/Delta
squared	100	2.66:1	62.4%	7.72
squared	1000	2.17:1	54.0%	10.7
squared	10000	3.24:1	69.1%	5.89
squared	100000	6.34:1	84.2%	1.05
squared	1000000	7.8:1	87.2%	0.1
squared	10000000	7.98:1	87.5%	0.01
sin	10000000	7.98:1	87.5%	0.01
linear	10000000	7.99:1	87.5%	0.01
random	10000000	7.99:1	87.5%	0.0

Table 1: Compression results for varying dataset sizes and patterns (selected highlights).

4 Discussion

The method significantly outperforms traditional point cloud compression (2:1 to 4:1) on synthetic data due to the structured patterns and independent sorting, inspired by [1]. The hybrid approach balances theoretical Huffman efficiency with practical binary I/O, achieving up to 7.99:1 compression on datasets up to 10 million points. However, real-world data (e.g., LIDAR scans) may yield lower ratios (50-67%) due to less predictable patterns. The scalability to large n highlights the method's potential, though execution time and memory usage remain areas for optimization in future releases.

5 Future Work

Future efforts will focus on testing with real-world datasets and optimizing execution time (e.g., parallelizing sorting). A v2.0 release is planned to include full Huffman binary encoding, further enhancing compression efficiency.

6 Conclusion

This work introduces a novel compression technique achieving up to 87.5% compression on synthetic point clouds. The 'CloudDelta' library, implemented in Elixir and available at `https://github.com/doctorcorral/clouddelta/` and `https://hex.pm/packages/cloud_delta`, showcases its potential, with further improvements anticipated.

References

[1] Michael Lopez, "Suppose we sort data (X, Y) with n points...," Stats Stack Exchange, 2025. `https://stats.stackexchange.com/...`