A Novel Set Partition Coding Algorithm For GeoTIFF Digital Elevation Models

Jack D. Carson[1]

[1]Booker T. Washington High School, Tulsa, Oklahoma

Abstract

LIDAR, the high-density laser scanning of geographical features, is an essential technology for driverless cars, industrial robotics, and smart infrastructure. Yet, vast file sizes, often well into the terabyte range, have made the transfer and storage of LIDAR Digital Elevation Maps (DEMs) a significant challenge. Without compressed LIDAR DEMs, automated systems instead must create in real time their own LIDAR point clouds; but these point clouds have errors and are limited in range. To solve this problem, I show a novel algorithm for the compression of rasterized DEMs using a set-partition-coding algorithm specifically designed for accurate and reliable encoding of geospatial altitude data. GeoTIFF Digital Elevation Maps are partitioned recursively to create a quad-tree structure of planar approximations. This scheme differentiates itself primarily through its use of information entropy as a characteristic for evaluating the accuracy of its own planar approximations. The algorithm defined in this paper, entitled General Purpose Geospatial Compression (GPGC), has been shown to exhibit compression ratios exceeding any similar scheme (from 25:1 to 73:1) at an unprecedented mean accuracy of 99.6% over a random sample of 135 United States Geological Survey continental rasters. The decision to make both the standard and C++ implementation of the GPGC algorithm open-source software greatly improves the accessibility of LIDAR geospatial data and make a significant impact in the field of DEM compression. Safer driverless cars, fewer aviation accidents, and more accurate meteorology are made possible due to the improved compression algorithm.

Keywords: *Compression; Information Theory; LIDAR; Information Entropy Encoding*

1. Introduction

Recent years have seen an increasing use of LIDAR – Light Detection and Ranging – in technologies such as ground collision avoidance for aircraft [1], autonomous vehicles [2], and robotics [3]. In their use of LIDAR, these innovations have inspired some of the most exciting developments in computer science and engineering [4], supplementing the long-standing use of LIDAR in precision agriculture [5], surveying [6], and land management [7]. Historically, with no need for real-time decision-making, the constraints around the storage and the transmission of LIDAR data were scarcely felt [8]. But, with autonomy's new and growing insistence that LIDAR data be *instantaneously* precise, the previously-ignored constraints are beginning to bind: a paltry 5,000 square feet of a city block comprise, for example, an overwhelming 47 megabytes [9]. Storage of these very large datasets is difficult, and it is impossible to transmit extensive geospatial data at the speed and accuracy required by autonomous devices, whether via broadband or cellular connections [10]. To circumvent this problem, the large-scale systems of Google Waymo [11], Boston Dynamics [12], and Lockheed Martin [13] have chosen to create LIDAR point clouds in real time, with mounted sensors mapping a three-dimensional environment for a vehicle as it travels through space [14].

Put colloquially, these systems do not attempt to access virtual navigation maps so much as try to "see" the world in the moment, avoiding perceived obstacles while still moving forward.

The use of LIDAR by Google Waymo and other companies is not without its problems, however. LIDAR has been limited in its accessibility due to the cost of high quality sensors and the unpredictable reliability of concurrently generated data [15], to name just two vexing issues. Tesla, for its part, opted out of research into LIDAR storage and transmission, concerned about both the cost (10,000 dollars per sensor) and the viability of drive-time data-generation for autonomous technologies [16]. Pre-analyzed data is a proposed answer to this problem [17]; yet, as improvements in digital telemetry begin to slow, this solution is viable only with better data encoding and digital compression. Compression, in particular, is essential to store and transfer the geospatial data needed for the safe use of autonomous technologies [18]. Unsurprisingly, better compression algorithms are a source of great commercial and academic interest [19] [20] [21].

This paper presents a novel approach for the compression of LIDAR data, promoting its transfer, storage, and usability in both autonomous and analytical applications. As noted before, LIDAR scanners generally measure their surroundings to create "point clouds" of geospatial data [22]. The primary uncompressed file format for point-cloud data, **LAS**, stores the data as 32-bit integers with offsets and scaling defined in the file header [23]. It is significant that, due to this conversion, even the uncompressed binary file must be decoded with some computation before the memory may be accessed by an application [24]. LAS stores points in contiguous 152-bit structures encoded into the binary that define the location of points relative to the source, as well as identification data for each point in the vast array [25]. Once the data is unpacked, most technologies create a large OBJ mesh to allow for topological analysis of the environment and the source's positioning within it [26]. This information redundancy is a product of the unstructured nature of pure scanner output. With millions of points in a sample environment, information becomes too large not only to share, but, also, to handle locally without the most powerful of machines. Thus, data is often quantized into a two-dimensional grid known as a **raster**, which, under most specifications, shrinks the 152-bit entries of LAS into simple 16-bit integers filling an array of statically defined size [27]. As with all quantization, this is a lossy procedure [28]. In the process of scanning a three-dimensional environment, there are no guarantees of homogeneity in point distribution or resolution. The process of **rasterization** attempts to structure this data by normalizing and interpolating a point cloud to fit into a grid structure.

Rasterization itself can hardly be thought of as compression, for it is a non-reversible procedure; any data other than the index of a point in the 2-D array is discarded. It is fortunate that the United States Geological Survey, in its efforts to map the entire United States, first with sonar from the Space Shuttle program and today with satellite LIDAR, provides both point-cloud and rasterized forms of its data; this is an attempt to eliminate any confounding variables that may occur during rasterization [29]. Despite the destruction of up to 80% of the points in a LIDAR dataset, the process of rasterization is often beneficial. It simplifies for analysis the 3-D meshes that are generated, allowing for workable and high speed quadrilateral meshes (as opposed to triangles) [30]. Furthermore, rasters can be easily indexed and points can be compared simply without having to deal with complicated conversions [31]. Simple rasters become **Digital Elevation Maps (DEM)** with the encoding of relevant geospatial metadata. The prolific **GeoTIFF** standard importantly provides the latitude and longitude of the most northwestern point, the geo-referenced difference between each point in terms of lat-long, and a coordinate reference system (CRS) facilitating accurate projections of data removed from the equator [32].

Rasterized DEMs lend themselves easily to sophisticated means of compression by the exploitation of the more advanced patterns that can be inferred from a 2-D array [33]. Functionally, in this sense, the rasterized DEMs are no different than images, for which the development of lossy and lossless **codecs** is already a mature field. Understandably, DEMs are often compressed much like images, most commonly using either the PNG lossless or the JPEG2000 lossy standard for compression [34]. This paper outlines a novel codec for lossy compression of rasterized DEMs using a unique partitioning codec known as **General Purpose Geospatial Compression (GPGC)**.

2. Unique Requirements

Ultimately, GPGC is not an image codec; it is fine-tuned for the geospatial data compression requirements of automation technologies. Lossy compression algorithms are traditionally marked by four attributes, each of which is tested in this paper: decoding speed, encoding speed, compression ratio, and accuracy [35]. Due to the critically important use of geospatial data in autonomous decision making, two attributes among the four stand out for their particular importance, decoding speed and accuracy. Geospatial data applications anticipate extremely large decompressed data sizes. It is paramount that a novel algorithm provides a decoder that operates at a low algorithmic complexity, and it is even more important that data be encoded extremely accurately [36]. The latter point here is difficult to overstate. Highly efficient modern image compression algorithms manage high compression ratios by tolerating artifacts within an image [37]. The treatment of artifacts in decompressed rasters is a point of distinction between image compression algorithms and other types of raster codecs such as GPGC. The JPEG standard relies on encoding of the Discrete Cosine Transform, which is a highly efficient method of encoding data based on the intersections of cosine waves [38]. This suffers from both aforementioned issues: it is comparatively slow to decode (although tolerable), and it is prone to misrepresenting critical areas in a 3-D elevation map [39]. The transform relies on continuous curves, and thus struggles to encode hard edges (as would be found commonly in urban environments) without aberration at high ratios [40].

JPEG, unlike GPGC, encodes each area of a raster indiscriminately [41]. It provides its 8×8 operation across each area of the present array. In geospatial data, where not every part of a DEM is of equal importance (the slopes of a mountain must be resolved at at perfect accuracy, while consistently rolling fields can afford small errors), greater efficiency can be achieved by intelligent analysis of the terrain a codec is compressing [42]. Furthermore, use of linear sloping provides less artificating, an aspect of compression that is essential, even critical, for terrain data [43]. Small, flagrant errors in a dataset can be catastrophic for autonomous technologies that rely on perfect data [44]. The elimination of these types of inaccuracies has been a key consideration in the development of GPGC.

3. Overview

General Purpose Geospatial Compression, as presented in this paper, is a set partition coding algorithm. It is able to represent larger rasters by recursively generating a quad-tree structure, with each leaf node on the tree containing a three dimensional vector and size to define a planar approximation for an area on the raster given that size. The data is encoded as a depth-first search of the tree structure and reconstructed to a grid at decode-time. The constructor of each partition calculates a planar approximation section of the raster it contains, filling the content of the relevant leaf node with the orthogonal vector. The accuracy of this planar approximation determines whether the algorithm will recurse and create a new sublevel to the tree. GPGC takes a novel approach to analyzing its partitions; while *JPEG2000* [45] and *GEDACS* [46], two competing compression algorithms, resolve regions based on maximum error thresholds, the presented codec notably interprets each differential as a probability. This allows for a smarter analysis, resolving areas of high **information entropy** at perfect resolution, while compressing regions of lower entropy at a very high efficiency. The average information entropy of a region per point is calculated with reference to the planar approximation, and the program **quadrisects** the relevant region, improving the resolution four-fold recursively until the quad-tree approximation is deemed sufficient by parameters defined to the encoder

4. Raster Representation

A compressible raster is defined by pixel values $p_{m,n}$, where (m, n) is the point coordinate in the raster. Multidimensional arrays are represented with bold letters, but GPGC does not anticipate any data of a greater dimension than 2. It is a convention to define the process of set partition coding with an uppercase omega:

$$\mathbf{R} = \Omega(\mathbf{C}).$$

The $\Omega(\mathbf{C})$ transform in this case is the process of generating hierarchical quadtrees representing the 2-D array $\mathbf{r}$ that are then decoded to form the decompressed raster $\mathbf{c}$.

5. Compression Procedures

5.1. Mosaic Raster Preprocessing

Upon opening the GeoTIFF file into a 2-D array, the encoder immediately pre-processes the raster based on its dimension to create a **mosaic** from the top left corner, attempting the fill the area with squares of side length 2^n. It fills the data with the largest possible size from the constraint. From there, it duplicates itself to form a 2×2 square overlapping the original raster. Assuming that the size of the outlying regions extends beyond the edges of the raster, it partitions itself and reduces itself until the edges are properly defined with fragments of 2^n-sized squares. Any partition that lies outside the raster entirely is discarded. Once an appropriate size is determined, its size and offset is pushed back to a a standard vector in the provided C++ implementation. The partitions defined here work in a similar manner to the partitioning in GPGC. However, the two methods should not be confused; the mosaics are entirely a pre-processing method to allow for the encoding of efficient, square, binary partitions. The encoding process occurs for each 2^n square that fills the mosaic and each square is encoded into a separate tree structure. When a new fragment is loaded into memory by the encoder, it generates a 16-bit prefix that identifies the beginning of a new tree. Following this magic, $\log_2(m)$ is encoded as an 16-bit integer, followed by the number of mosaics of equal size, where m represents the decompressed side length. This is a particularly important step. As can be visualized in Figure 1, layers often form of fragments with the same dimension. This is handled easily by simply encoding the number of contiguous tree structures with the same size to avoid providing a large header for an extremely small tree (as would be found near the edges of images). The interpretation of when a tree is filled is handled by the decoder as will be later described in detail.

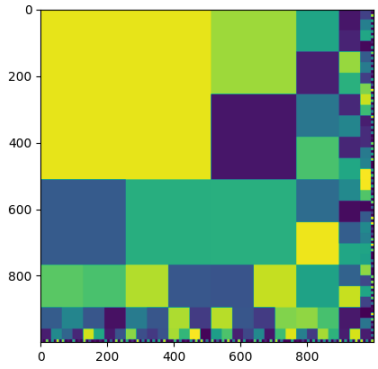

Figure 1. Example of mosaic pre-processor divisions

The order of the encoded fragments need not be in the order in which they were created recursively. To do so would actually require many additional headers to be generated. In the edges of a raster as sizes quickly change, a header would have to be defined for no more than two small trees. Because the fragments can be encoded in any order that allows unambiguous calculation of their offsets, it is obvious from Figure 1 that sizes decrease radially towards the edge. Therefore, the data can most efficiently be encoded into concentric bands radiating from the top left corner named the origin. The maximum binary size can be simply calculated as

$$H\lfloor \log_2(M) \rfloor$$

where H is a constant $16+8+8$ bit identifier of a new tree size and m is the maximum dimension of a raster. In practice, this is of a negligible size in an encoding, and it is believed to be advantageous to encoding rectangular data blocks or dealing with half-partitions.

5.2. Partition Matrix Representation

Because of the outlined mosaicing process, each partition is guaranteed to be square and of size 2^n. Each altitude encoded into the DEM can be interpreted as a position in a matrix. This notation is convenient because it allows unambiguous notation, but more significantly, we can perform relevant matrix operations on this structure. Following this, the process of generating a Least Squares Regression Plane for the topography contained within the matrix is straightforward.

If a partition $\mathbf{p}$ is represented as

$$\mathbf{p} = \begin{bmatrix} a_{11} & \dots & b_{1M} \\ \vdots & \ddots & \vdots \\ b_{M1} & \dots & b_{MM} \end{bmatrix},$$

it is simple to calculate the normal vector for a plane of best fit. If you construct a matrix with M_i^2 rows, where each row outlines an index for each point with a third column of all 1s. Following $A\mathbf{x} = b$, the orthogonal vector $\mathbf{x}$ can be calculated by defining b as a vector of rank 1 with each row aligned with the index defined in A as specified below:

$$A\mathbf{x} = b \Rightarrow \begin{bmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \\ \vdots & \vdots & \vdots \\ 1 & 2 & 1 \\ 2 & 2 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_m & 1 \end{bmatrix} \begin{bmatrix} \hat{i} \\ \hat{j} \\ \hat{k} \end{bmatrix} = \begin{bmatrix} C_{11} \\ C_{21} \\ \vdots \\ C_{12} \\ C_{22} \\ \vdots \\ C_{mn} \end{bmatrix}.$$

.

The orthogonal vector can be calculated through matrix arithmetic:

$$\mathbf{x}_m = \begin{bmatrix} \hat{i} \\ \hat{j} \\ \hat{k} \end{bmatrix} = (A^T A)^{-1} A^T B.$$

This fit vector $\mathbf{x}_m$ is calculated each time a partition object is constructed. The normal vectors are the key element of the set partition code. The rest of the algorithm is simply the means of evaluating the accuracy of the planar approximation created by the vector.

The array $\mathbf{p}$ is iterated upon, and the residual d at point (m, n) is calculated simply as

$$d_{mn} = |\mathbf{C}_{mn} - (\hat{i}m + \hat{j}n + \hat{k})|$$

At this point, the first of two parameters taken in by the encoder, σ, is considered. Sigma represents the standard deviation of expected errors. This is another key distinction from other set partition coding algorithms. Maximum error is not necessarily considered, and neither is average error (although these can be passed as flags to the official implementation). The only consideration is standard deviation. This allows us to interpret each point in a partition not as an integer residual, but instead as a probability. This distinction is fundamental to the entropy based encoder described ahead.

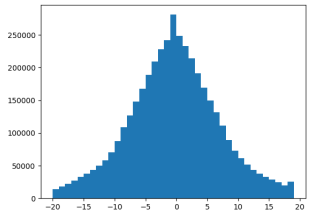

Figure 2. distribution of differentials

To convert this to a probability, the displacement is converted to a z-score. The traditional z-score formula $\frac{d-\mu}{\sigma}$ becomes $\frac{d}{\sigma}$ under the expectation that the average error above and below the regression plane will be zero. From this z-score, the probability of each point is calculated. The normal probability density function is integrated up to the desired z-score,

$$\mathcal{P}(\mathbf{C}_{mn}) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} \exp \frac{x^2}{2} dx$$

It is surprisingly easy to approximate this with crude method of just assuming a lower bound of -10 as $-\infty$ as the difference is less than 2×10^{-6} which is an acceptable error for this calculation. The unique contribution of GPGC is its use of entropy encoding. Literally, the entropy represents

the amount of binary required to express a certain probability, but, more generally, it is a helpful model for understanding the uncertainty of data. This is the critical use of it in geospatial data, where it is able to isolate areas that can be compressed aggressively from areas that require a greater resolution.

Following Claude Shannon's formula for information [38], the total information content of a row X in $\mathbf{C}$ is defined as

$$H(\mathbf{C}_X) = \sum_{n=1}^{K} \log_2 P(C_{Xn}).$$

It follows that the average information content of the entire chunk C is

$$H(\mathbf{C}) = \sum_{n=1}^{M} \sum_{m=1}^{M} \frac{\log_2 P(\mathbf{C}_{mn})}{M^2}.$$

5.3. Partitions

The Set Partition Coding (SPC) algorithm maintains acceptable resolution by dividing high-information chunks with size $2^n \times 2^n$ into four equal sub-chunks with size $2^{n-1} \times 2^{n-1}$ (quadrisecting). The partition objects are constructed recursively based on the average information entropy of the partition $\mathbf{C}$. If the average entropy is within a tolerance ζ that is supplied as a runtime flag to the executable, then a leaf node is bitshifted into a bytestream passed by pointer to the object. Otherwise, four child structures are generated and the memory is freed for the parent structure. Each child structure is obviously subjected to the same evaluation. At some point with a minimum size 2^1, the planar approximations will meet the entropy tolerance and be encoded into the bytestream. What happens incidentally under this procedure is an encoding of a depth-first search of a quadtree structure. This process is repeated along each recursion until the square of each leaf node equals the square of the original. Given a set $\mathbf{R}$ of encoded nodes that are the result of transform Ω, described above, with each with size m representing the original raster with size M, it is clear that

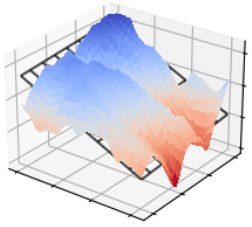

Figure 3. Planar approximation of high-entropy region

$$\sum_{i=0}^{|\mathbf{R}|} (\mathbf{R}_m^{(i)})^2 = M^2.$$

5.4. Encoding

Each partition of the raster can be represented as a four 16-bit primitives in the form $\{\hat{i}, \hat{j}, \hat{k}, M\}$ where M is the size of the partition. Elements $\hat{i}, \hat{j}, \hat{k}$ are 16 bit representations of the elements of orthogonal vector $\mathbf{x}_m$. The $\hat{i}$ and $\hat{j}$ elements are represented as 16-bit IEEE754 half-precision floats. This does lose an observable amount of data, but the precision of 32-bit floats is deemed unnecessary for the representations. It is acceptable for $\hat{i}, \hat{j}$ to be half-precision floats due to the small size of realistic altitudes and the tolerable error expected in each data point. Realistically, no values will be encountered on Earth requiring a greater range than unsigned 16-bit integers or more float precision than 2^{-14} (for a half-precision float). And $\hat{k}$ and M are 16-bit integers due to less need still for precision and more need for range. Furthermore, the compressed raster obviously cannot exceed the 16-bit precision of the original raster. In the official implementation, the four 16-bit quantities are serialized into a 64 bit object pointer, which is bitshifted into a stored position on the encoding object that is passed by pointer to each partition.

5.5. Decoding

The problem of decoding a quad tree is a simple one to a human, who are capable of immediately understanding the larger shape of the tree [39]. However, the obvious human solution is of an NP

efficiency and is absolutely not fit for decoding [40]. The problem is identifying from a position in a depth-first search the x and y offset of a partition within a raster. After the offsets are calculated, it is trivial to reconstruct the raster from the orthogonal vectors. There has been considerable academic work done on the reconstruction of quadtree decomposition [41] [42] [43], but GPGC opts for a simpler, and likely faster, approach. Provided an array of $\log_2 M$ for each partition in the tree and an original location of 0, the offset L of point d at position m in the lowest level partition can be calculated as

$$L_n = L_{n-1} + \frac{1}{2^m}$$

This formula is nearly impossible, and certainly impractical, to understand in terms of every variable, as many of the variables change recursively throughout the decoding process. It is more efficient to think of the formula in terms of the following algorithm:

Algorithm 1 Quadtree Raster Reconstruction

Require: $s = \{\log_2 x \in \mathbb{Z} : \forall x < 2^7\}$
$L_x = [0]$
$L_y = [0]$
$buf = [0]$
$i = 0$
while $i < |s|$ **do**
  while $buf_i < s_i$ **do**
    Increment buf_i
    for all $c \in 0 < x < 3$ **do**
      Insert buf_i at position $i + 1$ in buf
    end for
    Insert $L_{xi} + 2^{-buf_i}$ at position $i + 1$ in L_x
    Insert L_{xi} at position $i + 2$ in L_x
    Insert $L_{xi} + 2^{-buf_i}$ at position $i + 3$ in L_x

    Insert L_{yi} at position $i + 1$ in L_y
    Insert $L_{xi} + 2^{-buf_i}$ at position $i + 2$ in L_y
    Insert $L_{xi} + 2^{-buf_i}$ at position $i + 3$ in L_y
  end while
  $i++$
end while
end

6. Accuracy of Encoded Data

GPGC was tested against seven different compression algorithms. Three geospatial algorithms were considered: GEDACS (developed by NASA) [46], MrSID (Los Alamos National Laboratory) [47], and ECW (Hexagon AB) [48]. Three binary codecs were identified: ZSTD (Facebook) [49], PACKBITS (Apple) [50], and Deflate (also known as Lempel-Ziv 77) (MIT) [51]. Finally, GPGC was compared against the JPEG lossy image algorithm, seen as the gold standard for extremely high-efficient compression, although the reasons for its unsuitable use have been explored thoroughly in this paper. Using the GDAL CLI and the proprietary MSVC solution for GEDACS, each possible codec was implemented onto 135 randomly selected United States Geological Survey Continental 1-arc second rasters (12,967,201 points per raster) captured from the SRTM project. This dataset, downloaded from [29], was chosen due to its unprecedented reliability and the simplicity of the existing encoding, ensuring as many of the codecs could be tested as possible. Testing was done inside of a Jupyter IPython notebook where each file was evaluated for compression ratio and mean average error. To calculate the percent error of the compression against the measured

data, we can use the formula:

$$\text{MAE} = \frac{1}{M^2}\sum_{i=0}^{M^2} |z_{\text{original}} - z_{\text{decomp}}|.$$

Compared by compression ratio and mean average error (MAE), GPGC significantly outperforms its direct competitors. Most notably, the compression ratio of GPGC is triple that of its nearest competitor GEDACS, a similar NASA algorithm for geospatial rasters. Furthermore, the average ratio is more than double the posted ratio of ECW, a proprietary format owned by Hexagon AC and used for that company's autonomous vehicle project. (Note the median compression ratio for ECW has not been quantified in any peer-reviewed articles and is cited only from information that Hexagon itself provides.)

Compression Benchmarks					
Compression	Comp. Ratio			Percent	Average
Algorithm	25%	Med.	75%	Decrease	Error
GPGC	**8.2**	**24.7**	**72.8**	**95.97%**	**6.71**
GEDACS	7.7	8.48	9.2	88.21%	15.7
MrSID	4*	n/d*	10*	n/d*	0
ECW	n/d*	10*	15*	90%*	n/d
ZSTD PRED1	1.7	2.12	4	52.12%	0

The above table compares the compression ratio of GPGC at the 25th, 50th, and 75th percentile as compared with other algorithms. Percent decrease is a function of the median, observing the size of the median (50th percentile) compressed binary with the original file. It is primarily intended to aid the reader in understanding the magnitude of the compression ratio. On each percentile, GPGC leads the competition in terms of compression efficiency For Mean Average Error, GPGC also leads the similar lossy algorithms ECW and GEDACS. For ECW and MrSID, benchmarks could not be acquired from their respective proprietors. The numbers presented in the table are extrapolated from their published, if proprietary, benchmarks. These self-provided benchmarks have no peer review and likely represent a best-case realistic scenario for geospatial compression.

6.1. Performance

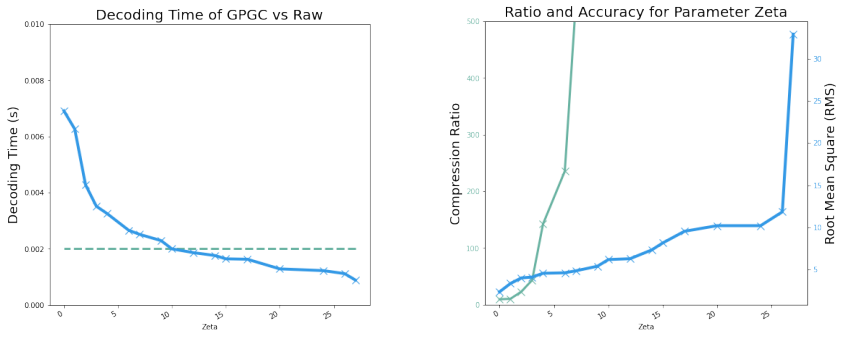

Figure 4. Altitude and Z-Score Histograms

Figure 4 plots the Root Mean Square Error (RMS) against the median compression ratio for a set of 10 test rasters that were evaluated at each level. Note the differing axes for each quantity. The compression ratio is purely a function of the number of leaf nodes. It follows that with a less stringent threshold, the data can be represented in fewer leaf nodes. Historically, quadtree decomposition algorithms such as GPGC have suffered with precision at these thresholds (which are, again, represented in terms of the parameter zeta). However, with clever bitwise operations

and efficient storage of data, the GPGC algorithm accomplishes high compression ratios at great accuracy and very high compression ratios when a greater tolerance is allowed. The algorithm is designed to work anywhere with a zeta between one and 25. However, the best results are achieved between two and five. At a zeta of one, meaning the compression will be nearly lossless, the compression ratio is often less than two, while, when zeta is greater than five, the error because unsuitable for real data and obvious on visual inspection. For all measurements, a very high information entropy threshold was used so as to show the relationship without interference.

An often overlooked but highly important part of a compression algorithm is its decoding speed. GPGC opts for a novel yet efficient decoder that is delineated in the previous section and that reconstructs a depth-first search of a quad-tree structure. The industry tool for decoding GeoTIFF and DTED rasters is GDAL [52]. GPGC is compared against the GDAL uncompressed speed at various values of zeta. Interestingly, above a zeta of 15, GPGC becomes faster than GDAL. The only explanation for this is that the bare-bones implementation of GPGC skips the complex header generics of the TIFF specification, allowing for slightly faster encoding when few operations need to be performed.

7. Discussion

The implementation of a more advanced and efficient LIDAR compression algorithm like GPGC holds the potential to significantly enhance public safety by improving the speed and accuracy of data processing in autonomous vehicles [53], drones [54], and industrial robots [55]. Most notably, the ability to process LIDAR data with greater alacrity and precision will reduce the likelihood of dangerous accidents, whether car crashes, ground collision of aircraft, or workplace mishaps [56]. This, in turn, will accelerate the adoption of these new, revolutionary autonomous technologies [57].

A better compression algorithm like GPGC will also result in notable cost savings, as the large datasets generated by LIDAR consume significant storage resources. Globally, 42 billion dollars a year is spent on data storage [58]. While not all of this stored data arises from LIDAR, it shows the extraordinary expense of the expanding informationalized world, some of which could be avoided through better compression.

An improved compression algorithm will, as a final matter, enhance accessibility to socially valuable LIDAR data in fields such as transportation [59], urban planning [60], and environmental monitoring [61]; more and better compressed data of these types can be deployed on hand-held devices, for example [62]. The increased access to LIDAR data would allow for greater research opportunities and provide businesses and governments with a more comprehensive understanding of their environments [63]. Other benefits of the novel GPGC algorithm could be seen in forensic pathology [64], bathymetry [65], and archaeology [66], to name just a few disciplines.

8. Conclusion

With improving LIDAR sensor resolution and growing data volume, how to efficiently store and transmit LIDAR data becomes a challenging problem in many 3D applications. To address this challenge, I propose a novel geospatial data compression algorithm named GPGC (General Purpose Geospatial Compression) that was implemented using the programming language C++. GPGC is specifically designed for geospatial data and aims to improve upon previous algorithms in terms of accuracy and efficiency. It uses a quad-tree set-partition-code to divide the data into a tree structure, with each node representing a sub-region of the data. The data in each node is then encoded as a vector representing the slope of the terrain in that region. This tree structure is further compressed using entropy coding and prediction-based techniques.

I evaluated GPGC on a variety of geospatial data sets and found that, compared to other compression algorithms, it was effective in terms of both accuracy and efficiency. GPGC was able to significantly reduce the size of the data while preserving the integrity of the information. GPGC was also designed to be more sensitive to outliers and extraordinary terrain features, making it particularly useful for applications where accuracy is critical.

The open source nature of GPGC allows it to be used on a wide range of computer hardware and makes the algorithm accessible to a broad community of users. This is an important advantage as users can easily integrate GPGC into their own systems and modify it to meet specific needs. The use of C++ as the implementation language also promotes efficient execution, making GPGC a suitable choice for use in resource-constrained environments such as embedded systems or on large datasets where performance is critical.

References

[1] A. Kotb, S. Hassan, and H. Hassan, "A comparative study among various algorithms for lossless airborne LIDAR data compression," *2018 14th International Computer Engineering Conference (ICENCO),* 2018.

[2] A. Varischio, F. Mandruzzato, M. Bullo, M. Giordani, P. Testolina, and M. Zorzi, "Hybrid point cloud semantic compression for automotive sensors: A performance evaluation," *ICC 2021 - IEEE International Conference on Communications*, 2021.

[3] B. Anand, V. Barsaiyan, M. Senapati, and P. Rajalakshmi, "Real time LIDAR point cloud compression and transmission for Intelligent Transportation System," *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, 2019.

[4] B. Douillard, J. Underwood, N. Kuntz, V. Vlaskine, A. Quadros, P. Morton, and A. Frenkel, "On the segmentation of 3D LIDAR point clouds," *2011 IEEE International Conference on Robotics and Automation*, 2011.

[5] B. Pradhan, S. Mansor, A. R. Ramli, A. R. Mohamed Sharif, and K. Sandeep, "LIDAR data compression using wavelets," *Remote Sensing for Environmental Monitoring, GIS Applications, and Geology V*, 2005.

[6] K. Koenig, B. Höfle, M. Hämmerle, T. Jarmer, B. Siegmann, and H. Lilienthal, "Comparative classification analysis of post-harvest growth detection from terrestrial LIDAR point clouds in precision agriculture," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 104, pp. 112–125, Jun. 2015.

[7] J. Evans, A. Hudak, R. Faux, and A. M. Smith, "Discrete return LIDAR in natural resources: Recommendations for project planning, data processing, and deliverables," *Remote Sensing*, vol. 1, no. 4, pp. 776–794, Oct. 2009.

[8] D. Saloman and G.Motta, *Handbook of data compression.* London: Springer London, 2010.

[9] S. Liu and J.-L. Gaudiot, "Autonomous vehicles lite self-driving technologies should start small, go slow," *IEEE Spectrum*, vol. 57, no. 3, pp. 36–49, Mar. 2020.

[10] C. Tu, E. Takeuchi, A. Carballo, and K. Takeda, "Point cloud compression for 3D LIDAR sensor using recurrent neural network with residual blocks," *2019 International Conference on Robotics and Automation (ICRA)*, 2019.

[11] "Combine LIDAR and cameras for 3D object detection - Waymo," *Louis Bouchard*, Mar. 25, 2022. https://www.louisbouchard.ai/waymo-lidar/ (accessed Sept. 14, 2022).

[12] M. Fallon, "Accurate and robust localization for walking robots fusing kinematics, inertial, vision and LIDAR," *Interface Focus*, vol. 8, no. 4, p. 20180015, Jun. 2018.

[13] B. W. Short, T. N. Bourbeau, L. Fuller, J. Curriden, and M. J. Dahlin, "Differentiation and application of global shutter flash LIDAR," opg.optica.org, Sep. 29, 2019. https://opg.optica.org/abstract.cfm?uri=LSC-2019-LM2B.3 (accessed Feb. 05, 2023).

[14] "Waypoint - The official Waymo blog: The Waymo Driver Handbook: Teaching an autonomous vehicle how to perceive and understand the world around it," *Waypoint – The official Waymo blog.* https://blog.waymo.com/2021/10/the-waymo-driver-handbook-perception.html

[15] D. Gohring, M. Wang, M. Schnurmacher, and T. Ganjineh, "Radar/LIDAR sensor fusion for car-following on highways," *The 5th International Conference on Automation, Robotics and Applications*, 2011.

[16] "Why LIDAR is doomed," www.voltequity.com. https://www.voltequity.com/article/why-lidar-is-doomed (accessed Oct. 15, 2022).

[17] C. Tu, E. Takeuchi, A. Carballo, and K. Takeda, "Real-Time Streaming Point Cloud Compression for 3D LIDAR Sensor Using U-Net," *IEEE Access*, vol. 7, pp. 113616–113625, 2019.

[18] I. Maksymova, C. Steger, and N. Druml, "Review of LIDAR sensor data acquisition and compression for automotive applications," *Proceedings*, vol. 2, no. 13, p. 852, Dec. 2018.

[19] E. Javanmardi, Y. Gu, M. Javanmardi, and S. Kamijo, "Autonomous vehicle self-localization based on abstract map and multi-channel LIDAR in urban area," *IATSS Research*, vol. 43, no. 1, pp. 1-13, 2019.

[20] F. Nardo, D. Peressoni, P. Testolina, M. Giordani, and A. Zanella, "Point cloud compression for efficient data broadcasting: A performance comparison," *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, 2022.

[21] G. Jóźków, C. Toth, M. Quirk, and D. Grejner-Brzezinska, "Compression strategies for LIDAR waveform cube," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 99, pp. 1-13, 2015.

[22] H. Houshiar and A. Nuchter, "3D point cloud compression using conventional image compression for efficient data transmission," *2015 XXV International Conference on Information, Communication and Automation Technologies (ICAT)*, 2015.

[23] H. Takizawa, K. Yamada, and T. Ito, "Vehicles detection using sensor fusion," *IEEE Intelligent Vehicles Symposium*, 2004.

[24] H. Yin, Y. Wang, L. Tang, X. Ding, S. Huang, and R. Xiong, "3D LIDAR map compression for efficient localization on resource constrained vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 2, pp. 837-852, 2021.

[25] I.M. Pu, *Fundamental data compression*, Oxford: Butterworth-Heinemann, 2006.

[26] I. Maksymova, C. Steger, and N. Druml, "Review of LIDAR sensor data acquisition and compression for automotive applications," *EUROSENSORS 2018*, 2018.

[27] J.A. Storer, *Data compression: Methods and theory*. Rockville: Computer Science Press, 1988.

[28] J. Cui, H. Zou, X. Kong, X. Yang, X. Zhao, Y. Liu, W. Li, F. Wen, and H. Zhang, "Poconet: Slam-oriented 3D LIDAR point cloud online compression network," *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[29] "The National Map - Data Delivery | U.S. Geological Survey," www.usgs.gov. https://www.usgs.gov/the-national-map-data-delivery.

[30] K. Sato, R. Shinkuma, T. Sato, E. Oki, T. Iwai, D. Kanetomo, and K. Satoda, "Prioritized transmission control of point cloud data obtained by LIDAR Devices," *IEEE Access*, vol. 8, pp. 113779-113789, 2020.

[31] L. Qingqing, J. P. Queralta, T. N. Gia, H. Tenhunen, Z. Zou, and T. Westerlund, "Visual odometry offloading in internet of vehicles with compression at the edge of the network," *2019 Twelfth International Conference on Mobile Computing and Ubiquitous Network (ICMU)*, 2019.

[32] N. Ritter and M. Ruth, "The GeoTiff data interchange standard for raster geographic images," *International Journal of Remote Sensing*, vol. 18, no. 7, pp. 1637–1647, May 1997.

[33] M. Latella, F. Sola, and C. Camporeale, "A density-based algorithm for the detection of individual trees from LIDAR Data," *Remote Sensing*, vol. 13, no. 2, p. 322, 2021.

[34] N. Certad, W. Morales-Alvarez, and C. Olaverri-Monreal, "Road markings segmentation from LIDAR point clouds using reflectivity information," *2022 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, 2022.

[35] R. Pandian, "Evaluation of image compression algorithms," *IEEE Xplore*, Feb. 01, 2015. https://ieeexplore.ieee.org/abstract/document/7108244 (accessed Feb. 05, 2023).

[36] S. Dhawan, "A review of image compression and comparison of its algorithms," *International Journal of Electronics Communication Technology* 2.1 (2011): 22-26.

[37] Q. Wang, L. Jiang, X. Sun, J. Zhao, Z. Deng, and S. Yang, "An efficient LIDAR point cloud map coding scheme based on segmentation and frame-inserting network," *Sensors*, vol. 22, no. 14, p. 5108, 2022.

[38] A. M. Raid, W. M. Khedr, M. A. El-dosuky, and W. Ahmed, "JPEG image compression using Discrete Cosine Transform - A survey," arXiv:1405.6147 [cs], May 2014, Accessed: Feb. 05, 2023. [Online]. Available: https://arxiv.org/abs/1405.6147.

[39] P. Ruiz, M. Ross, and J. Sah, "A database of Tree Islands within the Mustang Corner Fire Incident of 2008." Accessed: Feb. 05, 2023. [Online]. Available: https://digitalcommons.fiu.edu/cgi/viewcontent.cgi?article=1084context=sercrp

[40] R. W. Wolcott and R. M. Eustice, "Fast LIDAR localization using multiresolution Gaussian mixture maps," *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[41] R. Wang, L. Liu, and W. Shi, "HydraSpace: Computational data storage for autonomous vehicles," *2020 IEEE 6th International Conference on Collaboration and Internet Computing (CIC)*, 2020.

[42] R.-T. Juang, "The implementation of remote monitoring for autonomous driving," *2019 4th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*, 2019.

[43] S. A. Bagloee, M. Tavana, M. Asadi, and T. Oliver, "Autonomous vehicles: Challenges, opportunities, and future implications for transportation policies," *Journal of Modern Transportation*, vol. 24, no. 4, p. 284-303, 2016.

[44] S. K. Lodha, E. J. Kreps, D. P. Helmbold, and D. Fitzpatrick, "Aerial LIDAR data classification using support vector machines (SVM)," *Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)*, 2006.

[45] "JPEG2000 Image Compression Fundamentals, Standards and Practice: Image Compression Fundamentals, Standards and Practice | BibSonomy," www.bibsonomy.org. https://www.bibsonomy.org/bibtex/372a39b3a4576d2e5ec886628f002e1c (accessed Feb. 05, 2023).

[46] M. A. Skoog, K. Prosser, and L. Hook, "Ground Collision Avoidance System (Igcas)," ntrs.nasa.gov, Apr. 2017, Accessed: Feb. 05, 2023. [Online]. Available: https://ntrs.nasa.gov/citations/20170004339

[47] MrSID (2021), MrSID Home. (Accessed Nov. 11, 2022).

[48] ECW (2013), ECW Home. (Accessed Nov. 1, 2022).

[49] ZSTD (2016), ZSTD Home. (Accessed October 22, 2022).

[50] PackBits (2020), PackBits Home. (Accessed December 12, 2022).

[51] Deflate (1996), Deflate Home. (Accessed on January 3, 2023).

[52] GDAL (2023), GDAL Home. (Accessed on January 30, 2023).

[53] X. Sun, S. Wang, M. Wang, S. S. Cheng, and M. Liu, "An advanced LIDAR point cloud sequence coding scheme for autonomous driving," *Proceedings of the 28th ACM International Conference on Multimedia*, 2020.

[54] X. Sun, S. Wang, M. Wang, Z. Wang, and M. Liu, "A novel coding architecture for LIDAR point cloud sequence," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5637-5644, 2020.

[55] X. T. Nguyen, H. Kim, and H.-J. Lee, "A gradient-aware line sampling algorithm for LIDAR scanners," *IEEE Sensors Journal*, pp. 1-1, 2020.

[56] X. T. Nguyen, H. Kim, and H.-J. Lee, "An efficient sampling algorithm with a K-NN expanding operator for depth data acquisition in a LIDAR System," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 12, pp. 4700-4714, 2020.

[57] X. Zhou, C. R. Qi, Y. Zhou, and D. Anguelov, "Riddle: LIDAR data compression with Range Image Deep Delta encoding," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[58] P. Taylor, "Topic: Data storage," *Statista*, [Online]. Available: https://www.statista.com/topics/3150/data-storage/. [Accessed: 17-Jan-2023].

[59] Y. Lee and S. Park, "A deep learning-based perception algorithm using 3D LIDAR for autonomous driving: Simultaneous segmentation and detection network (ssadnet)," *Applied Sciences*, vol. 10, no. 13, p. 4486, 2020.

[60] Z. Zhu et al., "Understanding an urbanizing planet: Strategic directions for remote sensing," *Remote Sensing of Environment*, vol. 228, pp. 164–182, Jul. 2019.

[61] H. Yan, "Dimensional analysis of regional environmental planning based on NPP/VIIRS lighting data," *Computational Intelligence and Neuroscience*, vol. 2022, pp. 1–10, Aug. 2022.

[62] V.-I. Ungureanu, B.-A. Trutiu, I. Silea, P. Negirla, C. Zimbru, and R.-C. Miclea, "Automatic mapping of a room using LIDAR-based measuring sensor," *2019 22nd International Conference on Control Systems and Computer Science (CSCS)*, 2019.

[63] M. S. Nabers, "LIDAR technology achieving objectives for the public sector in new ways ," Strategic Partnerships, Inc., Jan. 28, 2022. https://www.spartnerships.com/lidar-technology-achieving-objectives-for-the-public-sector-in-new-ways/ (accessed Feb. 05, 2023).

[64] . Maiese, A. C. Manetti, C. Ciallella, and V. Fineschi, "The introduction of a new diagnostic tool in forensic pathology: LIDAR sensor for 3D autopsy documentation," *Biosensors*, vol. 12, no. 2, p. 132, Feb. 2022.

[65] A. Szafarczyk and C. Toś, "The Use of Green Laser in LIDAR Bathymetry: State of the Art and Recent Advancements," *Sensors*, vol. 23, no. 1, p. 292, Dec. 2022.

[66] B. Štular, E. Lozić, and S. Eichert, "Airborne LIDAR-Derived Digital Elevation Model for Archaeology," *Remote Sensing*, vol. 13, no. 9, p. 1855, May 2021.