10.31341/jios.45.1.15
Open Access
UDC 004.421
Original Scientific Paper

ΔRLE: Lossless data compression algorithm using delta transformation and optimized bit-level run-length encoding

Branislav Madoš *branislav.mados@tuke.sk*
Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Košice, Slovak Republic

Zuzana Bilanová *zuzana.bilanova@tuke.sk*
Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Košice, Slovak Republic

Ján Hurtuk *jan.hurtuk@tuke.sk*
Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Košice, Slovak Republic

Abstract

Lossless data compression algorithms can use statistical redundancy to represent data using a fewer number of bits in comparison to the original uncompressed data. Run-Length Encoding is one of the simplest lossless compression algorithms in terms of understanding its principles and software implementation, as well as in terms of temporal and spatial complexity. If this principle is applied to individual bits of original uncompressed data without respecting the byte boundaries, this approach is referred to as bit-level Run-Length Encoding. Algorithm for lossless data compression, proposed in this paper, optimizes bit-level Run-Length Encoding data compression, uses special encoding of repeating data blocks, and, if necessary, combines it with delta data transformation or representation of data in its original form intending to increase compression efficiency compared to a conventional bit-level Run-Length Encoding approach. The advantage of the algorithm proposed in this paper is in the increase of the compression ratio in comparison to the bit-level Run-Length Encoding, with the higher time and space consumption as the trade-off. Test results, that were obtained by compression of the segmentation metadata of different volume datasets take place in the last part of the paper.

Keywords: data compression, lossless data compression, run-length encoding, RLE, bit-level run-length encoding, three dimensional data, volume data, sparse voxel octrees, sparse voxel directed acyclic graph, symmetry aware sparse voxel directed acyclic graph

1. Introduction

Information and Communication Technologies (ICT) produce enormous amounts of data daily. Some are discarded after they are created, transferred, and used, but a large portion is subsequently archived, extending the total amount of data currently included in ICT. The data transfer itself represents a significant burden for ICT, so we are looking for ways how to model, optimize and minimize it [1][2]. A large amount of data exists in multiple copies - in uncompressed or compressed forms. In practice, some types of data are even preferentially and almost exclusively stored and transmitted in a compressed version and often happens that they exist in their original form only for a short period of time - during their acquisition and initial processing. Examples are multimedia data such as images, audio sequences, and especially video sequences. A feature film in compressed form requires a storing capacity in GBs of the secondary storage device depending on the required quality and the encoder used. Its uncompressed version of 1920×1080 pixels (FullHD) at 24b/pixel and 25 frames per second with a recording time of 120 minutes requires up to 1TB of storage space on the secondary storage device. Storage of the uncompressed version and its distribution whether through storage media for the secondary storage devices or over computer networks would be impractical and costly for mass use.

Lossy data compression with a higher compression ratio is mostly used in multimedia. Lossless compression usually does not reach such high compression ratios but may be similarly successful in certain cases with domain-specific algorithms. An example could be the compression of a geometry of voxelized three-dimensional images. While in the uncompressed form its binary representation at the resolution of $64K^3$ voxels (65536 × 65536 × 65536) at 1b/voxel may have a volume up to 32TB. The compression using octant trees and the data structures derived from them allows lossless compression to compress the data down to hundreds of MB depending on the image content. Data structures useful for storing a compressed form of this geometry of voxelized image data include Sparse Voxel Octrees (SVO), Efficient Sparse Voxel Octrees (ESVO) [3], Sparse Voxel Directed Acyclic Graphs (SVDAG) [4] or Symmetry-aware Sparse Voxel Directed Acyclic Graphs (SSVDAG) [5]. Out-of-Core compression algorithm for creating the SVO data structure is described in detail in [6].

In both lossless and lossy algorithms, there are universal as well as domain-specific compression algorithms with different data compression success rates. Their existence is given not only by historical development but also by different temporal and spatial complexity which justifies the use of less efficient compression algorithms that provide lower memory requirements and processing power of the processor while keeping a lower but sufficient compression ratio. Often, such lightweight compression algorithms are implemented on devices that have a significantly limited amount of memory, lower processing power or there is a requirement of a high data throughput such as in data streaming.

Run-Length Encoding (RLE) is one of the simplest methods of lossless data compression that replaces sequences (runs) of the same elements by single element

code and the number of elements in the run (length of the symbol sequence). Element code and related encoded length of the run is called the RLE packet (RLEP).

The history of RLE employment dates back to 1967, when it has been used in television signal transmission [7]. Data that are about to be encoded can be a one-dimensional (1D) stream of data, or may be multidimensional as in the case of images that are a two-dimensional (2D) grids of pixels or a volumetric data that are a three-dimensional grids of voxels (3D).

RLE schemes used for image compression are divided into bit-level, byte-level, and pixel-level compression schemes depending on the type of an encoding atomic element.

Bit-level compression schemes encode runs of bits while ignoring byte or even word boundaries. Typically, one byte is needed for the RLE packet encoding where one of the bits is used to represent the respective bit of the symbol and the 7 other bits represent an unsigned integer representation of the number of these bits in the run. Their number can be in the range $< 0; 127 >$. The byte-level encoding schemes encode runs of identical bytes and most often they create double-byte RLE packets where one byte is the run length in the range $< 0; 255 >$ and is referred to as a *count value* and the second byte represents the value of the symbol itself and is referred to as a *run value*. Three-byte schemes use an extra byte referred to as a *flag value* which is a sign that two more bytes of the RLE packet would follow that encode the symbol and the run length. Thus, sequences of uncompressed data and RLE compressed data can be combined. Pixel level encoding schemes consider multi-byte information that is forming the code of the pixel as the one symbol, and run length is the number of those multi-byte symbols in the run. There is typically one byte used to represent this length value.

RLE variants are known, while encoding images, in which there are scanlines parallel to the x-axis or the y-axis. Zig-zag patterns or 2D tiles are also used. Linearization of data using Morton order [8] is often used in volumetric datasets. Repeat scanline packet or vertical replication packet is an encoding scheme in which the RLE packet contains information on how many times the scanline is consecutively repeated. This uses an image property when several scanlines placed in the image in the sequence do not differ from each other.

Also, there are lossy versions of RLE compression, when different approaches to information discarding, which makes it lossy, are used. For example, changing the overall value of a particular symbol in the stream to make the run homogenous and run-length bigger or changing the Least Significant Bits (LSB) of encoded symbols are used. This approach results in a compression ratio increase when applying RLE.

This paper aims to propose an Out-of-Core algorithm for lossless data compression, which, thanks to the proposed optimization of the bit-level RLE data encoding, along with an alternative application of the delta data transformation, would increase the compression ratio compared to the classical lossless RLE compression, while it still maintains low spatial and temporal complexity. Another goal is to reduce the potential negative effect when unwanted and sometimes significant inflation of data could occur in case of RLE compression of data that is not suitable for compression using this kind of compression algorithm.

Contribution of the paper is in:

- Design of the Out-of-Core lossless compression algorithm using optimized bit-level Run-Length Encoding along with an alternative application of the delta data transformation as well as the design of a binary data structure intended to store such compressed data.
- Optimization of the bit-level run-length encoding by modifying the structure and length of the bit-level RLE packet in which only the length of the run is encoded in its binary representation and the symbol itself is encoded by the RLE packet position in the stream of these RLE packets.
- Optimization of the RLE packet binary representation when it is composed of the variable number of encoding words whose binary representation has a variable length in number of bits.
- Encoding words length optimization separately for each original data block with simultaneous dissociation for encoding word lengths for 0 and 1 symbols.

The structure of the paper is as follows.

Section 2 deals with the related works in the field of lossless data compression using Run-Length Encoding (RLE) paying particular attention to lossless RLE compression schemes and especially the bit-level RLE compression. There are also areas, where the RLE compression has been applied to, mentioned. Because of the broad range of publications in the field, especially when the application of RLE in different fields is taken into account, the section is focused on the papers that are closely related to the work presented in this paper.

Section 3 of the paper presents a detailed description of the data structure proposed as part of this research for the storage of data after ΔRLE data compression. Four different types of data blocks are mentioned and described in detail.

Section 4 of the paper introduces the ΔRLE algorithm for lossless data compression.

Section 5 of the paper summarizes the results of testing within this paper designed ΔRLE lossless compression algorithm using real-life volumetric binarized medical images. Those were obtained by multiple medical imaging methods including Computed Tomography (CT) and Nuclear Magnetic Resonance (NMR), which have been thresholded in the preprocessing phase. The classical bit-level RLE algorithm is compared with different degrees of optimization of the bit-level RLE compression and finally with the full-featured bit-level ΔRLE compression algorithm proposed in this paper with the application of all four types of data blocks, including delta data transformation and optimized RLE data encoding.

Section 6 of the paper presents conclusions, that summarizes the success of the proposed algorithm in data compression. Conclusions were made by testing the algorithm on various volumetric datasets.

2. Related works

Algorithms for data encoding which are incorporating Run-Length Encoding are often part of the data gathering, processing, transmitting, and storing in different kinds of medical applications. In [9] Run-Length Encoding was used in the compression of

data for electrocardiography (ECG) and [10] incorporates compression of ECG data using not only RLE but also Discrete Wavelet Transform (DWT) compression algorithm. The compression of data obtained by Electroencephalography (EEG) combining Discrete Cosine Transform (DCT) and Run-Length Encoding was proposed in [11]. In [12] was performed analysis of hybrid lossy and lossless compression using RLE.

Run-Length Encoding was used for enhancement of user experience in the field of volume datasets reading from secondary storage when bit-level RLE compressed three-dimensional binary images were used as the metadata describing volume datasets [13]. In [14] evaluation of four different encoding schemas for optimization of bit-level Run-Length Encoding within lossless compression of binary images was performed.

Lossless data compression schemes for volume datasets compression, including RLE were considered in [15] and in [16] volume datasets compression using RLE was mentioned. RLE encoded sparse level sets were introduced in [17]. Volumetric datasets characterization using a set of Run-Length Encoding features has been described in [18].

Interesting sources of information regarding data compression in general, that mention also different versions of Run-Length Encoding lossless and lossy compression of text, images - including binary images - and other kinds of data are [19] [20] [21] [22]. Very interesting overview of the compression techniques including lossless compression and RLE represents [23].

3. Description of the proposed ΔRLE data structure

The next chapter of the paper deals with a detailed description of the binary data structure intended for storing the target data compressed using the ΔRLE compression algorithm.

3.1. Data structure

Data structure has its own Global Head (GH) described in section 3.1.1. The original data ready for compression is forming the stream of bits that is divided into blocks with a length of n bits. Since the number of bits of the original data does not necessarily be divisible by n, the last block may be shorter. A Data is thus divided into m original data blocks (ODB). Data block can be left uncompressed as original data block and in that case a Raw Data Block (RawDB) is created - in more detail described in section 3.1.2. A bit-level RLE compression can be applied to the data block creating an RLE Data Block (RleDB) - in more detail described in section 3.1.3. Alternatively, a delta transformation followed by a bit-level RLE compression can be applied, creating a Delta-transformed Data Block (DelDB) - in more detail described in section 3.1.4. If the current block of data is the same as the previous block of data, the Repeating Data Block (RepDB) is used - see section 3.1.5 for details. Information on which of the alternatives was used along with related parameters setup is stored in the header by which each of the m data blocks is extended as shown in Figure 1.

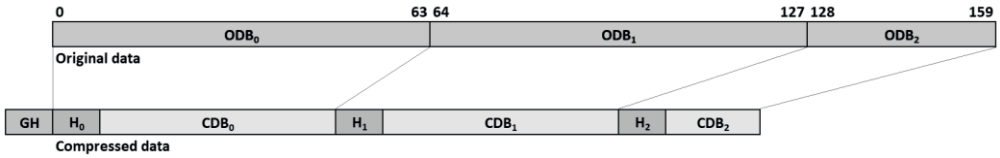

Figure 1. The original data consisting of 160b is divided into blocks with a length of 64b. Two blocks of a standard length 64b were created – named ODB_0 and ODB_1. the last block named ODB_2 is shorter and has a length of 32b. Compressed data consists of three Compressed Data Blocks (CDB) labeled CDB_0 to CDB_2 each with its own header labeled H_0 to H_2. There is a Global Head (GH) of the whole data structure in its introductory part.

3.1.1. Global Head

The Global Head (GH) consists of two encoding packets.

The first one is the GLobal Size Packet (GLSP) which carries the number of bits of the source data to be compressed. GLSP consists of one or more consecutive 32b encoding words (EW). The number of bits of source data can be calculated as the sum of the values that are encoded by the individual EWs in the GLSP packet. GLSP makes it possible to stop the decompression algorithm upon obtaining the last bit of the original data.

The second part of the GH is the Data Block Size Packet (DBSP) which consists of one or more consecutively stored 16b EWs. The DBSP carries information about the number of bits comprised in the Original Data Block (ODB). The value can be calculated as the sum of the values that each EW of the DBSP encodes. This information allows the algorithm to stop decompression of Compressed Data Block after the last bit of the corresponding Original Data Block was obtained.

EW meaning can be seen in Table 1. It is possible to find out if another EW will be concatenated using the value of EW as it is shown in Table 1. For GLSP $max = 2^{32} - 1$ and for DBSP $max = 2^{16} - 1$.

EW	Value	Another EW concatenation
$< 1; max - 1 >$	$< 1; max - 1 >$	NO
max	max	YES
0	max	NO

Table 1. Encoding Word values for GLobal Size Packet and Data Block Size Packet.

3.1.2. Raw Data Block

If it is not possible to achieve a sufficient reduction in the length of the binary representation of the original data block (ODB) consisting of *n* bits (it means the number of bits after compression of ODB is greater or equal to *n* + 2), by using an optimized bit-level RLE compression or by applying a delta transformation and subsequent optimized bit-level RLE compression, then this data block in the target data structure is represented in its original, uncompressed form. Such a block of data

is referred to as Raw Data Block (RawDB) in the target data structure. It consists of a RawDB Head (RawDBH) having a binary representation of '00' and occupying a constant length of 2b and an uncompressed binary representation of ODB consisting of n bits as shown in the Figure 2.

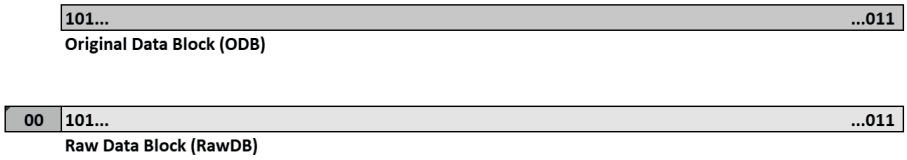

Figure 2. Original Data Block (ODB) with n bit length is transformed into Raw Data Block (RawDB) by creating a RawDBH header consisting of '00' and attaching original data of n bit ODB in unchanged form.

3.1.3. RLE Data Block

The original data can be compressed using in this paper proposed optimized bit-level Run Length Encoding. Rle Data Block (RleDB) has an RLE Data Block Header (RleDBH) followed by an RLE packet stream (RLEPs). Each of these RLE packets has a serial number in the stream (counting starts at 0) so it can be determined for each RLEP whether its serial number is even or odd as can be seen in Figure 3.

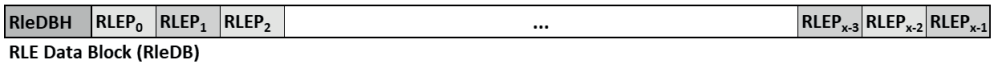

Figure 3. The Rle Data Block (RleDB) consists of the RleDBH header and the RLE packet stream (RLEPs). RLEPs are labeled by a serial number starting from 0. In this figure, there are x RLE packets referred to as $RLEP_0$ to $RLEP_{x-1}$.

3.1.3.1 Rle Data Block Header

The RleDBH header contains a two-bit string '01' identifying that a given block of data is compressed using the bit-level RLE. Then, there is a bit labeled as S indicating by its value, which symbol (0 or 1) run-lengths will encode RLEPs with an even sequence numbers. The odd RLEPs then encode the run-length of the inverted bit value (another symbol).

Follows 0 $Word\ Length$ ($0WL$) represented by 4b. This parameter shows the number of bits used in the EWs in the RLEPs that encode lengths of runs of the symbol 0. The last parameter is 1 $Word\ Length$ ($1WL$) represented by 4b and this parameter shows the number of bits used in the EWs in the RLEPs that encode lengths of runs of the symbol 1 as shown in Figure 4. Allowed values for $0WL$ and $1WL$ are from the range $< 2; 15 >$. The RleDBH header has a constant length of 11b.

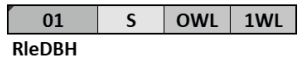

Figure 4. The RleDB Header (RleDBH) consists of an identifier represented by the string '01' followed by a bit indicating which symbols will encode even RLEPs followed by parameters $0WL$ and $1WL$ which on 4b represent the number of bits in the RLEPs encoding words.

3.1.3.2 RLE Packet

An RLE packet is composed of one EW or a sequence of several EWs. Bit length l of each EW is defined by the parameter $0WL$ for the RLEPs for symbol 0 resp. $1WL$ for the RLEPs for the symbols 1. The value of each EW can be determined according to Table 1 with the value $max = 2^l - 1$.

The length of the symbol run encoded in RLEP is given as the sum of the values of the individual EWs concatenated in the given RLEP as shown in Figure 5.

Figure 5. RLEP encodes the number of symbols in the run where each EW consists of 3 bits in this example and a) contains one EW set to 0 indicating that the run-length indicated by this EW_0 is 7 (refer Table 1) and that no other EW will be concatenated, b) the RLE Packet consists of two EWs, the first EW_0 is set to 7 and encodes a run-length of 7 while indicating that one more EW will follow. This EW_1 is set to 3 and therefore the run-length encoded by this EW is 3. At the same time, it indicates that no further EW will follow in a given RLEP. Overall, those two EWs indicate that 10 consecutive symbols will follow in a single run (7+3).

If too low $0WL$ or $1WL$ is selected, then it is necessary to use more concatenated EWs to represent a specific run length which increases the number of bits needed for encoding beyond their optimal number. If too large value of $0WL$ or $1WL$ is selected, then the EWs contain too many leading 0s which again causes the non-optimal number of bits to be used for encoding as shown in the example in Figure 6. Therefore, it is advisable to determine, based on the analysis of the particular data, the optimal length of EW which allows the use of a minimum number of bits concerning the selected encoding. Since for each RLEP it is possible to determine whether it encodes the run of symbols 0 or 1, it is possible to search for the optimal length separately for EWs encoding run-lengths of symbol 0 and separately for symbol 1. Since the data of the source file is divided into data blocks such an optimum can be found for each data block separately. This procedure of finding and using the optimal EW length for each data block, dissociated for EWs for symbols 0 and 1, is the basis of the bit-level RLE optimization proposed in this work.

Figure 6. Run-length encoding with the number of symbols 14 while selecting different lengths of EWs where a) the EW length set to 3 requires the concatenation of two EWs which means the use of overall 6b b) the use of an EW with a length of 4b allows optimal encoding and c) using an EW with the length of 5b means unnecessarily long bit sequence with one leading 0.

3.1.4. Delta-transformed Data Block

To achieve a higher compression ratio, it is possible to perform a delta transformation of the original data when differential information is generated between the original data of the Previous Data Block (PDB) and the currently compressed original data block, called Actual Data Block (ADB) by applying the XOR function to the respective pairs of bits.

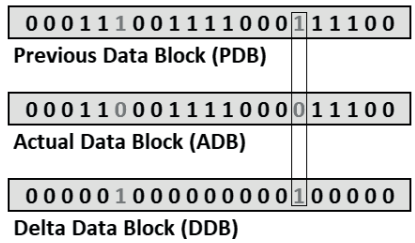

Figure 7. The Delta Data Block (DDB) is derived from the Previous Data Block (PDB) and the Actual Data Block (ADB) using the XOR function on the respective bit pairs. DDB can yield a higher compression ratio over the compression of the original ADB data.

If those two bits match, the result is symbol 0, if they do not match, the result is symbol 1. This creates a Delta Data Block (DDB) and in some cases this information can be better compressed using optimized bit-level RLE than the original ADB data as it shown in Figure 7.

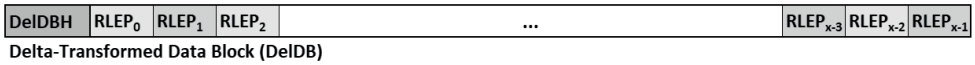

Figure 8. The Delta-transformed Data Block (DelDB) consists of a DelDBH header and a stream of RLE packets that are identified by a serial number starting from 0. There are x RLE packets visible in the figure labeled as $RLEP_0$ to $RLEP_{x-1}$.

DDB is then compressed using an optimized bit-level RLE as described in section 3.1.3 and that is how Delta-transformed Data Block (DelDB) is created with the structure that can be seen in Figure 8.

DelDB header (DelDBH) contains a two-bit string '10', symbol S and 4-bit $0WL$ and $1WL$ with the same meaning as it was described in the subchapter 3.1.3.1. The

DelDBH header has a constant length of 11b as it can be seen in Figure 9 and is followed by a stream of RLEPs with a structure analogous to that described in section 3.1.3.2.

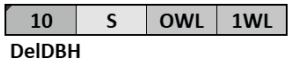

Figure 9. The Delta-transformed Data Block Header (DelDBH) consists of an identifier that is represented by a '10' string followed by a symbol S and 4-bit $0WL$ and $1WL$.

3.1.5. Repeating Data Block

If a particular data block (encoded with Raw Data Block, Rle Data Block, or Delta-transformed Data Block) is followed by several other data blocks, containing the same information, then it is possible to represent repeating blocks by Repeating Data Block (RepDB). This block is represented the RepDBH header whose two bits are constantly set to '11'. This identifier is followed by one or more 3-bit EWs that encode the number of consecutive data block repetitions. The value of each EW can be determined using Table I when $max = 7$.

If multiple EWs are concatenated, the number of repeating data blocks is determined as the sum of their repeating data block counts as shown in the example in Figure 10. Thus, for example if 6 same original data blocks are consecutive then the first one is represented as either a Raw Data Block, an RLE Data Block, or a Delta-transformed Data Block followed by a RepDB that indicates its 5 repetitions.

The total length of the $RepDB_{len}$ can be calculated by the formula:

$$RepDB_{len} = 2 + \left\lceil \frac{h}{7} \right\rceil \times 3[b] \tag{1}$$

where h is the number of repetitions:

Figure 10. RepDB Header (RepDBH) contains an identifier set to '11' and there are a) one EW that encodes 7 repetitions of the data block, and the EW indicates that no other EW will be concatenated b) two EWs where the first one encodes 7 repetitions of the data block and at the same time indicates the concatenation of the next EW which encodes 6 further repetitions while indicating that no further encoding word will be concatenated. This means a total of 13 repetitions of the previous data block.

4. ΔRLE compression algorithm

Compression algorithm assumes input that consists of:

- Stream of original data intended for compression,
- parameter m representing overall length of input stream in bits,
- parameter n representing length of Original Data Block (ODB) in bits.

Compression algorithm comprises following steps:

1. **PDB**(n - 1 : 0) = **0**; **REPS** = **0**; // PDB – Previous Data Block
2. LOAD **ADB**(n - 1 : 0); // ADB – Actual Data Block
3. IF (**ADB**(n – 1 : 0) = = **PDB**(n – 1 : 0)**)** { **REPS++;** GOTO 2; **}**
4. IF (**REPS** > 0) {CREATE **RepDB(REPS)**; STORE **RepDB**; **REPS** = 0;}
   // RepDB – Repeating Data Block
5. **min** = **n** + **2**;
6. **[minRleDB0, Rle0WL]** = findMin(**ADB**(n – 1 : 0), **0**, **2**, **15**);
7. **[minRleDB1, Rle1WL]** = findMin(**ADB**(n – 1 : 0), **1**, **2**, **15**);
8. **minRleDB** = **minRleDB0** + **minRLEDB1**;
9. **DDB**(n – 1 : 0) = **PDB**(n - 1 : 0) XOR **ADB**(n – 1 : 0); // DDB – Delta Data Block
10. **[minDelDB0, Del0WL]** = findMin(**DDB**(n – 1 : 0), **0**, **2**, **15**);
11. **[minDelDB1, Del1WL] =** findMin(**DDB**(n – 1 : 0), **1**, **2**, **15**);
12. **minDelDB** = **minDelDB0** + **minDelDB1**;
13. IF (**minRleDB** < **min** AND **minRleDB** ≤ **minDelDB**) {
    CREATE **RleDB(ADB**(n - 1 : 0)**, Rle0WL, Rle1WL)**; STORE **RleDB;** GOTO 16; }
    // RleDB – RLE Data Block
14. IF (**minDelDB** < **min** AND **minDelDB** < **minRleDB**) {
    CREATE **DelDB(DDB**(n - 1 : 0)**, Del0WL, Del1WL);** STORE **DelDB;** GOTO 16; }
    // DelDB – Delta-transformed Data Block
15. CREATE **RawDB**; STORE **RawDB**; // RawDB – Raw Data Block
16. IF (**m** = = 0) EXIT;
17. **PDB**(n – 1 : 0) = **ADB**(n – 1 : 0);
18. IF (**m** > **n**) { **m** = **m** – **n**; }
    ELSE { **n = m**; **m** = **0**; }
19. GOTO 2;

Function [a, b] = findMin(D(n – 1 : 0), S, L, H) is the function which:

- finds the minimal sum of bits of all RLE packets for symbol S and data block D(n - 1 : 0) when all encoding word lengths from the range <L; H> are tested and returns that minimal length as the output a,
- finds the optimal EW length for symbol S for data block D(n-1 : 0) and returns it as the output b.

5. Tests and discussion

One of the objectives of the algorithm design was to reduce the risk of significant inflation of data after compression in an unfavorable case. This goal has been met. In situations, when none of the original data blocks can be effectively compressed, each of these blocks is represented by RawDB in the target data structure. Each original data block is in that case accompanied by a two-bit header. Assuming that the original data is divided into *m* blocks and for simplicity, each of these blocks has the same number of *n* bits (including the last data block) then the size of the binary representation after the ΔRLE compression is quantifiable as $SIZE_{max}$:

$$SIZE_{max} = 32 \times \left\lceil \frac{s}{2^{32}-1} \right\rceil + 16 \times \left\lceil \frac{n}{2^{16}-1} \right\rceil + m \times (n+2)[b] \quad (2)$$

Where:

s is the number of bits of the source data

n is the number of bits in the original data block

m is the number of original data blocks

$SIZE_{max}$ is at the same time the largest number of bits that can be generated as the target data structure from the respective source data.

Maximum inflation INF_{max} for individual original data blocks is then quantifiable as the ratio of the Raw Data Block size $(n+2)$ to the corresponding size of Original Data Block (n) so it can be written as:

$$INF_{max} = \frac{n+2}{n} \quad (3)$$

With increasing value n the INF_{max} is approaching 1 as it is shown in the following statement:

$$INF_{max} = \lim_{n\to\infty} \frac{n+2}{n} = 1 \quad (4)$$

This implies that the larger the selected original data blocks are, the smaller the potential data inflation after compression is. If $n = 256$ then $INF_{max} = 1{,}00781$ while when $n = 1024$, the $INF_{max} = 1{,}00195$.

The most favorable (theoretical) case is when the compressed data is made up of the run of only one symbol 0. For compression then it can be advantageous that the Previous Data Block (PDB) buffer is at the start of compression homogenously filled with the symbol 0. The first Original Data Block of the source data is also homogeneously filled with the symbol 0 and therefore the Repeating Data Block (RepDB) can be used immediately. If it is possible to split data into *m* data blocks of *n* bits, then this RepDB will include all original data blocks of the source data.

To minimize the total number of bits used in the target data structure $SIZE_{min}$, it is important that the value of *m* is as low as possible, which means the number of the original data blocks is as low as possible. This is achieved by maximizing the value of *n* which is the number of bits in the original data block. Then $SIZE_{min}$ can be calculated as

$$SIZE_{min} = 32 \times \left\lceil \frac{s}{2^{32}-1} \right\rceil + 16 \times \left\lceil \frac{n}{2^{16}-1} \right\rceil + 2 + 3 \times \left\lceil \frac{m}{7} \right\rceil [b] \quad (5)$$

where

s is the number of bits of the source data

n is the number of bits in the original data block

m is the number of original data blocks

Thus, the lowest number of bits after data compression can be 53, if $m \leq 7$, $n \leq 65\,535$ and $s \leq 458\,745$.

5.1. Datasets

In the tests of the proposed ΔRLE compression algorithm, there were used volumetric datasets consisting of regular three-dimensional grids of scalar values. In the range of $< 0; 255 >$ values were saved to 8b and in the range of $< 0; 4095 >$ to 16b. Datasets had been obtained through medical imaging techniques such as Computed Tomography (CT) or Magnetic Resonance Imaging (MRI). Above these volumetric data a thresholding was performed whose output was the regular three-dimensional grid of scalar values with 1b/vox. The data was then linearized and the stream of bits obtained was subjected to compression via the classical RLE algorithm, RLE algorithm with properties that are part of the proposed ΔRLE algorithm, and finally a full-featured proposed ΔRLE algorithm. Parameters of examples of individual volumetric datasets used in the tests are summarized in Table 2 and visualizations of two-dimensional sections of these volumetric datasets can be seen in Figure 10.

	Volume dataset	Volume dimensions	Voxels [Millions]	b/vox	Size [MB]	Threshold
a	Head	256 × 256 × 113	7.40	16	14.13	200
b	Brain	256 × 256 × 109	7.14	16	13.63	1150
c	Visual Cortex	1024 × 1024 × 314	329.25	8	314.00	25
d	Abdomen	512 × 512 × 174	45.61	16	87.00	200
e	Human Skull	256 × 256 × 256	16.78	8	16.00	25
f	Pancreas	240 × 512 × 512	62.91	16	120.00	225

Table 2. Characteristics of medical imaging volume datasets that were binarized by using thresholding and used for the test purpose.

5.2. Test Results

In the first of the tests, a classical RLE algorithm was applied to the bit stream of the original uncompressed data which uses the 8b RLE packet for encoding in which one bit is used to encode the respective symbol and the other 7 bits are used to encode run-length. In Table 3 this encoding is referred to as RLE-I.

In the second test, the RLE packet described in section 3.1.3.2 was used. The encoding word length of this packet was constantly set to 8b. In Table 3, this encoding is referred to as RLE-II. As shown in the test results the change in the RLE packet encoding resulted in an increase in the compression ratio of 1.058 to 1.819 times in comparison to the RLE-I encoding.

In the third test, the RLE packet described in section 3.1.3.2 was used with the length of the binary representation of the encoding word of this packet being optimized ranging between $< 2; 15 >$ bits. In Table 3 this encoding is

referred to as RLE-III. For each dataset, there is also given the optimal EW length.

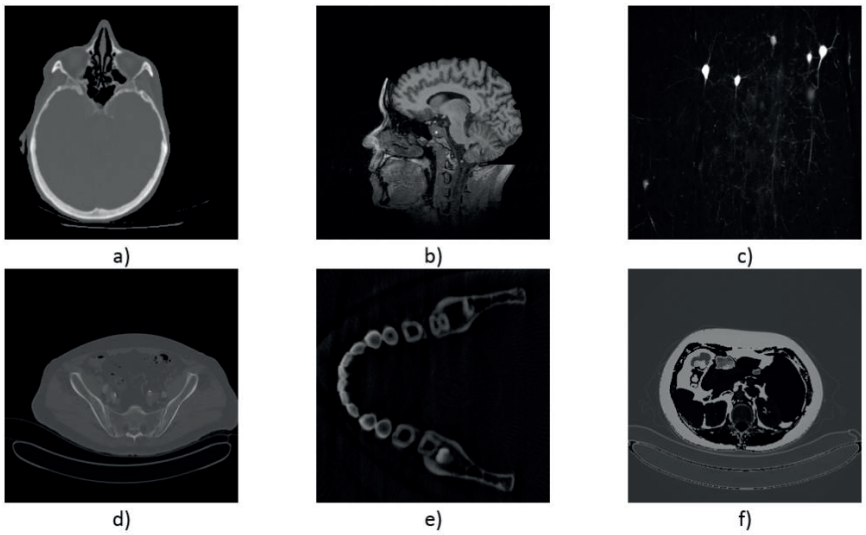

Figure 10. Slices of volume datasets obtained by different medical imaging methods that were used in tests of designed ΔRLE lossless compression algorithm.

As shown in Figure 11 for the dataset a - Head, the optimal length of the encoding packet was set to 7b which allowed a further increase in the compression ratio compared to RLE-I 1.143 times and compared to RLE-II 1.040 times. For all the datasets mentioned in this paper this change in encoding to RLE-III produced from 1.143 to 4.290 times higher compression ratio over RLE-I. Encoding word lengths from 6b to 12b have been shown to be optimal for the individual datasets.

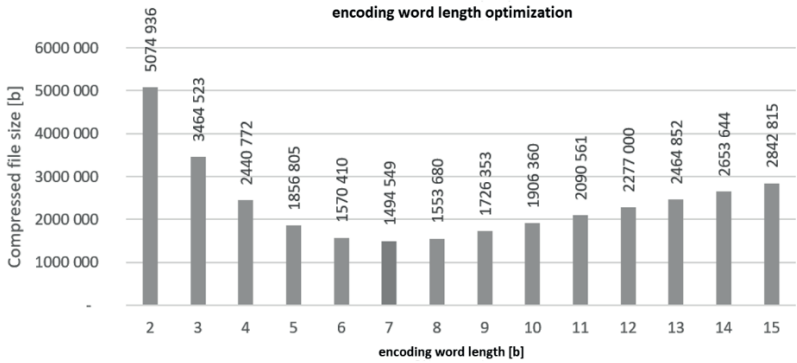

Figure 11. Optimization of encoding word length which is performed by the function findMin in compression algorithm (see chap. 4) allows to choose optimal encoding word length from the range of $< 2; 15 >$. For a particular dataset the optimal encoding word length is 7, when data after compression represent the smallest file size (1 494 549 b).

The fourth test criteria were pretty much the same as in the previous test, the only change was made that for the symbol 0 and 1 the EW length of RLE packets were dissociated. In Table 3 this encoding is referred to as RLE-IV. As shown by the results of the tests, this change in encoding made a 1.150 to 7.030 times increase in compression ratio over RLE-I encoding. The EW lengths for the symbol 0 were from the range of $< 2; 8 >$ b for different datasets and for the symbol 1 was in the range of $< 3; 13 >$ b. The largest difference in the EW lengths was shown in the dataset f - pancreas where for the symbol 0 the EW length was 2b and for the symbol 1 it was 13b. How the change in the EW length in the range of $< 2; 15 >$ b separately for the symbols 0 and 1 in case of the dataset a – Head influenced the length of compressed data is shown in Figure 12. For this dataset, the optimum EW length for the symbol 0 was set to 7b and for the symbol 1 to 6b.

The last of the tests applied all the properties of the algorithm proposed in this work when the data was divided into blocks with a length equal to the number of voxels on the edge of the respective dataset parallel to the x-axis. At the same time, encoding word length optimization was applied separately for the symbols 0 and 1 and all four types of data blocks were used i.e. RawDB, RleDB, DelDB, and RepDB. This encoding is referred to as ΔRLE in the results table. Compared to the original data (thresholded), a compression ratio of 3.972 to 48.911 was achieved and compared to the classical algorithm (RLE-I), compression rates of 1.24 to 3.44 times higher were achieved. The comparison of the compression ratio of classical RLE algorithm (referred to as RLE-I) and ΔRLE algorithm, proposed in this work, can be seen in Figure 13.

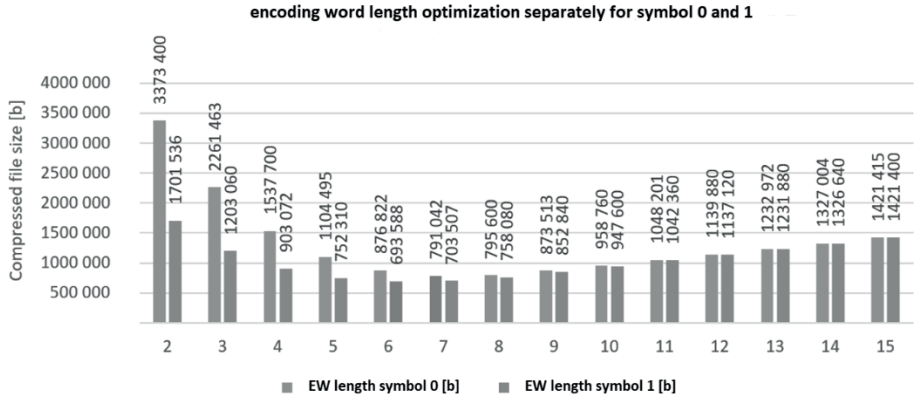

Figure 12. Optimization of EW length dissociated for symbol 0 and 1 allows to choose the optimal length of EW from the range of $< 2; 15 >$ b, for symbol 0 and for symbol 1, separately. As it can be seen from the graph, for particular dataset the optimal length of EW for symbol 0 is 7b and for symbol 1 is 6b. In that case the data after compression represent the smallest file size (693 588 b + 791 042 b = 1 484 630 b).

	Dataset					
	a	b	c	d	e	f
Original data	7 405 568	7 143 424	329 252 864	45 613 056	16 777 216	62 914 560
RLE-I	1 708 056	2 074 792	113 812 768	6 200 848	5 393 216	4 418 528
	4.336[*1]	3.443	2.893	7.356	3.111	14.239
	23.06[*2]	29.04	34.57	13.59	32.15	7.02
RLE-II	1 553 680	1 913 272	107 207 224	4 906 792	5 096 336	2 429 064
	4,766	3,734	3,071	9,296	3,292	25,901
	20.98	26.78	32.56	10.76	30.38	3.86
RLE-III	1 494 549	1 815 443	96 845 382	4 906 792	4 628 934	1 030 032
	4.955	3.935	3.400	9.296	3.624	61.080
	20.18	25.41	29.41	10.76	27.59	1.64
WL [b]	7	7	6	8	6	12
RLE-IV	1 484 630	1 749 466	78 515 216	4 902 737	3 892 132	628 533
	4.988	4.083	4.193	9.304	4.311	100.097
	20.05	24.49	23.85	10.75	23.20	1.00
0WL / 1WL [b]	7 / 6	7 / 5	7 / 3	8 / 9	7 / 3	2 / 13
ΔRLE	1 376 192	1 563 792	68 217 184	4 452 256	4 223 248	1 286 312
	5.381	4.568	4.827	10.232	3.972	48.911
	18.58	21.89	20.72	9.76	25.17	2.04

Table 3. Size of original data and size of data after compression for individual compression algorithms and their settings for particular datasets. Compression ratios ([*1]) and the percentage of data size after compression as compared to the original data ([*2]) are shown, too.

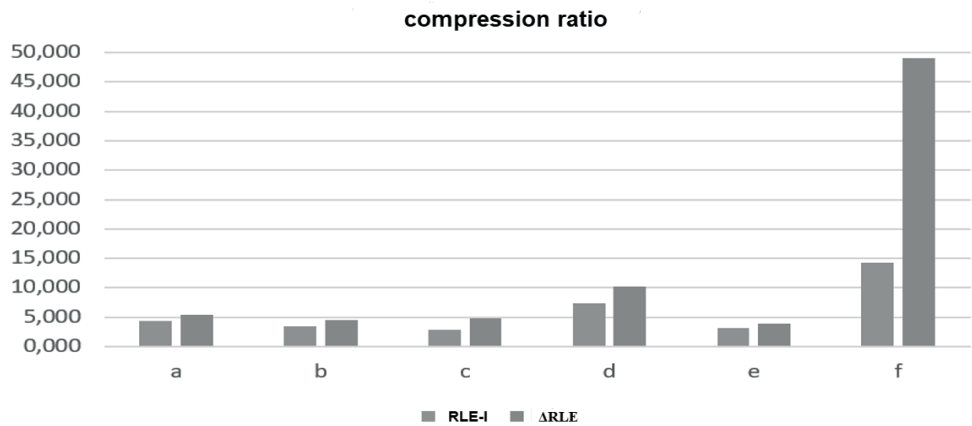

Figure 13. Compression ratio of classical RLE algorithm (referred to as RLE-I) and ΔRLE algorithm for individual datasets.

Table 4 and Figure 14 show the number of individual data block types and their percentage for each dataset. It was not necessary to use RawDB for compression.

	Dataset					
	a	b	c	d	e	f
RawDB	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%	0 0.00%
RleDB	11 428 39.50%	13 221 47.38%	187 766 58.40%	48 024 53,91%	38 581 58.61%	41 775 15.93%
DelDB	11 897 41.13%	7 646 27.40%	41 912 13.03%	13 461 15.11%	24 540 37.70%	124 0.05%
RepDB	5603 19.37	7037 25.22%	91 858 28.57%	27 603 30.98%	2 415 3.69%	220245 84.02%

Table 4. The number of individual types of data blocks in the target data structure along with their percentage representation.

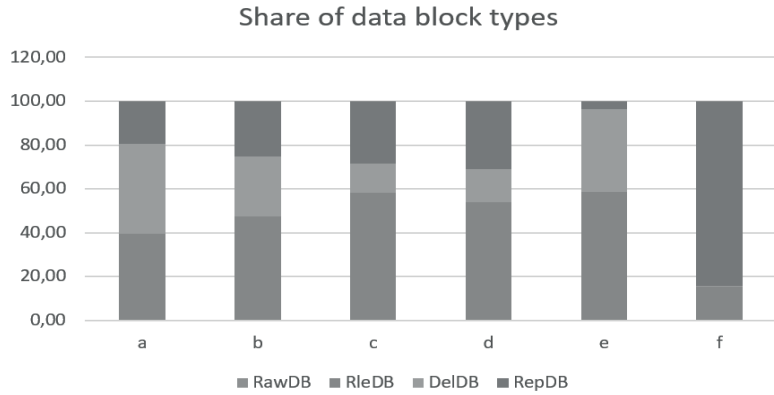

Figure 14. Graphical representation of data block types in ΔRLE compressed data for individual datasets.

Compression		Dataset					
		a	b	C	d	e	f
RLE	Time [s]	0.017	0.016	0.740	0.099	0.038	0.134
	Data throughput [MB]	52.73	52.89	53.02	55.00	52.80	55.81
ΔRLE	Time [s]	0.051	0.051	2.071	0.185	0.133	0.058
	Data throughput [MB]	17.25	16.68	18.95	29.38	15.08	129.46
	Ratio	3.06	3.17	2.80	1.87	3.50	0.43

Table 5. Compression time and data throughput for RLE and ΔRLE algorithms for various volume datasets. Data throughput is calculated in relation to uncompressed data amount.
(*1) Ratio represents time of DRLE compression / time of RLE compression.

Execution time measurement was performed using program implementation in C language, applying the same programming approaches where it was applicable, in

compression and decompression programs for RLE and ΔRLE. Tests were performed on the computer with Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz, 8GB RAM, Debian Linux version 4.19.0-6, gcc version 8.3.0. To eliminate influence of the data transfer from secondary storage, that can be of different technology and data transfer speed, the time measurement started when all input data were stored in operating memory and stopped when output data were also stored in the operating memory. Due the binary representation of the ΔRLE compressed data, that is not byte aligned, and time consuming analysis of optimal encoding word lengths, compression time of ΔRLE was 3.50 times longer in comparison to RLE in the worst case. However, effective work with repeating data blocks, especially when there is higher number of those blocks consecutively in ΔRLE, allowed ΔRLE compression to use 0.43 of the time in comparison to RLE in most optimistic case, as it is shown in the Table 5. Same reasons allowed slower decompression time of ΔRLE ranging from 2.66 times in the worst case to 1.73 times in the best case, in comparison to RLE, as it is shown in the Table 6.

Decompression		Dataset					
		a	b	c	d	e	f
RLE	Time [s]	0.007	0.007	0.321	0.037	0.017	0.045
	Data throughput [MB]	124.66	124.04	122.34	146.96	114.67	167.41
ΔRLE	Time [s]	0.018	0.018	0.772	0.084	0.046	0.077
	Data throughput [MB]	49.45	47.22	50.83	64.65	43.17	96.87
	Ratio*[1]	2.52	2.63	2.41	2.27	2.66	1.73

Table 6. Decompression time and data throughput for RLE and ΔRLE algorithms for various volume datasets. Data throughput is calculated in relation to uncompressed data amount.
(*1) Ratio represents time of DRLE decompression / time of RLE decompression.

Memory consumption for variables storing includes the use of 64b variable for storing the information on the size of uncompressed data, for both RLE and ΔRLE and another 32b for storing of the data block size in case of ΔRLE. Overall size of the space for storing variables is 12B for both RLE compressor and decompressor and 57B for ΔRLE compressor and 33B for ΔRLE decompressor. ΔRLE compressor and decompressor include also $2 \times n$ bits space for the storage of two data blocks of n-bits size and 32B Look-Up-Table. ΔRLE also uses another 64B data structure for encoding words length optimization.

6. Conclusions

The paper deals with the problematics of data compression and specifically lossless data compression as part of the research of the domain-specific data structures dedicated to volume data representation.

There is an introduction of the Run-Length Encoding compression principle and several modifications of RLE, including bit-level RLE, byte-level RLE, and pixel-

level RLE with scan-line option were mentioned in the first part of the paper. Related works in the field of RLE compression principle and their use in different application areas including medical applications are mentioned in the related works part of the paper.

In the second part of the paper enhanced bit-level ΔRLE algorithm is introduced, which optimizes the RLE packet and brings multi encoding word RLE packets with various number of encoding words and with optimized and variable encoding word lengths. Dissociation of RLEPs for symbol 0 and symbol 1 are part of this modification as well. ΔRLE algorithm includes the possibility to use four different data blocks - raw data block, RLE encoded data block, delta transformed RLE encoded data block, which combines processing of the data block into the form of differential information, which is then encoded similarly as the RLE encoded data block, and last of them is repeating data block, which indicates that the data block is identical with the previous data block. This allows not only to optimize RLE encoding but also to choose the best possibility of how to encode each data block. This results in a better compression ratio.

In the last part of the paper there are results of tests described, in which volumetric datasets that were obtained by medical imaging methods and then thresholded were compressed, using the classical RLE method, alternatively using to the different levels optimized RLE methods and finally using ΔRLE algorithm introduced in this paper. The results show that in all volume dataset cases there was a possibility to compress data with the compression ratio higher than 1. In comparison to the classical RLE compression algorithm, the compression ratio was higher in all cases, when 1.24 to 3.43 times higher compression ratio in comparison to the classical RLE was obtained.

There is a possibility to get potentially even higher compression ratio optimizing the length of data blocks according to a particular dataset when each data block has the same size (except last data block) or optimizing the length of each data block separately. This optimization was not included in this research and proposed data structure and algorithm of its construction.

Acknowledgments

This work was supported by KEGA Agency of the Ministry of Education, Science, Research and Sport of the Slovak Republic under Grant No. 003TUKE-4/2017 Implementation of Modern Methods and Education Forms in the Area of Security of Information and Communication Technologies towards Requirements of Labor Market and under Grant No. 077TUKE-4/2015 Promoting the interconnection of Computer and Software Engineering using the KPIkit.

References

[1] A. Pekár, M. Chovanec, L. Vokorokos, E. Chovancová, P. Feciľák and M. Michalko, “Adaptive Aggregation of Flow Records“, Computing and Informatics, vol. 37, no. 1, pp. 142-164, Jan. 2018.

[2] W. Steingartner, V. Novitzká and W. Schreiner, "Coalgebraic Operational Semantics for an Imperative Language", Computing and Informatics, vol. 38, no. 5, pp. 1181–1209, Sep. 2019.

[3] S. Laine and T. Karras, "Efficient sparse voxel octrees", In Proc. of ACM SIGGRAPH 2010 Symposium on Interactive 3D Graphics and Games, 2010, pp. 55-63.

[4] V. Kämpe, E. Sintorn and U. Assarsson, "High Resolution Sparse Voxel DAGs", ACM Transactions on Graphics, vol. 32, no. 4, pp. 8, July 2013.

[5] A. J. Villanueva, F. Marton and E. Gobbetti, "SSVDAGs: Symmetry-aware Sparse Voxel DAGs", In Proc. of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '16), 2016, pp. 7-14.

[6] J. Baert, A. Lagae and P. Dutré, "Out-of-core Construction of Sparse Voxel Octrees", In Proc. of the 5th High-Performance Graphics Conference, 2013, pp. 27-32.

[7] A. H. Robinson and C. Cherry, "Results of a prototype television bandwidth compression scheme", In Proc. of the IEEE, Institute of Electrical and Electronics Engineers (IEEE), 1967, pp. 356–364.

[8] G. M. Morton, "A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing", *Research Report*. International Business Machines Corporation (IBM), Ottawa, Canada, pp. 20, March 1966.

[9] S. Akhter and M.A. Haque, "ECG compression using run length encoding", In 18th European Signal Processing Conference, 2010, pp. 1645-1649

[10] S. A. Chouakri, M.M. Benaiad and A. Taleb-Ahmed, "Run length encoding and wavelet transform based ECG compression algorithm for transmission via IEEE802.11b WLAN channel", In Proc. of the 4th International Symposium on Applied Sciences in Biomedical and Communication Technologies (ISABEL '11), 2011, pp. 1-4.

[11] M. Alsenwi, M. Saeed, T. Ismail, H. Mostafa and S. Gabran, "Hybrid compression technique with data segmentation for electroencephalography data", In 2017 29th International Conference on Microelectronics (ICM), 2017, pp. 1-4.

[12] M. Alsenwi, T. Ismail and H. Mostafa, "Performance analysis of Hybrid Lossy/Lossless Compression Techniques for EEG data", In 2016 28th International Conference on Microelectronics, 2016, pp. 1-4.

[13] B. Madoš, A. Baláž, N. Ádám, J. Hurtuk and Z. Bilanová, "Algorithm Design for User Experience Enhancement of Volume Dataset Reading from Storage Using 3D Binary Image as the Metadata", In SAMI 2019 -

IEEE 17th World Symposium on Applied Machine Intelligence and Informatics, 2019, pp. 269-274.

[14] B. Madoš and N. Ádám, "Evaluation of Different Encoding Schemas for Optimization of Run-Length Encoding within Lossless Compression of Binary Images", In 23rd International Conference on Intelligent Engineering Systems (INES), 2019, pp. 75-80.

[15] P. Komma, J. Fischer and F. Duffner, "Lossless Volume Data Compression Schemes", In SimVIS Simulation und Visualisierung 2007, 2007, pp. 169-182.

[16] B. Curless and M. Levoy, "A volumetric method for building complex models from range images", In Proc. of the 23rd annual conference on Computer graphics and interactive techniques (SIGGRAPH '96), 1996, pp. 303-312.

[17] B. Houston, M. Wiebe and Ch. Batty, "RLE sparse level sets", In ACM SIGGRAPH 2004 Sketches (SIGGRAPH '04), Ronen Barzel (Ed.). ACM, New York, NY, USA, 2004, pp. 137.

[18] D. Xu, S.A. Kurani, J.D. Furst and D.S. Raicu, "Run-length encoding for volumetric texture", In The 4th IASTED International Conference on Visualization, Imaging, and Image Processing, 2004, pp. 452-458.

[19] G. Held, *Data Compression: Techniques and Applications*, *Hardware and Software Considerations*, second edition, John Wiley & Sons, NY, 1987.

[20] D. Salomon, *Data compression*, *The Complete Reference,* fourth edition, Springer Science + Business Media, LLC, 2007.

[21] T. D. Lynch, *Data Compression Techniques and Applications*, first edition, Lifetime Learning Publications, Belmont, CA, USA, 1985.

[22] J. A. Storer, *Data Compression: Methods and Theory*, Computer Science Press, Rockville, MD, USA, 1988.

[23] K. Sayood, *Introduction to data compression – fourth edition*, Morgan Kaufmann, Elsevier, 2012.