Error resilient lossless compression based on binary RLE

Charles Guyon, Bernard Besserer
Laboratoire MIA (Mathématiques, Image et Applications) - Université La Rochelle
Av. Michel Crépeau, 17042 LA ROCHELLE cedex - FRANCE
bernard.besserer@univ-lr.fr

Abstract

The billion pictures shot on consumer cameras or computer generated are mostly stored in compressed data format, often vulnerable even to a single bit change. Obviously, the file format used for storage is an issue, especially at the end-of-life of the storage medium, when hardware become fragile and migration is necessary, increasing the probability of being exposed to bit errors while reading, transmitting and re-writing data.

Error resiliency was a guideline for the JPEG2000 standard [9] and [5], besides this, some formats have already been designed for lossless compression (PNG, JPEG-LS, JPEG2000, BMF, TMW, LOCO-I, CALIC, ...). However only few academic works target efficient lossless compression combined with error resiliency. This early approach points only luminance image, and aim to reduce local redundancies to approaching Shannon's entropy. First, an adaptative predictor based on a causal neighborhood is used to estimate the value of the current pixel, and the error signal is subsequently processed, up to run-length encoding of bitplanes using bit-patterns to store the length of the runs. At many stages of the processing, the impact of a possible bit error is minimized and after run-length encoding, the data is protected with an EDC/ECC (Error Detection Code / Error Correction Code) using $\log_2(N)$ metadata for that usage. Last, a bit shuffle (exactly a Bit-Reversal Permutation[6]) is applied for protection against burst errors. Results will be shown and compared to JPEG2000 images.

Introduction

Indeed, as concerned by image files, dealing with uncompressed data guarantees high resilience to errors. But we estimate that combining lossless compression at high ratio with EDC and ECC will allow us to behave like CD-DA (Compact Disc - Digital Audio) strategy against error: correct as much as you can but if error still remains, minimize the audio/visual impact (*ie.* in image case, perform data interpolation or inpainting).

Data preservation has been identified as a looming crisis of upcoming dramatic proportion. As research libraries and archives are discovering, "born-digital" materials those initially created in electronic form are much more complicated and costly to preserve than anticipated. Especially, due to the incredibly fast growing market for digital camera and camera-phones, the preservation of digital photography has become a major challenge for society. Today, of the world's digital data, approximately 90% reside on mass storage (well designed for access, but not for long-term preservation) or removable media technology such as magnetic tape and optical disks, facing physical deterioration of medium and hardware obsolescence. Only continuous migration to newest storage medium and multiple copies can help to preserve the data. While professional photographers kept their images as uncompressed "raw" data files, highly resilient to errors, the billion pictures shot on consumer cameras are unfortunately stored as JPEG files, a file format that use high compression rates and which is very vulnerable even to a single bit change.

In fact, reliability of high density storage medium has always been the weak link in data storage. Clever EDC & ECC software algorithms and hardware redundancy (RAID) has been developed from the 80's, and are used at low-level in nearby any storage solution. Look back to the introduction of the CD-DA (Compact Disc - Digital Audio) in 1984, Philips contributed the Eight-to-Fourteen Modulation (EFM) which offers both a long playing time and a high resilience against disc handling damage such as scratches and fingerprints, while Sony contributed the error-correction method, Cross-Interleaved Reed-Solomon Code (CIRC). While robust even against long error burst, an audio sample can be corrupted, but since the data is store uncompressed, well designed CD-Players are able to discard outliers and interpolate the missing or corrupted sample from the neighborhood.

This paper targets image storage principles combining lossless compression and error resilience. It is clear that removing redundancies through compression increases the impact of corruption. Important point : the file must stay readable (and error-free openable) even under high error rates. The proposed scheme is well designed for any kind of multidimensional data ($\mathbb{Z}^n \to \mathbb{Z}$), provided that causal neighborhood and scan path is defined. Since we suggest a straighforward framework for compressing data from $\mathbb{Z}^n \to \mathbb{Z}^m$, by performing independently on each dimension of the codomain $\mathbb{Z}^m$ (involving those channels are decorrelated first).

Overview and basic principle

Almost any lossless compression algorithms uses two stages [3], [4] and [7] : The first one removes as much as possible redundancies applying a reversible transforms to the original data, minimizing the variance of the statistic distribution of the new data set (eg. M+S for audio, luma+chroma for color signals, ...). The data size is not yet reduced, but this transform increases the imbalance of the binary symbols: usually, the amount of zeros become predominant over ones.
The second compression stage then performs a statistic coding taking advantage of this kind of distribution.

In practice, the image is divided into data blocks. Each data block is compressed independently (thus allowing multithreading or parallel processing), and each data block is scanned by a given path (e.g. scanline). Local redundancy is removed by computing a error signal which is the difference between a predicted value (from the neighborhood) and the actual data. A RLE (Run-Lenght

Encoding) in bitplane order is then applied on this error data. This workflow sounds familiar, but:

- We really focused to reach the limit at each step: optimal calculation of the predictor weights, using variances classes, data shuffling and adaptive word length for encoding the runs for the RLE, ...
- At each step, we check the behaviour of our algorithm to minimize the impact of possible bit corruption. For example, data encoding was chosen / adapted to create small shifts by corruption.

After RLE encoding, we compute and store EDC/ECC data before applying a Bit-Reversal on-place Permutation on the compressed data, ensuring protection against error bursts.

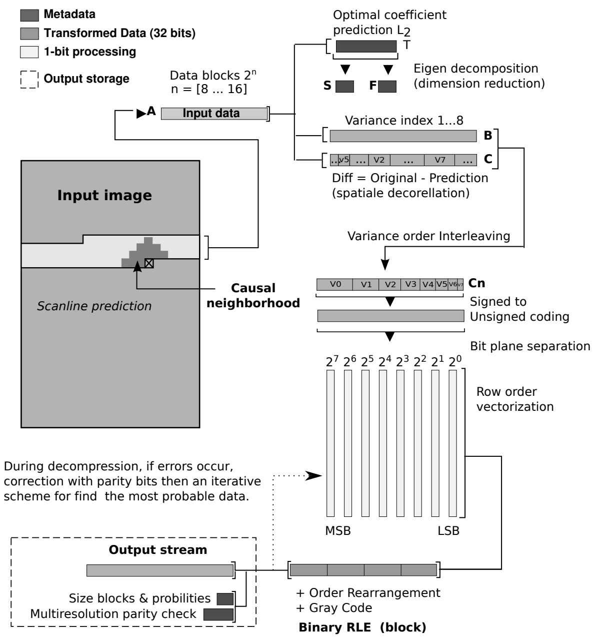

Metadata
Transformed Data (32 bits)
1-bit processing
Output storage
Data blocks 2^n
n = [8 ... 16]
Optimal coefficient prediction L_2
T
Eigen decomposition (dimension reduction)
S
F
A
Input data
Variance index 1...8
B
C
Diff = Original - Prediction (spatiale decorellation)
Input image
Scanline prediction
Causal neighborhood
Variance order Interleaving
V0 V1 V2 V3 V4 V5 V6 V7
Cn
Signed to Unsigned coding
Bit plane separation
2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0
Row order vectorization
MSB
LSB
During decompression, if errors occur, correction with parity bits then an iterative scheme for find the most probable data.
Output stream
Size blocks & probilities
Multiresolution parity check
+ Order Rearrangement
+ Gray Code
Binary RLE (block)

Figure 1. Complete compression data flow. Each step will be discussed below.

Causal prediction and redundancy reduction

Three types of redundancies are commonly assumed given digital images:

- **Local redundancies**: The central pixel is correlated to his neighborhood.
- **Frequential redundancies**: These more global approaches often estimate a small number of periodic basis function; the weighted sum of them provide a good approximation of the signal (Fourier, DCT, Wavelets, ...).
- **Pattern redundancies**: Similar patches can be found at distant location within the signal, may be at a different scale. Fractal compression and dictionary based compression take advantage of such redundancy, and such compression schemes work well on data with periodic patterns (text) ou textures.

A physical signal like a natural image can show up all three types of redundancies. For the lossless image compression treated in this paper, only the reduction of local redundancies is relevant. Reducing data by processing the other types of redundancy leads to approximations and therefore fit well for lossy compression methods (Lifting scheme SGWT[8] or frequency-domain quantification (JPEG) or Fractal compression[2]). Using principles designed for lossy compression and doing lossless is mistaken : JPEG2000 in lossless mode performs worse than PNG !

As shown in fig.1, the image is split into blocks, and the data within a block is scanned by a given path (say scanline) twice:

- The first scan computes, for each pixel, the variance of the neighborhood. The shape and size of the causal neighborhood (which is used by subsequent predictor) is fixed (ca. 12 pixels) and therefore is much larger than the neighborhood used by the PNG predictor. Based on these neighboring pixels, and using linear regression, the optimal values of the weights for a linear predictor which minimize the mean squared error can be computed:

A	:	Image block, vector of size n
p	:	Number of neighbor, scalar
V	:	Causal neighborhood, matrix $n \times p$
t	:	Weighting coefficient, vector of size p

  $$\underset{t}{\operatorname{argmin}} \ ||A - Vt||_2 \quad \rightarrow \quad t = (V^t V)^{-1} V^t A$$

  A different criterion like minimization over L_1 (minimize the absolute deviation) should be more relevant for maximize amount of zeros, but the computation stage of the weights is definitely much longer. A single average over the whole data block can be determined, but better prediction is reached if we consider neighborhood classes (for example, 8 classes based on the variance for each data block) and then optimum weights are computed for a predictor assigned to each class.

  T : Weight coefficients by variance ranges, matrix of size $p \times 8$
  $i = [0..7]$: Variance index
  $u_{-1} = 0, \ u_7 = +\infty$: Bound of variance's intervals
  $\Omega_i = [u_{i-1} \leq \sigma(V) \leq u_i]$: Data set with same range of variance

  $$A_i = A(\Omega_i), \quad V_i = V(\Omega_i), \quad T = t_i = (V_i^t V_i)^{-1} V_i^t A_i$$

  Therefore, a on-the-fly process clusters the scrutinized neighborhoods in 8 variance classes, computes optimum weights and remember the amount of neighborhoods belonging to each individual class (this information is used for data sorting).
- The second scan applies the choose the predictor according to the variance of the neighborhood, computes the error between the predicted value and the actual one, and store the error to an allocated storage for that class. Therefore, after this step, data is sorted according the variance of it's initial neighborhood (this sorting groups for example small valued error signals together). For reversal data reconstruction at decompression, only the histogram (amount of neighborhood belonging to each class) is needed.

As disclosed previously, each data block requires for the present compression scheme a certain amount of crucial metadata (ie. optimum weights for that block used by the predictor, ...). These metadata needs to be stored in a very safe way, for example using EDC/ECC and replication, or a different storage channel, and we assume this in our study.

We define a ratio expressing the amount of metadata to the amount of raw (to be compressed) data in a block : $Qm = \frac{Metadata}{Metadata+Raw}$. For a fixed neighborhood, the Qm ratio increases if data block shrinks (ie: $raw = 1024$ Bytes, $p = 12$, $Coefs = 12 \times 8 \times \frac{3}{2} = 144$ Bytes, $Qm > 12.3\%$). For best compression rate, it appears that this ratio should be adaptive to the complexity of the data. Compression of the metadata itself was also examined. Our experimentation shows us that a lossy compression scheme can be used here: PCA (Principal Component Analysis) and precision loss of the weights is absolutely not critical for the image reconstruction, which stays near to optimum if the block show a low variance. If the block is textured or noisy (weak spatial redundancy), the prediction itself is indeed inaccurate and weight precision pointless. This lossy compression scheme for this kind of metadata (predictor weights) is not described in this paper.

Binary RLE

If we leave the robusness topic out, premium choice for compression would be an arithmetic coder which, according to litterature and recent advances in compression, can achieve a compression rate close to Shannon's limit. However, a single bit error in a compressed (by an arithmetic coder) data stream leads to strong corruption and spatial shifting of reconstructed data (often complete bunches of 32 or 64 bits are corrupted).

We propose an alternative scheme based on binary RLE (Run Length Encoding) which has the following advantage with regard to error resilience: even without the subsequent EDC/ECC, a single bit swapping introduces only a small shift, and this shift can be minimized in some extents. Furthermore, binary RLE is faster than a-coder, since it does not require any multiplication/division operations and is well-suited for sparse signal representation.

```
Simple Binary RLE
S : Binaryinput
m : nbbitcoding
$\omega \leftarrow 2^m - 1$
$j \leftarrow -1$
for i in $0 \ldots n-1$
    if S[i] then
        $\Delta \leftarrow i - j$
        $j \leftarrow i$
        while $\Delta > \omega$
            $\Delta \leftarrow \Delta - \omega$
            send(0)
        end while
        send($\Delta$)
endfor
```

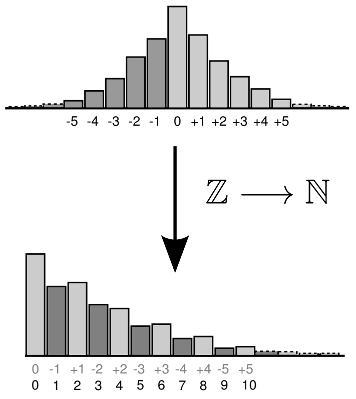
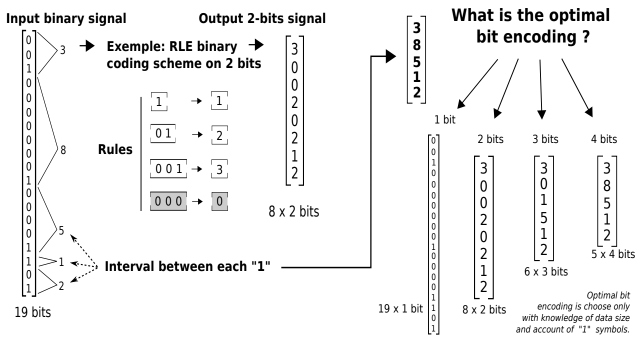


Figure 2. Left: the basic RLE algorithm. Right: the data remapping improving the imbalance in favor of zeroes, suited to handle error data (distribution centered at 0). The remapping algorithm is **if** $(x < 0x80)$ **then** $y \leftarrow SHL_1(x)$ **else** $y \leftarrow NEG(SHL_1(x) + 1)$ *where SHL is a binary Left SHift.*

The RLE encoding is rather classical. Since we work on a binary data stream, we store the length of the run to the next "1". The algorithm in pseudo-code is given in fig.2, and fig.3 illustrates RLE by example. The binary data stream to encode is a error stream (prediction minus actual value), and therefore contains signed values (the statistical distribution is centered at zero and shows a high peak at zero). So we improve the RLE by applying a bijection from signed to unsigned integers, remapping the coding of signed integer : the sequence -1,+1 usually coded as $11111111_b, 00000001_b$ will become $00000001_b, 000000010_b$. The purpose of this mapping is to significantly reduce the amount of 1 before RLE.

After this remapping, data is split in subblocks, according to binary bit rank and belonging to a variance class. For example, a subblock is the bit stream formed by all MSB bits of all values belonging to variance class V_0. Since data is encoded onto 8 bits and clustered in 8 variance class, our process handles $8 \times 8 = 64$ subblocks (all subblocks can be processed in parallel). We assume that each one of these subblock is now spatially decorrelated, and we focus on statistic encoding.



Input binary signal
Output 2-bits signal
Exemple: RLE binary coding scheme on 2 bits
Rules
1 → 1
0 1 → 2
0 0 1 → 3
0 0 0 → 0
8 x 2 bits
Interval between each "1"
19 bits
What is the optimal bit encoding ?
1 bit
2 bits
3 bits
4 bits
19 x 1 bit
8 x 2 bits
6 x 3 bits
5 x 4 bits
Optimal bit encoding is choose only with knowledge of data size and account of "1" symbols.

Figure 3. RLE by example : Given a binary stream (left), choosing the correct word size to encode the length of a run is critical. The figure shows encoding using 2 bits words, then shows, at the right side, the encoding using several options (1 bit words - no compression, 2bits, 3 bits and 4 bits words)

Major issue is now to choose the right number of bits for coding a run, as illustrated fig.3. If 8 bits are used (sequences up to 255 consecutive zeroes) and "one" symbols appear very often in this sequence, says every tenth "zero", the compression rate will be very low since 4 bits have been enough. On the other hand, choosing shorts words to encode a long run would rise the amount of [00...0] codewords, tagging a sequence of zeroes that exceeds coding capacity. Optimal length (m = word length to encode a run) given a given probability p of "ones" in a subblock can be precisely computed, and lead to a compression ratio $R_m(p)$:

$$R_1(p) = 1, \;\; R_m(p) = mp^2 \textstyle\sum_{x=0}^{+\infty} (1-p)^x \left\lceil \frac{x+1}{2^m-1} \right\rceil = \frac{mp}{1-(1-p)^{2^m-1}}$$

One can easily computes the probabilities of transitions (see fig.4) p_{12}, p_{23}, p_{34}, p_{45}, $p_{56} \ldots$ They are the roots of a polynomial with three monomials given by:

$$p_{m,m+1} = p = 1 - q, \quad (m+1)q^{2^m-1} - mq^{2^{m+1}-1} = 1$$

Remark, only p_{12} can only be solved analytically, $p_{12} = \frac{3-\sqrt{5}}{2} = 2 - \phi$, where ϕ is the golden ratio. The others are calculated iteratively by the following scheme:

$$q_0 = q_{approx}, \quad q_{i+1} = q_i - \frac{(m+1)q_i^{2^m-1} - mq_i^{2^{m+1}-1} - 1}{(2^m-1)(m+1)q_i^{2^m-2} - (2^{m+1}-1)mq_i^{2^{m+1}-2}}$$

Two reasons lead us to improve our model:

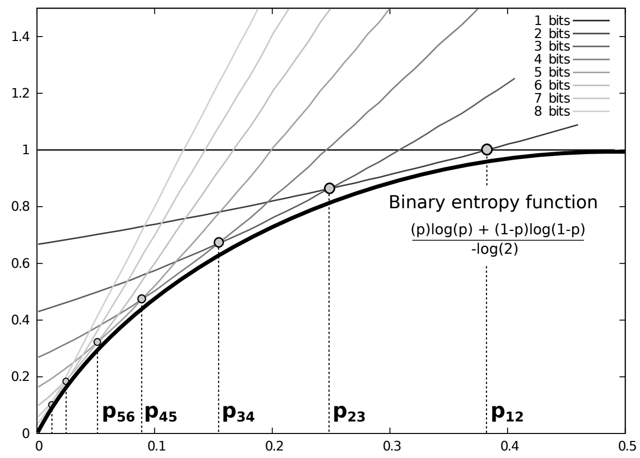


Figure 4. *Plot of compression rates $R_m(p)$ as a function of the probability p of "ones" in a binary stream. Bold curve: Binary entropy limit. Each straight ligne give $R_m(p)$ according to choosen word size m used for encoding the length of the runs. Best compression rate for a given p is achieved by picking the line segment close to the Shannon limit. The intersections of the straight lines have been calculated off-line and stored in a table () that the compression algorithm uses to find the best word size.*

- For computational efficiency, it is preferable to handle with usual machine word, (ie packets of 32-bits, rather than sending for example 3-bit one after the others).
- It is tempting to encode on fractional values of encoding bit. For example, an optimal compression ratio can be obtained using 2.5-bit rather 2 or 3-bit.

For these reasons, it is interesting to use bit patterns. As 32 is small, we can exhaustively enumerate all integer values a,b,c,d *such that* $a \times c + b \times d = 32$, $a = b+1$. The pattern *num*23 (see next tab) corresponds for instance to $4 \times 5 + 3 \times 4 = 32$. We can order it in different ways, for instance: [4 4 4 4 4 3 3 3 3] (splited) or [4 3 4 3 4 3 4 3 4] (enterleaved). An useful property is that the order of the bits does not affect the compression ratio, since the only the knowledge of their probability is need to estimate the ratio. This shows that the order does not matter. We exhaust all possibilities,

num	a	c	b	d
0	1	32	0	0
1	2	1	1	30
2	2	2	1	28
3	2	3	1	26
4	2	4	1	24
5	2	5	1	22
6	2	6	1	20
7	2	7	1	18
8	2	8	1	16
9	2	9	1	14
10	2	10	1	12
11	2	11	1	10
12	2	12	1	8
13	2	13	1	6
14	2	14	1	4
15	2	15	1	2
16	2	16	1	0
17	3	2	2	13
18	3	4	2	10
19	3	6	2	7
20	3	8	2	4
21	3	10	2	1
22	4	2	3	8
23	4	5	3	4
24	4	8	3	0
25	5	4	4	3
26	6	2	5	4
27	7	2	6	3
28	8	4	7	0
29	11	2	10	1
30	16	2	15	0
31	32	1	31	0

$\Longrightarrow$

Only useful patterns						
	Combos				Intervals	
num	a	c	b	d	divide by 2^{32}	decimal expansion
0	1	32	0	0		
					1640531526	0.381966011133
16	2	16	1	0		
					1071497016	0.249477339908
21	3	10	2	1		
					722660323	0.168257468147
22	4	2	3	8		
					658891454	0.153410121333
24	4	8	3	0		
					386576822	0.090006930288
25	5	4	4	3		
					297609193	0.069292539963
26	6	2	5	4		
					177737969	0.041382845724
27	7	2	6	3		
					84073371	0.019574857084
28	8	4	7	0		
					23220235	0.005406382261
29	11	2	10	1		
					2908179	0.000677113188
30	16	2	15	0		
					45426	0.000010576565
31	32	1	31	0		

Figure 5. *Show only useful patterns, that means they are the closest to the binary entropy function. For a given probability of "1", for example $p = 0.08$, we must encode runs on 5 bits (4 times) and 4 bits (3 times).*

By expanding, we obtain the general formula for any pattern consists of n values different encoding. In the pattern, there are c_1 times a_1 , c_2 times a_2 , etc ... $R_{gen}(p) = \frac{p\sum_{i=1}^{n} a_i c_i}{\sum_{i=1}^{n} c_i(1-(1-p)^{2^{a_i}-1})}$
For this we are interested (32-bits and only 2 encoding values in the pattern), we have: $R_{abcd}(p) = \frac{32p}{c(1-(1-p)^{2^a-1})+d(1-(1-p)^{2^b-1})}$
If we take the 32 possible combinations, we look for the subset $\forall p \in [0 : \frac{1}{2}]$, $P_{abcd}(p) = min$ (12 patterns, i.e. previous tab).
We must determine the 11 transitional probabilities between 2 pattern. As before, they are not analytically computable. However, we can calculate recursively (*Newton method*) and give the optimal rational approximate on a finite numerical 32-bits precision, which are given in descending order fig.5 and achieved by:

$$q_{i+1} = q_i - \frac{\bar{c}(1-q_i^{2^{\bar{a}}-1}) + \bar{d}(1-q_i^{2^{\bar{b}}-1}) - c(1-q_i^{2^a-1}) - d(1-q_i^{2^b-1})}{-\bar{c}(2^{\bar{a}}-1)q_i^{2^{\bar{a}}-2} - \bar{d}(2^{\bar{b}}-1)q_i^{2^{\bar{b}}-2} + c(2^a-1)q_i^{2^a-2} + d(2^b-1)q_i^{2^b-2}}$$

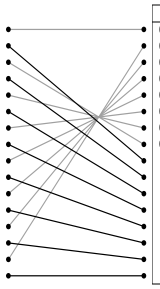
Where $a,b,c,d,\ \bar{a},\bar{b},\bar{c}$ *and* $\bar{d}$ *are parameters of two adjacent patterns*

Robusteness and Parity check

Before storing the compressed data, the binary coding is again remapped, this time to be more robust to data corruption. For example, if binary data 0011_b means 3 and encode the length of a run, a single bit change to 0111_b means 7, and introduce a significant shift in the data. Using rearrangement procedure and gray code (which is widely used to facilitate error correction) for coding the length of a run ensures that the effect of error is minimizes in case of a single bit corruption.

```
circ() procedure :   if $\mathtt{AND}(x,1)$ then $y \leftarrow \mathtt{ROR}_1(x)$
                                        else $y \leftarrow \mathtt{SHR}_1(\mathtt{NEG}(x))$
gray() procedure :   $z \leftarrow \mathtt{XOR}(y, \mathtt{SHR}_1(y))$
```

dec(x)	x	run(x)	y = circ(x)
0	0000	000000000000000	0000
1	0001	1	1000
2	0010	01	0111
3	0011	001	1001
4	0100	0001	0110
5	0101	00001	1010
6	0110	000001	0101
7	0111	0000001	1011
8	1000	00000001	0100
9	1001	000000001	1100
10	1010	0000000001	0011
11	1011	00000000001	1101
12	1100	000000000001	0010
13	1101	0000000000001	1110
14	1110	00000000000001	0001
15	1111	000000000000001	1111



y	run(x)	z = gray(y)
0000	000000000000000	0000
0001	00000000000001	0001
0010	000000000001	0011
0011	0000000001	0010
0100	00000001	0110
0101	000001	0111
0110	0001	0101
0111	01	0100
1000	1	1100
1001	001	1101
1010	00001	1111
1011	0000001	1110
1100	000000001	1010
1101	00000000001	1011
1110	0000000000001	1001
1111	000000000000001	1000

Figure 6. *Robustification using gray code for encoding the run lengths. A single bit change only alters the lenght by +1 or -1. In the pseudo-code, ROR means ROTate the binary word to the Right, and SHR SHift binary word to the Right.*

After remapping as gray code, all data is protected by EDC/ECC. The Error Correction Code is quite similar to a Extended Hamming Code [11], [1] and [10], except that the parity check is labelled as metadata and is assumed to be stored in a completely reliable way (replication, different storage channel, ...) as stated before. Like explained in subsection , the amount of metadata should be kept as low as possible to preserve the overall targeted compression rate of the method.

Ideally, the size of data should be 2^K bits ($K \in \mathbb{N}$), K has to be lowerer if probability of bit corruption is high. The EDC/ECC process should be able to recovering data up to a certain degree of data corruption. In case of error bursts or if corruption rate is high, the ECC is overwhelmed but EDC is still signaling the errors.

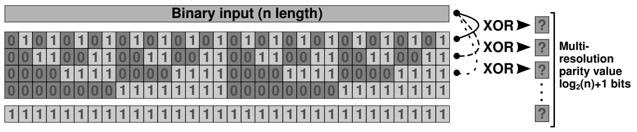


Figure 7. Parity check computation reach in single pass.

for i **in** $0\ldots n-1$ **do** **if** $T[i] \neq 0$ **then** $u \leftarrow u$ ^ (i&0x8000000) **endfor**

Checking data against EDC/ECC parity data is fast (linear complexity) and can be done directly on 32-bits words using a 16-bits lookup table.

If only one bit swap occured within a $n = 2^6 = 64$-bit block (this size is used for our experimentation), data is fully recovered. If two bits are altered, the error is detected and ECC information helps to reduce the set of possibilities for recovering from $\frac{n(n-1)}{2}$ to a subset S containing $\frac{n}{2}$ cases. If more bits errors arise, corruptness is often detected (much better if n is large) and the ECC is fooled and adopts recovering strategy corresponding to 1- or 2-bits errors. In fact, errors involving a odd number of bit changes are always detected, while even number of errors are often detected. According to below, the exact probability is given by $1 - \frac{1}{2n}$ (for example, a 1024 bits signal, the probability to detect a corrupted signal is 99.951%),

$$n : Length\ of\ signal \qquad r : Number\ of\ errors\ (even)$$

$$E_{n,r} = \{\forall\, i_0, i_1, i_2, \ldots i_{r-1} \in \mathbb{N} \mid \bigoplus_{k=0}^{r-1} i_k = 0, \\ 0 \leq i_0 < i_1 < \cdots < i_{r-1} \leq n-1\}$$

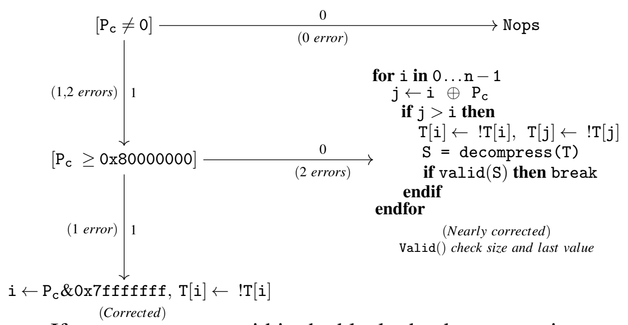
$$P_{n,r} = \frac{Card(E_{n,r})}{\binom{n}{r}} \qquad \sum_{k=0}^{n} P_{n,k} = \frac{U_n}{2^n},\ \ U_4 = 4,$$

$$U_n = \begin{cases} U_{n-1} + 1 & if\ \ n = 2^K + 1\ (K \in \mathbb{N}) \\ 2U_{n-1} + 1 & if\ \ n = 4K + 1\ and\ n \neq 2^K + 1 \\ 2(U_{n-1} - 1) & if\ \ n = 4K \\ 2(U_{n-1} - 1) + 1 & otherwise \end{cases}$$

$$U_n \approx \frac{2^n}{n}\ \ and\ \ U_{2^K} = \frac{2^{2^K}}{2^K},\ \text{thereby}\ \ \sum_{k=0}^{n/2} P_{n,2k} = \frac{1}{2n} \quad (n = 2^K)$$

Underneath, we illustrate the decoding strategy using $\mathtt{P_a}$ (stored parity check data) and $\mathtt{P_b}$ (parity check computed at the decompression stage). XORing $\mathtt{P_a}$ with $\mathtt{P_b}$ give parity change $\mathtt{P_c}$.

$$\mathtt{P_c} \leftarrow \mathtt{P_a} \ \oplus\ \mathtt{P_b}$$



```
for i in 0...n−1
  j ← i ⊕ Pc
   if j > i then
     T[i] ← !T[i],  T[j] ← !T[j]
     S = decompress(T)
     if valid(S) then break
   endif
endfor
```

If two errors occur within the block, the decompression process could try to rigorously recover the signal, using each possibility from subset S for decompressing the data. After decompression trial, a scoring function can evaluate each recovering option. The description of this scoring function is out of the scope of this contribution. The function (named `Valid()` in the previous pseudogram) can rely on different criteria like pixel neighborhood coherence or checking if the decompressed data fits in the data blocks used at compression, etc...

If scoring is beneath a threshold for all cases, the decompression algorithm can switch to a lossy stragegy, filling the data that has been corrupted using interpolation or inpainting techniques.

Results

We apply our compression to well known images used by the image processing communauty, in order to compare the compression ratio against other methods. All images 512×512, luminance only :

Image	Compression Rate	Image	Compression Rate
lena	0.558318	baboon	0.753539
peppers	0.578509	cameraman	0.624422
elaine	0.617273	couple	0.521871
goldhill	0.621971	barbara	0.618704

References

[1] Joseph Jean Boutros, Albert Guillén i Fàbregas, Ezio Biglieri, and Gilles Zémor. Low-density parity-check codes for nonergodic block-fading channels. *IEEE Trans. Inf. Theor.*, 56:4286–4300, September 2010.

[2] G.M. Davis. A wavelet-based analysis of fractal image compression. *Image Processing, IEEE Transactions on*, 7(2):141 –154, February 1998.

[3] Yuji Umezu Ichiro Matsuda, Nau Ozaki and Susumu Itoh. Lossless coding using variable block-size adaptive prediction optimized for each image. *Proceedings of 13th European Signal Processing Conference (EUSIPCO 2005)*, 2005.

[4] Andrew Penrose Neil and Neil A. Dodgson. Error resilient lossless image coding. In *In ICIP, Kobe*, 1999.

[5] Franco Liberati Paolo Buonora. A format for digital preservation of imagesn, a study on jpeg 2000 file robustness. *D-Lib Magazine*, 14, 2008.

[6] J.J. Rodriguez. An improved fft digit-reversal algorithm. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 37(8):1298 –1300, aug 1989.

[7] I. Sodagar, B.-B. Chai, and J. Wus. A new error resilience technique for image compression using arithmetic coding. In *Acoustics, Speech, and Signal Processing, 2000. ICASSP '00. Proceedings. 2000 IEEE International Conference on*, volume 6, pages 2127 –2130 vol.4, 2000.

[8] Yan Tang and Yu-long Mo. Second generation wavelet applied to lossless compression coding of image. *Journal of Shanghai University (English Edition)*, 4:225–229, 2000.

[9] David S. Taubman and Michael W. Marcellin. *JPEG 2000: Image Compression Fundamentals, Standards and Practice*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[10] M. C. Valenti and J. Sun. The umts turbo code and an efficient decoder implementation suitable for software defined radios. *International Journal of Wireless Information Networks*, 8:203–216, 2001.

[11] Guosen Yue, Li Ping, and Xiaodong Wang. Generalized low-density parity-check codes based on hadamard constraints. *IEEE Transactions on Information Theory*, 53(3):1058–1079, 2007.

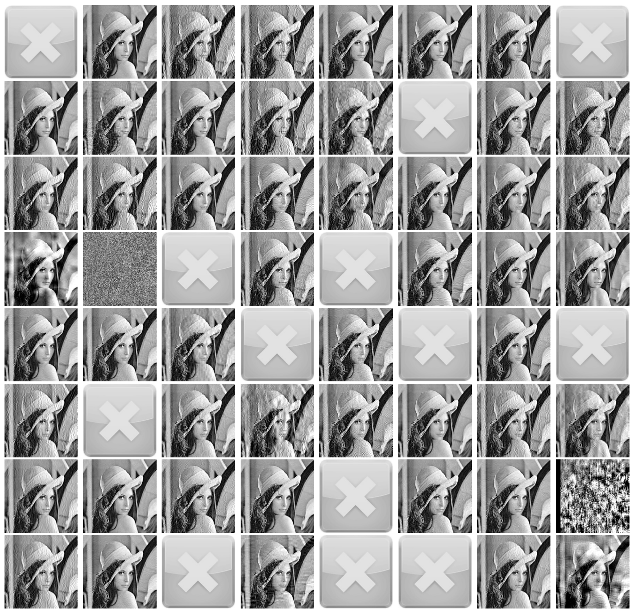


Figure 8. JPEG2000 Files stored with 100% quality. Average compression ratio is 0.58. Corruption of 100 random bits for each file. Results are very unpredictable, several files cannot be opened anymore with usual tools and image viewers (cross icons). By applying 200 random bit errors to each file, almost all of them fail to be opened.

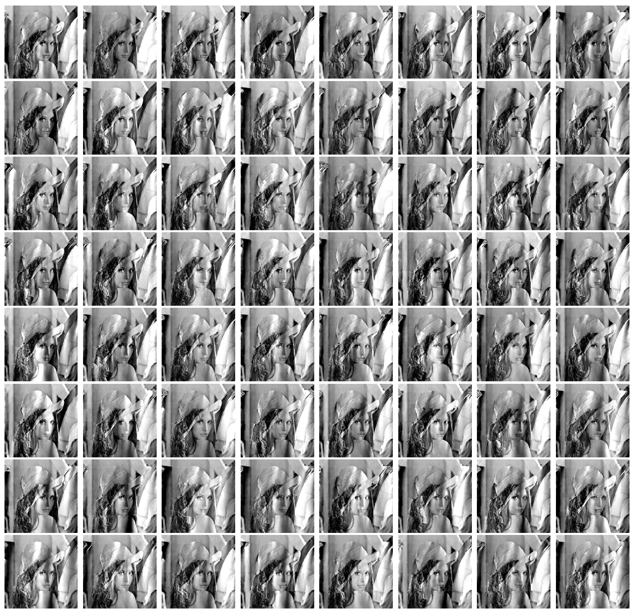


Figure 9. Storage using our compression scheme. Corruption of 1000 random bits per file (**10x** more than the JPEG2000 ones). **By applying only 100 random bit errors to each file, the EDC/ECC recovers all the data**. The average compression ratio is 0.55 (better than JPEG2000).

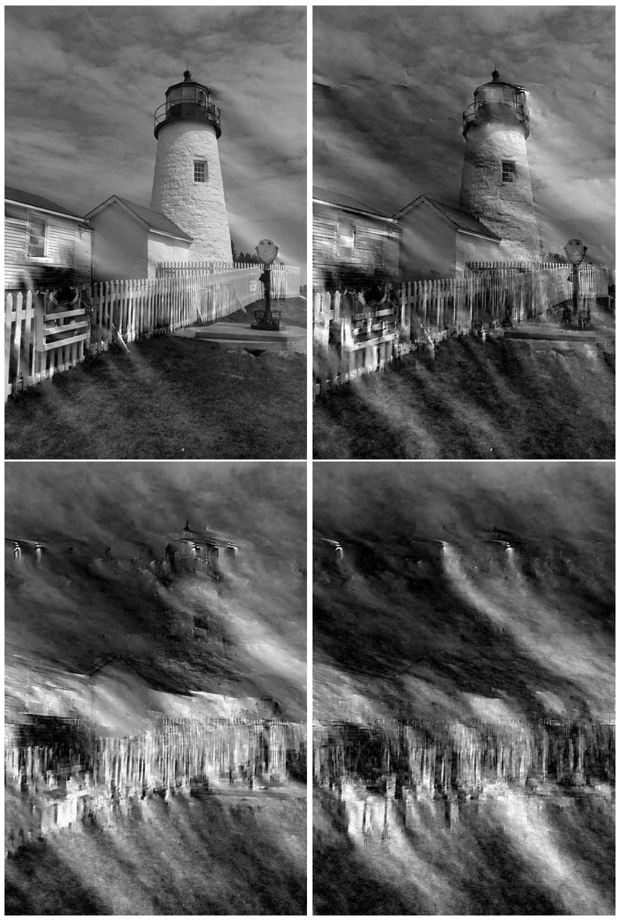


Figure 10. Even heavier error rates including bursts are applied to this second set of pictures. The image file stay always readable. Size is 512×768 pixel = 3145728 bits (8 bits/pixel luminance channel). From left to right, top to bottom : 0.01% corruption (314 bits altered), 0.1% (3145 bits), 1% (31457 bits) and 10% (314572 bits - that means that 1 bit out of 10 is swapped)

Author Biography

Charles GUYON *received in 2007 the B.S. and in 2009 the M.S. degrees in computer science applied to image processing from University of La Rochelle, France. He is currently working toward the Ph.D degree in MIA Laboratory (Mathematics, Image and Aplication) in La Rochelle. His current research interests include tensor low-rank decomposition, compressive sensing and background modeling for video surveillance.*

Bernard BESSERER *received an engineering degree (1986) and a PhD (1993) in applied Electronics, He is working in the field of image processing since his PhD, starting 1988 with computer vision for autonomous vehicles. In 1993, he got a position at the University of La Rochelle as assistant-professor and started 1995 research activities on methods and algorithms for digital film restoration and preservation (involved in the PRESTOSPACE project and the RESONANCES project, the later concerns an optical sound track restoration platform).*

Kodak test image	Compression Rate (our method)	Compression Rate PNG
image 14 (raft)	0.660096	**0.65276**

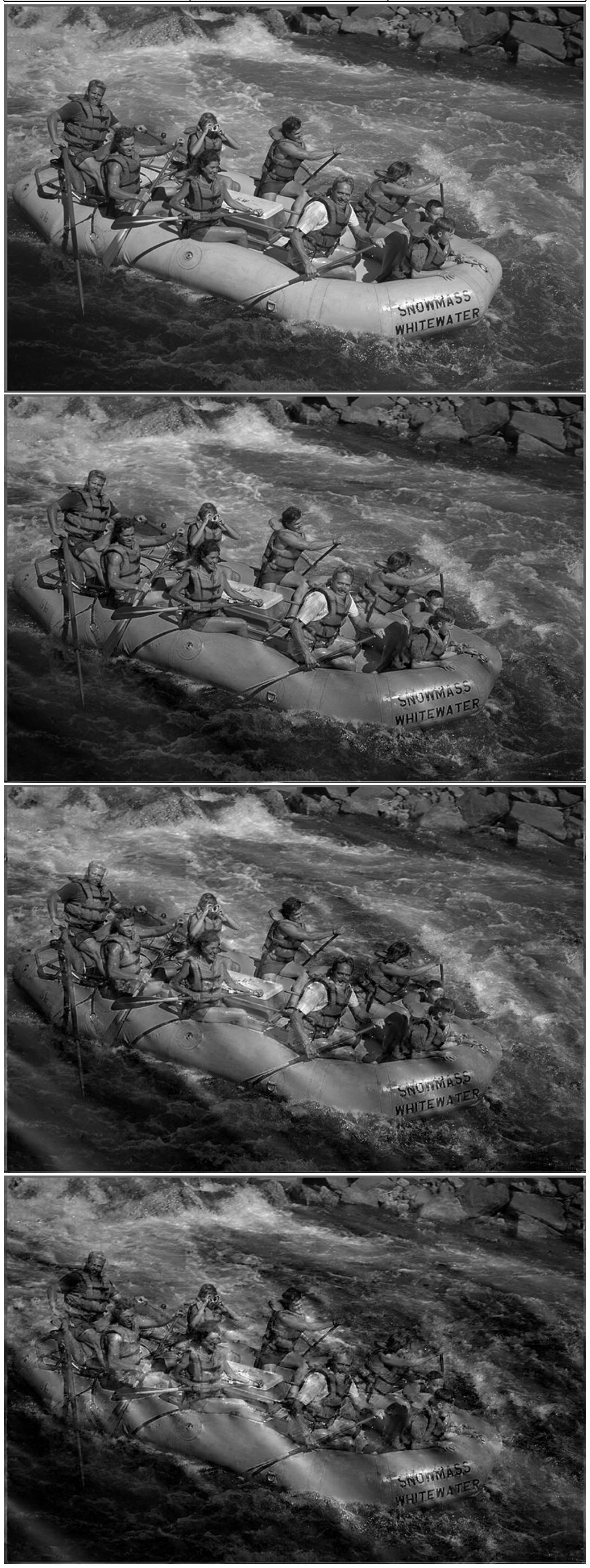


__Figure 11.__ This set of pictures illustrate the behaviour of the compression algorithm if the amount of bit errors exceed the ECC capacity. Size is 768×512 pixel = 3145728 bits (8 bits/pixel luminance channel). From top to bottom : Original, 0.01% corruption (314 bits altered), 0.04% (1258 bits) and 0.07% (2202 bits)

Kodak test image	Compression Rate (our method)	Compression Rate PNG
image 23 (parrots)	**0.475890**	0.49264

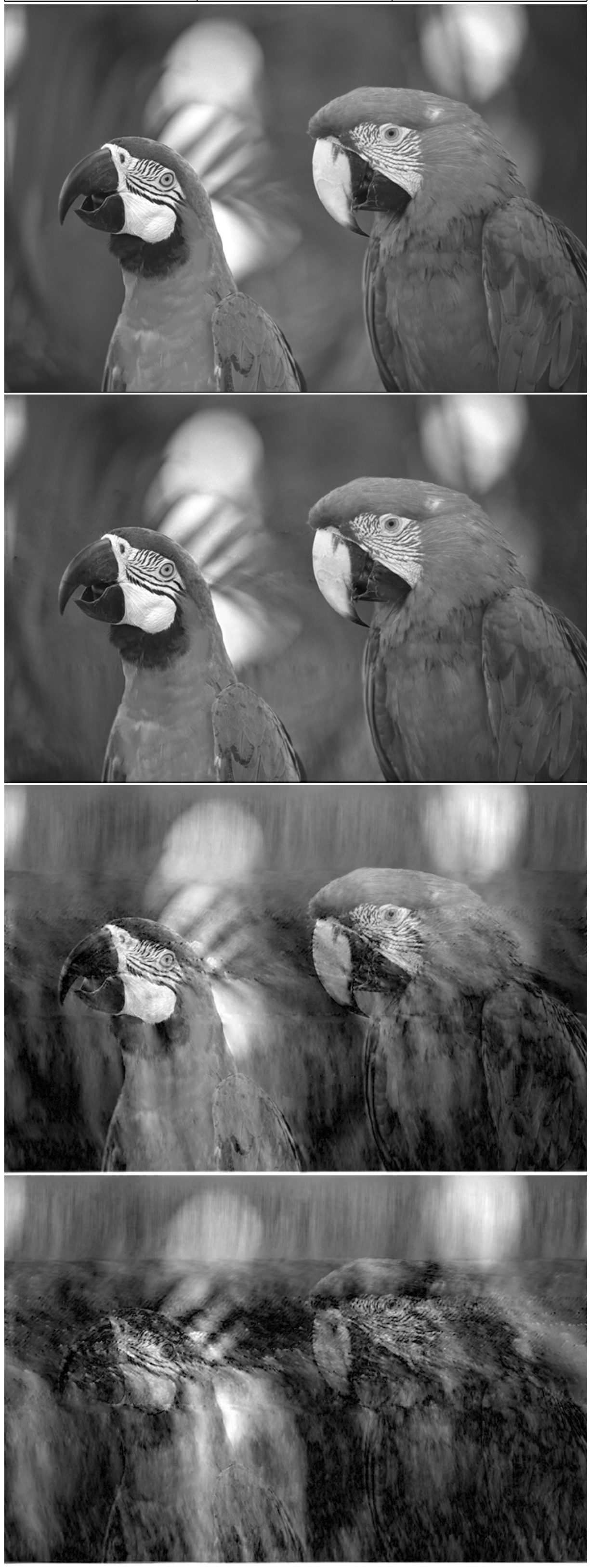


__Figure 12.__ Size is 768×512 pixel = 3145728 bits (8 bits/pixel luminance channel). From top to bottom : Original, 0.01% corruption (314 bits altered), 0.1% (3145 bits) and 1% (31457 bits)