**Title… TODO**

**William-Damian Balint**

**Abstract**

This paper gives a review of the state of the art lossless image compression algorithms and how they are combined in the most used image formats. It also evaluates the performance of the QOI format and shows how it can be stacked with other techniques to achieve better compression ratios than the well-established PNG format.

**1 Introduction**

Uncompressed images can occupy a large amount of space and can use a lot of bandwidth to be transferred from a device to another. For instance, a 1920x1080 colored image would take up 3×1920x1080 = 6220800 bytes, if stored uncompressed. Storing a one hour movie shot at 24 frames per second would occupy over 500 Gigabytes. This makes good compression schemes essential for today’s high-resolution and high-speed cameras.

This paper focuses on lossless compression, where the original uncompressed data must be recoverable from the compressed data. The second section reviews the state of the art compression methods, the third section presents how these algorithms are implemented in image formats and the fourth section contains benchmarks and experiments of these formats.

The word compression is used, but no algorithm can guarantee that it can achieve compression on any set of inputs, otherwise, the algorithm could be applied over and over until the data is represented in 0 bytes. In fact, all compression techniques may expand some inputs. This is why testing must be performed on data that is likely to be encountered in real-world scenarios and making sure that it compresses well.

**2 State of the art image compression techniques**

**2.1 Entropy Coding**

If every symbol in a sequence is encoded using the same number of bits, the code for each symbol is called a fixed-length code. Entropy coding techniques try to assign variable code lengths to symbols based on their probability, such that high-probability symbols are encoded in fewer bits.

These techniques assume that the probabilities of symbols are independent and identically-distributed. This is most often not the case, for example, in the English language the probability of ‘u’ is higher after ‘q’ than after any other letter. Compression schemes usually first use other techniques to leverage the dependencies of data and then add entropy coding at the end.

**2.1.1 Huffman Coding**

Huffman[1] presented a method for constructing optimum codes given a sequence of symbols. A sequence encoded using Huffman’s procedure has the property that no additional information about where a code begins and another one ends needs to be given if the code for each symbol is known by the decoder.

For this property to exist, every code must not be a prefix of another. For instance, a = 10, b = 01, c = 110, d = 1110 are valid codes as a sequence like 101100111100110110 can be broken up into 10-110-01-1110-01-10-110 resulting in “acbdbac”. On the other hand, for codes a = 10, b = 01, c = 100, d = 011 it is unclear what the first symbol in a sequence that starts with 10 is. If the 3rd bit is 0 then the sequence decodes to “c”, but then if the 4th bit is 1 the sequence decodes to “ab”, furthermore, if the 5th bit is 1 the sequence becomes “ad”.

Ensuring that no code is a prefix of another code makes decoding streaming data possible and also simplifies the decoding process in non-streaming scenarios.

To generate the codes, Huffman’s procedure constructs a binary tree as follows:

1. Create a leaf node for each symbol and add it to a priority queue based on their probabilities.
2. While there are more than two nodes in the queue:
   1. Remove the two nodes of lowest probability from the queue.
   2. Create a new node with these two nodes as children and the probability equal to the sum of the two nodes’ probabilities.
   3. Insert the new node in the queue.
3. Set the last remaining node as the root.
4. For each node in the tree, label the edge to its left child as 0 and the edge to its right child as 1.

The final encoding for each symbol is a concatenation of the labels on the path from the root to the symbol.

The following is an example of a possible encoding for the sequence: “daccabaadc” where the probabilities for each symbol are: P(a) = 0.4, P(b) = 0.1, P(c) = 0.3, P(d) = 0.2. Using the tree in Figure 1, we get the codes a = 0, b = 111, c = 10, d = 110. Thus, the sequence becomes 1100101001110011010.

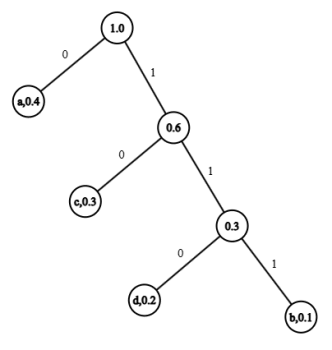


Fig. 1: Huffman tree generated to encode the sequence: “daccabaadc”

**2.1.2 Arithmetic Coding**

As opposed to Huffman Coding, Arithmetic Coding can encode a symbol in a rational number of bits on average. To illustrate this concept, let’s look at how many bits are needed to store one digit: there are 10 values so 4 bits are needed. But encoding pairs of two digits (100 values) requires 7 bits, so on average 3.5 bits are used per digit. Furthermore, encoding three digits would require 10 bits which would mean that 3.(3) bits are used per digit.

Arithmetic coding, introduced by Rissanen[2], doesn’t encode symbols separately like Huffman, but encodes the entire sequence of symbols into one real number between 0.0 and 1.0.

**2.2 Lempel-Ziv schemes**

**2.2.1 LZ77**

In their 1977 paper, Lempel and Ziv[3] introduced an algorithm for data compression based on removing duplicate occurrences of data, replacing the repeated elements with a pointer to existing data.

The pointer is a pair of an offset to the start of the sequence to be copied by the decoder and the length of it. Thus, the encoded data alternates between pointers and literals. For instance, the string “BANANA” may be encoded as “(0,0)B(0,0)A(0,0)N(-2,2)A”, where the parentheses represent pointers of the form (n,m), where *n* is the offset and *m* is the length. The encoded data must alternate between pointers and literals even if no repeated elements can be found, as is the case in the beginning. Thus, an empty pointer of the form (0,0) may be used.

**2.2.2 LZSS**

Storer and Szymanski[4], later improved on the method proposed by Lempel and Ziv by only emitting pointers when it is beneficial.

As an example, let us consider the string: “It was the best of times, it was the worst of times” which can get encoded as “It was the best of times, i(-26,10)wor(-27,11)”.

In the example above, LZSS achieves compression by referencing the repeating strings “t was the “ and “st of times”. However, the encoder must use a one-bit flag before every pointer or literal to tell the decoder what the next chunk of data represents.

**2.3 Burrows–Wheeler transform**

The Burrows-Wheeler transform[5] doesn’t compress the data, instead it rearanges a block of data to make it more compressible. Their algorithm transforms a string *S* of length *N* by forming *N* rotations (cyclic shifts) of *S* and sorting them lexicographically, extracting the last character from each rotation and forming a new string *L*. In addition to *L*, the algorithm also keeps track of the index *I* of the original string *S* in the sorted list of rotations.

To illustrate the technique, Table 1 shows how the string *S* = “banana” of length 6 is transformed into *L* = “nnbaaa” with *I* = 3.

| Index | Rotations | Sorted rotations |
| --- | --- | --- |
| 0 | banana | abana**n** |
| 1 | ananab | anaba**n** |
| 2 | nanaba | anana**b** |
| 3 | anaban | banan**a** |
| 4 | nabana | naban**a** |
| 5 | abanan | nanab**a** |

Table 1: Burrows-Wheeler transform for the string “banana”

In their paper, Burrows and Wheeler show how the original string can be retrieved in linear time and also how to use suffix trees to efficiently compute transformation. Later work by Seward[6] presents optimisations for sorting the rotations and compares them to other methods.

**2.3.1 Why the transformed string compresses well**

To see why this procedure might lead to effective compression, let’s consider a block of english text. We may assume that this block of text contains many instances of “the”. When the list of rotations gets sorted, all the rotations that start with “he” get bundled together and a large portion of them will end in ‘t’. Therefore, one region of the string *L* will contain a disproportionately large number of ‘t’ characters.

For example the string *S* = “it was the best of times, it was the worst of times” gets transformed into *L*= “se,ttssfftteww hhmmbootttts ii woeaaeerssii ”. It can be observed that instances of the same character tend to get grouped together.

**2.3.2 Run-length encoding**

Run-length encoding is a procedure that encodes data as pairs of literals and run-lengths. For instance, the string *L=* “hhhmootttt” could get encoded as“(h,3)(m,1)(o,2)(t,4)”.However, there are many ways to encode the run-lengths with minimal overhead.

Preston, Arnavut and Koc[7] have shown a compression pipeline based on the Burrows-Wheeler transform combined with run-length encoding and arithmetic coding that performs better on medical images than well established image compression formats such as JPEG-2000.

**2.3.4 Move-to-front transform**

This transformation uses a list of recently used symbols, encoding the data by replacing each symbol with its index in the list. After a symbol gets encoded, it gets moved to the front of the list. Thus, when a symbol appears frequently in an area of the original data, the encoded data will contain a large amount of small integers, which makes an entropy coder more effective.

The bzip2 format[8] uses the Burrows-Wheeler transform at its core, followed by the move-to-front transform, run-length encoding and huffman coding.

**2.4 Delta coding**

Delta coding is based on the observation that adjacent pixels tend to have similar values. Thus, instead of storing the values, it stores the differences between sequential bytes of data. For instance, the values 3, 6, 9, 8, 7, would get stored as 3, 3, 3, -1, -1. This doesn’t change the size of the input but it makes compression algorithms such as entropy coders more effective. In image compression, delta coding is usually performed separately on every channel since there doesn’t tend to be any correlation between values of separate channels.

**2.5 Context-adaptive binary arithmetic coding**

**3 Compressed image formats**

**3.1 The PNG format**

The PNG format consists of two main stages[9]. The first stage is called filtering and its purpose is not to do any compression, but to prepare the data for optimum compression in the second stage.

There are 5 types of filters that can be applied to every row in the image: None, Sub, Up, Average, and Paeth. The second stage is called “DEFLATE”[10] and it consists of LZSS followed by Huffman coding.

The “None” filter stores the row unmodified. The “Sub” filter stores the delta between adjacent pixels from the same row. The “Up” filter stores every pixel as the delta between itself and the pixel above it. The “Average” filter stores every pixel as the delta between itself and the average of the pixel above and the pixel to the left of it. The “Paeth” filter uses a prediction function based on the three neighbouring pixels (left, above, and upper left) and stores the deltas between pixel values and predicted values.

PNG encoders are allowed to choose any filter they want. However, the PNG specification recommends the following heuristic: compute the output of a row for each filter and select the one that gives the smallest sum of absolute values.

**3.2 The FLIF format**

FLIF[19] is a lossless image format based on MANIAC compression. MANIAC (Meta-Adaptive Near-zero Integer Arithmetic Coding) is a variant of CABAC (context-adaptive binary arithmetic coding), where the contexts are nodes of decision trees which are dynamically learned at encode time.

**3.3 The QOI format**

QOI is an image compression format that was developed for simplicity, but experiments showed that it achieved similar compression ratios to PNG and up to fifty times faster encoding[11]. The only compression methods it uses are: Run-length encoding, Delta coding and emitting backreferences to previously seen pixels. In order to emit backreferences, both the encoder and the decoder maintain an array of 64 recently seen RGB values[12].

The QOI format uses flags to specify the compression method of the next chunk of pixels. The QOI\_OP\_RGB flag is encoded in 8 bits and it signals to the decoder that the next 3 bytes represent the RGB values for the next pixel. This flag is only used when no other flag can be.

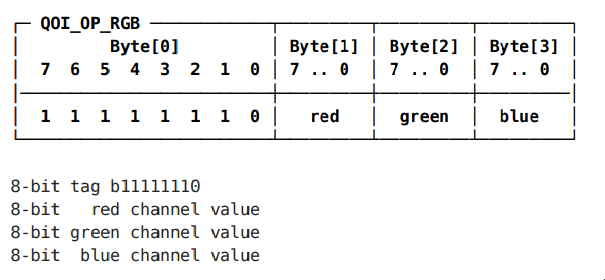


Fig. 2: Encoding of the QOI\_OP\_RGB flag

The QOI\_OP\_INDEX flag tells the decoder that the next pixel has the same RGB values as a previous pixel. It is encoded in 2 bits for the flag and the next 6 bits represent the index of the pixel to be copied.

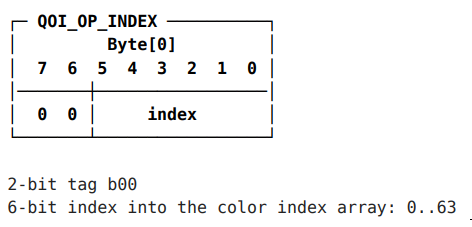


Fig. 3: Encoding of the QOI\_OP\_INDEX flag

The QOI\_OP\_DIFF flag tells the decoder that the next pixel is encoded as a delta from the previous pixel. This flag is only used when the delta for each channel is in the interval [-2, 1] so that it fits in 2 bits.

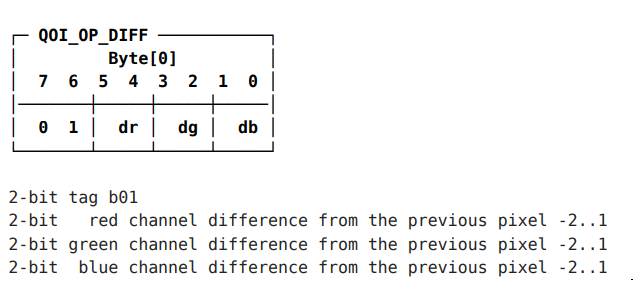


Fig. 4: Encoding of the QOI\_OP\_DIFF flag

The QOI\_OP\_RUN flag specifies the run-length of the previous pixel in 6 bits.

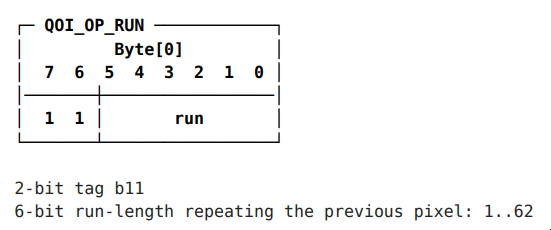


Fig. 5: Encoding of the QOI\_OP\_RUN flag

**4 Experiments**

We evaluated the compression performance of the presented file formats on two datasets. Additionally, to explore potential enhancements to QOI, we applied gzip (which uses the DEFLATE algorithm[10]), bzip2 (which uses the Burrows-Wheeler transform, followed by the Move-to-front transform, run-length encoding and huffman coding[8]). and 7z (which uses a Lempel-Ziv derivative called LZMA[17]) to already-compressed QOI files.

The datasets chosen for this experiment are:

1. WyohKnott[15] is a set of 100 diverse photographic images.
2. Rawzor RGB 8 bit[14] is a set of 14 high-resolution(like 7216x5412) photographic images chosen from a wide variety of sources, and each one picked to stress different aspects of algorithms.

The PNG files were encoded using an optimizer called PNGGauntlet[13]. The QOI images were encoded with [16]. The gzip encoder used compression level 9(highest), the bzip2 encoder used a dictionary size of 900KB(highest) and the 7z encoder used compression level 9, lzma2 and a dictionary size of 64MB. The FLIF images were encoded with [18] using the “best” option.

| Compression method | WyohKnott | Rawzor RGB 8bit |
| --- | --- | --- |
| PNG | 153445 KB | 220614 KB |
| QOI | 197723 KB | 326613 KB |
| QOI + gzip | 151955 KB | 272533 KB |
| QOI + bzip2 | 139898 KB | 248091 KB |
| QOI + 7z | 139085 KB | 253260 KB |
| FLIF | 88564 KB | 187680 KB |

Table 2: Comparison of compression on 2 datasets

**5 Conclusions**

In conclusion, combining QOI’s simple and fast encoding with bzip2 or LZMA lead to significant improvements in compression ratios, making it a competitor to PNG. Future research may examine the effects of adaptive arithmetic coding methods applied to QOI.

**References**

1. D. A. Huffman, "*A Method for the Construction of Minimum-Redundancy Codes*," in *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098-1101, Sept. 1952, doi: 10.1109/JRPROC.1952.273898
2. J. J. Rissanen, "*Generalized Kraft Inequality and Arithmetic Coding*," in *IBM Journal of Research and Development*, vol. 20, no. 3, pp. 198-203, May 1976, doi: 10.1147/rd.203.0198.
3. J. Ziv and A. Lempel, "*A universal algorithm for sequential data compression*," in *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337-343, May 1977, doi: 10.1109/TIT.1977.1055714.
4. Storer, James A., and Thomas G. Szymanski. "*Data compression via textual substitution.*" *Journal of the ACM (JACM)* 29.4 (1982): 928-951.
5. Burrows, Michael, D J Wheeler D I G I T A L, Robert W. Taylor, David J. Wheeler and David Wheeler. “*A Block-sorting Lossless Data Compression Algorithm.*” (1994).
6. J. Seward, "*On the performance of BWT sorting algorithms*," *Proceedings DCC 2000*. Data Compression Conference, Snowbird
7. C. Preston, Z. Arnavut and B. Koc, "*Lossless compression of medical images using Burrows-Wheeler Transformation with Inversion Coder*," 2015 *37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Milan, Italy, 2015, pp. 2956-2959, doi: 10.1109/EMBC.2015.7319012.
8. <https://sourceware.org/bzip2/manual/manual.pdf> (Accessed on the 27th of November 2024)
9. <http://www.libpng.org/pub/png/spec/1.2/png-1.2.pdf> (Accessed on the 30th of November 2024)
10. <https://datatracker.ietf.org/doc/html/rfc1950> (Accessed on the 30th of November 2024)
11. <https://qoiformat.org/benchmark/> (Accessed on the first of December 2024)
12. <https://qoiformat.org/qoi-specification.pdf> (Accessed on the second of December 2024)
13. <https://pnggauntlet.com/> (Accessed on the second of December 2024)
14. <https://imagecompression.info/test_images>/ (Accessed on the third of December 2024)
15. <https://github.com/WyohKnott/image-formats-comparison/tree/gh-pages/comparisonfiles> (Accessed on the third of December 2024)
16. <https://github.com/phoboslab/qoi/blob/master/qoi.h> (Accessed on the sixth of December 2024)
17. <https://www.7-zip.org/7z.html> (Accessed on the sixth of December 2024)
18. <https://github.com/FileMill/FLIF-GUI/releases> (Accessed on the fifth of December 2024)
19. <https://flif.info/spec.html> (Accessed on the fifth of December 2024