Compression for Petabytes of JPEG Images

Amir Said, Debargha Mukherjee
Media Technologies Lab
HP Laboratories Palo Alto
HPL-2007-158

entropy coding, lossless compression, compression standards, consumer photography

We propose a new method for parsing image files compressed with the baseline JPEG/JFIF standard, and re-encoding, without any image quality loss, its different components using more efficient compression algorithms. The new method is capable of achieving nearly 20% additional compression, with speeds of about 15 Mpixels/s for color images on a 3 GHz Intel Xeon processor. The advantage of using lossless compression is that it can be easily integrated to current systems and workflows, and in fact can be deployed in ways that are transparent to the user or the applications, while effectively saving storage and transmission bandwidth. These savings are particularly attractive for large scale imaging business, where the storage or bandwidth costs are a significant part of the business model.

External Posting Date: November 21, 2014 [Fulltext]
Internal Posting Date: October 21, 2007 [Fulltext]

Approved for External Publication

Compression for Petabytes of JPEG Images

Amir Said, Debargha Mukherjee
Media Technologies Lab
OST – HP Labs, Palo Alto
Amir.Said@hp.com, Debargha.Mukherjee@hp.com

Abstract

We propose a new method for parsing image files compressed with the baseline JPEG/JFIF standard, and re-encoding, without any image quality loss, its different components using more efficient compression algorithms. The new method is capable of achieving nearly 20% additional compression, with speeds of about 15 Mpixels/s for color images on a 3 GHz Intel Xeon processor. The advantage of using lossless compression is that it can be easily integrated to current systems and workflows, and in fact can be deployed in ways that are transparent to the user or the applications, while effectively saving storage and transmission bandwidth. These savings are particularly attractive for large scale imaging business, where the storage or bandwidth costs are a significant part of the business model.

1 Introduction

Due to the increasing availability of digital cameras, including those in cell phones and PDAs, large numbers of high-definition images are constantly being created. Figure 1 shows that—even if we exclude photos from camera phones—the number of worldwide digital photos that are being stored already reached hundreds of billions, and it is estimated that we will have *one trillion* digital photos by 2011 [12].

From a technical perspective, this number of photos represents an amount of information in the order of exabytes (10^{18} bytes), and all this data needs to be properly stored and managed [12]. The only reason we can deal with such large volume of data is because it is stored in a highly distributed fashion, i.e., we have millions of users, each typically keeping only a few thousand images.

We can still say that image compression is essential for keeping the data storage costs down, since almost all personal photos are stored in the JPEG compressed image

format. However, personal storage prices already fell so much that for the majority of users there is little incentive to push compression to its limits, and image quality is now clearly preferred over memory reduction. (This fact was empirically confirmed by analyzing the image compression ratios chosen by millions of users that upload their photos to Snapfish.)

On the other hand, there are many companies that provide imaging services, like photo storage and printing, and personal wireless communications, that need to store, transmit and manage very large volumes of images, in a much more centralized manner. For those companies (HP obviously included), compression can be much more valuable.

The amount of image data created by millions of users, with each uploading, transmitting, or storing thousands of photos, is quickly approaching the order of petabytes (10^{15} bytes). Consequently, the costs incurred with the transmission, storage, and management of such quantity of data, become quite substantial for a single company, and an important part of their cost structure and business model. In those cases it is economically very interesting to increase the efficiency of the compression used for images, even by relatively small amounts, like 10% (which would not be attractive enough for individual users).

2 Problem Statement

Currently the vast majority of personal digital photos is compressed using the baseline JPEG compression format [3]. However, it has been nearly 20 years after the development of JPEG's basic technology, and today there are image coding techniques that are significantly more efficient. In fact, the new JPEG2000 standard [6, 7] was created for its replacement, but since the widespread adoption of new standards depends on many factors, including the success of its predecessors (and JPEG had been extraordinarily successful), relatively very few applications use JPEG2000, and there is still much uncertainty about its widespread adoption on consumer products.

In addition, as indicated in Figure 1, there is already an extraordinarily large amount of images in the baseline JPEG format, and many have very high economic value. Unfortunately, trying to reduce costs by converting those images to new formats (transcoding), like JPEG2000 or the new proposed Microsoft's HD Photo, can be quite problematic. It is necessary to consider many unexpected costs and risks, including

- potential degradation of image quality at each conversion,
- unintended loss of image information (proprietary camera metadata, EXIF data, etc.),

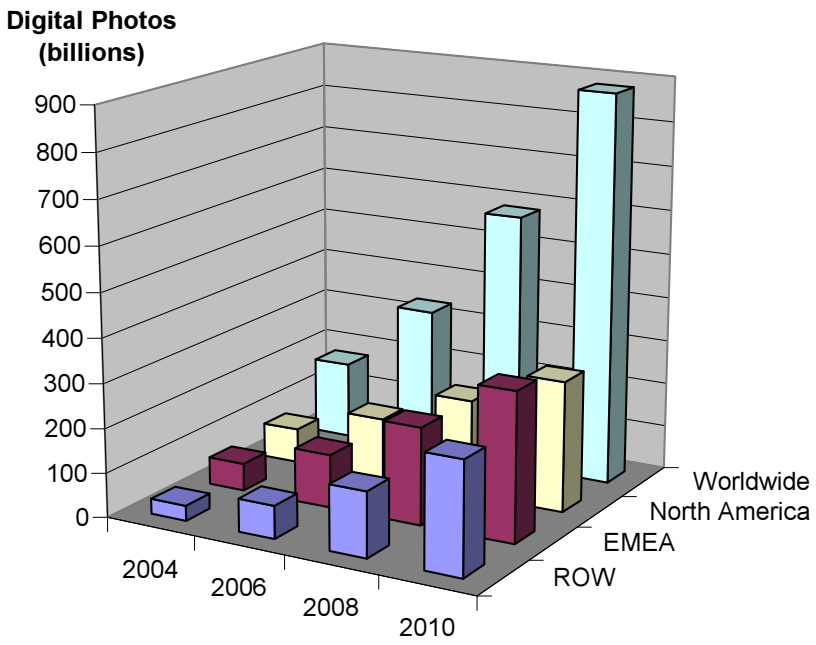


Figure 1: Estimated worldwide cumulative number of stored digital photos, not including photos from camera phones (source InfoTrends [12]).

- extra cost in the maintenance of multiple formats, changes in established workflows,
- unexpected costs with royalties, and other intellectual property issues,
- consumers get confused with formats, options, and conversions.

It is also possible to reduce file sizes by re-encoding the images using the JPEG format, but with settings for higher compression. While this is certainly an effective way to reduce bandwidth and storage, it causes irreversible loss in image quality. Thus, before using this approach it is necessary to consider how much degradation is acceptable, and to carefully evaluate possible unexpected consequences, including lawsuits for damaging content.

Under these circumstances, there are not many choices available, but it is still possible to reduce the costs of JPEG image storage and transmission. One particularly interesting solution is to re-compress the JPEG data in a lossless, but more efficient representation. With lossless compression every byte of the original JPEG file is preserved exactly, allowing reduction in costs, but with very little disruption to current workflows (more details in the next section). General-purpose methods for lossless data compression, commonly employing a variation of the Lempel-Ziv algorithms [1, 2, 8], are not suited for this application, since they yield only 1-3% additional compression when applied to JPEG files. A very different approach is needed.

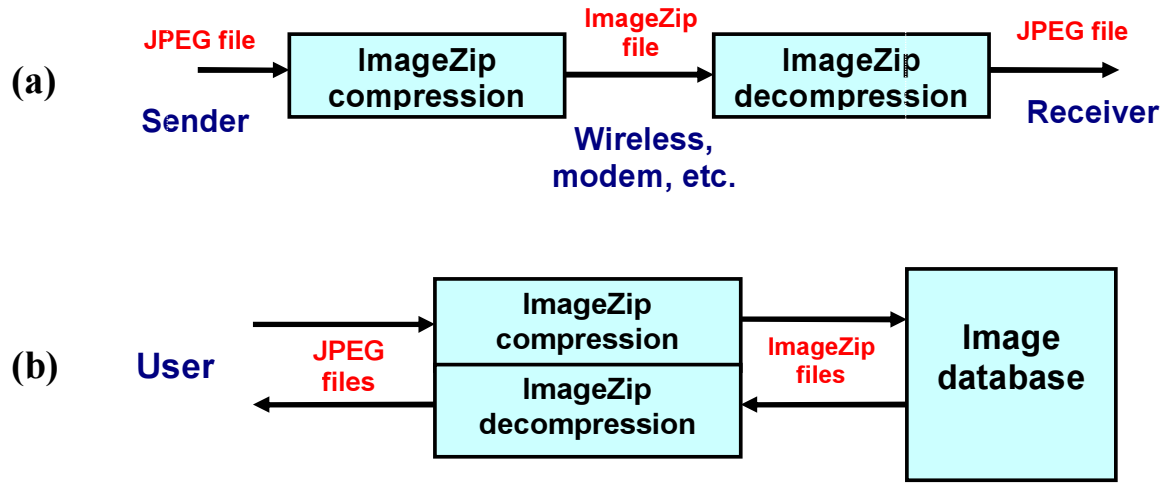


Figure 2: Application of ImageZip lossless extra-compression of JPEG files for (a) image transmission, and (b) image storage. The compression can be transparent to the user or application because every byte of the original JPEG file is preserved.

3 Our Solution: ImageZip

In order to obtain better compression it is necessary to parse the JPEG file, and re-encode its different components using more sophisticated compression methods. In this document we describe a method developed at HP Labs for such purpose, called *ImageZip,* with the name chosen to make clear that no data is lost, only better "packed," just like in a Zip file. Figure 2 shows two examples where ImageZip compression can reduce bandwidth and storage use. We can see that an advantage of ImageZip, compared to transcoding, is that it can be completely transparent to the final user or application, without producing any loss of information, and unintended change in the data.

It is important to avoid confusing the new compression method with others that have similar names. Figure 3 presents a diagram that helps understand the differences. On the top we have general purpose compression methods (i.e., that can be applied to any data file) like WinZip, commonly based on the Lempel-Ziv algorithms [8]. ImageZip compression does not belong to this class, the name "zip" was chosen only to help understanding how the method is used.

ImageZip is also quite different from the lossless profile of the first JPEG standard (lossless JPEG) [3], or the new JPEG standard for lossless compression, called JPEG-LS [5, 8]. These compression methods are applied directly to the bitmap (RGB) images.

ImageZip is meant to be applied to files that were created with the baseline compression method, as shown in the bottom Figure 3. It is called lossless, because there is no further loss of data at this compression stage, but it is important to know that there

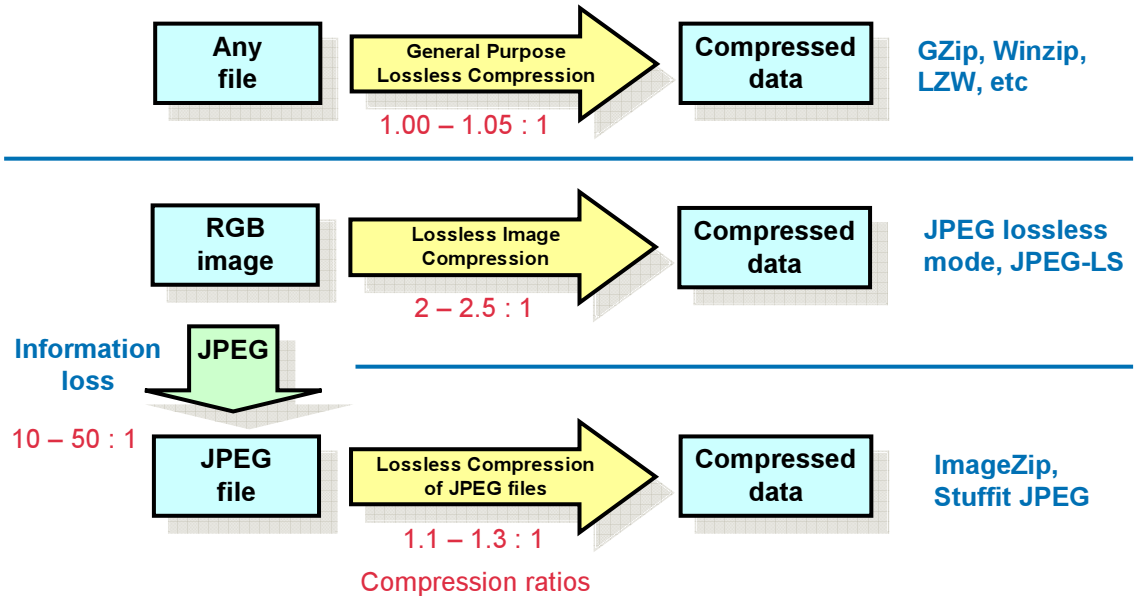


Figure 3: Comparison of different compression methods that are named lossless.

was loss when the JPEG compression was used.

We can also see in Figure 3 the common compression ratios for each method. When applied to images, or already compressed data, general purpose compression methods yield almost no further compression. Lossless image compression methods commonly yield only about 2:1 compression. JPEG compression, on the other hand, allows much higher compression because it discards information.

3.1 Technical Requirements

During the development of ImageZip we considered the following technical requirements.

1. While information theory predicts that increasingly better compression can be achieved with higher computational complexity, our method was meant for application to millions of images, and thus high computational costs are not acceptable. In addition, we did not consider asymmetric coding scenarios, i.e., those where encoding can be much slower than decoding, or vice-versa.

2. The compression improvement over baseline JPEG has to be large enough to compensate development and maintenance costs. Some estimates indicate that for very large image databases gains of about 10% or more are sufficiently attractive. However, note that this objective is in conflict with the first requirement, and achieving better compression in this case is not as easy as developing new image coding that

is superior to baseline JPEG, since those methods (like JPEG2000) can use different transforms. Because we need lossless compression of the JPEG-compressed data, our coding algorithms can only be applied to already-quantized coefficients of the DCT.

3. In collections with a large number of images, from different sources, we commonly find many images that do not satisfy the JPEG standard exactly. In addition, it is also necessary to deal with corrupted files (typically from digital cameras that are defective, with incorrect implementation, failing batteries, etc.). The implementation has to simultaneously include compressed flags to accommodate and code the deviations, but avoid program crashes due to the data corruption.

A very reliable implementation can be based on idea that, in case of doubt, it is better to keep the image in the original format (which is already compressed anyway), instead of risking a crash in the encoder/decoder. At the same time, any errors or violations to the standard that would not affect a displayed image (e.g., marker with incorrect bits), are compressed together with the image, so that after decompression exactly the same bits are restored.

3.2 Coding Method

Figure 4 shows how ImageZip compression is related to baseline JPEG. All the image components are processed and organized in the same manner, and the differences are in the final compression stages, when 8×8 blocks of DCT coefficients are entropy coded (i.e., efficiently represented with a small number of bits). Note that this simplifies considerably the conversion between JPEG and ImageZip files. Furthermore, it allows re-using many techniques that had been developed for JPEG, like optimized requantization, compressed domain scaling, lossless rotation, etc.

While in Figure 4 the coding stages (i.e., where JPEG and ImageZip are different) are not described in detail, it is important to keep in mind that the coding process requires quite sophisticated technology. In fact, this can be easily confirmed by counting the number of patents in the area (not to the mention patent infringement lawsuits).

Designing practical, efficient coding methods, is both art and science. On one hand, the fundamental theory, which is valid for any data source, has been established by Claude Shannon and other long ago [8, § 1]. On the other hand, practical methods have to take several conflicting objectives into account, and the innovation consists in choosing the right combination of techniques. For instance, below we describe some main factors to be taken into account in the design of an image coding method.

Flexibility: there is great variability in natural images, so a good compression method

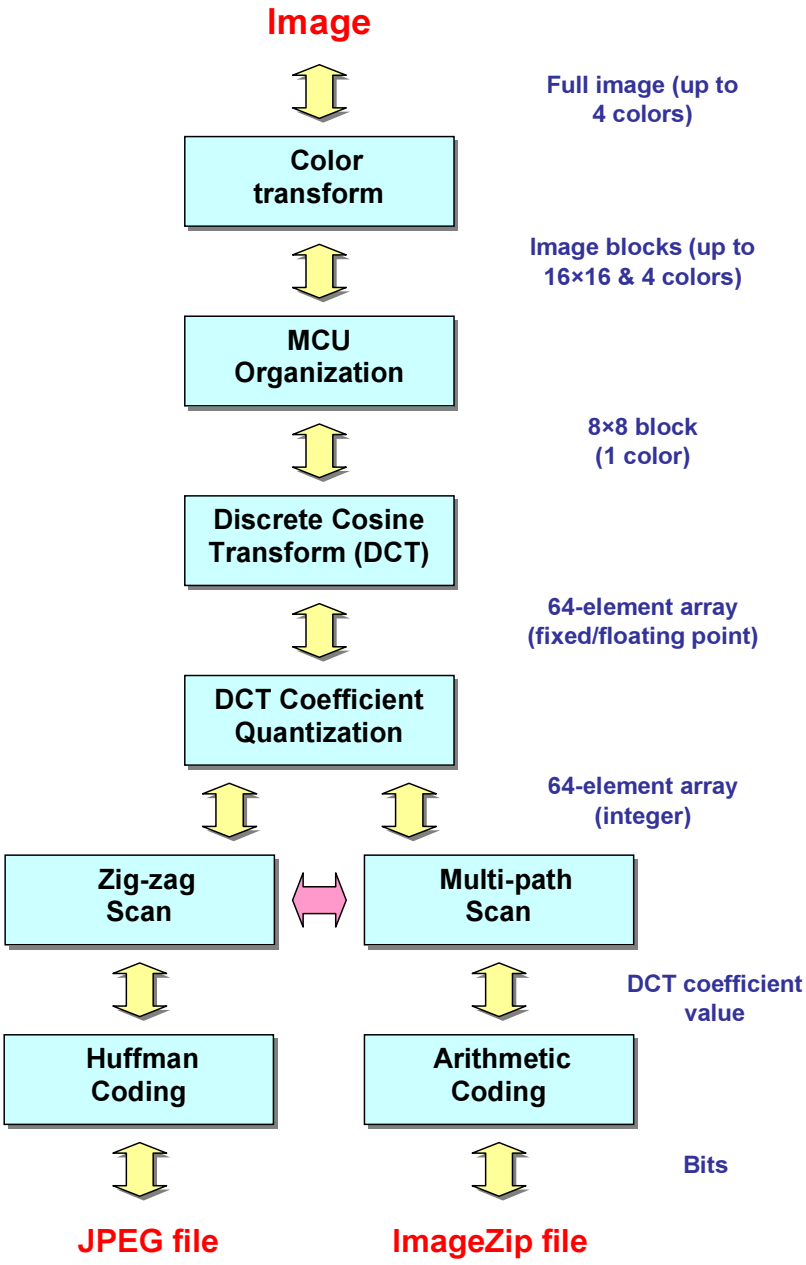


Figure 4: ImageZip uses nearly the same image coding components as JPEG. It differs only at a very low level, when DCT coefficients in 8×8 blocks are entropy coded, enabling efficient lossless format conversion.

should be equally efficient in images that look as different as landscapes, face portraits, group shots, etc.

Intelligence: the method quickly learn about an image's particular characteristics, and automatically create models that can improve compression.

Low complexity: information theory shows how to improve compression using increasingly more complex models. However, increasing a model's order can produce exponential growth in computational complexity, so a series of techniques have to be employed to keep the use of computational resources under control.

Structure: it is not possible to satisfy the conflicting objectives mentioned above without using some well defined structures, which have to be carefully chosen to represent properties that are particular to the data (in our case, the result of a two-dimensional discrete-cosine-transform applied to 8×8 image blocks).

In this section we present the methods used for coding in ImageZip, and explain why they were chosen. They leverage several years of image and data coding research at HP Labs, and more information is available in the references, especially the HP Labs technical reports [9, 10, 11, 13].

Due to space constraints, some details are not included, but it is interesting to note that in the coding method designed for ImageZip we have about 1,500 "intelligent agents" (adaptive models), that effectively learn only the necessary statistical properties in each image, in order to obtain good compression with very low complexity.

3.2.1 Scan of DCT Coefficients

An important part of the new compression method is a change in the order in which the 63 AC DCT coefficients are entropy-coded. JPEG follows the well-known zigzag order, starting from the low-frequency coefficients, and finishing with the end-of-block (EOB) symbol. The problem with this scan sequence is that it is only partially based on a statistical property that can be exploited for compression: it saves bits by compactly coding the zeros at the end, using the EOB) symbol. However, it is not very effective, especially to code blocks with more pronounced vertical or horizontal edge or gradient orientations, when the important coefficients are concentrated on block borders. This, in turn, requires using special schemes to deal with the runs of zero symbols that are interleaved with the larger coefficient values [4].

ImageZip uses a scan order that avoids these problems in two ways [13]. First, instead of using a single scan sequence for the whole block, it uses three fixed and independent 21-element scan sequences, as shown in Figure 5, which were chosen based on the statistical properties of the DCT coefficients. If we start from the high frequencies, then along each

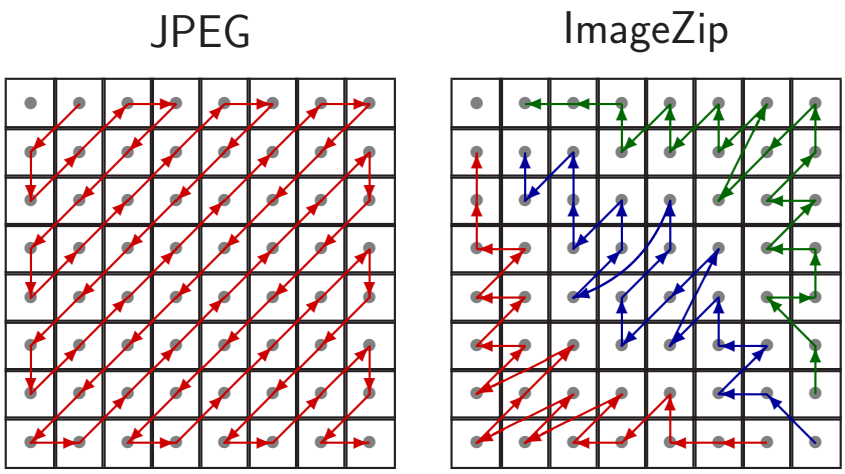


Figure 5: In the JPEG standard the DCT coefficients are coded following a single zigzag scan, from low frequencies, while in ImageZip they are divided in three sequences, which are coded following scans beginning from high frequencies.

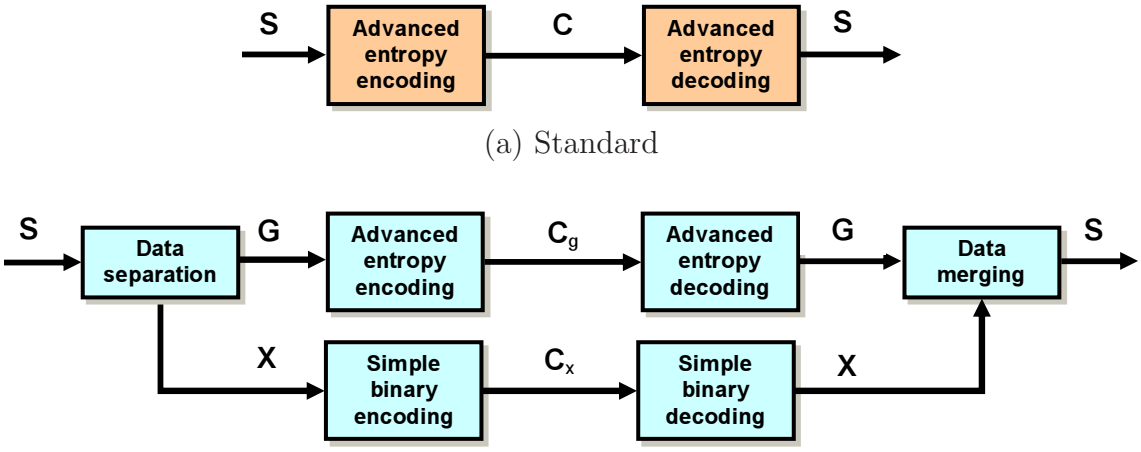
scan the variance of the DCT coefficients is expected to always increase. This property can be very effectively employed for additional compression, and in fact, ImageZip uses a new entropy coding method (described in Section 3.2.3, which was specifically designed to exploit this statistical feature.

3.2.2 Symbol Grouping

Symbol grouping is a technique that allows great reductions in the complexity of the entropy coding. In fact, one form of this technique is already used by the JPEG standard. The basic idea is that we do not want to spend any computational power trying to code data *components* that we know (from information theory principles) that cannot be compressed. A theoretical analysis of the symbol-grouping technique, including the reasons why it is so effective, is presented in report [11].

In the top of Figure 7 we show the form of symbol grouping used by JPEG. Each DCT coefficient value (a signed integer) is decomposed into three elements: group number, coefficient sign, and index. Complexity is reduced because only the group number is entropy coded, while the sign and indexes are saved in simple binary format ("raw" bits).

There is always a price to pay for this reduction in complexity, and the JPEG standard uses a form of symbol grouping that produces a small, but not insignificant loss. However, we have shown [11] that the loss in compression required by symbol grouping can be made negligible, even with small changes in complexity, and this property is used by the ImageZip compression. As shown in Figure 7, the ImageZip implementation



(a) Standard

(b) Using symbol grouping

Figure 6: Symbol group allows great reduction in computational complexity with negligible loss in compression by applying advanced entropy coding only to the information that can be effectively compressed.

reduces the redundancy (loss) to negligible values by partitioning the source alphabet into a larger number of groups. While JPEG uses at most 12 groups, we have at most 22 groups—nearly twice, but still much smaller than the thousands of symbols in the original alphabet.

3.2.3 Entropy Coding

In the design of entropy coding methods it is necessary to find compromise between compression efficiency and computational complexity. As we use more sophisticated coding methods, even with the symbol grouping technique defined in Section 3.2.2, we still can have large increases in complexity yielding only small gains in compression.

While designing ImageZip we had to consider that we needed compression that is significantly better than JPEG's, which is fairly efficient. For that reason, we decided to use some efficient implementations of adaptive arithmetic coding.

The theory and practice of arithmetic coding, and the development of efficient implementations, exploiting the fact that currently even low-cost embedded processors support fast 32-bit arithmetic, is presented in report [9] (which is also a chapter in [8, § 5]). Another report [10], presents the experimental results showing that its computational complexity now comparable to Huffman coding.

While adaptive arithmetic coding yields more compression than the Huffman codes used by JPEG, it is not sufficient. The compression had to be made significantly more efficient by exploiting the statistical properties of the DCT coefficient scan order pre-

Data type:																
0	±1	±2	±3	±4	±5	±6	±7	±8	±9	±10	±11	±12	±13	±14	±15	DCT coeff. value
0	1	2	2	3	3	3	3	4	4	4	4	4	4	4	4	Group number
	±	±, 0	±, 1	±, 0	±, 1	±, 2	±, 3	±, 0	±, 1	±, 2	±, 3	±, 4	±, 5	±, 6	±, 7	Sign, index

(a) JPEG

0	±1	±2	±3	±4	±5	±6	±7	±8	±9	±10	±11	±12	±13	±14	±15	DCT coeff. value
0	1	2	3	4	4	5	5	6	6	6	6	7	7	7	7	Group number
	±	±	±	±, 0	±, 1	±, 0	±, 1	±, 0	±, 1	±, 2	±, 3	±, 0	±, 1	±, 2	±, 3	Sign, index

(b) ImageZip

Figure 7: The symbol grouping for complexity reduction: group numbers are entropy coded, and sign and indexes are saved in simple binary form. JPEG has DCT coefficient values grouped according to $\lfloor \log_2(|n|+1) \rfloor$ (where n is an integer), while ImageZip groups symbols according to $\lfloor 2\log_2(|n|) \rfloor$, for $|n| > 1$, reducing the compression loss (i.e., redundancy).

sented in Section 3.2.1. The main idea behind the new coding method is to exploit the approximate order, simultaneously coding and dividing the data sequence in segments of elements with similar statistical properties. This way all these elements are coded with reduced-size alphabets, optimizing the coding process.

From the techniques described in the previous functions, the coding algorithm is applied to sequences of 21 non-negative integers, in the form $\{v_0, v_1, \ldots, v_{20}\}$, each equal to a group number of the corresponding DCT AC coefficient value, in one of the new scan sequences shown in Figure 5, Note that $v_i \in \{0, 1, \ldots, 21\}$ (cf. Section 3.2.2).

Following these definitions, the coding algorithm is basically defined as

1. For each 8×8 block, use a binary alphabet to code whether all symbols in all the three sequences are zero. This information can be coded using as context (for conditional entropy coding), for example, how many times the same event occurred on previous adjacent blocks. If they are all equal to zero then move to the next block. Otherwise go to step 2.

2. For each of the 3 scans in a block initialize state variables $t = 0$ and $n = 21$.

   (a) Code the position $p < n$ of the last coefficient in the sequence with value greater than t (called *segment separator*), using $p = -1$ to indicate if no such

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}	v_{16}	v_{17}	v_{18}	v_{19}	v_{20}
8	9	6	5	4	6	3	3	2	1	0	0	1	1	0	0	0	0	0	0	0

State		Separator position		Segment separator		Data segment	
n	t	Value	Alphabet	Value	Alphabet	Values	Alphabet
21	0	$p = 13$	$\{-1, 0, 1, \ldots, 20\}$	$v_{13} = 1$	$\{1, 2, \ldots, 21\}$	$\{v_{14}, v_{15}, \ldots, v_{20}\}$	$\{0\}$
13	1	$p = 8$	$\{-1, 0, 1, \ldots, 12\}$	$v_8 = 2$	$\{2, \ldots, 21\}$	$\{v_9, v_{10}, v_{11}, v_{12}\}$	$\{0, 1\}$
8	2	$p = 7$	$\{-1, 0, 1, \ldots, 7\}$	$v_7 = 3$	$\{3, \ldots, 21\}$	$\emptyset$	$\{0, 1, 2\}$
7	3	$p = 5$	$\{-1, 0, 1, \ldots, 6\}$	$v_5 = 6$	$\{4, \ldots, 21\}$	$\{v_6\}$	$\{0, 1, 2, 3\}$
5	6	$p = 1$	$\{-1, 0, 1, \ldots, 4\}$	$v_1 = 9$	$\{7, \ldots, 21\}$	$\{v_2, v_3, v_4\}$	$\{0, 1, \ldots, 6\}$
1	9	$p = -1$	$\{-1, 0\}$		$\{10, \ldots, 21\}$	$\{v_0\}$	$\{0, 1, \ldots, 9\}$

Figure 8: Example of data sequence with a typical set of 21 group numbers to be coded, and table with the corresponding results of the process of simultaneously subdividing and coding the sequence in segments with similar values. Different codes, with possibly different alphabets sizes are used for each type of information.

value is found. The separator position p is coded using an alphabet with symbols $\{-1, 0, 1, \ldots, n-1\}$.

(b) If $p \geq 0$ then code the value of the segment separator v_p using alphabet $\{t+1, t+2, \ldots, M\}$, where M is the maximum possible group number of a DCT coefficient.

(c) If $t > 0$ then code the values of all coefficients in the segment $\{v_{p+1}, v_{p+2}, \ldots, v_{n-1}\}$ using alphabet $\{0, 1, \ldots, t\}$.

(d) Set $t = v_p$ and $n = p$.

(e) If $p > 0$ then return to step 2(a)

(f) If last scan in the block then go to step 1, otherwise go to step 2.

Figure 8 shows an example of a 21-element sequence, a table with the data that is coded, and the corresponding code alphabets.

Note that the algorithm described above only provides an outline of how the data is coded. To obtain best results, it is combined with several other commonly-used techniques. For example, each type of data is coded using a combination of contexts, that depend on previously coded data and other factors, like the position of the element in the scan. Furthermore, the arithmetic codes are all based on adaptive models, i.e., they use a number of bits per symbol defined by probability estimates that are constantly being updated.

Table 1: Comparison of compression ratios and speed.

Method	Speed (seconds/Mpixel)	Compression
ImageZip	0.05	18%
IJPG decoder	0.16	NA
Stuffit	0.72	25%
JPEG + AC	0.13	12%

4 Evidence that the Solution Works

We currently have a fully functional implementation of this new compression method, and thanks to Snapfish, which allowed us to access JPEG files in their consumer photo database, our program has been already **tested in more than 1.5 million photos**! This was extremely important, because it enabled us to confirm that the technical requirements stated in Section 3.1 are being satisfied for the large variety of photos that we find in consumer applications. (See Section 6 for more comments on reliability.)

Figure 9 shows a typical distribution of compression ratios on sets of consumer photos. This particular histogram was obtained from 50,000 images randomly selected from the Snapfish database. Note that in nearly all the cases we get at least 13% extra compression, and while the average compression ratio is about 18%, many photos are compressed with larger ratios. Interestingly, the compression ratio averaged using the image size as weight is nearly the same as for individual images, showing that the same type of compression is obtained for small and large images.

Speed tests have shown that ImageZip has nearly symmetrical computational complexity, i.e., the conversion from JPEG to ImageZip and from ImageZip to JPEG take nearly the same time. The average conversion speed in the images tested for obtaining the graph in Figure 9 is about 15 Mpixels/second, for color images, on a 3 GHz Intel Xeon processor.

Table 1 shows a summary of the compression and speed comparisons, on a set of 50,000 photos of the Snapfish database.

5 Competitive Approaches

As explained before, the ImageZip solution is the result of many years of development of compression technologies at HP Labs, and each of its component techniques has been extensively tested and compared to competitive approaches. In this section we provide

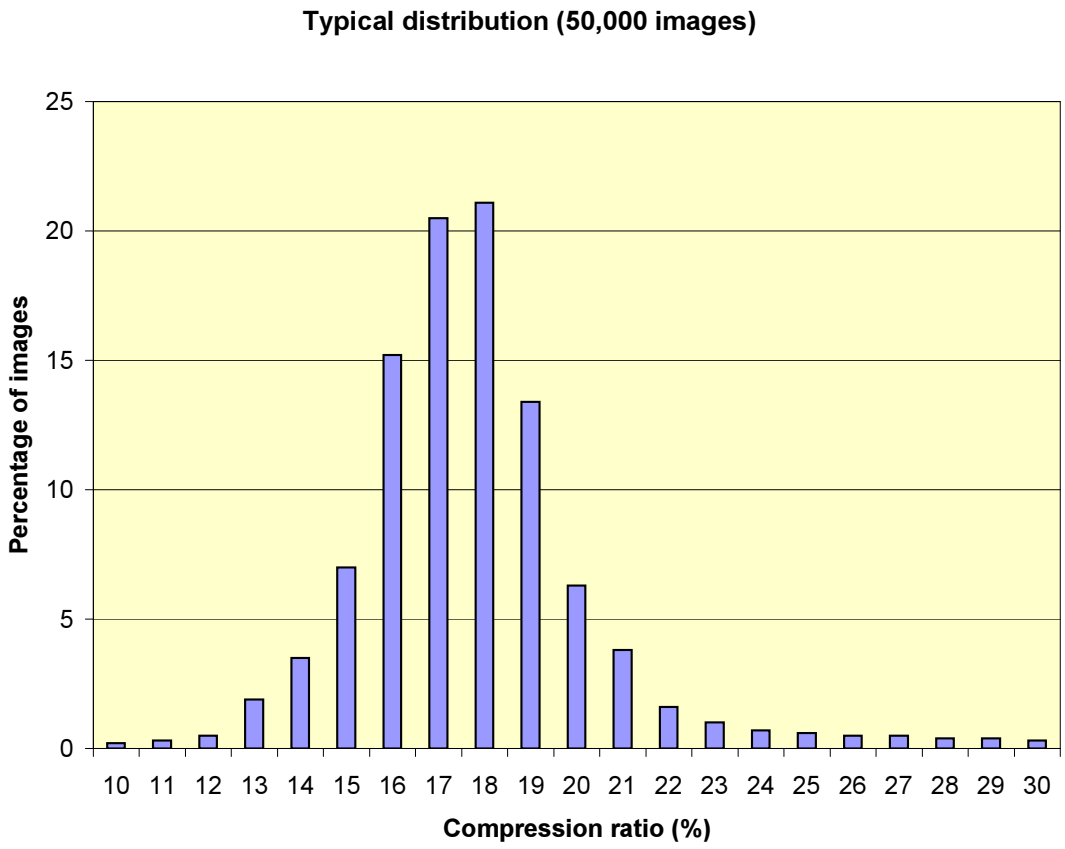


Figure 9: Histogram with typical distribution of extra compression ratios (i.e., over JPEG compression) obtained with ImageZip. Results obtained on 50,000 images randomly selected from a Snapfish database.

some more information on competitive approaches, for this specific application.

Since our objective and also the constraint was to create a solution for lossless compression of JPEG images, it only makes sense to compare against other solutions that also achieve the same objective. On the other hand, it does not make sense to compare against other image compression algorithms such as JPEG2000. In that sense, the only two other solutions that compete with ImageZip are:

1. JPEG with arithmetic coding, and
2. Stuffit (a commercial product).

JPEG with arithmetic coding achieves roughly 10–15% extra compression on an average, while Stuffit achieves about 22–28% compression. While ImageZip achieves 20% compression on average, it does so at a significantly lower complexity than Stuffit. In particular, ImageZip is 20 times as fast as Stuffit, and twice as fast as JPEG with arithmetic coding. This makes ImageZip a solution very suited for back-end compression for image storage databases.

6 Current Status

Due to the special need of also supporting JPEG photo files that violate the standard, a completely new parser of JPEG files was written for ImageZip. It supports all the standard baseline modes, including less common modes, like 4-color images. It does not support any files that are not in the baseline profile. The header information (EXIF, etc.) is also compressed in a lossless manner.

ImageZip's program was designed to be extremely reliable, and it continues to undergo reliability tests. At this stage, its implementation was tested in more than 1.5 million photos, of the Snapfish database. The compression program did not crashed in any of those images. On average, about 1.5% of the images were saved without further compression because they were too damaged, or in a format that is not supported.

The ImageZip solution has been delivered to HP Snapfish for potential use in back-end compression for their petabytes of customer JPEG images, which potentially can save Snapfish significant amount of storage costs. It has also been considered for licensing to cell phone carriers, which can save bandwidth in the transmission of photos from mobile phones.

7 Next Steps

There are many situations where it is very interesting to process photo images in the compressed domain, or use "shortcuts" for obtaining a more efficient form of image processing. As we can see in Figure 4, because it shares most of the "image compression pipeline" with JPEG, ImageZip can easily leverage nearly all the techniques developed for JPEG. We are currently working on adding new functionality to the ImageZip program, including image resizing and lossless rotation. In addition, while we found that the current implementation is quite fast, we are considering that to effectively work on millions of images, it is best to continue optimizing the code, and use high performance libraries, like Intel's IPP.

For instance, when resizing a JPEG image, it is not necessary to recover the full RGB image, and it is more efficient to do it in the YCrCb color space. Figure 10 shows a comparison of the processing stages required with and without integration of resizing and compression.[1] Note that the integrated version is much simpler, and thus can be significantly faster. Furthermore, by avoiding repeated resizing operations and rounding of the image pixel values, the integrated version can also produce images with better quality. The resizing program can also use new interpolation kernels [14], developed to simplify the determination of resizing filters that yield the best image quality, and that avoid some common artifacts created by low-complexity image downsizing.

References

[1] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inform. Theory,* vol. 23, pp. 337–343, 1977.

[2] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inform. Theory,* vol. 24, pp. 530–536, 1978.

[3] International Telecommunication Union (ITU), *Digital Compression and Coding of Continuous-Tone Still Image: Requirements and Guidelines*, ITU-T Recommendation T.81, Geneva, Switzerland, 1992 (JPEG/JFIF standard).

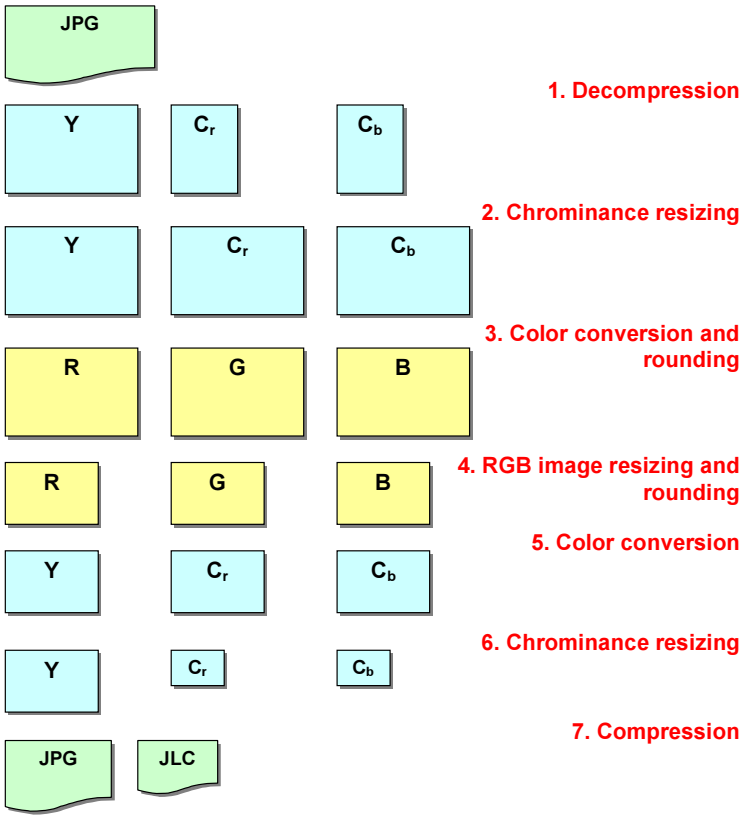
[4] N. Memon, *Adaptive Coding of DCT Coefficients by Golomb-Rice Codes,* Hewlett-Packard Laboratories Report, HPL–1998–146, Palo Alto, CA, 1998.

[5] M.J. Weinberger and G. Seroussi, *From LOCO-I to the JPEG-LS Standard,* Hewlett-Packard Laboratories Report, HPL–1999–3, Palo Alto, CA, Jan. 1999.

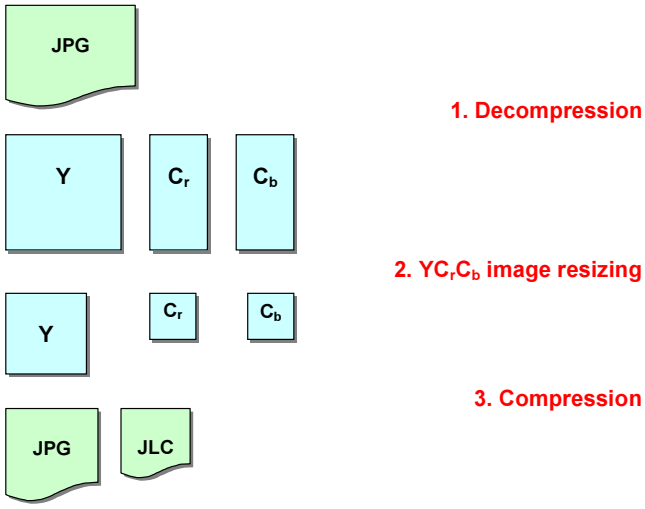
[6] D.S. Taubman and M.W. Marcellin, *JPEG 2000 Image Compression Fundamentals, Standards and Practice,* Kluwer Academic Publishers, Boston, 2002.

[7] D.S. Taubman and M.W. Marcellin, "JPEG2000: standard for interactive imaging," *Proc. IEEE,* Vol. 90, pp. 1336–1357, Aug. 2002.

[1] In this figure 'JLC' is the file format used by ImageZip.



(a) Independent image resizing and JPEG or ImageZip compression.



(b) Image resizing fully integrated with JPEG or ImageZip compression.

Figure 10: The integration of image resizing and compression can significantly reduce the overall computational complexity, and also improve the quality of the final images.

[8] K. Sayood (ed.), *Lossless Compression Handbook,* Academic Press, San Diego, CA, 2003.

[9] A. Said, *Introduction to Arithmetic Coding Theory and Practice,* Hewlett-Packard Laboratories Report, HPL–2004–76, Palo Alto, CA, April 2004.

[10] A. Said, *Comparative Analysis of Arithmetic Coding Computational Complexity,* Hewlett-Packard Laboratories Report, HPL–2004–75, Palo Alto, CA, April 2004.

[11] A. Said, *On the Reduction of Entropy Coding Complexity via Symbol Grouping: I - Redundancy Analysis and Optimal Alphabet Partition,* Hewlett-Packard Laboratories Report, HPL–2004–145, Palo Alto, CA, August 2004.

[12] Ed Lee, *The Image Storage and Management Dilemma,* InfoTrends Strategic Assessment, Weymouth, MA, Jan. 2007.

[13] A. Said, *Transform Coefficient Compression Using Multiple Scans,* US Patent 7,190,840 B2, March 2007 (filed Jan. 2002).

[14] A. Said, "A new class of filters for image interpolation and resizing," *IEEE Int. Conference on Image Processing,* San Antonio, TX, Sept. 2007.