FLIF: FREE LOSSLESS IMAGE FORMAT BASED ON MANIAC COMPRESSION

Jon Sneyers

Cloudinary Ltd.
jon@cloudinary.com

Pieter Wuille

Blockstream
pieter.wuille@gmail.com

ABSTRACT

We present a novel lossless image compression algorithm. It achieves better compression than popular lossless image formats like PNG and lossless JPEG 2000. Existing image formats have specific strengths and weaknesses: e.g. JPEG works well for photographs, PNG works well for line drawings or images with few distinct colors. For any type of image, our method performs as good or better (on average) than any of the existing image formats for lossless compression. Interlacing is improved compared to PNG, making the format suitable for progressive decoding and responsive web design.

Index Terms— Digital images, compression algorithms

1. INTRODUCTION

Lots of bandwidth and storage can be wasted by encoding images using a suboptimal file format. For instance, a particular image[1] might need about 46 MiB when stored uncompressed. If the user is sufficiently well informed, she will notice that it is a line drawing with few different colors, so she saves the image as a GIF file (1106 KiB) or a PNG file (804 KiB), or perhaps as a lossless WebP file (787 KiB). If the user is not so well informed, he might save the image as a PNG24 file (1.5 MiB), or much worse, as a lossy JPEG file (5.4 MiB), a lossy WebP file (2.4 MiB), or a lossless JPEG 2000 file (12 MiB).[2]

Lossy formats like JPEG (2000), WebP and BPG are great for photographs, but for other kinds of images, the compression artifacts can be visible. Image formats like GIF, PNG and lossless WebP work well for line drawings, but they're less suitable for large photographs. If 1) the image clearly belongs to one of these two categories (photograph / line drawing), 2) the user knows when to use which image format, 3) the user has the freedom to choose the format, and 4) a single quality setting suffices for all use cases of the image, then the existing formats are "good enough" in practice. However:

1) Many images are not photographs, nor line drawings, but something else or a combination of the two. For example, think of a poster that contains a photo combined with text elements, or a satelite image annotated with labels and arrows.

2) It is not easy to explain to non-technically oriented users when to use which image format (and which compression parameters). It would be more user-friendly and less error-prone to have only one format that "just works".

3) Many devices, applications, and web services only support one (or a few) image formats. As a result, sometimes users are forced to use suboptimal image formats.

4) Lossy compression by definition means that information is lost, and the sender of the image has to decide *how much* information gets lost. However, it is the receiver of the image who knows best what level of information loss (if any) is "acceptable" for their intended use of the image.

We believe that the FLIF image format proposed in this paper can improve the state of the art. New formats have been proposed in the past, but while improving the compression ratio, they kept the distinction between two "categories" of images largely intact: e.g. JPEG 2000, lossy WebP and BPG improve upon JPEG, while lossless WebP and PNG improve upon GIF. In practice, at least two image formats are used and needed: e.g. on the web, JPEG and PNG are currently standard. FLIF not just improves the compression ratio, but it also makes the old dichotomy obsolete: FLIF works well for photographs *and* for line drawings, and everything "in between".

2. COLOR TRANSFORMATIONS

We assume an RGB(A) source image. Perceptually, luma (i.e. the grayscale image) is more important than color (chroma). Lossy image compression aims to minimize perceptual error. Most lossy formats use a luma+chroma color transformation, with chroma subsampling and/or quantization. For lossless compression, color transforms are useful to decorrelate the color channels and to minimize perceptual error at intermediate steps of progressive decoding. FLIF uses the reversible YCoCg color transformation of lossless JPEG 2000 [1].

Bounds and Palette. Typically, an image only uses a small fragment of the full color space. We can improve compression by taking that into account. We record which values actually occur in each color channel. The smaller the range of values, the fewer bits are needed to encode a number in the range. For a singleton range no bits are needed at all.

Part of this research was performed while Jon Sneyers and Pieter Wuille were at the University of Leuven, Belgium, and while Jon Sneyers was granted a fellowship of FWO-Vlaanderen (Research Foundation - Flanders).

[1] This example refers to the 4000x4000 image `Circos.ppm` from Niels Fröhling's test set of sparse-color images.

[2] In case you are wondering: the FLIF file is 432 KiB.

		$\boldsymbol{TT}$	
	$\boldsymbol{TL}$	$\boldsymbol{T}$	$\boldsymbol{TR}$
$\boldsymbol{LL}$	$\boldsymbol{L}$	?	R
	$\boldsymbol{BL}$	B	$\boldsymbol{BR}$

Fig. 1. Pixels around an unknown pixel indicated by '?'.

Image formats like GIF and PNG can use a color palette to encode sparse-color images concisely. Their palette size is constrained since they encode pixels as (packed) bytes. FLIF supports arbitrary palette sizes and both 4-channel (YCoCgA) and 3-channel palettes (YCoCg), encoding the alpha channel separately if needed. The default maximum palette size is 512 — larger palettes rarely improve compression.

Color Buckets. The palette approach only covers images which use a very small fragment of the full color space (e.g. 2^{10} out of 2^{24} or 2^{32} possible colors). We propose a simple yet effective mechanism, called *color buckets*, that generalizes the idea of color palettes. It works as follows.

For each value of Y, we keep track of the Co values that occur for that value. As long as the number of distinct Co values for a given Y value is small, we maintain a list of discrete values (a "discrete bucket"); if that list becomes too large (according to some arbitrary threshold), then we replace it by an interval, storing only the lower and upper bounds (a "continuous bucket"). Next, for each combination of Y and Co, we record the Cg values in a similar way. To keep the total number of buckets reasonable, quantization is used and the mechanism is disabled for high bit-depth images.

3. IMAGE TRAVERSAL AND PIXEL PREDICTION

We have implemented two different methods to traverse the image. The first method is a simple scanline traversal. The second method (the default method) is a generalization of PNG's Adam7 interlacing; it could be called "Adam∞ interlacing". It allows progressive decoding.

In both methods, the differences between predicted pixel values and actual values are encoded. If the prediction is good, the differences are (close to) zero. The differences are signed. At first sight, their range is twice as large as the range of the original numbers. However, given the guess, only half of that range is valid. Furthermore, not all YCoCg triples correspond to valid RGB triples, further reducing the range. The color buckets (if available) are also used reduce the range and to snap the guessed value to a valid value: if the relevant color bucket is discrete, the guess gets rounded to the nearest value in the list; otherwise it gets clipped to the interval.

Scanline traversal (non-interlaced). The first traversal method is used in many image formats. The image is scanned line by line from top to bottom. Each line is scanned from left to right. When outputting a pixel, the value of the pixels above and to the left are already outputted, so those values can be used to compute a prediction that is also available at decode time (cf. Figure 1). As a prediction, we take the median of T (top), L (left), and $T + L - TL$ (gradient). This predictor is exactly the same as the one used in the FFV1 lossless video codec; it works well on smooth color gradients.

Adam∞ Interlacing. The aim of interlacing is to be able to progressively reconstruct a compressed image, perhaps even not loading the entire compressed stream if it is a large image and only a small preview is needed. FLIF uses a generalization of PNG's Adam7 interlacing. In each interlacing step, the number of pixels doubles.

The first step is simply one pixel: the pixel in the top-left corner. Then, in each interlacing step, either the number of rows doubles (a horizontal step), or the number of columns doubles (a vertical step). The final step is always a horizontal step, traversing all the odd-numbered rows of the image.

The pixels indicated in bold in Figure 1 are known at decode time. In a horizontal step, B is also known, while in a vertical step, R is known. We define three predictors:

1. The average of top and bottom $(T + B)/2$ (horizontal step) or left and right $(L + R)/2$ (vertical step);
2. The median of: the above average, the top-left gradient $T+L-TL$, and the gradient $L+B-BL$ or $T+R-TR$;
3. The median of 3 known neighbors (L, T, and B or R).

Channel Ordering. So far, we have only considered a single color channel. When using scanline traversal, we simply process all the channels one by one: first the alpha channel (if there is one), then the Y channel, then the Co channel, and finally the Cg channel. If the alpha value of a pixel is zero, then the pixel is completely transparent and its values in the other channels are irrelevant.[3] For this reason, the alpha channel is outputted first and the other channels are only encoded where the alpha value is nonzero.[4]

In interlaced traversal, the different channels are interleaved in such a way that the alpha and Y channels get encoded earlier at higher resolutions than the chroma channels. This implies that the preview images obtained by progressive decoding are automatically chroma subsampled.

4. ENTROPY CODING: MANIAC

Arithmetic coding [2], also known as range encoding, is a form of entropy coding based on a probability model for the encoded bits. Inspired by the FFV1 codec [3], we use a variant of context-adaptive binary arithmetic coding (CABAC).

We call our entropy encoding method "meta-adaptive near-zero integer arithmetic coding" (MANIAC). It is *meta*-adaptive since *the context model itself* is adapted to the data.

[3] At least, that is our opinion. For those with a different opinion, there is an encoder option (`-K`) to keep the color of fully invisible pixels intact.

[4] Still, those "irrelevant" pixels need to get Y, Co and Cg values, because they can be needed to predict other pixels. Interpolation is used.

4.1. Context-Adaptive Binary Arithmetic Coding

In CABAC, the probability model is adaptive. Initially we start with an arbitrary chance (e.g. 50%) for each bit. After processing a bit, the chance is updated. The aim is to learn the actual distribution from the observed past to hopefully better predict the future. If we have additional context information, we can use a different probability in each context. Correlation between the context and the bits leads to more accurate probabilities and better compression. Obviously all context information has to be available at decode time.

To encode integers in a near-zero interval, an exponent-mantissa representation is used, with a different context for each bit position. Our binarization is based on that of FFV1, with some improvements. First we output a single bit to indicate if $x = 0$. Then we output the sign of x, followed by the exponent, i.e. the number of bits $e = \lceil \log(|x|) \rceil$ needed to represent $|x|$. The number e is encoded in *unary notation* as e 1 bits followed by one 0 bit. We use different contexts for each of these exponent bits. Finally we output the mantissa bits; the leading 1 is omitted. Redundant bits are omitted.[5]

4.2. Context Model

In FFV1, quantized pixel differences are used as context information. Referring to Figure 1, the differences $L - TL$, $TL - T$, $T - TR$, $LL - L$, $TT - T$ are computed and quantized (using a logarithmic scale). For every combination of these 5 properties, a different context is defined. In total, 7563 different contexts (per color channel) can be used.

In FLIF, the contexts depend on the image traversal order. **Scanline traversal:** we use the same 5 differences as in FFV1 (but without quantization), and we also use the following extra properties: the prediction itself (the median of 3 values); a number indicating which of those 3 values was used; and the pixel value(s) in the 0, 1, 2 or 3 previously encoded channels. **Interlaced traversal:** the above 7 to 10 properties are also used, except that instead of $L - TL$ and $TL - T$, we use $L - (TL + BL)/2$ and $T - (TL + TR)/2$, and instead of $T - TR$ we use either $B - (BL + BR)/2$ (horizontal step) or $R - (TR + BR)/2)$ (vertical step). Additionally, the following properties are used: the difference between the two adjacent pixels from the previous interlacing step ($T - B$ in horizontal steps, $L - R$ in vertical steps), and for the chroma channels: the difference between the actual luma pixel value and the one predicted by the 'average' predictor.

[5]In general, suppose the value x to be encoded is in the interval $[a, b]$. If the interval does not contain zero, the zero-bit is omitted. If the signs of a and b are the same, the sign-bit is omitted. There can be a lower bound on the exponent, e.g. if $x \in [32, 80]$ then we know $e \geq 5$ so the first 5 exponent bits can be omitted. If the exponent has the largest possible value (e.g. 6 if $x \in [32, 80]$), then it does not need to be followed by a 0 bit. Finally, in the mantissa, some bits may be implied, e.g. if $x \in [32, 80]$ and $e = 6$ (so $x \in [64, 80]$), then the most significant bit of the mantissa (the one for 32) cannot be 1 since that would give a lower bound $x \geq 92$.

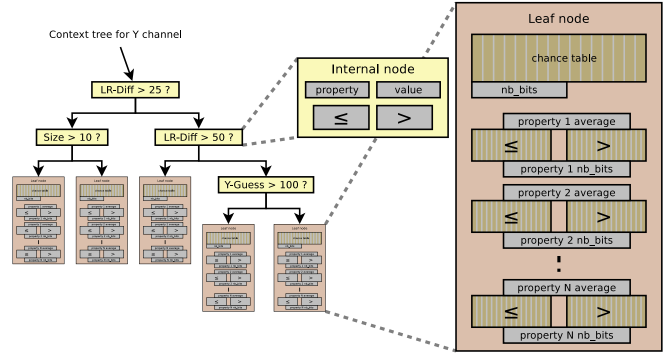

Fig. 2. MANIAC tree structure.

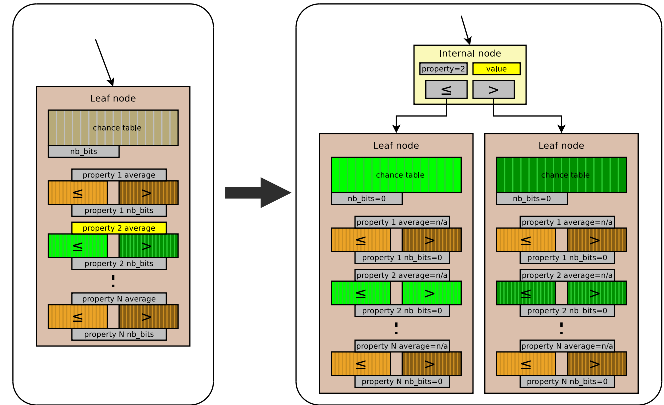

Fig. 3. Growing a MANIAC tree: one step.

4.3. MANIAC Tree Learning

When using static contexts like in FFV1, where the number of contexts is the product of domain sizes of each property, quantization has to be used in order to reduce the number of contexts. It is hard to get the number of contexts "just right": using too many contexts hurts compression because context adaptation is limited (few pixels per context); but with too few contexts, compression also suffers since pixels with different properties end up in the same context. Also, when the contexts are defined statically, a lot of the contexts are actually not used at all since the corresponding combination of properties simply does not occur in the source image.

By contrast, we propose a dynamic data structure as a context model. It is essentially a decision tree (actually one tree per channel), grown during encoding. Figure 2 shows an example MANIAC tree. Every internal (non-leaf) node has a test condition: an inequality comparing one of the context properties to a value. The child nodes correspond to the two test branches. During encoding, every leaf node contains one *actual* context (array of chances) and two *virtual* contexts per property. At decode time only the actual contexts are used.

For each encoded value, the decision tree is traversed until a leaf node is reached. Initially, the actual context is used to

Corpus			Lossless formats									JPEG*	
		(bit depth)	FLIF	FLIF*	WebP	BPG	PNG	PNG*	JP2*	JXR	JLS	100%	90%
Natural (photo)	[4]	8	1.002	**1.000**	1.234	1.318	1.480	2.108	1.253	1.676	1.242	1.054	0.302
	[4]	16	1.017	**1.000**	/	/	1.414	1.502	1.012	2.011	1.111	/	/
	[5]	8	1.032	**1.000**	1.099	1.163	1.429	1.664	1.097	1.248	1.500	1.017	0.302
	[6]	8	1.003	**1.000**	1.040	1.081	1.282	1.441	1.074	1.168	1.225	0.980	0.263
	[7]	8	1.032	**1.000**	1.098	1.178	1.388	1.680	1.117	1.267	1.305	1.023	0.275
	[8]	8	1.001	**1.000**	1.059	1.159	1.139	1.368	1.078	1.294	1.064	1.152	0.382
	[8]	12	1.009	**1.000**	/	1.854	2.053	2.378	2.895	5.023	2.954	/	/
Artificial	[9]	8	1.039	**1.000**	1.212	1.145	1.403	1.609	1.436	1.803	1.220	1.193	0.233
	[10]	8	**1.000**	1.095	1.371	1.649	1.880	2.478	4.191	7.619	3.572	5.058	2.322
	[11]	8	**1.000**	1.037	1.982	4.408	2.619	2.972	10.31	33.28	33.12	14.87	9.170
	[12]	8	1.106	1.184	**1.000**	2.184	1.298	1.674	3.144	3.886	2.995	3.186	1.155
	[13]	8	**1.000**	1.049	1.676	1.734	2.203	2.769	4.578	10.35	4.371	5.787	2.987

* : Format supports progressive decoding (interlacing).
/ : Unsupported bit depth.
Numbers are scaled so the best (smallest) lossless format corresponds to 1.

Fig. 4. Compressed corpus sizes using various image formats.

output the value, and a *cost estimate* (number of bits for the compressed output) is updated. For each of the properties, each leaf node maintains a running average of the property values encountered at that leaf; one virtual context is used for values below the average, the other is used for higher-than-average values. For each property we select the virtual context accordingly, and update its chances and cost estimate.

These cost estimates indicate which properties are most significant. If a property is irrelevant, then the sum of the costs for its two virtual contexts will be the same or higher than that of the actual context. If however a property is relevant, then using two different contexts depending on the value for that property will result in better compression. We compare the cost of the "best" pair of virtual contexts in a given leaf node with the cost of the actual context. If the cost difference gets larger than some fixed threshold, the leaf node becomes a decision node testing the "best" property. Figure 3 illustrates this. The MANIAC tree that we end up with is not necessarily optimal; future encoders can use other algorithms since the structure of the tree is part of the encoded bitstream.

MANIAC trees have three main advantages compared to using a fixed context array: 1) There is no need to used quantized property values, so we can distinguish near-identical property values; 2) Properties are only actually used if they contribute to better compression for the specific image; 3) The context tree scales with the image: for large, complex images, more contexts will be used than for small, simple images.

5. EXPERIMENTAL EVALUATION

We have evaluated (the reference encoder of) FLIF by comparing its compression density to that of other lossless image compression algorithms (PNG [14], JPEG 2000 [1], JPEG XR [15], JPEG-LS [16], WebP [17], BPG [18]) and to lossy JPEG [19] at maximum quality[6] and at 90% quality. To optimize non-interlaced PNG files, we used ZopfliPNG [20]; for interlaced PNGs, we used OptiPNG [21]. For BPG we used the options `-m 9 -e jctvc`; for WebP we used `-m 6 -q 100`. For the other formats we used default lossless options.

Figure 4 shows the results; see [22] for more details. On one corpus (geographic maps), WebP was the best format. On the other corpuses, FLIF won (sometimes very clearly).

In terms of encoding/decoding speed, (our implementation of) FLIF is somewhat slower than the other formats. This is explained in part by the inherent computational complexity of the algorithm, and in part by our implementation lacking low-level (hardware-specific) optimization. It is however fast enough for most practical applications.

The reference FLIF encoder [23, 24] is released under the terms of the GNU LGPL version 3 or later; the reference decoder is available under the Apache 2.0 license.

6. FLIF AND RESPONSIVE IMAGES

Lossy image compression is useful when storage or bandwidth are limited. Arguably, storage is becoming relatively ubiquitous, while bandwidth conditions have become increasingly variable (both in speed and price).

Responsive Web Design (RWD) aims to deal with various viewing devices and bandwidth conditions. The typical approach to responsive images is a mostly server-side solution where for each image, multiple files are created at various resolutions and quality settings.

We propose an alternative client-side approach, based on a single FLIF file per image. Progressive decoding (i.e. partial downloading) allows fine-grained control over the desired trade-off between image quality and transfer time and cost. This trade-off depends on the receiver and their intention: a quick preview, a closer look, a high-quality print, or further image processing without cumulative degradation.

7. CONCLUSION AND FUTURE WORK

FLIF is good at losslessly compressing various kinds of images, not just photographs. Based on Adam∞ interlacing and YCoCg interleaving, its advanced progressive decoding reduces the need for lossy compression. We hope that FLIF can be a step in the direction of a 'universal' image format.

MANIAC can be generalized to general-purpose compression. The underlying idea is to use machine learning to determine the most relevant features to construct the context model. Many machine learning techniques could be used; we have used relatively simple decision trees, but any kind of classifier could be used. Learning does not necessarily have to be fast — encoding time is usually much less important than decoding time. The only requirement is that the learned object (e.g. the decision tree) can be stored concisely and that it can be reconstructed quickly during decoding.

[6]Even at 100% quality, JPEG is lossy since its YCbCr transform reduces the number of possible colors from $2^{24} = 16\,777\,216$ to only ~ 4 million.

8. REFERENCES

[1] Joint Photographic Experts Group (JPEG), "JPEG 2000 standard, ISO/IEC 15444, ITU-T Recommendation T.800," 2004, http://www.jpeg.org/jpeg2000 (implementation: OpenJPEG v2.1.0).

[2] Ian H. Witten, Radford M. Neal, and John G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, pp. 520–540, June 1987.

[3] Michael Niedermayer, "Description of the FFV1 video codec," 2004, http://www.ffmpeg.org/~michael/ffv1.html.

[4] Nicola Asuni, "TESTIMAGES, category SAMPLING/," 2014, 40 photographs, 2448×2448 pixels, color, 8/16 bits per channel. See [25].

[5] Rich Franzen and Eastman Kodak, "Kodak Lossless True Color Image Suite," 2013, 24 photographs, 768×512 pixels, color, 8 bits per channel. http://r0k.us/graphics/kodak.

[6] Timothy B. Terriberry et al., "Downsampled very high resolution JPEGs from Wikipedia," 2015, 49 photographs, various resolutions, color, 8 bits per channel. See [22], "wikipedia-photos".

[7] Thomas Richter et al., "Test material for the JPEG Grand Image Compression Challenge at ICIP 2016," 2015, 14 photographs, various resolutions, color, 8 bits per channel. http://jpeg.org/static/icip_challenge.zip.

[8] Rainer Köster and Jürgen Abel, "Lukas 2D medical image corpus (8 bit and 16 bit)," 2015, 20 radiographs, various resolutions, grayscale, 8/16 per pixel. http://www.data-compression.info/Corpora/LukasCorpus.

[9] greentunic, "Fractal Pack 4," 2010, 20 images of fractal-based art, mostly 1600×1100, color, 8 bits per channel. http://greentunic.deviantart.com/art/Fractal-Pack-4-150661301.

[10] Zach Weinersmith, "Saturday Morning Breakfast Cereal (SMBC)," 2013, 20 digital cartoons, various resolutions, color, 8 bits per channel. http://www.smbc-comics.com/index.php?id=3000 until id=3020.

[11] Nicola Asuni, "TESTIMAGES, category PATTERN," 2014, 433 geometric patterns, 4096×2560 pixels, grayscale, 8 bits per pixel. See [25].

[12] OpenStreetMap, "Various maps of brussels," 2015, 44 geographic images, 1366×768, color, 8 bits per channel. http://www.openstreetmap.org, See [22], "openstreetmap".

[13] Pearson Scott Foresman, "Educational illustrations," 2015, 20 line art images, various resolutions, grayscale or color, 8 bits per channel. See [22], "PSF-line-drawings".

[14] Thomas Boutell et al., "Portable Network Graphics (PNG) specification, RFC 2083, ISO/IEC 15948:2004," October 1996, http://www.libpng.org/pub/png/.

[15] Joint Photographic Experts Group (JPEG), "JPEG XR image coding system, ISO/IEC 29199, ITU-T Recommendation T.832," 2012, http://www.jpeg.org/jpegxr (implementation: Microsoft JXR encoder v1.0).

[16] Joint Photographic Experts Group (JPEG), "JPEG-LS standard, ITU-T Recommendation T.87," 1998, http://www.jpeg.org/jpeg/jpegls.html.

[17] Google Inc., "WebP: A new image format for the web," October 2015, http://developers.google.com/speed/webp (latest version on git as of October 2015).

[18] Fabrice Bellard, "Better Portable Graphics (BPG), version 0.9.6," September 2015, http://bellard.org/bpg.

[19] Joint Photographic Experts Group (JPEG), "JPEG standard, ISO/IEC IS 10918-1, ITU-T Recommendation T.81," 1992, http://www.jpeg.org/jpeg.

[20] Lode Vandevenne and Jyrki Alakuijala, "ZopfliPNG," 2015, http://github.com/google/zopfli.

[21] Cosmin Truta et al., "OptiPNG version 0.7.5," 2014, http://optipng.sourceforge.net.

[22] Jon Sneyers et al., "Lossless still and animated image compression benchmarks," 2016, http://github.com/FLIF-hub/benchmarks.

[23] Jon Sneyers, "Free Lossless Image Format homepage," 2016, http://flif.info.

[24] Jon Sneyers, Pieter Wuille, et al., "FLIF reference implementation," 2016, http://github.com/FLIF-hub/FLIF.

[25] A. Giachetti N. Asuni, "TESTIMAGES: a large-scale archive for testing visual devices and basic image processing algorithms," in *Smart Tools & Apps for Graphics Conference*, 2014, http://testimages.org.