Hindawi Publishing Corporation
EURASIP Journal on Image and Video Processing
Volume 2007, Article ID 85385, 8 pages
doi:10.1155/2007/85385

Research Article

JPEG2000 Compatible Lossless Coding of Floating-Point Data

Bryan E. Usevitch

Department of Electrical and Computer Engineering, University of Texas at El Paso, El Paso, TX 79968-0523, USA

Received 14 August 2006; Revised 13 December 2006; Accepted 22 December 2006

Recommended by James E. Fowler

Many scientific applications require that image data be stored in floating-point format due to the large dynamic range of the data. These applications pose a problem if the data needs to be compressed since modern image compression standards, such as JPEG2000, are only defined to operate on fixed-point or integer data. This paper proposes straightforward extensions to the JPEG2000 image compression standard which allow for the efficient coding of floating-point data. These extensions maintain desirable properties of JPEG2000, such as lossless and rate distortion optimal lossy decompression from the same coded bit stream, scalable embedded bit streams, error resilience, and implementation on low-memory hardware. Although the proposed methods can be used for both lossy and lossless compression, the discussion in this paper focuses on, and the test results are limited to, the lossless case. Test results on real image data show that the proposed lossless methods have raw compression performance that is competitive with, and sometime exceeds, current state-of-the-art methods.

1. INTRODUCTION

Floating-point image compression has not received a lot of attention by the compression community. This is evidenced by the fact that modern image compression texts (see [1–3]) have no direct discussion of floating-point compression methods. Perhaps the reason floating-point compression has not been emphasized is that most digital-image data is acquired through the sampling and quantization of analog data, and thus starts as fixed-point data. However, there are applications, especially in the scientific community, where floating-point compression would be immediately applicable. Examples include post processed data, such as atmospherically corrected satellite data, and scientific simulation data, where the data begins and ends as floating-point data. Even though these floating-point application areas may be smaller than the fixed-point areas, the JPEG2000 standards body has created a committee (ISO/IEC JTC1/SC29/WG1 Part 10 committee) to propose enhancements to the JPEG2000 standard which specifically target scientific (including floating-point) compression applications.

Traditionally there have been two main methods for compressing floating-point data. The first method is to initially quantize the floating-point data into fixed-point data prior to compression. This method has the advantage that the plethora of fixed-point compression algorithms described in the textbooks and literature can then be used to compress the quantized data. A main disadvantage of this method is that it causes irreversible data loss. This data loss may not be acceptable from either a technological or political standpoint (we just spent 10 million dollars to acquire the data and you want to do what?). In fact, a main reason for using a floating-point representation in the first place is to avoid any sort of data loss.

The second traditional method for compression of floating-point data is to use a general lossless method such as gzip on the integer representation of the floating-point data. Candidate compressors include gzip, bzip2, rice-golomb, and JPEG2000 lossless. Note that this method implicitly operates on the integer representation of the floating-point data (4 bytes in the case of single precision), and not the actual floating-point values themselves. This is an important but subtle point. For example, quantizing a floating-point data set with widely varying exponents by keeping the upper N bits of each of the integer representations of the floating-point values would not result in a distortion optimal quantization. A main advantage of this second method for floating-point compression is that it incurs no data loss. However, the second method has the big disadvantage of loss of data flexibility. For example, partially decompressing to get a lossy representation at a target bit rate, or partially decompressing only one spatial region of the compressed data either very difficult or impossible.

The aforementioned disadvantages of these two traditional methods are overcome by the methods proposed in this paper, since these proposed methods are enhancements to JPEG2000 to allow for compression of floating-point data. Since they are based on JPEG2000, lossy and lossless decompression can be achieved from the same compressed data stream. Furthermore, all the other compressed data flexibility properties of JPEG2000, such as quality, resolution, and spatial scaling, are preserved. Prior to discussing these properties further, we first mention other recently published schemes for floating-point compression.

Several lossless floating-point methods have been recently proposed in the literature. Engelson et al. [4] proposed a method which uses one-dimensional polynomial prediction, where the prediction residual was represented as an integer so as to result in lossless compression. Ghido [5] proposes a method wherein the floating-point data is first losslessly mapped into an integer representation. This representation is transformed to improve coding efficiency and then coded. The method was devised for coding IEEE audio and is thus a one-dimensional method. The approach of Ratanaworabhan et al. [6] is different in that it does not convert the floating-point values to integers, but rather works directly on the integer representation of the floating-point values. It uses a predictor with a special hash function to code the prediction residuals. This method is also a one-dimensional method. The method of Lindstrom and Isenburg [7] is also different in that it is not a one-dimensional but rather a two- or three-dimensional method. Specifically, it uses a Lorenzo predictor that can do two- or three-dimensional prediction and thus can take better advantage of spatial correlations in images and data cubes. The prediction residuals are represented using an integer format in order to be lossless. A common theme from all these proposed methods is that in order to be lossless, floating-point values at some stage of the coding process must be represented as integers. This theme is continued in this paper where floating-point values are represented in what is here called *extended integer* (EI) format.

The floating-point compression methods proposed in this paper have several advantages over the previously mentioned methods. Most of these advantages are inherited directly from the original fixed-point JPEG2000 algorithm, since once the floating-point values are converted to EI format the coding process is nearly identical. However, some minor modifications need to be made to accommodate the EI format and to improve coding efficiency. These modifications include extending the maximum number of possible bits planes (beyond the current limit of 38 in JPEG2000), adding one or two additional coding passes every several bit planes, and introducing a lossless wavelet transform that handles the EI format. Note that these modifications are why the paper title refers to the proposed methods as “JPEG compatible” instead of “JPEG compliant,” since they cannot be run in current JPEG2000 Part 1 and Part 2 compliant codecs.

The first advantage of the proposed methods is that lossy and lossless decompression can be obtained from only one coded representation. Thus the end user does not have to choose between either a lossy or lossless representation at the outset of coding. Also, the amount of loss for lossy decoding can be chosen at decoding time and can be varied to whatever amount desired. Finally, since the codestream is embedded, the resulting coded representation is rate distortion optimal for the given number of bits decoded. Other floating-point compression algorithms in the literature do not offer this combined lossy/lossless decompression capability. For example, the method of [7] does offer either lossy or lossless compression, but the amount of loss has to be set at compression time and the resulting truncated data stream is not guaranteed to be a rate distortion optimal representation. One could attempt to mimic the lossy/lossless decompression of the proposed methods by applying the fixed-point JPEG2000 method directly to the 4-byte integer representation (for single precision) floating-point data. However, the lossy partial decompression of this data would not be rate distortion optimal (and in fact would be really bad for data with widely varying exponents) since the JPEG2000 method assumes a linear representation for its rate distortion computations, and the mapping from floating-point value to 4-byte integer is nonlinear.

The proposed methods also maintain the quality, resolution, and spatial scaling properties found in JPEG2000. For example, spatial scaling is made possible through the codeblock structure of the coding algorithm and allows for random spatial access to and partial decompression of only selected portions of the data. This ability would be especially advantageous in working with large data sets often found in scientific applications. The other floating-point compression methods are prediction-based and do not allow for the same level of random access to the data. As another example, resolution or size scaling would allow a partial decompression of the data to a much smaller size, such as a thumbnail image. This, coupled with spatial scaling, would allow for searching, zooming, and decompressing only areas of interest from the data. Note that all of these scalings require no extra coding or decoding, but can be accomplished with only a low complexity reordering of the data.

The proposed methods also offer a measure of error resilience. This is because codestreams for each codeblock are self-contained. Specifically, the arithmetic encoder is reset at the beginning of coding each codeblock, and the codeblock codestream ends with an identifiable marker (which cannot be reproduced by the data). Thus errors in one codeblock would not be propagated to subsequent codeblocks (unless of course the error occurs in the end of codeblock marker). Other floating-point methods are based on some form of prediction and thus will have a greater tendency to propagate errors.

The proposed methods are also amenable to implementation on low memory machines. This is because codeblock sizes are small (16×16 blocks are used for experiments in this paper, and the maximum size for the JPEG2000 standard is 1024×1024), and the wavelet transform can be implemented incrementally (see [3, Section 17.4]). This means that only a portion of the total data set needs to be in memory at any particular time, resulting in modest RAM requirements while still allowing for the compression of large data sets.

The proposed methods also follow the same coding path as used in JPEG2000. Specifically, the coding passes, context computations, bit scanning, lifting for wavelet transform, and arithmetic encoder are all identical to those used in JPEG2000. While this fact may not be important to end users, it is very beneficial to persons or companies implementing the proposed algorithms. It means that large portions of existing hardware and software developed for JPEG2000 can be used to implement the new methods, resulting in considerable design savings. Finally, it should be noted that in addition to much greater compressed data flexibility, the proposed methods also give good raw lossless compression performance. This will be demonstrated in Section 6.

One may ask if all the increased compressed data stream flexibility is necessary or important? Ultimately that question will be answered by the end user, and to date the end user has not been given a lot of options to experiment with. However, it should be noted that the ISO JPEG committee felt that compressed data flexibility was sufficiently important as to create an entirely new international image compression standard, namely, JPEG2000 to supplement the existing and widely used JPEG standard. The nontrivial nature of this task lends credence to the importance of compressed data flexibility.

The remainder of the paper proceeds as follows. Section 2 discusses floating-point formats and the representation of floating-point numbers using the EI format. Section 3 uses the EI format to derive a lossless, floating-point wavelet transform. Section 4 proposes modifications to JPEG2000 bit-plane coding which allow for the efficient coding of EIs. Section 5 proposes new context determination methods to be used in the bit-plane coding and entropy coding of floating-point data. Section 6 gives compression performance results and Section 7 gives conclusions. Also note that the paper has the following limitation. Even though the proposed methods apply equally well to lossy and lossless compression, the discussion in this paper focuses on, and the test results are limited to, the lossless case. This is also the case most emphasized in the literature on floating-point compression.

2. FLOATING POINT REPRESENTATIONS

The proposed extensions in this paper apply to binary floating-point representations in general, but will be explained and illustrated using 32-bit IEEE floating-point format. The 32-bit IEEE floating-point format is shown in Figure 1 and represents numbers as

$$(-1)^s \times 2^{e-127} \times (01.f) \quad \text{if } 0 < e < 255, \tag{1}$$

where s is the sign (0 for positive), e is an offset-by-127 exponent, and f is the mantissa fraction [8]. Floating-point values can be conceptualized as "extended integers" (EI) as shown in Figure 2. The number of bits required to represent an arbitrary set of floating-point numbers using the EI format is 278, which includes 254 bits for standard float values

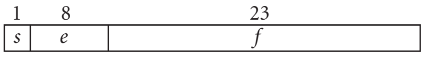

FIGURE 1: 32-bit IEEE floating-point representation showing the sign bit, 8 exponent bits, and 24 mantissa bits (the most significant 1 bit is implied).

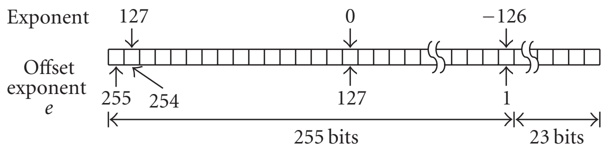

FIGURE 2: Extended integer representation of 32-bit IEEE floating-point format showing the 278-bit locations in which the 24 consecutive mantissa bits can be located.

($0 < e < 255$), 1 bit for exceptions ($e = 255$), and 23 bits for trailing fraction bits when $e = 0$.

A straightforward way to compress floating-point data would be to apply the JPEG2000 bit-plane coding algorithms directly to the EI format (neglecting for a moment the issue of the wavelet transform). The main modification needed would be to increase the maximum allowable number of bit planes, which is defined to be 38 in the current standard. The greatest advantage of this approach is that it preserves all the properties of JPEG2000, including highly scalable embedded bit streams and rate distortion optimality. The main disadvantage in directly applying the JPEG2000 bit-plane coding to the EI format is that it results in low coding efficiency. For example, consider the case of lossless coding of floating-point data using the EI format. Given that the number of bits in the EI format, 278, is approximately 9 times bigger than 32, and that the best lossless compression is about 2 : 1 or 3 : 1, compression could actually result in an expansion of approximately 3 to 4 for large dynamic range data. Methods given in Section 4 show how to reduce this bit-plane coding inefficiency.

3. LOSSLESS FLOATING POINT WAVELET TRANSFORM

This section derives a method for performing a lossless or reversible wavelet transform using floating-point data. This lossless wavelet transform is necessary if lossless JPEG2000 compression with resolution scaling is desired. Note that the wavelet transform is not a necessary step in JPEG2000 compression, but it does allow for resolution scaling and potential added efficiency through decorrelating the input coefficients.

The main disadvantage of the wavelet transform in the floating-point case is that it generally increases the dynamic range of the mantissa of each floating-point number. To see this note that before wavelet transforming, the EI representation of each coefficient can reside anywhere within the 278 possible bit locations. However, each EI is constrained in that its nonzero bits, the mantissa bits, must occupy a run of at most 24 consecutive bit locations. This fact will be used later to improve the efficiency of bit-plane coding these EI values.

Now consider the filter operation of the wavelet transform as an inner product:

$$w = \sum_{i=0}^{N-1} h_i c_i, \tag{2}$$

where h_i are the filter coefficients of a length N filter and c_i are the floating-point data coefficients. Since c_i and h_i both have mantissas of 24 bits, the resulting product term $h_i c_i$ will in general have a mantissa of length greater than 24 bits (possibly as large as 48 bits). In addition, since each of h_i and c_i can have greatly varying exponents, the sum w of these inner products has an EI representation that is much greater than a run of 24 bits. Simply, truncating the wavelet coefficient mantissa back to 24 bits, as would be done in standard floating-point arithmetic, would result in a wavelet transform that is not reversible (lossless). Thus a lossless wavelet transform cannot be derived using standard floating-point multiplication.

A solution to the lossless wavelet transform can be derived by recognizing that the EI format represents *integers*, albeit large ones. Thus applying the lossless integer wavelet transform from the original JPEG2000 standard, making allowances for the larger 278-bit integer size, results in a lossless wavelet transform for floating-point data. This proposed lossless floating-point wavelet transform has two main disadvantages. First, coding efficiency is potentially decreased since the coding of each wavelet coefficient can potentially require the coding of more than 24 mantissa bits. Second, because of the varying exponents of the original data, each wavelet coefficient will generally have a mantissa of varying bit lengths. Thus, the coder will have to send side information to the decoder to indicate when to stop coding of each coefficient because of these varying bit lengths. Note that the problem of varying mantissa lengths is not an issue in untransformed data since all of these coefficients have a set mantissa length of 24 bits. For these reasons, this paper considers the coding of both wavelet transformed and untransformed floating-point data. The coding of untransformed data has precedent in the original JPEG2000 standard in 2 places: (1) the coding of the LL band of the wavelet transformed data, and (2) the compression of untransformed bilevel image data [3]. Note also that eliminating the wavelet transform eliminates only one aspect of flexibility of the coded bit stream, namely, resolution or size scaling.

We close this section by commenting on the relative improvement in coding performance due to the wavelet transform in the fixed-point and floating-point compression cases. The distribution of wavelet transformed coefficients in a particular frequency band tends to have a Laplacian distribution [9, 10]. Thus a large percentage of the coefficients in the band have relatively small values. Small values for fixed-point representation directly translates into a small number of bit planes required to code the coefficient in a lossless manner. Thus the wavelet transform has the ability to greatly enhance the coding performance in the fixed-point case. In contrast, a small coefficient in floating-point representation does not necessarily translate into a smaller number of bit planes required for coding. Rather it just means that the initial bit-plane coded starts further down in the coding process. Thus, the number of bit planes required to code the smaller coefficient could actually increase due to the wavelet transform. As a result, the wavelet transform is not as beneficial to coding in the floating-point case as it is in the fixed-point case. This bit expansion can be ameliorated somewhat by choosing the truncation floor of the EI representation to be the smallest bit plane of the original data that has a nonzero value (for data without denormalized values the truncation floor would be 23 bits less than the smallest exponent of the original data).

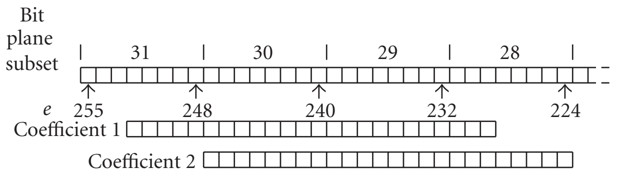

FIGURE 3: An example subset division where the number of bit planes per subset is $z = 8$. Coefficient 1 is eligible in subsets 31–28 while coefficient 2 is eligible in subsets 30–28.

4. BIT-PLANE CODING

The proposed algorithms in this paper use the concept of bit-plane coding the EI format with some modifications made to increase efficiency. The first modification deals with the number of coefficients that are coded in each bit plane. In standard JPEG2000 bit-plane coding, each bit from every coefficient is coded in each bit plane, starting with the most significant bit plane. This approach is inefficient for floating-point data since for any particular coefficient, the run of mantissa bits will occupy only a fraction of the 278 bits of the EI representation (24 of the 278 bits for untransformed data, and generally more bits for the wavelet transformed data). Define the active bits of the EI representation as the 24 mantissa bits for untransformed data, and the run of bits bounded the highest and lowest bit-plane-1 values for the wavelet transformed data. Therefore untransformed data has active bits in less than 9% ($24/278 \times 100$) of the total bit planes. Thus it is expected for floating-point data, having a large dynamic range, that only a subset of the coefficients will be coded in a particular bit plane. The first proposed modification is to divide the bit planes into subsets, similar to what EZW does using zero-trees and SPIHT does using set partitions. A coefficient is said to be *eligible* in a particular subset if it has active bits in any of the bit planes of that subset (see Figure 3).

A proposed subset division for defining coefficient eligibility is shown in Figure 3, which corresponds to a uniform division of the bit planes. For this division, a coefficient w_i is eligible in subset s if

$$\left\lfloor \frac{e_i}{z} \right\rfloor \geq s \geq \left\lfloor \frac{e_i - (N-1)}{z} \right\rfloor, \tag{3}$$

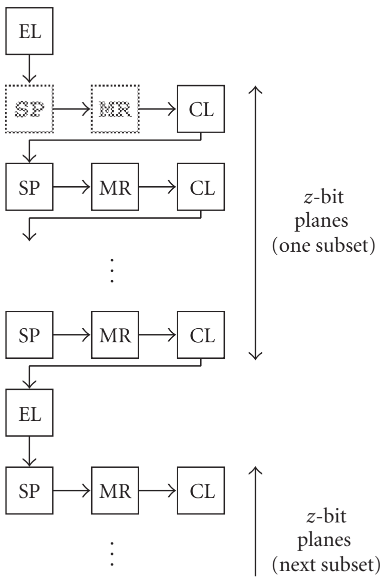

FIGURE 4: The bit-plane encoding passes showing the inclusion of the eligibility (EL) coding pass.

where e_i is the coefficient's offset exponent, z is the number of bit planes in each subset, and N is the number of mantissa bits required to represent the coefficient. The valid range of subsets is

$$\left\lfloor \frac{e_{\max}}{z} \right\rfloor = s_{\max} \geq s \geq 0, \tag{4}$$

where $e_{\max}$ is the largest coefficient exponent in the code-block.

The algorithm for coding floating-point values proceeds by first identifying those coefficients eligible in the largest subset $s_{\max}$. The locations of these coefficients in the largest subset are coded using standard and run mode coding from the JPEG2000 integer cleanup pass. The only difference being that no sign bits are coded. These locations can be viewed as forming a mask, herein called the eligibility mask (EM), indicating only those coefficients that are to be coded in the current subset. The coding of the eligibility map can be viewed as another pass herein referred to as the eligibility (EL) pass. After the eligible coefficients are identified, standard integer bit-plane coding is then performed, where the three passes (significance propagation, magnitude refinement, cleanup) are limited to those bits indicated by the EM. The relation of the EL coding pass to the standard integer bit-plane coding passes is shown in Figure 4.

The Standard integer bit-plane encoding proceeds up through the coding of bit-plane $z \times s_{\max}$. At this point, another EL pass is required since bit-plane $z \times s_{\max} - 1$ is in subset $s_{\max} - 1$ which may include more eligible coefficients. This next EP codes only those coefficients which become eligible in subset $s_{\max} - 1$, and these locations are added to the EM. Then z more bit planes are coded to reach the next subset boundary, followed by another EL coding pass. The process continues in a like manner until all bit planes are coded (see Figure 4). One exception needs to be mentioned. The maximum subset may skip coding some of its most significant bit planes if they are all zero. This is identical to identifying the first nonzero bit planes in standard JPEG2000 bit-plane coding [3].

The second modification to the JPEG2000 bit-plane coding deals with when to stop coding a coefficient. As mentioned above, stopping the coding of coefficients in the original JPEG2000 bit-plane coding is not an issue since each bit from every coefficient is coded in each bit plane. In floating-point bit-plane coding, all the useful coefficient information is contained in only a fraction of the bit planes. Thus the floating-point algorithm includes the concept of turning off the coding of a coefficient after all active bits have been coded. The variable used to track this is called alpha significance (denoted by α). This variable is identical to the significance function σ in JPEG2000 except in one regard. Specifically, it is set after the first significant bit in a coefficient is coded. However, it differs in that it is set to zero after all active bits in a coefficient have been coded. This corresponds to 24 bits (1 for the most significant bit and 23 for the magnitude refinement bits) for untransformed data. For wavelet transformed data, an extra pass is included to indicate when a coefficient has no remaining bits to be coded. This pass, denoted as "alpha off" (AO), only codes those coefficients that have been significant for at least 24 bit planes. For efficiency, this pass is only coded prior to every eligibility pass. Because this pass is coded only once every z bit planes, and assuming last bit to be coded to be uniformly distributed in the subset, this adds on average an extra $z/2$ zero bits that must be coded for wavelet transformed coefficients. The overhead of these extra bits is caused by the nonuniform mantissa length of each coefficient introduced by the wavelet transform. The use of α-significance improves coding performance since only those bits which contain relevant information are coded.

5. CONTEXT FORMATION AND SIGNIFICANCE PROPAGATION

Besides bit-plane coding coefficients, the JPEG2000 algorithm must create contexts to assist in the entropy coding of these bits. This section describes a new variable β-significance that is used in conjunction with the α-significance introduced in the previous section to form entropy coding contexts. These variables can be viewed as enhancements to the original significance function σ used in the fixed-point JPEG2000 bit-plane coding and context formation [3]. The significance function σ in JPEG2000 serves 3 purposes: (1) indicates the coefficients which are to be coded in the magnitude refinement pass, (2) determines the contexts for the arithmetic coding of symbols, (3) indicates neighbors of significant coefficients to be coded in the significance propagation pass. Purpose 1 of σ is filled by the α-significance (see previous section) in floating-point coding. The variable β-significance is introduced to fill purposes 2 and 3 of σ. Purposes 2 and 3 use the statistical correlation of significance among neighboring pixels to improve the entropy coding performance. However, this correlation decreases as the distance (as measured in bit planes) between the most significant bits in neighboring pixels increases. The β-significance is used to account for this decrease in correlation. The β-significance variable is set at the same time the α-significance variable is set, but is turned off after $B \leq 23$ bit planes. This models the fact that after B bit planes, the correlation among neighboring bit planes is negligible and thus not useful for coding purposes (see discussion on BYPASS mode on [3, page 504]). The exact value of B which gives best performance needs to be determined experimentally and is expected to be in the range of 4 to 15 for most image types. The β-significance variable is used in the same manner as σ in the original JPEG2000 for purposes of identifying neighbors of significant pixels for the significance propagation pass, or for determining contexts for entropy coding.

A main advantage of using the EI concept for compressing floating-point data is its simplicity. For bit-plane coding the method requires only minor changes to the JPEG2000 coding algorithm. These changes include the addition of an EL and AO pass every z-bit planes, and the tracking of two significance variables (α and β) instead of just one (σ). The other change required is an integer wavelet transform that can handle the larger bit size corresponding to EIs. Because of its simplicity, the EI method retains all the other very nice properties of JPEG2000 including fully embedded bit streams, and post-compression rate distortion optimization. It also maintains the homogeneity of the coding paths for integer and floating-point data, which is analogous to lossy and lossless compression following the same computational data paths.

6. TEST RESULTS

This section presents compression performance results of the proposed floating-point coding using the EI algorithm. Only lossless results are given since the correctness of the compression algorithms can be unconditionally verified, it avoids the issue of doing post compression rate distortion optimization, it makes it easy to directly compare the results with other compression algorithms, and the lossless case is the case emphasized in the literature for floating-point compression. The results of the proposed algorithm are for the two cases of untransformed and wavelet transformed image data, labeled *nowave* and *wave*, respectively. The parameters for the proposed algorithms are given in Table 1. The results for the proposed method should be considered as slightly optimistic since they do not include all of the overhead needed for packet formation in JPEG2000 (packet headers will have to be changed from the existing JPEG2000 headers to accommodate the new algorithm, and this issue was not pursued here). However, packet overhead in JPEG2000 is small and should not affect these results by more than a few percent. Also note that for efficiency, the truncation floor of the lossless wavelet transform was set to be 23 bits below the minimum positive exponent of the original data. This choice eliminates the expansion of data bits below the truncation floor while still maintaining a lossless wavelet transform.

Table 1: Parameters used in the proposed *nowave* and *wave* methods for the lossless floating-point compression tests.

Parameter	Value
Subset size	8
Levels of wavelet trans.	3[†]
Codeblock size	16×16
β-significance	8-bit planes

[†]For transformed data only.

The compression results are compared against the results of four additional lossless methods. The first two methods, labeled *gzip* and *bzip2*, apply the well-known gzip and bzip2 lossless compressors directly to the 4-byte integer representation of floating-point image files. The third method, labeled *gz4*, divides the original image into four one-byte integer images. Each one-byte image represents one-byte of the floating-point representation of the original floating-point values. The compressor *gzip* is then run on each of these four files, with the sum of their compressed sizes giving the resulting compressed image size. The fourth method, labeled *ljp2gz*, divides the original image into a 16-bit integer image and two 8-bit images. The 16-bit image represents the sign, exponent, and first seven (stored) mantissa bits of the floating-point representation of the original values. The two 8-bit images each represent the remaining least significant mantissa bits. The lossless JPEG2000 algorithm is run on the 16-bit image, and gzip is run on the two 8-bit files, with the compressed image size being the sum of the individual compressed sizes. The actual JPEG2000 compressor used is the publicly available JASPER implementation [11].

The proposed compression methods were tested on two different data sets. The first set consists of weather research forecast data for hurricane Isabel, and is publicly available on the Internet [12]. The data was generated by the National Center for Atmospheric Research and was used in the 2004 IEEE Visualization Contest. The data is in the form of volumetric cubes of dimensions $500 \times 500 \times 100$, and these cubes are generated at several time instances. For simplicity the 500×500 dimensions were truncated to 480×480 to be exact multiples of the codeblock size. One hundred 480×480 pixel images were created from this truncated block, and each image was compressed independently. JPEG2000 is primarily an image compression standard, but it does have some provisions for compressing volumetric data given in Part 2 of the standard. This decorrelation in the third dimension provided by Part 2 was not used in these experiments. The compression results, averaged over the 100 images for each variable, were computed for the 7th and 24th time instants for the temperature, pressure, cloud, precipitation, and U, V, and W wind speed variables. The results are shown in Tables 2 and 3.

From Tables 2 and 3, it appears that the *ljp2gz* method is the winner in terms of raw compression performance, although it had problems with the cloud variable data. The second and third best compression performance results probably go to *nowave* and *gz4*, where either one could be argued as being the second best. The Isabel data is exceptional in one

Table 2: Results of compressing hurricane Isabel data for seven variables at time step 7. Values are compression ratios (original size divided by compressed size) averaged over 100 images.

Image	gzip	bzip2	gz4	ljp2gz	Nowave	Wave
Temperature	1.24	1.34	1.79	1.86	1.54	**1.91**
Pressure	1.13	1.10	1.57	**1.73**	1.44	1.41
Cloud	8.03	8.06	8.21	7.18	**9.22**	5.06
Precip.	1.29	1.25	1.50	**1.58**	1.55	0.95
U wind speed	1.11	1.07	1.45	**1.66**	1.39	1.33
V wind speed	1.12	1.07	1.43	**1.65**	1.38	1.32
W wind speed	1.08	1.05	1.24	1.18	**1.27**	1.00

Table 3: Results of compressing hurricane Isabel data for seven variables at time step 24. Values are compression ratios (original size divided by compressed size) averaged over 100 images.

Image	gzip	bzip2	gz4	ljp2gz	Nowave	Wave
Temperature	1.24	1.36	1.71	**1.81**	1.52	1.74
Pressure	1.14	1.11	1.50	**1.68**	1.44	1.32
Cloud	9.34	9.60	8.70	6.61	**10.56**	4.47
Precip.	1.10	1.06	1.29	**1.40**	1.31	0.91
U wind speed	1.10	1.07	1.37	**1.58**	1.37	1.24
V wind speed	1.12	1.07	1.37	**1.58**	1.37	1.24
W wind speed	1.08	1.05	1.21	1.06	**1.23**	0.95

aspect. Approximately 0.5% of the original data points were not available, and these points were set to the value of 1e35[1] in the resultant data. Also, these missing points were scattered as singleton points throughout the data. These points negatively impact coding performance of probably all the tested algorithms, but especially hurt the performance of the proposed *wave* method. As mentioned in Section 3, taking the lossless wavelet transform of data with points with greatly varying exponents can greatly increase the number of mantissa bits needed to represent the resulting wavelet domain data. For example, the 1e35 coefficient starts at the 2^{116} bit value (or 243 offset bit plane), and the precipitation variable has data down to the 2^{-149} bit value (or -22 offset bit plane). Inner products using these wide values could result in coefficients needing 266 mantissa bits to represent, which is well beyond the original 24 mantissa bits needed. Due to filtering, these expanded range pixel values occupy a neighborhood around the original missing pixel value.

The second set of image data was floating-point weather image data generated by the battlescale forecast model (BFM) (more description of the data can be found in [13]). The data was originally given in $129 \times 129 \times 63$ data cubes for each variable, which was then truncated and parsed to give 63, 128×128 pixel, images. The results of compressing the temperature, pressure, water vapor, and U, V, and W wind speed variables averaged over two time steps (two data cubes) are shown in Table 4. Here the clear winner in terms of raw compression performance is the proposed *wave* method, although it does have a slight problem with the W wind speed data variable. The second and third best methods are probably the *gz4* and *nowave*, respectively.

[1] Values different from 1e35 were observed in the data, but all had (decimal) exponents greater than 35.

Table 4: Results of compressing battlescale forecast model data for six variables averaged over 2 time steps. Values are compression ratios (original size divided by compressed size) with each time step having 63 images.

Image	gzip	bzip2	gz4	ljp2gz	Nowave	Wave
Temperature	1.42	1.65	1.93	1.70	1.64	**2.15**
Pressure	1.25	1.39	1.72	1.45	1.54	**1.77**
Water Vapor	1.14	1.11	1.47	1.35	1.40	**1.57**
U wind speed	1.11	1.06	**1.39**	1.25	1.37	1.32
V wind speed	1.10	1.06	**1.39**	1.25	1.37	1.35
W wind speed	1.08	1.03	1.20	0.98	**1.23**	0.93

A conclusion that can be drawn from the experiment results is that no one algorithm gives the best compression performance for all types of data. However, the data also shows that the proposed *nowave* and *wave* methods give raw compression performance that is very competitive with state-of-the-art existing methods. The good compression performance, coupled with the benefits of a very flexible coded bitstream, make the proposed methods very attractive for use in lossless floating-point image compression.

Beside compression performance, algorithm complexity is another important consideration when choosing a compression method. Algorithm complexity is discussed here in terms of implementation complexity and computational complexity. Implementation complexity is defined as the effort required to implement a particular compression method in either software or hardware. Admittedly the effort required to implement floating-point JPEG2000 is greater that required to implement for example gzip. However, gzip, bzip, JPEG, and fixed-point JPEG2000 are all publicly and freely available. Thus, the end user does not have to exert a significant effort in implementing them. In a similar manner, implementations of the currently proposed JPEG2000-based floating-point methods will eventually also become publicly available. Thus the implementation cost of the different compression methods, at least to the end user, is negligible. The implementation cost for floating-point JPEG2000 is also reduced for manufacturers already having integer JPEG2000 implementation since much of the existing code and software can be reused.

Computational complexity is defined as the amount of machine resources, for example, CPU cycles, required to compress an image. Table 5 shows the results of measuring the computational complexity of various methods using the UNIX *time* command. The *time* output used was the "real" time, and the system used was a Pentium 4, 3 GHz, running the 2.6.12 kernel on Linux. The times for *gz4* and *ljp2gz* are somewhat overstated since the times for separating and combining the separate files was not included. Also, the *wave* and *nowave* code is currently "proof of concept" code and has not

Table 5: Real compression and decompression times measured using the UNIX *time* command, using the pressure variable from the first Battlescale Forecast Model data cube.

	gzip	bzip2	gz4	ljp2gz	nowave	wave
code	1.19	1.95	1.08	2.15	7.78	17.69
decode	0.69	1.17	0.76	2.41	5.48	12.89
total	1.87	3.12	1.84	4.55	13.25	30.58

been fully optimized in terms of its algorithms and coding. For example, after a certain number of bit planes below the most significant bit, the mantissa one and zero bits become equal probable. Thus there is no advantage to coding them with the arithmetic encoder (see the discussion of BYPASS mode on [3, page 504]). The current *wave* and *nowave* have not implemented the BYPASS mode and thus are not as computationally efficient as they could be. The results of Table 5 indicate that the proposed methods (even probably with optimization) are more computationally complex than other current floating-point methods. Since the proposed methods and integer JPEG2000 algorithm use the same bit-plane scanning, context formation, lifting wavelet transform, and arithmetic encoder, the main increase in computational complexity comes from the increased number of bit planes and increased wavelet transform complexity caused by the proposed methods' increased integer length.

7. CONCLUSIONS

This paper presented methods for compressing floating-point image data that can be incorporated into the JPEG2000 standard with very few modifications. Test results show that the raw lossless compression performance of the proposed algorithms on real floating-point image data is very competitive with current state-of-the-art methods. However, comparison of the proposed algorithms based solely on raw compression performance greatly understates the value of the proposed algorithms. Besides offering excellent floating-point lossless compression, the proposed algorithms retain all of the desirable properties of JPEG2000, such as partial lossy decompression from losslessly coded bit streams, and rate distortion optimal embedded bit streams that can be decoded with resolution, quality, or spatial ordering. Thus the proposed algorithms offer both very good compression performance, as well as higher compressed data flexibility. The results of this paper also show that the increased compressed data flexibility also comes at the cost of some increase in computational complexity.

ACKNOWLEDGMENTS

This work was supported by the National Imagery and Mapping Agency Grant NMA401-02-2017 and the Texas Instruments Foundation. The author thanks Dr. Sergio Cabrera and the Army Research Labs for providing the battlescale forecast model image data. Hurricane Isabel data was produced by the Weather Research and Forecast (WRF) model, courtesy of NCAR, and the US National Science Foundation (NSF). The author thanks the anonymous reviewer for suggesting the *gzip* and *ljp2gz* as methods for comparing compression results.

REFERENCES

[1] K. Sayood, *Introduction to Data Compression*, Morgan Kaufmann, San Francisco, Calif, USA, 2000.

[2] A. Gersho and R. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic, Norwell, Mass, USA, 1992.

[3] D. Taubman and M. Marcellin, *JPEG2000: Image Compression Fundamentals, Standards, and Practice*, Kluwer Academic, Boston, Mass, USA, 2002.

[4] V. Engelson, D. Fritzson, and P. Fritzson, "Lossless compression of high-volume numerical data from simulations," in *Proceedings of Data Compression Conference (DDC '00)*, p. 574, Snowbird, Utah, USA, March 2000.

[5] F. Ghido, "An efficient algorithm for lossless compression of IEEE float audio," in *Proceedings of Data Compression Conference (DDC '04)*, pp. 429–438, Snowbird, Utah, USA, March 2004.

[6] P. Ratanaworabhan, K. Jian, and M. Burtscher, "Fast lossless compression of scientific floating-point data," in *Proceedings of Data Compression Conference (DCC '06)*, pp. 133–142, Snowbird, Utah, USA, March 2006.

[7] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1245–1250, 2006.

[8] T. Instruments, *TMS320C3x User's Guide*, Texas Instruments, 1994.

[9] S. G. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 674–693, 1989.

[10] B. E. Usevitch, "A tutorial on modern lossy wavelet image compression: foundations of JPEG 2000," *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 22–35, 2001.

[11] M. Adams, "The JasPer Project," Source code and user manuals, http://www.ece.uvic.ca/~mdadams/jasper/.

[12] http://vis.computer.org/vis2004contest/data.html.

[13] O. M. Kosheleva, B. E. Usevitch, S. D. Cabrera, and E. Vidal Jr., "Rate distortion optimal bit allocation methods for volumetric data using JPEG 2000," *IEEE Transactions on Image Processing*, vol. 15, no. 8, pp. 2106–2112, 2006.