Lossless Coding of Floating Point Data with JPEG 2000 Part 10

Manuel N. Gamito[a] and Miguel Salles Dias[a]

[a]ADETTI/ISCTE, Av. das Forças Armadas, 1600–086 Lisboa, Portugal

ABSTRACT

JPEG 2000 Part 10 is a new work part of the ISO/IEC JPEG Committee dealing with the extension of JPEG 2000 technologies to three-dimensional data. One of the issues in Part 10 is the ability to encode floating point datasets. Many Part 10 use cases come from the scientific and engineering communities, where floating point data is often produced either from numerical simulations or from remote sensing instruments. This paper presents the technologies that are currently being developed to accommodate this Part 10 requirement. The coding of floating point datasets with JPEG 2000 requires two changes to the coding pipeline. Firstly, the wavelet transformation stage is optimized to correctly decorrelate data represented with the IEEE 754 floating point standard. Special IEEE 754 floating point values like Infinities and NaN's are signaled beforehand as they do not correlate well with other floating point values. Secondly, computation of distortion measures on the encoder side is performed in floating point space, rather than in integer space, in order to correctly perform rate allocation. Results will show that these enhancements to the JPEG 2000 coding pipeline lead to better compression results than Part 1 encoding where the floating point data had been retyped as integers.

Keywords: JPEG 2000 standard, JP3D standard, Floating Point Coding, IEEE 754 Standard, Lossless Compression

1. INTRODUCTION

JPEG 2000 Part 10 was approved as a new work item of the JPEG committee during the 25th WG1 meeting in Sydney, Australia.[1] The main purpose of Part 10, which became informally known as JP3D, is to extend the coding abilities of JPEG 2000 to three dimensional data. Part 2 of the standard can already provide a limited form of three dimensional coding by having a multi-component sequence of 2D slices. Cross-component correlation from Part 2 can then be applied, prior to compression. The problem with this approach is that a different compression strategy is applied along the third axis dimension relative to the other two axes. For example, the concepts of tiles, precincts and code blocks do not extend to the third dimension, for which only the coding facilities of Part 2, Annex J, are defined.[2] For these reasons, it was decided that a new standard was necessary to isotropically encode all three dimension axes, by extending the Part 1 coding technology to volume data, while still retaining the ability to code two-dimensional data in a compatible way with Part 1 and Part 2 compliant readers.

The ability of JP3D to provide isotropic compression of three dimensional data answers many use cases in the medical imaging community but additional goals were defined to address the scientific and engineering communities also.[3] These additional goals for JP3D are:

- The coding of floating point data.
- The coding of data over variable resolution grids.

This paper will focus on the facilities of JP3D for the coding of floating point data. A companion to this paper will focus on the aspects of variable resolution coding.[4] Floating point numbers are a ubiquitous data type in the scientific and engineering communities. These communities generate quite large datasets with floating point numbers, originating from either computer simulations or direct measurements of sensor data. The coding of floating point datasets is still very much in its infancy and, to date, a widely disseminated and efficient compression method for floating point data is not yet available. Currently, lossless compression of floating point datasets is performed by invoking the `compress`, the `zip` or

E-mail: mag@iscte.pt & jmd@iscte.pt, Telephone: +351 21 7903064, Fax: +351 21 7935300

the gzip UNIX compression utilities on the files containing the datasets. This strategy amounts to applying LZW-type compression algorithms, which are far from optimal for floating point data.

Storage of large floating point datasets is often done using proprietary or *ad hoc* file formats but some popular formats for scientific applications do exist. The HDF5 file format can store *n*-dimensional arrays in a variety of data types, including IEEE 754 single and double precision, in little or big endian byte order.[5] This file format allows compression of data by splitting the arrays into data chunks and compressing each chunk with a LZW algorithm prior to writing it to file. The PLOT3D file format is also quite common in the Computational Fluid Dynamics community for storing the results of fluid simulations.[6] PLOT3D does not feature any form of compression.

Section 2 will show previous work on the compression of floating point data. Section 3 will present the method that is currently being proposed for integration into JPEG 2000 Part 10. It will show how the proposed techniques build on top of the JPEG 2000 pipeline rather than being incompatible with it. Section 4 will give results for several two-dimensional scientific data from computational fluid dynamics. It will be seen how the proposed method retains all the scalability properties of JPEG 2000. Finally, Section 5 will present conclusions. Because, at this point, a fully working JP3D prototype has not yet been developed, results will be shown using an existing Part 1 two-dimensional coder which has been enhanced with the floating point coding technologies proposed for JP3D.

2. PRIOR ART

Existing methods for the lossless compression of floating point data can be roughly divided into two categories: predictive methods and transform methods. Predictive methods code the residual between a floating point number and an estimate for that number given the previously coded values. Transform methods code the data after having been passed through some type of decorrelating transform. In either case, the IEEE 754 floating point representation is used to represent a number with three parameters: a sign, an exponent and a mantissa.

Engelson *et al* have presented a method for the lossless compression of one-dimensional sequences of floating point numbers, which represent the solution of some ordinary differential equation.[7] In his work, the floating point numbers are treated as though they were integers. Treating floating point numbers as integers assumes that if two floating point numbers are close to each other then the difference between their two integer representations will be a small number. This assumption breaks down, however, when the two floating point numbers have different exponent values, even though they remain close in real space. Two coding options are proposed by Engelson: the coding of the *m*-th order differences between the integer representations of the numbers or the coding of the residuals between each number and an estimate of the same number obtained from an *m*-th order polynomial Lagrange extrapolation. The Lagrange extrapolator is computed from the *m* previous numbers in the sequence and, again, an integer representation for the floating point numbers is used. The difference values or the residual values are then inserted into the bitstream by first dropping all redundant sequences of zero or one bits from the most significant position in each value.

Isenburg *et al* developed a method for the lossless coding of triangular meshes, where the position of all vertices in the mesh are represented by three-dimensional vectors of floating point numbers.[8] Their coding method is based on a region-growing algorithm for triangular meshes. Starting from a seed triangle, the opposite vertices for the adjacent triangles are encoded and this procedure is iterated until all triangles in the mesh have been processed. A parallelogram predictor is used to estimate the position of a triangle vertex based on the vertices from an adjacent triangle that has already been processed. The residual vector between the actual and predicted vertex positions is then encoded, one coordinate at a time. This encoding of residual vectors resorts directly to the IEEE 754 format of floating point numbers. Each residual vector coordinate is comprised of a sign bit, a 8 bit exponent field and a 23 bit mantissa field in single precision format. All fields are entropy coded with an arithmetic encoder. The arithmetic encoder uses a probability context model for the sign, exponent and mantissa fields that is dependent on the value of the exponent for each floating point vector coordinate. This dependence of the sign and mantissa fields relative to the exponent field is motivated by the strong non-linearity of the IEEE 754 representation and is exploited again in our proposed coding method, as will be explained in Section 3.1.

Tao and Moorhead propose a wavelet transform method for the lossless coding of floating point data using biorthogonal B-spline wavelet filters, while manipulating directly the IEEE 754 representation.[9] During the computation of the wavelet transform, the number of mantissa bits must be increased in order to maintain the property of losslessness. The extra number of mantissa bits across the dataset is signalled with an octree, prior to transmission of the wavelet coefficients. The authors do not mention which method, if any, was used to further compress the wavelet coefficients. Trott, Moorhead *et*

IEEE 754 Number	Exponent (n)	Mantissa (m)
Single precision	8	23
Double precision	11	52

Table 1. Number of mantissa and exponent bits for single and double precision IEEE 754 floating point numbers.

al later proposed a slightly different wavelet transform method.[10] In this later work, the Haar wavelet basis is used and all floating point data values are converted from IEEE 754 single to double precision. The increased precision allows the computation of the wavelet coefficients to remain lossless. The sequence of double precision coefficients is then transmitted byte-wise with a Huffman encoder. Finally, Du proposes a variation of the two previous wavelet transform lossless coding methods with a more sophisticated encoding strategy for the wavelet coefficients.[11] With Du's method, the data is again expanded to double precision, prior to wavelet transformation. Then, the 11 bit exponent field and the upper 36 bits of the mantissa field, for each coefficient, are entropy coded separately. The remaining 16 bits of the mantissa field are run-length encoded. This is because, according to the author, the less significant bits of the mantissa are mostly zeroes due to the initial expansion of the data to double precision and therefore they can be efficiently run-length encoded.

Within the ISO/IEC JPEG Committee, the lossless coding of floating point data has been the study of research, from the start of the JP3D work part. Wohlberg and Brislawn have investigated different strategies for coding the IEEE 754 data. Some of those strategies included splitting the sign, mantissa and exponent fields of all numbers into separate JPEG 2000 components, converting all numbers to a same exponent or converting all numbers to a fixed point representation, among other options.[12, 13]

3. LOSSLESS CODING OF IEEE 754 DATA

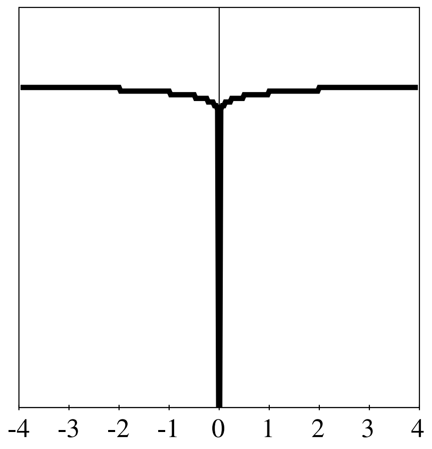
The most widely used binary representation for floating point numbers is the IEEE 754 specification, as it is implemented in all of the major computer architectures in use today.[14] It makes sense, therefore, to develop a JPEG 2000 lossless coding mechanism for IEEE 754 numbers. The IEEE 754 specification provides single and double precision floating point numbers, written in radix 2 format. A number is represented by three components: a sign bit, a n-bit exponent field $a = a_0a_1 \ldots a_n$ and a m-bit mantissa field $b = b_0b_1 \ldots b_m$. A single precision number fits into four bytes, while a double precision number fits into eight bytes. Table 1 gives the exponent and mantissa bit lengths for single and double precision numbers. The following expression gives the relation between a real number and its single precision floating point representation:

$$x = \pm[1.b_0b_1 \ldots b_{22}]_2 \times 2^{a-127} \tag{1}$$

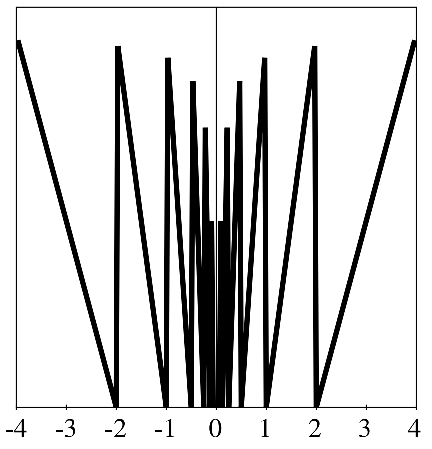
The mantissa field alone represents the fractional bits of a number in the range $[1, 2[$. This number is then multiplied by a power of two to give it the proper dynamic range. The one bit on the left side of the decimal point is always defined implicitly. The exponent is stored with a bias value of 127 added to it, to avoid having to work with signed exponents. The addition of the bias causes the exponent to always be a positive integer. The representation for double precision numbers is similar to (1) but an exponent bias of 1023 is used, instead of 127. The real number zero is represented with $a = 0$ and $b = 0$. A zero can be either positive or negative, depending on its sign bit. There is an exception to the rule of the implicit leftmost one bit, explained previously, which happens for *subnormal* numbers. These are the smallest numbers (other than zero) than can be represented with IEEE 754 and are characterized by a biased exponent $a = 0$ and a mantissa $b > 0$. The expression for a subnormal number, again in single precision, is:

$$x = \pm[0.b_0b_1 \ldots b_{22}]_2 \times 2^{-126} \tag{2}$$

The IEEE 754 representation is very non-linear. The distance between consecutive floating point numbers depends on the value of the exponent. As an example, the distance between two consecutive single precision subnormals is 1.4×10^{-45} while the distance between two consecutive single precision numbers at the largest exponent is 2.0×10^{31}. The disparity



(a) Biased Exponent



(b) Mantissa

Figure 1. Evolution of the biased exponent and mantissa fields for a uniform sequence of single precision IEEE 754 numbers.

is even worse for double precision. This non-linearity creates problems for the efficient compression of floating point data with JPEG 2000 Part 10 that did not exist for the compression of integer data with Part 1.

The starting point to losslessly compress IEEE 754 data with JPEG 2000 is to separate all floating point numbers into their sign, biased exponent and mantissa fields and to assign each field independently to a three component unsigned JPEG 2000 image, as initially proposed by Wohlberg.[13] The sign bit is assigned to image component 0, with a bitdepth of one. The biased exponent is assigned to image component 1 and the mantissa to image component 2. The bitdepths for these last two image components are as given in Table 1. A difficulty arises for double precision numbers because Part 1 specifies that no image component can have a bitdepth greater than 38 while the mantissa for a double precision number requires 52 bits.[15] The maximum component bitdepth will have to be amended in Part 10 if double precision numbers are to be coded. The prototype software, developed for this paper, had the range of its datatypes enlarged to allow a maximum bitdepth of 59 bits in the mantissa component.

3.1. Wavelet transformation of IEEE 754 Data

In order to motivate some of the arguments that will follow, we begin this section by studying a simple one-dimensional dataset and show the evolution of the biased exponent and mantissa components for the sequence of real numbers $x_i = i/32$. Figure 1 shows the exponent and mantissa plots for the set $\{x_i : |i| < 128, i \in \mathbb{Z}\}$ represented with single precision numbers. The biased exponent exhibits good correlation, varying slowly between 122 and 128 with the exception of the origin $x_0 = 0$, which causes a discontinuity. The wavelet transform of this exponent component should be able to decorrelate much of the data were it not for the discontinuity at the origin, which will cause some large wavelet coefficients to appear at the high-pass subbands. The mantissa component exhibits linear change inside the intervals characterized by a constant exponent. Two of those intervals, evident in the mantissa plot, are $[1, 2[$, for which $a = 127$, and $[2, 4[$, for which $a = 128$. Similarly to the exponent component, the wavelet transform for the mantissa should be able to decorrelate well the data except for the discontinuities occurring at the transition between intervals with different exponents. The point to remember is that smooth variations of floating point numbers, like the one of Fig. 1, can lead to very sharp but localized variations of their exponent and mantissa values. These sharp variations induce potentially large wavelet coefficients after the wavelet transform stage of JPEG 2000. The large wavelet coefficients, in turn, require a large number of bits to be encoded and cause an undesirable increase in bitrate for the compressed dataset.

Considering now actual two or three dimensional floating point datasets, the optimal strategy is to apply the wavelet transform only inside the regions where the data is known to change smoothly, thereby avoiding the discontinuities. The smooth regions can be easily identified in the data. For the sign component, the wavelet transform can be applied everywhere, as there are no discontinuities. For the biased exponent component, regions are formed according to one of the following two criteria: 1) the data inside the region is everywhere different from zero. 2) the data inside the region is everywhere equal to plus or minus zero. For the mantissa component, the regions are characterized by having associated sign and exponent values which are constant*. This type of region-based wavelet transformation is possible with the use of a *Shape-Adaptive Discrete Wavelet Transform*, or SA-DWT, as defined by Li and Li.[16] The SA-DWT filters the data with the wavelet transform kernel inside arbitrary irregularly shaped regions and applies the appropriate symmetric extensions at the boundaries of these regions. The SA-DWT can work with any wavelet kernel although only the 5x3 kernel of Part 1 is relevant for this paper.

To properly decode the wavelet transformed IEEE 754 data, a Part 10 reader will need to have access to extra signalling information in order to identify the zero regions in the exponent component. Details about this extra signalling information will be given in the next section. The sequence of steps a Part 10 reader will have to take, during its inverse wavelet transformation stage, is the following:

1. Apply the standard IDWT to the transformed sign component.
2. With access to additional signalling information that must have been decoded already, apply the SA-IDWT to the regions defined in the transformed exponent component.
3. With access to the data in the exponent component, decoded in the previous step, apply the SA-IDWT to the regions of constant sign and exponent defined in the transformed mantissa component.

Because the sign is the component 0, the exponent is the component 1 and the mantissa is the component 2 of a JPEG 2000 image, the sequence of inverse wavelet transform steps is guaranteed to take the order defined above. In the case of early codestream termination only an approximation of the sign and exponent components can be decoded. This will induce errors in the shape of the regions to be decoded inside the mantissa component and thus further increase the distortion of the decoded data. Such behaviour is acceptable because a truncated codestream is not expected to be lossless, while the scalability of JPEG 2000 is still retained.

Figures 2(a), 2(b) and 2(c) show the sign, exponent and mantissa components, respectively, for a double precision dataset numerically obtained from a computational fluid model. The exponent component has a small dynamic range, varying only over a small number of close values. The image contrast in Fig. 2(b) had to be increased, otherwise the variation in exponent values would not have been visible. The mantissa component has many small regions of constant exponent and is, therefore, full of high frequency content caused by the sharp transitions across adjacent exponent regions. Figure 3 shows several rate-distortion curves, of the mantissa component only, for this dataset. The PSNR for these curves was computed with the expression:

$$\mathrm{PSNR} = 10 \log \frac{\left(2^{52} - 1\right)^2}{\mathrm{MSE}} \tag{3}$$

The mean squared error (MSE) is computed in the usual way. The application of one level of the standard DWT, such as a Part 1 encoder would do, actually degrades the coding performance, relative to an encoding with no application of the wavelet transform. This is because of the need to encode losslessly the many sharp transitions in the mantissa component, which consume much bitrate. The application of one level of the SA-DWT, instead of the DWT, corrects this situation and produces a more efficient compression, relative to no application of the wavelet transform, as should be expected.

*The zero regions will have $a = 0$ in the exponent component and $b = 0$ in the mantissa component. These regions need not be wavelet transformed since their wavelet coefficients are going to remain at zero. The sign of these zero regions is coded by the wavelet transform of the sign component.

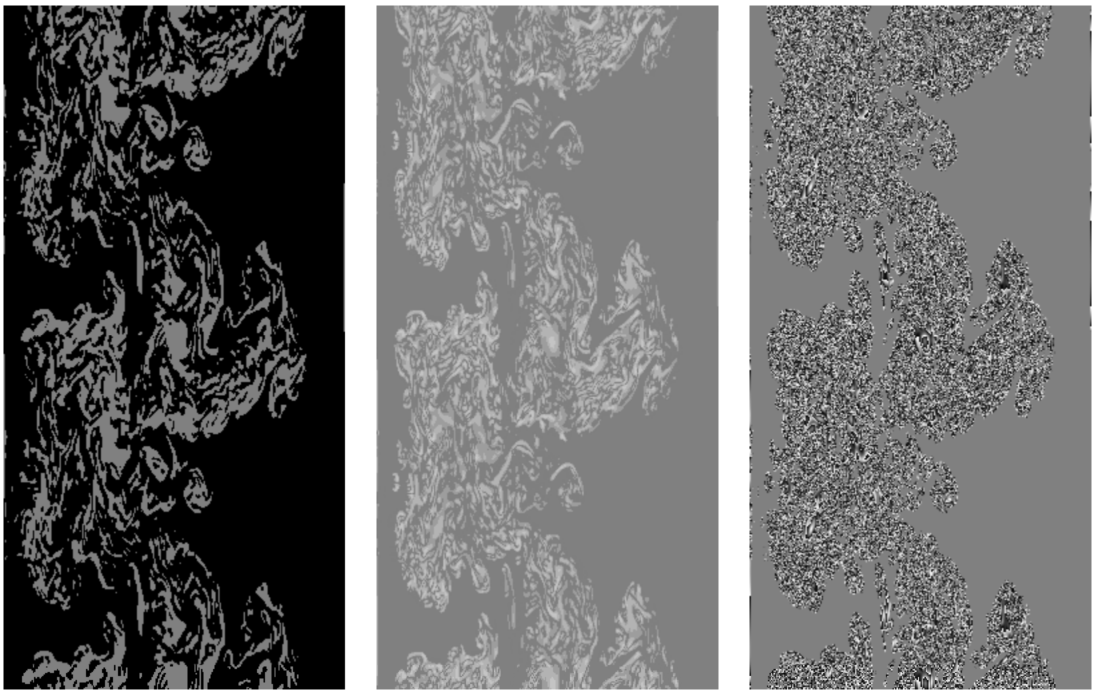


Figure 2. Sign, exponent and mantissa components for a two-dimensional fluid dynamics dataset in double precision IEEE 754 format.

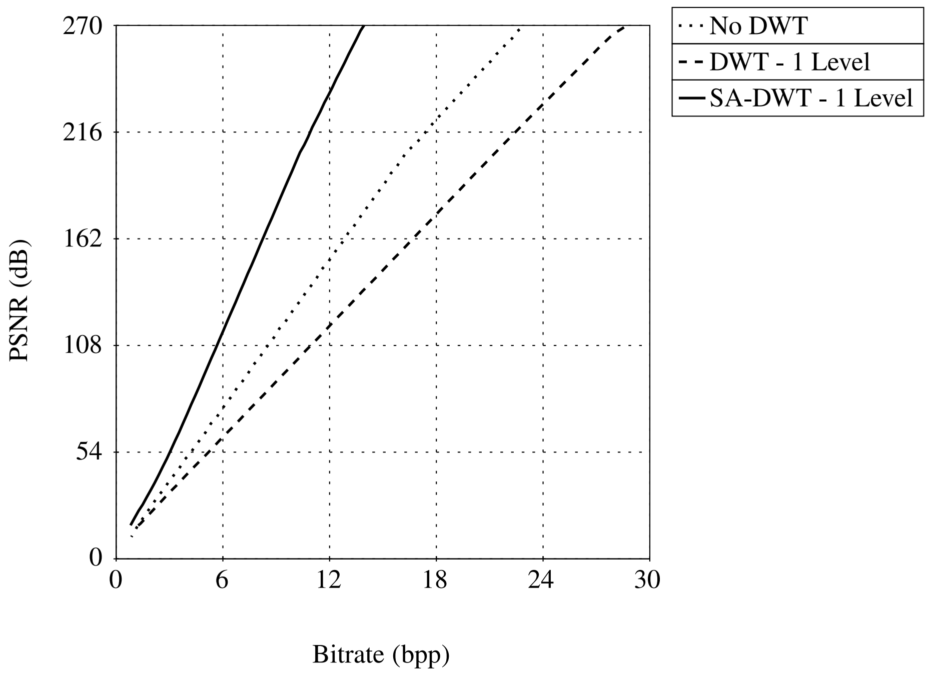


Figure 3. Rate-distortion curves for the mantissa component of the fluid dynamics dataset in Fig. 2 with no wavelet transform, one level of the DWT and one level of the SA-DWT.

3.2. Signalling of IEEE 754 Special Numbers

The IEEE 754 specification defines three special numbers. One of them is zero (positive or negative) and was considered in the previous section. The other two special numbers are Infinity (positive or negative) and Not-a-Number (NaN). Infinities are the result of floating point operations like $1/0$. They are represented in single precision with $a = 255$ and $b = 0$. NaN's are used to signal the result of erroneous computations and could be obtained, for example, by trying to compute $\sqrt{-1}$. The single precision representation for a NaN is $a = 255$ and $b > 0$[†]. IEEE 754 uses the mantissa bits of a NaN to distinguish between two types of Not-A-Numbers: signalling NaN's and quiet NaN's.[17] A signalling NaN, when it first occurs as the result of a computation, raises a signal, which can then be trapped by an appropriate callback function supplied by the application. A quiet NaN's, like the name implies, does not have this behaviour. A lossless compression algorithm for IEEE 754 must be able to retain the signalling behaviour of NaN's by coding their mantissa bits.

The probability of occurrence of Infinities or NaN's in actual floating point datasets is very low but a Part 10 encoder should still be prepared to handle them. The problem with Infinities and NaN's is the same problem that existed with zeroes, as explained in the previous section. They do not correlate well with normal numbers around them, in terms of the wavelet transform, and introduce discontinuities in the exponent and mantissa components which incur a large penalty in bitrate. Zeroes, Infinities and NaN's must be isolated inside special regions and the SA-DWT must be applied to them alone. An additional signalling of these special regions must be done beforehand so that a Part 10 compliant reader knows their location by the time it reaches its Inverse Wavelet Transformation stage. The signalling is done with an octree coding method and is placed inside the main header of the JPEG 2000 codestream. An additional comma code is used to flag each leaf node in the octree as being either a normal or subnormal IEEE 754 number (the most likely case), a zero, an Infinity or a NaN (the least likely case). The octree, once generated, encodes the distribution of the special IEEE 754 regions throughout the dataset. It is scanned in a breadth-first manner, as the following pseudo-code illustrates. Sorted lists are used to keep track of which octree nodes need to be checked for subdivision at the next higher subdivision level.

```
initialize list to root node of octree
call Breadth_Encode(list)

Procedure Breadth_Encode(list):
    initialize new list to empty
    for every node in list
        if the node  is subdivided
            /* signalling of interior nodes */
            output "0"
            for all children of node
                add child to new list
        else
            /* signalling of leaf nodes */
            if not at highest subdivision level
                output "1"
            output comma code for content of leaf node
    /* scan next higher subdivision level of octree */
    call Breadth_Encode(new list)
```

The binary output of the octree scanning procedure is fed into the JPEG 2000 MQ binary arithmetic coder for additional entropy coding.[18] The MQ coder prevents the words in the range `0xFF90` to `0xFFFF` from ever being inserted into the codestream. This is because those words are reserved for the markers defined by the standard. By using the MQ coder, it is ensured that the octree signalling part of the final codestream will be free from words that might be mistakenly interpreted as JPEG 2000 markers. Presently, the MQ coder's uniform probability context model is being used to perform entropy coding of the octree signalling bits. This model is far from optimal and a probability context model, suitable for octree signalling, should be added to the MQ coder. The compressed bitstream, resulting from IEEE 754 special region

[†]In the remainder of this paper, we will consider only single precision representations. Extension to double precision representations is trivial.

signalling, is encapsulated inside a proposed new SOF (Start of Float) marker segment, which must be present in the main header of the JPEG 2000 codestream. If the compressed bitstream is larger than the maximum allowed marker segment length of 65534 bytes, it can be split into several consecutive SOF markers segments. The complete bitstream will then be reconstructed by a Part 10 reader.

The sign of Infinities is encoded in component 0 of the JPEG 2000 image, along with the signs of zeroes and of all other values in the dataset. The constant exponent $a = 255$, used by both Infinities and NaN's, is trivially wavelet transformed. The shape-adaptive wavelet transform, in the exponent component, now isolates regions occupied by zeroes, Infinities or NaN's. Also, the mantissa inside a region with NaN's goes through the SA-DWT, just like the mantissas for any other numbers. It may happen, during the process of decoding a truncated codestream, that extraneous Infinities or NaN's may be erroneously inserted into the decoded dataset. This may happen because the process of dequantizing truncated wavelet coefficients, followed by the inverse wavelet transformation, may generate numbers with exponent 255 that did not exist in the original dataset. The possibility of introducing these extraneous special numbers into the decoded dataset is potentially dangerous, especially if the dataset undergoes further processing. NaN values will corrupt all computations that are performed on them and post-processing software may not be prepared to handle Infinities. These dangers can be avoided, in a Part 10 reader, by comparing every special number against the octree signalling that will already have been obtained prior to the decoding of the wavelet coefficients. If an Infinity or a NaN is decoded and it is not signalled in the octree, it is not a value from the original dataset and it is replaced by a zero. Zeroes are much less harmful for post-processing of the decoded data by additional software.

3.3. Rate Allocation for IEEE 754 Data

Rate allocation is purely an encoder issue. The purpose of rate allocation is to define where, in the JPEG 2000 codestream, should each coded wavelet coefficient be inserted in order to provide a continuously improving approximation to the final dataset, as more and more bytes are extracted from the codestream by a reader. Stated differently, given a desired bitrate, rate allocation uses a linear optimization algorithm to decide which wavelet coefficients are used to minimize image distortion, subject to the constraint that the total desired bitrate should not be exceeded. The rate allocation algorithm, therefore, needs to measure image distortion, given a set of truncated wavelet coefficients. Part 1 coders measure distortion for integer data. If this were done for the three integer components of a floating point dataset, the resulting rate allocation would be far from optimal. Distortion measures for rate allocation must be adapted to the specifics of the IEEE 754 representation. We will consider in this section a slightly different version of (1) for the representation of a floating point number:

$$x = s \cdot b \cdot 2^a \tag{4}$$

The sign $s = \pm 1$ multiplies by the mantissa b, which, in this version, includes the implicit one bit to the left of the decimal point (or the implicit zero bit, if x is a subnormal). The exponent a is in unbiased form. A Part 1 coder would compute the total distortion D for a dataset as $D = \sum_i d_i$, where d_i is the distortion caused by the truncation error for dataset sample x_i. The distortion d_i of a floating point number, according to Part 1, is the sum of the squared truncation errors Δs_i, Δb_i and Δa_i, for the sign, the mantissa and the exponent, respectively:

$$d_i = \Delta {s_i}^2 + \Delta {b_i}^2 + \Delta {a_i}^2 = \left(s_i - \hat{s}_i\right)^2 + \left(b_i - \hat{b}_i\right)^2 + \left(a_i - \hat{a}_i\right)^2 \tag{5}$$

The hat symbol over a parameter represents its value after some of the least significant bits have been truncated. The problem with the distortion measure (5) is that it gives equal weight to the sign, exponent and mantissa truncation errors. Consider, for example, a number where only the least significant bit of the mantissa has been truncated. In the two extreme cases, the distortion will be 2.0×10^{-89}, if x_i is a subnormal, or 4.1×10^{62}, if x_i has the largest biased exponent value. The distortion (5), however, will always give 1.4×10^{-14}, independently of the exponent.

We propose to compute the distortion measure in wavelet coefficient space, rather than in the dataset domain space. The total distortion is obtained by summing the distortions of all wavelet coefficients, over all subbands:

$$D = \sum_i G_i \sum_j d_j^i \tag{6}$$

In this expression, d_j^i is the distortion caused by the truncation error of the y_j^i wavelet coefficient at the i subband. The distortion in each subband must be multiplied by the subband gain G_i in order to correctly compute the total distortion. Recipes for the computation of the subband gains are available in the literature.[18] The correct distortion measure for a truncated wavelet coefficient is the sum of the squared truncation errors Δs_j^i, Δb_j^i and Δa_j^i for the sign, mantissa and exponent, respectively, of the wavelet coefficient y_j^i in subband i but these truncation errors must be properly weighted to include the non-linear effects of the IEEE 754 representation.

The truncation error for the exponent is approximated by the first term of a Taylor series expansion:

$$\Delta a_j^i = \ln 2 \cdot x_j^i \cdot \{\Delta y_j^i\}_a + O\big(\{\Delta y_j^i\}_a^2\big) \tag{7}$$

In the above expression, $\{\Delta y_j^i\}_a$ is the truncation error for the exponent of wavelet coefficient y_j^i and the floating point number x_j^i is a low-pass version of the dataset sample x_j in the resolution space of subband i. The low-pass datasets x_j^i are obtained by filtering and decimating the original dataset with a low-pass filter. For example, if the initial dataset has a resolution of 512 × 512 samples, the second decomposition level of the wavelet transform will have subbands with a resolution of 128 × 128 samples. In this case, the low-pass and decimation filter will be applied twice to the original dataset. For the implementation used in this paper, the low-pass dataset samples were obtained by averaging a square of four adjacent floating point samples, from the dataset of the current decomposition level, down to a single sample in the dataset of the next decomposition level. The low-pass floating point datasets x_j^i are computed during the wavelet transformation stage of the encoder and are kept in memory, to be used later during rate allocation. The expression (7) is only an approximate linear estimate of the true exponent truncation error. The estimate is more accurate when only the least significant bits of a wavelet coefficient exponent are truncated and degrades increasingly as the truncation progresses towards the most significant bits.

The truncation error for the mantissa has an exact expression, as opposed to the approximation (7) for the truncation error of the exponent:

$$\Delta b_j^i = s_j^i \cdot 2^{a_j^i} \cdot \{\Delta y_j^i\}_b \tag{8}$$

The expression $\{\Delta y_j^i\}_b$ represents the truncation error for the mantissa of the wavelet coefficient y_j^i at the i subband. The sign $s_j^i = \{x_j^i\}_s$ is obtained from the low-pass dataset sample x_j^i. The exponent $a_j^i = \{x_j^i\}_a$ is obtained in a similar way.

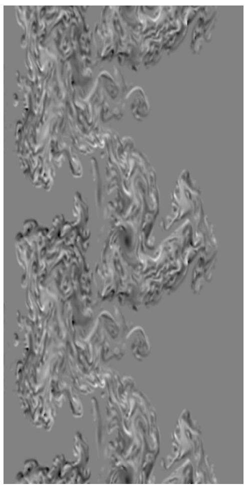
Finally, the truncation error for the sign is:

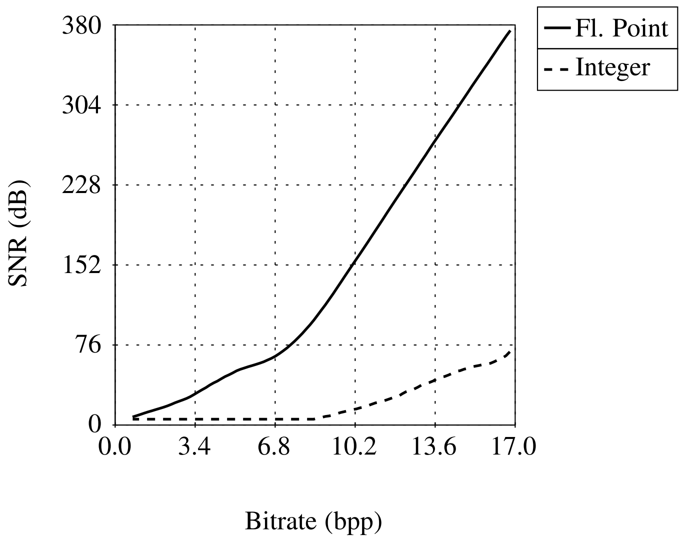
$$\Delta s_j^i = b_j^i \cdot 2^{a_j^i} \cdot \{\Delta y_j^i\}_s = |\, x_j^i \,| \cdot \{\Delta y_j^i\}_s \tag{9}$$

After experimentation, it was found that the dynamic range for the truncation errors, given in the three previous equations, was too small and caused precision problems while running the linear optimization algorithm of the rate allocator. The dynamic range was amplified by multiplying all three truncation errors with 2^{23}, which is two to the power of the number of mantissa bits. This causes the dynamic range to be comparable to the dynamic range of the truncation errors in (5) for a Part 1 coder.

To summarize, the distortion measure for floating point data, in a Part 10 encoder, is computed by summing in (6) the distortions d_j^i for all wavelet coefficients. The distortion of each wavelet coefficient is the sum of the three squared and weighted truncation errors:

$$\begin{aligned}
d_j^i &= \big(\Delta a_j^i\big)^2 + \big(\Delta b_j^i\big)^2 + \big(\Delta s_j^i\big)^2 && \text{(10a)}\\
\Delta a_j^i &\approx 2^{23} \cdot \ln 2 \cdot x_j^i \cdot \{\Delta y_j^i\}_a && \text{(10b)}\\
\Delta b_j^i &= 2^{23} \cdot s_j^i \cdot 2^{a_j^i} \{\Delta y_j^i\}_b && \text{(10c)}\\
\Delta s_j^i &= 2^{23} \cdot |\, x_j^i \,| \cdot \{\Delta y_j^i\}_s && \text{(10d)}
\end{aligned}$$


(a) Dataset



(b) Rate-distortion curves

Figure 4. The `fenris-ccvort1500` double precision dataset and its rate-distortion curves for floating point distortion and integer distortion.

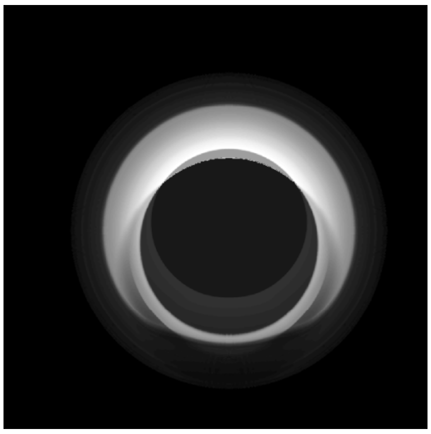
When compressing a single precision dataset, the distortion must be computed with double precision to avoid floating point overflows. When compressing a double precision dataset, the distortion must be computed with long double precision (12 bytes). In practice, however, double precision is sufficient for the most common datasets. For example, a double precision dataset where $|x_i| < 1$ will never cause floating point overflows. Only double precision datasets containing very large numbers should use long double precision for rate allocation.

4. RESULTS

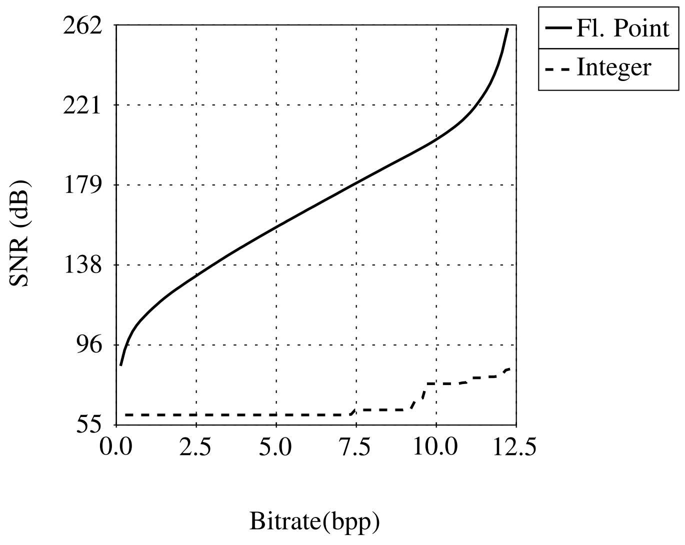
Figure 4(a) shows a double precision dataset, with a resolution of 256×512 samples. It is the same fluid simulation dataset that was used in Figure 2. Figure 4(b) shows the rate-distortion curves for this dataset when Part 1 integer rate allocation or Part 10 floating point rate allocation are used. Three levels of wavelet decomposition are used. The performance is computed in terms of the SNR, instead of the PSNR, since the peak value for a floating point data type would be inordinately large. Rate allocation based on integer data is clearly suboptimal. The SNR settles too early at at value of 6 dB and becomes insensitive to further codestream truncation from that point on. Also, the rate-distortion curve for integer rate allocation starts with a much lower value than the rate-distortion curve for floating point rate allocation. This is because the integer rate allocator begins truncating exponents from the start. The floating point rate allocator delays truncation of exponents until much later, knowing that truncated exponents incur a large distortion for IEEE 754 data. The point where the floating point rate allocator finally begins to truncate exponents is noticeable as an inflection in the graph in the vicinity of 7 bpp.

Dataset	Raw	Compressed	%	Octree	%
`fenris-ccvort1500`	1048576	277941	73.49	9397	3.38
`rage-dens1516`	4194304	1629465	61.15	1048	0.06

Table 2. Percentage of compression for the total codestream and percentage of overhead for octree coding of IEEE 754 special numbers, relative to the compressed codestream length, for the two datasets. Codestream lengths are in bytes.



(a) Dataset



(b) Rate-distortion curves

Figure 5. The `rage-dens1516` single precision dataset and its rate-distortion curves for floating point distortion and integer distortion.

Figure 5(a) shows a single precision dataset, with a resolution of 1024 × 1024 samples. It is the output of a fluid dynamics numerical simulator and represents fluid density. Fluid density takes only positive values in the interval $[0, 1[$ and, consequently, the sign component for this dataset is everywhere zero. Figure 5(b) shows the rate-distortion curves, with three levels of wavelet decomposition. Again, rate allocation with integer data is suboptimal, while rate allocation with IEEE 754 presents a normal rate-distortion curve.

Table 2 shows several statistics for these two datasets, most importantly the overall percentage of compression and the percentage of the total codestream occupied by the octree coding for IEEE 754 special numbers. These datasets do not feature any NaN's or Infinities so octree coding is required only to signal the positions of zeroes inside the datasets. It can be seen in the table that signalling of special numbers occupies a negligible portion of the codestream and therefore does not contribute significantly to the total bitrate. The use of proper context models for octree coding, in the JPEG 2000 MQ arithmetic coder, should improve these figures even more. The percentage of the codestream occupied by signalling of special numbers is much larger for `fenris-ccvort1500` than for `rage-dens1516` because the former has a more complex distribution of zero regions throughout the dataset and so it requires more bits to signal those regions. The `rage-dens1516` dataset features only a narrow border of zeroes, surrounding the circular structure in the center, which takes comparatively fewer bits to be signalled.

5. CONCLUSIONS

Lossless compression of IEEE 754 floating point data is possible within the framework of the JPEG 2000 coding standard. The modifications that need to be made to the JPEG 2000 coding pipeline are three: 1) In the wavelet transformation stage, the standard wavelet transform must be replaced by a shape-adaptive wavelet transform. 2) In the main header of the JPEG 2000 codestream, a new marker segment must be inserted, containing a bitstream that signals the location of IEEE 754 special numbers (zeroes, Infinities and NaN's) within the dataset to be coded. 3) On the encoder side, the rate allocation algorithm must be modified to correctly compute distortion measures for floating point numbers, with truncated bits, rather than integer numbers. These modifications are not so drastic as to cause incompatibilities with the existing Part 1 coding technologies. In fact, the coder implemented for this paper can compress either integer data, with Part 1, or floating point data by calling slightly different versions of the same routines.

Compression ratios of between 60% and 70% have been observed for both single precision and double precision datasets. Such high compression ratios have not been reported before in the existing literature. Furthermore, all the advantages of the JPEG 2000 coding standard are retained for floating point data, more importantly, distortion scalability and resolution scalability. It is still necessary to test the coding of Regions-of-Interest (ROI's) for floating point data, although it is envisaged that no major technical hurdles will be found. The next step will be the development of a lossy coding mechanism for floating point data, within JPEG 2000. The emphasis here will be not so much on the wavelet transform, since this can be computed in the native floating point format of the data, but on more advanced quantization techniques that can handle the large dynamic ranges exhibited by floating point datasets.

ACKNOWLEDGMENTS

The authors would like to thank Chris Brislawn, the editor of JP3D, for his guidance. The authors would also like to thank Bill Rider and Randall Kanzleiter, at Los Alamos National Laboratory, for graciously allowing the use of the `fenris-ccvort1500` and `rage-dens1516` datasets for the purpose of testing JP3D technologies.

REFERENCES

1. C. Brislawn, "JP3D: Proposal for work item subdivision." ISO/IEC JTC1/SC29/WG1 document #2379, 2001.
2. M. Boliek, "JPEG 2000 Part 2: Final draft before international standard." ISO/IEC JTC1/SC29/WG1 document #2679, July 2002.
3. C. Brislawn and P. Schelkens, "JP3D: Scope and requirements document, version 5.0." ISO/IEC JTC1/SC29/WG1 document #3022, 2003.
4. M. Gamito and M. Dias, "Variable resolution coding with JPEG 2000 Part 10," in *Applications of Digital Image Processing XXVII*, A. Tescher, ed., SPIE Proceedings, 2004.
5. M. Folk, A. Cheng, and K. Yates, "HDF5: A file format and i/o library for high performance computing applications," in *Proceedings of Supercomputing'99 (CD-ROM)*, ACM SIGARCH and IEEE, (Portland, OR), Nov. 1999.
6. P. Walatka and P. Buning, *PLOT3D User's Manual.* NASA Ames Research Center, March 1990. NASA Technical Memorandum 101067.
7. V. Engelson, D. Fritzon, and P. Fritzon, "Lossless compression of high-volume numerical data from simulations," in *Data Compression Conference 2000*, J. Storer and M. Cohn, eds., p. 574, IEEE Computer Society, March 2000.
8. M. Isenburg, P. Lindstrom, and J. Snoeyink, "Lossless compression of floating-point geometry." to appear in Proceedings of CAD2004, May 2004.
9. H. Tao and R. J. Moorhead, "Progressive transmission of scientific data using biorthogonal wavelet transform," in *Proceedings of the Conference on Visualization*, R. D. Bergeron and A. E. Kaufman, eds., pp. 93–99, IEEE Computer Society Press, Oct. 1994.
10. A. Trott, R. Moorhead, and J. McGinley, "Wavelets applied to lossless compression and progressive transmission of floating point data in 3-d curvilinear grids," in *Proceedings of the Conference om Visualization*, R. Yagel and G. Nielson, eds., pp. 385–388, IEEE Computer Society Press, October 1996.
11. X. Du, "Multiresolutional visualization of evolving distributed simulations using wavelets and MPI," Master's thesis, Mississippi State University. Department of Electrical and Computer Engineering, 1997.
12. B. Wohlberg and C. Brislawn, "JPEG 2000 Part 10: Floating point coding." ISO/IEC JTC1/SC29/WG1 document #2644, 2002.
13. B. Wohlberg and C. Brislawn, "Extending the JPEG 2000 image coding standard to support floating point data." ISO/IEC JTC1/SC29/WG1 document #3020, 2003.
14. D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys* **23**, pp. 5–48, Mar. 1990.
15. *ISO/IEC 15444-1. JPEG2000 Image Coding System*, 2000.
16. S. Li and W. Li, "Shape-adaptive discrete wavelet transforms for arbitrarily shaped visual object coding," *IEEE Trans. on Circuits and Systems for Video Technology* **10**, pp. 725–743, August 2000.
17. Stevenson, D., chairman, Floating-Point Working Group, Microprocessor Standards Subcommittee, "IEEE standard for binary floating point arithmetic (IEEE/ANSI 754-1985)," tech. rep., IEEE, 1985.
18. D. Taubman and M. Marcellin, *JPEG2000 Image Compression: Fundamentals, Standards and Practice*, Kluwer Academic Publishers, 2002.