ELSEVIER

Signal Processing: *Image Communication* 17 (2002) 3–48

SIGNAL PROCESSING: IMAGE COMMUNICATION

www.elsevier.com/locate/image

An overview of the JPEG 2000 still image compression standard

Majid Rabbani*, Rajan Joshi

Eastman Kodak Company, Rochester, NY 14650, USA

Abstract

In 1996, the JPEG committee began to investigate possibilities for a new still image compression standard to serve current and future applications. This initiative, which was named JPEG 2000, has resulted in a comprehensive standard (ISO 15444|ITU-T Recommendation T.800) that is being issued in six parts. Part 1, in the same vein as the JPEG baseline system, is aimed at minimal complexity and maximal interchange and was issued as an International Standard at the end of 2000. Parts 2–6 define extensions to both the compression technology and the file format and are currently in various stages of development. In this paper, a technical description of Part 1 of the JPEG 2000 standard is provided, and the rationale behind the selected technologies is explained. Although the JPEG 2000 standard only specifies the decoder and the codesteam syntax, the discussion will span both encoder and decoder issues to provide a better understanding of the standard in various applications.

Keywords: JPEG 2000; Image compression; Image coding; Wavelet compression

1. Introduction and background

The Joint Photographic Experts Group (JPEG) committee was formed in 1986 under the joint auspices of ISO and ITU-T[1] and was chartered with the "*digital compression and coding of continuous-tone still images*". The committee's first published standard [55,32], commonly known as the JPEG standard,[2] provides a toolkit of compression techniques from which applications can select various elements to satisfy particular requirements. This toolkit includes the following components: (i) the JPEG baseline system, which is a simple and efficient discrete cosine transform (DCT)-based lossy compression algorithm that uses Huffman coding, operates only in sequential mode, and is restricted to 8 bits/pixel input; (ii) an extended system, which introduces enhancements to the baseline algorithm to satisfy a broader set of applications; and (iii) a lossless mode, which is based on a predictive coding approach using either Huffman or arithmetic coding and is independent of the DCT. The JPEG baseline algorithm has since enjoyed widespread use in many digital imaging applications. This is due, in part, to its technical merits and status as a royalty-free international standard, but perhaps more so, it is

*Corresponding author.

E-mail address: majid.rabbani@kodak.com (M. Rabbani).

[1] Formerly known as the Consultative Committee for International Telephone and Telegraph (CCITT).

[2] Although JPEG Part 1 became an International Standard in 1993, the technical description of the algorithm was frozen as early as 1988.

0923-5965/02/$ - see front matter
PII: S0923-5965(01)00024-8

due to the free and efficient software that is available from the Independent JPEG Group (IJG) [57].

Despite the phenomenal success of the JPEG baseline system, it has several shortcomings that become increasingly apparent as the need for image compression is extended to such emerging applications as medical imaging, digital libraries, multimedia, internet and mobile. While the extended JPEG system addresses some of these shortcomings, it does so only to a limited extent and in some cases, the solutions are hindered by intellectual property rights (IPR) issues. The desire to provide a broad range of features for numerous applications in a single compressed bit-stream prompted the JPEG committee in 1996 to investigate possibilities for a new compression standard that was subsequently named JPEG 2000. In March 1997 a call for proposals was issued [58,59], seeking to produce a standard to "*address areas where current standards failed to produce the best quality or performance*", "*provide capabilities to markets that currently do not use compression*", and "*provide an open system approach to imaging applications*".

In November 1997, more than 20 algorithms were evaluated, and a wavelet decomposition approach was adopted as the backbone of the new standard. A comprehensive requirements document was developed that defined all the various application areas of the standard, along with a set of mandatory and optional requirements for each application. In the course of the ensuing three years, and after performing hundreds of technical studies known as "core experiments", the standard evolved into a state-of-the-art compression system with a diverse set of features, all of which are supported in a single compressed bit-stream.

The JPEG 2000 standard is scheduled to be issued in six parts. Part 1, in the same vein as the JPEG baseline system, defines a core coding system that is aimed at minimal complexity while satisfying 80% of the applications [60]. In addition, it defines an optional file format that includes essential information for the proper rendering of the image. It is intended to be available on a royalty and fee-free basis and was issued as an International Standard (IS) in December 2000. Parts 2–6 define extensions to both the compression technology and the file format and are in various stages of development. The history and the timeline of the various parts of the standard are shown in Table 1.

Part 2 is aimed at enhancing the performance of Part 1 with more advanced technology, possibly at the expense of higher complexity [61]. It is intended to serve those applications where maximal interchange is less important than meeting specific requirements. The codestream generated by part 2 encoders is usually not decodable by Part 1 decoders, and some of the technology in Part 2 might be protected by IPR. Part 3 defines motion JPEG 2000 (MJP2) and is primarily based on the technology in Part 1 with the addition of a file format [62]. It results in an encoder that is significantly less complex than the popular MPEG family of standards (due to lack of motion estimation) and provides full random access to the individually coded frames (albeit at the

Table 1
Timeline of the various parts of the JPEG 2000 standard[a]

Part	Title	CFP	WD	CD	FCD	FDIS	IS
1	JPEG 2000 image coding system: core coding system	97/03	99/03	99/12	00/03	00/10	00/12
2	JPEG 2000 image coding system: extensions	97/03	00/03	00/08	00/12	01/07	01/10
3	Motion JPEG 2000	99/12	00/07	00/12	01/03	01/07	01/10
4	Conformance testing	99/12	00/07	00/12	01/07	01/11	02/03
5	Reference software	99/12	00/03	00/07	00/12	01/08	01/11
6	Compound image file format	97/03	00/12	01/03	01/11	02/03	02/05

[a] CFP = Call for Proposals, WD = Working Draft, CD = Committee Draft, FCD = Final Committee Draft, FDIS = Final Draft International Standard, IS = International Standard.

expense of compression efficiency). It is intended for applications such as digital still cameras with burst capture mode, video editing in post-production environments, and digital cinema archive and distribution. Part 4 defines conformance testing [63], similar to the role of JPEG Part 2, to ensure a high-quality implementation of the standard. As mentioned earlier, a key factor in the JPEG baseline system's success as a widely used standard was the availability of efficient and free software. Part 5 defines a reference software implementation for Part 1 of the JPEG 2000 standard [64]. Currently, two implementations are available. One is a Java implementation by the JJ2000 group [65] consisting of Canon Research France, Ericsson and EPFL. The other is a C implementation by Image Power and University of British Columbia [2]. Finally, Part 6 defines a compound image file format for document scanning and fax applications [66].

It is noteworthy that the real incentive behind the development of the JPEG 2000 system was not just to provide higher compression efficiency compared to the baseline JPEG system. Rather, it was to provide a new image representation with a rich set of features, all supported within the same compressed bit-stream, that can address a variety of existing and emerging compression applications. In particular, the Part 1 of the standard addresses some of the shortcomings of baseline JPEG by supporting the following set of features:

- Improved compression efficiency.
- Lossy to lossless compression.
- Multiple resolution representation.
- Embedded bit-stream (progressive decoding and SNR scalability).
- Tiling.
- Region-of-interest (ROI) coding.
- Error resilience.
- Random codestream access and processing.
- Improved performance to multiple compression/decompression cycles.
- A more flexible file format.

The JPEG 2000 standard makes use of several recent advances in compression technology in order to achieve these features. For example, the low-complexity and memory efficient block DCT of JPEG has been replaced by the full-frame discrete wavelet transform (DWT). The DWT inherently provides a multi-resolution image representation while also improving compression efficiency due to good energy compaction and the ability to decorrelate the image across a larger scale. Furthermore, integer DWT filters can be used to provide both lossless and lossy compression within a single compressed bit-stream. Embedded coding is achieved by using a uniform quantizer with a central deadzone (with twice the step-size). When the output index of this quantizer is represented as a series of binary symbols, a partial decoding of the index is equivalent to using a quantizer with a scaled version of the original step-size, where the scaling factor is a power of two. To encode the binary bitplanes of the quantizer index, JPEG 2000 has replaced the Huffman coder of baseline JPEG with a context-based adaptive binary arithmetic coder with renormalization-driven probability estimation, known as the MQ coder. The embedded bit-stream that results from bitplane coding provides SNR scalability in addition to the capability of compressing to a target file size. Furthermore, the bitplanes in each subband are coded in independent rectangular blocks and in three fractional bitplane passes to provide an optimal embedded bit-stream, improved error resilience, partial spatial random access, ease of certain geometric manipulations, and an extremely flexible codestream syntax. Finally, the introduction of a canvas coordinate system facilitates certain operations in the compressed domain such as cropping, rotations by multiples of 90°, flipping, etc.

Several excellent review papers about JPEG 2000 Part 1 have recently appeared in the literature [3,13,16,18,27,37], and a comprehensive book describing all of the technical aspects of the standard has been published [45]. In this paper, a technical description of the fundamental building blocks of JPEG 2000 Part 1 is provided, and the rationale behind the selected technologies is explained. Although the JPEG 2000 standard only specifies the decoder and the codestream syntax, many specific decoder implementation issues have been omitted in our presentation in the interest of brevity. Instead, the emphasis has been placed on

general encoder and decoder technology issues to provide a better understanding of the standard in various applications. Therefore, readers who plan on implementing the standard should ultimately refer to the actual standard [60].

This paper is organized as follows. In Section 2, the fundamental building blocks of the JPEG 2000 Part 1 standard, such as pre-processing, DWT, quantization, and entropy coding are described. In Section 3, the syntax and organization of the compressed bit-stream is explained. In Section 4, various rate control strategies that can be used by the JPEG 2000 encoder for achieving an optimal SNR or visual quality for a given bit-rate are discussed. In Section 5, the tradeoffs between the various choices of encoder parameters are illustrated through an extensive set of examples. Finally, Section 6 contains a brief description of some additional JPEG 2000 features such as ROI, error resilience and file format, as well as a summary of the technologies used in Part 2.

2. JPEG 2000 fundamental building blocks

The fundamental building blocks of a typical JPEG 2000 encoder are shown in Fig. 1. These components include pre-processing, DWT, quantization, arithmetic coding (tier-1 coding), and bit-stream organization (tier-2 coding). In the following, each of these components is discussed in more detail.

The input image to JPEG 2000 may contain one or more components. Although a typical color image would have three components (e.g., RGB or YC_bC_r), up to 16 384 (2^{14}) components can be specified for an input image to accommodate multi-spectral or other types of imagery. The sample values for each component can be either signed or unsigned integers with a bit-depth in the range of 1–38 bits. Given a sample with a bit-depth of B bits, the unsigned representation would correspond to the range $(0, 2^B-1)$, while the signed representation would correspond to the range $(-2^{B-1}, 2^{B-1}-1)$. The bit-depth, resolution, and signed versus unsigned specification can vary for each component. If the components have different bit-depths, the most significant bits of the components should be aligned to facilitate distortion estimation at the encoder.

2.1. Pre-processing

The first step in pre-processing is to partition the input image into rectangular and non-overlapping tiles of equal size (except possibly for those tiles at the image borders). The tile size is arbitrary and can be as large as the original image itself (i.e., only one tile) or as small as a single pixel. Each tile is compressed independently using its own set of specified compression parameters. Tiling is particularly useful for applications where the amount of available memory is limited compared to the image size.

Next, unsigned sample values in each component are level shifted (DC offset) by subtracting a fixed value of 2^{B-1} from each sample to make its value symmetric around zero. Signed sample values are not level shifted. Similar to the level shifting performed in the JPEG standard, this operation simplifies certain implementation issues (e.g., numerical overflow, arithmetic coding context specification, etc.), but has no effect on the coding efficiency. Part 2 of the JPEG 2000 standard allows for a generalized DC offset, where a user-defined offset value can be signaled in a marker segment.

Finally, the level-shifted values can be subjected to a forward point-wise intercomponent transformation to decorrelate the color data. One restriction on applying the intercomponent transformation is that the components must have identical

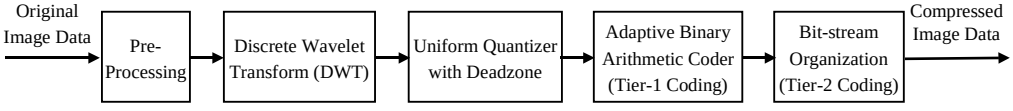


Fig. 1. JPEG 2000 fundamental building blocks.

bit-depths and dimensions. Two transform choices are allowed in Part 1, where both transforms operate on the first three components of an image tile with the implicit assumption that these components correspond to red–green–blue (RGB). One transform is the *irreversible color transform* (ICT), which is identical to the traditional RGB to YC_bC_r color transformation and can only be used for lossy coding. The forward ICT is defined as

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.16875 & -0.33126 & 0.500 \\ 0.500 & -0.41869 & -0.08131 \end{pmatrix} \times \begin{pmatrix} R \\ G \\ B \end{pmatrix}. \quad (1)$$

This can alternatively be written as

$$Y = 0.299(R-G) + G + 0.114(B-G),$$

$$C_b = 0.564(B-Y), \quad C_r = 0.713(R-Y),$$

while the inverse ICT is given by

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.0 & 0 & 1.402 \\ 1.0 & -0.34413 & 0.71414 \\ 1.0 & 1.772 & 0 \end{pmatrix} \times \begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix}. \quad (2)$$

The other transform is the *reversible color transform* (RCT), which is a reversible integer-to-integer transform that approximates the ICT for color decorrelation and can be used for both lossless and lossy coding. The forward RCT is defined as

$$Y = \left\lfloor \frac{R+2G+B}{4} \right\rfloor, \quad U = R-G, \quad V = B-G, \quad (3)$$

where $\lfloor w \rfloor$ denotes the largest integer that is smaller than or equal to w. The Y component has the same bit-depth as the RGB components while the U and V components have one extra bit of precision. The inverse RCT, which is capable of exactly recovering the original RGB data, is given by

$$G = Y - \left\lfloor \frac{U+V}{4} \right\rfloor, \quad R = U+G, \quad B = V+G. \quad (4)$$

At the decoder, the decompressed image is subjected to the corresponding inverse color transform if necessary, followed by the removal of the DC level shift. Since each component of each tile is treated independently, the basic compression engine for JPEG 2000 will only be discussed with reference to a single tile of a monochrome image.

2.2. The discrete wavelet transform (DWT)

The block DCT transformation in baseline JPEG has been replaced with the full frame DWT in JPEG 2000. The DWT has several characteristics that make it suitable for fulfilling some of the requirements set forth by the JPEG 2000 committee. For example, a multi-resolution image representation is inherent to DWT. Furthermore, the full-frame nature of the transform decorrelates the image across a larger scale and eliminates blocking artifacts at high compression ratios. Finally, the use of integer DWT filters allows for both lossless and lossy compression within a single compressed bit-stream. In the following, we first consider a one-dimensional (1-D) DWT for simplicity, and then extend the concepts to two dimensions.

2.2.1. The 1-D DWT

The forward 1-D DWT at the encoder is best understood as successive applications of a pair of low-pass and high-pass filters, followed by down-sampling by a factor of two (i.e., discarding odd indexed samples) after each filtering operation as shown in Fig. 2. The low-pass and high-pass filter pair is known as *analysis filter-bank*. The low-pass

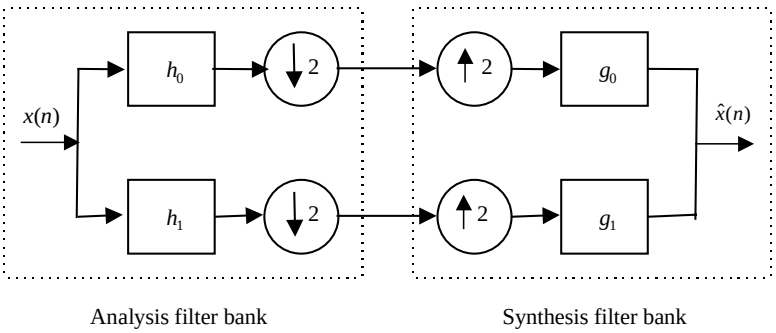


Fig. 2. 1-D, 2-band wavelet analysis and synthesis filter-bank.

filter preserves the low frequencies of a signal while attenuating or eliminating the high frequencies, thus resulting in a blurred version of the original signal. Conversely, the high-pass filter preserves the high frequencies in a signal such as edges, texture and detail, while removing or attenuating the low frequencies.

Consider a 1-D signal $x(n)$ (such as the pixel values in a row of an image) and a pair of low-pass and high-pass filters designated by $h_0(n)$ and $h_1(n)$, respectively. An example of a low-pass filter is $h_0(n) = (-1\ 2\ 6\ 2\ -1)/8$, which is symmetric and has five integer coefficients (or taps). An example of a high-pass filter is $h_1(n) = (-1\ 2\ -1)/2$, which is symmetric and has three integer taps. The analysis filter-bank used in this example was first proposed in [17] and is often referred to as the $(5, 3)$ filter-bank, indicating a low-pass filter of length five and a high-pass filter of length three. To ensure that the filtering operation is defined at the signal boundaries, the 1-D signal must be extended in both directions. When using odd-tap filters, the signal is symmetrically and periodically extended as shown in Fig. 3. The extension for even-tap filters (allowed in Part 2 of the standard) is more complicated and is explained in Part 2 of the standard document [61].

The filtered samples that are output from the forward DWT are referred to as *wavelet coefficients*. Because of the downsampling process, the total number of wavelet coefficients is the same as the number of original signal samples. When the DWT decomposition is applied to sequences with an odd number of samples, either the low pass or the high-pass sequence will have one additional sample to maintain the same number of coefficients as original samples. In JPEG 2000, this choice is dictated by the positioning of the input signal with respect to the canvas coordinate system, which will be discussed in Section 3. The (h_0, h_1) filter pair is designed in such a manner that after downsampling the output of each filter by a factor of two, the original signal can still be completely recovered from the remaining samples in the absence of any quantization errors. This is referred to as the *perfect reconstruction* (PR) property.

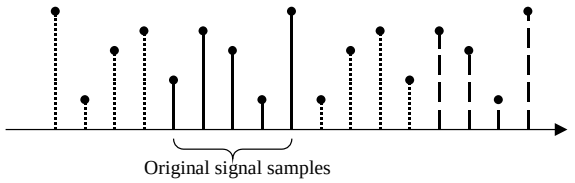


Fig. 3. Symmetric and periodic extension of the input signal at boundaries.

Reconstruction from the wavelet coefficients at the decoder is performed with another pair of low-pass and high-pass filters (g_0, g_1), known as the *synthesis filter-bank*. Referring to Fig. 2, the downsampled output of the low-pass filter $h_0(n)$ is first upsampled by a factor of two by inserting zeroes in between every two samples. The result is then filtered with the synthesis low-pass filter, $g_0(n)$. The downsampled output of the high-pass filter $h_1(n)$ is also upsampled and filtered with the synthesis high-pass filter, $g_1(n)$. The results are added together to produce a reconstructed signal $\hat{x}(n)$,

which assuming sufficient precision, will be identical to $x(n)$ because of the PR property.

For perfect reconstruction, the analysis and synthesis filters have to satisfy the following two conditions:

$$H_0(z)G_0(z) + H_1(z)G_1(z) = 2, \tag{5}$$

$$H_0(-z)G_0(z) + H_1(-z)G_1(z) = 0, \tag{6}$$

where $H_0(z)$ is the Z-transform of $h_0(n)$, $G_0(z)$ is the Z-transform of $g_0(n)$, etc. The condition in Eq. (6) can be satisfied by choosing

$$G_0(z) = -cz^{-l}H_1(-z) \quad \text{and} \quad G_1(z) = cz^{-l}H_0(-z), \tag{7}$$

where l is an integer constant and c is a scaling factor. Combining this result with Eq. (5) indicates that the analysis filter pair (h_0, h_1) has to be chosen to satisfy

$$-cz^{-l}H_0(z)H_1(-z) + cz^{-l}H_1(z)H_0(-z) = 2. \tag{8}$$

The constant l represents a delay term that imposes a restriction on the spatial alignment of the analysis and synthesis filters, while the constant c affects the filter normalization. The filter-bank that satisfies these conditions is known as the *bi-orthogonal* filter-bank. This name stems from the fact that h_0 and g_1 are orthogonal to each other and h_1 and g_0 are orthogonal to each other. A particular class of bi-orthogonal filters is one where the analysis and synthesis filters are FIR and linear phase (i.e., they satisfy certain symmetry conditions) [47]. Then, it can be shown that in order to satisfy Eq. (8), the analysis filters h_0 and h_1 have to be of unequal lengths. If the filters have an odd number of taps, their length can differ only by an odd multiple of two.

While the (5, 3) filter-bank is a prime example of a bi-orthogonal filter-bank with integer taps, the filter-banks that result in the highest compression efficiency often have floating point taps [48]. The most well-known filter-bank in this category is the Daubechies (9, 7) filter-bank, introduced in [5] and characterized by the filter taps given in Table 2. For comparison, the analysis and synthesis filter taps for the integer (5, 3) filter-bank are specified in Table 3. It can be easily verified that these filters satisfy Eqs. (7) and (8) with $l = 1$ and $c = 1.0$. As is evident from Tables 2 and 3, the filter h_0 is centered at zero while h_1 is centered at -1. As a result, the downsampling operation effectively retains the even indexed samples from the low-pass output and the odd indexed samples from the high-pass output sequence.

After the 1-D signal has been decomposed into two bands, the low-pass output is still highly correlated and can be subjected to another stage of two-band decomposition to achieve additional decorrelation. In comparison, there is generally little to be gained by further decomposing the high-pass output. In most DWT decompositions, only the low-pass output is further decomposed to

Table 2
Analysis and synthesis high-pass filter taps for floating point Daubechies (9, 7) filter-bank

n	Low-pass, $h_0(n)$	Low-pass, $g_0(n)$
0	+0.602949018236360	+1.115087052457000
±1	+0.266864118442875	+0.591271763114250
±2	−0.078223266528990	−0.057543526228500
±3	−0.016864118442875	−0.091271763114250
±4	+0.026748757410810	

n	High-pass, $h_1(n)$	n	High-pass, $g_1(n)$
−1	+1.115087052457000	1	+0.602949018236360
−2, 0	−0.591271763114250	0, 2	−0.266864118442875
−3, 1	−0.057543526228500	−1, 3	−0.078223266528990
−4, 2	+0.091271763114250	−2, 4	+0.016864118442875
		−3, 5	+0.026748757410810

Table 3
Analysis and synthesis filter taps for the integer (5, 3) filter-bank

n	$h_0(n)$	$g_0(n)$	n	$h_1(n)$	n	$g_1(n)$
0	3/4	+1	−1	+1	1	+3/4
±1	1/4	+1/2	−2, 0	−1/2	0, 2	−1/4
±2	−1/8				−1, 3	−1/8

produce what is known as a *dyadic* or *octave* decomposition. Part 1 of the JPEG 2000 standard supports only dyadic decompositions, while Part 2 also allows for the further splitting of the high-frequency bands.

2.2.2. The 2-D DWT

The 1-D DWT can be easily extended to two dimensions (2-D) by applying the filter-bank in a separable manner. At each level of the wavelet decomposition, each row of a 2-D image is first transformed using a 1-D horizontal analysis filter-bank (h_0, h_1). The same filter-bank is then applied vertically to each column of the filtered and subsampled data. The result of a one-level wavelet decomposition is four filtered and subsampled images, referred to as *subbands*. Given the linear nature of the filtering process, the order in which the horizontal and the vertical filters are applied does not affect the final values of the 2-D subbands. In a 2-D dyadic decomposition, the lowest frequency subband (denoted as the LL band to indicate low-pass filtering in both directions) is further decomposed into four smaller subbands, and this process may be repeated until no tangible gains in compression efficiency can be achieved. Fig. 4 shows a 3-level, 2-D dyadic decomposition and the corresponding labeling for each subband. For example, the subband label kHL indicates that a horizontal high-pass (H) filter has been applied to the rows, followed by a vertical low-pass (L) filter applied to the columns during the kth level of the DWT decomposition. As a convention, the subband 0LL refers to the original image (or image tile). Fig. 5 shows a 3-level, 2-D DWT decomposition of the Lena image using the (9, 7) filter-bank as specified in Table 2, and it clearly demonstrates the energy compaction property of the DWT (i.e., most of the image energy is found in the lower frequency subbands). To better visualize the subband energies, the AC subbands (i.e., all the subbands except for LL) have been scaled up by a factor of four. However, as will be explained in Section 2.2.3, in order to show the actual contribution of each subband to the overall image energy, the wavelet coefficients in each

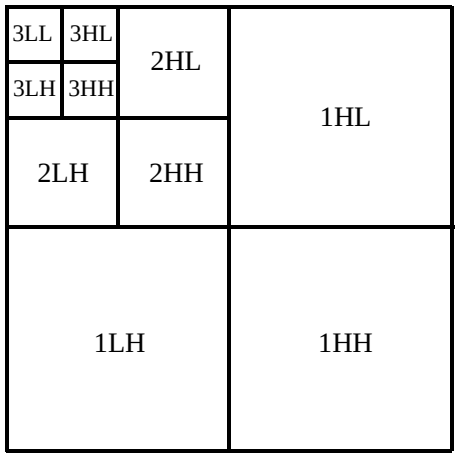


Fig. 4. 2-D, 3-level wavelet decomposition.

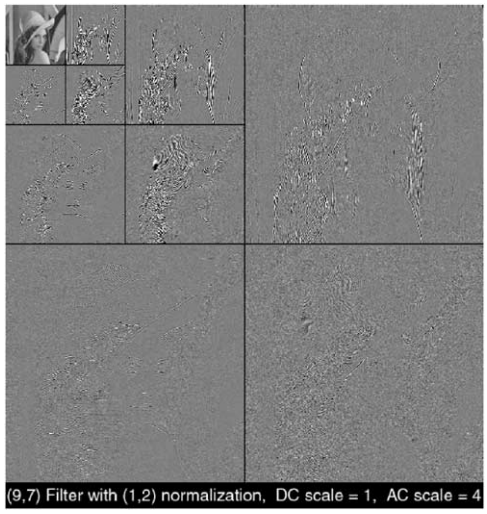


Fig. 5. 2-D, 3-level wavelet decomposition of Lena using the (9, 7) filter-bank.

Table 4
L_2-norms of the DWT subbands after a 2-D, 3-level wavelet decomposition

Subband	$(\sqrt{2}, \sqrt{2})$ normalization		(1, 2) normalization	
	(5, 3) filter-bank	(9, 7) filter-bank	(5, 3) filter-bank	(9, 7) filter-bank
3LL	0.67188	1.05209	5.37500	8.41675
3HL	0.72992	1.04584	2.91966	4.18337
3LH	0.72992	1.04584	2.91966	4.18337
3HH	0.79297	1.03963	1.58594	2.07926
2HL	0.79611	0.99841	1.59222	1.99681
2LH	0.79611	0.99841	1.59222	1.99681
2HH	0.92188	0.96722	0.92188	0.96722
1HL	1.03833	1.01129	1.03833	1.01129
1LH	1.03833	1.01129	1.03833	1.01129
1HH	1.43750	1.04044	0.71875	0.52022

subband should be scaled by the weights given in the last column of Table 4.

The DWT decomposition provides a natural solution for the multiresolution requirement of the JPEG 2000 standard. The lowest resolution at which the image can be reconstructed is referred to as resolution zero. For example, referring to Fig. 4, the 3LL subband would correspond to resolution zero for a 3-level decomposition. For an N_L-level[3] DWT decomposition, the image can be reconstructed at $N_L + 1$ resolutions. In general, to reconstruct an image at resolution r $(r > 0)$, subbands $(N_L - r + 1)$HL, $(N_L - r + 1)$LH and $(N_L - r + 1)$HH need to be combined with the image at resolution $(r - 1)$. These subbands are referred to as belonging to resolution r. Resolution zero consists of only the N_LLL band. If the subbands are encoded independently, the image can be reconstructed at any resolution level by simply decoding those portions of the codestream that contain the subbands corresponding to that resolution and all the previous resolutions. For example, referring to Fig. 4, the image can be reconstructed at resolution two by combining the resolution one image and the three subbands labeled 2HL, 2LH and 2HH.

[3] N_L is the notation that is used in the JPEG 2000 document to indicate the number of resolution levels, although the subscript L might be somewhat confusing as it would seem to indicate a variable.

2.2.3. Filter normalization

The output of an invertible forward transform can generally have any arbitrary normalization (scaling) as long as it is undone by the inverse transform. In case of DWT filters, the analysis filters h_0 and h_1 can be normalized arbitrarily. Referring to Eq. (8), the normalization chosen for the analysis filters will influence the value of c, which in turn determines the normalization of the synthesis filters, g_0 and g_1. The normalization of the DWT filters is often expressed in terms of the DC gain of the low-pass analysis filter h_0, and the Nyquist gain of the high-pass analysis filter h_1. The DC gain and the Nyquist gain of a filter $h(n)$, denoted by G_{DC} and $G_{Nyquist}$, respectively, are defined as

$$G_{DC} = \left|\sum_n h(n)\right|, \quad G_{Nyquist} = \left|\sum_n (-1)^n h(n)\right|. \quad (9)$$

The (9, 7) and the (5, 3) analysis filter-banks as defined in Tables 2 and 3 have been normalized so that the low-pass filter has a DC gain of 1 and the high-pass filter has a Nyquist gain of 2. This is referred to as the (1, 2) normalization and it is the one adopted by Part 1 of the JPEG 2000 standard. Other common normalizations that have appeared in the literature are $(\sqrt{2}, \sqrt{2})$ and (1, 1). Once the normalization of the analysis filter-bank has been specified, the normalization of the synthesis filter-bank is automatically determined by reversing the order and multiplying by the scalar constant c of Eq. (8).

In the existing JPEG standard, the scaling of the forward DCT is defined to create an *orthonormal* transform, which has the property that the sum of the squares of the image samples is equal to the sum of the squares of the transform coefficients (Parseval's theorem). Furthermore, the orthonormal normalization of the DCT has the useful property that the mean-squared error (MSE) of the quantized DCT coefficients is the same as the MSE of the reconstructed image. This provides a simple means for quantifying the impact of coefficient quantization on the reconstructed

image MSE. Unfortunately, this property does not hold for a DWT decomposition.

Each wavelet coefficient in a 1-D DWT decomposition can be associated with a basis function. The reconstructed signal $\hat{x}(n)$ can be expressed as a weighted sum of these basis functions, where the weights are the wavelet coefficients (either quantized or unquantized). Let $\psi_m^b(n)$ denote the basis function corresponding to a coefficient $y_b(m)$, the mth wavelet coefficient from subband b. Then,

$$\hat{x}(n) = \sum_b \sum_m y_b(m)\psi_m^b(n). \tag{10}$$

For a simple 1-level DWT, the basis functions for the wavelet coefficients in the low-pass or the high-pass subbands are shifted versions of the corresponding low-pass or high-pass synthesis filters, except near the subband boundaries. In general, the basis functions of a DWT decomposition are not orthogonal; hence, Parseval's theorem does not apply. Woods and Naveen [50] have shown that for quantized wavelet coefficients under certain assumptions on the quantization noise, the MSE of the reconstructed image can be 10 approximately expressed as a weighted sum of the MSE of the wavelet coefficients, where the weight for subband b is

$$\alpha_b^2 = \sum_n \left|\psi^b(n)\right|^2. \tag{11}$$

The coefficient α_b is referred to as the *L_2-norm*[4] for subband b. For an orthonormal transform, all the α_b values would be unity. The knowledge of the L_2-norms is essential for the encoder, because they represent the contribution of the quantization noise of each subband to the overall MSE and are a key factor in designing quantizers or prioritizing the quantized data for coding.

The DWT filter normalization impacts both the L_2-norm and the dynamic range of each subband. Given the normalization of the 1-D analysis filter-bank, the *nominal* dynamic range of the 2-D subbands can be easily determined in terms of the bit-depth of the source image sample, R_I. In particular, for the $(1,2)$ normalization, the kLL subband will have a nominal dynamic range of R_I bits. However, the *actual* dynamic range might be slightly larger. In JPEG 2000, this situation is handled by using guard bits to avoid the overflow of the subband value. For the $(1,2)$ normalization, the nominal dynamic ranges of the kLH and kHL subbands are $R_I + 1$, while that of the kHH subband is $R_I + 2$.

Table 4 shows the L_2-norms of the DWT subbands after a 3-level decomposition with either the $(9,7)$ or the $(5,3)$ filter-bank and using either the $(\sqrt{2},\sqrt{2})$ or the $(1,2)$ filter normalization. Clearly, the $(\sqrt{2},\sqrt{2})$ normalization results in a DWT that is closer to an orthonormal transform (especially for the $(9,7)$ filter-bank), while the $(1,2)$ normalization avoids the dynamic range expansion at each level of the decomposition.

2.2.4. DWT implementation issues and the lifting scheme

In the development of the existing DCT-based JPEG standard, great emphasis was placed on the implementation complexity of the encoder and decoder. This included issues such as memory requirements, number of operations per sample, and amenability to hardware or software implementation, e.g., transform precision, parallel processing, etc. The choice of the 8×8 block size for the DCT was greatly influenced by these considerations.

In contrast to the limited buffering required for the 8×8 DCT, a straightforward implementation of the 2-D DWT decomposition requires the storage of the entire image in memory. The use of small tiles reduces the memory requirements without significantly affecting the compression efficiency (see Section 5.1.1). In addition, some clever designs for line-based processing of the DWT have been published that substantially reduce the memory requirements depending on the size of the filter kernels [14]. Recently, an alternative implementation of the DWT has been proposed, known as the *lifting scheme* [15,41–43]. In addition to providing a significant reduction in the memory and the computational complexity of the DWT, lifting provides in-place computation of the wavelet coefficients by overwriting the memory

[4] We have ignored the fact that, in general, the L_2-norm for the coefficients near the subband boundaries are slightly different than the rest of the coefficients in the subband.

locations that contain the input sample values. The wavelet coefficients computed with lifting are identical to those computed by a direct filter-bank convolution, in much the same manner as a fast Fourier transform results in the same DFT coefficients as a brute force approach. Because of these advantages, the specification of the DWT kernels in JPEG 2000 is only provided in terms of the lifting coefficients and not the convolutional filters.

The lifting operation consists of several steps. The basic idea is to first compute a trivial wavelet transform, also referred to as the *lazy* wavelet transform, by splitting the original 1-D signal into odd and even indexed subsequences, and then modifying these values using alternating prediction and updating steps. Fig. 6 depicts an example of the lifting steps corresponding to the integer (5, 3) filter-bank. The sequences $\{s_i^0\}$ and $\{d_i^0\}$ denote the even and odd sequences, respectively, resulting from the application of the lazy wavelet transform to the input sequence.

In JPEG 2000, a *prediction* step consists of predicting each odd sample as a linear combination of the even samples and subtracting it from the odd sample to form the prediction error $\{d_i^1\}$. Referring to Fig. 6, for the (5, 3) filter-bank, the prediction step consists of predicting $\{d_i^0\}$ by averaging the two neighboring even sequence pixels and subtracting the average from the odd sample value, i.e.,

$$d_i^1 = d_i^0 - \tfrac{1}{2}(s_i^0 + s_{i+1}^0). \tag{12}$$

Due to the simple structure of the (5, 3) filter-bank, the output of this stage, $\{d_i^1\}$, is actually the high-pass output of the DWT filter. In general, the number of even pixels employed in the prediction and the actual weights applied to the samples depend on the specific DWT filter-bank.

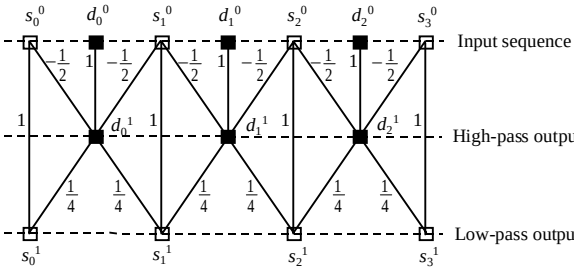


Fig. 6. Lifting prediction/update steps for the (5, 3) filter-bank.

An *update* step consists of updating the even samples by adding to them a linear combination of the already modified odd samples, $\{d_i^1\}$, to form the updated sequence $\{s_i^1\}$. Referring to Fig. 6, for the (5, 3) filter-bank, the update step consists of the following:

$$s_i^1 = s_i^0 + \tfrac{1}{4}(d_{i-1}^1 + d_i^1). \tag{13}$$

For the (5, 3) filter-bank, the output of this stage, $\{s_i^1\}$, is actually the low-pass output of the DWT filter. Again, the number of odd pixels employed in the update and the actual weights applied to each sample depend on the specific DWT filter-bank. The prediction and update steps are generally iterated N times, with different weights used at each iteration. This can be summarized as

$$d_i^n = d_i^{n-1} + \sum_k P_n(k) s_k^{n-1}, \quad n \in [1, 2, \ldots, N], \tag{14}$$

$$s_i^n = s_i^{n-1} + \sum_k U_n(k) d_k^n, \quad n \in [1, 2, \ldots, N], \tag{15}$$

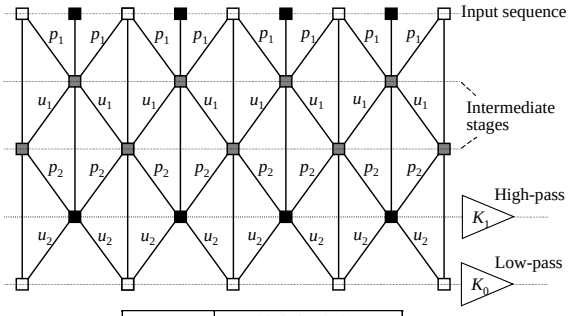
where $P_n(k)$ and $U_n(k)$ are, respectively, the prediction and update weights at the nth iteration. For the (5, 3) filter-bank $N = 1$, while for the Daubechies (9, 7) filter-bank, $N = 2$. The output of the final prediction step will be the high-pass coefficients up to a scaling factor K_1, while the output of the final update step will be the low-pass coefficients up to a scaling constant K_0. For the (5, 3) filter-bank, $K_0 = K_1 = 1$. The lifting steps corresponding to the (9, 7) filter-bank (as specified in Table 2) are shown in Fig. 7. The general block diagram of the lifting process is shown in Fig. 8.

A nice feature of the lifting scheme is that it makes the construction of the inverse transform straightforward. Referring to Fig. 8 and working from right to left, first the low-pass and high-pass wavelet coefficients are scaled by $1/K_0$ and $1/K_1$ to produce $\{s_i^N\}$ and $\{d_i^N\}$. Next, $\{d_i^N\}$ is taken through the update stage $U_N(z)$ and subtracted from $\{s_i^N\}$ to produce $\{s_i^{N-1}\}$. This process continues, where each stage of the prediction and update is undone in the reverse order that it was

constructed at the encoder until the image samples have been reconstructed.

2.2.5. Integer-to-integer transforms

Although the input image samples to JPEG 2000 are integers, the output wavelet coefficients are floating point when using floating point DWT filters. Even when dealing with integer filters such as the (5, 3) filter-bank, the precision required for achieving mathematically lossless performance increases significantly with every level of the wavelet decomposition and can quickly become unmanageable. An important advantage of the lifting approach is that it can provide a convenient framework for constructing integer-to-integer DWT filters from any general filter specification [1,10].



p_1	- 1.586134342059924
u_1	- 0.052980118572961
p_2	+0.882911075530934
u_2	+0.443506852043971
$K_1 = 1/K_0$	+1.230174104914001

Fig. 7. Lifting steps for the (9, 7) filter-bank.

This can be best understood by referring to Fig. 9, where quantizers are inserted immediately after the calculation of the prediction and the update terms but before modifying the odd or the even sample value. The quantizer typically performs an operation such as truncation or rounding to the nearest integer, thus creating an integer-valued output. If the values of K_0 and K_1 are approximated by rational numbers, it is easy to verify that the resulting system is mathematically invertible despite the inclusion of the quantizer. If the underlying floating point filter uses the (1, 2) normalization and $K_0 = K_1 = 1$ (as is the case for the (5, 3) filter-bank), the final low-pass output will have roughly the same bit precision as that of the

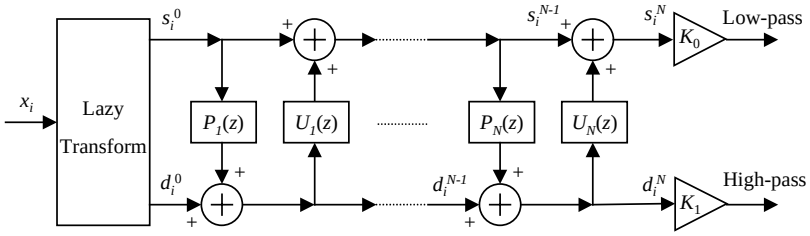


Fig. 8. General block diagram of the lifting process.

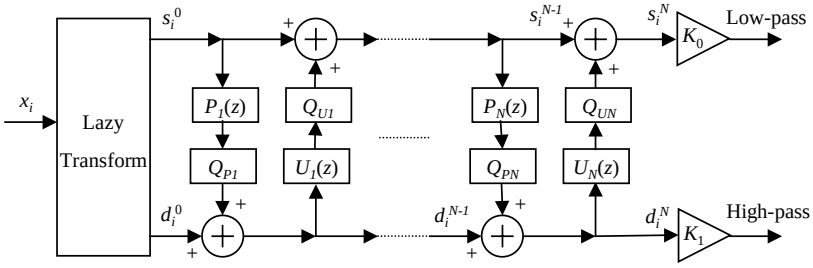


Fig. 9. General block diagram of a forward integer-to-integer transform using lifting.

input sample while the high-pass output will have an extra bit of precision. This is because input samples with a large enough dynamic range (e.g., 8 bits or higher), rounding at each lifting step have a negligible effect on the nominal dynamic range of the output.

As described in the previous section, the inverse transformation is simply performed by undoing all the prediction and update steps in the reverse order that they were performed at the encoder. However, the resulting integer-to-integer transform is nonlinear and hence when extended to two dimensions, the order in which the transformation is applied to the rows or the columns will impact the final output. To recover the original sample values losslessly, the inverse transform must be applied in exactly the reverse row–column order of the forward transform. An extensive performance evaluation and analysis of reversible integer-to-integer DWT for image compression has been published in [1].

As an example, consider the conversion of the $(5,3)$ filter-bank into an integer-to-integer transform by adding the two quantizers $Q_{\mathrm{P1}}(w) = -\lfloor -w \rfloor$ and $Q_{\mathrm{U1}}(w) = \lfloor w + 1/2 \rfloor$ to the prediction and update steps, respectively, in the lifting diagram of Fig. 6. The resulting forward transform is given by

$$\begin{aligned} y(2n+1) &= x(2n+1) - \left\lfloor \frac{x(2n) + x(2n+2)}{2} \right\rfloor, \\ y(2n) &= x(2n) + \left\lfloor \frac{y(2n-1) + y(2n+1) + 2}{4} \right\rfloor. \end{aligned} \tag{16}$$

The required precision for the low-pass band stays roughly the same as the original sample while the precision of the high-pass band grows by one bit. The inverse transform, which losslessly recovers the original sample values, is given by

$$\begin{aligned} x(2n) &= y(2n) - \left\lfloor \frac{y(2n-1) + y(2n+1) + 2}{4} \right\rfloor, \\ x(2n+1) &= y(2n+1) - \left\lfloor \frac{x(2n) + x(2n+2)}{2} \right\rfloor. \end{aligned} \tag{17}$$

2.2.6. DWT filter choices in JPEG 2000 Part 1

Part 1 of the JPEG 2000 standard has adopted only two choices for the DWT filters. One is the Daubechies $(9,7)$ floating point filter-bank (as specified in Table 2), which has been chosen for its superior lossy compression performance. The other is the lifted integer-to-integer $(5,3)$ filter-bank, also referred to as the *reversible* $(5,3)$ *filter-bank*, as specified in Eqs. (16) and (17). This choice was driven by requirements for low implementation complexity and lossless capability. The performance of these filters is compared in Section 5.1.3. Part 2 of the standard allows for arbitrary filter specifications in the codestream, including filters with an even number of taps.

2.3. Quantization

The JPEG baseline system employs a uniform quantizer and an inverse quantization process that reconstructs the quantized coefficient to the midpoint of the quantization interval. A different step-size is allowed for each DCT coefficient to take advantage of the sensitivity of the human visual system (HVS), and these step-sizes are conveyed to the decoder via an 8×8 quantization table (q-table) using one byte per element. The quantization strategy employed in JPEG 2000 Part 1 is similar in principle to that of JPEG, but it has a few important differences to satisfy some of the JPEG 2000 requirements.

One difference is in the incorporation of a central deadzone in the quantizer. It was shown in [40] that the R–D optimal quantizer for a continuous signal with Laplacian probability density (such as DCT or wavelet coefficients) is a uniform quantizer with a central deadzone. The size of the optimal deadzone as a fraction of the step-size increases as the variance of the Laplacian distribution decreases; however, it always stays less than two and is typically closer to one. In Part 1, the deadzone has twice the quantizer step-size as depicted in Fig. 10, while in Part 2, the size of the deadzone can be parameterized to have a different value for each subband.

Part 1 adopted the deadzone with twice the step-size due to its optimal embedded structure. Briefly, this means that if an M_b-bit quantizer index resulting from a step-size of Δ_b is transmitted progressively starting with the most significant bit (MSB) and proceeding to the least significant bit

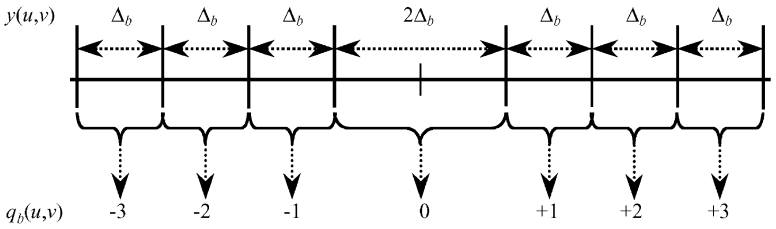


Fig. 10. Uniform quantizer with deadzone with step-size Δ_b.

(LSB), the resulting index after decoding only N_b bits is identical to that obtained by using a similar quantizer with a step-size of $\Delta_b 2^{M_b-N_b}$. This property allows for *SNR scalability*, which in its optimal sense means that the decoder can cease decoding at any truncation point in the codestream and still produce exactly the same image that would have been encoded at the bit-rate corresponding to the truncated codestream. This property also allows a target bit-rate or a target distortion to be achieved exactly, while the current JPEG standard generally requires multiple encoding cycles to achieve the same goal. This allows an original image to be compressed with JPEG 2000 to the highest quality required by a given set of clients (through the proper choice of the quantization step-sizes) and then disseminated to each client according to the specific image quality (or target filesize) requirement without the need to decompress and recompress the existing codestream. Importantly, the codestream can also be reorganized in other ways to meet the various requirements of the JPEG 2000 standard as will be described in Section 3.

Another difference is that the inverse quantization of JPEG 2000 explicitly allows for a reconstruction bias from the quantizer midpoint for non-zero indices to accommodate the skewed probability distribution of the wavelet coefficients. In JPEG baseline, a simple biased reconstruction strategy has been shown to improve the decoded image PSNR by about 0.25 dB [34]. Similar gains can be expected with the biased reconstruction of wavelet coefficients in JPEG 2000. The exact operation of the quantization and inverse quantization is explained in more detail in the following sections.

2.3.1. Quantization at the encoder

For each subband b, a basic quantizer step-size Δ_b is selected by the user and is used to quantize all the coefficients in that subband. The choice of Δ_b can be driven by the perceptual importance of each subband based on HVS data [4,19,31,49], or it can be driven by other considerations such as rate control. The quantizer maps a wavelet coefficient $y_b(u,v)$ in subband b to a quantized index value $q_b(u,v)$, as shown in Fig. 10. The quantization operation is an encoder issue and can be implemented in any desired manner. However, it is most efficiently performed according to

$$q_b(u,v) = \mathrm{sign}(y_b(u,v)) \left\lfloor \frac{|y_b(u,v)|}{\Delta_b} \right\rfloor. \tag{18}$$

The step-size Δ_b is represented with a total of two bytes; an 11-bit mantissa μ_b, and a 5-bit exponent ε_b, according to the relationship

$$\Delta_b = 2^{R_b-\varepsilon_b}\left(1+\frac{\mu_b}{2^{11}}\right), \tag{19}$$

where R_b is the number of bits representing the nominal dynamic range of the subband b, which is explained in Section 2.2.3. This limits the largest possible step-size to about twice the dynamic range of the input sample (when μ_b has its maximum value and $\varepsilon_b = 0$), which is sufficient for all practical cases of interest. When the reversible $(5,3)$ filter-bank is used, Δ_b is set to one by choosing $\mu_b = 0$ and $\varepsilon_b = R_b$. The quantizer index $q_b(u,v)$ will have M_b bits if fully decoded, where $M_b = G + \varepsilon_b - 1$. The parameter G is the number of guard bits signaled to the decoder, and it is typically one or two.

Two modes of signaling the value of Δ_b to the decoder are possible. In one mode, which is similar to the q-table specification used in the current

JPEG, the (ε_b, μ_b) value for every subband is explicitly transmitted. This is referred to as *expounded quantization*. The values can be chosen to take into account the HVS properties and/or the L_2-norm of each subband in order to align the bitplanes of the quantizer indices according to their true contribution to the MSE. In another mode, referred to as *derived quantization*, a single value (ε_0, μ_0) is sent for the LL subband and the (ε_b, μ_b) values for each subband are derived by scaling the Δ_0 value by some power of two depending on the level of decomposition associated with that subband. In particular,

$$(\varepsilon_b, \mu_b) = (\varepsilon_0 - N_L + n_b, \mu_0), \tag{20}$$

where N_L is the total number of decomposition levels and n_b is the decomposition level corresponding to subband b. It is easy to show that Eq. (20) scales the step-sizes for each subband according to a power of two that best approximates the L_2-norm of a subband relative to the LL band (refer to Table 4). This procedure approximately aligns the quantized subband bitplanes according to their proper MSE contribution.

2.3.2. Inverse quantization at the decoder

When the irreversible (9, 7) filter-bank is used, the reconstructed transform coefficient, $Rq_b(u, v)$, for a quantizer step-size of Δ_b is given by

$$Rq_b(u,v) = \begin{cases} (q_b(u,v) + \gamma)\Delta_b & \text{if } q_b(u,v) > 0, \\ (q_b(u,v) - \gamma)\Delta_b & \text{if } q_b(u,v) < 0, \\ 0 & \text{otherwise,} \end{cases} \tag{21}$$

where $0 \leqslant \gamma < 1$ is a reconstruction parameter arbitrarily chosen by the decoder. A value of $\gamma = 0.50$ results in midpoint reconstruction as in the existing JPEG standard. A value of $\gamma < 0.50$ creates a reconstruction bias towards zero, which can result in improved reconstruction PSNR when the probability distribution of the wavelet coefficients falls off rapidly away from zero (e.g., a Laplacian distribution). A popular choice for biased reconstruction is $\gamma = 0.375$. If all of the M_b bits for a quantizer index are fully decoded, the step-size is equal to Δ_b. However, when only N_b bits are decoded, the step-size in Eq. (21) is equivalent to $\Delta_b 2^{M_b - N_b}$. The reversible (5, 3) filter-bank is treated the same way (with $\Delta_b = 1$), except when the index is fully decoded to achieve lossless reconstruction, in which case $Rq_b(u, v) = q_b(u, v)$.

2.4. Entropy coding

The quantizer indices corresponding to the quantized wavelet coefficients in each subband are entropy encoded to create the compressed bit-stream. The choice of the entropy coder in JPEG 2000 is motivated by several factors. One is the requirement to create an embedded bit-stream, which is made possible by bitplane encoding of the quantizer indices. Bitplane encoding of wavelet coefficients has been used by several well-known embedded wavelet coders such as EZW [38] and SPIHT [36]. However, these coders use coding models that exploit the correlation between subbands to improve coding efficiency.

Unfortunately, this adversely impacts error resilience and severely limits the flexibility of a coder to arrange the bit-stream in an arbitrary progression order. In JPEG 2000, each subband is encoded independently of the other subbands. In addition, JPEG 2000 uses a block coding paradigm in the wavelet domain as in the embedded block coding with optimized truncation (EBCOT) algorithm [44], where each subband is partitioned into small rectangular blocks, referred to as *codeblocks*, and each codeblock is independently encoded. The nominal dimensions of a codeblock are free parameters specified by the encoder but are subject to the following constraints: they must be an integer power of two; the total number of coefficients in a codeblock can not exceed 4096; and the height of the codeblock cannot be less than four.

The independent encoding of the codeblocks has many advantages including localized random access into the image, parallelization, improved cropping and rotation functionality, improved error resilience, efficient rate control, and maximum flexibility in arranging progression orders (see Section 3). It may seem that failing to exploit inter-subband redundancies would have a sizable adverse effect on coding efficiency. However, this is more than compensated by the finer scalability that results from multiple-pass encoding of the codeblock bitplanes. By using an efficient rate

control strategy that independently optimizes the contribution of each codeblock to the final bitstream (see Section 4.2), the JPEG 2000 Part 1 encoder achieves a compression efficiency that is superior to other existing approaches [46].

Fig. 11 shows a schematic of the multiple bitplanes that are associated with the quantized wavelet coefficients. The symbols that represent the quantized coefficients are encoded one bit at a time starting with the MSB and proceeding to the LSB. During this progressive bitplane encoding, a quantized wavelet coefficient is called *insignificant* if the quantizer index is still zero (e.g., the example coefficient in Fig. 11 is still insignificant after encoding its first two MSBs). Once the first non-zero bit is encoded, the coefficient becomes significant, and its sign is encoded. Once a coefficient becomes significant, all subsequent bits are referred to as *refinement* bits. Since the DWT packs most of the energy in the low-frequency subbands, the majority of the wavelet coefficients will have low amplitudes. Consequently, many quantized indices will be insignificant in the earlier bitplanes, leading to a very low information content for those bitplanes. JPEG 2000 uses an efficient coding method for exploiting the redundancy of the bitplanes known as context-based adaptive binary arithmetic coding.

2.4.1. Arithmetic coding and the MQ-coder

Arithmetic coding uses a fundamentally different approach from Huffman coding in that the entire sequence of source symbols is mapped into a single codeword (albeit a very long codeword). This codeword is developed by recursive interval partitioning using the symbol probabilities, and the final codeword represents a binary fraction that points to the subinterval determined by the sequence.

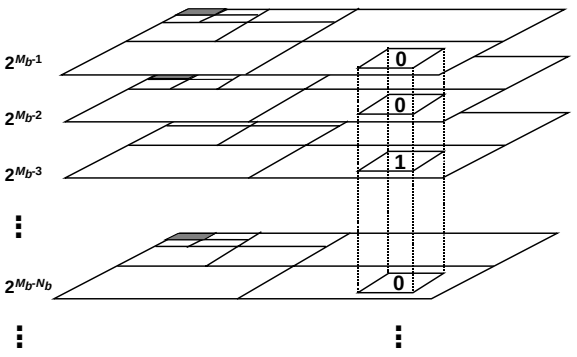


Fig. 11. Bitplane coding of quantized wavelet coefficients.

An adaptive binary arithmetic coder can be viewed as an encoding device that accepts the binary symbols in a source sequence, along with their corresponding probability estimates, and produces a codestream with a length at most two bits greater than the combined ideal codelengths of the input symbols [33]. Adaptivity is provided by updating the probability estimate of a symbol based upon its present value and history. In essence, arithmetic coding provides the compression efficiency that comes with Huffman coding of large blocks, but only a single symbol is encoded at a time. This single-symbol encoding structure greatly simplifies probability estimation, since only individual symbol probabilities are needed at each sub-interval iteration (not the joint probability estimates that are necessary in block coding). Furthermore, unlike Huffman coding, arithmetic coding does not require the development of new codewords each time the symbol probabilities change. This makes it easy to adapt to the changing symbol probabilities within a codeblock of quantized wavelet coefficient bitplanes.

Practical implementations of arithmetic coding are always less efficient than an ideal one. Finite-length registers limit the smallest probability that can be maintained, and computational speed requires approximations, such as replacing multiplies with adds and shifts. Moreover, symbol probabilities are typically chosen from a finite set of allowed values, so the true symbol probabilities must often be approximated. Overall, these restrictions result in a coding inefficiency of approximately 6% compared to the ideal codelength of the symbols encoded [32]. It should be noted that even the most computationally efficient implementations of arithmetic coding are significantly more complex than Huffman coding in both software and hardware.

One of the early practical implementations of adaptive binary arithmetic coding was the Q-coder developed by IBM [33]. Later, a modified version of the Q-coder, known as the QM-coder, was chosen as the entropy coder for the JBIG standard and the extended JPEG mode [32]. However, IPR issues have hindered the use of the QM-coder in the JPEG standard. Instead, the JPEG 2000 committee adopted another modification of the Q-coder, named the MQ-coder. The MQ-coder was also adopted for use in the JBIG2 standard [67]. The companies that own IPR on the MQ-coder have made it available on a license-free and royalty-free basis for use in the JPEG 2000 standard. Differences between the MQ and the QM coders include "bit stuffing" versus "byte stuffing", decoder versus encoder carry resolution, hardware versus software coding convention, and the number of probability states. The specific details of these coders are beyond the scope of this paper, and the reader is referred to [39] and the MQ-coder flowcharts in the standard document [60]. We mention in passing that the specific realization of the "bit stuffing" procedure in the MQ-coder (which costs about 0.5% in coding efficiency), creates a redundancy such that any two consecutive bytes of coded data are always forced to lie in the range of hexadecimal "0000" through "FF8F" [45]. This leaves the range of "FF90" through "FFFF" unattainable by coded data, and the JPEG 2000 syntax uses this range to represent unique marker codes that facilitate the organization and parsing of the bit-stream as well as improve error resilience.

In general, the probability distribution of each binary symbol in a quantized wavelet coefficient is influenced by all the previously coded bits corresponding to that coefficient as well as the value of its immediate neighbors. In JPEG 2000, the probability of a binary symbol is estimated from a *context* formed from its current significance state as well as the significance states of its immediate eight neighbors as determined from the previous bitplane and the current bitplane, based on coded information up to that point. In context-based arithmetic coding, separate probability estimates are maintained for each context, which is updated according to a finite-state machine every time a symbol is encoded in that context.[5] For each context, the MQ-coder can choose from a total of 46 probability states (estimates), where states 0 through 13 correspond to start-up states (also referred to as *fast-attack*) and are used for rapid convergence to a stable probability estimate. States 14 through 45 correspond to steady-state probability estimates and once entered from a start-up state, can never be left by the finite-state machine. There is also an additional non-adaptive state (state 46), which is used to encode symbols with equal probability distribution, and can neither be entered nor exited from any other probability state.

2.4.2. Bitplane coding passes

The quantized coefficients in a codeblock are bitplane encoded independently from all other codeblocks when creating an embedded bit-stream. Instead of encoding the entire bitplane in one coding pass, each bitplane is encoded in three sub-bitplane passes with the provision of truncating the bit-stream at the end of each coding pass. A main advantage of this approach is near-optimal embedding, where the information that results in the largest reduction in distortion for the smallest increase in file size is encoded first. Moreover, a large number of potential truncation points facilitates an optimal rate control strategy where a target bit-rate is achieved by including those coding passes that minimize the total distortion.

Referring to Fig. 12, consider the encoding of a single bitplane from a codeblock in three coding passes (labeled A, B and C), where a fraction of the bits are encoded at each pass. Let the distortion and bit-rate associated with the reconstructed image prior and subsequent to the encoding of the entire bitplane be given by (D_1, R_1) and (D_2, R_2), respectively. The two coding paths ABC and CBA correspond to coding the same data in a different order, and they both start and end at the same rate-distortion points. However, their embedded performances are

[5] In the MQ-coder implementation, a symbol's probability estimate is actually updated only when at least one bit of coded output is generated.

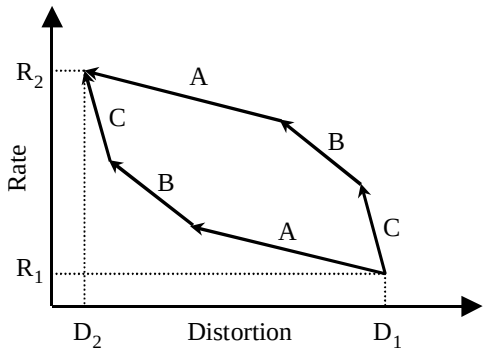


Fig. 12. R–D path for optimal embedding.

significantly different. In particular, if the coded bit-stream is truncated at any intermediate point during the encoding of the bitplane, the path ABC would have less distortion for the same rate, and hence would possess a superior embedding property. In creating optimal embedding, the data with the highest distortion reduction per average bit of compressed representation should be coded first [23].

For a coefficient that is still insignificant, it can be shown that given reasonable assumptions about its probability distribution, the distortion reduction per average bit of compressed representation increases with increasing probability of becoming significant, p_s [23,30]. For a coefficient that is being refined, the distortion reduction per average bit is smaller than an insignificant coefficient, unless p_s for that coefficient is less than 1%. As a result, optimal embedding can theoretically be achieved by first encoding the insignificant coefficients starting with the highest p_s until that probability reaches about 1%. At that point, all the refinement bits should be encoded, followed by all the remaining coefficients in the order of their decreasing p_s. However, the calculation of the p_s values for each coefficient is a tedious and approximate task, so the JPEG 2000 coder instead divides the bitplane data into three groups and encodes each group during a fractional bitplane pass. Each coefficient in a block is assigned a binary state variable called its *significance state* that is initialized to zero (insignificant) at the start of the encoding. The significance state changes from zero to one (significant) when the first non-zero magnitude bit is found. The context vector for a given coefficient is the binary vector consisting of the significance states of its eight immediate neighbor coefficients as shown in Fig. 13. During the first pass, referred to as the *significance propagation* pass, the insignificant coefficients that have the highest probability of becoming significant, as determined by their immediate eight neighbors, are encoded. In the second pass, known as the *refinement* pass, the significant coefficients are refined by their bit representation in the current bitplane. Finally, during the *cleanup* pass, all the remaining coefficients in the bitplane are encoded as they have the lowest probability of becoming significant. The order in which the data in each pass are visited is data dependent and follows a deterministic stripe-scan order with a height of four pixels as shown in Fig. 14. This stripe-based scan has been shown to facilitate software and hardware implementations [26]. The bit-stream can be truncated at the end of each

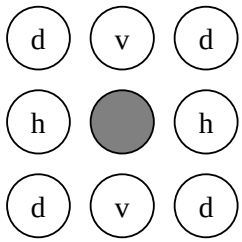


Fig. 13. Neighboring pixels used in context selection.

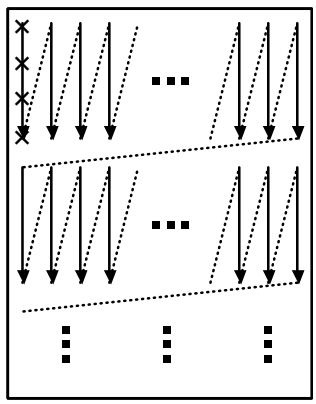


Fig. 14. Scan order within a codeblock.

coding pass. In the following, each coding pass is described in more detail.

2.4.2.1. Significance propagation pass. During this pass, the insignificant coefficients that have the highest probability of becoming significant in the current bitplane are encoded. The data is scanned in the stripe order shown in Fig. 14, and every sample that has at least one significant immediate neighbor, based on coded information up to that point, is encoded. As soon as a coefficient is coded, its significance state is updated so that it can effect the inclusion of subsequent coefficients in that coding pass. The significance state of the coefficient is arithmetic coded using contexts that are based on the significance states of its immediate neighbors. In general, the significance states of the eight neighbors can create 256 different contexts,[6] however, many of these contexts have similar probability estimates and can be merged together. A context reduction mapping reduces the total number of contexts to only nine to improve the efficiency of the MQ-coder probability estimation for each context. Since the codeblocks are encoded independently, if a sample is located at the codeblock boundary, only its immediate neighbors that belong to the current codeblock are considered and the significance state of the missing neighbors are assumed to be zero. Finally, if a coefficient is found to be significant, its sign needs to be encoded. The sign value is also arithmetic encoded using five contexts that are determined from the significance and the sign of the coefficient's four horizontal and vertical neighbors.

2.4.2.2. Refinement pass. During this pass, the magnitude bit of a coefficient that has already become significant in a previous bitplane is arithmetic encoded using three contexts. In general, the refinement bits have an even distribution unless the coefficient has just become significant in the previous bitplane (i.e., the magnitude bit to be encoded is the first refinement bit). This condition is first tested and if it is satisfied, the magnitude bit is encoded using two coding contexts based on the significance of the eight immediate neighbors. Otherwise, it is coded with a single context regardless of the neighboring values.

2.4.2.3. Cleanup pass. All the remaining coefficients in the codeblock are encoded during the cleanup pass. Generally, the coefficients coded in this pass have a very small p_s value and are expected to remain insignificant. As a result, a special mode, referred to as the run mode, is used to aggregate the coefficients that have the highest probability of remaining insignificant. More specifically, a run mode is entered if all the four samples in a vertical column of the stripe have insignificant neighbors. In the run mode, a binary symbol is arithmetic encoded in a single context to specify whether all the four samples in the vertical column remain insignificant. An encoded value of zero implies insignificance for all four samples, while an encoded value of one implies that at least one of the four samples becomes significant in the current bitplane. An encoded value of one is followed by two additional arithmetic encoded bits that specify the location of the first nonzero coefficient in the vertical column. Since the probability of these additional two bits is nearly evenly distributed, they are encoded with a *uniform* context, which uses state 46 of the MQ-coder as its probability estimate. It should be noted that the run mode has a negligible impact on the coding efficiency, and it is primarily used to improve the throughput of the arithmetic coder through symbol aggregation.

After the position of the first nonzero coefficient in the run is specified, the remaining samples in the vertical column are encoded in the same manner as in the significance propagation pass and use the same nine coding contexts. Similarly, if at least one of the four coefficients in the vertical column has a significant neighbor, the run mode is disabled and all the coefficients in that column are coded according to the procedure employed for the significance propagation pass.

For each codeblock, the number of MSB planes that are entirely zero is signaled in the bit-stream. Since the significance state of all the coefficients in

[6] Technically, the combination where all the neighbors are insignificant cannot happen in this pass. However, this combination is given its own context (labeled zero) and is used during the cleanup pass.

the first non-zero MSB is zero, only the cleanup pass is applied to the first non-zero bitplane.

2.4.3. Entropy coding options

The coding models used by the JPEG 2000 entropy coder employ 18 coding contexts in addition to a uniform context according to the following assignment. Contexts 0–8 are used for significance coding during the significance propagation and cleanup passes, context 9 is used for run coding the cleanup pass, contexts 10–14 are used for sign coding, contexts 15–17 are used during the refinement pass. Each codeblock employs its own MQ-coder to generate a single arithmetic codeword for the entire codeblock. In the default mode, the coding contexts for each codeblock are initialized at the start of the coding process and are not reset at any time during the encoding process. Furthermore, the resulting codeword can only be truncated at the coding pass boundaries to include a different number of coding passes from each codeblock in the final codestream. All contexts are initialized to uniform probabilities except for the zero context (all insignificant neighbors) and the run context, where the initial less probable symbol (LPS) probabilities are set to 0.0283 and 0.0593, respectively.

In order to facilitate the parallel encoding or decoding of the sub-bitplane passes of a single codeblock, it is necessary to decouple the arithmetic encoding of the sub-bitplane passes from one another. Hence, JPEG 2000 allows for the termination of the arithmetic coded bit-stream as well as the re-initialization of the context probabilities at each coding pass boundary. If any of these two options is flagged in the codestream, it must be executed at every coding pass boundary. The JPEG 2000 also provides for another coding option known as *vertically stripe-causal* contexts. This option is aimed at enabling the parallel decoding of the coding passes as well as reducing the external memory utilization. In this mode, during the encoding of a certain stripe of a codeblock, the significances of the samples in future stripes within that codeblock are ignored. Since the height of the vertical columns is four pixels, this mode only affects the pixels in the last row of each stripe. The combination of these three options, namely arithmetic coder termination, re-initialization at each coding pass boundary, and the vertically stripe-causal context, is often referred to as the *parallel* mode.

Another entropy coding option, aimed at reducing computational complexity, is the *lazy* coding mode, where the arithmetic coder is entirely bypassed in certain coding passes. More specifically, after the encoding of the fourth most significant bitplane of a codeblock, the arithmetic coder is bypassed during the encoding of the first and second sub-bitplane coding passes of subsequent bitplanes. Instead, their content is included in the codestream as raw data. In order to implement this mode, it is necessary to terminate the arithmetic coder at the end of the cleanup pass preceding each raw coding pass and to pad the raw coding pass data to align it with the byte boundary. However, it is not necessary to re-initialize the MQ-coder context models. The lazy mode can also be combined with the parallel mode. The impact of the lazy and parallel modes on the coding efficiency is studied in Section 5.1.5.

2.4.4. Tier-1 and tier-2 coding

The arithmetic coding of the bitplane data is referred to as *tier-1* (T1) coding. Fig. 15 illustrates a simple example of the compressed data generated at the end of tier-1 encoding. The example image (shown at the top right of Fig. 15) is of size 256×256 with two levels of decomposition, and the codeblock size is 64×64. Each square box in the figure represents the compressed data associated with a single coding pass of a 22 single codeblock. Since the codeblocks are independently encoded, the compressed data corresponding to the various coding passes can be arranged in different configurations to create a rich set of progression orders to serve different applications. The only restriction is that the sub-bitplane coding passes for a given codeblock must appear in a causal order starting from the most significant bitplane. The compressed sub-bitplane coding passes can be aggregated into larger units named *packets*. This process of packetization along with its supporting syntax, as will be explained in Section 3, is often referred to as *tier-2* (T2) coding.

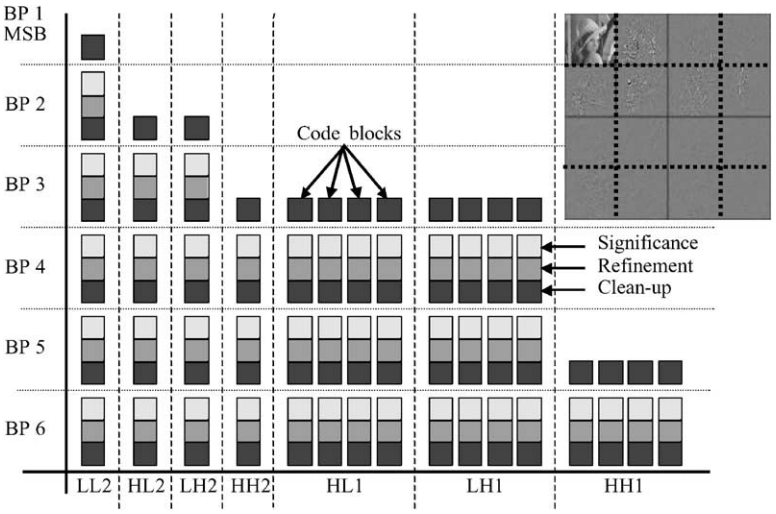


Fig. 15. Example of compressed data associated with various sub-bitplane coding passes.

3. JPEG 2000 bit-stream organization

JPEG 2000 offers significant flexibility in the organization of the compressed bit-stream to enable such features as random access, region of interest coding, and scalability. This flexibility is achieved partly through the various structures of components, tiles, subbands, resolution levels, and codeblocks that are discussed in Section 2. These structures partition the image data into: (1) color channels (through components); (2) spatial regions (through tiles); (3) frequency regions (through subbands and resolution levels), and (4) space–frequency regions (through codeblocks). Tiling provides access to the image data over large spatial regions, while the independent coding of the codeblocks provides access to smaller units. Codeblocks can be viewed as a tiling of the coefficients in the wavelet domain. JPEG 2000 also provides an intermediate space-frequency structure known as a *precinct*. A precinct is a collection of spatially contiguous codeblocks from all subbands at a particular resolution level.

In addition to these structures, JPEG 2000 organizes the compressed data from the codeblocks into units known as *packets* and *layers* during the tier-2 coding step. For each precinct, the compressed data for the codeblocks is first organized into one or more packets. A packet is simply a continuous segment in the compressed codestream that consists of a number of bitplane coding passes for each codeblock in the precinct. The number of coding passes can vary from codeblock to codeblock (including zero coding passes). Packets from each precinct at all resolution levels in a tile are then combined to form layers. In order to discuss packetization of the compressed data, it is first necessary to introduce the concepts of *resolution grids* and *precinct partitions*. Throughout the following discussion, it will be assumed that the image has a single tile and a single component. The extension to multiple tiles and components (which are possibly subsampled) is straightforward, but tedious, and it is not necessary for understanding the basic concepts. Section B.4 of the JPEG 2000 standard [60] provides a detailed description and examples for the more general case.

3.1. Canvas coordinate system

During the application of the DWT to the input image, successively lower resolution versions of the input image are created. The input image can be thought of as the highest resolution version. The pixels of the input image are referenced with respect to a high-resolution grid, known as the *reference grid*. The reference grid is a rectangular

grid of points with indices from $(0,0)$ to $(Xsiz\text{-}1, Ysiz\text{-}1)$.[7] If the image has only one component, each image pixel corresponds to a high-resolution grid. In case of multiple components with differing sampling rates, the samples of each component are at integer multiples of the sampling factor on the high-resolution grid. An *image area* is defined by the parameters $(XOsiz, YOsiz)$ that specify the upper left corner of the image, and extends to $(Xsiz\text{-}1, Ysiz\text{-}1)$ as shown in Fig. 16.

The spatial positioning of each resolution level, as well as each subband, is specified with respect to its own coordinate system. We will refer to each coordinate system as a *resolution grid*. The collection of these coordinate systems is known as the *canvas coordinate system*. The relative positioning of the different coordinate systems corresponding to the resolution levels and subbands is defined in Section B.5 of the JPEG 2000 standard [60], and is also specified later in this section. The advantage of the canvas coordinate system is that it facilitates the compressed domain implementation of certain spatial operations, such as cropping and rotation by multiples of 90°. As will be described in Section 5.1.6, proper use of the canvas coordinate system improves the performance of the JPEG 2000 encoder in case of multiple compression cycles when the image is being cropped between compression cycles.

3.2. Resolution grids

Consider a single component image that is wavelet transformed with N_L decomposition levels, creating $N_L + 1$ distinct resolution levels. An image at resolution level r $(0 \leqslant r \leqslant N_L)$ is represented by the subband $(N_L - r)$LL. Recall from Section 2.2.2 that the image at resolution r $(r > 0)$ is formed by combining the image at resolution $(r-1)$ with the subbands at resolution r, i.e. subbands $(N_L - r + 1)$HL, $(N_L - r + 1)$LH and $(N_L - r + 1)$HH. The image area on the high-resolution reference grid as specified by $(Xsiz, Ysiz)$ and $(XOsiz, YOsiz)$ is propagated to lower resolution levels as follows. For the image area at resolution level r $(0 \leqslant r \leqslant N_L)$ the upper left-hand corner is (xr_0, yr_0) and the lower right-hand corner is $(xr_1 - 1, yr_1 - 1)$, where

$$xr_0 = \left\lceil \frac{XOsiz}{2^{N_L - r}} \right\rceil, \quad yr_0 = \left\lceil \frac{YOsiz}{2^{N_L - r}} \right\rceil, \quad xr_1 = \left\lceil \frac{Xsiz}{2^{N_L - r}} \right\rceil \quad \text{and} \quad yr_1 = \left\lceil \frac{Ysiz}{2^{N_L - r}} \right\rceil, \tag{22}$$

and $\lceil w \rceil$ denotes the smallest integer that is greater than or equal to w.

The high-resolution reference grid is also propagated to each subband as follows. The positioning of the subband n_bLL is the same as that of the image at a resolution of $(N_L - n_b)$. The positioning of subbands n_bHL, n_bLH, and n_bHH is specified as

$$(xb_0, yb_0) = \begin{cases} \left(\left\lceil \dfrac{XOsiz - 2^{n_b - 1}}{2^{n_b}} \right\rceil, \left\lceil \dfrac{YOsiz}{2^{n_b}} \right\rceil \right) & \text{for } n_b\text{HL band,} \\ \left(\left\lceil \dfrac{XOsiz}{2^{n_b}} \right\rceil, \left\lceil \dfrac{YOsiz - 2^{n_b - 1}}{2^{n_b}} \right\rceil \right) & \text{for } n_b\text{LH band,} \\ \left(\left\lceil \dfrac{XOsiz - 2^{n_b - 1}}{2^{n_b}} \right\rceil, \left\lceil \dfrac{YOsiz - 2^{n_b - 1}}{2^{n_b}} \right\rceil \right) & \text{for } n_b\text{HH band.} \end{cases} \tag{23}$$

The coordinates (xb_1, yb_1) can be obtained from Eq. (23) by substituting $XOsiz$ with $Xsiz$ and $YOsiz$ with $Ysiz$. The extent of subband b is from $(xb0, yb0)$ to $(xb_1 - 1, yb_1 - 1)$. These concepts are best illustrated by a simple example. Consider a 3-level wavelet decomposition of an original image of size 768 (columns) × 512 (rows). Let the upper left reference grid point $(XOsiz, YOsiz)$ be $(7, 9)$ for the image area. Then, $(Xsiz, Ysiz)$ is $(775, 521)$. Resolution one extends from $(2, 3)$ to $(193, 130)$ while subband 3HL,

[7] The coordinates are specified as (x, y), where x refers to the column index and y refers to the row index.

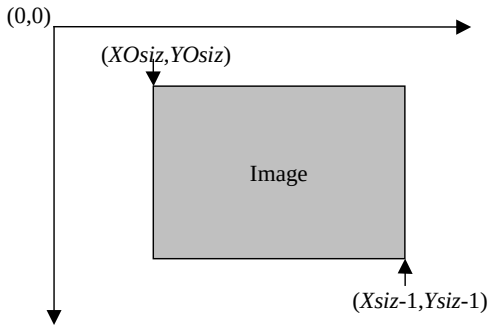


Fig. 16. The canvas coordinate system.

which belongs to resolution one, extends from $(1, 2)$ to $(96, 65)$.

3.3. Precinct and codeblock partitioning

Each resolution level of a tile is further partitioned into rectangular regions known as *precincts*. Precinct partitioning makes it easier to access the wavelet coefficients corresponding to a particular spatial region of the image. The precinct partition at resolution r induces a precinct partitioning of the subbands at *the same resolution level*, i.e. subbands (N_L-r+1)HL, (N_L-r+1)LH and (N_L-r+1)HH. The precinct size can vary from resolution to resolution, but is restricted to be a power of two. Each subband is also divided into rectangular codeblocks with dimensions that are a power of two. The precinct and codeblock partitions are both anchored at $(0, 0)$. Each precinct boundary coincides with a codeblock boundary, but the reverse is not true, because a precinct may consist of multiple codeblocks.

Codeblocks from all resolution levels are constrained to have the same size, except due to the constraints imposed by the precinct size. For codeblocks having the same size, those from lower resolutions correspond to progressively larger regions of the original image. For example, for a three-level decomposition, a 64×64 codeblock in subbands 1LL, 2LL and 3LL corresponds to original image regions of size 128×128, 256×256 and 512×512, respectively. This diminishes the ability of the codeblocks to provide spatial localization. To alleviate this problem, the codeblock size at a given resolution is bounded by the precinct size at that resolution. For example, consider a 768×512 image that we wish to partition into six 256×256 regions for efficient spatial access. For a codeblock size of 64×64, the precinct sizes for resolutions 0–3 can be chosen to be 32×32, 32×32, 64×64 and 128×128, respectively. In this case, the actual codeblock size for the 3LL, 3LH, 3HL and 3HH subbands would be 32×32. Fig. 17 shows the precinct partitions for a three-level decomposition of a 768×512 image. The highlighted precincts in resolutions 0–3 correspond roughly to the same 256×256 region in the original image.

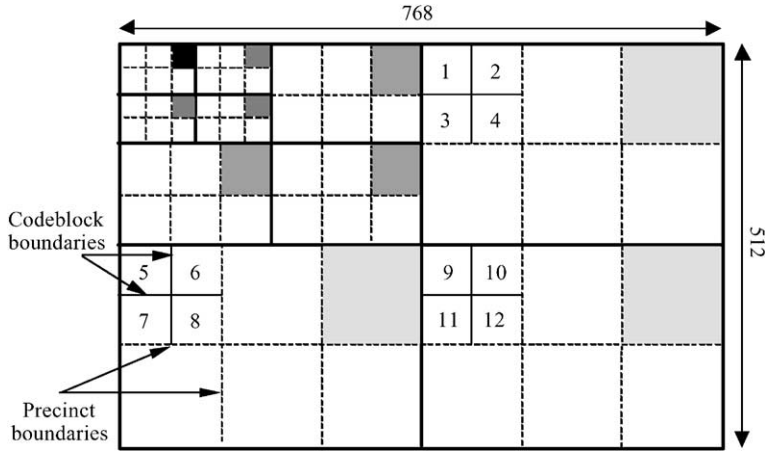


Fig. 17. Examples of precincts and codeblocks.

3.4. Layers and packets

The compressed bit-stream for each codeblock is distributed across one or more layers in the codestream. All of the codeblocks from all subbands and components of a tile contribute compressed data to each layer. For each codeblock, a number of consecutive coding passes (including zero) is included in a layer. Each layer represents a quality increment. The number of coding passes included in a specific layer can vary from one codeblock to another and is typically determined by the encoder as a result of post-compression rate-distortion optimization as will be explained in Section 4.2. This feature offers great flexibility in ordering the codestream. It also enables spatially adaptive quantization. Recall that all the codeblocks in a subband must use the same quantizer step-size. However, the layers can be formed in such a manner that certain codeblocks, which are deemed perceptually more significant, contribute a greater number of coding passes to a given layer. As discussed in Section 2.3.1, this reduces the effective quantizer step-size for those codeblocks by a power of two compared to other codeblocks with less coding passes in that layer.

The compressed data belonging to a specific tile, component, resolution, layer and precinct is aggregated into a packet. The compressed data in a packet needs to be contiguous in the codestream. If a precinct contains data from more than one subband, it appears in the order HL, LH and HH. Within each subband, the contributions from codeblocks appear in the raster order. Fig. 17 shows an example of codeblocks belonging to a precinct. The numbering of the codeblocks represents the order in which the coded data from the codeblocks will appear in a packet.

3.5. Packet header

A packet is the fundamental building block in a JPEG 2000 codestream. Each packet starts with a *packet header*. The packet header contains information about the number of coding passes for each codeblock in the packet. It also contains the length of the compressed data for each codeblock. The first bit of a packet header indicates whether the packet contains data or is empty. If the packet is non-empty, codeblock inclusion information is signaled for each codeblock in the packet. This information indicates whether any compressed data from a codeblock is included in the packet. If compressed codeblock data has already been included in a previous packet, this information is signaled using a single bit. Otherwise, it is signaled with a separate *tag-tree* for the corresponding precinct. The tag-tree is a hierarchical data structure that is capable of exploiting spatial redundancy. If codeblock data is being included for the first time, the number of most significant bitplanes that are entirely zero is also signaled with another set of tag-trees for the precinct. After this, the number of coding passes for the codeblock and the length of the corresponding compressed data are signaled.

The arithmetic encoding of the bitplanes is referred to as tier-1 coding, whereas the packetization of the compressed data and encoding of the packet header information is known as tier-*2* coding. In order to change the sequence in which the packets appear in the codestream, it is necessary to decode the packet header information, but it is not necessary to perform arithmetic decoding. This allows the codestream to be reorganized with minimal computational complexity.

3.6. Progression order

The order in which packets appear in the codestream is called the progression order and is controlled by specific markers. Regardless of the ordering, it is necessary that coding passes for each codeblock appear in the codestream in causal order from the most significant bit to the least significant bit. For a given tile, four parameters are needed to uniquely identify a packet. These are component, resolution, layer and position (precinct). The packets for a particular component, resolution and layer are generated by scanning the precincts in a raster order. All the packets for a tile can be ordered by using nested "*for loops*" where each "*for loop*" varies one parameter from the above list. By changing the nesting order of the "*for loops*", a number of different progression

orders can be generated. JPEG 2000 Part 1 allows only five progression orders, which have been chosen to address specific applications. They are (i) layer–resolution–component–position progression; (ii) resolution–layer–component–position progression; (iii) resolution–position–component–layer progression; (iv) position–component–resolution–layer progression; and (v) component–position–resolution–layer progression. These progression orders share some similarities with the different modes of the extended DCT-based JPEG standard as will be pointed out in the subsequent subsections.

To illustrate these different orderings, consider a three-component color image of size 768×512 with two layers and three decomposition levels (corresponding to four resolution levels). The precinct partition is as shown in Fig. 17. The component, resolution, layer and position are indexed by c, r, l and k, respectively. It is possible that the components of an image have different number of resolution levels. In that case, the LL subbands of different components are aligned.

3.6.1. Layer–resolution–component–position progression (LRCP)

This type of progression is obtained by arranging the packets in the following order:

```
for each l = 0, 1
  for each r = 0, 1, 2, 3
    for each c = 0, 1, 2
      for each k = 0, 1, 2, 3, 4, 5
        packet for component c, resolution r,
        layer l, and position k.
```

This type of progression order is useful in an image database browsing application, where progressively refining the quality of an image may be desirable. This mode has no exact counterpart in the existing JPEG. However, the "sequential progressive" mode of extended JPEG (component non-interleaved format) provides similar functionality for a single resolution image.

3.6.2. Resolution–layer–component–position progression (RLCP)

This type of progression order is obtained by interleaving the "*for loops*" in the order r, l, c and k, starting with the outermost "*for loop*". It is useful in a client–server application, where different clients might demand images at different resolutions. This progression order is similar to "hierarchical progressive" mode of extended JPEG where each resolution is further encoded with the "sequential progressive" mode (component non-interleaved format).

3.6.3. Resolution–position–component–layer progression (RPCL)

This type of progression order is obtained by interleaving the "*for loops*" in the order r, k, c and l, starting with the outermost "*for loop*". It can be used when resolution scalability is needed, but within each resolution, it is desirable that all packets corresponding to a precinct appear contiguously in the compressed bit-stream. The "resolution–position–component" order for a single layer can be obtained using the 27 "hierarchical progressive" mode of extended JPEG with each resolution encoded with baseline JPEG (component interleaved format).

3.6.4. Position–component–resolution–layer progression (PCRL)

This type of progression order is obtained by arranging the "*for loops*" in the order k, c, r and l, starting with the outermost "*for loop*". It should be used if it is desirable to refine the image quality at a particular spatial location. The "position–component" order is similar to JPEG baseline where the image is sequentially compressed by compressing component interleaved 8×8 blocks in a raster order fashion.

3.6.5. Component–position–resolution–layer progression (CPRL)

This type of progression order is obtained by arranging the "*for loops*" in the order c, k, r and l, starting with the outermost "*for loop*". It should be used if it is desirable to obtain highest quality image for a particular spatial location only for a specific image component. The "component–position" order is similar to the JPEG baseline where the image is sequentially compressed by

compressing each color component separately in a raster order fashion.

In the last three progression orders, the "*for loop*" corresponding to the variable k, which determines the order in which the precincts appear in the codestream, can become complicated if different components have different precinct sizes as is explained in the standard document [60]. The JPEG 2000 syntax offers the flexibility of changing from one progression order to another in the middle of the codestream. For example, a digital camera image might start out in the RLCP order to provide a thumbnail. The order then may be switched to LRCP to facilitate rate control and truncation after the image has been captured.

Figs. 18–20 illustrate some of these progression orders for the "boy" image (768×512, monochrome). In these examples, the DWT has three decomposition levels, the $(9, 7)$ filter-bank is used, and the precinct sizes for the subbands at resolutions 0, 1, 2 and 3 are 32×32, 32×32, 64×64 and 128×128, respectively. The codeblock size is 64×64, except for resolutions 0 and 1 where the codeblock size is constrained to the precinct size of 32×32. Thus, there are four resolutions, six precincts per resolution, and two layers, resulting in 48 packets. Fig. 18 shows the LRCP progression order (Section 3.6.1). The image has been reconstructed at the two quality levels of 0.125 bits/pixel and 0.5 bits/pixel by decoding 24 and 48 packets, respectively. Fig. 19 illustrates the RLCP ordering (Section 3.6.2). The figure shows images reconstructed after decoding resolutions $0, 1, 2$ and 3 ($12, 24, 36$ and 48 packets), respectively. Fig. 20 illustrates the PCRL ordering (Section 3.6.4). The image has been reconstructed after decoding 32 packets corresponding to the first four precincts.

It should be noted that due to the prediction step in the "hierarchical progressive" mode of extended JPEG, before decoding any data at a given resolution, it is necessary to fully decode all the data corresponding to the lower resolution versions of the image. This inter-resolution dependency makes it impossible to achieve 28 certain progression orders, e.g. LRCP. Also, rearranging the JPEG compressed data from one progression

Fig. 18. Example of layer progressive bit-stream ordering, (left) 0.125 bpp; (right) 0.50 bpp.

Fig. 19. Example of resolution progressive bit-stream ordering.

Fig. 20. Example of spatially progressive bit-stream ordering (four precincts decoded).

mode to another generally requires an inverse DCT, for example, when converting from the "hierarchical progressive" to the "sequential progressive" mode. With JPEG 2000, a given progression order can be converted into another without the need for arithmetic decoding or inverse wavelet transform by simply rearranging the packets. This only requires decoding of the packet headers to determine the length of each packet.

4. Rate control

Rate control refers to the process of generating an optimal image for a target file size (bit-rate) and is strictly an encoder issue. The criterion for optimality can be based on mean squared error (MSE) between the original and reconstructed image, visual distortion, or any other metric. In the existing JPEG standard, the user only has control over the selection of the quantization and Huffman tables, which does not provide an easy mechanism for compressing an image to a desired bit-rate. A typical JPEG rate control algorithm starts with a basic q-table and iteratively modifies the q-table elements (e.g., by a scale factor) until the desired bit-rate is achieved. In contrast, the embedded block coding scheme of the JPEG 2000 and its flexible codestream syntax allow for the generation of a rate-distortion (R–D) optimized codestream for a given file size. Each JPEG 2000 encoder can perform its own optimization (based on the distortion metric used) to generate a codestream that conforms to the standardized syntax. In the following, a brief discussion of several possible approaches to JPEG 2000 rate control is provided.

4.1. Rate control using explicit q-table

One approach is to use an explicit q-table similar to JPEG, where a quantizer step-size is specified for each subband and signaled explicitly as header information. As mentioned before, this approach suffers from the drawback that the q-table needs to be modified (e.g., scaled) iteratively to achieve the desired bit-rate. Although there is no need to perform the wavelet transform at each iteration, the quantization and coding processes still need to be performed.

In most applications, humans are the ultimate judges of perceived image quality, and it is important to consider the properties of the HVS when designing a q-table [4,19,31,49]. The general approach to q-table design is to take advantage of the sensitivity variations of the HVS to different spatial frequencies. Although q-tables can be designed through actual observer experiments, as was done in developing the example q-tables specified in the existing JPEG standard, such experiments are laborious and must be repeated each time that the viewing conditions are changed. A more efficient approach is to use a contrast sensitivity function (CSF) model as described in [19]. The application of a CSF model requires knowledge of the intended viewing conditions, such as viewing distance, displayed pixel size, display noise, and light adaptation level. In general, the use of a CSF model implies that the q-tables are designed for threshold-level compression. That is, the q-tables will produce errors that are barely perceptible in the compressed image under the viewing conditions for which the q-tables were developed.

4.2. Rate control using the EBCOT algorithm

In [44], Taubman proposed an efficient rate control method for the EBCOT compression algorithm that achieves a desired rate in a single iteration with minimum distortion. This method

can also be used by a JPEG 2000 encoder, with several possible variations.

In the basic approach, each subband is first quantized using a very fine step-size, and the bitplanes of the resulting codeblocks are entropy coded. This typically generates more coding passes for each codeblock than will be eventually included in the final codestream. If the quantizer step-size is chosen to be small enough, the R–D performance of the algorithm is independent of the initial choice of the step-size. Next, a Lagrangian R–D optimization is performed to determine the number of coding passes from each codeblock that should be included in the final compressed bit-stream to achieve the desired bit-rate. If more than a single layer is desired, this process can be repeated at the end of each layer to determine the additional number of coding passes from each codeblock that need to be included in the next layer.

The Lagrangian R–D optimization works in the following manner. The compressed bit-stream from each codeblock contains a large number of potential truncation points that can occur at the end of each sub-bitplane pass. The wavelet coefficients $y(u,v)$ contained in a codeblock of subband b are initially quantized with a step-size of Δ_b, resulting in an M_b-bit quantizer index for each coefficient. If the codeblock bit-stream is truncated so that only N_b bits are decoded, the effective quantizer step-size for the coefficients is $\Delta_b 2^{M_b-N_b}$. The inclusion of each additional bit-plane in the compressed bit-stream will decrease the effective quantizer step-size by a factor of two. However, the effective quantizer step-size might not be the same for every coefficient in a given codeblock due to the inclusion of some coefficients in the sub-bitplane at which the truncation occurs. For each sub-bitplane, the increase in bit-rate and the reduction in distortion resulting from the inclusion of that sub-bitplane in the bit-stream are calculated. The distortion measure selected is usually MSE or visually weighted MSE, although any general distortion measure that is additive across codeblocks can be used. Let the total number of codeblocks for the entire image be P, and let the 30 codeblocks in the image be denoted by B_i, $1 \leqslant i \leqslant P$. For a given truncation point t in codeblock B_i, the associated weighted MSE distortion D_i^t is given by

$$D_i^t = \alpha_b^2 \sum_{u,v} w_i(u,v)[y_i(u,v) - y_i^t(u,v)]^2, \tag{24}$$

where u and v represent the coefficient row and column indices within the codeblock B_i; $y_i(u,v)$ is the original coefficient value; $y_i^t(u,v)$ is the quantized coefficient value for truncation point t; $w_i(u,v)$ is a weighting factor for coefficient $y_i(u,v)$; and α_b is the L_2-norm for subband b. Under certain assumptions about the quantization noise, this distortion is additive across codeblocks. At the given truncation point t, the size of the associated compressed bit-stream (i.e., the rate) for the codeblock B_i is determined and denoted by $R_i{}^t$.

Given a total bit budget of R bytes for the compressed bit-stream, the EBCOT rate control algorithm finds the truncation point for each codeblock that minimizes the total distortion D. This is equivalent to finding the optimal bit allocation for all of the codeblocks, R_i^*, $1 \leqslant i \leqslant P$, such that

$$D = \sum_i D_i^* \quad \text{is minimized subject to} \quad \sum_i R_i^* \leqslant R. \tag{25}$$

In the JPEG 2000 literature, this rate control algorithm is also referred to as *post-compression R–D optimization*. If the weighting factor $w_i(u,v)$ is set to unity for all subband coefficients, the distortion metric reduces to the mean-squared error. A visual weighting strategy can also be used in conjunction with the EBCOT rate control algorithm as will be discussed next.

4.2.1. Fixed visual weighting

The CSF model used to design the q-tables for explicit quantization of the wavelet coefficients can also be used to derive the weighting factors w_i (u,v). For example, once the CSF-based quantization step-sizes have been computed for a given viewing condition (Section 4.1), the weighting factor for all the coefficients in a subband can be set equal to the square of the reciprocal of these step-sizes. Table J-24 from Part 1 of the JPEG 2000 standard [60] lists recommended frequency weightings for three different viewing conditions. This approach is known as *fixed visual weighting*.

4.2.2. Bit-stream truncation and layer construction

In the EBCOT rate control algorithm, an image is compressed in such a way that the minimum distortion is achieved at the desired bit-rate. However, it is sometimes desirable to truncate an existing JPEG 2000 codestream to achieve a smaller bit-rate. For example, this scenario could take place in a digital camera where the already captured and compressed images have to be truncated to enable storage of a newly captured image. The question that arises is whether the truncated bit-stream also achieves the minimum distortion for the smaller bit-rate. In other words, we want the visual quality of the image from the truncated codestream to be as close as possible to the visual quality of the image that would be produced by compressing directly to that bit-rate.

This property can be achieved only if the image was initially encoded with a number of layers using the LRCP ordering of the packets as described in Section 3.6.1. The layers can be designed using R–D optimization so that the minimum distortion for the resulting bit-rate is achieved at each layer boundary. However, the quality of the resulting truncated image might not be optimal if the truncation point for the desired bit-rate does not fall on a layer boundary. This is because the non-boundary truncation of a layer in LCRP ordering will result in a number of packets being discarded. If the desired bit-rates or quality levels are known in advance for a given application, it is recommended that the layers be constructed accordingly. If the exact target bit-rates are not known a priori, it is recommended that a large number of layers (e.g., 50) be formed. This provides the ability to approximate a desired bit-rate while still truncating at a layer boundary. As demonstrated in Section 5.1.7, the impact of the resulting overhead on PSNR is quite small.

4.2.3. Progressive visual weighting

In fixed visual weighting, the visual weights are chosen according to a single viewing condition. However, if the bit-stream is truncated, this viewing condition may be inappropriate for the reduced quality image. Consider a case where the compressed bit-stream has a number of layers, each corresponding to a potential truncation point. If the bit-stream is truncated at a layer boundary with a very low bit-rate, the resulting image quality would be poor and the image might be viewed at a larger viewing distance than the one intended for the original compressed bit-stream. As a result, in some applications it might be desirable to have each layer correspond to a different viewing condition. In an embedded coder such as JPEG 2000, it is not possible to change the subband quantization step-sizes for each layer. However, if a nominal viewing condition can be associated with each layer, a corresponding set of visual weighting factors $w_i(u, v)$ can be used during the R–D optimization process for that layer [22]. This is known as *progressive visual weighting*.

5. Performance comparison of JPEG 2000 encoder options

The JPEG 2000 standard offers a number of encoder options that directly affect the coding efficiency, speed and implementation complexity. In this section, we primarily compare the effects of various coding options on the coding efficiency for lossless compression and on the rate-distortion performance for lossy compression. It is more difficult to accurately compare the speed and implementation complexity of different coding options, so we only point out the relative speed/complexity advantages of certain options.

To obtain the reported results, the three test images shown in Fig. 21, "Bike", "Café" and "Woman" of size 2048 (columns) $\times$ 2560 (rows) were chosen from the JPEG 2000 test set. All three images are grayscale and have a bit-depth of 8 bits/sample. For lossy compression, distortion was characterized by the peak signal to noise ratio (PSNR), which for an 8-bit decompressed image is defined as

$$\mathrm{PSNR} = 10 \log_{10}\left(\frac{255^2}{\mathrm{MSE}}\right), \tag{26}$$

where MSE refers to the mean squared error between the original image and the reconstructed image. In most cases, the results are presented as

Fig. 21. Test images (from left to right) Bike, Café, Woman.

the average PSNR of the three images. We use the average PSNR instead of the PSNR corresponding to the average MSE in accordance to the practice of JPEG 2000 core experiments.

During the development of the JPEG 2000 standard, the committee maintained a software implementation of an encoder and decoder that contained all the technologies considered for the inclusion in the standard as of that time. This was also accompanied by a textual description of the technologies. Both the software and the textual description were referred to as the Verification Model (VM). After each meeting of the JPEG committee, the VM was updated to reflect any approved modifications. For the results in this section, we used the JPEG 2000 Verification Model Version 8.6 (VM8.6) [68]. During each simulation, only one 32 parameter was varied while the others were kept constant in order to study the effect of a single parameter on compression performance.

As a reference implementation, we used the following set of compression parameters: single tile; 5 levels of wavelet decomposition; 64×64 codeblocks; and a single layer. The subbands at each resolution were treated as a single precinct. In case of irreversible $(9,7)$ lossy compression, the reciprocal of the L_2-norm was used as the fundamental quantization step-size for each subband. In the case of reversible $(5,3)$ lossless and lossy compression, the quantizer step-size was set to unity for all subbands as required by the JPEG 2000 standard. Hence, when using the reversible $(5,3)$ filter-bank for lossy compression, rate control is possible only by discarding bits from the integer representation of the index for the quantized wavelet coefficient.

The results for lossy coding are reported for bit-rates of 0.0625, 0.125, 0.5, 1.0 and 2.0 bits/pixel (bpp). To achieve a target bit-rate, the compressed codeblock bit-streams were truncated to form a single layer. The truncation points are determined using the EBCOT post-compression R–D optimization procedure as described in Section 4.2. We chose this alternative instead of varying the quantizer step-size for the following reason. Suppose that a particular step-size is used to achieve a target bit-rate without any truncation of the compressed bit-stream. Then, varying a single coding parameter while keeping the step-size the same results in a different distortion as well as a different rate. In that case, the only meaningful way to compare results is by plotting the rate-distortion curves (as opposed to single R–D points). Hence, it is more effective to present the comparisons in a tabular form by comparing the PSNR's of different coding options for fixed bit-rates by using the EBCOT rate control algorithm.

5.1. Lossy results

5.1.1. Tile size

JPEG 2000 allows spatial partitioning of the image into tiles. Each tile is wavelet transformed and coded independently. In fact, a different number of decomposition levels can be specified for each component of each tile. Smaller tile sizes are particularly desirable in memory-constrained applications or when access to only a small portion of the image is desired (e.g., remotely roaming

over a large image). Table 5 compares the R–D performance of the JPEG 2000 encoder for various tile sizes with the (9, 7) filter. It is evident that the compression performance decreases with decreasing tile size, particularly at low bit-rates. Furthermore, at low bit-rates where the tile boundaries are visible in the reconstructed image, the perceived quality of the image might be lower than that indicated by the PSNR. The impact of the boundary artifacts can be reduced by using post-processing techniques, such as those employed in reducing the blocking artifacts in low bit-rate DCT-based JPEG and MPEG images. Part 2 of the JPEG 2000 standard offers the option of using single-sample overlap DWT (SSO-DWT), which reduces edge artifacts at the tile boundaries.

5.1.2. Codeblock size

The codeblocks in JPEG 2000 are rectangular with user-defined dimensions that are identical for all subbands. Each dimension has to be a power of two, and the total number of samples in a codeblock cannot exceed 4096. Furthermore, when the precinct size in a particular subband is less than the codeblock size, the codeblock size is set equal to the precinct size.

Table 6 compares the effect of varying the codeblock size on R–D performance with the (9, 7) filter. There is very little loss of PSNR (maximum of 0.14 dB) in going from a codeblock size of 64×64 to a codeblock size of 32×32. However, codeblock sizes smaller than 32×32 result in a significant drop in PSNR. There are several factors that contribute to this phenomenon. One factor is the overhead information contained in the packet header. The packet header contains information about the number of coding passes and the length of compressed data for each codeblock, so the total size of the header information increases with an increasing number of codeblocks. Another factor is the independent encoding of each codeblock that requires the re-initialization of the arithmetic coding models. As the codeblock size becomes smaller, the number of samples required to adapt to the underlying probability models constitutes a greater portion of the total number of

Table 5
Comparison of R–D performance for different tile sizes with the (9, 7) filter-bank

Rate (bits/pixel)	Average PSNR in dB				
	No tiling	512×512	256×256	192×192	128×128
0.0625	22.82	22.73 (−0.09)	22.50 (−0.32)	22.22 (−0.60)	21.79 (−1.03)
0.125	24.84	24.77 (−0.07)	24.59 (−0.25)	24.38 (−0.46)	24.06 (−0.78)
0.25	27.61	27.55 (−0.06)	27.41 (−0.20)	27.20 (−0.41)	26.96 (−0.65)
0.5	31.35	31.30 (−0.05)	31.19 (−0.16)	30.99 (−0.36)	30.82 (−0.53)
1.0	36.22	36.19 (−0.03)	36.11 (−0.11)	35.96 (−0.26)	35.85 (−0.37)
2.0	42.42	42.40 (−0.02)	42.34 (−0.08)	42.22 (−0.20)	42.16 (−0.26)

Table 6
Comparison of R–D performance for various codeblock sizes with the (9, 7) filter-bank

Rate (bits/pixel)	Average PSNR in dB			
	64×64	32×32	16×16	8×8
0.0625	22.82	22.78 (−0.04)	22.62 (−0.20)	22.27 (−0.55)
0.125	24.84	24.78 (−0.06)	24.57 (−0.27)	24.13 (−0.71)
0.25	27.61	27.52 (−0.09)	27.23 (−0.38)	26.63 (−0.98)
0.5	31.35	31.22 (−0.13)	30.84 (−0.51)	30.04 (−1.31)
1.0	36.22	36.09 (−0.13)	35.68 (−0.54)	34.70 (−1.52)
2.0	42.42	42.28 (−0.14)	41.83 (−0.59)	40.70 (−1.72)

samples encoded. In addition, the pixels that lie on the boundary of a codeblock have an incomplete context since pixels from neighboring codeblocks cannot be used in forming the coding contexts. As the codeblock size decreases, the percentage of boundary pixels with incomplete contexts increases.

It can also be concluded from Table 6 that the loss in compression performance with decreasing codeblock size is more pronounced at higher bit-rates. This can be explained as follows. When the bit-rate is high, more coding passes are encoded, and the inefficiencies due to model mismatch and incomplete contexts add up. In comparison, at low bit-rates, many codeblocks from the higher frequency subbands contribute no compressed data to the compressed bit-stream. The JPEG 2000 bit-stream syntax has provisions to signal this information very efficiently, and for these codeblocks, a smaller size has almost no impact on the coding efficiency. Moreover, these high frequency subbands represent a large percentage of the total number of codeblocks.

5.1.3. DWT filters

Part 1 of JPEG 2000 offers a choice of either the $(9,7)$ or the $(5,3)$ filter-bank for lossy compression. Fig. 22 compares the energy compaction of the $(9,7)$ and the $(5,3)$ filter-banks graphically. Each subband has been scaled with its L_2-norm to reflect its proper contribution to the overall energy. Moreover, for better visualization of the subband energies, the AC subbands of both images have been scaled up by a factor of two, while the LL subbands have been scaled down by a factor of eight. It can be seen that the LL subband of the $(9,7)$ filter-bank has a higher contrast, which implies superior energy compaction.

Table 7 compares the R–D performance of the two filter-banks. The $(9,7)$ filter-bank consistently outperforms the $(5,3)$ filter-bank with the performance gap increasing with increasing bit-rate. However, it should be noted that the $(5,3)$ filter-bank can also perform lossless compression. Thus, for a particular image, when the target bit-rate equals the lossless bit-rate, the $(5,3)$ filter-bank would result in zero MSE or infinite PSNR, whereas the $(9,7)$ filter-bank would result in a non-zero MSE. Thus for bit-rates in the range of 4.0 bits/pixel or above, the $(5,3)$ filter-bank

Table 7
Comparison of R–D performance of the irreversible $(9,7)$ and the reversible $(5,3)$ filter-banks

Rate (bits/pixel)	Average PSNR in dB	
	Irreversible $(9,7)$	Reversible $(5,3)$
0.0625	22.82	22.37 (−0.45)
0.125	24.84	24.37 (−0.47)
0.25	27.61	27.04 (−0.57)
0.5	31.35	30.74 (−0.61)
1.0	36.22	35.48 (−0.74)
2.0	42.42	41.33 (−1.09)

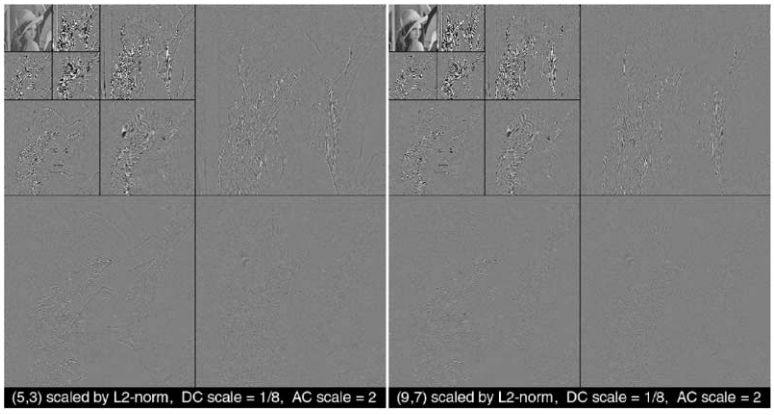


Fig. 22. Energy compaction comparison between the irreversible $(5,3)$ and $(9,7)$ filter-banks.

Table 8
Comparison of R–D performance for various levels of decomposition with the (9, 7) filter-bank

Rate (bits/pixel)	Average PSNR in dB				
	5 levels	4 levels	3 levels	2 levels	1 level
0.0625	22.82	22.77 (−0.05)	22.47 (−0.35)	21.50 (−1.30)	17.68 (−5.12)
0.125	24.84	24.80 (−0.04)	24.62 (−0.22)	23.91 (−0.93)	21.67 (−3.17)
0.25	27.61	27.57 (−0.04)	27.45 (−0.16)	26.94 (−0.67)	25.54 (−2.07)
0.5	31.35	31.33 (−0.02)	31.24 (−0.11)	30.87 (−0.48)	29.71 (−1.64)
1.0	36.22	36.21 (−0.01)	36.15 (−0.07)	35.91 (−0.31)	35.15 (−1.07)
2.0	42.42	42.42 (−0.01)	42.37 (−0.05)	42.26 (−0.16)	41.71 (−0.71)

performs better than the (9, 7) filter-bank. In addition, the 34 (5, 3) filter-bank has much less computational complexity as it can be implemented using only integer arithmetic.

5.1.4. Wavelet decomposition levels

The number of decomposition levels affects the coding efficiency of a JPEG 2000 encoder as well as the number of resolutions at which an image can be decompressed. In general, the number of decomposition levels does not impact the computational complexity significantly because only the LL band is further split at each level. Table 8 compares the R–D performance of the JPEG 2000 encoder for different numbers of decomposition levels with the (9, 7) filter. Our simulations show that the PSNRs resulting from five and eight levels of decomposition are practically indistinguishable. Thus, a 5-level decomposition is adequate even for high-resolution images. Finally, the loss is greatest at lower bit-rates and it tapers off with increasing bit-rate.

5.1.5. Lazy, parallel and lazy–parallel modes

As mentioned in Section 2.4, JPEG 2000 provides several entropy coding options that facilitate the parallel processing of the quantized coefficient bitplanes. The collection of these coding options is termed the parallel mode. Another option that reduces the computational complexity of the entropy encoder (especially at high bit-rates) is the lazy mode, where only the cleanup pass is arithmetic encoded after the fourth most significant bitplane. Table 9 shows the R–D performance of the parallel, lazy and lazy–parallel modes relative to the reference implementation. It can be seen that the loss in PSNR is generally small (0.01–0.3 dB) and increases with increasing bit-rate.

5.1.6. Effect of multiple compression cycles

Table 10 examines the effect of multiple compression cycles on PSNR where an image is compressed and reconstructed multiple times to the same bit-rate. Our reference implementation

Table 9
R–D performance of "lazy", "parallel" and "lazy-parallel" modes with the (9, 7) filter-bank

Rate (bits/pixel)	Average PSNR in dB			
	Reference	Lazy	Parallel	Lazy–parallel
0.0625	22.82	22.81 (−0.01)	22.76 (−0.06)	22.75 (−0.07)
0.125	24.84	24.82 (−0.02)	24.76 (−0.08)	24.74 (−0.10)
0.25	27.61	27.57 (−0.04)	27.49 (−0.12)	27.46 (−0.15)
0.5	31.35	31.28 (−0.07)	31.19 (−0.16)	31.14 (−0.21)
1.0	36.22	36.10 (−0.12)	36.03 (−0.19)	35.94 (−0.28)
2.0	42.42	42.28 (−0.14)	42.22 (−0.20)	42.12 (−0.30)

Table 10
R–D performance of multiple compression cycles with the (9, 7) filter-bank

Rate (bits/pixel)	Average PSNR in dB			
	1 iteration	4 iterations	8 iterations	16 iterations
0.0625	22.82	22.78 (−0.04)	22.77 (−0.05)	22.76 (−0.06)
0.125	24.84	24.80 (−0.04)	24.78 (−0.06)	24.76 (−0.08)
0.25	27.61	27.57 (−0.04)	27.56 (−0.05)	27.54 (−0.07)
0.5	31.35	31.32 (−0.03)	31.30 (−0.05)	31.28 (−0.07)
1.0	36.22	36.19 (−0.03)	36.17 (−0.05)	36.16 (−0.06)
2.0	42.42	42.39 (−0.03)	42.37 (−0.05)	42.36 (−0.06)

Table 11
R–D performance of multiple compression cycles with the (9, 7) filter-bank

Rate (bits/pixel)	Average PSNR in dB				
	Reference	No canvas coordinate system		Canvas coordinate system	
		4 iterations	16 iterations	4 iterations	16 iterations
0.0625	22.82	21.14 (−1.68)	18.58 (−4.24)	22.78 (−0.04)	22.76 (−0.06)
0.125	24.84	22.74 (−2.10)	20.30 (−4.54)	24.80 (−0.04)	24.76 (−0.08)
0.25	27.61	25.16 (−2.45)	22.75 (−4.86)	27.57 (−0.04)	27.54 (−0.07)
0.5	31.35	28.61 (−2.74)	26.40 (−4.95)	31.32 (−0.03)	31.28 (−0.07)
1.0	36.22	33.30 (−2.92)	31.29 (−4.93)	36.19 (−0.03)	36.16 (−0.06)
2.0	42.42	39.26 (−3.16)	37.08 (−5.34)	42.39 (−0.03)	42.36 (−0.06)

with the (9, 7) filter was used in all cases. The post-compression R–D optimization engine is used to achieve the desired bit-rate at each iteration. It can be seen from the table that multiple compression cycles cause very little degradation (0.03–0.08 dB) in compression performance when the compression parameters are held constant.

Table 11 examines the effect of multiple compression cycles when one image column is cropped from the left side in between compression cycles. Two scenarios are explored. In one case, the image is always anchored at (0, 0) so that the canvas coordinate system shifts by one column as the image is cropped in between compression cycles. This changes the alignment of the code-blocks. Furthermore, the column index for the samples changes from odd to even and even to odd, which results in a completely different set of wavelet coefficients. In the other case, the anchoring point is shifted to preserve the codeblock alignment using the canvas coordinate system. In this case, only the wavelet coefficients near the boundary of the image are affected by cropping. From the table it can be seen that maintaining the codeblock alignment leads to superior compression performance. More performance comparisons can be found in [20].

5.1.7. JPEG 2000 versus JPEG baseline

Table 12 compares the R–D performance of JPEG 2000 with JPEG baseline at equivalent bit-rates for the reference test set. Our reference implementation with the (9, 7) filter-bank was used. The JPEG baseline PNSR results were generated by iteratively compressing with JPEG baseline to within 1% of the file size of the JPEG 2000-compressed image (including the file headers). The IJG code with the example luminance q-table and a local Huffman table was used for this purpose [57]. For at least one image from our test set, rates of 0.0625 and 0.125 bits/pixel were not achievable even when using a q-table

with all the entries set to the highest possible value of 255; hence JPEG baseline results for those rates are not listed in Table 12. It can be seen that the use of JPEG 2000 results in about 2–4 dB higher PSNR than JPEG baseline depending on the bit-rate.

5.2. Lossless results

5.2.1. Reversible color transform (RCT)

It is well known that decorrelating the components of an image by applying a color transform improves the coding efficiency. For example, RGB images are routinely transformed into YC_bC_r before applying JPEG compression. In a similar fashion, a lossless component transform can be beneficial when used in conjunction with lossless coding. Table 13 compares the performance of the JPEG 2000 algorithm for lossless coding, with and without applying the RCT transform. The results are based on using the reversible (5, 3) filter-bank with the reference set of compression parameters. Instead of using our reference 8-bit test images, we used the 24-bit color version of "Lena" and "Baboon" images (of size 512×512), in addition to 24-bit versions of the "Bike" and "Woman" images. From the table it can be seen that applying the RCT transform prior to lossless compression results in savings of 0.16–2.39 bpp, which is quite significant in the context of lossless coding.

5.2.2. Lossless encoder options

Tables 14–17 summarize the lossless compression performance of Part 1 of the JPEG 2000 standard as a function of tile size, number of decomposition levels, codeblock size, and

Table 12
R–D performance of JPEG2 000 and JPEG baseline for the "Lena" image

Rate (bits/pixel)	Average PSNR in dB	
	JPEG 2000	JPEG baseline
0.0625	22.82	—
0.125	24.84	—
0.25	27.61	25.65
0.5	31.35	28.65
1.0	36.22	32.56
2.0	42.42	38.24

Table 13
Comparison of lossless bit-rates for color images with and without RCT

Image	Bit-rate in bits/pixel	
	No RCT	RCT
Lena	13.789	13.622
Baboon	18.759	18.103
Bike	13.937	11.962
Woman	13.892	11.502

Table 14
Comparison of average lossless bit-rates (bits/pixel) for different tile sizes

No tiling	512×512	256×256	128×128	64×64	32×32
4.797	4.801	4.811	4.850	5.015	5.551

Table 15
Comparison of average lossless bit-rates (bits/pixel) for different number of decomposition levels

5 levels	4 levels	3 levels	2 levels	1 level	0 levels
4.797	4.798	4.802	4.818	4.887	5.350

Table 16
Comparison of average lossless bit-rates (bits/pixel) for different codeblock sizes

64×64	32×32	16×16	8×8
4.797	4.846	5.005	5.442

Table 17
Comparison of average lossless bit-rates (bits/pixel) for "lazy", "parallel" and "lazy–parallel" modes

Reference	Lazy	Parallel	Lazy–parallel
4.797	4.799	4.863	4.844

lazy–parallel modes. The bit rates have been averaged over the three test images "Café", "Bike" and "Woman" and the reversible (5, 3) filter-bank has been used. A rather surprising finding is that the average lossless performance difference between the one-level and five-level decompositions is very small (<0.1 bpp). This suggests that the three-pass bitplane entropy coding scheme and the associated contexts efficiently exploit the redundancy of correlated samples. There is a small (although significant) performance penalty when using a codeblock size of 16×16 or smaller, or a tile size of 64×64 or smaller. Finally, there is only a slight decrease in coding efficiency when using the "lazy", "parallel" or "lazy–parallel" modes.

Table 18 compares the effect of multiple layers on the lossless coding efficiency. As mentioned in Section 4.2.2, in order to facilitate bit-stream truncation, it is desirable to construct as many layers as possible. However, the number of packets increases linearly with the number of layers, which also increases the overhead associated with the packet headers. As can be seen from the table, the performance penalty for using 50 layers is small for lossless compression. However, this penalty is expected to increase at lower bit-rates [27]. Whereas, increasing the number of layers from 7 to 50 does not linearly increase the lossless bit-rate since the header information for the increased number of packets is coded more efficiently. In particular, the percentage of codeblocks that do not contribute to a given packet increases with the number of layers, and the packet header syntax allows this information to be coded very efficiently using a single bit.

5.2.3. Lossless JPEG 2000 versus JPEG-LS

Table 19 compares the lossless performance of JPEG 2000 with JPEG-LS [69]. Although the JPEG-LS has only a small performance advantage (3.4%) over JPEG 2000 for the images considered in this study, it has been shown that for certain classes of imagery (e.g., the "cmpnd1" compound document from the JPEG 2000 test set), the JPEG-LS bit-rate is only 60% of that of JPEG 2000 [27].

Table 18
Comparison of average lossless bit-rates (bits/pixel) for different number of layers

1 layer	7 layer	50 layer
4.797	4.809	4.829

Table 19
Comparison of average lossless bit-rates (bits/pixel) for JPEG 2000 and JPEG-LS

JPEG 2000	JPEG-LS
4.797	4.633

5.3. Bitplane entropy coding results

In this section, we examine the redundancy contained in the various bitplanes of the quantized wavelet coefficients. These results were obtained by quantizing the wavelet coefficients of the "Lena" image with the default quantization step-size for VM8.6 ("-step 1/128.0"). Since "Lena" is an 8-bit image, the actual step-size used for each band was 2.0 divided by the L_2-norm of that band. This had the effect that equal quantization errors in each subband had roughly the same contribution to the reconstructed image MSE. Hence, the bitplanes in different subbands were aligned by their LSBs. Eleven of the resulting bitplanes were encoded starting with the most significant bitplane.

One way to characterize the redundancy is to count the number of bytes that are generated by each sub-bitplane coding pass. The number of bytes generated from each sub-bitplane coding pass are not readily available unless each coding pass is terminated. However, during post-compression R–D optimization, VM8.6 computes the number of additional bytes needed to uniquely decode each coding pass using a "near optimal length calculation" algorithm [68]. It is not guaranteed that the "near optimal length calculation" algorithm will determine the minimum number of bytes needed for unique decoding. Moreover, it is necessary to flush the MQ-coder registers for estimation of the number of bytes. This means that the estimated bytes for a coding pass contain some data from the next coding pass,

which can lead to some unexpected results. With these caveats in mind, Table 20 contains the number of bytes generated from each sub-bitplane coding pass. The estimated bytes for each coding pass were summed across all the codeblocks in the image to generate these entries.

During the encoding of the first bitplane, there is only a cleanup pass and 36 coefficients turn significant. All of these significant coefficients belong to the 5LL subband. In the refinement pass of the next bitplane, only these 36 coefficients are refined. Surprisingly, the first refinement bit for

Table 20
Coded bytes resulting for sub-bitplane passes of "Lena" image

Bitplane number	"Significance" bytes	"Refinement" bytes	"Clean up" bytes	Total for current BP	Total for all BPs
1	0	0	21	21	21
2	18	0	24	42	63
3	38	13	57	108	171
4	78	37	156	271	442
5	224	73	383	680	1122
6	551	180	748	1479	2601
7	1243	418	1349	3010	5611
8	2315	932	2570	5817	11428
9	4593	1925	5465	11983	23411
10	10720	3917	12779	27416	50827
11	25421	8808	5438	39667	90494

Fig. 23. Reconstructed Lena image after decoding bit-planes 1 through 9 (from left to right and top to bottom).

Table 21
Coding statistics resulting from the encoding of wavelet coefficient bitplanes of "Lena" image

BP	Compression ratio	Rate (bits/pixel)	PSNR (dB)	Percent refined	Percent significant	Percent insignificant
1	12483	0.000641	16.16	0.00	0.01	99.99
2	4161	0.00192	18.85	0.01	0.04	99.95
3	1533	0.00522	21.45	0.05	0.06	99.89
4	593	0.0135	23.74	0.11	0.12	99.77
5	233	0.0343	26.47	0.23	0.32	99.43
6	101	0.0792	29.39	0.57	0.75	98.68
7	47	0.170	32.54	1.32	1.59	97.09
8	23	0.348	35.70	2.91	3.10	93.99
9	11.2	0.714	38.87	6.01	6.33	87.66
10	5.16	1.55	43.12	12.34	15.78	71.88
11	2.90	2.76	49.00	28.12	25.08	46.80

all of these 36 coefficients are zero. Due to the fast model adaptation of the MQ-coder, very few refinement bits are generated for the second bitplane. This, in conjunction with the possibility of overestimating the number of bytes in the cleanup pass of the first bitplane, leads to the rather strange result that the refinement pass for the second bitplane requires zero bytes. It is also interesting that the number of bytes needed to encode a given bitplane is usually greater than the total number of bytes used to encode all of the bitplanes prior to it (except for bitplane 11).

Fig. 23 shows images reconstructed from the first nine bitplanes, and Table 21 provides the corresponding PSNRs. Table 21 also shows the percentage of the coefficients that are refined at each bitplane; the percentage of the coefficients that are found to be significant at each bitplane; and the percentage of the coefficients that remain insignificant after the completion of the encoding of a bitplane. It is interesting to note that about 72% of the coefficients still remain insignificant after encoding the tenth bitplane.

6. Additional features and Part 2 extensions

6.1. Region of interest (ROI) coding

In some applications, it might be desirable to encode certain portions of the image (called the *region of interest* or ROI) at a higher level of quality relative to the rest of the image (called the background). Alternatively, one might want to prioritize the compressed data corresponding to the ROI relative to the background so that it appears earlier in the codestream. This feature is desirable in progressive transmission in case of early termination of the codestream.

Region of interest coding can be accomplished by encoding the quantized wavelet coefficients corresponding to the ROI with a higher precision relative to the background, e.g., by scaling up the ROI coefficients or scaling down the background coefficients. A scaling based ROI encoding method would generally proceed as follows [6]. First, the ROI(s) are identified in the image domain. Next, a binary mask in the wavelet domain, known as the *ROI mask*, is generated. The ROI mask has a value of one at those coefficients that contribute to the reconstruction of the ROI and has a value of zero elsewhere. The shape of the ROI mask is determined by the image domain ROI as well as the wavelet filter-bank, and it can be computed in an efficient manner for most regular ROI shapes [29]. Prior to entropy coding, the bitplanes of the coefficients belonging to the ROI mask are shifted up (or the background bitplanes are shifted down[8]) by a desired amount that can vary from

[8] The main idea is to store the magnitude bits of the quantized coefficients in the most significant part of the implementation register so that any potential precision overflow would only impact the LSB of the background coefficients.

one ROI to another within the same image. The ROI shape information (in the image domain) and the scaling factor used for each ROI is also encoded and included in the codestream. In general, the overhead associated with the encoding of an arbitrary shaped ROI might be large unless the ROI has a regular shape, e.g., a rectangle or a circle, which can be described with a small set of parameters. At the decoder, the ROI shape and scaling factors are decoded, and the quantized wavelet coefficients within each ROI (or background) coefficient are scaled to their original values.

The procedure described above requires the generation of an ROI mask at both the encoder and decoder, as well as the encoding and decoding of the ROI shape information. This increased complexity is balanced by the flexibility to encode ROIs with multiple qualities and to control the quality differential between the ROI and the background. To minimize decoder complexity while still providing ROI capability, JPEG 2000 Part 1 has adopted a specific implementation of the scaling based ROI approach known as the *Maxshift* method [12].

In the Maxshift method, the ROI mask is generated in the wavelet domain, and all wavelet coefficients that belong to the background are examined and the coefficient with the largest magnitude is identified. Next, a value s is determined such that 2^s is larger than the largest magnitude background coefficient, and all bitplanes of the background coefficients are shifted down by s bits. This insures that the smallest nonzero ROI coefficient is still larger than the largest background coefficient as shown in Fig. 24. The presence of ROI is signaled to the decoder by a marker segment and the value of s is included in the codestream. The decoder first entropy decodes all the wavelet coefficients. Those coefficients whose values are less than 2^s belong to the background and are scaled up to their original value. In the Maxshift method, the decoder is not required to generate an ROI mask or to decode any ROI shape information. Furthermore, the encoder can encode any arbitrary shape ROI within each subband, and it does not need to encode the ROI shape information (although it may still need to generate an ROI mask). The main disadvantage of the Maxshift method is that ROIs with multiple quality differentials cannot be encoded.

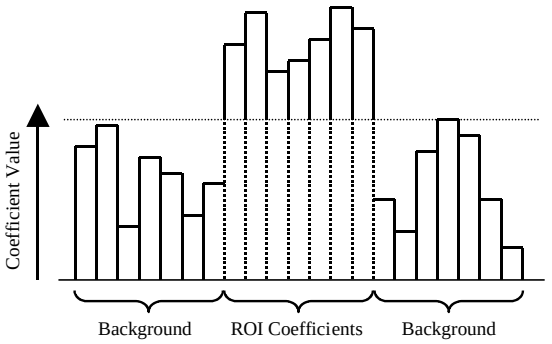


Fig. 24. Maxshift method of ROI coding in Part 1.

In the Maxshift method, the ROI coefficients are prioritized in the codestream so that they are received (decoded) before the background. However, if the entire codestream is decoded, the background pixels will eventually be reconstructed to the same level of quality as that of the ROI. In certain applications, it may be desirable to encode the ROI to a higher level of quality than the background even after the entire codestream has been decoded. The complete separation of the ROI and background bitplanes in the Maxshift method can be used to achieve this purpose. For example, all the wavelet coefficients are quantized to the precision desired for the ROI. The ROI coefficients are encoded first, followed by the encoding of the background coefficients in one or more layers. By discarding a number of layers corresponding to the background coefficients, any desired level of quality can be achieved for the background.

Since the encoding of the ROI and the background coefficients in the Maxshift method are completely disjoint processes, it might seem that the ROI needs to be completely decoded before any background information is reconstructed. However, this limitation can be circumvented to some extent. For example, if the data is organized in the resolution progressive mode, the ROI data is decoded first followed by the background data for each resolution. As a result, at the start of decoding for each resolution, the reconstructed

image will contain all the background data corresponding to the lower resolutions. Alternatively, due to the flexibility in defining the ROI shape for each subband, the ROI mask at each resolution or subband can be modified to include some background information. For example, the entire LL subband can be included in the ROI mask to provide low resolution information about the background in the reconstructed image.

Experiments show that for the lossless coding of images with ROIs, the Maxshift method increases the bit rate by 1–8% (depending on the image size and the ROI size and shape) compared to the lossless coding of the image without ROI [12]. This is a relatively small cost for achieving the ROI functionality.

6.2. Error resilience

Many emerging applications of the JPEG 2000 standard require the delivery of the compressed data over communications channels with different error characteristics. For example, wireless communication channels are susceptible to random and burst channel errors, while internet communication is prone to data loss due to traffic congestion. To improve the transmission performance of JPEG 2000 in error prone environments, Part 1 of the standard provides several options for error resilience. The error resilience tools are based on different approaches such as compressed data partitioning and resynchronization, error detection, and Quality of Service (QoS) transmission based on priority. The error resilience bit-stream syntax and tools are provided both at the entropy coding level and the packet level [24,28].

As discussed before, one of the main differences between the JPEG 2000 coder and previous embedded wavelet coders is in the independent encoding of the codeblocks. Among the many advantages of this approach is improved error resilience, since any errors in the bit-stream, corresponding to a codeblock will be contained within that codeblock. In addition, certain entropy coding options described in Section 2.4.3 can be used to improve error resilience. For example, the arithmetic coder can be terminated at the end of each coding pass and the context probability models can be reset. The optional lazy mode allows the bypassing of the arithmetic coder for the first two coding passes of each bitplane and can help protect against catastrophic error propagation that is characteristic of all variable-length coding schemes. Finally, JPEG 2000 provides for the insertion of error resilience *segmentation symbols* at the end of the cleanup pass of each bitplane that can serve as error detection. The segmentation symbol is a binary "1010" symbol whose presence is signaled in the marker segments. It is coded with the uniform arithmetic coding context, and its correct decoding at the end of each bitplane confirms the correctness of the decompressed data corresponding to that bitplane. If the segmentation symbol is not decoded correctly, the data for that bitplane and all the subsequent bitplanes corresponding to that codeblock should be discarded. This is because the data encoded in the subsequent coding passes of that codeblock depend on the previously coded data.

Error resilience at the packet level can be achieved by using *resynchronization markers*, which provide for spatial partitioning and resynchronization. This marker is placed in front of each packet in a tile, and it numbers the packets sequentially starting at zero. Also, the packet headers can be moved to either the main header (for all tiles) or the tile header to create what is known as *short packets*. In a QoS transmission environment, these headers can be protected more heavily than the rest of the data. If there are errors present in the packet compressed data, the packet headers can still be associated with the correct packet by using the sequence number included in the resynchronization marker. The combination of these error resilience tools can often provide adequate protection in some of the most demanding error-prone environments.

6.3. File format

Most digital imaging standards provide a file format structure to encapsulate the coded image data. While the codestream specifies the compressed image, the file format serves to provide

useful information about the characteristics of the image and its proper use and display. Sometimes the file format includes redundant information that is also included in the codestream, but such information is useful in that it allows trivial manipulation of the file without any knowledge of the codestream syntax. A minimal file format, such as the one used in the JPEG baseline system, includes general information about the number of image components, their corresponding resolutions and bit depths, etc. However, two important components of a more comprehensive file format are *colorspace* and *metadata*. Without this information, an application might not know how to use or display an image properly. The colorspace defines how the decoded component values relate to real world spectral information (e.g., sRGB or YC_bC_r), while the metadata provides additional information about the image. For example, metadata can be used to describe how the image was created (e.g., the camera type or photographer's name) as well as describe how the image should be used (e.g., IPRs related to the image, default display resolution, etc.). It also provides the opportunity to extract information about an image without the need to decode it, which enables fast text-based search in databases. The SPIFF file format defined in Part 3 extensions of the existing JPEG standard [56] was targeted at 8-bit per component sRGB and YC_bC_r images, and there was limited capability for metadata. The file format defined by the JPEG 2000 standard is much more flexible with respect to both the colorspace specification and the metadata embedding.

Part 1 of the JPEG 2000 standard defines a file format referred to as JP2. Although this file format is an optional part of the standard, it is expected to be used by many applications. It provides a flexible, but restricted, set of data structures to describe the coded image data. In order to balance flexibility with interoperability, the JP2 format defines two methods of colorspace specification. One method (known as the *Enumerated* method) limits flexibility, but provides a high degree of interoperability by directly specifying only two colorspaces, sRGB and gray scale (with YC_bC_r support being added through an amendment). Another method known as the *Restricted ICC* (International Color Consortium [53]) method, allows for the specification of a colorspace using a subset of standard ICC profiles, referred to in the ICC specification as three-channel matrix-based and monochrome input profiles. These profiles, which specify a transformation from the reconstructed code-values to the profile connection space (PCS), contain at most three 1-D look-up tables followed by a 3×3 matrix. These profile types were chosen because of their simplicity. The Restricted ICC method can simply be thought of as a data structure that specifies a set of colorspace transformation equations. Finally, the JP2 file format also allows for displaying palletized images, i.e., single component images where the value of the single component represents an index into a palette of colors.

The JP2 file format also defines two mechanisms for defining and embedding metadata in a compressed file. The first method uses a universal unique identifier (UUID) while the second method uses XML [54]. For both methods, the individual blocks of metadata can be embedded almost anywhere in the file. Although very few metadata fields have been defined in the JP2 file format, its basic architecture provides a strong foundation for extension.

Part 2 of the standard defines extensions to the JP2 file format, encapsulated in an extended file format called JPX. These extensions increase the colorspace flexibility by providing more enumerated color spaces (and also allows vendors to register additional values for colorspaces) as well as providing support for all ICC profiles. They also add the capability for specifying a combination of multiple images using composition or animation, and add a large number of metadata fields to specify image history, content, characterization and IPR.

6.4. Part 2: extensions

Decisions that were made by the JPEG 2000 committee about which technologies to include in Part 1 of the JPEG 2000 standard depended

on a number of factors including coding efficiency, computational complexity, and performance for a generic class of images. In addition, there was a strong desire to keep Part 1 free of IPR issues. Most of the technologies that were excluded from Part 1 of the JPEG 2000 standard due to the aforementioned reasons have been included in Part 2. In addition, the file format has been extended as described in Section 6.3. A special feature of the technologies included in Part 2 is the ability to adapt the compression parameters to a specific class of images. Part 2 became a Final Draft International Standard (FDIS) in July of 2001 and is expected to become an International Standard (IS) in November of 2001. The following is a brief description of some of the technologies that are included in Part 2.

6.4.1. Generalized offsets

In Part 1 of the JPEG 2000 standard, unsigned image components with a bit-depth of B bits are shifted down by 2^{B-1}. In Part 2, the default offset is the same as in Part 1, but a generalized offset maybe specified for every image component. This offset is applied before applying any component transformation. For images with sharply peaked histograms, using a generalized offset can result in significantly improved compression performance. When generalized offsets are being used, care must be taken to adjust the number of guard bits to prevent overflows.

6.4.2. Variable scalar quantization offset

This option extends the default scalar quantization method of Part 1 to allow deadzones of different widths for each subband when using floating-point filters. The size of the deadzone is specified by the parameter nz_b, which must lie in the half-open range of $[-1, 1)$. Given the quantizer step-size Δ_b of a particular subband, the size of its deadzone is given by $2(1-nz_b)\ \Delta_b$. A value of $nz_b = 0$ corresponds to a deadzone width that is twice the step-size (as in JPEG 2000 Part 1), while a value of $nz_b = 0.5$ corresponds to a uniform quantizer as is used in the existing JPEG standard. As shown in [46], the resulting parameterized family of the deadzone quantizers still maintains the embedded property described in Section 2.3. In particular, if an M_b-bit quantizer index resulting from a step-size of Δ_b is transmitted progressively starting with the MSB and proceeding to the LSB, the resulting index after decoding only N_b bits is identical to that obtained by using a quantizer with a step-size of $\Delta_b 2^{M_b-N_b}$ and a deadzone parameter of $(nz_b/2^{M_b-N_b})$. For nonzero values of nz_b, the deadzone width rapidly converges to twice the step-size with decreasing N_b, while for a value of $nz_b = 0$, the width of the deadzone always remains at twice the step-size and the resulting embedded quantizers have exactly the same structure.

6.4.3. Trellis coded quantization

Trellis coded quantization (TCQ) [7,21,25] is a form of spatially varying scalar quantization with delayed-decision coding. The wavelet coefficients contained in each codeblock are scanned in the same order as described in Section 2.4.2, and each coefficient is quantized using one of four separate scalar quantizers that have an approximately uniform structure. The specific choice of the scalar quantizer is governed by the restrictions imposed by a finite state machine that is represented as a trellis with eight states. The optimal sequence of the states (i.e., the sequence of the quantizer indices for a particular codeblock) is determined at the encoder by using the Viterbi algorithm. Visual testing of TCQ with the ISO test images has shown a significant improvement in the reconstructed quality of low contrast image features such as textures, skin tones, road features, cloth, woodgrain, and fruit surfaces [11]. The tradeoff is that the computational complexity is significantly increased by using TCQ.

6.4.4. Visual masking

Masking refers to the decreased visibility of a signal due to the presence of a suprathreshold background signal. When applying visual masking to the encoding of the wavelet coefficients, the amplitude of the coefficients is considered to be the background (mask), while the quantization error is considered to be the signal, whose visibility should be minimized. In Part 2 of JPEG 2000, the masking properties of the visual system are

exploited by applying a non-linearity to the wavelet coefficients prior to quantization. The characteristics of the non-linearity may depend on the amplitude of the wavelet coefficient being quantized (referred to as *self-contrast masking*) as well as the quantized amplitude of the neighboring wavelet coefficients (referred to as *neighborhood masking*). The use of visual masking typically improves the reconstructed image quality for low-resolution displays for which the visual system CSF is essentially flat. A key area of improvement is in low amplitude textures such as skin, and the improvement typically becomes greater as the image becomes more complex [51,52]. Another area of improvement is in the appearance of edges with zero transition width in digitally generated graphics images. Finally, in certain applications where images are compressed to a fixed size, the use of visual masking often creates more consistent image quality with variations in image content.

6.4.5. Arbitrary decomposition of tile-components

In Part 2 of the JPEG 2000 standard, it is possible to specify an arbitrary wavelet decomposition for each tile-component. First, the tile-component is decomposed using the octave decomposition that is allowed in Part 1. Then, the subbands resulting from the octave decomposition may be split further. Different subbands may be split further to different depths. Subbands can also be split to different depths in the horizontal and vertical directions, thus allowing subbands to have differing sub-sampling factors in the horizontal and vertical directions. This mode is useful when the intent is to optimize the wavelet decomposition to a particular class of images or even for an individual image [35]. Another application is in memory conservation, where the number of vertical decompositions might be less than horizontal to reduce the line buffering required for the wavelet transform.

6.4.6. Transformation of images

Part 1 of the JPEG 2000 standard permits the use of only two wavelet decomposition filter-banks, the irreversible $(9,7)$ filter-bank and the reversible $(5,3)$ filter-bank. In Part 2 of the standard, arbitrary user specified wavelet decomposition filters [8,9] are permitted, and their category (even- or odd-length), type (irreversible or reversible), and weights are signaled in the codestream. This allows the optimization of the filter coefficients for a particular class of images.

6.4.7. Single sample overlap discrete wavelet transform (SSO-DWT)

When non-overlapping tiles are used, the artifacts at the tile boundaries can be objectionable at low bit-rates. Part 2 of the standard allows the use of tiles that overlap by a single row and column to eliminate the tile-boundary artifacts. The advantage is that the single sample overlap wavelet transformation can still be carried out in a block-based fashion, requiring a much smaller amount of memory than performing a wavelet transformation of the entire image.

6.4.8. Multiple component transformations

Part 1 of the JPEG 2000 standard allows the use of only two inter-component color transformations to decorrelate the components; the ICT that is used with irreversible wavelet filters, and the RCT that is used with reversible wavelet filters. Both of these transformations are designed for three-component RGB input images, so their utility is limited when the image components belong to a different color space or when there are more than three components (e.g., LANDSAT images with six components or CMYK images with four components). Part 2 of the standard provides two general approaches for decorrelating multi-component data. One approach is a generalized method of forming linear combinations of components to reduce their correlation. This may include a linear predictive transform to remove recursive dependencies (e.g., Gramm–Schmidt procedure), or a decorrelating transform (e.g., KLT). Another approach is using a default or a user-specified one-dimensional wavelet transformation in the component direction to decorrelate the components.

6.4.9. Non-linear transformation

Part 2 of the JPEG 2000 standard offers two ways of non-linear transformation of the component samples before any inter-component transform is applied to increase coding efficiency. The two non-linear transformations are gamma-style and look up table (LUT) style. This feature is especially useful when the image components are in the linear intensity domain, but it is desirable to bring them to a perceptually uniform domain for compression that is more efficient. For example, the output of a 12-bit linear sensor or scanner can be transformed to 8 bits using a gamma or a logarithmic function.

6.4.10. Extensions to region of interest (ROI) coding

Part 1 of the JPEG 2000 standard includes limited ROI capability provided by the Maxshift method as described in Section 6.1. The Maxshift method has the advantage that it does not require the transmission of the ROI shape and it can accommodate arbitrary-shaped ROIs. On the other hand, it cannot arbitrarily control the quality of each ROI with respect to the background. Part 2 extends the ROI capability by allowing the wavelet coefficients corresponding to a given ROI to be scaled by an arbitrary scaling factor. However, this necessitates the sending of the ROI information explicitly to the decoder, which adds to the decoder complexity. Part 2 supports only rectangular and elliptic ROIs, and the ROI mask is constructed at the decoder based on the shape information included in the code-stream.

Acknowledgements

The authors would like to thank Paul Jones for his careful review of the entire manuscript and many helpful suggestions that significantly improved the clarity of its presentation. The authors would also like to thank Brian Banister for generating the bitplane results in Section 5.3, and Scott Houchin for providing the material on file format in Section 6.3.

References

[1] M.D. Adams, F. Kossentini, Reversible integer-to-integer wavelet transforms for image compression: performance evaluation and analysis, IEEE Trans. Image Process. 9 (6) (June 2000) 1010–1024.

[2] M.D. Adams, F. Kossentini, JasPer: a software-based JPEG-2000 codec implementation, in: Proceedings of the IEEE International Conference on Image Processing, Vancouver, CA, September 2000. (Also, refer to the JasPer home page at http://www.ece.ubc.ca/~mdadams/jasper/.)

[3] M.D. Adams, H. Man, F. Kossentini, T. Ebrahimi, JPEG 2000: the next generation still image compression standard, ISO/IEC JTC1/SC29/WG1 N1734, June 2000.

[4] M. Albanesi, S. Bertoluzza, Human vision model and wavelets for high-quality image compression, in: Proceedings of the Fifth International Conference in Image Processing and its Applications, Edinburgh, UK, July 1995, Vol. 410, pp. 311–315.

[5] M. Antonini, M. Barlaud, P. Mathieu, I. Daubechies, Image coding using wavelet transform, IEEE Trans. Image Process. 1 (2) (April 1992) 205–220.

[6] E. Atsumi, N. Farvardin, Lossy/lossless region-of-interest image coding based on set partitioning in hierarchical trees, in: Proceedings of the IEEE International Conference on Image Processing, Chicago, USA, October 1998, pp. 87–91.

[7] A. Bilgin, P.J. Sementilli, M.W. Marcellin, Progressive image coding using trellis coded quantization, IEEE Trans. Image Process. 8 (11) (November 1999) 1638–1643.

[8] J.N. Bradley, C.M. Brislawn, Compression of fingerprint data using wavelet vector quantization image compression algorithm, Technical Report, LA-UR-92-1507, Los Alamos National Lab, USA, 1992.

[9] C.M. Brislawn, Classification of nonexpansive symmetric extension transforms for multirate filter-banks, Appl. Comput. Harmon. Anal. 3 (1996) 337–357.

[10] R.C. Calderbank, I. Daubechies, W. Sweldens, B.-L. Yeo, Wavelet transforms that map integers to integers, Appl. Comput. Harmon. Anal. 5 (3) (1998) 332–369.

[11] T.T. Chinen, T.J. Flohr, M.W. Marcellin, TCQ in JPEG 2000, in: Proceedings of the SPIE, San Diego, USA, July/August 2000, Vol. 4115, pp. 552–560.

[12] C. Christopoulos, J. Askelof, M. Larsson, Efficient methods for encoding regions of interest in the upcoming JPEG 2000 still image coding standard, IEEE Signal Process. Lett. 7 (9) (September 2000) 247–249.

[13] C. Christopoulos, A. Skodras, T. Ebrahimi, The JPEG 2000 still image coding system: an overview, IEEE Trans. Consumer Electron. 46 (4) (November 2000) 1103–1127.

[14] C. Chrysafis, A. Ortega, Line-based, reduced memory, wavelet image compression, IEEE Trans. Image Process. 9 (3) (March 2000) 378–389.

[15] I. Daubechies, W. Sweldens, Factoring wavelet transforms into lifting steps, J. Fourier Anal. Appl. 4 (3) (1998) 247–269.
[16] T. Ebrahimi, D. Santa Cruz, J. Askelöf, M. Larsson, C. Christopoulos, JPEG 2000 still image coding versus other standards, in: Proceedings of the SPIE, San Diego, CA, USA, July/August 2000, Vol. 4115, pp. 446–454.
[17] D. Le Gall, A. Tabatabai, Subband coding of digital images using symmetric kernel filters and arithmetic coding techniques, in: Proceedings of the International Conference on Acoustics, Speech Signal Processing, New York, USA, April 1988, pp. 761–764.
[18] M. Gormish, D. Lee, M.W. Marcellin, JPEG 2000: overview, architecture, and applications, in: Proceedings of the IEEE International Conference on Image Processing, Vancouver, CA, September 2000.
[19] P. Jones, S. Daly, R. Gaborski, M. Rabbani, Comparative study of wavelet and DCT decompositions with equivalent quantization and encoding strategies for medical images, in: Proceedings of the SPIE, San Diego, CA, USA, February 1995, Vol. 2431, pp. 571–582.
[20] R.L. Joshi, M. Rabbani, M. Lepley, Comparison of multiple compression cycle performance for JPEG and JPEG 2000, in: Proceedings of the SPIE, San Diego, CA, USA, July/August 2000, Vol. 4115, pp. 492–501.
[21] J.H. Kasner, M.W. Marcellin, B.R. Hunt, Universal trellis coded quantization, IEEE Trans. Image Process. 8 (12) (December 1999) 1677–1687.
[22] J. Li, Visual progressive coding, in: Proceedings of the SPIE, San Jose, USA, January 1999, Vol. 3653.
[23] J. Li, S. Lei, An embedded still image coder with rate-distortion optimization, IEEE Trans. Image Process. 8 (7) (July 1999) 913–924.
[24] J. Liang, R. Talluri, Tools for robust image and video coding in JPEG 2000 and MPEG-4 standards, in: Proceedings of the SPIE, San Jose, CA, January 1999, Vol. 3653, pp. 40–51.
[25] M.W. Marcellin, T.R. Fischer, Trellis coded quantization of memoryless and gauss–markov sources, IEEE Trans. Commun. 38 (1) (January 1990) 82–93.
[26] M. Marcellin, T. Flohr, A. Bilgin, D. Taubman, E. Ordentlich, M. Weinberger, G. Seroussi, C. Chrysafis, T. Fischer, B. Banister, M. Rabbani, R. Joshi, Reduced complexity entropy coding, ISO/IEC JTC1/SC29/WG1 Document N1312, June 1999.
[27] M.W. Marcellin, M. Gormish, A. Bilgin, M. Boliek, An Overview of JPEG-2000, in: Proceedings of the Data Compression Conference, Snowbird, UT, USA, March 2000, pp. 523–541.
[28] I. Moccagata, S. Sodagar, J. Liang, H. Chen, Error Resilient Coding in JPEG-2000 and MPEG-4, IEEE J. Select. Areas Commun. 18 (6) (June 2000) 899–914.
[29] D. Nister, C. Christopoulos, Lossless region of interest with embedded wavelet image coding, Signal Processing 78 (1) (1999) 1–17.
[30] E. Ordentlich, M.J. Weinberger, G. Seroussi, A low complexity modeling approach for embedded coding of wavelet coefficients, in: Proceedings of the Data Compression Conference, Snowbird, UT, USA, March 1998, pp. 408–417.
[31] T. O'Rourke, R. Stevenson, Human visual system based wavelet decomposition for image compression, J. Visual Commun. Image Represent. 6 (1995) 109–121.
[32] W.B. Pennebaker, J.L. Mitchell, JPEG Still Image Data Compression Standard, Van Nostrand Reinhold, New York, 1993.
[33] W.B. Pennebaker, J.L. Mitchell, G.G. Langdon Jr., R.B. Arps, An overview of the basic principles of the Q-coder adaptive binary arithmetic coder, IBM J. Res. Development 32 (6) (November 1988) 717–726.
[34] J.R. Price, M. Rabbani, Biased reconstruction for JPEG decoding, Signal Process. Lett. 6 (12) (December 1999) 297–299.
[35] K. Ramchandran, M. Vetterli, Best wavelet packet bases in a rate-distortion sense, IEEE Trans. Image Process. 2 (2) (April 1993) 160–175.
[36] A. Said, W.A. Pearlman, A new fast and efficient image codec based on set partitioning in hierarchical trees, IEEE Trans. Circuits Systems Video Technol. 6 (3) (June 1996) 243–250.
[37] D. Santa-Cruz, T. Ebrahimi, An analytical study of JPEG 2000 functionalities, in: Proceedings of the IEEE International Conference on Image Processing, Vancouver, CA, September 2000.
[38] J.M. Shapiro, Embedded image coding using zero trees of wavelet coefficients, IEEE Trans. Signal Process. 41 (12), (December 1993) 3445–3462.
[39] M.J. Slattery, J.L. Mitchell, The Qx-coder, IBM J. Res. Development 42 (6) (November 1998) 767–784.
[40] G. Sullivan, Efficient scalar quantization of exponential and Laplacian variables, IEEE Trans. Inform. Theory 42 (5) (September 1996) 1365–1374.
[41] W. Sweldens, The lifting scheme: a construction of second generation wavelets, Siam J. Math. Anal. 29 (2) (1997) 511–546.
[42] W. Sweldens, The lifting scheme: a custom-design construction of biorthogonal wavelets, Appl. Comput. Harmon. Anal. 3 (2) (1996) 186–200.
[43] W. Sweldens, The lifting scheme: a new philosophy in biorthogonal wavelet constructions, Proc. SPIE 2569 (1995) 68–79.
[44] D. Taubman, High performance scalable image compression with EBCOT, IEEE Trans. Image Process. 9 (7) (July 2000) 1158–1170.
[45] D. Taubman, M.W. Marcellin, JPEG 2000: Image Compression Fundamentals, Practice and Standards, Kluwer Academic Publishers, Dordrecht, 2001.
[46] D. Taubman, E. Ordentlich, M.J. Weinberger, G. Seroussi, Embedded block coding in JPEG 2000, Signal Processing: Image Communication 17 (1) (2002) 49–72.
[47] M. Vetterli, J. Kovacevic, Wavelet and Subband Coding, Prentice-Hall, Englewood Cliffs, NJ, 1995.

[48] J.D. Villasenor, B. Belzer, J. Liao, Wavelet filter evaluation for image compression, IEEE Trans. Image Process. 4 (8) (August 1995) 1053–1060.
[49] A.B. Watson, G.Y. Yang, J.A. Solomon, J. Villasenor, Visibility of wavelet quantization noise, IEEE Trans. Image Process. 6 (8) (August 1997) 1164–1175.
[50] J.W. Woods, T. Naveen, A filter based bit allocation scheme for subband compression of HDTV, IEEE Trans. Image Process. 1 (3) (July 1992) 436–440.
[51] W. Zeng, S. Daly, S. Lei, Point-wise extended visual masking for JPEG 2000 image compression, in: Proceedings of the IEEE International Conference on Image Processing, Vancouver, CA, September 2000.
[52] W. Zeng, S. Daly, S. Lei, Visual optimization tools in JPEG 2000, in: Proceedings of the IEEE International Conference on Image Processing, Vancouver, CA, September 2000.
[53] International Color Consortium, ICC Profile Format Specification, ICC.1: 1998–09.
[54] W3C, Extensible Markup Language (XML) 1.0, 2nd Edition, October 2000, http://www.w3.org/TR/Rec-xml.
[55] Information technology – digital compression and coding of continuous-tone still images – Part 1: requirements and guidelines, ISO/IEC International Standard 10918-1, ITU-T Rec. T.81, 1993.
[56] Information technology – digital compression and coding of continuous-tone still images – Part 3: extensions, ISO/IEC International Standard 10918-3, ITU-T Rec. T.84, 1995.
[57] Independent JPEG Group, JPEG Library (version 6b), available from http://www.ijg.org/ or ftp://ftp.uu.net/graphics/jpeg/ (tar.gz format archive).
[58] Call for contributions for JPEG 2000 (JTC 1.29.14, 15444): image coding system, ISO/IEC JTC1/SC29/WG1 N505, March 1997.
[59] New work item: JPEG 2000 image coding system, ISO/IEC JTC1/SC29/WG1 N390R, March 1997.
[60] Information technology – JPEG 2000 image coding system, ISO/IEC International Standard 15444-1, ITU Recommendation T.800, 2000.
[61] Information Technology – JPEG 2000 Image Coding System: Part II Extensions, ISO/IEC Final Draft International Standard 15444-2, ITU Recommendation T.801, August 2001.
[62] Information Technology – JPEG 2000 Image Coding System: Motion JPEG 2000, ISO/IEC Final Draft International Standard 15444-3, ITU Recommendation T.802, September 2001.
[63] Information Technology – JPEG 2000 Image Coding System: Compliance Testing, ISO/IEC Committee Draft 15444-4, ITU Recommendation T.803, July 2001.
[64] Information Technology – JPEG 2000 Image Coding System: Part 5 – Reference Software, ISO/IEC Final Draft International Standard 15444-5, ITU Recommendation T.804, August 2001.
[65] JJ2000: An implementation of JPEG 2000 in JAVA™, available at http://jj2000.epfl.ch.
[66] Information Technology – JPEG 2000 Image Coding System: Part 6 – Compound Image File Format, ISO/IEC Pre-Release Final Committee Draft 15444-6, ITU Recommendation T.805, October 2001.
[67] JBIG2 bi-level image compression standard, ISO/IEC International Standard 14492 and ITU Recommendation T.88, 2000.
[68] JPEG 2000 Verification Model 8.6 (Software), ISO/IEC JTC1/SC29/WG1 N1894, December 2000.
[69] Information technology – lossless and near lossless compression of continuous-tone still images, ISO/IEC International Standard 14495-1 and ITU Recommendation T.87, 1999.