1

JPEG 2000 Core Coding System (Part 1)

Majid Rabbani, Rajan L. Joshi, and Paul W. Jones

1.1 Introduction

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.' In 1993, the committee published an International Standard known as JPEG Part 1 (ISO/IEC, 1993; Pennebaker and Mitchell, 1993), which provides a toolkit of compression techniques from which applications can select various elements to satisfy particular requirements. This toolkit includes: (i) the JPEG baseline system, which is a discrete cosine transform (DCT)-based lossy compression algorithm that uses Huffman coding and is restricted to 8 bits/pixel input; (ii) an extended system, which provides enhancements to the baseline algorithm, such as 12 bits/pixel input and arithmetic coding, to satisfy a broader set of applications; and (iii) a lossless mode, which uses a predictive coding method that is independent of the DCT. Of these various components, only the JPEG baseline system has seen widespread adoption. While the JPEG baseline system offers good compression efficiency with modest computational complexity, its success as an image compression standard can really be attributed to the availability of a free software implementation that was released by the Independent JPEG Group (IJG).[2] The IJG code allowed developers to build efficient and robust JPEG applications quickly.

Despite the phenomenal success of the JPEG baseline system, it has various limitations that hinder its use in applications such as medical diagnostic imaging, mobile communications, digital cinema, enhanced internet browsing, and multimedia. The extended JPEG

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

[2] Independent JPEG Group, JPEG Library (Version 6b), available from http://www.ijg.org/ or ftp://ftp.uu.net/graphics/jpeg/ (tar.gz format archive).

The JPEG 2000 Suite Edited by Peter Schelkens, Athanassios Skodras and Touradj Ebrahimi

system addresses only some of these limitations, while also being subject to intellectual property rights (IPR) issues for certain technology components. 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 (ISO/IEC, (1997a, 1997b)), 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.' After evaluating more than 20 algorithms and performing hundreds of technical studies known as 'core experiments,' the JPEG committee issued JPEG 2000 Part 1 as an International Standard in December 2000 (ISO/IEC International Standard 15444-1, ITU Recommendation T.800).

In the same vein as the JPEG baseline system, Part 1 of JPEG 2000 defines a core coding system that provides high coding efficiency with minimal complexity, and is aimed at satisfying 80% of potential applications (ISO/IEC International Standard 15444-1, ITU Recommendation T.800). In addition, it defines an optional file format that includes essential information for the proper rendering of the image. To help drive the adoption of Part 1, the JPEG 2000 committee worked diligently to ensure that it could be made available on a royalty- and fee-free basis. Most of the technologies that were excluded from Part 1 of the JPEG 2000 standard because of IPR or complexity issues were included in Part 2 (ISO/IEC International Standard 15444-2, ITU Recommendation T.801). The division of the JPEG 2000 standard into various parts allows an application to minimize complexity by using only those parts that are needed to satisfy its requirements.

The incentive behind the development of the JPEG 2000 system was not just to provide higher compression efficiency than the baseline JPEG system. Instead, 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, Part 1 of the JPEG 2000 standard addresses some of the shortcomings of baseline JPEG by providing the following features:

- Improved compression efficiency
- Lossy to lossless compression
- Multiple resolution representation
- Embedded bit-stream, including progressive decoding and signal-to-noise ratio (SNR) scalability
- Tiling
- Region-of-interest (ROI) coding
- Error resilience
- Random code-stream access and processing
- Improved performance to multiple compression/decompression cycles
- Flexible file format

The JPEG 2000 standard makes use of several advances in compression technology in order to achieve these features. The block-based DCT of JPEG has been replaced by the full-frame discrete wavelet transform (DWT). The DWT inherently provides a multiresolution image representation, and it also improves compression efficiency because of 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 dead zone (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 coarser quantization, where the effective quantizer step size is equivalent to the scaling of the original step size by a power of two. To encode the binary bit-planes of the quantizer index, JPEG 2000 has replaced the Huffman encoder of baseline JPEG with a context-based adaptive binary arithmetic encoder that is known as the MQ-coder. The embedded bit-stream that results from bit-plane coding provides SNR scalability in addition to the capability of compressing to a target file size. Furthermore, the bit-planes in each subband are encoded in independent rectangular blocks and in three fractional bit-plane passes to provide an optimal embedded bit-stream, improved error resilience, partial spatial random access, ease of certain geometric manipulations, and an extremely flexible code-stream 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., in addition to improved performance to multiple compression/decompression cycles.

Several excellent review papers on JPEG 2000 Part 1 have appeared in the literature (Adams *et al.*, 2000; Christopoulos, Skodras, and Ebrahimi, 2000; Ebrahimi *et al.*, 2000; Gormish, Lee, and Marcellin, 2000; Marcellin *et al.*, 2000; Rabbani and Joshi, 2000; Santa-Cruz and Ebrahimi, 2000; Taubman and Marcellin, 2002). Two comprehensive books describing the technical aspects of the standard have been published (Acharya and Tsai, 2005; Taubman and Marcellin, 2001) and a Special Journal Issue was dedicated to JPEG 2000 (SPIC, 2002). The goal of this chapter is to provide a technical description of the fundamental building blocks of JPEG 2000 Part 1 and to explain the rationale behind the selected technologies. Emphasis is placed on general encoder and decoder technology issues to provide a better understanding of the standard in various applications, and many specific implementation issues have been omitted. As a result, readers who plan on implementing the standard are encouraged to refer to the actual standard (ISO/IEC International Standard 15444-1, ITU Recommendation T.800). It is worth noting that the standard document is written from the standpoint of the decoder, while the focus of this chapter is mainly from the standpoint of the encoder.

This chapter is organized as follows. In Section 1.2, the fundamental building blocks of the JPEG 2000 Part 1 standard, such as preprocessing, DWT, quantization, and entropy coding, are described. In Section 1.3, the syntax and organization of the compressed bit-stream is explained. In Section 1.4, 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 1.5, the tradeoffs between the various choices of encoder parameters are illustrated through an extensive set of examples. Finally, Section 1.6 contains a brief description of some additional JPEG 2000 Part 1 features such as ROI, error resilience, and file format.

1.2 JPEG 2000 Fundamental Building Blocks

The fundamental building blocks of a JPEG 2000 encoder are shown in Figure 1.1. These components include preprocessing, DWT, quantization, arithmetic coding (tier-1 coding),

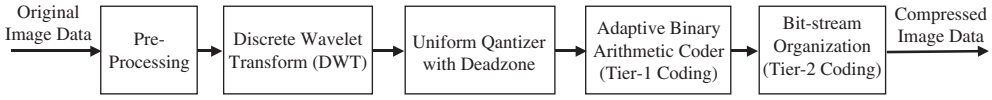


Figure 1.1 JPEG 2000 fundamental building blocks

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 multispectral 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 to 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 when estimating the distortion at the encoder.

1.2.1 Preprocessing

The first step in preprocessing is to partition the input image into rectangular and nonoverlapping 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 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 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

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.16875 & -0.33126 & 0.500 \\ 0.500 & -0.41869 & -0.08131 \end{pmatrix} \times \begin{pmatrix} R \\ G \\ B \end{pmatrix}. \tag{1.1}$$

This can alternatively be written as

$$Y = 0.299(R - G) + G + 0.114(B - G), \quad C_b = 0.564(B - Y), \quad C_r = 0.713(R - Y),$$

while the inverse ICT is given by

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.0 & 0 & 1.402 \\ 1.0 & -0.34413 & -0.71414 \\ 1.0 & 1.772 & 0 \end{pmatrix} \times \begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix}. \tag{1.2}$$

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

$$\tilde{Y} = \left\lfloor \frac{R + 2G + B}{4} \right\rfloor, \quad D_b = B - G, \quad D_r = R - G, \tag{1.3}$$

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

$$G = \tilde{Y} - \left\lfloor \frac{D_b + D_r}{4} \right\rfloor, \quad B = D_b + G, \quad R = D_r + G. \tag{1.4}$$

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.

Because 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 component, after the application of an intercomponent transform if one is used.

1.2.2 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 multiresolution image representation is inherent to the 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.

1.2.2.1 The 1-D DWT

The forward 1-D DWT at the encoder is best understood as successive applications of a pair of lowpass and highpass filters, followed by downsampling by a factor of two

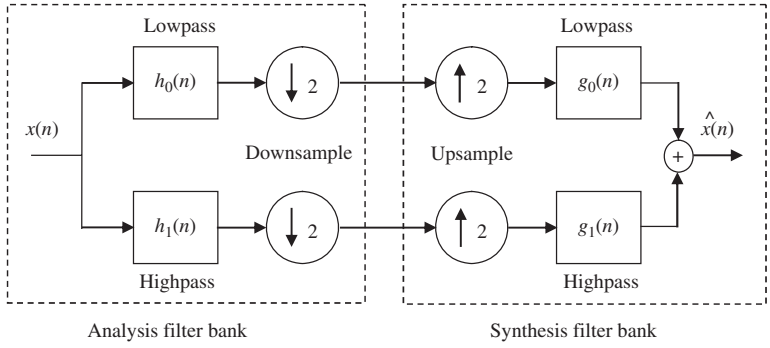


Figure 1.2 1-D, two-band wavelet analysis and synthesis filter banks

(e.g. discarding odd indexed samples) after each filtering operation, as shown in Figure 1.2. The lowpass and highpass filter pair is known as an *analysis filter bank*. The lowpass filter preserves the low frequencies of a signal while attenuating or eliminating the high frequencies, and the result is a blurred version of the original signal. Conversely, the highpass 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 lowpass and highpass filters designated by $h_0(n)$ and $h_1(n)$, respectively. An example of a lowpass 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 highpass 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 LeGall and Tabatabai (1988) and is often referred to as the (5, 3) filter bank, indicating a lowpass filter of length five and a highpass 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 Figure 1.3. The extension for even-tap filters (allowed in Part 2 of the standard) is more complicated and is explained in JPEG 2000 Part 2 (ISO International Standard 15444-2, ITU Recommendation T.801).

The filtered samples that are outputted 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 lowpass or the highpass 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

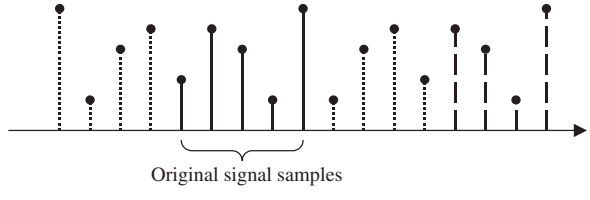


Figure 1.3 Symmetric and periodic extension of the input signal at boundaries

input signal with respect to the canvas coordinate system, which will be discussed in Section 1.3.1. 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.

Reconstruction of a signal from the wavelet coefficients at the decoder is performed with another pair of lowpass and highpass filters (g_0, g_1), known as the *synthesis filter bank*. Referring to Figure 1.2, the downsampled output of the lowpass 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 lowpass filter $g_0(n)$. The downsampled output of the highpass filter $h_1(n)$ is also upsampled and filtered with the synthesis highpass 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:

$$H_0(z)G_0(z) + H_1(z)G_1(z) = 2, \tag{1.5}$$

$$H_0(-z)G_0(z) + H_1(-z)G_1(z) = 0, \tag{1.6}$$

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 Equation (1.6) can be satisfied by choosing

$$G_0(z) = -cz^{-l}H_1(-z) \quad \text{and} \quad G_1(z) = cz^{-l}H_0(-z), \tag{1.7}$$

where l is an integer constant and c is a scaling factor. Combining this result with Equation (1.5) indicates that the analysis filter pair (h_0, h_1) has to be chosen to satisfy

$$-cz^{-l}H_0(z)H_1(-z) + cz^{-l}H_1(z)H_0(-z) = 2. \tag{1.8}$$

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 *biorthogonal* 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 biorthogonal filters is one where the analysis and synthesis filters are finite impulse response (FIR) and linear phase (i.e. they satisfy certain symmetry conditions) (Vetterli and Kovacevic, 1995). Then, it can be shown that in order to satisfy Equation (1.8), 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 for even-tap filters the length difference can only be an even multiple of two.

While the (5, 3) filter bank is a prime example of a biorthogonal filter bank with integer taps, the filter banks that result in the highest compression efficiency often have floating-point taps (Villasenor, Belzer, and Liao, 1995). The most well-known filter bank in this category is the Daubechies (9, 7) filter bank, introduced in Antonini *et al.* (1992) and characterized by the filter taps given in Table 1.1. For comparison, the analysis and synthesis filter taps for the integer (5, 3) filter bank are specified in Table 1.2. It can be easily verified that these filters satisfy Equations (1.7) and (1.8) with $l = 1$ and $c = 1.0$. As

Table 1.1 Analysis and synthesis filter taps for the floating-point Daubechies (9, 7) filter bank

n	Lowpass, $h_0(n)$	Lowpass, $g_0(n)$
0	+0.602949018236360	+1.115087052457000
±1	+0.266864118442875	+0.591271763114250
±2	–0.078223266528990	–0.057543526228500
±3	–0.016864118442875	–0.091271763114250
±4	+0.026748757410810	

n	Highpass, $h_1(n)$
–1	+1.115087052457000
–2, 0	–0.591271763114250
–3, 1	–0.057543526228500
–4, 2	+0.091271763114250

n	Highpass, $g_1(n)$
1	+0.602949018236360
0, 2	–0.266864118442875
–1, 3	–0.078223266528990
–2, 4	+0.016864118442875
–3, 5	+0.026748757410810

Table 1.2 Analysis and synthesis filter taps for the integer (5, 3) filter bank

n	$h_0(n)$	$g_0(n)$
0	3/4	+1
±1	1/4	+1/2
±2	–1/8	

n	$h_1(n)$
–1	+1
–2, 0	–1/2

n	$g_1(n)$
1	+3/4
0, 2	–1/4
–1, 3	–1/8

is evident from Tables 1.1 and 1.2, 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 lowpass output and the odd-indexed samples from the highpass output sequence, where the indices are defined relative to the reference grid (Section 1.3.1).

The frequency responses of the (9, 7) and (5, 3) analysis filter pairs are shown in Figure 1.4. For convenience, the filter amplitudes have been normalized to approximately

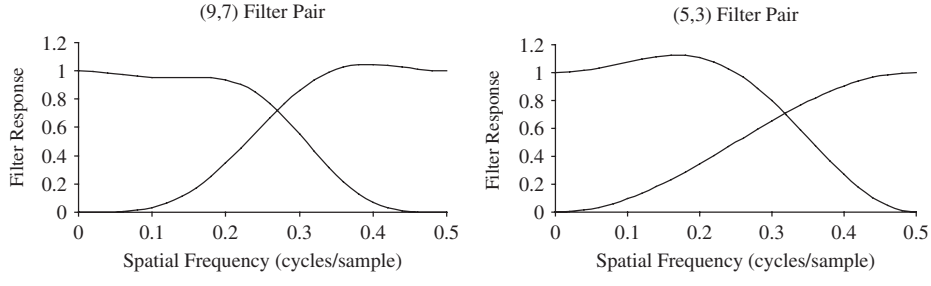


Figure 1.4 Frequency responses of (9, 7) and (5, 3) filter pairs with (1, 1) normalization

the same range (refer to Section 1.2.2.4 for the actual normalizations). It is easy to see the lowpass and highpass nature of the filter pairs, with the (9, 7) filter pair having better frequency discrimination than the (5, 3) filter pair. However, even the (9, 7) filter pair has substantial overlap between the lowpass and highpass filters. For either filter pair, the frequency content that is above a normalized frequency of 0.25 cycles/sample for the lowpass filter and below 0.25 cycles/sample for the highpass filter will alias when the filtered outputs are decimated by a factor of two. While it may seem surprising, the perfect reconstruction property guarantees that the aliased content will be canceled during reconstruction in the absence of any errors in the wavelet coefficients.

After the 1-D signal has been decomposed into two frequency bands, the lowpass 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 highpass output. In most DWT decompositions, only the lowpass output is further decomposed to 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. Figure 1.5 shows an example of the effective filter responses that are produced by recursive filtering of the lowpass output in a dyadic decomposition. This example uses the (9, 7) filter pair and illustrates a five-level decomposition, which produces six frequency bands (i.e. resolution levels). The frequency discrimination of the frequency bands becomes increasingly tighter as one moves from the highest frequency band to the lowest frequency band.

1.2.2.2 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. 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 an ideal implementation.

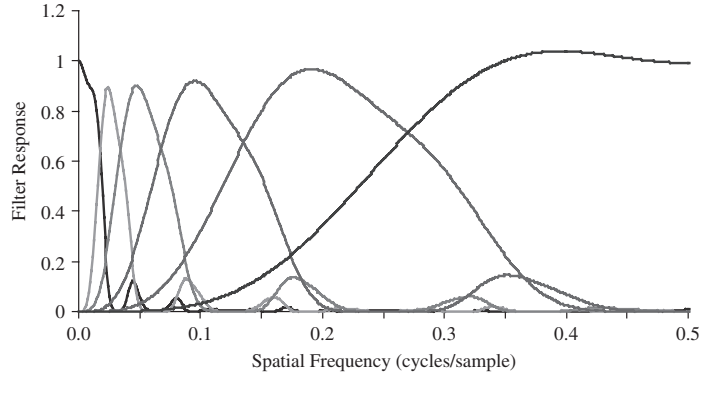


Figure 1.5 Frequency responses for a five-level dyadic decomposition (six frequency bands) using the (9, 7) filter pair (see Plate 1)

This separable filtering with a lowpass/highpass pair creates four frequency bands, and correspondingly there are four sets of wavelet coefficients. The sets of wavelet coefficients represent filtered and subsampled versions of the input image, which are referred to as subband images, or simply *subbands*. Thus, one-level, 2-D wavelet decomposition produces four subbands. In a 2-D dyadic decomposition, the lowest frequency subband (denoted as the LL band to indicate lowpass filtering in both directions) is further decomposed into four smaller subbands, and this process may be repeated as desired to achieve gains in compression efficiency and/or for easy access to lower resolution versions of the image. Figure 1.6 shows a three-level, 2-D dyadic decomposition and the corresponding labeling for each subband. For example, the subband label kHL indicates that a horizontal highpass (H) filter has been applied to the rows, followed by a vertical lowpass (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). Figure 1.7 shows a three-level, 2-D DWT decomposition of the 'Lena' image using the (9, 7) filter bank as specified in Table 1.1, 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 visualize the subband energies better, 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 1.2.2.4, in order to show the actual contribution of each subband to the overall image energy, the wavelet coefficients in each subband should be scaled by the weights given in the last column of Table 1.3.

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 Figure 1.6, the 3LL subband would correspond to resolution zero for a three-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

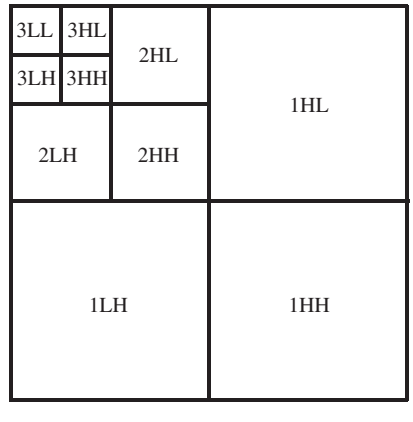


Figure 1.6 Labeling of subbands produced by a three-level, 2-D wavelet decomposition

[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.

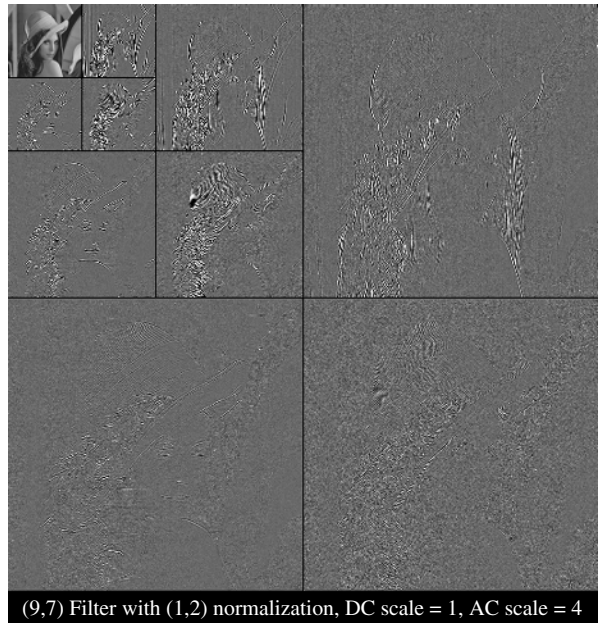


Figure 1.7 Subbands for a three-level, 2-D wavelet decomposition of Lena using the (9, 7) filter bank

Table 1.3 L_2-norms of the DWT subbands after a 2-D, three-level wavelet decomposition

	$(\sqrt{2}, \sqrt{2})$ Normalization		(1, 2) Normalization	
Subband	(5, 3) filter	(9, 7) filter	(5, 3) filter	(9, 7) filter
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

$(r - 1)$. These subbands (excluding the lower-resolution image) are referred to as belonging to resolution r. Resolution zero consists of only the N_LLL band. Because the subbands are encoded independently, the image can be reconstructed at any resolution level by simply decoding those portions of the code-stream that contain the subbands corresponding to that resolution and all the previous resolutions. For example, referring to Figure 1.6, the image can be reconstructed at resolution two by combining the resolution one image and the three subbands labeled 2HL, 2LH, and 2HH.

1.2.2.3 The DWT as a Basis Function Decomposition

The DWT can also be viewed as a basis function decomposition. The basis functions are simply the impulse responses of the spatial filters that comprise the wavelet filter bank. In a dyadic decomposition, the low-frequency filters have large basis functions (corresponding to a narrow range of frequencies) while the high-frequency filters have small basis functions (corresponding to a wide range of frequencies). Because the analysis and synthesis filter banks are different, the corresponding analysis and synthesis basis functions will also be different.

The basis function viewpoint is most useful when considering the reconstruction process that occurs with the inverse DWT during decompression. The basis functions are the fundamental building blocks that are used to reconstruct an image, and the wavelet coefficients indicate how much of each basis function is needed at a given point in the image. Wavelet coefficient errors that are introduced during compression will result in artifacts in the decompressed image that have the appearance of either isolated basis functions or combinations of basis functions. Figure 1.8 shows the 2-D synthesis basis functions that are used to reconstruct images with the (9, 7) filter pair and a three-level dyadic decomposition. It can be seen that the basis functions vary in size (corresponding to the various resolution levels) and also in orientation (corresponding to the various lowpass/highpass filtering combinations in the horizontal and vertical directions).

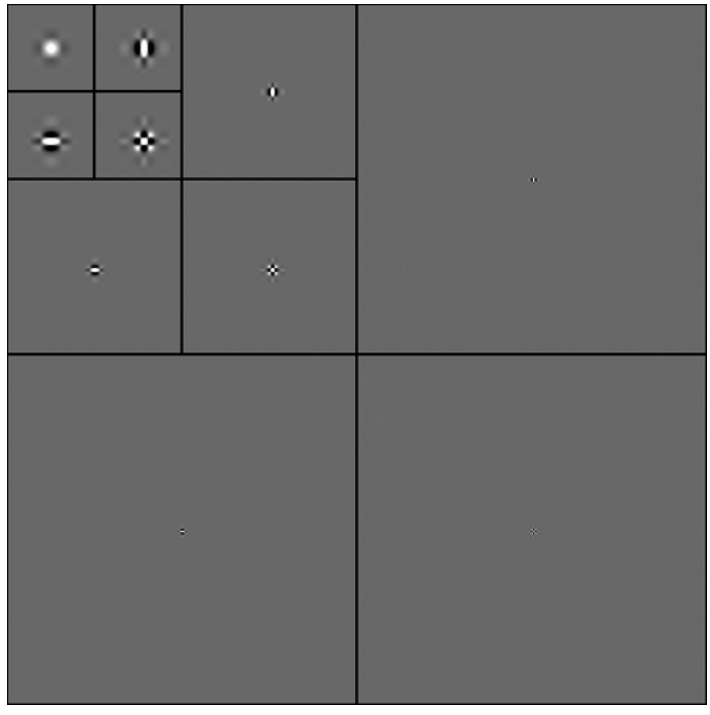


Figure 1.8 Wavelet basis functions for reconstructing an image from a three-level decomposition using the (9, 7) synthesis filters

A key aspect of the DWT is that adjacent basis functions overlap each other during image reconstruction. This overlap results in compression artifacts that are much smoother than the blocky artifacts that are produced by the nonoverlapping basis functions of the block-based DCT that is used in JPEG and MPEG compression. This is one reason why wavelet-based compression techniques can often yield superior quality at low bit rates as compared to DCT-based techniques. Figure 1.9 illustrates the smooth nature of

(a)

(b)

Figure 1.9 Reconstruction of the Lena image using wavelet basis functions from a three-level decomposition: (a) single lowest frequency basis function and (b) four lowest frequency basis functions

images that are reconstructed with wavelet basis functions. In Figure 1.9(a), the Lena image is reconstructed with only the lowest frequency basis function of the three-level decomposition; in Figure 1.9(b), the four lowest frequency basis functions are used for the reconstruction.

1.2.2.4 Wavelet 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 the case of DWT filters, the analysis filters h_0 and h_1 can be normalized arbitrarily. Referring to Equation (1.8), 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 lowpass analysis filter h_0 and the Nyquist gain of the highpass 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

$$G_{DC} = \left|\sum\nolimits_n h(n)\right|, \quad G_{Nyquist} = \left|\sum\nolimits_n (-1)^n h(n)\right|. \tag{1.9}$$

The (9, 7) and the (5, 3) analysis filter banks as defined in Tables 1.1 and 1.2 have been normalized so that the lowpass filter has a DC gain of 1 and the highpass filter has a Nyquist gain of 2. This is referred to as the (1, 2) normalization and 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 Equation (1.8).

In the baseline 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 of the mean squared error (MSE) of the quantized DCT coefficients being equal to 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.

If we consider a 1-D DWT, the reconstructed signal $\hat{x}(n)$ can be expressed as a weighted sum of the 1-D 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,

$$\hat{x}(n) = \sum_b \sum_m y_b(m)\psi_m^b(n). \tag{1.10}$$

In general, the basis functions of a DWT decomposition are not orthogonal; hence, Parseval's theorem does not apply. Woods and Naveen (1992) have shown that for quantized wavelet coefficients under certain assumptions on the quantization noise, the MSE of the

reconstructed image can be approximately expressed as a weighted sum of the MSE of the wavelet coefficients, where the weight for subband b is

$$\alpha_b^2 = \sum_n \left|\psi^b(n)\right|^2. \tag{1.11}$$

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 tile component R_I (after application of an intercomponent transform, if one is used). 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 1.3 shows the L_2-norms of the DWT subbands after a three-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.

1.2.2.5 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, which included such issues 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 1.5.1.1). 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 (Chrysafis and Ortega, 2000). An alternative implementation of the DWT has also been developed, known as the *lifting scheme* (Daubechies and Sweldens, 1998; Sweldens, 1995, 1996, 1998). In addition to providing a significant reduction in the memory and the computational complexity of the

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

DWT, lifting provides in-place computation of the wavelet coefficients by overwriting the memory 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. Figure 1.10 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 Figure 1.10, for the (5, 3) filter bank, the prediction step consists of averaging the two neighboring even sequence pixels and subtracting the average from the odd sample value, i.e.

$$d_i^1 = d_i^0 - \frac{1}{2}\left(s_i^0 + s_{i+1}^0\right). \tag{1.12}$$

Because of the simple structure of the (5, 3) filter bank, the output of this stage, $\{d_i^1\}$, is actually the highpass 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.

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 Figure 1.10, for the (5, 3) filter bank, the update step consists of the following:

$$s_i^1 = s_i^0 + \frac{1}{4}\left(d_{i-1}^1 + d_i^1\right). \tag{1.13}$$

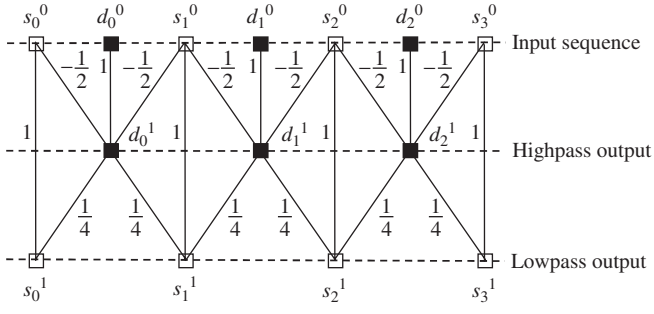


Figure 1.10 Lifting prediction/update steps for the (5, 3) filter bank

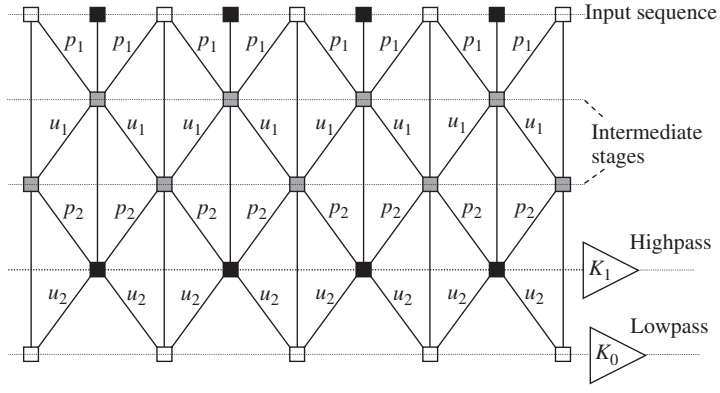
For the (5, 3) filter bank, the output of this stage, $\{s_i^1\}$, is actually the lowpass output of the DWT filter. Again, the number of odd pixels employed in the update and the actual weights applied to each sample depend on the specific DWT filter bank. The prediction and update steps are generally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

$$d_i^n = d_i^{n-1} + \sum_k P_n(k) s_k^{n-1}, \quad n \in [1, 2, \ldots, N], \tag{1.14}$$

$$s_i^n = s_i^{n-1} + \sum_k U_n(k) d_k^n, \quad n \in [1, 2, \ldots, N], \tag{1.15}$$

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 highpass coefficients up to a scaling factor K_1, while the output of the final update step will be the lowpass 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 1.1) are shown in Figure 1.11. The general block diagram of the lifting process is shown in Figure 1.12.

A nice feature of the lifting scheme is that it makes the construction of the inverse transform straightforward. Referring to Figure 1.12 and working from right to left, first the lowpass and highpass 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



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

Figure 1.11 Lifting steps for the (9, 7) filter bank

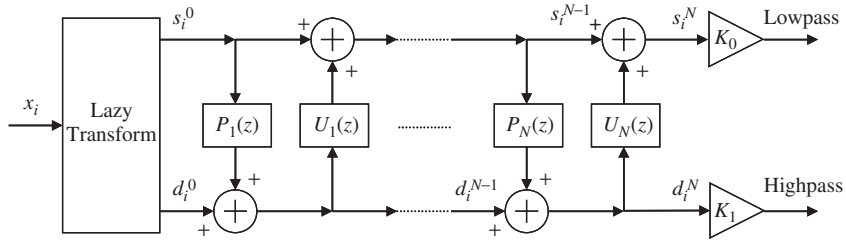


Figure 1.12 General block diagram of the lifting process

update is undone in the reverse order that it was constructed at the encoder until the image samples have been reconstructed.

1.2.2.6 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 (Adams and Kossentini, 2000; Calderbank *et al.*, 1998).

This can be best understood by referring to Figure 1.13, 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 lowpass output will have roughly the same bit precision as that of the input sample, while the highpass output will have an extra bit of precision. This is because, for input samples with a large enough dynamic range

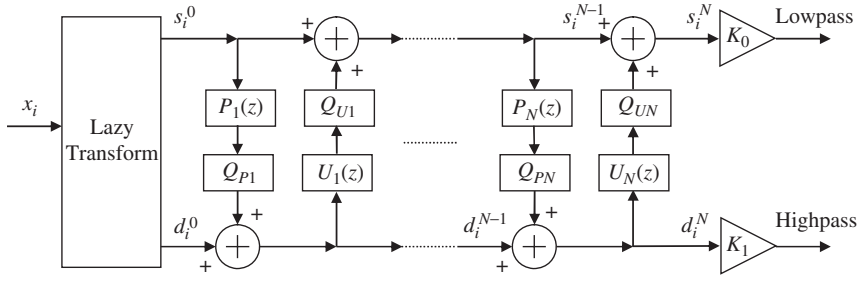


Figure 1.13 General block diagram of a forward integer-to-integer transform using lifting

(e.g. 8 bits or higher), rounding at each lifting step has 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. In a JPEG 2000 encoder, the columns are processed first with the integer-to-integer wavelet transform, followed by the rows. Hence, in a JPEG 2000 decoder, the order of the 2-D reversible wavelet transform is rows, followed by columns. An extensive performance evaluation and analysis of reversible integer-to-integer DWT for image compression has been published in Adams and Kossentini (2000).

As an example, consider the conversion of the (5, 3) filter bank into an integer-to-integer transform by adding the two quantizers $Q_{P1}(w) = -\lfloor -w \rfloor$ and $Q_{U1}(w) = \lfloor w + 1/2 \rfloor$ to the prediction and update steps, respectively, in the lifting diagram of Figure 1.10. The resulting forward transform is given by

$$\begin{cases} y(2n+1) = x(2n+1) - \left\lfloor \dfrac{x(2n) + x(2n+2)}{2} \right\rfloor \\ y(2n) = x(2n) + \left\lfloor \dfrac{y(2n-1) + y(2n+1) + 2}{4} \right\rfloor \end{cases}. \tag{1.16}$$

The required precision of the lowpass band stays roughly the same as the original sample, while the precision of the highpass band grows by one bit. The inverse transform, which losslessly recovers the original sample values, is given by

$$\begin{cases} x(2n) = y(2n) - \left\lfloor \dfrac{y(2n-1) + y(2n+1) + 2}{4} \right\rfloor \\ x(2n+1) = y(2n+1) - \left\lfloor \dfrac{x(2n) + x(2n+2)}{2} \right\rfloor \end{cases}. \tag{1.17}$$

1.2.2.7 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 1.1), 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 Equations (1.16) and (1.17). This choice was driven by requirements for low implementation complexity and lossless capability. The performance of these filters is compared in Section 1.5.1.3. The mathematical properties of the (5, 3) and (9, 7) filters, such as Riesz bounds, order of approximation, and regularity, have been studied in great detail (Unser and Blu, 2003), but are beyond the scope of this chapter. Part 2 of the JPEG 2000 standard allows for arbitrary filter specifications in the code-stream, including filters with an even number of taps.

1.2.3 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 dead zone in the quantizer. It was shown in Sullivan (1996) that the rate-distortion (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 dead zone. The size of the optimal dead zone 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 dead zone has twice the quantizer step size, as depicted in Figure 1.14, while in Part 2, the size of the dead zone can be parameterized to have a different value for each subband.

Part 1 adopted the dead zone with twice the step size due to its optimal embedded structure (Marcellin *et al.*, 2002). 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 (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 code-stream and still produce exactly the same image that would have been encoded at the bit-rate corresponding to the truncated code-stream. This property also allows a target bit-rate or a target distortion to be achieved exactly, while the baseline 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 file size) requirement without the need to decompress and recompress the existing code-stream. Importantly, the code-stream can also be reorganized in other ways to meet the various requirements of the JPEG 2000 standard, as will be described in Section 1.3.

Another difference is that the inverse quantization of JPEG 2000 explicitly allows for a reconstruction bias from the quantizer midpoint for nonzero indices to accommodate the

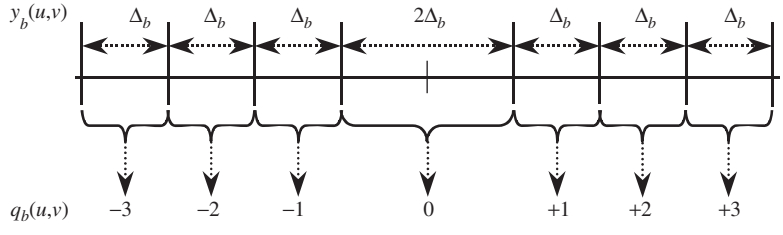


Figure 1.14 Uniform quantizer with a central dead zone with step size Δ_b

skewed probability distribution of the wavelet coefficients. In the JPEG baseline, a simple biased reconstruction strategy has been shown to improve the decoded image PSNR by approximately 0.25 dB (Price and Rabbani, 1999). 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.

1.2.3.1 Quantization at the Encoder

For each subband b, a 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 (Albanesi and Bertoluzza, 1995; Jones, 2007; Jones *et al.*, 1995; O'Rourke and Stevenson, 1995; Watson *et al.*, 1997) 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 Figure 1.14. The quantization operation is an encoder issue and can be implemented in any desired manner. However, it is most efficiently performed according to

$$q_b(u, v) = sign(y_b(u, v)) \left\lfloor \frac{|y_b(u, v)|}{\Delta_b} \right\rfloor. \tag{1.18}$$

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:

$$\Delta_b = 2^{R_b - \varepsilon_b} \left(1 + \frac{\mu_b}{2^{11}}\right), \tag{1.19}$$

where R_b is the number of bits representing the nominal dynamic range of the subband b, which is explained in Section 1.2.2.4. This limits the largest possible step size to approximately 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 is typically one or two.

For the irreversible (9, 7) wavelet transform, 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 (Zheng, Daly, and Lei, 2002) and/or the L_2-norm of each subband in order to align the bit-planes 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,

$$(\varepsilon_b, \mu_b) = (\varepsilon_0 - N_L + n_b, \mu_0), \tag{1.20}$$

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 Equation (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 1.3). This procedure approximately aligns the quantized subband bit-planes according to their proper MSE contribution.

1.2.3.2 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

$$Rq_b(u, v) = \begin{cases} (q_b(u, v) + \gamma)\Delta_b, & \text{if } q_b(u, v) > 0, \\ (q_b(u, v) - \gamma)\Delta_b, & \text{if } q_b(u, v) < 0, \\ 0, & \text{otherwise,} \end{cases} \tag{1.21}$$

where $0 \leq \gamma < 1$ 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 toward zero, which can result in improved reconstruction PSNR (peak SNR) 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 Equation (1.21) is equivalent to $\Delta_b 2^{M_b - N_b}$. The reversible (5, 3) filter bank is treated in the same way (with $\Delta_b = 1$), except when the index is fully decoded to achieve lossless reconstruction, in which case $Rq_b(u, v) = q_b(u, v)$.

1.2.4 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 encoder in JPEG 2000 is motivated by several factors. One is the requirement to create an embedded bit-stream, which is made possible by bit-plane encoding of the quantizer indices. Bit-plane encoding of wavelet coefficients has been used by several well-known embedded wavelet encoders such as EZW (Shapiro, 1993) and SPIHT (Said and Pearlman, 1996). However, these encoders use coding models that exploit the correlation between subbands to improve coding efficiency. This adversely impacts error resilience and severely limits the flexibility of an encoder 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 (Taubman and Marcellin, 2001), where each subband is partitioned into small rectangular blocks, referred to as *code-blocks*, and each code-block is independently encoded. The nominal dimensions of a code-block 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 code-block cannot exceed 4096; and neither the height nor the width of the code-block can be less than four.

The independent encoding of the code-blocks 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 1.3.6). 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 for by the finer scalability that results from multiple-pass encoding of the code-block bit-planes. By using an efficient rate control strategy that independently optimizes the contribution of each code-block to the final bit-stream (see Section 1.4.2), the JPEG 2000 Part 1 encoder achieves a compression efficiency that is superior to other existing approaches (Taubman *et al.*, 2002).

Figure 1.15 shows a schematic of the multiple bit-planes 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 bit-plane encoding, a quantized wavelet coefficient is called *insignificant* if the quantizer index is still zero (e.g. the example coefficient in Figure 1.15 is still insignificant after encoding its first two MSBs). Once the first nonzero 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. Because 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 bit-planes, leading to a very low information content for those bit-planes. JPEG 2000 uses an efficient coding method for exploiting the redundancy of the bit-planes known as context-based adaptive binary arithmetic coding.

1.2.4.1 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.

An adaptive binary arithmetic encoder 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 code-stream with a length at most two bits greater than the combined ideal code-lengths of the input symbols (Pennebaker *et al.*, 1988). Adaptivity

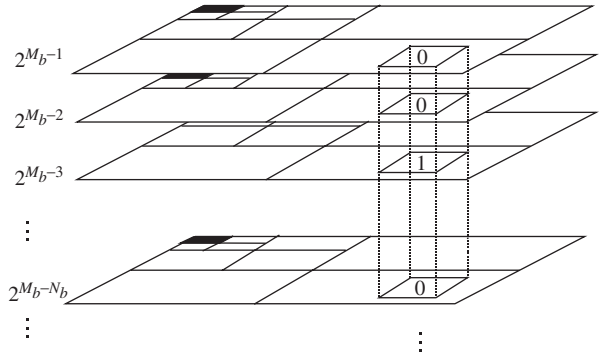


Figure 1.15 Bit-plane coding of quantized wavelet coefficients

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 because only individual symbol probabilities are needed at each subinterval 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 code-block of quantized wavelet coefficient bit-planes.

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 code-length of the symbols encoded (Pennebaker and Mitchell, 1993). 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 (Pennebaker *et al.*, 1988). Later, a modified version of the Q-coder, known as the QM-coder, was chosen as the entropy encoder for the JBIG standard and the extended JPEG mode (Pennebaker and Mitchell, 1993). However, IPR issues have hindered the use of the QM-coder in the JPEG standard. As a result, 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 (ISO/IEC, 2000). The companies that own the IPR on the MQ-coder have made it available on a license-fee-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 chapter and the reader is referred to (Slattery and Mitchell, 1998) and the MQ-coder flowcharts in the standard document (ISO/IEC International Standard 15444-1, ITU Recommendation T.800). We mention in passing that the specific realization of the 'bit stuffing' procedure in the MQ-coder (which costs approximately 0.5% in coding efficiency) creates a redundancy such that any two consecutive bytes of encoded data are always forced to lie in the range of hexadecimal '0000' through 'FF8F' (Taubman and Marcellin, 2001). This leaves the range of 'FF 90' through 'FFFF' unattainable by encoded 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 encoded bits corresponding to that coefficient, as well as by 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 as well as the significance information of its immediate eight neighbors as determined from the previous bit-plane and the current bit-plane, based on encoded 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 0–13 correspond to start-up states (also referred to as *fast-attack*) and are used for rapid convergence to a stable probability estimate. States 14–45 correspond to steady-state probability estimates, and once this range of states has been entered from a start-up state, it can never be left by the finite-state machine. There is also an additional nonadaptive 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.

1.2.4.2 Bit-Plane Coding Passes

The quantized coefficients in a code-block are bit-plane-encoded independently from all other code-blocks when creating an embedded bit-stream. Instead of encoding the entire bit-plane in one coding pass, each bit-plane is encoded in three sub-bit-plane 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, the large number of potential truncation points facilitates the implementation of an optimal rate control strategy where a target bit-rate is achieved by including those coding passes that minimize the total distortion.

Referring to Figure 1.16, consider the encoding of a single bit-plane from a code-block 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 bit-plane 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 significantly different. In particular, if the encoded bit-stream is truncated at any intermediate point during the encoding of the bit-plane, the path ABC would have less distortion for the same rate, and hence would possess a superior embedding property. In optimal embedding, the data with the highest distortion

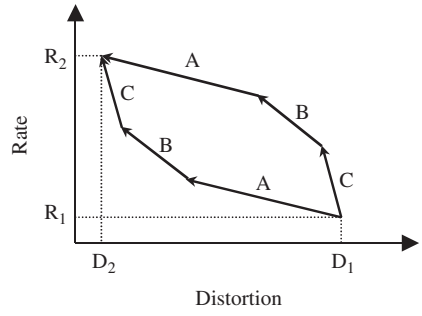


Figure 1.16 R-D path for optimal embedding

[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.

reduction per average bit of compressed representation should be encoded first (Li and Lei, 1999).

For a coefficient that is still insignificant, it can be shown that given reasonable assumptions regarding its probability distribution, the distortion reduction per average bit of compressed representation increases with increasing probability of becoming significant, p_s (Li and Lei, 1999; Ordentlich, Weinberger, and Seroussi, 1998). 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 encoder instead divides the bit-plane data into three groups and encodes each group during a fractional bit-plane pass. Each coefficient in a block is assigned a binary state variable called its *significance state*, which is initialized to zero (insignificant) at the start of the encoding. The significance state changes from zero to one (significant) when the first nonzero 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 Figure 1.17. 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 bit-plane. Finally, during the *cleanup* pass, all the remaining coefficients in the bit-plane, which have the lowest probability of becoming significant, are encoded. The order in which the coefficients in each pass are visited is data dependent and follows a deterministic stripe-scan order with a height of four pixels, as shown in Figure 1.18. This stripe-based scan has been shown to facilitate software and hardware implementations (Marcellin *et al.*, 1999). In the following, each coding pass is described in more detail.

Significance Propagation Pass

During this pass, the insignificant coefficients that have the highest probability of becoming significant in the current bit-plane are encoded. The data are scanned in the stripe order shown in Figure 1.18. Every sample that is currently insignificant but has at least one significant immediate neighbor, based on encoded information up to that point, is encoded. As soon as a coefficient is encoded, its significance state is updated so that it can effect the inclusion of subsequent coefficients in that coding pass. The significance state of the

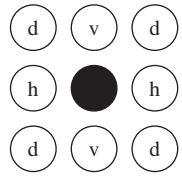


Figure 1.17 Neighboring pixels used in context selection

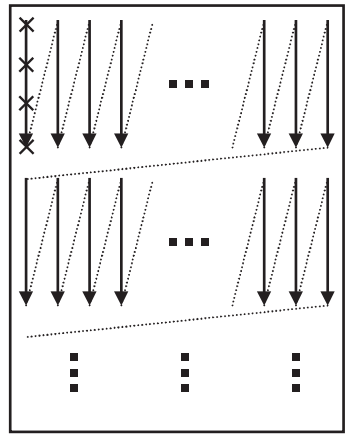


Figure 1.18 Scan order within a code-block

coefficient is arithmetic encoded 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. Because the code-blocks are encoded independently, if a sample is located at the code-block boundary, only its immediate neighbors that belong to the current code-block are considered and the significance states of the missing neighbors are assumed to be zero. Finally, if a coefficient becomes significant in this coding pass, 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.

Refinement Pass

During this pass, the magnitude bit of a coefficient that has already become significant in a previous bit-plane 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 bit-plane (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 encoded with a single context regardless of the neighboring values.

Cleanup Pass

All the remaining coefficients in the code-block are encoded during the cleanup pass. Generally, the coefficients that are encoded 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

[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.

mode, is used to aggregate the coefficients that have the highest probability of remaining insignificant. More specifically, a run mode is entered if all of 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 of 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 bit-plane. 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. Because the probabilities of these additional two bits are 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 is primarily used to improve the throughput of the arithmetic encoder 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 encoded according to the procedure employed for the significance propagation pass.

For each code-block, the MSB planes that are entirely zero are skipped, and the number of such planes is signaled in the bit-stream. Because the significance state of all the coefficients in the first nonzero MSB is zero, only the cleanup pass is applied to the first nonzero bit-plane. Subsequent bit-planes employ all three coding passes (significance propagation, magnitude refinement, and cleanup).

1.2.4.3 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, contexts 9–13 are used for sign coding, contexts 14–16 are used during the refinement pass, and an additional context is used for run coding during the cleanup pass. Each code-block employs its own MQ-coder to generate an arithmetic code-stream for the entire code-block. In the default mode, the coding contexts for each code-block 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 code-block in the final code-stream. 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 0.030053 and 0.063012, respectively.

In order to facilitate the parallel encoding or decoding of the sub-bit-plane passes of a single code-block, it is necessary to decouple the arithmetic encoding of the sub-bit-plane passes from one another. Hence, JPEG 2000 allows for the termination of the arithmetic-encoded bit-stream as well as the reinitialization of the context probabilities at each coding pass boundary. If any of these two options is flagged in the code-stream, 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 code-block, the significances of the samples in future stripes within that code-block are ignored. Because 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 encoder termination, reinitialization 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 *selective arithmetic coding bypass* mode, where the arithmetic encoder is entirely bypassed in certain coding passes. It is common to refer to this as the 'lazy' coding mode. More specifically, after the encoding of the fourth most significant bit-plane of a code-block, the arithmetic encoder is bypassed during the encoding of the first and second sub-bit-plane coding passes (i.e. significance propagation and refinement) of subsequent bit-planes. Instead, their content is included in the code-stream as raw data. In order to implement this mode, it is necessary to terminate the arithmetic encoder 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 reinitialize the MQ-coder context models. The lazy mode can also be combined with the parallel mode in the *lazy–parallel* mode. The impact of the lazy, parallel, and lazy–parallel modes on the coding efficiency is studied in Section 1.5.1.5.

1.2.4.4 Tier-1 and Tier-2 Coding

The arithmetic coding of the bit-plane data is referred to as *tier-1* coding. Figure 1.19 illustrates a simple example of the compressed data generated at the end of tier-1 encoding.

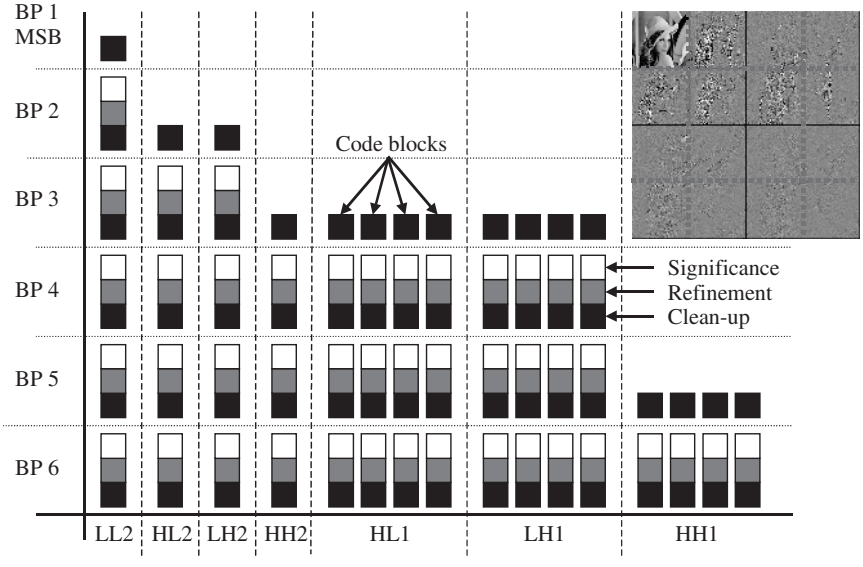


Figure 1.19 Example of compressed data associated with various sub-bit-plane coding passes

The example image (whose wavelet-transformed subbands are shown at the top right of Figure 1.19) is of size 256×256 with two levels of decomposition, and the code-block size is 64×64. Each square box in the figure represents the compressed data associated with a single coding pass of a single code-block. Because the code-blocks 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-bit-plane coding passes for a given code-block must appear in a causal order starting from the most significant bit-plane. The compressed sub-bit-plane coding passes can be aggregated into larger units called *packets*. This process of packetization along with its supporting syntax, as will be explained in Section 1.3, is often referred to as *tier-2* coding.

1.3 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 code-blocks that are discussed in Section 1.2. 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 code-blocks). Tiling provides access to the image data over large spatial regions, while the independent coding of the code-blocks provides access to smaller units. Code-blocks 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 code-blocks from all subbands at a particular resolution level.

In addition to these structures, JPEG 2000 organizes the compressed data from the code-blocks into units known as *packets* and *layers* during the tier-2 coding step. For each precinct, the compressed data for the code-blocks is first organized into one or more packets. A packet is simply a continuous segment in the compressed code-stream that consists of a number of bit-plane coding passes for each code-block in the precinct. The number of coding passes can vary from code-block to code-block (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 sub-sampled) is straightforward, but tedious, and it is not necessary for understanding the basic concepts. Section B.4 of the JPEG 2000 Part 1 standard (ISO/IEC International Standard 15444-1, ITU Recommendation T.800) provides a detailed description and examples for the more general case.

1.3.1 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*-1, *Ysiz*−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*), which specify the upper left corner of the image, and extends to (*Xsize*−1, *Ysiz*−1), as shown in Figure 1.20.

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 (ISO/IEC International Standard 15444-1, ITU Recommendation T.800), 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 1.5.1.6, proper use of the canvas coordinate system improves the performance of the JPEG 2000 encoder in the case of multiple compression cycles when the image is being cropped between compression cycles.

1.3.2 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 ($0 \leq r \leq N_L$) is represented by the subband $(N_L - r)$LL. Recall from Section 1.2.2.2 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

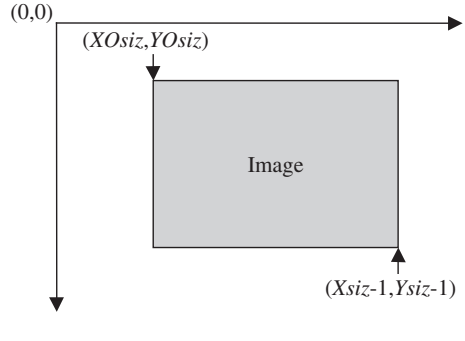


Figure 1.20 The reference grid

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

the image area at resolution level r ($0 \leq r \leq N_L$), the upper left-hand corner is (xr_0, yr_0) and the lower right-hand corner is $(xr_1 - 1, yr_1 - 1)$, where

$$xr_0 = \left\lceil \frac{XOsiz}{2^{N_L - r}} \right\rceil, yr_0 = \left\lceil \frac{YOsiz}{2^{N_L - r}} \right\rceil, xr_1 = \left\lceil \frac{Xsiz}{2^{N_L - r}} \right\rceil \text{ and } yr_1 = \left\lceil \frac{Ysiz}{2^{N_L - r}} \right\rceil, \quad (1.22)$$

and $\lfloor w \rfloor$ 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

$$(xb_0, yb_0) = \begin{cases} \left(\left\lceil \frac{XOsiz - 2^{n_b - 1}}{2^{n_b}} \right\rceil, \left\lceil \frac{YOsiz}{2^{n_b}} \right\rceil \right) & \text{for } n_b \text{ HL band,} \\ \left(\left\lceil \frac{XOsiz}{2^{n_b}} \right\rceil, \left\lceil \frac{YOsiz - 2^{n_b - 1}}{2^{n_b}} \right\rceil \right) & \text{for } n_b \text{ LH band,} \\ \left(\left\lceil \frac{XOsiz - 2^{n_b - 1}}{2^{n_b}} \right\rceil, \left\lceil \frac{YOsiz - 2^{n_b - 1}}{2^{n_b}} \right\rceil \right) & \text{for } n_b \text{ HH band.} \end{cases} \quad (1.23)$$

The coordinates (xb_1, yb_1) can be obtained from Equation (1.23) by substituting *XOsiz* by *Xsiz* and *YOsiz* by *Ysiz*. The extent of subband b is from (xb_0, yb_0) to $(xb_1 - 1, yb_1 - 1)$. These concepts are best illustrated by a simple example. Consider a three-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, which belongs to resolution one, extends from (1, 2) to (96, 65).

1.3.3 Precinct and Code-Block Partitioning

Precincts represent a coarser partition of the wavelet coefficients than code-blocks, and they provide an efficient way to access spatial regions of an image. A precinct is defined as a group of code-blocks from all of the subbands at a specific resolution, such that the group nominally corresponds to the same spatial region in the original image. Figure 1.21 shows an example of precinct and code-block partitions for a three-level decomposition of a 768 × 512 image (single tile-component). In this example, there are six precincts at each resolution level and each precinct corresponds to a 256 × 256 region in the original image.

In Figure 1.21, the highlighted precincts in resolutions 0–3 correspond roughly to the same 256 × 256 region in the original image.[8] For the subbands at resolution 1, i.e. subbands 3HL, 3LH, and 3HH, a 32 × 32 region nominally corresponds to a 256 × 256 region in the original image. Hence, the precinct size in the subbands 3HL, 3LH, and 3HH is 32 × 32. However, in JPEG 2000, the size of the precinct partition is actually defined at the resolution level, instead of the subband level. With this convention, the precinct size at resolution 1 is 64 × 64, which nominally corresponds to a 256 × 256 region in the original image. Note that the precinct size in the subbands at resolution 1 is half (32 × 32) that of the precinct size in the image at resolution 1 (64 × 64). In general,

[8] Here we have neglected expansion of the region due to the support of the wavelet filters.

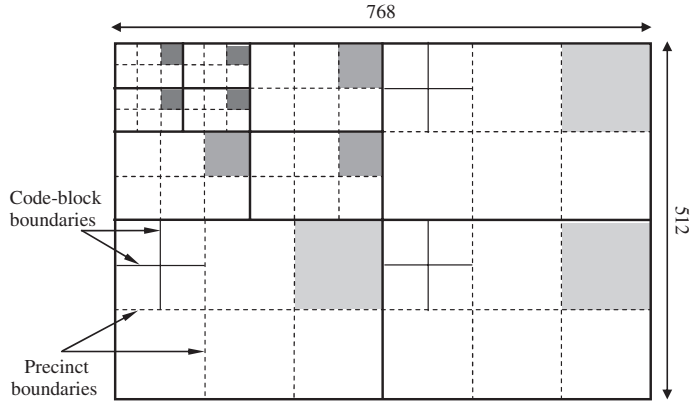


Figure 1.21 Examples of precincts and code-blocks

the precinct partition at a resolution $r\,(r > 0)$ induces an effective precinct partitioning of the subbands at *the same resolution level*. Except for resolution 0, the size of the induced precinct partition in the subbands is half (in each dimension) of the precinct size at the corresponding resolution. For resolution 0, the size of the induced precinct partition in the subband matches the precinct size at image resolution 0 because the image resolution 0 is the same as subband N_LLL. For the example in Figure 1.21, the precinct sizes at resolutions 0–3 are 32×32, 64×64, 128×128, and 256×256, respectively, and the induced subband precinct sizes for resolutions 0–3 are 32×32, 32×32, 64×64, and 128×128, respectively.

While the precinct size can vary from resolution to resolution, it is always restricted to be a power of two. Recall that each subband is also divided into rectangular code-blocks with dimensions that are a power of two. The precinct and code-block partitions are both anchored at (0, 0). Each precinct boundary coincides with a code-block boundary, but the reverse is not necessarily true because a precinct may consist of multiple code-blocks. The induced subband precinct size also imposes some constraints on the code-block size at a given resolution level. Code-blocks from all resolution levels are constrained to have the same size, except when constrained by the induced subband precinct size. For example, in Figure 1.21, the nominal code-block size is chosen to be 64×64, but the induced subband precinct size for subbands 3LL, 3HL, 3LH, and 3HH (i.e. the subbands that correspond to resolutions 0 and 1) is 32×32. This restricts the code-block size for resolutions 0 and 1 to be 32×32.

The precinct size can be chosen so that an entire subband belongs to a single precinct or the induced precinct size matches the code-block size. In the former case, the amount of overhead due to the introduction of precincts is very small, but the spatial accessibility is very poor. In the latter case, the amount of overhead is high due to the large number of code-blocks, but spatial accessibility is very precise.

1.3.4 Layers and Packets

The compressed bit-stream for each code-block is distributed across one or more layers in the code-stream. All of the code-blocks from all subbands and components of a tile

contribute compressed data to each layer. For each code-block, a number of consecutive coding passes (possibly including zero) are included in a layer. Each layer represents a quality increment. The number of coding passes included in a specific layer can vary from one code-block to another and is typically determined by the encoder as a result of postcompression rate-distortion optimization, as will be explained in Section 1.4.2. This feature offers great flexibility in ordering the code-stream. It also enables spatially adaptive quantization. Recall that all the code-blocks in a subband must use the same quantizer step size. However, the layers can be formed in such a manner that certain code-blocks, which are deemed perceptually more significant, contribute a greater number of coding passes to a given layer. As discussed in Section 1.2.3, this reduces the effective quantizer step size for those code-blocks by a power of two compared to other code-blocks 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 code-stream. 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 code-blocks appear in the raster order. Figure 1.21 shows an example of code-blocks belonging to a precinct. The numbering of the code-blocks represents the order in which the encoded data from the code-blocks will appear in a packet.

1.3.5 Packet Header

A packet is the fundamental building block in a JPEG 2000 code-stream. Each packet starts with a *packet header*. The packet header contains information regarding the number of coding passes for each code-block in the packet. It also contains the length of the compressed data for each code-block. The first bit of a packet header indicates whether the packet contains data or is empty. If the packet is nonempty, code-block inclusion information is signaled for each code-block in the packet. This information indicates whether any compressed data from a code-block is included in the packet. If compressed code-block data has already been included in a previous packet, a single bit is used to signal this information. Otherwise, it is signaled with a separate *tag tree* for each subband of the corresponding precinct. The tag tree is a hierarchical data structure that is capable of exploiting spatial redundancy. If code-block data are being included for the first time, the number of most significant bit-planes 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 code-block and the length of the corresponding compressed data are signaled.

1.3.5.1 Tag Trees

The concept of tag tree encoding, which is a particular type of quadtree structure, is best illustrated by an example. Consider a 2-D array of nonnegative integers consisting of two rows and three columns, as shown at the top of Figure 1.22. The array values may represent the layer number in which a code-block is included for the first time. Alternatively, the array values may represent the number of most significant bit-planes that are entirely zero. The values of the original array are the leaf nodes of the tag tree structure. The node value at the next level in the tag tree is the minimum of the values of the child nodes. Nominally, there are four child nodes, but there may be less than

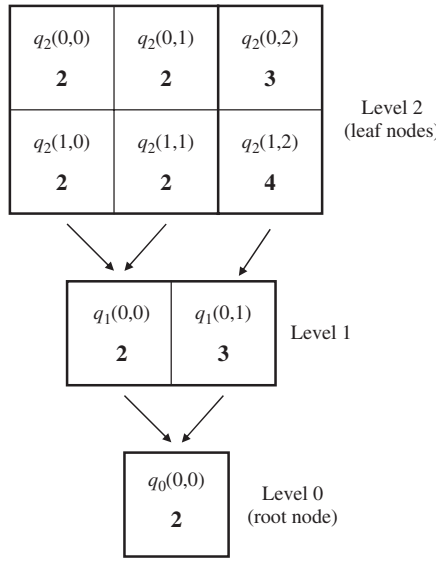


Figure 1.22 Example of a tag tree

four at the right and bottom edges, as shown in Figure 1.22. This process of forming a parent node from the minimum of the child nodes is repeated until only a single root node remains. The value of a particular tag tree node is denoted by $q_z(i, j)$, where z is the level within the quadtree, with leaf nodes at the highest level and the root of the quadtree at level 0, and the row and column indices are represented by i and j, respectively. We will use $q_z(i, j)$ to refer to the tag tree node as well as the actual value of the tag tree node. The exact meaning should be clear from the context.

To encode a specific leaf node from the tag tree, all of the ancestors of the leaf node are encoded, starting with root node at level 0 and proceeding to the higher levels. Any ancestor that has previously been encoded is skipped. As a first example of tag tree encoding, consider the encoding of the 2-D array values of Figure 1.22 in a raster scan order, starting with $q_2(0, 0)$. The ancestor nodes of $q_2(0, 0)$ are $q_1(0, 0)$ and $q_0(0, 0)$. The root node, $q_0(0, 0)$, is encoded first, using a very simple code where a node value of $B \geq 0$ is represented by B zeros followed by a one. Thus, $q_0(0, 0) = 2$ is encoded as '**001**.' The next node to be encoded is $q_1(0, 0)$. For nodes at level 1 or higher, only the difference between the current node and the parent node is encoded using the code described above. This difference is always nonnegative because of the way the tag tree is constructed using the minimum value for the parent node. The difference between $q_1(0, 0)$ and $q_0(0, 0)$ is 0, which is encoded as '**1**.' Similarly, $q_2(0, 0)$ is encoded as '**1**.' For $q_2(0, 1)$, its ancestors, $q_0(0, 0)$ and $q_1(0, 0)$, have already been encoded. Thus, $q_2(0, 1)$ is encoded as '**1**.' To encode $q_2(0, 2)$, it is necessary to encode $q_1(0, 1)$ first. The nodes $q_1(0, 1)$ and $q_2(0, 2)$ are encoded as '**011**.' The remaining nodes, $q_2(1, 0)$, $q_2(1, 1)$, and $q_2(1, 2)$, are encoded as '**1101**.' Thus the entire array is encoded as '**0011110111101**.'

As a second example of tag tree encoding, suppose that each entry in the 2-D array represents the number of zero MSB bit-planes for each code-block belonging to a

particular subband and precinct. As described previously, this information is included in the packet header immediately after the first time a code-block makes a contribution to a layer. As a result, the order of tag tree encoding may not follow a raster scan order. As an example, suppose that code-blocks corresponding to $q_2(0, 2)$ and $q_2(1, 2)$ are included for the first time in layer 0 and the remaining code-blocks are included for the first time in layer *1*. Then, the order in which the leaf nodes get encoded is $q_2(0, 2)$, $q_2(1, 2)$, $q_2(0, 0)$, $q_2(0, 1)$, $q_2(1, 0)$, and $q_2(1, 1)$. The encoding order including the ancestor nodes is $q_0(0, 0)$, $q_1(0, 1)$, $q_2(0, 2)$, $q_2(1, 2)$, $q_1(0, 0)$, $q_2(0, 0)$, $q_2(0, 1)$, $q_2(1, 0)$, and $q_2(1, 1)$. Following the method described in the previous paragraph, the tag tree encoding is '**0010110111111**.' This encoding has exactly 13 bits as before. Thus, the number of bits produced by a tag tree encoding does not depend on the order in which the leaf nodes are encoded.

The method that was just described is suitable for encoding the tag tree containing zero MSB bit-planes information. However, it is not suitable for encoding the code-block inclusion tag tree. This is because it is only necessary to indicate whether a code-block contributes to the current layer if it has not contributed to any previous layer. It is not necessary to specify the exact layer number in which it gets included for the first time. Following the encoding method described above would mean that inclusion information for all the layers is aggregated upfront, which leads to a suboptimal embedding. To rectify this shortcoming, the tag tree encoding method can be modified to make it hierarchical.

To illustrate hierarchical encoding of a tag tree, assume that the values of the 2-D array in Figure 1.22 represent the layers in which the code-blocks are included for the first time. Instead of coding the exact layer number in which the code-block is included for the first time, the only information included for packet w (corresponding to layer w) is whether the code-block is included in layer w for the first time. This is accomplished by hierarchical encoding of the tag tree. It is necessary to keep track of two more variables for each node in the tag tree, the current value cv and the state S. A state value of 1 indicates that the exact value for a tag tree node can be deduced from the information already encoded. The tag tree decoder mimics the encoding steps.

At the beginning of the tag tree encoding procedure, the current value and state for each node is initialized to zero. The interpretation of the current value for a tag tree node is as follows. If the state S is 0, $q_z(i, j) \geq cv_z(i, j)$. Let the total number of layers be W and let the leaf nodes in the tag tree be at level Z. A leaf node $q_Z(i, j)$ has ancestors $q_z(i_z, j_z)$, $0 \leq z < Z$, where $i_z = \lfloor i/2^{(Z-z)} \rfloor$ and $j_z = \lfloor j/2^{(Z-z)} \rfloor$.

Encode:

for each layer w, $w = 0 : (W - 1)$
    for each leaf node $q_Z(i, j)$
        for each $q_z(i_z, j_z)$, $0 \leq z \leq Z$
            if $(S_z(i_z, j_z) == 0)$
                if $(z > 0)$ and $cv_z(i_z, j_z) < cv_{(z-1)}(i_{z-1}, j_{z-1})$
                    $cv_z(i_z, j_z) = cv_{(z-1)}(i_{z-1}, j_{z-1}$ // Due to tag tree construction
                if $(cv_z(i_z, j_z) \leq w)$
                    if $(q_z(i_z, j_z) \leq w)$ Emit '**1**,' set $S_z(i_z, j_z) = 1$
                    else emit '**0**,' increment $cv_z(i_z, j_z)$ by 1.

Decode:

for each layer w, $w = 0 : (W - 1)$
  for each leaf node $q_Z(i, j)$
    for each $q_z(i_z, j_z)$, $0 \le z \le Z$
      if $(S_z(i_z, j_z) = 0)$
        if $(z > 0)$ and $cv_z(i_z, j_z) < cv_{(z-1)}(i_{z-1}, j_{z-1})$
          $cv_z(i_z, j_z) = cv_{(z-1)}(i_{z-1}, j_{z-1})$ // Due to tag tree construction
        if $(cv_z(i_z, j_z) \le w)$
          if next bit is '**1**,' set $S_z(i_z, j_z) = 1$ and
          $q_z(i_z, j_z) = cv_z(i_z, j_z)$
          else increment $cv_z(i_z, j_z)$ by 1.

Referring to the tag tree of Figure 1.22, we will now illustrate the hierarchical encoding process. All the state and current value variables are initialized to 0. For $w = 0$, $cv_0(0, 0) \le w$ and $q_0(0, 0) > w$. Hence, a '**0**' bit is emitted and $cv_0(0, 0)$ is incremented to 1. From the interpretation of cv, this implies that $q_0(0, 0) \ge cv_0(0, 0) = 1$. As a result of the manner in which the tag tree is constructed, $q_z(i_z, j_z) \ge 1, 0 \le z < Z$. Thus, all the remaining states are incremented by 1 and no further bits are emitted for layer 0. Then for $w = 1$, $cv_0(0, 0) \le w$ and $q_0(0, 0) > w$. Hence a '**0**' bit is emitted and $cv_0(0, 0)$ is incremented to 2. As before, all the remaining states are incremented to 2 and no further bits are emitted for layer 1. For $w = 2$, $cv_0(0, 0) \le w$ and $q_0(0, 0) \le w$. Hence a '**1**' is emitted and $S_0(0, 0)$ is set to 1. Now proceeding to $q_1(0, 0)$, $q_2(0, 0)$, and $q_2(0, 1)$, a '**1**' is emitted for each, and the corresponding states are set to 1. Proceeding to $q_1(0, 1)$, $cv_1(0, 1) \le w$ and $q_1(0, 1) > w$. Hence a '**0**' bit is emitted and $cv_1(0, 1)$ is incremented to 3. Then $cv_2(0, 1)$ is set to 3 and no further bits are emitted. For $q_2(1, 0)$ and $q_2(1, 1)$, a '**1**' is emitted for each and their states are set to 1. Proceeding to $q_2(1, 2)$, $cv_2(1, 2)$ is incremented to 3 to equal $cv_1(0, 1)$ and no bits are emitted. For $w = 3$, a '**1**' is emitted for $q_1(0, 1)$ as well as $q_2(0, 2)$, and their states are set to 1. Proceeding to $q_2(1, 2)$, a '**0**' bit is emitted and $cv_2(1, 2)$ is incremented to 4. Finally, for $w = 4$, a '**1**' is emitted for $q_2(1, 2)$ and its state is set to 1. Thus, the encoded tag tree bit-stream is '**0 0 1111011 110 1**.' The extra spaces indicate layer separation.

Note that the encoded tag tree also consists of 13 bits for this hierarchical encoding method. Furthermore, the number of 1's and the number of 0's are also exactly the same as in the case of nonhierarchical encoding. Because of its general applicability, the hierarchical encoding method is used for encoding all tag trees in JPEG 2000.

1.3.6 Progression Order

The arithmetic encoding of the bit-planes 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 code-stream, it is necessary to decode the packet header information, but it is not necessary to perform arithmetic decoding. This allows the code-stream to be reorganized with minimal computational complexity.

The order in which packets appear in the code-stream is called the progression order and is controlled by specific markers. Regardless of the ordering, it is necessary that coding passes for each code-block appear in the code-stream 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 Figure 1.21. 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 numbers of resolution levels. In that case, the LL subbands of different components are aligned.

1.3.6.1 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 noninterleaved format) provides similar functionality for a single resolution image.

1.3.6.2 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 the 'hierarchical progressive' mode of extended JPEG where each resolution is further encoded with the 'sequential progressive' mode (component noninterleaved format).

1.3.6.3 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. For JPEG systems, the 'resolution–position–component' order for a single layer can be obtained using the hierarchical progressive mode of extended JPEG with each resolution encoded with baseline JPEG (component interleaved format).

1.3.6.4 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 the JPEG baseline where the image is sequentially compressed by compressing the component interleaved 8×8 blocks in a raster order fashion.

1.3.6.5 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 the 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 code-stream, can become complicated if different components have different precinct sizes, as explained in the standard document (ISO/IEC International Standard 15444-1, ITU Recommendation T.800). 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.

Figures 1.23 to 1.25 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 at resolutions 0, 1, 2, and 3 are 32×32, 64×64, 128×128, and 256×256, respectively. The code-block size is 64×64, except for resolutions 0 and 1, where the code-block size is constrained to the subband precinct size of 32×32. Thus, there are four resolutions, six precincts per resolution, and two layers, resulting in 48 packets. Figure 1.23 shows the LRCP progression order (Section 1.3.6.1). 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. Figure 1.24 illustrates the RLCP ordering (Section 1.3.6.2). The figure shows images reconstructed after decoding resolutions 0, 1, 2, and 3 (12, 24, 36, and 48 packets), respectively. Figure 1.25 illustrates the PCRL ordering (Section 1.3.6.4). The image has been reconstructed after decoding 32 packets corresponding to the first four precincts.

Figure 1.23 Example of layer progressive bit-stream ordering: (left) 0.125 bpp; (right) 0.50 bpp

Figure 1.24 Example of resolution progressive bit-stream ordering

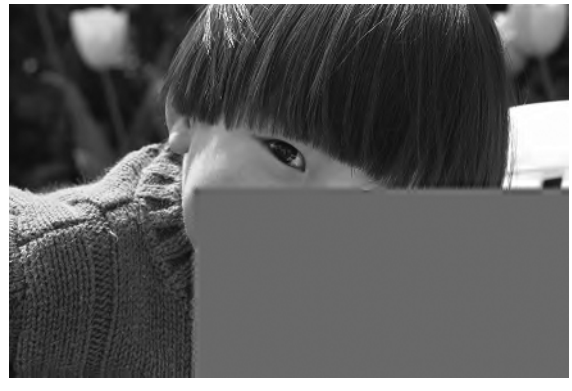


Figure 1.25 Example of spatially progressive bit-stream ordering (four precincts decoded)

It should be noted that because of the prediction step in the hierarchical progressive mode of the 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 interresolution dependency makes it impossible to achieve certain progression orders, e.g. LRCP. Also, rearranging the JPEG compressed data from one progression mode to another generally requires an inverse DCT, e.g. 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. However, the decoding of packet headers can be avoided by inclusion of PLT (or PLM) marker segments, which are described in the next subsection.

1.3.7 Code-Stream Organization and Syntax

A JPEG 2000 code-stream consists of two fundamental types of data: (1) compressed data in the form of packets and (2) syntactical data in the form of markers and marker segments that define the characteristics of the image and delimit the code-stream. Certain markers and marker segments are combined to form headers, and the JPEG 2000 standard defines two kinds of headers, a main header and a tile-part header. At the highest structural level, a JPEG 2000 code-stream consists of a main header, one or more tile parts, and an end of code-stream (EOC) marker. Multiple tile parts are formed, if desired, by breaking the compressed data for a tile at any packet boundary. Each tile part consists of a tile-part header and compressed data for the tile part as a sequence of packets. Tile parts from different tiles can be interleaved in the code-stream. Figure 1.26 shows an example of a JPEG 2000 code-stream consisting of two tiles, where tile 0 has a single tile part and tile 1 has two tile parts.

1.3.7.1 Markers and Marker Segments

A marker is always two bytes, and the first byte is always 0xFF.[9] The second byte denotes the specific marker, with a value in the range 0x01 to 0xFE. The use of the 0xFF byte as a prefix allows the markers to be located easily when a code-stream is parsed. Typically, a marker is followed by a parameter list, and together the marker and the parameter list form a marker segment. The marker occupies the first two bytes of a marker segment. The marker is followed by a two-byte length parameter, which is an unsigned integer in the big-endian format that specifies the length of the marker segment. The length of the marker segment includes the two bytes for the length parameter itself but does not include the two bytes for the marker. The marker segments are always multiples of 8 bits and all multibyte parameter values in a marker segment are big endian. There are six types of marker and marker segments:

- delimiting markers and marker segments: start of code-stream (SOC), start of tile part (SOT), start of data (SOD), and end of code-stream (EOC);
- fixed information marker segments: image and tile size (SIZ);

[9] The prefix 0x indicates that the number following the prefix is in hexadecimal notation.

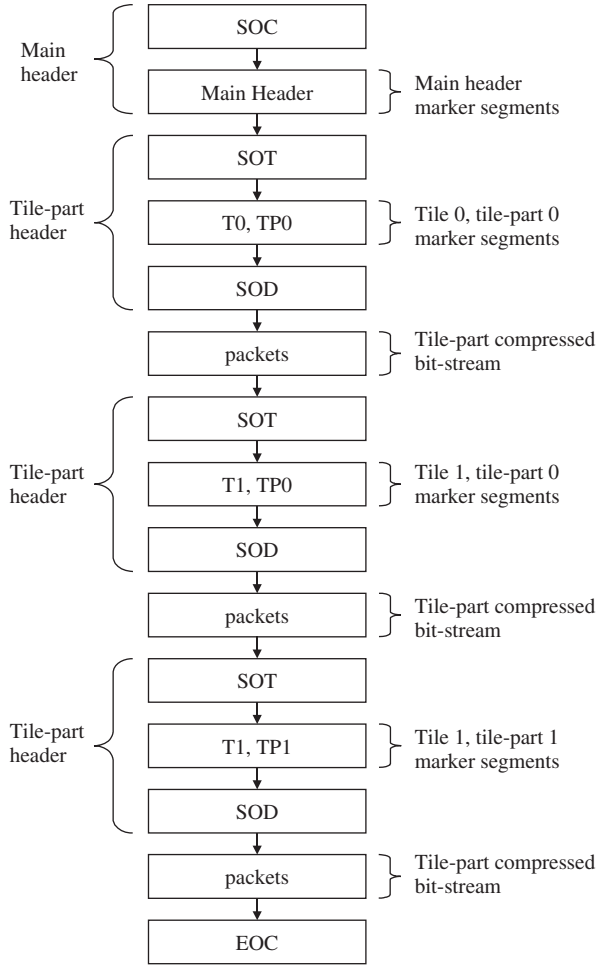


Figure 1.26 JPEG 2000 code-stream organization

- functional marker segments: coding style default (COD), coding style component (COC), region of interest (RGN), quantization default (QCD), quantization component (QCC), and progression order change (POC);
- pointer marker segments: tile-part lengths (TLM), packet length, main header (PLM), packet length, tile-part header (PLT), packed packet headers, main header (PPM), and packed packet headers, tile-part header (PPT);
- in bit-stream markers and marker segments: start of packet (SOP) and end of packet header (EPH);
- informational marker segments: component registration (CRG) and comment (COM).

The delimiting and fixed information markers and marker segments must be present in specific locations in all JPEG 2000 compliant code-streams. The code-stream must begin with the SOC marker, followed by the SIZ marker in the main header. Each tile-part

header must contain the SOT marker as its first marker segment. The SOD marker must be present after each tile-part header. Finally, a valid JPEG 2000 code-stream must end with the EOC marker.

The SIZ marker segment as well as the functional marker segments contain information concerning the image and the code-stream. The SIZ marker segment contains information regarding the image and tile sizes, and their positioning with respect to the reference grid, as well as component subsampling factors and bit-depths. The COD marker segment contains information related to coding parameters such as precinct and code-block sizes, entropy-coding mode, number of layers, progression order, etc. The COC marker segment contains the same type of information as the COD marker, but it applies to a specific component. The QCD and QCC marker segments contain quantizer step size information for all components and for a specific component, respectively. The RGN marker segment can be used to specify a region of interest and the POC marker is used to signify a change in the progression order.

The pointer marker segments provide length information or pointers into the code-stream. They are useful for providing random access to certain parts of the code-stream without having to parse the whole code-stream. The SOP marker segment may appear before each packet and the EPH marker may appear immediately after a packet header. The SOP and EPH markers are useful for error detection and synchronization. Finally, CRG and COM are informational marker segments. They are not necessary for correct decoding of the JPEG 2000 code-stream. The CRG marker specifies the specific registration of different components, which may be helpful for rendering purposes. The COM marker segment allows storage of unstructured comment information.

1.3.7.2 Allowable Marker Segments and Scope

The markers and marker segments that can be present in the main header are SOC, SIZ, COD, COC, QCD, QCC, RGN, POC, PPM, TLM, PLM, CRG, and COM. The SOC, SIZ, COD, and QCD are required in the main header, while the rest of the markers are optional. The SOC and SIZ markers must be the first two markers in the main header. The rest of the marker segments can appear in an arbitrary order within the main header.

For each tile-part header, SOT must be the first marker segment and SOD must be the last marker. Each tile-part header may optionally contain POC, PPT, PLT, or COM marker segments. If the tile-part header is the first for a given tile, it can optionally contain COD, COC, QCD, QCC, or RGN marker segments.

The scope of a marker segment depends on its type. The scope of a marker segment in the main header extends to the whole image, while the scope of a marker segment in a tile-part header extends only to the corresponding tile part. If a marker segment is component specific, its scope is restricted accordingly. For example, if a COC marker segment is present in a tile-part header, its scope extends to the specific component in that tile part. When there are multiple marker segments with overlapping scopes, the following precedence rules apply. Within a tile-part header, a component-specific marker segment takes precedence over a general tile-part header marker segment for the specific component. Similarly, within the main header, a component-specific segment takes precedence over a general main header marker segment for that specific component. Furthermore, a marker segment appearing in a tile-part header takes precedence over a marker segment appearing in the main header for the specific tile. As an example, consider

a code-stream with the main header containing COD and COC marker segments and a tile-part header also containing COD and COC marker segments. For this example, the marker segments in decreasing order of precedence are: tile-part COC, tile-part COD, main COC, and main COD.

1.4 JPEG 2000 Rate Control

Rate control refers to the process of allocating bits to an image and is strictly an encoder issue. In constant bit-rate (CBR) systems, the goal is to achieve a target file size with the highest possible image quality. In variable bit-rate (VBR) systems, the goal is to achieve a target image quality with the smallest possible file size. Depending upon the application, either CBR or VBR may be preferable. The metric that is used to assess image quality is typically the mean squared error between the original and reconstructed image. However, the use of MSE is primarily motivated by mathematical convenience, and it is well known that MSE does not always correlate well with perceived quality. As a result, visually weighted MSE or more sophisticated visual distortion metrics can also be used (Liu, Karam, and Watson, 2006: Marziliano *et al.*, 2004).

The structure of the JPEG 2000 encoding process provides new opportunities for rate control that are not available in conventional JPEG encoding. In the existing JPEG standard, the user only has control over the quantization and Huffman tables, and there is no easy mechanism for compressing an image to a desired bit-rate. A typical JPEG rate control algorithm for CBR encoding starts with a basic q-table and iteratively modifies the q-table elements (e.g. by a scale factor) until the desired file size (bit-rate) is achieved. In contrast, the embedded block encoding scheme of the JPEG 2000 and its flexible code-stream syntax allow for the noniterative generation of an R-D optimized code-stream for a given file size or a given image quality level. Each JPEG 2000 encoder can perform its own optimization (based on the distortion metric used) to generate a code-stream that conforms to the standardized syntax. In the following, a brief discussion of several possible approaches to JPEG 2000 rate control is provided.

1.4.1 Rate Control Using an Explicit q-Table

One approach to rate control is to use an explicit q-table (or q-tables when encoding color images) in a manner similar to JPEG, where a quantizer step size is specified for each subband and signaled explicitly as header information. However, for CBR encoding, this approach suffers from the same drawback as JPEG in that the q-table needs to be modified (e.g. scaled or otherwise adjusted) iteratively to achieve the desired bit-rate. Although there is no need to perform the wavelet transform at each iteration, the quantization and encoding processes still need to be performed repeatedly.

The use of explicit q-tables is more applicable to VBR systems, where the goal is constant image quality and the bit-rate will fluctuate with the image content. Because human observers are the ultimate judges of image quality in most applications, it is necessary to consider the properties of the HVS when designing a q-table (Albanesi and Bertoluzza, 1995; Jones, 2007; Jones *et al.*, 1995; O'Rourke and Stevenson, 1995; Watson *et al.*, 1997; Zeng, Daly, and Lei, 2002). The general approach to perceptually based q-table design is to take advantage of the sensitivity variations of the HVS to different spatial

frequencies. The DWT in JPEG 2000 conveniently provides a frequency decomposition of the input image, and the HVS response can be mapped on to the wavelet subbands. Although visually based 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 the viewing conditions are changed. A more efficient approach is to use a computational model of the contrast sensitivity function (CSF) as described in Jones *et al.* (1995). The CSF quantifies the detection threshold of an observer to different spatial frequencies, and the goal in applying the CSF to image compression is to ensure that all compression errors are kept below the detection threshold for specified viewing conditions. If this goal is achieved, the compressed image quality is referred to as *visually lossless* for the specified viewing conditions. The viewing conditions include such parameters as the viewing distance, displayed pixel size, display noise, and light adaptation level. The type of analysis that is needed to determine visually lossless q-tables for JPEG 2000 is described in Jones (2007). It is noted that the use of explicit q-tables can also be combined with the rate-distortion optimization method of the next section if the visually based VBR encoding is subject to a maximum file size constraint.

1.4.2 Rate Control Using the EBCOT Algorithm (PCRD-opt)

In 2000, 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, including achieving a desired distortion with the minimum bit-rate.

In the basic approach, each subband is first quantized using a very fine step size, and the bit-planes of the resulting code-blocks are entropy encoded. This typically generates more coding passes for each code-block than will be eventually included in the final code-stream. This situation is known as *overcoding*, and it represents a computational inefficiency that is necessary to achieve the desired rate control benefits with this method. Heuristics can be used to reduce the amount of overcoding, but they require stable image statistics to perform well. 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 code-block 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 code-block 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 code-block contains a large number of potential truncation points that can occur at the end of each sub-bit-plane pass. The wavelet coefficients $y(u, v)$ contained in a code-block 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 code-block 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

code-block due to the inclusion of some coefficients in the sub-bit-plane at which the truncation occurs. For each sub-bit-plane, the increase in bit-rate and the reduction in distortion resulting from the inclusion of that sub-bit-plane 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 code-blocks can be used. Let the total number of code-blocks for the entire image be P and let the code-blocks in the image be denoted by B_i, $1 \leq i \leq P$. For a given truncation point t in code-block B_i, the associated weighted MSE distortion D_i^t is given by

$$D_i^t = \alpha_b^2 \sum_{u,v} w_i(u, v) \left[y_i(u, v) - y_i^t(u, v) \right]^2, \tag{1.24}$$

where u and v represent the coefficient row and column indices within the code-block 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 for the quantization noise, this distortion is additive across code-blocks. At the given truncation point t, the size of the associated compressed bit-stream (i.e. the rate) for the code-block 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 code-block that minimizes the total distortion D. This is equivalent to finding the optimal bit allocation for all of the code-blocks, R_i^*, $1 \leq i \leq P$, such that

$$D = \sum_i D_i^* \text{ is minimized subject to } \sum_i R_i^* \leq R. \tag{1.25}$$

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

1.4.2.1 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 1.4.1), 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 (ISO/IEC International Standard 15444-1, ITU Recommendation T.800) lists recommended frequency weightings for three different viewing conditions. This approach is known as *fixed visual weighting*.

1.4.2.2 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 code-stream 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 code-stream 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 1.3.6.1. 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 nonboundary 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 1.5.2.2, the impact of the resulting overhead on PSNR is quite small.

1.4.2.3 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 encoder 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 (Li, 1999). This is known as *progressive visual weighting*.

1.5 Performance Comparison of the 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 compare the speed and implementation complexity of different coding options accurately, so we only point out the relative speed/complexity advantages of certain options.

To obtain the reported results, the three test images shown in Figure 1.27, 'Bike,' 'Café,' and 'Woman' of size 2048 (columns) $\times$ 2560 (rows) were chosen from the JPEG

Figure 1.27 Test images (from left to right) Bike, Café, Woman

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

$$\mathrm{PSNR} = 10 \log_{10} \left(\frac{255^2}{\mathrm{MSE}} \right), \tag{1.26}$$

where MSE is the mean squared error between the original image and the reconstructed image. In most cases, the results are presented as 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) (ISO/IEC, 2000). During each simulation, only one 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; five levels of wavelet decomposition; 64×64 code-blocks; and a single layer. The subbands at each resolution were treated as a single precinct. In the 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, 0.25, 0.5, 1.0, and 2.0 bits/pixel (bpp). To achieve a target bit-rate, the compressed code-block bit-streams

were truncated to form a single layer. The truncation points are determined using the EBCOT postcompression R-D optimization procedure as described in Section 1.4.2. 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 PSNRs of different encoding options for fixed bit-rates by using the EBCOT rate control algorithm.

1.5.1 Lossy Results

1.5.1.1 Tile Size

JPEG 2000 allows spatial partitioning of the image into tiles. Each tile is wavelet transformed and encoded 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) without the extra complexity of small precincts. Table 1.4 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 a single-sample overlap DWT (SSO-DWT), which reduces edge artifacts at the tile boundaries.

1.5.1.2 Code-Block Size

The code-blocks 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 code-block cannot exceed 4096. Furthermore, when the induced subband

Table 1.4 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 1.5 Comparison of R-D performance for various code-block 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)

precinct size for a particular subband is less than the code-block size, the code-block size is set equal to the induced subband precinct size.

Table 1.5 compares the effect of varying the code-block 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 code-block size of 64×64 to a code-block size of 32×32. However, code-block 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 regarding the number of coding passes and the length of compressed data for each code-block, so the total size of the header information increases with an increasing number of code-blocks. Another factor is the independent encoding of each code-block that requires the reinitialization of the arithmetic encoding models. As the code-block size becomes smaller, the number of samples required to adapt to the underlying probability models constitutes a greater portion of the total number of samples encoded. In addition, the pixels that lie on the boundary of a code-block have an incomplete context because pixels from neighboring code-blocks cannot be used in forming the coding contexts. As the code-block size decreases, the percentage of boundary pixels with incomplete contexts increases.

It can also be concluded from Table 1.5 that the loss in compression performance with decreasing code-block 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 that are attributable to model mismatch and incomplete contexts add up. In comparison, at low bit-rates, many code-blocks 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, so for these code-blocks, 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 code-blocks.

1.5.1.3 DWT Filters

Part 1 of JPEG 2000 offers a choice of either the (9, 7) or the (5, 3) filter bank for lossy compression. Figure 1.28 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

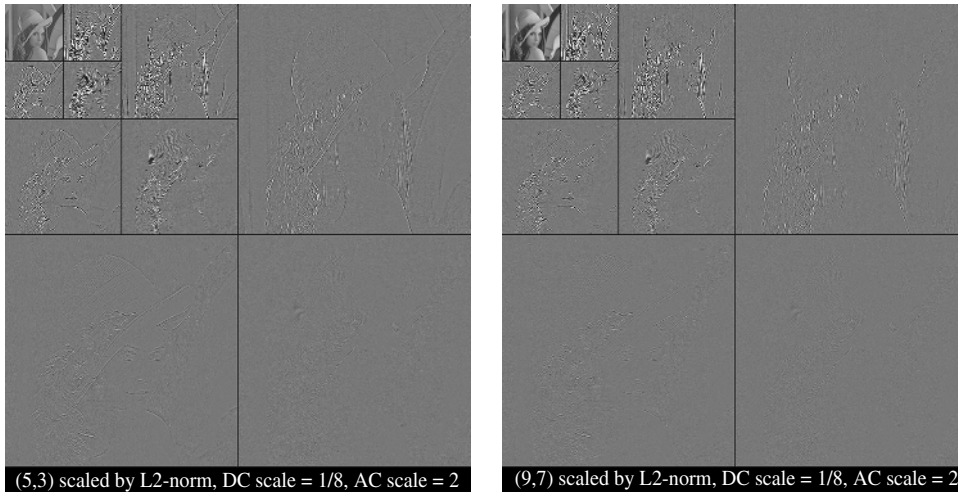


Figure 1.28 Energy compaction comparison between the irreversible (5, 3) and (9, 7) filter banks

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. However, it is worth noting that the (5, 3) filter has much less computational complexity than the (9, 7) filter.

Table 1.6 compares the R-D performance of the two filter banks in the lossy compression mode. 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 is also capable of performing lossless compression. When the target lossy bit-rate equals the lossless bit-rate for a particular image, the (5, 3) filter bank can produce zero MSE (or infinite PSNR) in the lossless mode, whereas the (9, 7) filter bank always produces a nonzero MSE. Lossless compression ratios are typically around 2:1 and thus, for bit-rates in the range of 4.0 bits/pixel or higher (with 8-bit images), lossless compression with the (5, 3) filter bank will outperform lossy compression with the (9, 7)

Table 1.6 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)

filter bank. Specific lossless compression results for the (5, 3) filter bank are presented in Section 1.5.2.

1.5.1.4 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 1.7 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. The use of fewer than five levels results in a loss in coding efficiency, with increasing loss as the number of levels is reduced. The loss is greatest at lower bit-rates and tapers off with increasing bit-rate. We can conclude that a five-level decomposition is adequate in terms of coding efficiency, although it still may be desirable to use more than five levels to provide easy access to lower resolution versions of high-resolution images.

1.5.1.5 Lazy, Parallel, and Lazy–Parallel Modes

As mentioned in Section 1.2.4, JPEG 2000 provides several entropy-coding options that facilitate the parallel processing of the quantized coefficient bit-planes. The collection of these encoding 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 (i.e. selective arithmetic coding bypass) mode, where only the cleanup pass is arithmetic encoded after the fourth most significant bit-plane. Table 1.8 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.

1.5.1.6 Effect of Multiple Compression Cycles

Table 1.9 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

Table 1.7 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)

Table 1.8 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 1.9 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 1.10 R-D performance of multiple compression cycles with cropping with the (9, 7) filter bank

Rate (bits/pixel)	Average PSNR in dB				
		No canvas coordinate system		Canvas coordinate system	
	Reference	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)

implementation with the (9, 7) filter was used in all cases. The postcompression R-D optimization engine is used to achieve the desired bit-rate at each iteration. It can be seen from Table 1.9 that multiple compression cycles cause very little degradation (0.03–0.08 dB) in compression performance when the compression parameters are held constant.

Table 1.10 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 code-block alignment using the canvas coordinate system. In this case, only the wavelet coefficients near the boundary of the image are affected by cropping. From Table 1.10, it can be seen that maintaining the code-block alignment leads to superior compression performance. More performance comparisons can be found in Joshi, Rabbani, and Lepley (2000).

1.5.1.7 JPEG 2000 versus JPEG Baseline

Table 1.11 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.[10] 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 1.11. It can be seen that the use of JPEG 2000 results in approximately 2–4 dB higher PSNR than JPEG baseline depending on the bit-rate.

1.5.2 Lossless Results

1.5.2.1 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 1.12

Table 1.11 R-D performance of JPEG 2000 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

[10] Independent JPEG Group, JPEG Library (Version 6b), available from http://www.ijg.org/ or ftp://ftp.uu.net/graphics/jpeg/ (tar.gz format archive).

Table 1.12 Comparison of lossless bit-rates for color images with and without RCT

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

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 (i.e. 8 bits per component) 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.

1.5.2.2 Lossless Encoder Options

Tables 1.13 to 1.16 summarize the lossless compression performance of Part 1 of the JPEG 2000 standard as a function of tile size, number of decomposition levels, code-block size, and 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

Table 1.13 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 1.14 Comparison of average lossless bit-rates (bits/pixel) for different numbers 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 1.15 Comparison of average lossless bit-rates (bits/pixel) for different code-block sizes

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

Table 1.16 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

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 bit-plane 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 code-block 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 1.17 compares the effect of multiple layers on the lossless coding efficiency. As mentioned in Section 1.4.2.2, 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 (Marziliano *et al.*, 2004), whereas increasing the number of layers from 7 to 50 does not linearly increase the lossless bit-rate because the header information for the increased number of packets is encoded more efficiently. In particular, the percentage of code-blocks that do not contribute to a given packet increases with the number of layers, and the packet header syntax allows this information to be encoded very efficiently using a single bit.

1.5.2.3 Lossless JPEG 2000 versus JPEG-LS

Table 1.18 compares the lossless performance of JPEG 2000 with JPEG-LS (ISO/IEC, 1999). 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

Table 1.17 Comparison of average lossless bit-rates (bits/pixel) for different numbers of layers

1 layer	7 layers	50 layers
4.797	4.809	4.829

Table 1.18 Comparison of average lossless bit-rates (bits/pixel) for JPEG 2000 and JPEG-LS

JPEG 2000	JPEG-LS
4.797	4.633

of imagery (e.g. the 'cmpnd 1' compound document from the JPEG 2000 test set), the JPEG-LS bit-rate is only 60% of that of JPEG 2000 (Marziliano *et al.*, 2004).

1.5.3 Bit-Plane Entropy Coding Results

In this section, we examine the redundancy contained in the various bit-planes 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'). Because 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 bit-planes in different subbands were aligned by their LSBs. Eleven of the resulting bit-planes were encoded, starting with the most significant bit-plane.

One way to characterize the redundancy is to count the number of bytes that are generated by each sub-bit-plane coding pass. The number of bytes generated from each sub-bit-plane coding pass is not readily available unless each coding pass is terminated. However, during postcompression 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 (ISO/IEC, 2000). It is not guaranteed that the near optimal length calculation algorithm will determine the minimum number of bytes needed for unique decoding. 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 this caveat in mind, Table 1.19 contains the number of bytes generated from each sub-bit-plane coding pass. The estimated bytes for each coding pass were summed across all the code-blocks in the image to generate these entries.

During the encoding of the first bit-plane, 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 bit-plane, only these 36 coefficients are refined. Surprisingly, the first refinement bit for 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

Table 1.19 Encoded bytes resulting for sub-bit-plane passes of the Lena image

Bit-plane Number	Significance bytes	Refinement bytes	Cleanup 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

second bit-plane. This, in conjunction with the possibility of overestimating the number of bytes in the cleanup pass of the first bit-plane, leads to the rather strange result that the refinement pass for the second bit-plane requires zero bytes. It is also interesting that the number of bytes needed to encode a given bit-plane is usually greater than the total number of bytes used to encode all of the bit-planes prior to it (except for bit-plane 11).

Figure 1.29 shows images reconstructed from the first nine bit-planes and Table 1.20 provides the corresponding PSNRs. Table 1.20 also shows the percentage of the coefficients that are refined at each bit-plane, the percentage of the coefficients that are found to be significant at each bit-plane, and the percentage of the coefficients that remain insignificant after completion of the encoding of a bit-plane. It is interesting to note that approximately 72% of the coefficients still remain insignificant after encoding the tenth bit-plane.

Figure 1.29 Reconstructed Lena image after decoding bit-planes 1–9 (from left to right and top to bottom)

Table 1.20 Coding statistics resulting from encoding the first eleven wavelet coefficient bit-planes of the 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

Figure 1.30 provides a graphic representation of the data that is contained in Table 1.20 for bit-planes 4, 6, 8, and 11. For each bit-plane, the green pixels denote the location of the wavelet coefficients that become significant during the significance propagation pass for that bit-plane, the red pixels denote the location of those coefficients that turn significant during the cleanup pass, the black pixels denote the location of the coefficients that get refined, and the white pixels denote coefficients that still remain insignificant after the encoding of that bit-plane.

1.6 Additional Features of JPEG 2000 Part 1

1.6.1 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 code-stream. This feature is desirable in progressive transmission in case of early termination of the code-stream.

ROI 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 (Atsumi and Farvardin, 1998). 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 (Marziliano *et al.*, 2004). Prior to entropy coding, the bit-planes of the coefficients belonging to the ROI mask are shifted up (or the background bit-planes

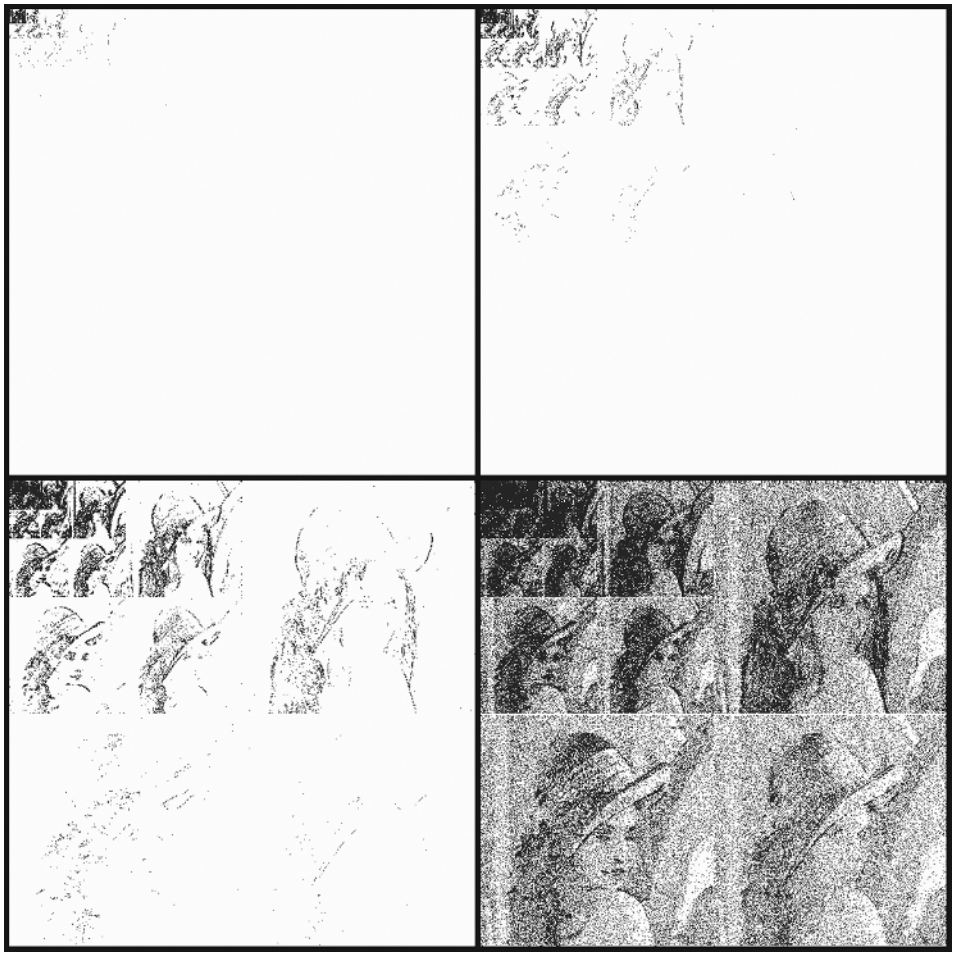


Figure 1.30 From left to right and top to bottom are shown bit-planes 4, 6, 8, and 11 of the JPEG 2000 encoded Lena image as described in Table 1.20. Green pixels denote the location of the wavelet coefficients that become significant during the significance propagation pass in that bit-plane, red pixels denote the coefficients that turn significant during the cleanup pass in that bit-plane, black pixels denote coefficients that are refined, and white pixels denote coefficients that remain insignificant after completion of the three coding passes (see Plate 2)

are shifted down[11] by a desired amount that can vary from 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 code-stream. 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

[11] 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.

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 *Max-shift* method (Christopoulos, Askelof, and Larsson, 2000; Nister and Christopoulos, 1999).

In the Max-shift 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. To ensure that the smallest nonzero ROI coefficient is still larger than the largest background coefficient, s LSBs are added to each wavelet coefficient and all bit-planes of the background coefficients are shifted down by s bits, as shown in Figure 1.31. The presence of ROI is signaled to the decoder by a marker segment and the value of s is included in the code-stream. At the decoder, those wavelet coefficients whose values are more than 2^s belong to the foreground and are scaled down to their original value. In the Max-shift 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 Max-shift method is that ROIs with multiple quality differentials cannot be encoded.

In the Max-shift method, the ROI coefficients are prioritized in the code-stream so that they are received (decoded) before the background. However, if the entire code-stream 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 code-stream has been

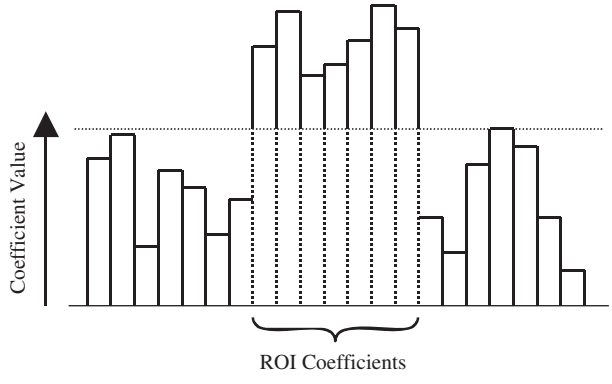


Figure 1.31 Max-shift method of ROI coding in JPEG 2000 Part 1

decoded. The complete separation of the ROI and background bit-planes in the Max-shift 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.

Given that the encoding of the ROI and the background coefficients in the Max-shift 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 are organized in the resolution progressive mode, the ROI data are 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, because of 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 regarding the background in the reconstructed image.

Experiments show that for the lossless coding of images with ROIs, the Max-shift method increases the bit rate by 1–8% (depending on image size and ROI size and shape) compared to the lossless coding of the images without ROI (Christopoulos, Askelof, and Larsson, 2000). This is a relatively small cost for achieving the ROI functionality.

1.6.2 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 (Liang and Talluri, 1999; Moccagata *et al.*, 2000).

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 code-blocks. Among the many advantages of this approach is improved error resilience, because any errors in the bit-stream, corresponding to a code-block, will be contained within that code-block. In addition, certain entropy-coding options described in Section 1.2.4.3 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 bit-plane after the fourth most significant bit-plane 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 bit-plane that can serve in error detection. The segmentation symbol is a binary ‘ 1010’ symbol whose presence is signaled in the marker segments. It is encoded with the uniform arithmetic coding context, and its correct decoding at the end of each bit-plane confirms the correctness of the decompressed data corresponding to that bit-plane. If the segmentation symbol is not decoded correctly, the data for that bit-plane and all the subsequent bit-planes corresponding to that code-block should be discarded. This is because the data encoded in the subsequent coding passes of that code-block depend on the previously encoded data.

Error resilience at the packet level can be achieved by using the SOP marker, which provides for spatial partitioning and resynchronization. This marker is placed in front of each packet in a tile and numbers the packets sequentially starting at zero. Also, the packet headers can be moved to either the main header (for all tiles, using PPM markers) or the tile header (using PPT markers), 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.

1.6.3 File Format

Most digital imaging standards provide a file format structure to encapsulate the encoded-image data. While the code-stream specifies the compressed image, the file format serves to provide useful information regarding the characteristics of the image and its proper use and display. Sometimes the file format includes redundant information that is also included in the code-stream, but such information is useful in that it allows trivial manipulation of the file without any knowledge of the code-stream syntax. A minimal file format, such as the one used in the JPEG baseline system, includes general information regarding the number of image components, their corresponding resolutions and bit depths, etc. However, two important components of a more comprehensive file format are *color space* and *metadata*. Without this information, an application might not know how to use or display an image properly. The color space 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 regarding 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 to 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 regarding an image without the need to decode it, which enables a fast text-based search in databases. The SPIFF file format defined in Part 3 extensions of the existing JPEG standard (ISO/IEC, 1995) 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 color space 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

encoded-image data. In order to balance flexibility with interoperability, the JP2 format defines two methods of color space specification. One method (known as the *enumerated* method) limits flexibility, but provides a high degree of interoperability by directly specifying only three color spaces, sRGB, gray scale, and sYCC. Another method known as the *restricted ICC* (International Color Consortium) (ICC, 1998) method, allows for the specification of a color space 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 color space 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 provides 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 (W3C, 2006). 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 color space flexibility by providing more enumerated color spaces (and also allow vendors to register additional values for color spaces) 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.

Acknowledgments

The authors would like to thank Brian Banister for generating the bit-plane results in Section 1.5.3 and Roddy Shuler for reviewing this chapter for technical accuracy.

References

Acharya, T. and Tsai, P.-S. (2005) *JPEG 2000 Standard for Image Compression – Concepts, Algorithms and VLSI Architectures*, John Wiley & Sons, Inc., Hoboken, NJ.

Adams, M. D. and Kossentini, F. (2000) Reversible integer-to-integer wavelet transforms for image compression: performance evaluation and analysis, *IEEE Transactions on Image Processing*, **9**(6), June, 1010–1024.

Adams, M. D., Man, H., Kossentini, F. and Ebrahimi, T. (2000) JPEG 2000: The Next Generation Still Image Compression Standard, ISO/IEC JTC1/SC29/WG1N1734, June 2000.

Albanesi, M. and Bertoluzza, S. (1995) Human vision model and wavelets for high-quality image compression, in *Proceedings of the 5th International Conference on Image Processing and Its Applications*, vol. 410, Edinburgh, UK, July 1995, pp. 311–315.

Antonini, M., Barlaud, M., Mathieu, P. and Daubechies, I. (1992) Image coding using wavelet transform, *IEEE Transactions on Image Processing*, **1**(2), April, 205–220.

Atsumi, E. and Farvardin, N. (1998) 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, IL, October 1998, pp. 87–91.

Calderbank, R. C., Daubechies, I., Sweldens, W. and Yeo, B.-L. (1998) Wavelet transforms that map integers to integers, *Applied and Computational Harmonic Analysis*, **5**(3), July, 332–369.

Christopoulos, C., Askelof, J. and Larsson, M. (2000) Efficient methods for encoding regions of interest in the upcoming JPEG 2000 Still Image Coding Standard, *IEEE Signal Processing Letters*, **7**(9), September, 247–249.

Christopoulos, C., Skodras, A. and Ebrahimi, T. (2000) The JPEG 2000 still image coding system: an overview, *IEEE Transactions on Consumer Electronics*, **46**(4), November, 1103–1127.

Chrysafis, C. and Ortega, A. (2000) Line-based, reduced memory, wavelet image compression, *IEEE Transactions on Image Processing*, **9**(3), March, 378–389.

Daubechies, I. and Sweldens, W. (1998) Factoring wavelet transforms into lifting steps, *Journal of Fourier Analysis Applications*, **4**(3), 247–269.

Ebrahimi, T., Santa-Cruz, D., Askelöf, J., Larsson, M. and Christopoulos, C. (2000) JPEG 2000 still image coding versus other standards, in *Proceedings of SPIE*, vol. 4115, San Diego, CA, July/August 2000, pp. 446–454.

Gormish, M. J., Lee, D. and Marcellin, M. W. (2000) JPEG 2000: overview, architecture, and applications, in *Proceedings of the IEEE International Conference on Image Processing*, Vancouver, Canada, September 2000.

ICC (International Color Consortium) (1998) ICC Profile Format Specification, ICC.1: 1998-09.

ISO/IEC (1993) Information Technology – Digital Compression and Coding of Continuous-Tone Still Images – Part 1: Requirements and Guidelines, ISO/IEC International Standard 10918-1, ITU-T Recommendation T.81.

ISO/IEC (1995) Information Technology – Digital Compression and Coding of Continuous-Tone Still Images – Part 3: Extensions, ISO/IEC International Standard 10918-3, ITU-T Recommendation T.84.

ISO/IEC (1997a) Call for Contributions for JPEG 2000 (JTC 1.29.14, 15444): Image Coding System, ISO/IEC JTC1/SC29/WG1N505, March 1997.

ISO/IEC (1997b) New Work Item: JPEG 2000 Image Coding System, ISO/IEC JTC1/SC29/WG1N390R, March 1997.

ISO/IEC (1999) Information Technology – Lossless and Near Lossless Compression of Continuous-Tone Still Images, ISO/IEC International Standard 14495-1, ITU Recommendation T.87.

ISO/IEC (2000) JPEG 2000 Verification Model 8.6 (Software), ISO/IEC JTC1/SC29/WG1N1894, December 2000.

Jones, P. W. (2007) Efficient JPEG 2000 VBR compression with true constant quality, *SMPTE Journal on Motion Imaging*, **7/8**, July/August.

Jones, P., Daly, S., Gaborski, R. and Rabbani, M. (1995) Comparative study of wavelet and DCT decompositions with equivalent quantization and encoding strategies for medical images, in *Proceedings of SPIE*, vol. 2431, San Diego, CA, February 1995, pp. 571–582.

Joshi, R. L., Rabbani, M. and Lepley, M. (2000) Comparison of multiple compression cycle performance for JPEG and JPEG 2000, in *Proceedings of SPIE*, vol. 4115, San Diego, CA, July/August 2000, pp. 492–501.

LeGall, D. and Tabatabai, A. (1988) Subband coding of digital images using symmetric kernel filters and arithmetic coding techniques, in *Proceedings of International Conference on Acoustic Speech and Signal Processing*, New York, April 1988, pp. 761–764.

Li, J. (1999) Visual progressive coding, in *Proceedings of SPIE*, vol. 3653, San Jose, CA, January 1999.

Li, J. and Lei, S. (1999) An embedded still image coder with rate-distortion optimization, *IEEE Transactions on Image Processing*, **8**(7), July, 913–924.

Liang, J. and Talluri, R. (1999) Tools for robust image and video coding in JPEG 2000 and MPEG-4 standards, in *Proceedings of SPIE*, vol. 3653, San Jose, CA, January 1999, pp. 40–51.

Liu, Z., Karam, L. J. and Watson, A. B. (2006) JPEG 2000 encoding with perceptual distortion control, *IEEE Transactions on Image Processing*, **15**(7), July, 1763–1778.

Marcellin, M., Flohr, T., Bilgin, A., Taubman, D., Ordentlich, E., Weinberger, M., Seroussi, G., Chrysafis, C., Fischer, T., Banister, B., Rabbani, M. and Joshi, R. (1999) Reduced Complexity Entropy Coding, ISO/IEC JTC1/SC29/WG1 Document N1312, June 1999.

Marcellin, M. W., Gormish, M. J., Bilgin, A. and Boliek, M. P. (2000) An overview of JPEG-2000, in *Proceedings of the Data Compression Conference*, Snowbird, UT, March 2000, pp. 523–541.

Marcellin, M. W., Lepley, M. A., Bilgin, A., Flohr, T. J., Chinen, T. T. and Kasner, J. H. (2002) An overview of quantization in JPEG 2000, *Signal Processing: Image Communications*, **17**(1), January, 73–84.

Marziliano, P., Dufaux, F., Winkler, S. and Ebrahimi, T. (2004) Perceptual blur and ringing metrics: application to JPEG 2000, *Signal Processing: Image Communication*, **19**(2), February, 163–172.

Moccagata., I., Sodagar, S., Liang, J. and Chen, H. (2000) Error resilient coding in JPEG-2000 and MPEG-4, *IEEE Journal of Selected Areas in Communications*, **18**(6), June, 899–914.

Nister, D. and Christopoulos, C. (1999) Lossless region of interest with embedded wavelet image coding, *Signal Processing*, **78**(1), October, 1–17.

Ordentlich, E., Weinberger, M. J. and Seroussi, G. (1998) A low complexity modeling approach for embedded coding of wavelet coefficients, in *Proceedings of the Data Compression Conference*, Snowbird, UT, March 1998, pp. 408–417.

O'Rourke, T. and Stevenson, R. (1995) Human visual system based wavelet decomposition for image compression, *Journal of Visual Communications and Image Representation*, **6**(2), June, 109–121.

Pennebaker, W. B. and Mitchell, J. L. (1993) *JPEG Still Image Data Compression Standard*, Van Nostrand Reinhold, New York.

Pennebaker, W. B., Mitchell, J. L., Langdon Jr, G. G. and Arps, R. B. (1988) An overview of the basic principles of the Q-coder adaptive binary arithmetic coder, *IBM Journal of Research Development*, **32**(6), November, 717–726.

Price, J. R. and Rabbani, M. (1999) Biased reconstruction for JPEG decoding, *Signal Processing Letters*, **6**(12), December, 297–299.

Rabbani, M. and Joshi, R. L. (2000) An overview of the JPEG 2000 still image compression standard, *Signal Processing: Image Communications*, **17**(1), January, 3–48.

Said, A. and Pearlman, W. A. (1996) A new fast and efficient image codec based on set partitioning in hierarchical trees, *IEEE Transactions on Circuits System Video Technology*, **6**(3), June, 243–250.

Santa-Cruz, D. and Ebrahimi, T. (2000) An analytical study of JPEG 2000 functionalities, in *Proceedings of IEEE International Conference on Image Processing*, Vancouver, Canada, September 2000.

Shapiro, J. M. (1993) Embedded image coding using zero trees of wavelet coefficients, *IEEE Transactions on Signal Processing*, **41**(12), December, 3445–3462.

Slattery, M. J. and Mitchell, J. L. (1998) The Qx-coder, *IBM Journal of Research Development*, **42**(6), November, 767–784.

SPIC (2002) Special Issue on JPEG 2000 still image compression standard, *Signal Processing: Image Communication*, **17**(1), January.

Sullivan, G. (1996) Efficient scalar quantization of exponential and Laplacian variables, *IEEE Transactions on Information Theory*, **42**(5), September, 1365–1374.

Sweldens, W. (1995) The lifting scheme: a new philosophy in biorthogonal wavelet constructions, in *Proceedings of SPIE*, vol. 2569, September 1995, pp. 68–79.

Sweldens, W. (1996) The lifting scheme: a custom-design construction of biorthogonal wavelets, *Applied and Computational Harmonic Analysis*, **3**(2), April, 186–200.

Sweldens, W. (1998) The lifting scheme: a construction of second generation wavelets, *Siam Journal of Mathematical Analysis*, **29**(2), March, 511–546.

Taubman, D. (2000) High performance scalable image compression with EBCOT, *IEEE Transactions on Image Processing*, **9**(7), July, 1158–1170.

Taubman, D. and Marcellin, M. W. (2001) *JPEG 2000: Image Compression Fundamentals, Practice and Standards*, Kluwer Academic Publishers, Boston, MA.

Taubman, D. S. and Marcellin, M. W. (2002) JPEG 2000: standard for interactive imaging, *Proceedings of the IEEE*, **90**(8), August, 1336–1357.

Taubman, D., Ordentlich, E., Weinberger, M. J. and Seroussi, G. (2002) Embedded block coding in JPEG 2000, *Signal Processing: Image Communications*, **17**(1), January, 49–72.

Unser, M. and Blu, T. (2003) Mathematical properties of the JPEG 2000 wavelet filters, *IEEE Transactions on Image Processing*, **12**(9), September, 1080–1090.

Vetterli, M. and Kovacevic, J. (1995) *Wavelet and Subband Coding*, Prentice Hall, Englewood Cliffs, NJ.

Villasenor, J. D., Belzer, B. and Liao, J. (1995) Wavelet filter evaluation for image compression, *IEEE Transactions on Image Processing*, **4**(8), August, 1053–1060.

W3C (2006) W3C, Extensible Markup Language (XML) 1.0, 4th edition, September 2006. Available at: http://www.w3.org/TR/Rec-xml.

Watson, A. B., Yang, G. Y., Solomon, J. A. and Villasenor, J. (1997) Visibility of wavelet quantization noise, *IEEE Transactions on Image Processing*, **6**(8), August, 1164–1175.

Woods, J. W. and Naveen, T. (1992) A filter based bit allocation scheme for subband compression of HDTV, *IEEE Transactions on Image Processing*, **1**(3), July, 436–440.

Zeng, W., Daly, S. and Lei, S. (2002) An overview of visual optimization tools in JPEG 2000, *Signal Processing: Image Communications*, **17**(1), January, 85–105.