ISO/IEC JTC 1/SC 29/WG 1 **N 2412**
Date: September 1, 2001

ISO/IEC JTC 1/SC 29/WG 1
(ITU-T SG8)

Coding of Still Pictures

JBIG	**JPEG**
Joint Bi-level Image Experts Group	Joint Photographic Experts Group

TITLE: The JPEG-2000 Still Image Compression Standard

SOURCE: Michael D. Adams
Dept. of Electrical and Computer Engineering
University of British Columbia
Vancouver, BC, Canada V6T 1Z4
E-mail: mdadams@ece.ubc.ca

PROJECT: JPEG 2000

STATUS:

REQUESTED ACTION:

DISTRIBUTION: Public

The JPEG-2000 Still Image Compression Standard* (Last Revised: September 1, 2001)

Michael D. Adams
Dept. of Elec. and Comp. Engineering, University of British Columbia
Vancouver, BC, Canada V6T 1Z4
`mailto:mdadams@ieee.org http://www.ece.ubc.ca/˜mdadams`

Abstract—**JPEG 2000, a new international standard for still image compression, is discussed at length. A high-level introduction to the JPEG-2000 standard is given, followed by a detailed technical description of the JPEG-2000 Part-1 codec.**

Keywords—**JPEG 2000, still image compression/coding, standards.**

I. Introduction

Digital imagery is pervasive in our world today. Consequently, standards for the efficient representation and interchange of digital images are essential. To date, some of the most successful still image compression standards have resulted from the ongoing work of the Joint Photographic Experts Group (JPEG). This group operates under the auspices of Joint Technical Committee 1, Subcommittee 29, Working Group 1 (JTC 1/SC 29/WG 1), a collaborative effort between the International Organization for Standardization (ISO) and International Telecommunication Union Standardization Sector (ITU-T). Both the JPEG [1, 2] and JPEG-LS [3] standards were born from the work of the JPEG committee. For the last few years, the JPEG committee has been working towards the establishment of a new standard known as JPEG 2000 (i.e., ISO/IEC 15444). The fruits of these labors are now coming to bear, as JPEG-2000 Part 1 (i.e., ISO/IEC 15444-1 [4]) has recently been approved as a new international standard.

In this paper, we provide a detailed technical description of the JPEG-2000 Part-1 codec, in addition to a brief overview of the JPEG-2000 standard. This exposition is intended to serve as a reader-friendly starting point for those interested in learning about JPEG 2000. Although many details are included in our presentation, some details are necessarily omitted. The reader should, therefore, refer to the standard [4] before attempting an implementation. The JPEG-2000 codec realization in the JasPer software [5–7] may also serve as a practical guide for implementors. (See Appendix A for more information about JasPer.)

The remainder of this paper is structured as follows. Section II begins with a overview of the JPEG-2000 standard. This is followed, in Section III, by a detailed description of the JPEG-2000 Part-1 codec. Finally, we conclude with some closing remarks in Section IV. Throughout our presentation, a basic understanding of image coding is assumed.

*This document is a revised version of the JPEG-2000 tutorial that I wrote which appeared in the JPEG working group document WG1N1734. The original tutorial contained numerous inaccuracies, some of which were introduced by changes in the evolving draft standard while others were due to typographical errors. Hopefully, most of these inaccuracies have been corrected in this revised document. In any case, this document will probably continue to evolve over time. Subsequent versions of this document will be made available from my home page (the URL for which is provided with my contact information).

II. JPEG 2000

The JPEG-2000 standard supports lossy and lossless compression of single-component (e.g., grayscale) and multi-component (e.g., color) imagery. In addition to this basic compression functionality, however, numerous other features are provided, including: 1) progressive recovery of an image by fidelity or resolution; 2) region of interest coding, whereby different parts of an image can be coded with differing fidelity; 3) random access to particular regions of an image without needing to decode the entire code stream; 4) a flexible file format with provisions for specifying opacity information and image sequences; and 5) good error resilience. Due to its excellent coding performance and many attractive features, JPEG 2000 has a very large potential application base. Some possible application areas include: image archiving, Internet, web browsing, document imaging, digital photography, medical imaging, remote sensing, and desktop publishing.

A. Why JPEG 2000?

Work on the JPEG-2000 standard commenced with an initial call for contributions [8] in March 1997. The purpose of having a new standard was twofold. First, it would address a number of weaknesses in the existing JPEG standard. Second, it would provide a number of new features not available in the JPEG standard. The preceding points led to several key objectives for the new standard, namely that it should: 1) allow efficient lossy and lossless compression within a single unified coding framework, 2) provide superior image quality, both objectively and subjectively, at low bit rates, 3) support additional features such as region of interest coding, and a more flexible file format, 4) avoid excessive computational and memory complexity. Undoubtedly, much of the success of the original JPEG standard can be attributed to its royalty-free nature. Consequently, considerable effort has been made to ensure that minimally-compliant JPEG-2000 codec can be implemented free of royalties[1].

B. Structure of the Standard

The JPEG-2000 standard is comprised of numerous parts, several of which are listed in Table I. For convenience, we will refer to the codec defined in Part 1 of the standard as the baseline codec. The baseline codec is simply the core (or minimal functionality) coding system for the JPEG-2000 standard. Parts 2 and 3 describe extensions to the baseline codec that are useful for certain specific applications such as intraframe-style video

[1]Whether these efforts ultimately prove successful remains to be seen, however, as there are still some unresolved intellectual property issues at the time of this writing.

compression. In this paper, we will, for the most part, limit our discussion to the baseline codec. Some of the extensions proposed for inclusion in Part 2 will be discussed briefly. Unless otherwise indicated, our exposition considers only the baseline system.

For the most part, the JPEG-2000 standard is written from the point of view of the decoder. That is, the decoder is defined quite precisely with many details being normative in nature (i.e., required for compliance), while many parts of the encoder are less rigidly specified. Obviously, implementors must make a very clear distinction between normative and informative clauses in the standard. For the purposes of our discussion, however, we will only make such distinctions when absolutely necessary.

III. JPEG-2000 Codec

Having briefly introduced the JPEG-2000 standard, we are now in a position to begin examining the JPEG-2000 codec in detail. The codec is based on wavelet/subband coding techniques [9,10]. It handles both lossy and lossless compression using the same transform-based framework, and borrows heavily on ideas from the embedded block coding with optimized truncation (EBCOT) scheme [11,12]. In order to facilitate both lossy and lossless coding in an efficient manner, reversible integer-to-integer [13, 14] and nonreversible real-to-real transforms are employed. To code transform data, the codec makes use of bit-plane coding techniques. For entropy coding, a context-based adaptive binary arithmetic coder [15] is used—more specifically, the MQ coder from the JBIG2 standard [16]. Two levels of syntax are employed to represent the coded image: a code stream and file format syntax. The code stream syntax is similar in spirit to that used in the JPEG standard.

The remainder of Section III is structured as follows. First, Sections III-A to III-C, discuss the source image model and how an image is internally represented by the codec. Next, Section III-D examines the basic structure of the codec. This is followed, in Sections III-E to III-M by a detailed explanation of the coding engine itself. Next, Sections III-N and III-O explain the syntax used to represent a coded image. Finally, Section III-P briefly describes some extensions proposed for inclusion in Part 2 of the standard.

A. Source Image Model

Before examining the internals of the codec, it is important to understand the image model that it employs. From the codec's point of view, an image is comprised of one or more components (up to a limit of 2^{14}), as shown in Fig. 1(a). As illustrated in Fig. 1(b), each component consists of a rectangular array of samples. The sample values for each component are integer valued, and can be either signed or unsigned with a precision from 1 to 38 bits/sample. The signedness and precision of the sample data are specified on a per-component basis.

All of the components are associated with the same spatial extent in the source image, but represent different spectral or auxiliary information. For example, a RGB color image has three components with one component representing each of the red, green, and blue color planes. In the simple case of a grayscale image, there is only one component, corresponding to the luminance plane. The various components of an image need not

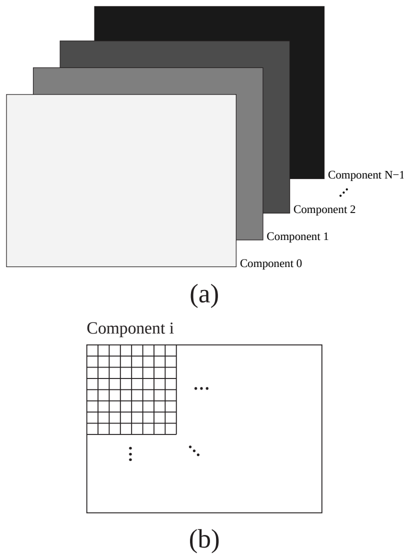


Fig. 1. Source image model. (a) An image with N components. (b) Individual component.

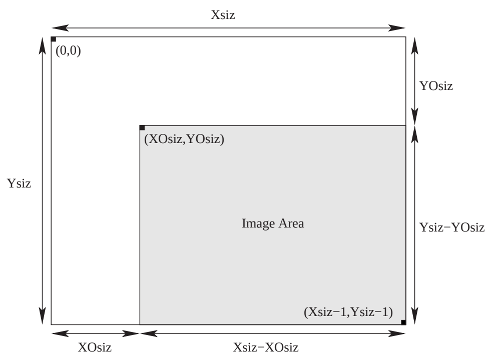


Fig. 2. Reference grid.

be sampled at the same resolution. Consequently, the components themselves can have different sizes. For example, when color images are represented in a luminance-chrominance color space, the luminance information is often more finely sampled than the chrominance data.

B. Reference Grid

Given an image, the codec describes the geometry of the various components in terms of a rectangular grid called the reference grid. The reference grid has the general form shown in Fig. 2. The grid is of size Xsiz $\times$ Ysiz with the origin located at its top-left corner. The region with its top-left corner at $(\text{XOsiz}, \text{YOsiz})$ and bottom-right corner at $(\text{Xsiz} - 1, \text{Ysiz} - 1)$ is called the image area, and corresponds to the picture data to be represented. The width and height of the reference grid cannot exceed $2^{32} - 1$ units, imposing an upper bound on the size of an image that can be handled by the codec.

All of the components are mapped onto the image area of the reference grid. Since components need not be sampled at the full resolution of the reference grid, additional information is required in order to establish this mapping. For each component, we indicate the horizontal and vertical sampling period in units of the reference grid, denoted as XRsiz and YRsiz, re-

TABLE I
PARTS OF THE STANDARD

Part	Title	Purpose
1	Core coding system	Specifies the core (or minimum functionality) codec for the JPEG-2000 family of standards.
2	Extensions	Specifies additional functionalities that are useful in some applications but need not be supported by all codecs.
3	Motion JPEG-2000	Specifies extensions to JPEG-2000 for intraframe-style video compression.
4	Conformance testing	Specifies the procedure to be employed for compliance testing.
5	Reference software	Provides sample software implementations of the standard to serve as a guide for implementors.

spectively. These two parameters uniquely specify a (rectangular) sampling grid consisting of all points whose horizontal and vertical positions are integer multiples of XRsiz and YRsiz, respectively. All such points that fall within the image area, constitute samples of the component in question. Thus, in terms of its own coordinate system, a component will have the size $\left(\left\lceil \frac{\text{Xsiz}}{\text{XRsiz}} \right\rceil - \left\lceil \frac{\text{XOsiz}}{\text{XRsiz}} \right\rceil\right) \times \left(\left\lceil \frac{\text{Ysiz}}{\text{YRsiz}} \right\rceil - \left\lceil \frac{\text{YOsiz}}{\text{YRsiz}} \right\rceil\right)$ and its top-left sample will correspond to the point $\left(\left\lceil \frac{\text{XOsiz}}{\text{XRsiz}} \right\rceil, \left\lceil \frac{\text{YOsiz}}{\text{YRsiz}} \right\rceil\right)$. Note that the reference grid also imposes a particular alignment of samples from the various components relative to one another.

From the diagram, the size of the image area is $(\text{Xsiz} - \text{XOsiz}) \times (\text{Ysiz} - \text{YOsiz})$. For a given image, many combinations of the Xsiz, Ysiz, XOsiz, and YOsiz parameters can be chosen to obtain an image area with the same size. Thus, one might wonder why the XOsiz and YOsiz parameters are not fixed at zero while the Xsiz and Ysiz parameters are set to the size of the image. As it turns out, there are subtle implications to changing the XOsiz and YOsiz parameters (while keeping the size of the image area constant). Such changes affect codec behavior in several important ways, as will be described later. This behavior allows a number of basic operations to be performed efficiently on coded images, such as cropping, horizontal/vertical flipping, and rotation by an integer multiple of 90 degrees.

C. Tiling

In some situations, an image may be quite large in comparison to the amount of memory available to the codec. Consequently, it is not always feasible to code the entire image as a single atomic unit. To solve this problem, the codec allows an image to be broken into smaller pieces, each of which is independently coded. More specifically, an image is partitioned into one or more disjoint rectangular regions called tiles. As shown in Fig. 3, this partitioning is performed with respect to the reference grid by overlaying the reference grid with a rectangular tiling grid having horizontal and vertical spacings of XTsiz and YTsiz, respectively. The origin of the tiling grid is aligned with the point $(\text{XTOsiz}, \text{YTOsiz})$. Tiles have a nominal size of $\text{XTsiz} \times \text{YTsiz}$, but those bordering on the edges of the image area may have a size which differs from the nominal size. The tiles are numbered in raster scan order (starting at zero).

By mapping the position of each tile from the reference grid to the coordinate systems of the individual components, a partitioning of the components themselves is obtained. For example, suppose that a tile has an upper left corner and lower right corner with coordinates $(\text{tx}_0, \text{ty}_0)$ and $(\text{tx}_1 - 1, \text{ty}_1 - 1)$, respectively. Then, in the coordinate space of a particular component, the tile would have an upper left corner and lower right corner with coordinates $(\text{tcx}_0, \text{tcy}_0)$ and $(\text{tcx}_1 - 1, \text{tcy}_1 - 1)$, respectively, where

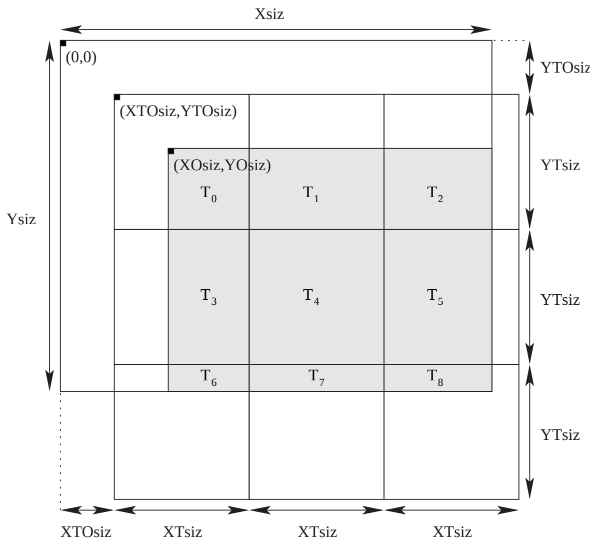


Fig. 3. Tiling on the reference grid.

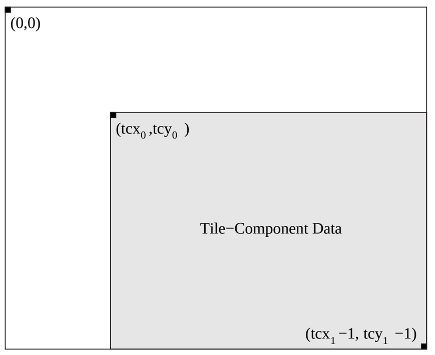


Fig. 4. Tile-component coordinate system.

$$(\text{tcx}_0, \text{tcy}_0) = (\lceil \text{tx}_0/\text{XRsiz} \rceil, \lceil \text{ty}_0/\text{YRsiz} \rceil) \tag{1a}$$
$$(\text{tcx}_1, \text{tcy}_1) = (\lceil \text{tx}_1/\text{XRsiz} \rceil, \lceil \text{ty}_1/\text{YRsiz} \rceil). \tag{1b}$$

These equations correspond to the illustration in Fig. 4. The portion of a component that corresponds to a single tile is referred to as a tile-component. Although the tiling grid is regular with respect to the reference grid, it is important to note that the grid may not necessarily be regular with respect to the coordinate systems of the components.

D. Codec Structure

The general structure of the codec is shown in Fig. 5 with the form of the encoder given by Fig. 5(a) and the decoder given by Fig. 5(b). From these diagrams, the key processes associated with the codec can be identified: 1) preprocessing/postprocessing, 2) intercomponent transform, 3) intracomponent transform, 4) quantization/dequantization, 5) tier-1 coding, 6) tier-2 coding, and 7) rate control. The decoder structure essentially mirrors that of the encoder. That is, with the exception of rate control, there is a one-to-one correspondence between functional blocks in the encoder and decoder. Each functional block in the decoder either exactly or approximately inverts the effects of its corresponding block in the encoder. Since tiles are coded independently of one another, the input image is (conceptually, at least) processed one tile at a time. In the sections that follow, each of the above processes is examined in more detail.

E. Preprocessing/Postprocessing

The codec expects its input sample data to have a nominal dynamic range that is approximately centered about zero. The preprocessing stage of the encoder simply ensures that this expectation is met. Suppose that a particular component has P bits/sample. The samples may be either signed or unsigned, leading to a nominal dynamic range of $[-2^{P-1}, 2^{P-1}-1]$ or $[0, 2^P - 1]$, respectively. If the sample values are unsigned, the nominal dynamic range is clearly not centered about zero. Thus, the nominal dynamic range of the samples is adjusted by subtracting a bias of 2^{P-1} from each of the sample values. If the sample values for a component are signed, the nominal dynamic range is already centered about zero, and no processing is required. By ensuring that the nominal dynamic range is centered about zero, a number of simplifying assumptions could be made in the design of the codec (e.g., with respect to context modeling, numerical overflow, etc.).

The postprocessing stage of the decoder essentially undoes the effects of preprocessing in the encoder. If the sample values for a component are unsigned, the original nominal dynamic range is restored. Lastly, in the case of lossy coding, clipping is performed to ensure that the sample values do not exceed the allowable range.

F. Intercomponent Transform

In the encoder, the preprocessing stage is followed by the forward intercomponent transform stage. Here, an intercomponent transform can be applied to the tile-component data. Such a transform operates on all of the components together, and serves to reduce the correlation between components, leading to improved coding efficiency.

Only two intercomponent transforms are defined in the baseline JPEG-2000 codec: the irreversible color transform (ICT) and reversible color transform (RCT). The ICT is nonreversible and real-to-real in nature, while the RCT is reversible and integer-to-integer. Both of these transforms essentially map image data from the RGB to YCrCb color space. The transforms are defined to operate on the first three components of an image, with the assumption that components 0, 1, and 2 correspond to the red, green, and blue color planes. Due to the nature of these transforms, the components on which they operate must be sampled at the same resolution (i.e., have the same size). As a consequence of the above facts, the ICT and RCT can only be employed when the image being coded has at least three components, and the first three components are sampled at the same resolution. The ICT may only be used in the case of lossy coding, while the RCT can be used in either the lossy or lossless case. Even if a transform can be legally employed, it is not necessary to do so. That is, the decision to use a multicomponent transform is left at the discretion of the encoder. After the intercomponent transform stage in the encoder, data from each component is treated independently.

The ICT is nothing more than the classic RGB to YCrCb color space transform. The forward transform is defined as

$$\begin{bmatrix} V_0(x,y) \\ V_1(x,y) \\ V_2(x,y) \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.16875 & -0.33126 & 0.5 \\ 0.5 & -0.41869 & -0.08131 \end{bmatrix} \begin{bmatrix} U_0(x,y) \\ U_1(x,y) \\ U_2(x,y) \end{bmatrix} \tag{2}$$

where $U_0(x,y)$, $U_1(x,y)$, and $U_2(x,y)$ are the input components corresponding to the red, green, and blue color planes, respectively, and $V_0(x,y)$, $V_1(x,y)$, and $V_2(x,y)$ are the output components corresponding to the Y, Cr, and Cb planes, respectively. The inverse transform can be shown to be

$$\begin{bmatrix} U_0(x,y) \\ U_1(x,y) \\ U_2(x,y) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.34413 & -0.71414 \\ 1 & -1.772 & 0 \end{bmatrix} \begin{bmatrix} V_0(x,y) \\ V_1(x,y) \\ V_2(x,y) \end{bmatrix} \tag{3}$$

The RCT is simply a reversible integer-to-integer approximation to the ICT (similar to that proposed in [14]). The forward transform is given by

$$V_0(x,y) = \left\lfloor \tfrac{1}{4}\left(U_0(x,y) + 2U_1(x,y) + U_2(x,y)\right) \right\rfloor \tag{4a}$$
$$V_1(x,y) = U_2(x,y) - U_1(x,y) \tag{4b}$$
$$V_2(x,y) = U_0(x,y) - U_1(x,y) \tag{4c}$$

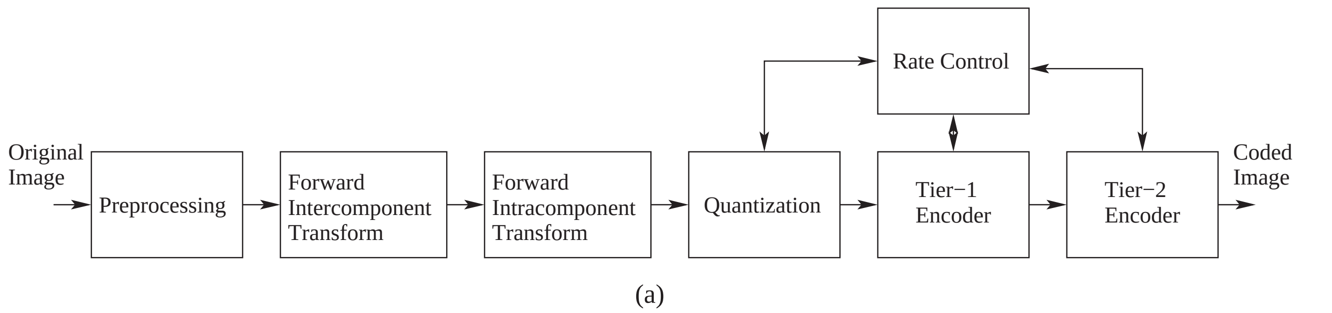
where $U_0(x,y)$, $U_1(x,y)$, $U_2(x,y)$, $V_0(x,y)$, $V_1(x,y)$, and $V_2(x,y)$ are defined as above. The inverse transform can be shown to be

$$U_1(x,y) = V_0(x,y) - \left\lfloor \tfrac{1}{4}\left(V_1(x,y) + V_2(x,y)\right) \right\rfloor \tag{5a}$$
$$U_0(x,y) = V_2(x,y) + U_1(x,y) \tag{5b}$$
$$U_2(x,y) = V_1(x,y) + U_1(x,y) \tag{5c}$$

The inverse intercomponent transform stage in the decoder essentially undoes the effects of the forward intercomponent transform stage in the encoder. If a multicomponent transform was applied during encoding, its inverse is applied here. Unless the transform is reversible, however, the inversion may only be approximate due to the effects of finite-precision arithmetic.

G. Intracomponent Transform

Following the intercomponent transform stage in the encoder is the intracomponent transform stage. In this stage, transforms that operate on individual components can be applied. The particular type of operator employed for this purpose is the wavelet transform. Through the application of the wavelet transform, a component is split into numerous frequency bands (i.e., subbands). Due to the statistical properties of these subband signals,



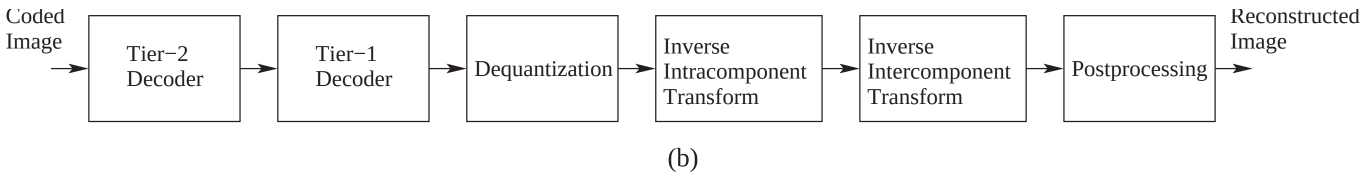
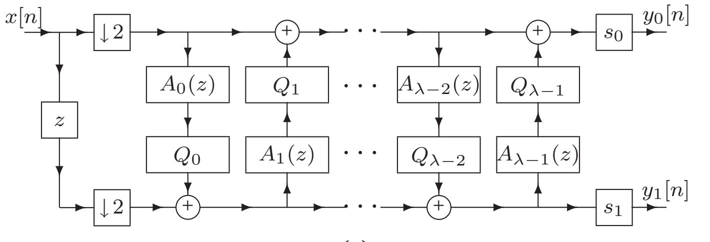


Fig. 5. Codec structure. The structure of the (a) encoder and (b) decoder.

the transformed data can usually be coded more efficiently than the original untransformed data.

Both reversible integer-to-integer [13, 17–19] and nonreversible real-to-real wavelet transforms are employed by the baseline codec. The basic building block for such transforms is the 1-D 2-channel perfect-reconstruction (PR) uniformly-maximally-decimated (UMD) filter bank (FB) which has the general form shown in Fig. 6. Here, we focus on the lifting realization of the UMDFB [20, 21], as it can be used to implement the reversible integer-to-integer and nonreversible real-to-real wavelet transforms employed by the baseline codec. In fact, for this reason, it is likely that this realization strategy will be employed by many codec implementations. The analysis side of the UMDFB, depicted in Fig. 6(a), is associated with the forward transform, while the synthesis side, depicted in Fig. 6(b), is associated with the inverse transform. In the diagram, the $\{A_i(z)\}_{i=0}^{\lambda-1}$, $\{Q_i(x)\}_{i=0}^{\lambda-1}$, and $\{s_i\}_{i=0}^{1}$ denote filter transfer functions, quantization operators, and (scalar) gains, respectively. To obtain integer-to-integer mappings, the $\{Q_i(x)\}_{i=0}^{\lambda-1}$ are selected such that they always yield integer values, and the $\{s_i\}_{i=0}^{1}$ are chosen as integers. For real-to-real mappings, the $\{Q_i(x)\}_{i=0}^{\lambda-1}$ are simply chosen as the identity, and the $\{s_i\}_{i=0}^{1}$ are selected from the real numbers. To facilitate filtering at signal boundaries, symmetric extension [22] is employed. Since an image is a 2-D signal, clearly we need a 2-D UMDFB. By applying the 1-D UMDFB in both the horizontal and vertical directions, a 2-D UMDFB is effectively obtained. The wavelet transform is then calculated by recursively applying the 2-D UMDFB to the lowpass subband signal obtained at each level in the decomposition.

Suppose that a $(R-1)$-level wavelet transform is to be employed. To compute the forward transform, we apply the analysis side of the 2-D UMDFB to the tile-component data in an iterative manner, resulting in a number of subband signals being produced. Each application of the analysis side of the 2-D UMDFB yields four subbands: 1) horizontally and vertically lowpass (LL), 2) horizontally lowpass and vertically highpass (LH), 3) horizontally highpass and vertically lowpass (HL), and 4) horizontally and vertically highpass (HH). A $(R-1)$-level wavelet decomposition is associated with R resolution levels, numbered from 0 to $R-1$, with 0 and $R-1$ corresponding to the coarsest and finest resolutions, respectively. Each subband of the decomposition is identified by its orientation (e.g., LL, LH, HL, HH) and its corresponding resolution level (e.g., $0, 1, \ldots, R-1$). The input tile-component signal is considered to be the LL$_{R-1}$ band. At each resolution level (except the lowest) the LL band is further decomposed. For example, the LL$_{R-1}$ band is decomposed to yield the LL$_{R-2}$, LH$_{R-2}$, HL$_{R-2}$, and HH$_{R-2}$ bands. Then, at the next level, the LL$_{R-2}$ band is decomposed, and so on. This process repeats until the LL$_0$ band is obtained, and results in the subband structure illustrated in Fig. 7. In the degenerate case where no transform is applied, $R=1$, and we effectively have only one subband (i.e., the LL_0 band).



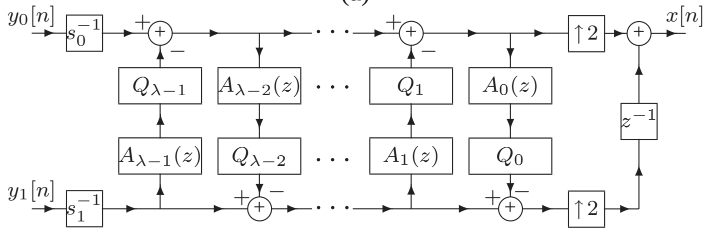


Fig. 6. Lifting realization of 1-D 2-channel PR UMDFB. (a) Analysis side. (b) Synthesis side.

As described above, the wavelet decomposition can be associated with data at R different resolutions. Suppose that the top-left and bottom-right samples of a tile-component have coordinates $(\mathrm{tcx}_0, \mathrm{tcy}_0)$ and $(\mathrm{tcx}_1 - 1, \mathrm{tcy}_1 - 1)$, respectively. This

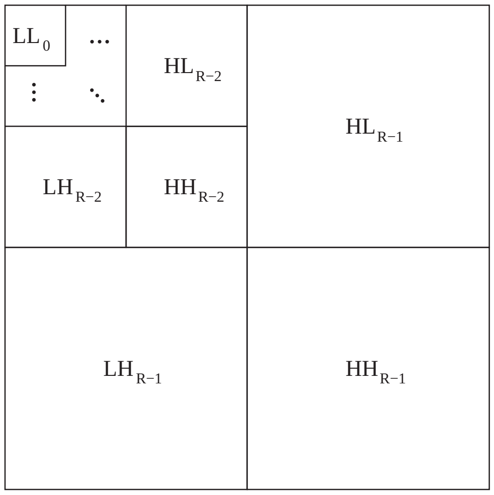


Fig. 7. Subband structure.

being the case, the top-left and bottom-right samples of the tile-component at resolution r have coordinates $(\text{trx}_0, \text{try}_0)$ and $(\text{trx}_1 - 1, \text{try}_1 - 1)$, respectively, given by

$$(\text{trx}_0, \text{try}_0) = \left(\left\lceil \text{tcx}_0/2^{R-r-1}\right\rceil, \left\lceil \text{tcy}_0/2^{R-r-1}\right\rceil\right) \quad (6a)$$

$$(\text{trx}_1, \text{try}_1) = \left(\left\lceil \text{tcx}_1/2^{R-r-1}\right\rceil, \left\lceil \text{tcy}_1/2^{R-r-1}\right\rceil\right) \quad (6b)$$

where r is the particular resolution of interest. Thus, the tile-component signal at a particular resolution has the size $(\text{trx}_1 - \text{trx}_0) \times (\text{try}_1 - \text{try}_0)$.

Not only are the coordinate systems of the resolution levels important, but so too are the coordinate systems for the various subbands. Suppose that we denote the coordinates of the upper left and lower right samples in a subband as $(\text{tbx}_0, \text{tby}_0)$ and $(\text{tbx}_1 - 1, \text{tby}_1 - 1)$, respectively. These quantities are computed as

$$(\text{tbx}_0, \text{tby}_0) = \begin{cases} \left(\left\lceil \frac{\text{tcx}_0}{2^{R-r-1}} \right\rceil, \left\lceil \frac{\text{tcy}_0}{2^{R-r-1}} \right\rceil\right) & \text{for LL band} \\ \left(\left\lceil \frac{\text{tcx}_0}{2^{R-r-1}} - \frac{1}{2} \right\rceil, \left\lceil \frac{\text{tcy}_0}{2^{R-r-1}} \right\rceil\right) & \text{for HL band} \\ \left(\left\lceil \frac{\text{tcx}_0}{2^{R-r-1}} \right\rceil, \left\lceil \frac{\text{tcy}_0}{2^{R-r-1}} - \frac{1}{2} \right\rceil\right) & \text{for LH band} \\ \left(\left\lceil \frac{\text{tcx}_0}{2^{R-r-1}} - \frac{1}{2} \right\rceil, \left\lceil \frac{\text{tcy}_0}{2^{R-r-1}} - \frac{1}{2} \right\rceil\right) & \text{for HH band} \end{cases} \quad (7a)$$

$$(\text{tbx}_1, \text{tby}_1) = \begin{cases} \left(\left\lceil \frac{\text{tcx}_1}{2^{R-r-1}} \right\rceil, \left\lceil \frac{\text{tcy}_1}{2^{R-r-1}} \right\rceil\right) & \text{for LL band} \\ \left(\left\lceil \frac{\text{tcx}_1}{2^{R-r-1}} - \frac{1}{2} \right\rceil, \left\lceil \frac{\text{tcy}_1}{2^{R-r-1}} \right\rceil\right) & \text{for HL band} \\ \left(\left\lceil \frac{\text{tcx}_1}{2^{R-r-1}} \right\rceil, \left\lceil \frac{\text{tcy}_1}{2^{R-r-1}} - \frac{1}{2} \right\rceil\right) & \text{for LH band} \\ \left(\left\lceil \frac{\text{tcx}_1}{2^{R-r-1}} - \frac{1}{2} \right\rceil, \left\lceil \frac{\text{tcy}_1}{2^{R-r-1}} - \frac{1}{2} \right\rceil\right) & \text{for HH band} \end{cases} \quad (7b)$$

where r is the resolution level to which the band belongs, R is the number of resolution levels, and tcx_0, tcy_0, tcx_1, and tcy_1 are as defined in (1). Thus, a particular band has the size $(\text{tbx}_1 - \text{tbx}_0) \times (\text{tby}_1 - \text{tby}_0)$. From the above equations, we can also see that $(\text{tbx}_0, \text{tby}_0) = (\text{trx}_0, \text{try}_0)$ and $(\text{tbx}_1, \text{tby}_1) = (\text{trx}_1, \text{try}_1)$ for the LL_r band, as one would expect. (This should be the case since the LL_r band is equivalent to a reduced resolution version of the original data.) As will be seen, the coordinate systems for the various resolutions and subbands of a tile-component play an important role in codec behavior.

By examining (1), (6), and (7), we observe that the coordinates of the top-left sample for a particular subband, denoted $(\text{tbx}_0, \text{tby}_0)$, are partially determined by the XOsiz and YOsiz parameters of the reference grid. At each level of the decomposition, the parity (i.e., oddness/evenness) of tbx_0 and tby_0 affects the outcome of the downsampling process (since downsampling is shift variant). In this way, the XOsiz and YOsiz parameters have a subtle, yet important, effect on the transform calculation.

Having described the general transform framework, we now describe the two specific wavelet transforms supported by the baseline codec: the 5/3 and 9/7 transforms. The 5/3 transform is reversible, integer-to-integer, and nonlinear. This transform was proposed in [13], and is simply an approximation to a linear wavelet transform proposed in [23]. The 5/3 transform has an underlying 1-D UMDFB with the parameters:

$$\lambda = 2, \quad A_0(z) = -\tfrac{1}{2}(z+1), \quad A_1(z) = \tfrac{1}{4}(1+z^{-1}), \quad (8)$$
$$Q_0(x) = -\lfloor -x \rfloor, \quad Q_1(x) = \left\lfloor x + \tfrac{1}{2} \right\rfloor, \quad s_0 = s_1 = 1.$$

The 9/7 transform is nonreversible and real-to-real. This transform, proposed in [10], is also employed in the FBI fingerprint compression standard [24] (although the normalizations differ). The 9/7 transform has an underlying 1-D UMDFB with the parameters:

$$\lambda = 4, \quad A_0(z) = \alpha_0(z+1), \quad A_1(z) = \alpha_1(1+z^{-1}), \quad (9)$$
$$A_2(z) = \alpha_2(z+1), \quad A_3(z) = \alpha_3(1+z^{-1}),$$
$$Q_i(x) = x \text{ for } i = 0, 1, 2, 3,$$
$$\alpha_0 \approx -1.586134, \quad \alpha_1 \approx -0.052980, \quad \alpha_2 \approx 0.882911,$$
$$\alpha_3 \approx 0.443506, \quad s_0 \approx 1.230174, \quad s_1 = 1/s_0.$$

Since the 5/3 transform is reversible, it can be employed for either lossy or lossless coding. The 9/7 transform, lacking the reversible property, can only be used for lossy coding. The number of resolution levels is a parameter of each transform. A typical value for this parameter is six (for a sufficiently large image). The encoder may transform all, none, or a subset of the components. This choice is at the encoder's discretion.

The inverse intracomponent transform stage in the decoder essentially undoes the effects of the forward intracomponent transform stage in the encoder. If a transform was applied to a particular component during encoding, the corresponding inverse transform is applied here. Due to the effects of finite-precision arithmetic, the inversion process is not guaranteed to be exact unless reversible transforms are employed.

H. Quantization/Dequantization

In the encoder, after the tile-component data has been transformed (by intercomponent and/or intracomponent transforms), the resulting coefficients are quantized. Quantization allows greater compression to be achieved, by representing transform coefficients with only the minimal precision required to obtain the desired level of image quality. Quantization of transform coefficients is one of the two primary sources of information loss in the coding path (the other source being the discarding of coding pass data as will be described later).

Transform coefficients are quantized using scalar quantization with a deadzone. A different quantizer is employed for the coefficients of each subband, and each quantizer has only one parameter, its step size. Mathematically, the quantization process is defined as

$$V(x,y) = \lfloor |U(x,y)| / \Delta \rfloor \operatorname{sgn} U(x,y) \qquad (10)$$

where Δ is the quantizer step size, $U(x,y)$ is the input subband signal, and $V(x,y)$ denotes the output quantizer indices for the subband. Since this equation is specified in an informative clause of the standard, encoders need not use this precise formula. This said, however, it is likely that many encoders will, in fact, use the above equation.

The baseline codec has two distinct modes of operation, referred to herein as integer mode and real mode. In integer mode, all transforms employed are integer-to-integer in nature (e.g., RCT, 5/3 WT). In real mode, real-to-real transforms are employed (e.g., ICT, 9/7 WT). In integer mode, the quantizer step sizes are always fixed at one, effectively bypassing quantization and forcing the quantizer indices and transform coefficients to be one and the same. In this case, lossy coding is still possible, but rate control is achieved by another mechanism (to be discussed later). In the case of real mode (which implies lossy coding), the quantizer step sizes are chosen in conjunction with rate control. Numerous strategies are possible for the selection of the quantizer step sizes, as will be discussed later in Section III-L.

As one might expect, the quantizer step sizes used by the encoder are conveyed to the decoder via the code stream. In passing, we note that the step sizes specified in the code stream are relative and not absolute quantities. That is, the quantizer step size for each band is specified relative to the nominal dynamic range of the subband signal.

In the decoder, the dequantization stage tries to undo the effects of quantization. Unless all of the quantizer step sizes are less than or equal to one, the quantization process will normally result in some information loss, and this inversion process is only approximate. The quantized transform coefficient values are obtained from the quantizer indices. Mathematically, the dequantization process is defined as

$$U(x,y) = (V(x,y) + r \operatorname{sgn} V(x,y))\, \Delta \qquad (11)$$

where Δ is the quantizer step size, r is a bias parameter, $V(x,y)$ are the input quantizer indices for the subband, and $U(x,y)$ is the reconstructed subband signal. Although the value of r is not normatively specified in the standard, it is likely that many decoders will use the value of one half.

I. Tier-1 Coding

After quantization is performed in the encoder, tier-1 coding takes place. This is the first of two coding stages. The quantizer indices for each subband are partitioned into code blocks. Code blocks are rectangular in shape, and their nominal size is a free parameter of the coding process, subject to certain constraints, most notably: 1) the nominal width and height of a code block must be an integer power of two, and 2) the product of the nominal width and height cannot exceed 4096.

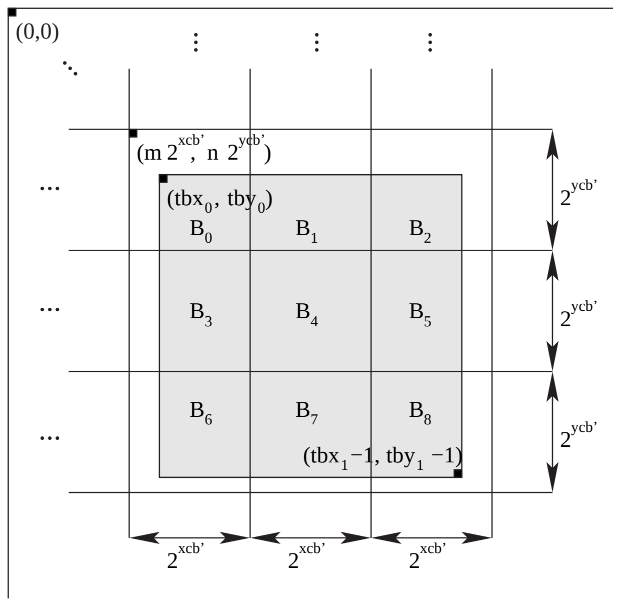


Fig. 8. Partitioning of subband into code blocks.

Suppose that the nominal code block size is tentatively chosen to be $2^{\text{xcb}} \times 2^{\text{ycb}}$. In tier-2 coding, yet to be discussed, code blocks are grouped into what are called precincts. Since code blocks are not permitted to cross precinct boundaries, a reduction in the nominal code block size may be required if the precinct size is sufficiently small. Suppose that the nominal code block size after any such adjustment is $2^{\text{xcb'}} \times 2^{\text{ycb'}}$ where xcb' $\leq$ xcb and ycb' $\leq$ ycb. The subband is partitioned into code blocks by overlaying the subband with a rectangular grid having horizontal and vertical spacings of $2^{\text{xcb'}}$ and $2^{\text{ycb'}}$, respectively, as shown in Fig. 8. The origin of this grid is anchored at $(0,0)$ in the coordinate system of the subband. A typical choice for the nominal code block size is 64×64 (i.e., xcb $= 6$ and ycb $= 6$).

Let us, again, denote the coordinates of the top-left sample in a subband as $(\text{tbx}_0, \text{tby}_0)$. As explained in Section III-G, the quantity $(\text{tbx}_0, \text{tby}_0)$ is partially determined by the reference grid parameters XOsiz and YOsiz. In turn, the quantity $(\text{tbx}_0, \text{tby}_0)$ affects the position of code block boundaries within a subband. In this way, the XOsiz and YOsiz parameters have an important effect on the behavior of the tier-1 coding process (i.e., they affect the location of code block boundaries).

After a subband has been partitioned into code blocks, each of the code blocks is independently coded. The coding is performed using the bit-plane coder described later in Section III-J. For each code block, an embedded code is produced, comprised of numerous coding passes. The output of the tier-1 encoding process is, therefore, a collection of coding passes for the various code blocks.

On the decoder side, the bit-plane coding passes for the various code blocks are input to the tier-1 decoder, these passes are decoded, and the resulting data is assembled into subbands. In this way, we obtain the reconstructed quantizer indices for each subband. In the case of lossy coding, the reconstructed quantizer indices may only be approximations to the quantizer indices originally available at the encoder. This is attributable to the fact that the code stream may only include a subset of the coding passes generated by the tier-1 encoding process. In the lossless case, the reconstructed quantizer indices must be same as the original indices on the encoder side, since all coding passes must be included for lossless coding.

J. Bit-Plane Coding

The tier-1 coding process is essentially one of bit-plane coding. After all of the subbands have been partitioned into code blocks, each of the resulting code blocks is independently coded using a bit-plane coder. Although the bit-plane coding technique employed is similar to those used in the embedded zerotree wavelet (EZW) [25] and set partitioning in hierarchical trees (SPIHT) [26] codecs, there are two notable differences: 1) no interband dependencies are exploited, and 2) there are three coding passes per bit plane instead of two. The first difference follows from the fact that each code block is completely contained within a single subband, and code blocks are coded independently of one another. By not exploiting interband dependencies, improved error resilience can be achieved. The second difference is arguably less fundamental. Using three passes per bit plane instead of two reduces the amount of data associated with each coding pass, facilitating finer control over rate. Also, using an additional pass per bit plane allows better prioritization of important data, leading to improved coding efficiency.

As noted above, there are three coding passes per bit plane. In order, these passes are as follows: 1) significance, 2) refinement, and 3) cleanup. All three types of coding passes scan the samples of a code block in the same fixed order shown in Fig. 10. The code block is partitioned into horizontal stripes, each having a nominal height of four samples. If the code block height is not a multiple of four, the height of the bottom stripe will be less than this nominal value. As shown in the diagram, the stripes are scanned from top to bottom. Within a stripe, columns are scanned from left to right. Within a column, samples are scanned from top to bottom.

The bit-plane encoding process generates a sequence of symbols for each coding pass. Some or all of these symbols may be entropy coded. For the purposes of entropy coding, a context-based adaptive binary arithmetic coder is used—more specifically, the MQ coder from the JBIG2 standard [16]. For each pass, all of the symbols are either arithmetically coded or raw coded (i.e., the binary symbols are emitted as raw bits with simple bit stuffing). The arithmetic and raw coding processes both ensure that certain bit patterns never occur in the output, allowing such patterns to be used for error resilience purposes.

Cleanup passes always employ arithmetic coding. In the case of the significance and refinement passes, two possibilities exist, depending on whether the so called arithmetic-coding bypass mode (also known as lazy mode) is enabled. If lazy mode is enabled, only the significance and refinement passes for the four most significant bit planes use arithmetic coding, while the remaining such passes are raw coded. Otherwise, all significance and refinement passes are arithmetically coded. The lazy mode allows the computational complexity of bit-plane coding to be significantly reduced, by decreasing the number of symbols that must be arithmetically coded. This comes, of course, at the cost of reduced coding efficiency.

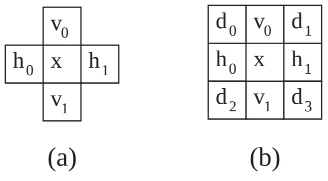


Fig. 9. Templates for context selection. The (a) 4-connected and (b) 8-connected neighbors.

As indicated above, coding pass data can be encoded using one of two schemes (i.e., arithmetic or raw coding). Consecutive coding passes that employ the same encoding scheme constitute what is known as a segment. All of the coding passes in a segment can collectively form a single codeword or each coding pass can form a separate codeword. Which of these is the case is determined by the termination mode in effect. Two termination modes are supported: per-pass termination and per-segment termination. In the first case, only the last coding pass of a segment is terminated. In the second case, all coding passes are terminated. Terminating all coding passes facilitates improved error resilience at the expense of decreased coding efficiency.

Since context-based arithmetic coding is employed, a means for context selection is necessary. Generally speaking, context selection is performed by examining state information for the 4-connected or 8-connected neighbors of a sample as shown in Fig. 9.

In our explanation of the coding passes that follows, we focus on the encoder side as this facilitates easier understanding. The decoder algorithms follow directly from those employed on the encoder side.

J.1 Significance Pass

The first coding pass for each bit plane is the significance pass. This pass is used to convey significance and (as necessary) sign information for samples that have not yet been found to be significant and are predicted to become significant during the processing of the current bit plane. The samples in the code block are scanned in the order shown previously in Fig. 10. If a sample has not yet been found to be significant, and is predicted to become significant, the significance of the sample is coded with a single binary symbol. If the sample also happens to be significant, its sign is coded using a single binary symbol. In pseudocode form, the significance pass is described by Algorithm 1.

Algorithm 1 Significance pass algorithm.

```
1: for each sample in code block do
2:     if sample previously insignificant and predicted to become significant
       during current bit plane then
3:         code significance of sample /* 1 binary symbol */
4:         if sample significant then
5:             code sign of sample /* 1 binary symbol */
6:         endif
7:     endif
8: endfor
```

If the most significant bit plane is being processed, all samples are predicted to remain insignificant. Otherwise, a sample is predicted to become significant if any 8-connected neighbor has already been found to be significant. As a consequence of this prediction policy, the significance and refinement passes for the most significant bit plane are always empty (and need not be explicitly coded).

The symbols generated during the significance pass may or may not be arithmetically coded. If arithmetic coding is employed, the binary symbol conveying significance information is coded using one of nine contexts. The particular context used is selected based on the significance of the sample's 8-connected neighbors and the orientation of the subband with which the sample is associated (e.g., LL, LH, HL, HH). In the case that arithmetic coding is used, the sign of a sample is coded as the difference between the actual and predicted sign. Otherwise, the sign is coded directly. Sign prediction is performed using the significance and sign information for 4-connected neighbors.

J.2 Refinement Pass

The second coding pass for each bit plane is the refinement pass. This pass signals subsequent bits after the most significant bit for each sample. The samples of the code block are scanned using the order shown earlier in Fig. 10. If a sample was found to be significant in a previous bit plane, the next most significant bit of that sample is conveyed using a single binary symbol. This process is described in pseudocode form by Algorithm 2.

Algorithm 2 Refinement pass algorithm.

```
1: for each sample in code block do
2:   if sample found significant in previous bit plane then
3:     code next most significant bit in sample /* 1 binary symbol */
4:   endif
5: endfor
```

Like the significance pass, the symbols of the refinement pass may or may not be arithmetically coded. If arithmetic coding is employed, each refinement symbol is coded using one of three contexts. The particular context employed is selected based on if the second MSB position is being refined and the significance of 8-connected neighbors.

J.3 Cleanup Pass

The third (and final) coding pass for each bit plane is the cleanup pass. This pass is used to convey significance and (as necessary) sign information for those samples that have not yet been found to be significant and are predicted to remain insignificant during the processing of the current bit plane.

Conceptually, the cleanup pass is not much different from the significance pass. The key difference is that the cleanup pass conveys information about samples that are predicted to remain insignificant, rather than those that are predicted to become significant. Algorithmically, however, there is one important difference between the cleanup and significance passes. In the case of the cleanup pass, samples are sometimes processed in groups, rather than individually as with the significance pass.

Recall the scan pattern for samples in a code block, shown earlier in Fig. 10. A code block is partitioned into stripes with a nominal height of four samples. Then, stripes are scanned

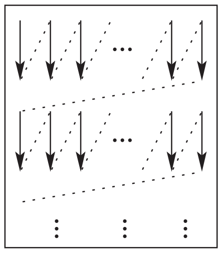


Fig. 10. Sample scan order within a code block.

from top to bottom, and the columns within a stripe are scanned from left to right. For convenience, we will refer to each column within a stripe as a vertical scan. That is, each vertical arrow in the diagram corresponds to a so called vertical scan. As will soon become evident, the cleanup pass is best explained as operating on vertical scans (and not simply individual samples).

The cleanup pass simply processes each of the vertical scans in order, with each vertical scan being processed as follows. If the vertical scan contains four samples (i.e., is a full scan), significance information is needed for all of these samples, and all of the samples are predicted to remain insignificant, a special mode, called aggregation mode, is entered. In this mode, the number of leading insignificant samples in the vertical scan is coded. Then, the samples whose significance information is conveyed by aggregation are skipped, and processing continues with the remaining samples of the vertical scan exactly as is done in the significance pass. In pseudocode form, this process is described by Algorithm 3.

Algorithm 3 Cleanup pass algorithm.

```
1: for each vertical scan in code block do
2:   if four samples in vertical scan and all previously insignificant and un-
     visited and none have significant 8-connected neighbor then
3:     code number of leading insignificant samples via aggregation
4:     skip over any samples indicated as insignificant by aggregation
5:   endif
6:   while more samples to process in vertical scan do
7:     if sample previously insignificant and unvisited then
8:       code significance of sample if not already implied by run /* 1 binary
         symbol */
9:       if sample significant then
10:        code sign of sample /* 1 binary symbol */
11:      endif
12:    endif
13:  endwhile
14: endfor
```

When aggregation mode is entered, the four samples of the vertical scan are examined. If all four samples are insignificant, an all-insignificant aggregation symbol is coded, and processing of the vertical scan is complete. Otherwise, a some-significant aggregation symbol is coded, and two binary symbols are then used to code the number of leading insignificant samples in the vertical scan.

The symbols generated during the cleanup pass are always arithmetically coded. In the aggregation mode, the aggregation symbol is coded using a single context, and the two symbol run length is coded using a single context with a fixed uniform probability distribution. When aggregation mode is not employed, significance and sign coding function just as in the case of the significance pass.

K. Tier-2 Coding

In the encoder, tier-1 encoding is followed by tier-2 encoding. The input to the tier-2 encoding process is the set of bit-plane coding passes generated during tier-1 encoding. In tier-2 encoding, the coding pass information is packaged into data units called packets, in a process referred to as packetization. The resulting packets are then output to the final code stream. The packetization process imposes a particular organization on coding pass data in the output code stream. This organization facilitates many of the desired codec features including rate scalability and progressive recovery by fidelity or resolution.

A packet is nothing more than a collection of coding pass data. Each packet is comprised of two parts: a header and body. The header indicates which coding passes are included in the packet, while the body contains the actual coding pass data itself. In the code stream, the header and body may appear together or separately, depending on the coding options in effect.

Rate scalability is achieved through (quality) layers. The coded data for each tile is organized into L layers, numbered from 0 to $L-1$, where $L \geq 1$. Each coding pass is either assigned to one of the L layers or discarded. The coding passes containing the most important data are included in the lower layers, while the coding passes associated with finer details are included in higher layers. During decoding, the reconstructed image quality improves incrementally with each successive layer processed. In the case of lossy compression, some coding passes may be discarded (i.e., not included in any layer) in which case rate control must decide which passes to include in the final code stream. In the lossless case, all coding passes must be included. If multiple layers are employed (i.e., $L > 1$), rate control must decide in which layer each coding pass is to be included. Since some coding passes may be discarded, tier-2 coding is the second primary source of information loss in the coding path.

Recall, from Section III-I, that each coding pass is associated with a particular component, resolution level, subband, and code block. In tier-2 coding, one packet is generated for each component, resolution level, layer, and precinct 4-tuple. A packet need not contain any coding pass data at all. That is, a packet can be empty. Empty packets are sometimes necessary since a packet must be generated for every component-resolution-layer-precinct combination even if the resulting packet conveys no new information.

As mentioned briefly in Section III-G, a precinct is essentially a grouping of code blocks within a subband. The precinct partitioning for a particular subband is derived from a partitioning of its parent LL band (i.e., the LL band at the next higher resolution level). Each resolution level has a nominal precinct size. The nominal width and height of a precinct must be a power of two, subject to certain constraints (e.g., the maximum width and height are both 2^{15}). The LL band associated with each resolution level is divided into precincts. This is accomplished by overlaying the LL band with a regular grid having horizontal and vertical spacings of 2^{PPx} and 2^{PPy}, respectively, as shown in Fig. 11, where the grid is aligned with the origin of the LL band's coordinate system. The precincts bordering on the edge of the subband may have dimensions smaller than

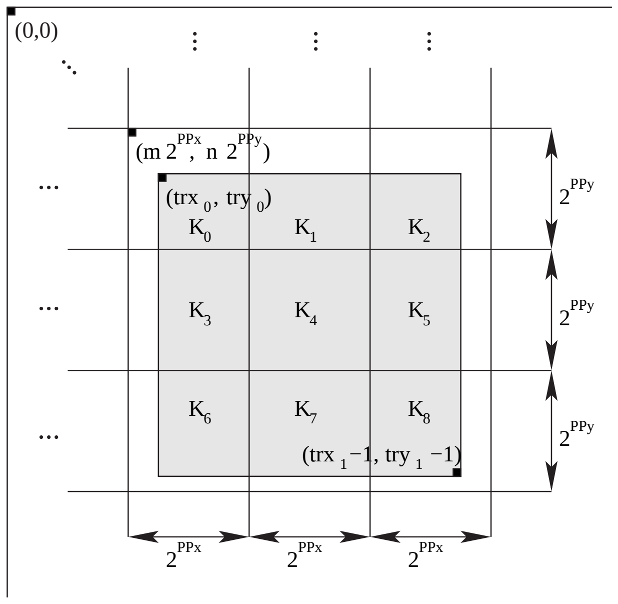


Fig. 11. Partitioning of resolution into precincts.

the nominal size. Each of the resulting precinct regions is then mapped into its child subbands (if any) at the next lower resolution level. This is accomplished by using the coordinate transformation $(u, v) = (\lceil x/2 \rceil, \lceil y/2 \rceil)$ where (x, y) and (u, v) are the coordinates of a point in the LL band and child subband, respectively. Due to the manner in which the precinct partitioning is performed, precinct boundaries always align with code block boundaries. Some precincts may also be empty. Suppose the nominal code block size is $2^{\text{xcb'}} \times 2^{\text{ycb'}}$. This results nominally in $2^{\text{PPx'}-\text{xcb'}} \times 2^{\text{PPy'}-\text{ycb'}}$ groups of code blocks in a precinct, where

$$\text{PPx'} = \begin{cases} \text{PPx} & \text{for } r = 0 \\ \text{PPx} - 1 & \text{for } r > 0 \end{cases} \tag{12}$$

$$\text{PPy'} = \begin{cases} \text{PPy} & \text{for } r = 0 \\ \text{PPy} - 1 & \text{for } r > 0 \end{cases} \tag{13}$$

and r is the resolution level.

Since coding pass data from different precincts are coded in separate packets, using smaller precincts reduces the amount of data contained in each packet. If less data is contained in a packet, a bit error is likely to result in less information loss (since, to some extent, bit errors in one packet do not affect the decoding of other packets). Thus, using a smaller precinct size leads to improved error resilience, while coding efficiency is degraded due to the increased overhead of having a larger number of packets.

More than one ordering of packets in the code stream is supported. Such orderings are called progressions. There are five builtin progressions defined: 1) layer-resolution-component-position ordering, 2) resolution-layer-component-position ordering, 3) resolution-position-component-layer ordering, 4) position-component-resolution-layer ordering, and 5) component-position-resolution-layer ordering. The sort order for the packets is given by the name of the ordering, where position refers to precinct number, and the sorting keys are listed from most significant to least significant. For example, in the case of the first ordering given above, packets are ordered first by layer, second by resolution, third by component, and last by precinct. This corresponds to a progressive recovery by fidelity scenario. The second ordering above is associated with progressive recovery by resolution. The three remaining orderings are somewhat more esoteric. It is also possible to specify additional user-defined progressions at the expense of increased coding overhead.

In the simplest scenario, all of the packets from a particular tile appear together in the code stream. Provisions exist, however, for interleaving packets from different tiles, allowing further flexibility on the ordering of data. If, for example, progressive recovery of a tiled image was desired, one would probably include all of the packets associated with the first layer of the various tiles, followed by those packets associated with the second layer, and so on.

In the decoder, the tier-2 decoding process extracts the various coding passes from the code stream (i.e., depacketization) and associates each coding pass with its corresponding code block. In the lossy case, not all of the coding passes are guaranteed to be present since some may have been discarded by the encoder. In the lossless case, all of the coding passes must be present in the code stream.

In the sections that follow, we describe the packet coding process in more detail. For ease of understanding, we choose to explain this process from the point of view of the encoder. The decoder algorithms, however, can be trivially deduced from those of the encoder.

K.1 Packet Header Coding

The packet header corresponding to a particular component, resolution level, layer, and precinct, is encoded as follows. First, a single binary symbol is encoded to indicate if any coding pass data is included in the packet (i.e., if the packet is non-empty). If the packet is empty, no further processing is required and the algorithm terminates. Otherwise, we proceed to examine each subband in the resolution level in a fixed order. For each subband, we visit the code blocks belonging to the precinct of interest in raster scan order as shown in Fig. 12. To process a single code block, we begin by determining if any new coding pass data is to be included. If no coding pass data has yet been included for this code block, the inclusion information is conveyed via a quadtree-based coding procedure. Otherwise, a binary symbol is emitted indicating the presence or absence of new coding pass data for the code block. If no new coding passes are included, we proceed to the processing of the next code block in the precinct. Assuming that new coding pass data are to be included, we continue with our processing of the current code block. If this is the first time coding pass data have been included for the code block, we encode the number of leading insignificant bit planes for the code block using a quadtree-based coding algorithm. Then, the number of new coding passes, and the length of the data associated with these passes is encoded. A bit stuffing algorithm is applied to all packet header data to ensure that certain bit patterns never occur in the output, allowing such patterns to be used for error resilience purposes. The entire

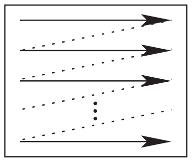


Fig. 12. Code block scan order with a precinct.

packet header coding process is summarized by Algorithm 4.

Algorithm 4 Packet header coding algorithm.

```
1:  if packet not empty then
2:      code non-empty packet indicator /* 1 binary symbol */
3:      for each subband in resolution level do
4:          for each code block in subband precinct do
5:              code inclusion information /* 1 binary symbol or tag tree */
6:              if no new coding passes included then
7:                  skip to next code block
8:              endif
9:              if first inclusion of code block then
10:                 code number of leading insignificant bit planes /* tag tree */
11:             endif
12:             code number of new coding passes
13:             code length increment indicator
14:             code length of coding pass data
15:         endfor
16:     endfor
17: else
18:     code empty packet indicator /* 1 binary symbol */
19: endif
20: pad to byte boundary
```

K.2 Packet Body Coding

The algorithm used to encode the packet body is relatively simple. The code blocks are examined in the same order as in the case of the packet header. If any new passes were specified in the corresponding packet header, the data for these coding passes are concatenated to the packet body. This process is summarized by Algorithm 5.

Algorithm 5 Packet body coding algorithm.

```
1: for each subband in resolution level do
2:     for each code block in subband precinct do
3:         if (new coding passes included for code block) then
4:             output coding pass data
5:         endif
6:     endfor
7: endfor
```

L. Rate Control

In the encoder, rate control can be achieved through two distinct mechanisms: 1) the choice of quantizer step sizes, and 2) the selection of the subset of coding passes to include in the code stream. When the integer coding mode is used (i.e., when only integer-to-integer transforms are employed) only the first mechanism may be used, since the quantizer step sizes must be fixed at one. When the real coding mode is used, then either or both of these rate control mechanisms may be employed.

When the first mechanism is employed, quantizer step sizes are adjusted in order to control rate. As the step sizes are in-

Type	Length (if required)	Parameters (if required)
16 bits	16 bits	variable lengt

Fig. 13. Marker segment structure.

creased, the rate decreases, at the cost of greater distortion. Although this rate control mechanism is conceptually simple, it does have one potential drawback. Every time the quantizer step sizes are changed, the quantizer indices change, and tier-1 encoding must be performed again. Since tier-1 coding requires a considerable amount of computation, this approach to rate control may not be practical in computationally-constrained encoders.

When the second mechanism is used, the encoder can elect to discard coding passes in order to control the rate. The encoder knows the contribution that each coding pass makes to rate, and can also calculate the distortion reduction associated with each coding pass. Using this information, the encoder can then include the coding passes in order of decreasing distortion reduction per unit rate until the bit budget has been exhausted. This approach is very flexible in that different distortion metrics can be easily accommodated (e.g., mean squared error, visually weighted mean squared error, etc.).

For a more detailed treatment of rate control, the reader is referred to [4] and [11].

M. Region of Interest Coding

The codec allows different regions of an image to be coded with differing fidelity. This feature is known as region-of-interest (ROI) coding. In order to support ROI coding, a very simple yet flexible technique is employed as described below.

When an image is synthesized from its transform coefficients, each coefficient contributes only to a specific region in the reconstruction. Thus, one way to code a ROI with greater fidelity than the rest of the image would be to identify the coefficients contributing to the ROI, and then to encode some or all of these coefficients with greater precision than the others. This is, in fact, the basic premise behind the ROI coding technique employed in the JPEG-2000 codec.

When an image is to be coded with a ROI, some of the transform coefficients are identified as being more important than the others. The coefficients of greater importance are referred to as ROI coefficients, while the remaining coefficients are known as background coefficients. Noting that there is a one-to-one correspondence between transform coefficients and quantizer indices, we further define the quantizer indices for the ROI and background coefficients as the ROI and background quantizer indices, respectively. With this terminology introduced, we are now in a position to describe how ROI coding fits into the rest of the coding framework.

The ROI coding functionality affects the tier-1 coding process. In the encoder, before the quantizer indices for the various subbands are bit-plane coded, the ROI quantizer indices are scaled upwards by a power of two (i.e., by a left bit shift). This scaling is performed in such a way as to ensure that all bits of the ROI quantizer indices lie in more significant bit planes than the potentially nonzero bits of the background quantizer indices. As a consequence, all information about ROI quantizer indices will be signalled before information about background ROI indices. In this way, the ROI can be reconstructed at a higher fidelity than the background.

Before the quantizer indices are bit-plane coded, the encoder examines the background quantizer indices for all of the subbands looking for the index with the largest magnitude. Suppose that this index has its most significant bit in bit position $N-1$. All of the ROI indices are then shifted N bits to the left, and bit-plane coding proceeds as in the non-ROI case. The ROI shift value N is included in the code stream.

During decoding, any quantizer index with nonzero bits lying in bit plane N or above can be deduced to belong to the ROI set. After the reconstructed quantizer indices are obtained from the bit-plane decoding process, all indices in the ROI set are then scaled down by a right shift of N bits. This undoes the effect of the scaling on the encoder side.

The ROI set can be chosen to correspond to transform coefficients affecting a particular region in an image or subset of those affecting the region. This ROI coding technique has a number of desirable properties. First, the ROI can have any arbitrary shape and be disjoint. Second, there is no need to explicitly signal the ROI set, since it can be deduced by the decoder from the ROI shift value and the magnitude of the quantizer indices.

N. Code Stream

In order to specify the coded representation of an image, two different levels of syntax are employed by the codec. The lowest level syntax is associated with what is referred to as the code stream. The code stream is essentially a sequence of tagged records and their accompanying data.

The basic building block of the code stream is the marker segment. As shown in Fig. 13, a marker segment is comprised of three fields: the type, length, and parameters fields. The type (or marker) field identifies the particular kind of marker segment. The length field specifies the number of bytes in the marker segment. The parameters field provides additional information specific to the marker type. Not all types of marker segments have length and parameters fields. The presence (or absence) of these fields is determined by the marker segment type. Each type of marker segment signals its own particular class of information.

A code stream is simply a sequence of marker segments and auxiliary data (i.e., packet data) organized as shown in Fig. 14. The code stream consists of a main header, followed by tile-part header and body pairs, followed by a main trailer. A list of some of the more important marker segments is given in Table II. Parameters specified in marker segments in the main header serve as defaults for the entire code stream. These default settings, however, may be overridden for a particular tile by specifying new values in a marker segment in the tile's header.

All marker segments, packet headers, and packet bodies are a multiple of 8 bits in length. As a consequence, all markers are byte-aligned, and the code stream itself is always an integral number of bytes.

TABLE II
TYPES OF MARKER SEGMENTS

Type	Description
Start of codestream (SOC)	Signals the start of a code stream. Always the first marker segment in the code stream (i.e., the first marker segment in the main header).
End of codestream (EOC)	Signals the end of the code stream. Always the last marker segment in the code stream.
Start of tile-part (SOT)	Indicates the start of a tile-part header. Always the first marker segment in a tile-part header.
Start of data (SOD)	Signal the end of the tile-part header. Always the last marker segment in the tile-part header. The tile body follows immediately after this marker segment.
Image and tile size (SIZ)	Conveys basic image characteristics (e.g., image size, number of components, precision of sample values), and tiling parameters. Always the second marker segment in the code stream.
Coding style default (COD)	Specifies coding parameters (e.g., multicomponent transform, wavelet/subband transform, tier-1/tier-2 coding parameters, etc.).
Coding style component (COC)	Specifies a subset of coding parameters for a single component.
Quantization default (QCD)	Specifies quantization parameters (i.e., quantizer type, quantizer parameters).
Quantization component (QCC)	Specifies quantization parameters for a single component.
Region of interest (RGN)	Specifies region-of-interest coding parameters.

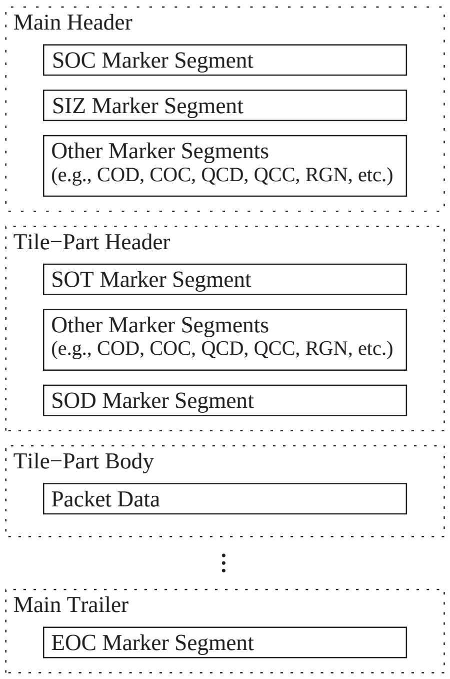


Fig. 14. Code stream structure.

O. File Format

A code stream provides only the most basic information required to decode an image (i.e., sufficient information to deduce the sample values of the decoded image). While in some simple applications this information is sufficient, in other applications additional data is required. To display a decoded image, for example, it is often necessary to know additional characteristics of an image, such as the color space of the image data and opacity attributes. Also, in some situations, it is beneficial to know other information about an image (e.g., ownership, origin, etc.) In order to allow the above types of data to be specified, an additional level of syntax is employed by the codec. This level of syntax is referred to as the file format. The file format is used to convey both coded image data and auxiliary information about the image. Although this file format is optional, it undoubtedly will be used extensively by many applications, particularly computer-based software applications.

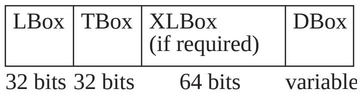


Fig. 15. Box structure.

The basic building block of the file format is referred to as a box. As shown in Fig. 15, a box is nominally comprised of four fields: the LBox, TBox, XLBox, and DBox fields. The LBox field specifies the length of the box in bytes. The TBox field indicates the type of box (i.e., the nature of the information contained in the box). The XLBox field is an extended length indicator which provides a mechanism for specifying the length of a box whose size is too large to be encoded in the length field alone. If the LBox field is 1, then the XLBox field is present and contains the true length of the box. Otherwise, the XLBox field is not present. The DBox field contains data specific to the particular box type. Some types of boxes may contain other boxes as data. As a matter of terminology, a box that contains other boxes in its DBox field is referred to as a superbox. Several of the more important types of boxes are listed in Table III.

A file is a sequence of boxes. Since certain types of boxes are defined to contain others, there is a natural hierarchical structure to a file. The general structure of a file is shown in Fig. 16. The JPEG-2000 signature box is always first, providing an indication that the byte stream is, in fact, correctly formatted. The file type box is always second, indicating the version of the file format to which the byte stream conforms. Although some constraints exist on the ordering of the remaining boxes, some flexibility is also permitted. The header box simply contains a number of other boxes. The image header box specifies several basic characteristics of the image (including image size, number of components, etc.). The bits per component box indicates the precision and signedness of the component samples. The color specification box identifies the color space of image data (for display purposes) and indicates which components map to which type of spectral information (i.e., the correspondence between components and color/opacity planes). Every file must contain at least one contiguous code stream box. (Multiple contiguous code stream boxes are permitted in order to facilitate the

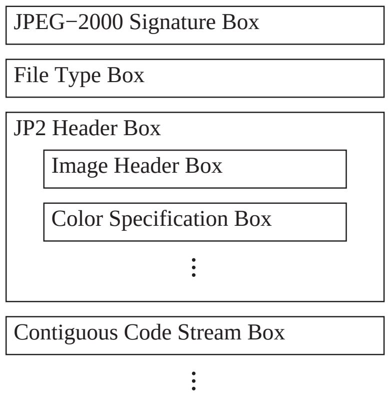


Fig. 16. File format structure.

specification of image sequences to support trivial animation effects.) Each contiguous code stream box contains a code stream as data. In this way, coded image data is embedded into a file. In addition to the types of boxes discussed so far, there are also box types for specifying the capture and display resolution for an image, palette information, intellectual property information, and vendor/application-specific data.

Although some of the information stored at the file format level is redundant (i.e., it is also specified at the code stream level), this redundancy allows trivial manipulation of files without any knowledge of the code stream syntax. The file name extension "jp2" is to be used to identify files containing data in the JP2 file format.

P. Extensions

Although the baseline codec is quite versatile, there may be some applications that could benefit from additional features not present in the baseline system. To this end, Part 2 of the standard [27] is to define numerous extensions to the baseline codec. Some of the extensions likely to be defined include the following: 1) independent regions, a generalization of tiling whereby an image may be partitioned into arbitrarily-shaped and possibly overlapped regions; 2) additional intercomponent transforms (e.g., multidimensional wavelet/subband transforms); 3) additional intracomponent transforms (e.g., subband transforms based on arbitrary filters and decomposition trees, different filters in the horizontal and vertical directions); 4) overlapped transforms; 5) additional quantization methods such as trellis coded quantization [28]; 6) enhanced ROI support (e.g., a mechanism for explicitly signalling the shape of the ROI and an arbitrary shift value); and 7) extensions to the file format including support for additional color spaces and compound documents.

IV. Conclusions

In this paper, we commenced with a high-level introduction to the JPEG-2000 standard, and proceeded to study the JPEG-2000 codec in detail. With its excellent coding performance and many attractive features, JPEG-2000 will no doubt become a widely used standard in the years to come.

Appendix

I. JasPer

JasPer is a collection of software (i.e., a library and application programs) for the coding and manipulation of images. This software is written in the C programming language. Of particular interest here, the JasPer software provides an implementation of the JPEG-2000 Part-1 codec. The JasPer software was developed with the objective of providing a free JPEG-2000 codec implementation for anyone wishing to use the JPEG-2000 standard. This software has also been submitted to the ISO for use as a reference implementation in Part 5 of the JPEG-2000 standard.

The JasPer software is available for download from the JasPer Project home page (i.e., `http://www.ece.ubc.ca/~mdadams/jasper`). In the near future, the JasPer software should also become available from the web site of Image Power, Inc. (i.e., `http://www.imagepower.com`) and the JPEG web site (i.e., `http://www.jpeg.org/software`). For more information about JasPer, the reader is referred to [5–7].

Acknowledgment

The author would like to thank Image Power, Inc. for supporting the development of the JasPer software. Also, the author would like to express his gratitude to Dr. Faouzi Kossentini for past collaborations that led to the author's involvement with JPEG-2000 standardization activities (hence, making a paper like this possible).

References

[1] ISO/IEC, *ISO/IEC 10918-1:1994, Information technology - Digital compression and coding of continuous-tone still images: Requirements and guidelines*, 1994.
[2] G. K. Wallace, "The JPEG still picture compression standard," *Communications of the ACM*, vol. 34, no. 4, pp. 30–44, Apr. 1991.
[3] ISO/IEC, *ISO/IEC 14495-1, Lossless and near-lossless compression of continuous-tone still images - baseline*, 2000.
[4] ISO/IEC, *ISO/IEC 15444-1: Information technology—JPEG 2000 image coding system—Part 1: Core coding system*, 2000.
[5] M. D. Adams, "JasPer project home page," `http://www.ece.ubc.ca/~mdadams/jasper`, 2000.
[6] M. D. Adams and F. Kossentini, "JasPer: A software-based JPEG-2000 codec implementation," in *Proc. of IEEE International Conference on Image Processing*, Vancouver, BC, Canada, Oct. 2000.
[7] M. D. Adams, "JasPer software reference manual," Documentation distributed with the JasPer software, 2001.
[8] "ISO/IEC call for contributions, JPEG 2000," ISO/IEC JTC 1/SC 29/WG 1 N505, Mar. 1997.
[9] A. S. Lewis and G. Knowles, "Image compression using the 2-D wavelet transform," *IEEE Trans. on Image Processing*, vol. 1, no. 2, pp. 244–250, Apr. 1992.
[10] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Trans. on Image Processing*, vol. 1, no. 2, pp. 205–220, Apr. 1992.
[11] D. Taubman, "High performance scalable image compression with ebcot," in *Proc. of IEEE International Conference on Image Processing*, Kobe, Japan, 1999, vol. 3, pp. 344–348.
[12] D. Taubman, "High performance scalable image compression with EBCOT," *IEEE Trans. on Image Processing*, vol. 9, no. 7, pp. 1158–1170, July 2000.
[13] A. R. Calderbank, I. Daubechies, W. Sweldens, and B.-L. Yeo, "Wavelet transforms that map integers to integers," *Applied and Computational Harmonic Analysis*, vol. 5, no. 3, pp. 332–369, July 1998.
[14] M. J. Gormish, E. L. Schwartz, A. F. Keith, M. P. Boliek, and A. Zandi, "Lossless and nearly lossless compression of high-quality images," in *Proc. of SPIE*, San Jose, CA, USA, Mar. 1997, vol. 3025, pp. 62–70.

TABLE III
BOX TYPES

Type	Description
JPEG-2000 Signature	Identifies the file as being in the JP2 format. Always the first box in a JP2 file.
File Type	Specifies the version of the format to which the file conforms. Always the second box in a JP2 file.
JP2 Header	Specifies information about the image aside from the coded image data itself. (A superbox.)
Image Header	Specifies the size and other basic characteristics of the image.
Color Specification	Specifies the colorspace to which the image sample data belongs.
Contiguous Code Stream	Contains a code stream.

[15] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[16] ISO/IEC, *ISO/IEC 14492-1, Lossy/lossless coding of bi-level images*, 2000.

[17] H. Chao, P. Fisher, and Z. Hua, "An approach to integer wavelet transforms for lossless for image compression," in *Proc. of International Symposium on Computational Mathematics*, Guangzhou, China, Aug. 1997, pp. 19–38.

[18] M. D. Adams, "Reversible wavelet transforms and their application to embedded image compression," M.A.Sc. thesis, Department of Electrical and Computer Engineering, University of Victoria, Victoria, BC, Canada, Jan. 1998, Available from `http://www.ece.ubc.ca/~mdadams`.

[19] M. D. Adams and F. Kossentini, "Reversible integer-to-integer wavelet transforms for image compression: Performance evaluation and analysis," *IEEE Trans. on Image Processing*, vol. 9, no. 6, pp. 1010–1024, June 2000.

[20] W. Sweldens, "The lifting scheme: A custom-design construction of biorthogonal wavelets," *Applied and Computational Harmonic Analysis*, vol. 3, no. 2, pp. 186–200, 1996.

[21] W. Sweldens, "The lifting scheme: A construction of second generation wavelets," *SIAM Journal of Mathematical Analysis*, vol. 29, no. 2, pp. 511–546, Mar. 1998.

[22] C. M. Brislawn, "Preservation of subband symmetry in multirate signal coding," *IEEE Trans. on Signal Processing*, vol. 43, no. 12, pp. 3046–3050, Dec. 1995.

[23] D. Le Gall and A. Tabatabai, "Sub-band coding of digital images using symmetric short kernel filters and arithmetic coding techniques," in *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing*, New York, NY, USA, Apr. 1988, vol. 2, pp. 761–764.

[24] C. M. Brislawn, J. N. Bradley, R. J. Onyshczak, and T. Hopper, "The FBI compression standard for digitized fingerprint images," Preprint, 1996, Available from `ftp://ftp.c3.lanl.gov/pub/WSQ/documents/spie96.ps.Z`.

[25] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. on Signal Processing*, vol. 41, no. 12, pp. 3445–3462, Dec. 1993.

[26] A. Said and W. A. Pearlman, "A new fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 243–250, June 1996.

[27] ISO/IEC, *ISO/IEC 15444-2, Information technology - JPEG 2000 extensions*, 2000.

[28] J. H. Kasner, M. W. Marcellin, and B. R. Hunt, "Universal trellis coded quantization," *IEEE Trans. on Image Processing*, vol. 8, no. 12, pp. 1677–1687, Dec. 1999.

Michael D. Adams was born in Kitchener, ON, Canada in 1969. He received the B.A.Sc. degree in computer engineering from the University of Waterloo, Waterloo, ON, Canada in 1993, and the M.A.Sc. degree in electrical engineering from the University of Victoria, Victoria, BC, Canada in 1998. He is currently pursuing the Ph.D. degree in electrical engineering at the University of British Columbia, Vancouver, BC, Canada. From 1993 to 1995, Michael was a member of technical staff at Bell-Northern Research (now Nortel Networks) in Ottawa, ON, Canada where he developed real-time software for fiber-optic telecommunication systems.

Michael is the recipient of a Natural Sciences and Engineering Research Council (of Canada) Postgraduate Scholarship. He is a voting member of the Canadian Delegation to ISO/IEC JTC 1/SC 29, a member of ISO/IEC JTC 1/SC 29/WG 1 (i.e., the JPEG/JBIG working group), and has been an active participant in the JPEG-2000 standardization effort, serving as a coeditor of the JPEG-2000 Part-5 standard and principal author of one of the first JPEG-2000 implementations (i.e., JasPer). His research interests include digital signal processing, wavelets, multirate systems, image coding, and multimedia systems.