Modern Signal Processing
MSRI Publications
Volume **46**, 2003

Image Compression: The Mathematics of JPEG 2000

JIN LI

Abstract. We briefly review the mathematics in the coding engine of JPEG 2000, a state-of-the-art image compression system. We focus in depth on the transform, entropy coding and bitstream assembler modules. Our goal is to present a general overview of the mathematics underlying a state of the art scalable image compression technology.

1. Introduction

Data compression is a process that creates a compact data representation from a raw data source, usually with an end goal of facilitating storage or transmission. Broadly speaking, compression takes two forms, either *lossless* or *lossy*, depending on whether or not it is possible to reconstruct exactly the original datastream from its compressed version. For example, a data stream that consists of long runs of 0s and 1s (such as that generated by a black and white fax) would possibly benefit from simple *run-length encoding*, a lossless technique replacing the original datastream by a sequence of counts of the lengths of the alternating substrings of 0s and 1s. Lossless compression is necessary for situations in which changing a single bit can have catastrophic effects, such as in machine code of a computer program.

While it might seem as though we should always demand lossless compression, there are in fact many venues where exact reproduction is unnecessary. In particular, media compression, which we define to be the compression of image, audio, or video files, presents an excellent opportunity for lossy techniques. For example, not one among us would be able to distinguish between two images which differ in only one of the 2^{29} bits in a typical 1024×1024 color image. Thus distortion is tolerable in media compression, and it is the content, rather than

Keywords: Image compression, JPEG 2000, transform, wavelet, entropy coder, subbitplane entropy coder, bitstream assembler.

the exact bits, that is of paramount importance. Moroever, the size of the original media is usually very large, so that it is essential to achieve a considerably high *compression ratio* (defined to be the ratio of the size of the original data file to the size of its compressed version). This is achieved by taking advantage of *psychophysics* (say by ignoring less perceptible details of the media) and by the use of *entropy coding*, the exploitation of various information redundancies that may exist in the source data.

Conventional media compression solutions focus on a static or one-time form of compression — i.e., the compressed bitstream provides a static representation of the source data that makes possible a unique reconstruction of the source, whose characteristics are quantified by a compression ratio determined at the time of encoding. Implicit in this approach is the notion of a "one shoe fits all" technique, an outcome that would appear to be variance with the multiplicity of reconstruction platforms upon which the media will ultimately reside. Often, different applications may have different requirements for the compression ratio as well as tolerating various levels of compression distortion. A publishing application may require a compression scheme with very little distortion, while a web application may tolerate relatively large distortion in exchange for smaller compressed media.

Recently *scalable compression* has emerged as a category of media compression algorithms capable of trading between compression ratio and distortion *after* generating an initially compressed *master bitstream*. Subsets of the master then may be extracted to form particular *application bitstreams* which may exhibit a variety of compression ratios. (I.e., working from the master bitstream we can achieve a range of compressions, with the concomitant ability to reconstruct coarse to fine scale characteristics.) With scalable compression, compressed media can be tailored effortlessly for applications with vastly different compression ratio and quality requirements, a property which is particularly valuable in media storage and transmission.

In what follows, we restrict our attention to image compression, in particular, focusing on the *JPEG 2000 image compression standard*, and thereby illustrate the mathematical underpinnings of a modern scalable media compression algorithm. The paper is organized as follows. The basic concepts of the scalable image compression and its applications are discussed in Section 2. JPEG 2000 and its development history are briefly reviewed in Section 3. The transform, quantization, entropy coding, and bitstream assembler modules are examined in detail in Sections 4 to 7. Readers interested in further details may refer to [1; 2; 3].

2. Image Compression

Digital images are used every day. A digital image is essentially a 2D data array $x(i, j)$, where i and $j$index the row and column of the data array, and

$x(i, j)$is referred to as a pixel. Gray-scale images assign to each pixel a single scalar intensity value G, whereas color images traditionally assign to each pixel a *color vector* (R, G, B), which represent the intensity of the red, green, and blue components, respectively. Because it is the content of the digital image that matters, the underlying 2D data array may undergo big changes while still conveying the content to the user with little or no perceptible distortion. An example is shown in Figure 1. On the left the classic image processing test case *Lena* is shown as a 512×512 grey-scale image. To the right of the original are several applications, each showing different sorts of compression. The first application illustrates the use of *subsampling* in order to fit a smaller image (in this case 256×256). The second application uses JPEG (the predecessor to JPEG 2000) to compress the image to a bitstream, and then decode the bitstream back to an image of size 512×512. Although in each case the underlying 2D data array is changed tremendously, the primary content of the image remains intelligible.

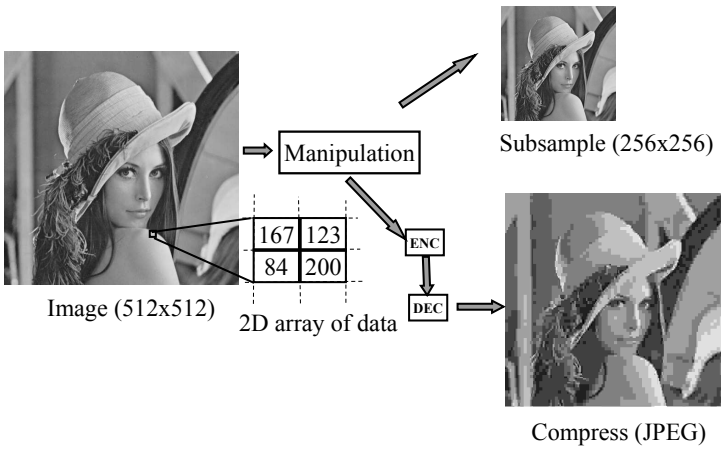

Figure 1. Souce digital image and compressions.

Each of the applications above results in a reduction in the amount of source image data. In this paper, we focus our attention on JPEG 2000, which is a next generation image compression standard. JPEG 2000 distinguishes itself from older generations of compression standards not only by virtue of its higher compression ratios, but also by its many new functionalities. The most noticeable among them is its scalability. From a compressed JPEG 2000 bitstream, it is possible to extract a subset of the bitstream that decodes to an image of variable quality and resolution (inversely correlated with its accompanying compression ratio), and/or variable spatial locality.

Scalable image compression has important applications in image storage and delivery. Consider the application of digital photography. Presently, digital

cameras all use non-scalable image compression technologies, mainly JPEG. A camera with a fixed amount of the memory can accommodate a small number of high quality, high-resolution images, or a large number of low quality, low-resolution images. Unfortunately, the image quality and resolution must be determined before shooting the photos. This leads to the often painful trade-off between removing old photos to make space for new exciting shots, and shooting new photos of poorer quality and resolution. Scalable image compression makes possible the adjustment of image quality and resolution *after* the photo is shot, so that instead, the original digital photos always can be shot at the highest possible quality and resolution, and when the camera memory is filled to capacity, the compressed bitstream of existing shots may be truncated to smaller size to leave room for the upcoming shots. This need not be accomplished in a uniform fashion, with some photos kept with reduced resolution and quality, while others retain high resolution and quality. By dynamically trading between the number of images and the image quality, the use of precious camera memory is apportioned wisely.

Web browsing provides another important application of scalable image compression. As the resolution of digital cameras and digital scanners continues to increase, high-resolution digital imagery becomes a reality. While it is a pleasure to view a high-resolution image, for much of our web viewing we'd trade the resolution for speed of delivery. In the absence of scalable image compression technology it is common practice to generate multiple copies of the compressed bitstream, varying the spatial region, resolution and compression ratio, and put all copies on a web server in order to accommodate a variety of network situations. The multiple copies of a fixed media source file can cause data management headaches and waste valuable server space. Scalable compression techniques allow a single scalable master bitstream of the compressed image on the server to serve all purposes. During image browsing, the user may specify a region of interest (ROI) with a certain spatial and resolution constraint. The browser then only downloads a subset of the compressed media bitstream covering the current ROI, and the download can be performed in a progressive fashion so that a coarse view of the ROI can be rendered very quickly and then gradually refined as more and more bits arrive. Therefore, with scalable image compression, it is possible to browse large images quickly and on demand (see e.g., the Vmedia project [25]).

3. JPEG 2000

3.1. History. JPEG 2000 is the successor to JPEG. The acronym JPEG stands for Joint Photographic Experts Group. This is a group of image processing experts, nominated by national standard bodies and major companies to work to produce standards for continuous tone image coding. The official title of the committee is "ISO/IEC JTC1/SC29 Working Group 1", which often appears in

the reference document. The JPEG members select a DCT based image compression algorithm in 1988, and while the original JPEG was quite successful, it became clear in the early 1990s that new wavelet-based image compression schemes such as CREW (compression with reversible embedded wavelets) [5] and EZW (embedded zerotree wavelets) [6] were surpassing JPEG in both performance and available features, such as scalability. It was time to begin to rethink the industry standard in order to incorporate these new mathematical advances.

Based on industrial demand, the JPEG 2000 research and development effort was initiated in 1996. A call for technical contributions was issued in March 1997 [17]. The first evaluation was performed in November 1997 in Sydney, Australia, where twenty-four algorithms were submitted and evaluated. Following the evaluation, it was decided to create a JPEG 2000 "verification model" (VM) which was a reference implementation (in document and in software) of the working standard. The first VM (VM0) is based on the wavelet/trellis coded quantization (WTCQ) algorithm submitted by SAIC and the University of Arizona (SAIC/UA) [18]. At the November 1998 meeting, the algorithm EBCOT (embedded block coding with optimized truncation) was adopted into VM3, and the entire VM software was re-implemented in an object-oriented manner. The document describing the basic JPEG 2000 decoder (part I) reached committee draft (CD) status in December 1999. JPEG 2000 finally became an international standard (IS) in December 2000.

3.2. JPEG. In order to understand JPEG 2000, it is instructive to revisit the original JPEG. As illustrated by Figure 2, JPEG is composed of a sequence of four main modules.

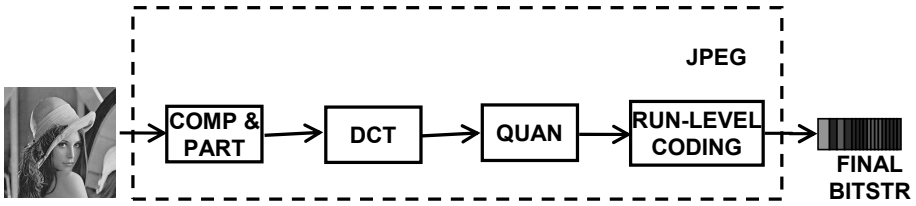

Figure 2. Operation flow of JPEG.

The first module (COMP & PART) performs *component and tile separation*, whose function is to cut the image into manageable chunks for processing. *Tile separation* is simply the separation of the image into spatially non-overlapping tiles of equal size. *Component separation* makes possible the decorrelation of color components. For example, a color image, in which each pixel is normally represented with three numbers indicating the levels of red, green and blue (RGB) may be transformed to LCrCb (luminance, chrominance red and chrominance blue) space.

After separation, each tile of each component is then processed separately according to a *discrete cosine transform* (DCT). This is closely related to the Fourier transform (see [30], for example). The coefficients are then *quantized*. Quantization takes the DCT coefficients (typically some sort of floating point number) and turns them into an integer. For example, simple rounding is a form of quantization. In the case of JPEG, we apply rounding plus a mask which applies a system of weights reflecting various psychoacoustic observations regarding human processing of images [31]. Finally, the coefficients are subjected to a form of *run-level encoding,* where the basic symbol is a run-length of zeros followed by a non-zero level, the combined symbol is then Huffman encoded.

3.3. Overview of JPEG 2000. Like JPEG, JPEG 2000 standardizes the decoder and the bitstream syntax. The operation flow of a typical JPEG 2000 encoder is shown in Figure 3.

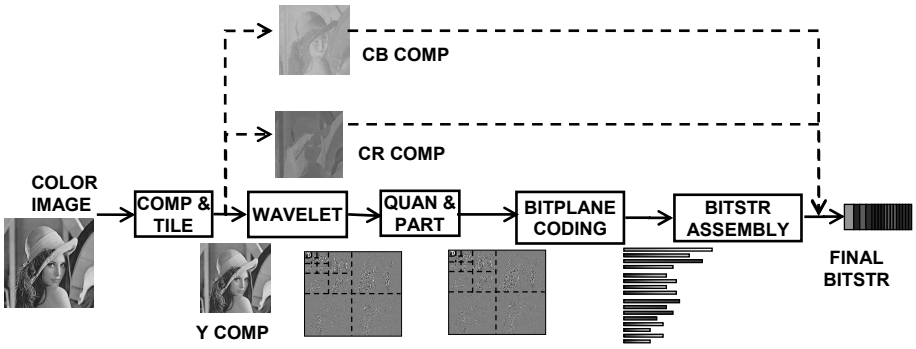

Figure 3. Flowchart for JPEG 2000.

We again start with a component and tile separation module. After this preprocessing, we now apply a *wavelet transform* which yields a sequence of *wavelet coefficients*. This is a key difference between JPEG and JPEG 2000 and we explain it in some detail in Section 4. We next quantize the wavelet coefficients which are then regrouped to facilitate localized spatial and resolution access, where by "resolution" we mean effectively the "degree" of the wavelet coefficient, as the wavelet decomposition is thought of as an expansion of the original data vector in terms of a basis which accounts for finer and finer detail, or increasing resolution. The degrees of resolution are organized into *subbands*, which are divided into non-overlapping rectangular blocks. Three spatially co-located rectangles (one from each subband at a given resolution level) form a *packet partition*. Each packet partition is further divided into *code-blocks*, each of which is compressed by a subbitplane coder into an *embedded bitstream* with

a *rate-distortion curve* that records the distortion and rate at the end of each subbitplane. The embedded bitstream of the code-blocks are assembled into packets, each of which represents an increment in quality corresponding to one level of resolution at one spatial location. Collecting packets from all packet partitions of all resolution level of all tiles and all components, we form a layer that gives one increment in quality of the entire image at full resolution. The final JPEG 2000 bitstream may consist of multiple layers.

We summarize the main differences:

(1) **Transform module: wavelet versus DCT.** JPEG uses 8×8 discrete cosine transform (DCT), while JPEG 2000 uses a wavelet transform with lifting implementation (see Section 4.1). The wavelet transform provides not only better energy compaction (thus higher coding gain), but also the resolution scalability. Because the wavelet coefficients can be separated into different resolutions, it is feasible to extract a lower resolution image by using only the necessary wavelet coefficients.

(2) **Block partition: spatial domain versus wavelet domain.** JPEG partitions the image into 16×16 macroblocks in the space domain, and then applies the transform, quantization and entropy coding operation on each block separately. Since blocks are independently encoded, annoying blocking artifacts becomes noticeable whenever the coding rate is low. On the contrary, JPEG 2000 performs the partition operation in the wavelet domain. Coupled with the wavelet transform, there is no blocking artifact in JPEG 2000.

(3) **Entropy coding module: run-level coefficient coding versus bitplane coding.** JPEG encodes the DCT transform coefficients one by one. The resultant block bitstream can not be truncated. JPEG 2000 encodes the wavelet coefficients bitplane by bitplane (i.e., sending all zeroth order bits, then first order, etc. Details are in Section 4.3). The generated bitstream can be truncated at any point with graceful quality degradation. It is the bitplane entropy coder in JPEG 2000 that enables the bitstream scalability.

(4) **Rate control: quantization module versus bitstream assembly module.** In JPEG, the compression ratio and the amount of distortion is determined by the quantization module. In JPEG 2000, the quantization module simply converts the float coefficient of the wavelet transform module into an integer coefficient for further entropy coding. The compression ratio and distortion is determined by the bitstream assembly module. Thus, JPEG 2000 can manipulate the compressed bitstream, e.g., convert a compressed bitstream to a bitstream of higher compression ratio, form a new bitstream of lower resolution, form a new bitstream of a different spatial area, by operating only on the compressed bitstream and without going through the entropy coding and transform module. As a result, JPEG 2000 compressed bitstream can be reshaped (transcoded) very efficiently.

4. The Wavelet Transform

4.1. Introduction. Most existing high performance image coders in applications are *transform based coders*. In the transform coder, the image pixels are converted from the spatial domain to the transform domain through a linear orthogonal or bi-orthogonal transform. A good choice of transform accomplishes a decorrelation of the pixels, while simultaneously providing a representation in which most of the energy is usually restricted to a few (realtively large) coefficients. This is the key to achieving an efficient coding (i.e., high compression ratio). Indeed, since most of the energy rests in a few large transform coefficients, we may adopt entropy coding schemes, e.g., run-level coding or bitplane coding schemes, that easily locate those coefficients and encodes them. Because the transform coefficients are highly decorrelated, the subsequent quantizer and entropy coder can ignore the correlation among the transform coefficients, and model them as independent random variables.

The optimal transform (in terms of decorrelation) of an image block can be derived through the *Karhunen–Loeve (K-L) decomposition*. Here we model the pixels as a set of statistically dependent random variables, and the K-L basis is that which achieves a diagonalization of the (empirically determined) covariance matrix. This is equivalent to computing the SVD (singular value decomposition) of the covariance matrix (see [28] for a thorough description). However, the K-L transform lacks an efficient algorithm, and the transform basis is content dependent (in distinction, the Fourier transform, which uses the sampled exponentials, is not data dependent).

Popular transforms adopted in image coding include *block-based transforms*, such as the DCT, and wavelet transforms. The DCT (used in JPEG) has many well-known efficient implementations [26], and achieves good energy compaction as well as coefficient decorrelation. However, the DCT is calculated independently in spatially disjoint pixel blocks. Therefore, coding errors (i.e., lossy compression) can cause discontinuities between blocks, which in turn lead to annoying blocking artifacts. In contrary, the wavelet transform operates on the entire image (or a tile of a component in the case of large color image), which both gives better energy compaction than the DCT, and no post-coding blocking artifact. Moreover, the wavelet transform decomposes the image into an *L-level dyadic wavelet pyramid*. The output of an example 5-level dyadic wavelet pyramid is shown in Figure 4.

There is an obvious recursive structure generated by the following algorithm: lowpass and highpass filters (explained below, but for the moment, assume that these are convolution operators) are applied independently to both the rows and columns of the image. The output of these filters is then organized into four new 2D arrays of one half the size (in each dimension), yielding a LL (lowpass, lowpass) block, LH (lowpass, highpass), HL block and HH block. The algorithm is then applied recursively to the LL block, which is essentially a lower resolution

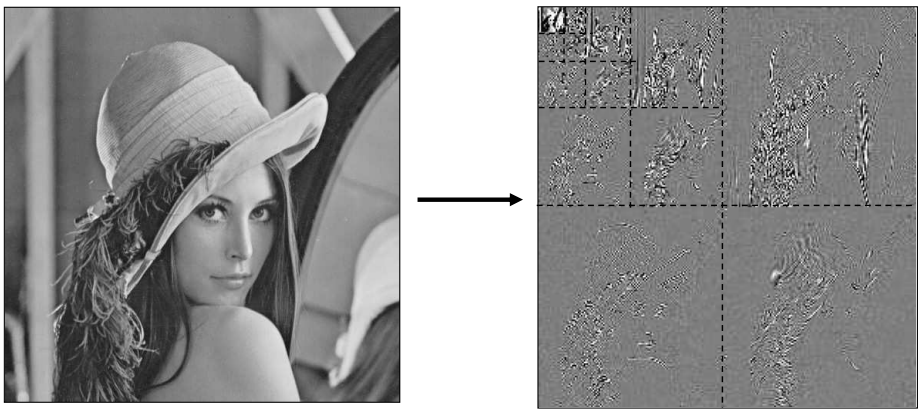

Figure 4. A 5-level dyadic wavelet pyramid.

or smoothed version of the original. This output is organized as in Figure 4, with the southwest, southeast, and northeast quadrants of the various levels housing the LH, HH, and HL blocks respectively. We examine their structure as well as the algorithm in Sections 4.2 and 4.3. By not using the wavelet coefficients at the finest M levels, we can reconstruct an image that is 2^M times smaller in both the horizontal and vertical directions than the original one. The multiresolution nature (see [27], for example) of the wavelet transform is ideal for resolution scalability.

4.2. Wavelet transform by lifting. Wavelets yield a signal representation in which the low order (or lowpass) coefficients represent the most slowly changing data while the high order (highpass) coefficients represent more localized changes. It provides an elegant framework in which both short term anomaly and long term trend can be analyzed on an equal footing. For the theory of wavelet and multiresolution analysis, we refer the reader to [7; 8; 9].

We develop the framework of a one-dimensional wavelet transform using the z-transform formalism. In this setting a given (bi-infinite) discrete signal $x[n]$ is represented by the Laurent series $X(z)$ in which $x[n]$ is the coefficient of z^n. The z-transform of a FIR filter (*finite impulse response*, meaning Laurent series with a finite number of nonzero coefficients, and thus a Laurent polynomial) $H(z)$ is represented by a Laurent polynomial

$$H(z) = \sum_{k=p}^{q} h(k) z^{-k} \qquad \text{of } \textit{degree } |H| = q - p.$$

Thus the length of a filter is the degree of its associated polynomial plus one. The sum or difference of two Laurent polynomials is again a Laurent polynomial and the product of two Laurent polynomials of degree a and b is a Laurent polynomial

of degree $a + b$. Exact division is in general not possible, but division with remainder is possible. This means that for any two nonzero Laurent polynomials $a(z)$ and $b(z)$, with $|a(z)| \geq |b(z)|$, there will always exist a Laurent polynomial $q(z)$ with $|q(z)| = |a(z)| - |b(z)|$ and a Laurent polynomial $r(z)$ with $|r(z)| < |b(z)|$ such that

$$a(z) = b(z)q(z) + r(z).$$

This division is not necessarily unique. A Laurent polynomial is invertible if and only if it is of degree zero, i.e., if it is of the form cz^p.

The original signal $X(z)$ goes through a low and high-pass analysis FIR filter pair $G(z)$ and $H(z)$. These are simply the independent convolutions of the original data sequence against a pair of masks, and constitute perhaps the most basic example of a *filterbank* [27]. The resulting pair of outputs are subsampled by a factor of two. To reconstruct the original signal, the low and high-pass coefficients $\gamma(z)$ and $\lambda(z)$ are upsampled by a factor of two and pass through another pair of synthesis FIR filters $G'(z)$ and $H'(z)$. Although IIR (infinite impulse response) filters can also be used, the infinite response leads to an infinite data expansion, an undesirable outcome in our finite world. According to filterbank theory, if the filters satisfy the relations

$$G(z)G(z^{-1}) + H'(z)H(z^{-1}) = 2,$$
$$G(z)G(-z^{-1}) + H'(z)H(-z^{-1}) = 0,$$

the aliasing caused by the subsampling will be cancelled, and the reconstructed signal $Y(z)$ will be equal to the original. Figure 5 provides an illustration.

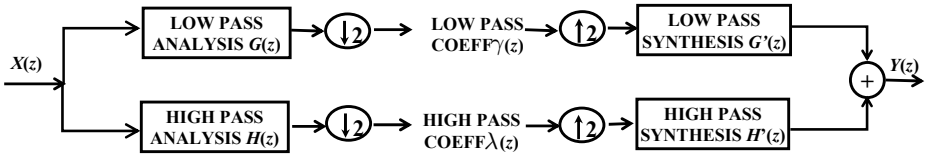

Figure 5. Convolution implementation of one dimensional wavelet transform.

A wavelet transform implemented in the fashion of Figure 5 with FIR filters is said to have a *convolutional implementation*, reflecting the fact that the signal is convolved with the pair of filters (h, g) that form the *filter bank*. Note that only half the samples are kept by the subsampling operator, and the other half of the filtered samples are thrown away. Clearly this is not efficient, and it would be better (by a factor of one-half) to do the subsampling before the filtering. This leads to an alternative implementation of the wavelet transform called *lifting* approach. It turns out that all FIR wavelet filters can be factored into lifting step. We explain the basic idea in what follows. For those interested in a deeper understanding, we refer to [10; 11; 12].

The subsampling that is performed at the forward wavelet, and the upsampling that is used in the inverse wavelet transform suggest the utility of a decomposition of the z-transform of the signal/filter into an even and odd part given by subsampling the z-transform at the even and odd indices, respectively:

$$H(z) = \sum_n h(n) z^{-n} \qquad \begin{cases} H_e(z) = \sum_n h(2n) z^{-n} & \text{(even part)}, \\ H_o(z) = \sum_n h(2n+1) z^{-n} & \text{(odd part)}. \end{cases}$$

The odd/even decomposition can be rewritten as

$$H(z) = H_e(z^2) + z^{-1} H_o(z^2) \quad \text{with} \quad \begin{cases} H_e(z) = \frac{1}{2}\big(H(z^{1/2}) + H(-z^{1/2})\big), \\ H_o(z) = \frac{1}{2} z^{1/2}\big(H(z^{1/2}) - H(-z^{1/2})\big). \end{cases}$$

With this we may rewrite the wavelet filtering and subsampling operation (i.e., the lowpass and highpass components, $\gamma(z)$ and $\lambda(z)$, respectively) using the even/odd parts of the signal and filter as

$$\begin{aligned} \gamma(z) &= G_e(z) X_e(z) + z^{-1} G_o(z) X_o(z), \\ \lambda(z) &= H_e(z) X_e(z) + z^{-1} H_o(z) X_o(z), \end{aligned}$$

which can be written in matrix form as

$$\begin{pmatrix} \gamma(z) \\ \lambda(z) \end{pmatrix} = P(z) \begin{pmatrix} X_e(z) \\ z^{-1} X_o(z) \end{pmatrix},$$

where $P(z)$ is the polyphase matrix

$$P(z) = \begin{pmatrix} G_e(z) & G_o(z) \\ H_e(z) & H_o(z) \end{pmatrix}.$$

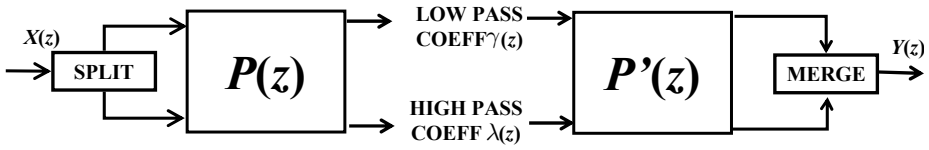

Figure 6. Single stage wavelet filter using polyphase matrices.

The forward wavelet transform now becomes the left part of Figure 6. Note that with polyphase matrix, we perform the subsampling (split) operation before the signal is filtered, which is more efficient than the description illustrated by Figure 5, in which the subsampling is performed after the signal is filtered. We move on to the inverse wavelet transform. It is not difficult to see that the odd/even subsampling of the reconstructed signal can be obtained through

$$\begin{pmatrix} Y_e(z) \\ z Y_o(z) \end{pmatrix} = P(z) \begin{pmatrix} \gamma(z) \\ \lambda(z) \end{pmatrix},$$

where $P'(z)$ is a dual polyphase matrix

$$P'(z) = \begin{pmatrix} G'_e(z) & G'_o(z) \\ G'_e(z) & H'_o(z) \end{pmatrix}.$$

The wavelet transform is invertible if the two polyphase matrices are inverse to each other:

$$P'(z) = P(z)^{-1} = \frac{1}{H_o(z)G_e(z) - H_e(z)G_o(z)} \begin{pmatrix} H_o(z) & -G_o(z) \\ -H_e(z) & G_e(z) \end{pmatrix}.$$

If we constrain the determinant of the polyphase matrix to be one, i.e., $H_o(z)G_e(z) - H_e(z)G_o(z) = 1$, then not only are the polyphase matrices invertible, but the inverse filter has a simple relationship to the forward filter:

$$\begin{aligned} G'_e(z) &= H_o(z), & H'_e(z) &= -G_o(z), \\ G'_o(z) &= -H_e(z), & H'_o(z) &= G_2(z), \end{aligned}$$

which implies that the inverse filter is related to the forward filter by the equations

$$G'_e(z) = z^{-1}H(-z^{-1}), \qquad H'(z) = -z^{-1}G(-z^{-1})$$

The corresponding pair of filters (g, h) is said to be *complementary*. Figure 6 illustrates the forward and inverse transforms using the polyphase matrices.

With the Laurent polynomial and polyphase matrix, we can factor a wavelet filter into the lifting steps. Starting with a complementary filter pair (g, h), assume that the degree of filter g is larger than that of filter h. We seek a new filter g^{new} satisfying

$$g(z) = h9z)t(z^2) + g^{\text{new}}(z),$$

where $t(z)$ is a Laurent polynomial. Both $t(z)$ and g^{new}(z) can be calculated through long division [10]. The new filter g^{new} is complementary to filter h, as the polyphase matrix satisfies

$$\begin{aligned} P(z) &= \begin{pmatrix} H_e(z)t(z) + G_e^{\text{new}}(z) & H_o(z)t(z) + G_o^{\text{new}}(z) \\ H_e(z) & H_o(z) \end{pmatrix} \\ &= \begin{pmatrix} 1 & t(z) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} G_e^{\text{new}}(z) & G_o^{\text{new}}(z) \\ H_e(z) & H_o(z) \end{pmatrix} = \begin{pmatrix} 1 & t(z) \\ 0 & 1 \end{pmatrix} P^{\text{new}}(z). \end{aligned}$$

Obviously, the determinant of the new polyphase matrix $P^{\text{new}}(z)$ also equals one. By performing the operation iteratively, it is possible to factor the polyphase matrix into a sequence of lifting steps:

$$P(z) = \begin{pmatrix} K_1 & \\ & K_2 \end{pmatrix} \prod_{i=0}^{m} \left(\begin{pmatrix} 1 & t_i(z) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ s_i(z) & 1 \end{pmatrix} \right).$$

The resultant lifting wavelet can be shown in Figure 7.

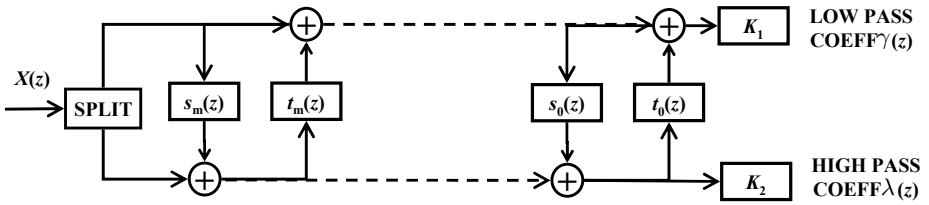

Figure 7. Multi-stage forward lifting wavelet using polyphase matrices.

Each lifting stage above can be directly inverted. Thus we can invert the entire wavelet:

$$P'(z) = P(z)^{-1} = \begin{pmatrix} 1/K_1 & \\ & 1/K_2 \end{pmatrix} \prod_{i=m}^{0} \left(\begin{pmatrix} 1 & 0 \\ -s_i(z) & 1 \end{pmatrix} \begin{pmatrix} 1 & -t_i(z) \\ 0 & 1 \end{pmatrix} \right).$$

We show the inverse lifting wavelet using polyphase matrices in Figure 8, which should be compared with Figure 7. Only the direction of the data flow has changed.

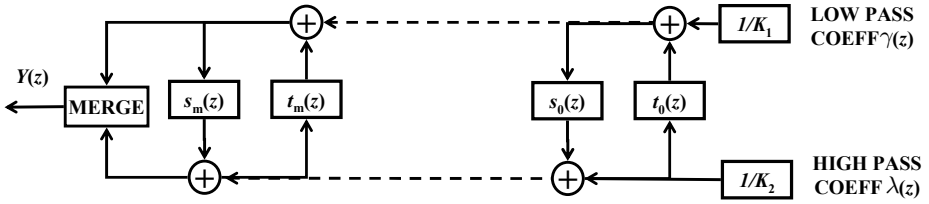

Figure 8. Multi-stage inverse lifting wavelet using polyphase matrices.

4.3. Bi-orthogonal 9-7 wavelet and boundary extension. The default wavelet filter used in JPEG 2000 is the bi-orthogonal 9-7 wavelet [20]. It is a 4-stage lifting wavelet, with lifting filters $s_1(z) = f(a, z)$, $t_1(z) = f(b, z)$, $s_2(z) = f(c, z)$, $t_0(z) = f(d, z)$, where f, the *dual lifting step*, is of the form

$$f(p, z) = pz^{-1} + p.$$

The quantities a, b, c and d are the *lifting parameters* at each stage.

The next several figures illustrate the filterbank. The input data is indexed as $\ldots, x_0, x_1, \ldots, x_n, \ldots$, and the lifting operation is performed from right to left, stage by stage. At this moment, we assume that the data is of infinite length, and we will discuss boundary extension later. The input data are first partitioned into two groups corresponding to even and odd indices. During each lifting stage, only one of the group is updated. In the first lifting stage, the odd index data points x_1, x_3, ... are updated:

$$x'_{2n+1} = x_{2n+1} + a * (x_{2n} + x_{2n+2}),$$

where a and x'_{2n+1} are respectively the first stage lifting parameter and outcome. The entire operation corresponds to the filter $s_1(z)$ represented in Figure 8. The circle in Figure 9 illustrates one such operation performed on x_1.

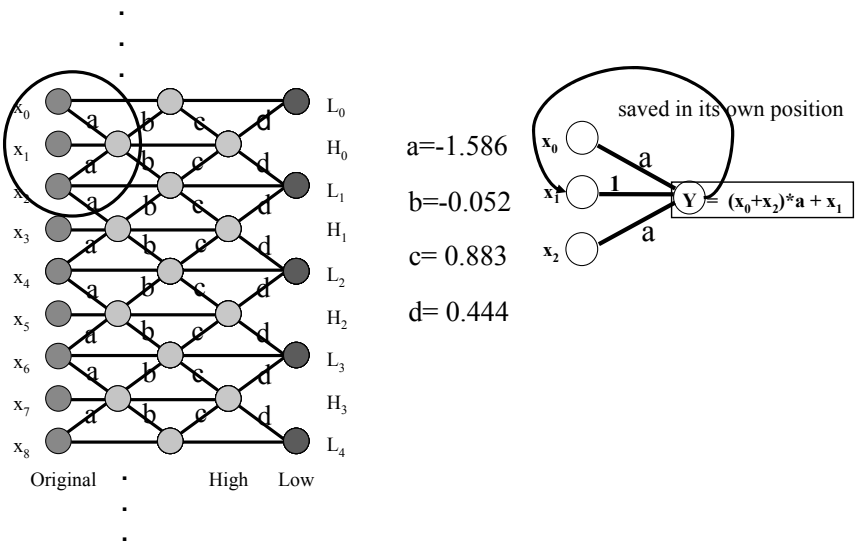

Figure 9. Bi-orthogonal 9-7 wavelet.

The second stage lifting, which corresponds to the filter $t_1(z)$ in Figure 8, updates the data at even indices:

$$x''_{2n} = x_{2n} + b * (x'_{2n-1} + x'_{2n+1}),$$

where b and x''_{2n} are the second stage lifting parameter and output. The third and fourth stage lifting can be performed similarly:

$$H_n = x'_{2n+1} + c * (x''_{2n} + x''_{2n+2}),$$
$$L_n = x''_{2n} + d * (H_{n-1} + H_n),$$

where H_n and L_n are the resultant high and low-pass coefficients. The value of the lifting parameters a, b, c, d are shown in Figure 9.

As illustrated in Figure 10, we may invert the dataflow, and derive an inverse lifting of the 9-7 bi-orthogonal wavelet.

Since the actual data in an image transform is finite in length, boundary extension is a crucial part of every wavelet decomposition scheme. For a symmetric odd-tap filter (the bi-orthogonal 9-7 wavelet falls into this category), symmetric boundary extension can be used. The data are reflected symmetrically along the boundary, with the boundary points themselves not involved in the reflection. An example boundary extension with four data points x_0, x_1, x_2 and x_3

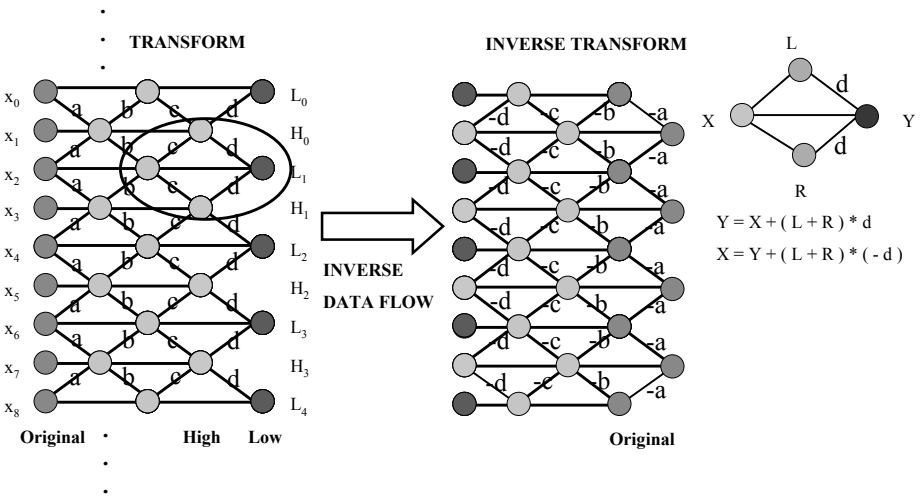

Figure 10. Forward and inverse lifting (9-7 bi-orthogonal wavelet).

is shown in Figure 11. Because both the extended data and the lifting structure are symmetric, all the intermediate and final results of the lifting are also symmetric with respect to the boundary points. Using this observation, it is sufficient to double the lifting parameters of the branches that are pointing toward the boundary, as shown in the middle of Figure 11. Thus, the boundary extension can be performed without additional computational complexity. The inverse lifting can again be derived by inverting the dataflow, as shown in the right of Figure 11. Again, the parameters for branches that are pointing toward the boundary points are doubled.

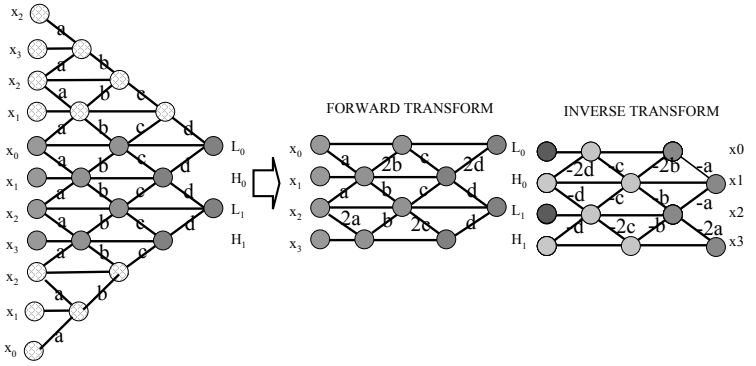

Figure 11. Symmetric boundary extension of bi-orthogonal 9-7 wavelet on 4 data points.

4.4. Two-dimensional wavelet transform. To apply a wavelet transform to an image we need to use a 2D version. In this case it is common to apply the wavelet transform separately in the horizontal and vertical directions. This approach is called the *separable 2D wavelet transform*. It is possible to design a *nonseparable* 2D wavelet (see [32], for example), but this generally increases computational complexity with little additional coding gain. A sample one-scale separable 2D wavelet transform is shown in Figure 12. The 2D data array representing the image is first filtered in the horizontal direction, which results in two subbands: a horizontal low-pass and a horizontal high-pass subband. These subbands are then passed through a vertical wavelet filter. The image is thus decomposed into four subbands: LL (low-pass horizontal and vertical filter), LH (low-pass vertical and high-pass horizontal filter), HL (high-pass vertical and low-pass horizontal filter) and HH (high-pass horizontal and vertical filter). Since the wavelet transform is linear, we may switch the order of the horizontal and vertical filters yet still reach the same effect. By further decomposing subband LL with another 2D wavelet (and iterating this procedure), we derive a *multiscale dyadic wavelet pyramid*. Recall that such a wavelet was illustrated in Figure 4.

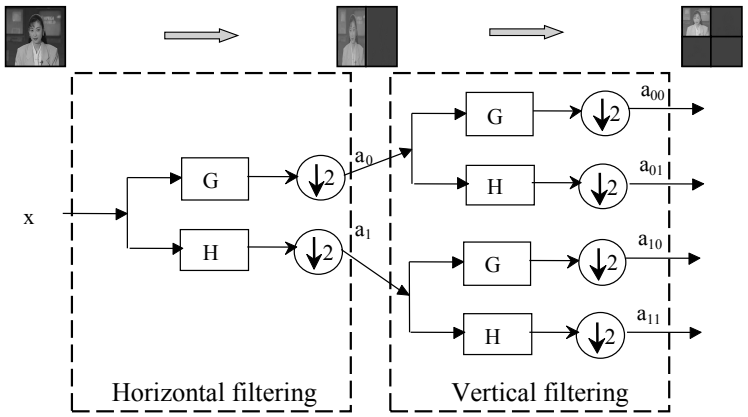

Figure 12. A single scale 2D wavelet transform.

4.5. Line-based lifting. A trick in implementing the 2D wavelet transform is *line-based lifting*, which avoids buffering the entire 2D image during the vertical wavelet lifting operation. The concept can be shown in Figure 13, which is very similar to Figure 9, except that here each circle represents an entire line (row) of the image. Instead of performing the lifting stage by stage, as in Figure 9, line-based lifting computes the vertical low- and high-pass lifting, one line at a time. The operation can be described as follows:

Step 1: Initialization, phase 1. Three lines of coefficients x_0, x_1 and x_2 are processed. Two lines of lifting operations are performed, and intermediate results x_1' and x_0'' are generated.

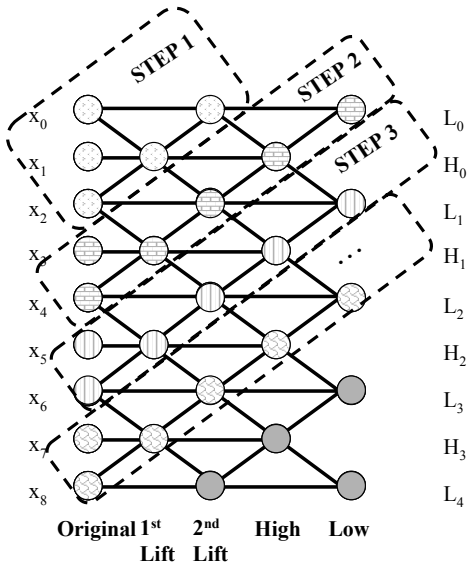

Figure 13. Line-based lifting wavelet (bi-orthogonal 9-7 wavelet).

Step 2: Initialization, phase 2. Two additional lines of coefficients x_3 andx_4 are processed. Four lines of lifting operations are performed. The outcomes are the intermediate results x_3' and x_4'', and the first line of low and high-pass coefficients L_0 and H_0.

Step 3: Repeated processing. During the normal operation, the line based lifting module reads in two lines of coefficients, performs four lines of lifting operations, and generates one line of low and high-pass coefficients.

Step 4: Flushing. When the bottom of the image is reached, symmetrical boundary extension is performed to correctly generate the final low and high-pass coefficients.

For the 9-7 bi-orthogonal wavelet, with line-based lifting, only six lines of working memory are required to perform the 2D lifting operation. By eliminating the need to buffer the entire image during the vertical wavelet lifting operation, the cost to implement 2D wavelet transform can be greatly reduced

5. Quantization and Partitioning

After the wavelet transform, all wavelet coefficients are uniformly quantized according to the rule

$$w_{m,n} = \operatorname{sign} s_{m,n} \left\lfloor \frac{|s_{m,n}|}{\delta} \right\rfloor,$$

where $s_{m,n}$ is the transform coefficient, $w_{m,n}$ is the quantization result, δ is the quantization step size, $\operatorname{sign}(x)$ returns the sign of coefficient x, and $\lfloor\ \rfloor$ is the floor function. The effect of quantization is demonstrated in Figure 14.

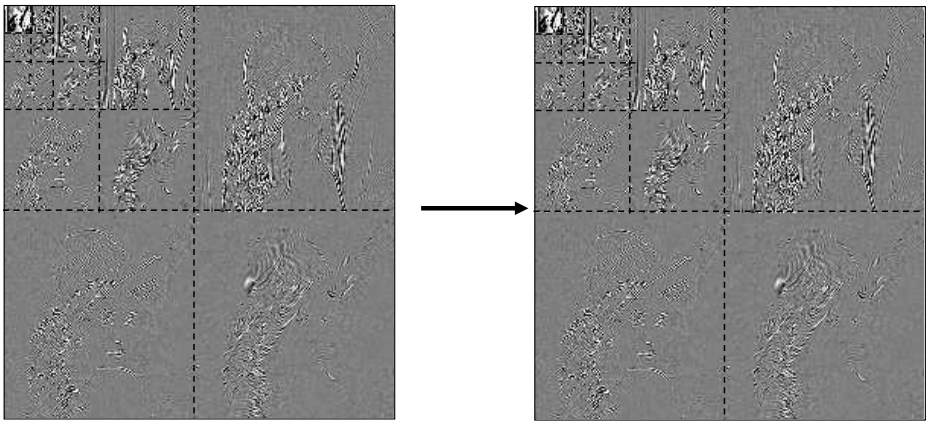

Figure 14. Effect of quantization.

The quantization process of JPEG 2000 is very similar to that of a conventional coder such as JPEG. However, the functionality is very different. In a conventional coder, since the quantization result is losslessly encoded, the quantization process determines the allowable distortion of the transform coefficients. In JPEG 2000, the quantized coefficients are lossy encoded through an embedded coder, thus additional distortion can be introduced in the entropy coding steps. Thus, the main functionality of the quantization module is to map the coefficients from floating representation into integer so that they can be more efficiently processed by the entropy coding module. The image coding quality is not determined by the quantization step size δ but by the subsequent bitstream assembler. The default quantization step size in JPEG 2000 is rather fine, e.g., $\delta = \frac{1}{128}$.

The quantized coefficients are then partitioned into packets. Each subband is divided into non-overlapping rectangles of equal size, as described above, this means three rectangles corresponding to the subbands HL, LH, HH of each resolution level. The packet partition provides spatial locality as it contains information needed for decoding image of a certain spatial region at a certain resolution.

The packets are further divided into non-overlapping rectangular code-blocks, which are the fundamental entities in the entropy coding operation. By applying the entropy coder to relatively small code-blocks, the original and working data of the entire code-blocks can reside in the cache of the CPU during the entropy coding operation. This greatly improves the encoding and decoding speed. In JPEG 2000, the default size of a code-block is 64×64. A sample partition and code-blocks are shown in Figure 15. We mark the partition with solid thick lines. The partition contains quantized coefficients at spatial location $(128, 128)$

to $(255, 255)$ of the resolution 1 subbands LH, HL and HH. It corresponds to the resolution 1 enhancement of the image with spatial location $(256, 256)$ to $(511, 511)$. The partition is further divided into twelve 64×64 code-blocks, which are shown as numbered blocks in Figure 15.

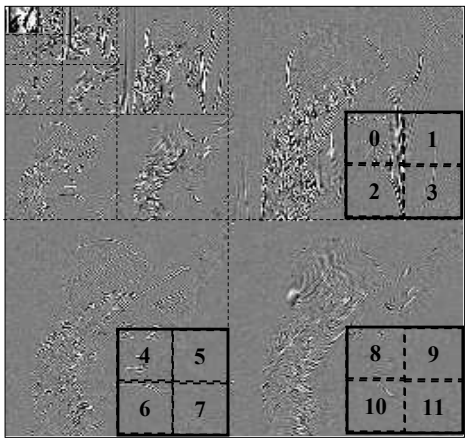

Figure 15. A sample partition and code-blocks.

6. Block Entropy Coding

Following the partitioning, each code-block is then independently encoded through a subbitplane entropy coder. As shown in Figure 16, the input of the block entropy coding module is the code-block, which can be represented as a 2D array of data. The output of the module is a embedded compressed bitstream, which can be truncated at any point and still be decodable, and a rate-distortion (R-D) curve (see Figure 16).

It is the responsibility of the block entropy coder to measure both the coding rate and distortion during the encoding process. The coding rate is derived directly through the length of the coding bitstream at certain instances, e.g., at the end of each subbitplane. The coding distortion is obtained by measuring the distortion between the original coefficient and the reconstructed coefficient at the same instance.

JPEG 2000 employs a subbitplane entropy coder. In what follows, we examine three key parts of the coder: the coding order, the context, and the arithmetic MQ-coder.

6.1. Embedded coding. Assume that each quantized coefficient $w_{m,n}$ is represented in the binary form as

$$\pm b_1 b_2 \dots b_n,$$

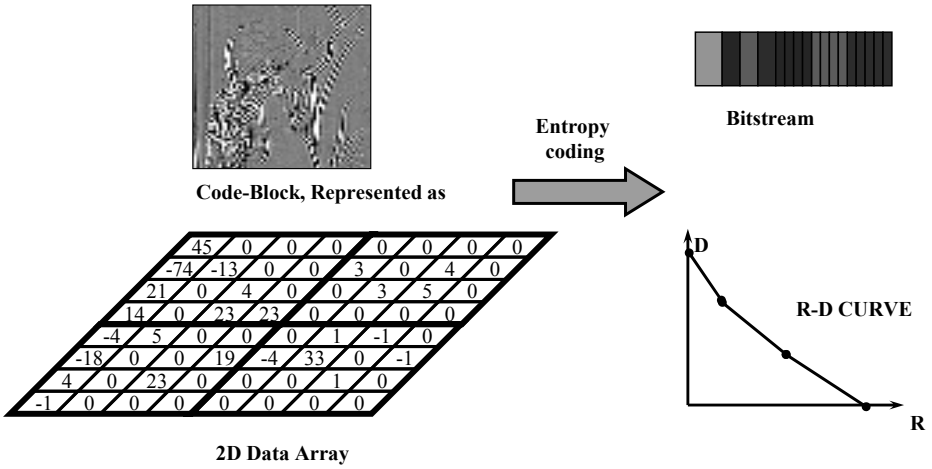

Figure 16. Block entropy coding.

where b_1 is the *most significant bit* (MSB), and b_n is the *least significant bit* (LSB), and $\pm$ represents the sign of the coefficient. It is the job of the entropy coding module to first convert this array of bits into a single sequence of binary bits, and then compress this bit sequence with a lossless coder, such as an arithmetic coder [22]. A *bitplane* is defined as the group of bits at a given level of significance. Thus, for each codeblock there is a bitplane consisting of all MSBs, one of all LSBs, and one for each of the significance levels that occur in between. By coding the more significant bits of all coefficients first, and coding the less significant bits later, the resulting compressed bitstream is said to have *the embedding property*, reflecting the fact that a bitstream of lower compression rate can be obtained by simply truncating a higher rate bitstream, so that the entire output stream has embedded in it bitstreams of lower compression that still make possible of partial decoding of all coefficients. A sample binary representation of the coefficient can be shown in Figure 17. Since representing bits in a 2D block results in a 3D bit array (the 3^{rd} dimension is bit significance) which is very difficult to draw, we only show the binary representation of a column of coefficients as a 2D bit array in Figure 17. However, keep in mind that the true bit array in a code-block is 3D.

The bits in the bit array are very different, both in their statistical property and in their contribution to the quality of the decoded code-block. The sign is obviously different from that of the coefficient bit. The bits at different significance level contributes differently to the quality of the decoded code-blocks. And even within the same bitplane, bits may have different statistical property and contribution to the quality of decoding. Let b_M be a bit in a coefficient x. If all more significant bits in the same coefficient x are '0's, the coefficient x is said to be insignificant (because if the bitstream is terminated at this point or before, coefficient x will be reconstructed to zero), and the current bit b_M is to

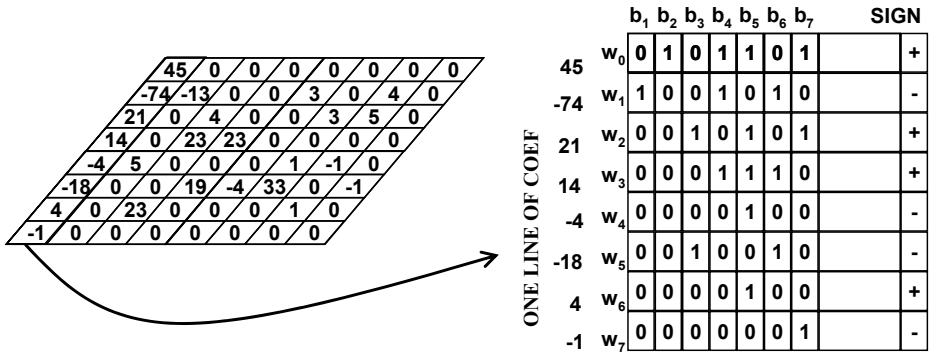

Figure 17. Coefficients and binary representation.

be encoded in the mode of significance identification. Otherwise, the coefficient is said to be significant, and the bit b_M is to be encoded in the mode of refinement. Depending on the sign of the coefficient, the coefficient can be positive significant or negative significant. We distinguish between significance identification and refinement bits because the significance identification bit has a very high probability of being 0, and the refinement bit is usually equally distributed between 0 and 1. The sign of the coefficient needs to be encoded immediately after the coefficient turns significant, i.e., a first non-zero bit in the coefficient is encoded. For the bit array in Figure 17, the significance identification and the refinement bits are shown with different shades in Figure 18.

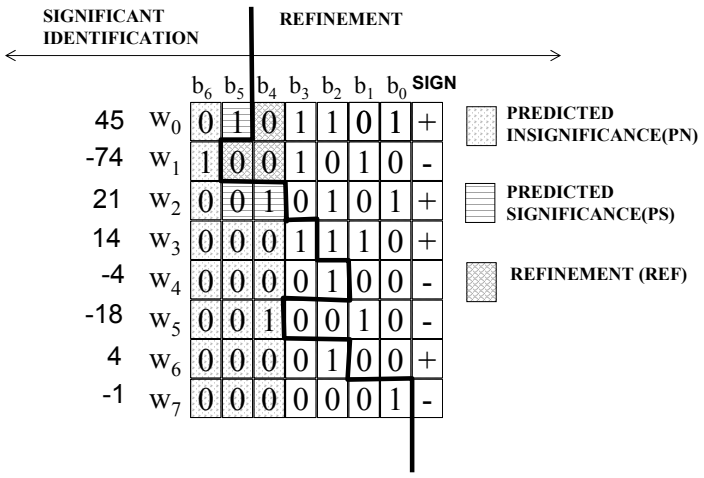

Figure 18. Embedded coding of bit array.

6.2. Context. It has been pointed out [14; 21] that the statistics of significant identification bits, refinement bits, and signs can vary tremendously. For example, if a quantized coefficient$x_{i,j}$ is of large magnitude, its neighbor coefficients may be of large magnitude as well. This is because a large coefficient locates an anomaly (e.g., a sharp edge) in the smooth signal, and such an anomaly usually causes a cluster of large wavelet coefficients in the neighborhood as well. To account for such statistical variation, we entropy encode the significant identification bits, refinement bits and signs with context, each of which is a number derived from already coded coefficients in the neighborhood of the current coefficient. The bit array that represents the data is thus turned into a sequence of *bit-context pairs*, as shown in Figure 19, which is subsequently encoded by a *context adaptive entropy coder*. In the bit-context pair, it is the bit information that is actually encoded. The context associated with the bit is determined from the already encoded information. It can be derived by the encoder and the decoder alike, provided both use the same rule to generate the context. Bits in the same context are considered to have similar statistical properties, so that the entropy coder can measure the probability distribution within each context and efficiently compress the bits.

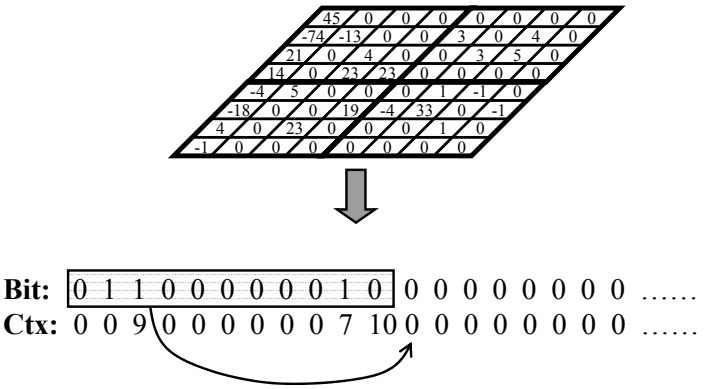

Figure 19. Coding bits and contexts. The context is derived from information from the already coded bits.

In the following, we describe the contexts that are used in the significant identification, refinement and sign coding of JPEG 2000. For the rational of the context design, we refer to [2; 19]. Determining the context of significant identification bit is a two-step process:

Step 1: Neighborhood statistics. For each bit of the coefficient, the number of significant horizontal, vertical and diagonal neighbors are counted as h,$v$and d, as shown in Figure 20.

Step 2: Lookup table. According to the direction of the subband that the coefficient is located (LH, HL, HH), the context of the encoding bit is indexed

LH subband (also LL) (vertically high-pass)				HL subband (horizontally high-pass)				HH subband (diagonally high-pass)		
h	v	d	context	h	v	d	context	d	$h+v$	context
2	x	x	8	x	2	x	8	≥ 3	x	8
1	≥ 1	x	7	≥ 1	1	x	7	2	≥ 1	7
1	0	≥ 1	6	0	1	≥ 1	6	2	0	6
1	0	0	5	0	1	0	5	1	≥ 2	5
0	2	x	4	2	0	x	4	1	1	4
0	1	x	3	1	0	x	3	1	0	3
0	0	≥ 2	2	0	0	≥ 2	2	0	≥ 2	2
0	0	1	1	0	0	1	1	0	1	1
0	0	0	0	0	0	0	0	0	0	0

Table 1. Context for the significance identification coding.

through one of the three tables shown in Table 1. A total of nine context categories are used for significance identification coding. The table lookup process reduces the number of contexts and enables probability of the statistics within each context to be quickly obtained.

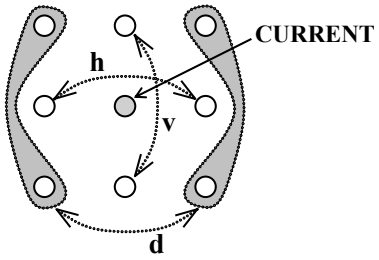

Figure 20. Number of significant neighbors: horizontal (h), vertical (v) and diagonal (d).

To determine the context for sign coding, we calculate a horizontal sign count $h$and a vertical sign count v. The sign count takes a value of -1 if both horizontal/vertical coefficients are negative significant; or one coefficient is negative significant, and the other is insignificant. It takes a value of $+1$ if both horizontal/vertical coefficients are positive significant; or one coefficient is positive significant, and the other is insignificant. The value of the sign count is 0 if both horizontal/vertical coefficients are insignificant; or one coefficient is positive significant, and the other is negative significant.

With the horizontal and vertical sign count h and v, an expected sign and a context for sign coding can then be calculated according to Table 2.

To calculate the context for the refinement bits, we measure if the current refinement bit is the first bit after significant identification, and if there is any significant coefficients in the immediate eight neighbors, i.e., $h+v+d>0$. The context for the refinement bit is tabulated in Table 3.

Sign count										
	H	−1	−1	−1	0	0	0	1	1	1
	V	−1	0	1	−1	0	1	−1	0	1
Expected sign		−	−	−	−	+	+	+	+	+
Context		13	12	11	10	9	10	11	12	13

Table 2. Context and the expected sign for sign coding.

Context 14: Current refinement bit is the first bit after significant identification and there is no significant coefficient in the eight neighbors.
Context 15: Current refinement bit is the first bit after significant identification and there is at least one significant coefficient in the eight neighbors.
Context 16: Current refinement bit is at least two bits away from significant identification.

Table 3. Context for the refinement bit.

6.3. MQ-coder: context dependent entropy coder. Through the aforementioned process, a data array is turned into a sequence of bit-context pairs, as shown in Figure 19. All bits associated with the same context are assumed to be independently and identically distributed. Let the number of contexts be N, and let there be n_i bits in context i, within which the probability of the bits taking value 1 is p_i. Using classic Shannon information theory [15; 16] the entropy of such a bit-context sequence can be calculated as

$$H = \sum_{i=0}^{N-1} n_i\big(-p\log_2 p_i - (1-p_i)\log_2(1-p_i)\big). \tag{6–1}$$

The task of the context entropy coder is thus to convert the sequence of bit-context pairs into a compact bitstream representation with length as close to the Shannon limit as possible, as shown in Figure 21. Several coders are available for such task. The coder used in JPEG 2000 is the MQ-coder. In the following, we focus the discussion on three key aspects of the MQ-coder: general arithmetic coding theory, fixed point arithmetic implementation and probability estimation. For more details, we refer to [22; 23].

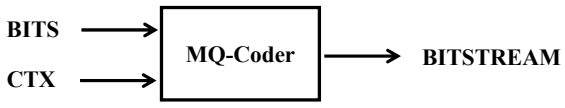

Figure 21. Input and output of the MQ-coder.

6.3.1. The Elias coder. The basic theory of the MQ-coder can be traced to the Elias Coder [24], or recursive probability interval subdivision. Let $S_0S_1S_2\ldots S_n$ be a series of binary bits that is sent to the arithmetic coder. Let P_i be the probability that the bit S_i be 1. We may form a binary representation (the coding bitstream) of the original bit sequence by the following process:

Step 1: Initialization. Let the initial probability interval be $(0, 1)$. We denote the current probability interval as $(C, C+A)$, where C is the bottom of the probability interval, and A is the size of the interval. At the initialization, we have $C = 0$ and $A = 1$.

Step 2: Probability interval subdivision. The binary symbols $S_0S_1S_2\ldots S_n$ are encoded sequentially. For each symbol S_i, the probability interval $(C, C+A)$ is subdivided into two sub-intervals $\big(C,\ C+A(1-P_i)\big)$ and $\big(C+A(1-P_i),\ C+A\big)$. Depending on whether the symbol S_i is 1, one of the two subintervals is selected:

$$\begin{cases} C \leftarrow C, \quad A \leftarrow A(1-P_i), & \text{if } S_i = 0, \\ C \leftarrow A(1-P_i), \quad A \leftarrow AP_i, & \text{if } S_i = 1. \end{cases} \tag{6–2}$$

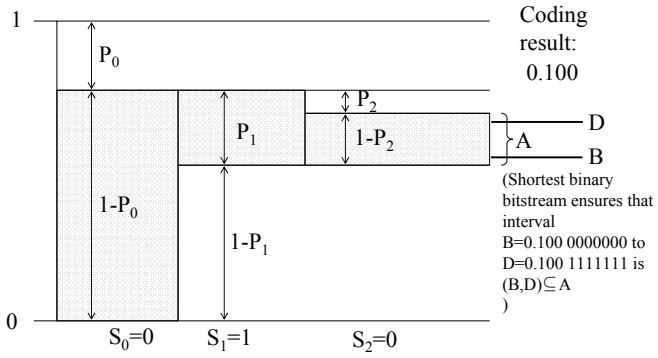

Figure 22. Probability interval subdivision.

Step 3: Bitstream output. Let the final coding bitstream be $k_1k_2\ldots k_m$, where m is the compressed bitstream length. The final bitstream creates an uncertainty interval where the lower and upper bound can be determined as

$$\begin{aligned} \text{Upperbound} \qquad & D = 0.k_1k_2\cdots k_m111\ldots, \\ \text{Lowerbound} \qquad & B = 0.k_1k_2\cdots k_m000\ldots. \end{aligned}$$

As long as the uncertainty interval (B, D) is contained in the probability interval $(C, C+A)$, the coding bitstream uniquely identifies the final probability interval, and thus uniquely identifies each subdivision in the Elias coding process. The entire binary symbol strings $S_0S_1S_2\ldots S_n$ can thus be recovered from the compressed representation. It can be shown that it is possible to find a final coding bitstream with length

$$m \le \lceil -\log_2 A \rceil + 1$$

to represent the final probability interval $(C, C+A)$. Notice that A is the probability of the occurrence of the binary strings $S_0S_1S_2\ldots S_n$, and the entropy of the original symbol stream can be calculated as

$$H = \sum_{S_0S_1\cdots S_n} -A \log_2 A.$$

The arithmetic coder thus encodes the binary string within 2 bits of its entropy limit, no matter how long the symbol string is. This is very efficient.

6.3.2. The arithmetic coder: finite precision arithmetic operations. Exact implementation of Elias coding requires infinite precision arithmetic, an unrealistic assumption in real applications. Using finite precision, the arithmetic coder is developed from Elias coding. Observing the fact that the coding interval A becomes very small after a few operations, we may normalize the coding interval parameter C and A as

$$C = 1.5 \cdot [0.k_1k_2\cdots k_L] + 2^{-L} \cdot 1.5 \cdot C_x, \qquad A = 2^{-L} \cdot 1.5 \cdot A_x,$$

where L is a normalization factor determining the magnitude of the interval A, while A_x and C_x are fixed-point integers representing values between $(0.75, 1.5)$ and $(0, 1.5)$, respectively. Bits $k_1k_2\ldots k_m$ are the output bits that have already been determined (in reality, certain carryover operations have to be handled to derive the true output bitstream). By representing the probability interval with the normalization L and fixed-point integers A_x and C_x, it is possible to use fixed-point arithmetic and normalization operations for the probability interval subdivision operation. Moreover, since the value of A_x is close to 1.0, we may approximate $A_x \cdot P_i$ with P_i, the interval sub-division operation (6–2) calculated as

$$\begin{aligned} C_x &= C_x, & A_x &= A_x - P_i, & &\text{if } S_i = 0, \\ C_x &= C + A_x - P_i, & A_x &= P_i, & &\text{if } S_i = 1, \end{aligned}$$

which can be done quickly without any multiplication. The compression performance suffers a little, as the coding interval now has to be approximated with a fixed-point integer, and $A_x \cdot P_i$ is approximated with P_i. However, experiments show that the degradation in compression performance is less than three percent, which is well worth the saving in implementation complexity.

6.3.3. Probability estimation. In the arithmetic coder it is necessary to estimate the probability P_i for each binary symbol S_i to take the value 1. This is where context comes into play. Within each context, it is assumed that the symbols are independently identically distributed. We may then estimate the probability of the symbol within each context through observation of the past behaviors of symbols in the same context. For example, if we observe n_i symbols in context

i, with o_i symbols to be 1, we may estimate the probability that a symbol takes on the value 1 in context i through Bayesian estimation as

$$P_i = \frac{o_i + 1}{n_i + 2}.$$

In the MQ-coder [22], probability estimation is implemented through a state-transition machine. It may estimate the probability of the context more efficiently, and may take into consideration the non-stationary characteristic of the symbol string. Nevertheless, the principle is still to estimate the probability based on past behavior of the symbols in the same context.

6.4. Coding order: subbitplane entropy coder. In JPEG 2000, because the embedded bitstream of a code-block may be truncated, the coding order, which is the order that the data array is turned into bit-context pair sequence, is of paramount importance. A sub-optimal coding order may allow important information to be lost after the coding bitstream is truncated, and lead to severe coding distortion. It turns out that the optimal coding order first encodes those bits with the steepest rate-distortion slope, which is defined as the coding distortion decrease per bit spent [21]. Just as the statistical properties of the bits are different in the bit array, their contribution of the coding distortion decrease per bit is also different.

Consider a bit b_i in the i-th most significant bitplane, where there are a total of n bitplanes. If the bit is a refinement bit, then previous to the coding of the bit, the uncertainty interval of the coefficient is $(A, A+2^{n-i})$. After the refinement bit has been encoded, the coefficient lies either in $(A, A+2^{n-i-1})$ or in $(A+2^{n-i}, A+2^{n-i-1})$. If we further assume that the value of the coefficient is uniformly distributed in the uncertainty interval, we may calculate the expected distortion before and after the coding as

$$D_{\mathrm{pre,REF}} = \int_A^{A+2^{n-i}} (x - A - 2^{n-i-1})^2 \, dx = \tfrac{1}{12}\, 4^{n-i},$$
$$D_{\mathrm{post,REF}} = \tfrac{1}{12}\, 4^{n-i-1}.$$

Since the value of the coefficient is uniformly distributed in the uncertainty interval, the probability for the refinement bit to take the values 0 and 1 is equal, thus, the coding rate of the refinement bit is:

$$R_{\mathrm{REF}} = H(b_i) = 1 \text{ bit}. \tag{6–3}$$

The rate-distortion slope of the refinement bit at the i-th most significant bitplane is thus:

$$s_{\mathrm{REF}}(i) = \frac{D_{\mathrm{prev,REF}} - D_{\mathrm{post,REF}}}{R_{\mathrm{REF}}} = \frac{\frac{1}{12}\, 4^{n-i} - \frac{1}{12}\, 4^{n-i-1}}{1} = 4^{n-i-2} \tag{6–4}$$

In the same way, we may calculate the expected distortion decrease and coding rate for a significant identification bit at the i-th most significant bitplane. Before

the coding of the bit, the uncertainty interval of the coefficient ranges from -2^{n-i} to 2^{n-i}. After the bit has been encoded, if the coefficient becomes significant, it lies in $(-2^{n-i}, -2^{n-i-1})$ or $(+2^{n-i-1}, +2^{n-i})$ depending on the sign of the coefficient. If the coefficient is still insignificant, it lies in $(-2^{n-i-1}, 2^{n-i-1})$. We note that if the coefficient is still insignificant, the reconstructed coefficient before and after coding both will be 0, which leads to no distortion decrease (coding improvement). The coding distortion only decreases if the coefficient becomes significant. Assuming the probability that the coefficient becomes significant is p, and the coefficient is uniformly distributed within the significance interval $(-2^{n-i}, -2^{n-i-1})$ or $(+2^{n-i-1}, +2^{n-i})$, we may calculate the expected coding distortion decrease as

$$D_{\text{prev,SIG}} - D_{\text{post,SIG}} = p\,\frac{9}{4}\,4^{n-i} \tag{6–5}$$

The entropy of the significant identification bit can be calculated as

$$R_{\text{SIG}} = -(1-p)\log_2(1-p) - p\log_2 p + p\cdot 1 = p + H(p),$$

where $H(p) = -(1-p)\log_2(1-p) - p\log_2 p$ is the entropy of the binary symbol with the probability of 1 being p. In (6–5), we account for the one bit which is needed to encode the sign of the coefficient if it becomes significant.

We may then derive the expected rate-distortion slope for the significance identification bit coding as

$$s_{\text{SIG}}(i) = \frac{D_{\text{prev,SIG}} - D_{\text{post,SIG}}}{R_{\text{SIG}}} = \frac{9}{1 + H(p)/p}\,4^{n-i-2}$$

From this and (6–4), we arrive at the following conclusions:

Conclusion 1. The more significant bitplane that the bit is located, the earlier it should be encoded.

A key observation is, within the same coding category (significance identification/refinement), one more significance bitplane translates into 4 times more contribution in distortion decrease per coding bit spent. Therefore, the code-block should be encoded bitplane by bitplane.

Conclusion 2. Within the same bitplane, we should first encode the significance identification bit with a higher probability of significance.

It can be shown that the function $H(p)/p$ increases monotonically as the probability of significance decreases. As a result, the higher probability of significance, the higher contribution of distortion decrease per coding bit spent.

Conclusion 3. Within the same bitplane, the significance identification bit should be encoded earlier than the refinement bit if the probability of significance is higher than 0.01.

It is observed that the insignificant coefficients with no significant coefficients in its neighborhood usually have a probability of significance below 0.01, while insignificant coefficients with at least one significant neighbor usually have a higher probability of significance.

As a result of these three conlusions, the entropy coder in JPEG 2000 encodes the code-block bitplane by bitplane, from the most significant bitplane to the least significant bitplane; and within each bitplane, the bit array is further ordered into three subbitplanes: the predicted significance (PS), the refinement (REF) and the predicted insignificance (PN).

Using the data array in Figure 23 as an example, we illustrate the block coding order of JPEG 2000 with a series of sub-figures in Figure 23. Each sub-figure shows the coding of one subbitplane. The block coding order of JPEG 2000 is as follows:

Step 1: The most significant bitplane, the PN subbitplane of b_1. (See Figure 23(a).)

First, the most significant bitplane is examined and encoded. Since at first, all coefficients are insignificant, all bits in the MSB bitplane belong to the PN subbitplane. Whenever a 1 bit is encountered (rendering the corresponding coefficient non-zero) the sign of the coefficient is encoded immediately afterwards. With the information of those bits that have already been coded and the signs of the significant coefficients, we may figure out an uncertain range for each coefficient. The reconstruction value of the coefficient can also be set, e.g., at the middle of the uncertainty range. The outcome of our sample bit array after the coding of the most significant bitplane is shown in Figure 23(a). We show the uncertainty range and the reconstruction value of each coefficient under columns "value" and "range" in the sub-figure, respectively. As the coding proceeds, the uncertainty range shrinks, and brings better and better representation to each coefficient.

Step 2: The PS subbitplane of b_2. (See Figure 23(b).)

After all bits in the most significant bitplane have been encoded, the coding proceeds to the PS subbitplane of the second most significant bitplane (b_2). The PS subbitplane consists of bits of the coefficients that are not significant, but has at least one significant neighbor. The corresponding subbitplane coding is shown in Figure 23(b). In this example, coefficients w_0 and w_2 are the neighbors of the significant coefficient w_1, and they are encoded in this pass. Again, if a 1 bit is encountered, the coefficient becomes significant, and its sign is encoded right after. The uncertain ranges and reconstruction value of the coded coefficients are updated according to the newly coded information.

Step 3: The REF subbitplane of b_2. (See Figure 23(c).)

The coding then moves to the REF subbitplane, which consists of the bits of the coefficients that are already significant in the past bitplane. The significance status of the coefficients is not changed in this pass, and no sign

of coefficients is encoded.

Step 4: The PN subbitplane of b_2. (See Figure 23(d).)

Finally, the rest of the bits in the bitplane are encoded in the PN subbitplane pass, which consists of the bits of the coefficients that are not significant and have no significant neighbors. Sign is again encoded once a coefficient turns into significant.

Steps 2, 3, and 4 are repeated for the following bitplanes, with the subbitplane coding ordered being PS, REF and PN for each bitplane. The block entropy coding continues until certain criteria, e.g., the desired coding rate or coding quality has been reached, or all bits in the bit array have been encoded. The output bitstream has the embedding property. If the bitstream is truncated, the more significant bits of the coefficients can still be decoded. An estimate of each coefficient is thus obtained, albeit with a relatively large uncertain range.

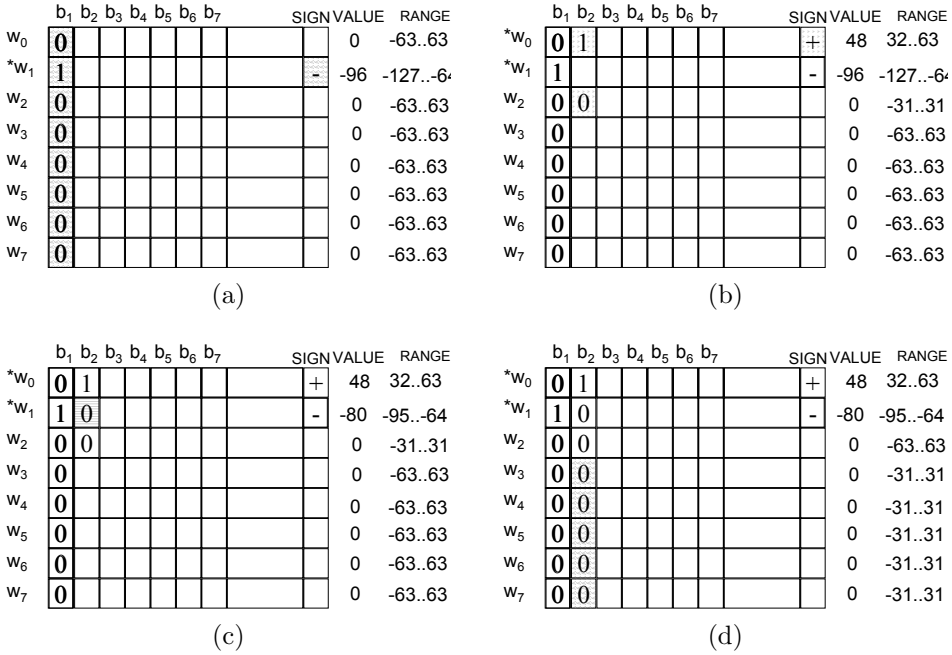

Figure 23. Order of coding: (a) Bitplane b_1, subbitplane PN, then bitplane b_2, subbitplanes (b) PS, (c) REF and (d) PN.

7. The Bitstream Assembler

The embedded bitstream of the code-blocks are assembled by the bitstream assembler module to form the compressed bitstream of the image. As described in section 6, the block entropy coder not only produces an embedded bitstream for each code-block i, but also records the coding rate R_i^k and distortion D_i^k

at the end of each subbitplane, where k is the index of the subbitplane. The bitstream assembler module determines how much bitstream of each code-block is put to the final compressed bitstream. It determines a truncation point n_i for each code-block so that the distortion of the entire image is minimized upon a rate constraint:

$$\min \sum_i D^{n_i} i, \qquad \text{with} \quad \sum_i R^{n_i} i \le B. \tag{7–1}$$

Since there are a discrete number of truncation points n_i, the constraint minimization problem of equation (7–1) can be solved by distributing bits first to the code-blocks with the steepest distortion per rate spent. The process of bit allocation and assembling can be performed as follows:

Step 1: Initialization. We initialize all truncation points to zero: $n_i = 0$.

Step 2: Incremental bit allocation. For each code block i, the maximum possible gain of distortion decrease per rate spent is calculated as

$$S_i = \max_{k>n_i} \frac{D_i^{n_i} - D_i^k}{R_i^k - R_i^{n_i}}.$$

We call S_i the rate-distortion slope of the code-block i. The code-block with the steepest rate-distortion slope is selected, and its truncation point is updated as

$$n_i^{\text{new}} = \arg_{k>n_i} \left(\frac{D^{n_i} i - D^k i}{R_i^k - R^{n_i} i} = S_i \right).$$

A total of $R_i^{n_i^{\text{new}}} - R_i^{n_i}$ bits are sent to the output bitstream. This leads to a distortion decrease of $D_i^{n_i} - D_i^{n_i^{\text{new}}}$. It can be easily proved that this is the maximum distortion decrease achievable for spending $R_i^{n_i^{\text{new}}} - R_i^{n_i}$ bits.

Step 3: Repeat Step 2 until the required coding rate B is reached.

The above optimization procedure does not take into account the last segment problem, i.e., when the coding bits available is smaller than $R_i^{n_i^{\text{new}}} - R_i^{n_i}$ bits. However, in practice, usually the last segment is very small (within 100 bytes), so that the residual sub-optimally is not a big concern.

Following exactly the optimization procedure above is computationally complex. The process can be speeded up by first calculating a convex hull of the R-D slope of each code-block i, as follows:

Step 1: Set $\boldsymbol{S}$ to the set of all truncation points.

Step 2: Set p to the first truncation point in $\boldsymbol{S}$.

Step 3: Do until p is the last truncation point in $\boldsymbol{S}$:

(i) Set k to the next truncation point after p in $\boldsymbol{S}$.

(ii) Set $S_i^k = \dfrac{D_i^p - D_i^k}{R_i^k - R_i^p}$.

(iii) If p is not the first truncation point in $\boldsymbol{S}$ and $S_i^k \geq S_i^p$, remove p from $\boldsymbol{S}$ and move p back one truncation point in $\boldsymbol{S}$; otherwise, set $p = k$.

(iv) [End of current iteration. Restart at step 3(i), unless p is the last truncation point in $\boldsymbol{S}$.]

Once the R-D convex hull is calculated, the optimal R-D optimization becomes simply the search of a global R-D slope λ, where the truncation point of each code-block is determined by:

$$n_i = \arg\max_k \left(S_i^k > \lambda\right)$$

Putting the truncated bitstream of all code-blocks together, we obtain a compressed bitstream associated with each R-D slope λ. To reach a desired coding bitrate B, we just search the minimum λ whose associated bitstream satisfies the rate inequality (7–1). The R-D optimization procedure can be illustrated in Figure 24.

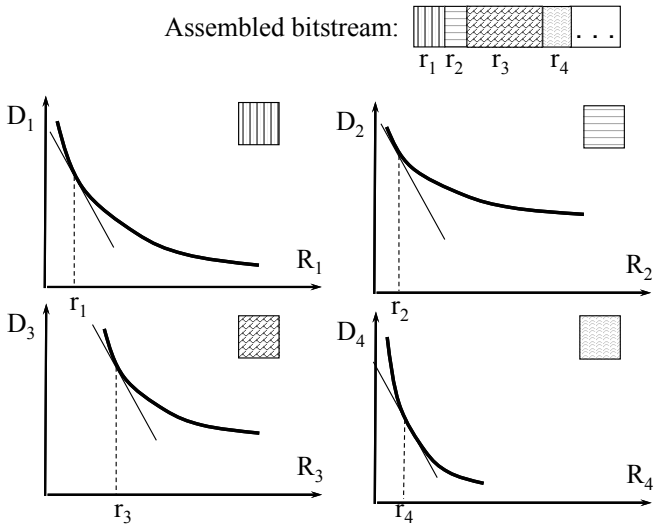

Figure 24. Bitstream assembler: for each R-D slope λ, a truncation point can be found at each code-block. The slope λ should be the minimum slope that the allocated rate for all code-blocks is smaller than the required coding rate B.

To form a compressed image bitstream with progressive quality improvement property, so that we may gradually improve the quality of the received image as more and more bitstream arrives, we may design a series of rate points, $B^{(1)}, B^{(2)}, \ldots, B^{(n)}$. A sample rate point set is 0.0625, 0.125, 0.25, 0.5, 1.0 and 2.0 bpp (bit per pixel). For an image of size 512×512, this corresponds to a compressed bitstream size of 2k, 4k, 8k, 16k, 32k and 64k bytes. First, the global R-D slope $\lambda^{(1)}$ for rate point $B^{(1)}$ is calculated. The first set of truncation point

of each code-block $n_i^{(1)}$is thus derived. These bitstream segments of the code-blocks of one resolution level at one spatial location is grouped into a packet. All packets that consist of the first segment bitstream form the first layer that represents the first quality increment of the entire image at full resolution. Then, we may calculate the second global R-D slope $\lambda^{(2)}$ corresponding to the rate point $B^{(2)}$. The second truncation point of each code-block $n_i^{(2)}$ can be derived, and the bitstream segment between the first $n_i^{(1)}$ and the second $n_i^{(2)}$ truncation points constitutes the second bitstream segment of the code-blocks. We again assemble the bitstream of the code-blocks into packets. All packets that consist of the second segment bitstreams of the code-blocks form the second layer of the compressed image. The process is repeated until all n layers of bitstream are formed. The resultant JPEG 2000 compressed bitstream is thus generated and can be illustrated with Figure 25.

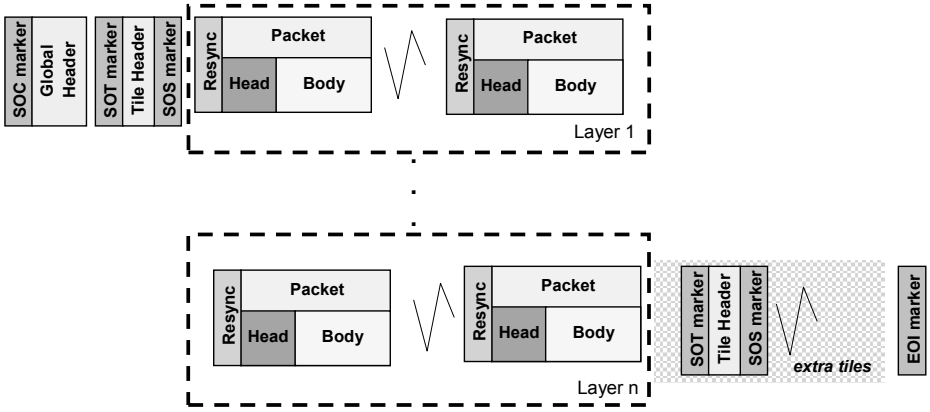

Figure 25. JPEG 2000 bitstream syntax. SOC = start of image (codestream) marker; SOT = start of tile marker; SOS = start of scan marker; EOI = end of image marker.

8. The Performance of JPEG 2000

Finally, we briefly demonstrate the compression performance of JPEG 2000. We compare JPEG 2000 with the traditional JPEG standard. The test image is the "Bike" standard image (gray, 2048×2560), shown in Figure 26. Three modes of JPEG 2000 are tested, and are compared against two modes of JPEG. The JPEG modes are progressive (P-DCT) and sequential (S-DCT) both with optimized Huffman tables [4]. The JPEG 2000 modes are single layer with the bi-orthogonal 9-7 wavelet (S-9,7), six layer progressive with the bi-orthogonal 9-7 wavelet (P6-9,7), and 7 layer progressive with the (3,5) wavelet (P7-3,5). The JPEG 2000 progressive modes have been optimized for 0.0625, 0.125, 0.25, 0.5, 1.0, 2.0 bpp and lossless for the 5×3 wavelet. The JPEG progressive mode uses

a combination of spectral refinement and successive approximation. We show the performance comparison in Figure 27.

Figure 26. Original "Bike" test image.

JPEG 2000 results are significantly better than JPEG results for all modes and all bit-rates on this image. Typically JPEG 2000 provides only a few dB improvement from 0.5 to 1.0 bpp but substantial improvement below 0.25 bpp and above 1.5 bpp. Also, JPEG 2000 achieves scalability at almost no additional

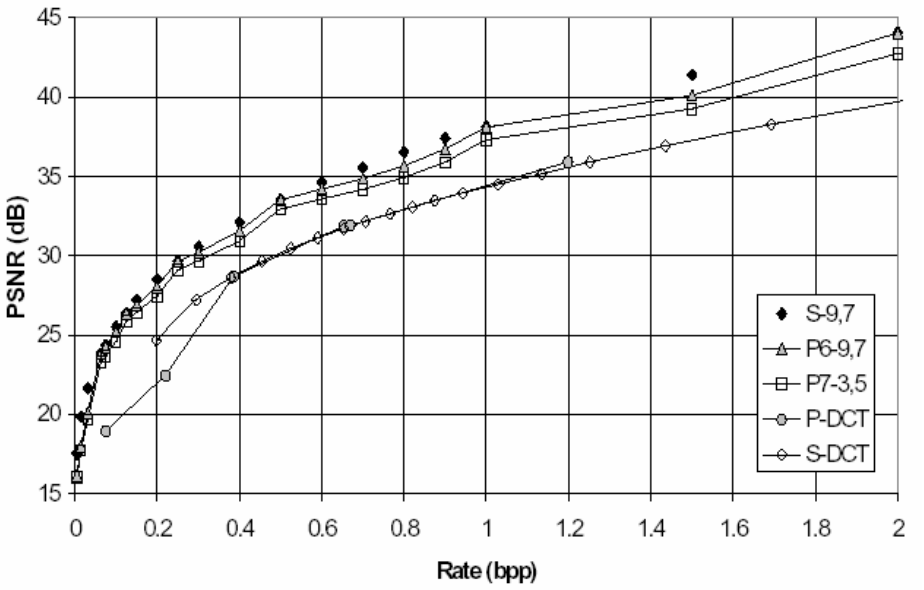

Figure 27. Performance comparison: JPEG 2000 versus JPEG. From [1], courtesy of the authors, Marcellin et al.

cost. The progressive performance is almost as good as the single layer JPEG 2000 without the progressive capability. The slight difference is due solely to the increased signaling cost for the additional layers (which changes the packet headers). It is possible to provide "generic rate scalability" by using upwards of fifty layers. In this case the "scallops" in the progressive curve disappear, but the overhead may be slightly increased.

References

[1] M. W. Marcellin, M. Gormish, A. Bilgin, M. P. Boliek, "An overview of JPEG2000", pp. 523–544 in *Proc. of the Data Compression Conference*, Snowbird (UT), March 2000.

[2] M. W. Marcellin and D. S. Taubman, *Jpeg2000: Image Compression Fundamentals, Standards, and Practice*, Kluwer International Series in Engineering and Computer Science, Secs 642.

[3] ISO/IEC JTC1/SC29/WG1/N1646R, "JPEG 2000 Part I final committee draft, version 1.0", March 2000, http://www.jpeg.org/public/fcd15444-1.pdf

[4] William B. Pennebaker, Joan L. Mitchell, *Jpeg: Still image data compression standard*, Kluwer Academic Publishers, September 1992.

[5] A. Zandi, J. D. Allen, E. L. Schwartz, and M. Boliek, "CREW: compression with reversible embedded wavelets", pp. 212–221 in *Proc. of IEEE Data Compression Conference*, Snowbird (UT), March 1995.

[6] J. Shapiro, "Embedded image coding using zerotree of wavelet coefficients", *IEEE Trans. Signal Processing* **41** (1993), 3445–3462.

[7] S. Mallat, *A wavelet tour of signal processing*, *Academic Press*, 1998.

[8] I. Daubechies, *Ten lectures on wavelets*, second ed., SIAM, Philadelphia, 1992.

[9] C. S. Burrus, R. A. Gopinath and H. Guo, *Introduction to wavelets and wavelet transforms, a primer*, Prentice Hall, Upper Saddle River (NJ), 1998.

[10] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps", *J. Fourier Anal. Appl.* **4**:3 (1998).

[11] W. Sweldens, "Building your own wavelets at home", in: *Wavelets in Computer Graphics*, ACM SIGGRAPH Course Notes, 1996.

[12] C. Valen, "A really friendly guide to wavelets", http://perso.wanadoo.fr/polyvalens/clemens/wavelets/wavelets.html.

[13] J. Li, P. Cheng, and J. Kuo, "On the improvements of embedded zerotree wavelet (EZW) coding", pp. 1490–1501 in *SPIE: Visual Communication and Image Processing*, vol. 2501, Taipei, Taiwan, May 1995.

[14] M. Boliek, "New work item proposal: JPEG 2000 image coding system", *ISO/IEC JTC1/SC29/WG1 N390*, June 1996.

[15] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, Wiley, New York, 1991.

[16] T. M. Cover and J. A. Thomas, "Elements of information theory: online resources", http://www-isl.stanford.edu/~jat/eit2/index.shtml.

[17] ISO/IEC JTC1/SC29/WG1 N505, "Call for contributions for JPEG 2000 (ITC 1.29.14, 15444): image coding system", March 1997.

[18] J. H. Kasner, M. W. Marcellin and B. R. Hunt, "Universal trellis coded quantization", *IEEE Trans. Image Processing* **8**:12 (Dec. 1999), 1677–1687.

[19] D. Taubman, "High performance scalable image compression with EBCOT", *IEEE Trans. Image Processing* **9**:7 (July 2000), 1158–1170.

[20] M. Antonini, M. Barlaud, P. Mathieu and I. Daubechies, "Image coding using wavelet transform", *IEEE Trans. Image Processing*, **1**:2 (Apr. 1992), 205–220.

[21] J. Li and S. Lei, "An embedded still image coder with rate-distortion optimization", *IEEE Trans. Image Processing* **8**:7 (July 1999), 913–924.

[22] ISO/IEC JTC1/SC29/WG1 N1359, "Information technology — coded representation of picture and audio information — lossy/lossless coding of bi-level images", 14492 Final Committee Draft, July 1999.

[23] W. Pennebaker, J. Mitchell, G. Langdon, and R. Arps, "An overview of the basic principles of the *q*-coder adaptive binary arithmetic coder", *IBM J. Res. Develop* **32**:6 (1988), 717–726.

[24] Ian H. Witten, Radford M. Neal, and John G. Cleary, "Arithmetic coding for data compression," *Communications of the ACM* **30**:6 (1987), 520–540.

[25] J. Li and H. Sun, "A virtual media (Vmedia) interactive image browser", *IEEE Trans. Multimedia*, Sept. 2003.

[26] K. R. Rao and P. Yip, Rao, "Discrete cosine transform: algorithms, advantages, applications". Academic Press, Boston, 1990.

[27] S. Mallat, *A wavelet tour of signal processing*, Academic Press, 1998.

[28] P. P. Vaidyanathan, *Multirate systems and filter banks*, Prentice-Hall, Englewood Cliffs (NJ), 1993.

[29] R. Gonzalez and R. Woods, *Digital image processing*, Addison-Wesley, Reading (MA), 1992.

[30] K. R. Rao and D. F. Elliott, *Fast transforms: algorithms, analyses and applications*, Academic Press, New York, 1982.

[31] R. Rosenholtz and A. B. Watson, "Perceptual adaptive JPEG coding", pp. 901–904 in *Proc. IEEE International Conference on Image Processing*, Lausanne, Switzerland, 1994.

[32] G. V. Auwera, A. Munteanu, and J. Cornelis, "Evaluation of a quincunx wavelet filter design approach for quadtree-based embedded image coding", pp. 190–194 in *Proc. of the IEEE International Conference on Image Processing (ICIP)*, Vancouver, Canada, Sept. 2000.

Jin Li
Microsoft Research
Communication Collaboration and Signal Processing
One Microsoft Way, Bld. 113/3161
Redmond, WA 98052
jinl@microsoft.com