Image Compression - the Mechanics of the JPEG 2000

Jin Li

Microsoft Research, Signal Processing, One Microsoft Way, Bld. 113/3374, Redmond, WA 98052
Email: jinl@microsoft.com

ABSTRACT

We briefly review the mechanics in the coding engine of the JPEG 2000, a start-of-the-art image compression system. The transform, entropy coding and bitstream assembler modules are examined in details. Our goal is to enable the readers to have a good understanding of the modern scalable media compression technologies without being swarmed by the details.

Keywords: Image compression, JPEG 2000, transform, wavelet, entropy coder, sub-bitplane entropy coder, bitstream assembler.

1. INTRODUCTION

Compression is a process that creates a compact data representation for storage and transmission purposes. Media compression usually involves the use of special compression tools because media is different from the generic data. Generic data file, such as a computer executable program, a Word document, must be compressed losslessly. Even a single bit error may render the data useless. On the other hand, distortion is tolerable in the media compression process; because it is the content of the media that is of paramount importance, rather than the exact bit of the media. Since the size of the original media, whether it is an image, a sound clip, or a movie clip, is usually very large, it is essential to compress the media at a very high compression ratio. Such high ratio media compression is usually achieved through two mechanisms: a) to ignore the media components that are less perceptible, and b) to use entropy coding to explore information redundancies exist in the source data.

Different applications may have different requirement of the compression ratio and tolerance of the compression distortion. A publish application may require a compression scheme with very little distortion, while a web application may tolerate relatively large distortion in exchange of a smaller compressed media. Recently, a category of media compression algorithms termed scalable compression emerges to offer the ability to trade between the compression ratio and distortion even after the compressed bitstream has been generated. In scalable compression, a media is first compressed into a master bitstream, where a subset of the master bitstream may be extracted to form an application bitstream with a higher compression ratio. With scalable compression, a compressed media can be quickly tailored for applications with vastly different compression ratio and quality requirement, which is especially useful in media storage and transmission.

In the following part of the paper, we use image compression, and in particular the JPEG 2000 image compression standard, to illustrate important mechanics in 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. The JPEG 2000 standard and its development history are briefly reviewed in Section 3. The transform, quantization, entropy coding, and bitstream assembler modules are examined in details from Section 4-7. Our goal is to describe the key mechanics of the JPEG 2000 coding engine so that the readers may get a good understanding of the standard without being swarmed by the details. For the readers who are further interested, they may refer to [1][2][3].

2. IMAGE COMPRESSION

Digital images are used every day now. 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 each of the data point $x(i,j)$ is referred as a pixel. For the gray image, each pixel is of an intensity value G. For color image, each pixel consists of 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 most, the underlying 2D data array may undergo big changes, and still convey the content to the user. An example is shown in Figure 1, where the original image *Lena* is shown at left as a 2D array of 512x512. Operations may be performed on the original image so that it is suited for a particular application. For example, when the display space is tight, we may subsample the original image to a smaller image of size 128x128, as shown in the upper-right of Figure 1. Another possible operation is to extract a rectangular region of the image starting from coordinate (256,256) to (384,384), as shown at the middle-right. The entire image may also be compressed into a compact bitstream representation, e.g. by JPEG, as shown in the bottom-right. In each case, the underlying 2D data array is changed tremendously. Nevertheless, the primary content of the image, the face of the girl remains legible to the user.

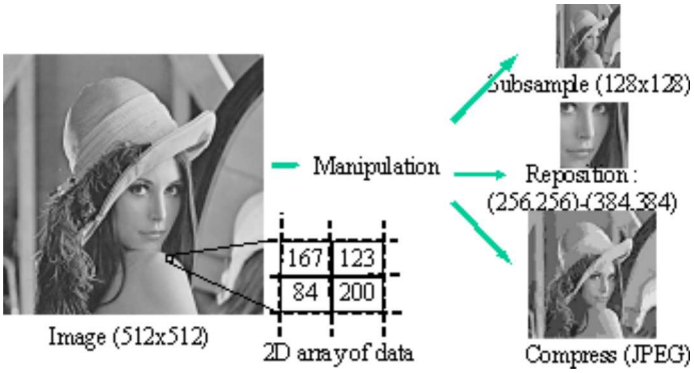

Figure 1 Digital image and image manipulation.

Among the operations, the compression creates a compact representation of the image data. It is an essential operation for image storage and transmission. The JPEG 2000, a next generation image compression standard, is a highly scalable image compression algorithm. From the compressed JPEG 2000 bitstream, it is possible to extract a subset of the bitstream that decodes to an image of lower quality (with higher compression ratio), lower-resolution, and/or smaller spatial region. In other words, instead of manipulating the image in the space domain as shown in Figure 1, we may manipulate the image directly on the compressed domain, and form a new bitstream that suits the application better.

Scalable image compression has important applications in image storage and delivery. Let us first examine the application of digital photography. Right now, digital cameras on the market 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, image quality and resolution setting has to be determined before the shooting of photos. This leads to painful trade off between removing lovely photos to make space for new exciting shots, and shooting photos with poor quality and resolution setting. With scalable image compression, it is possible for the digital cameras to adjust the image quality and resolution after the photo is shot. The camera may always shot images at the highest possible quality and resolution setting. When the camera memory is used up, the compressed bitstream of existing shots may be truncated to smaller size to leave room for the upcoming shots. Through dynamically trading between the number of images and the image quality, the precious camera memory is not wasted, and the quality of the shot is maximized.

Another important application of the scalable image compression is for the web browsing. As the resolution of the digital camera and the digital scanner becomes higher and higher, high-resolution digital image becomes a reality. It is very pleasing to view a high-resolution image, however, it is equally painful to wait for the long compressed bitstream to be delivered over the web. Before scalable image compression technology is available, it is common practice to generate and put on the web multiple copies of the compressed bitstream with different spatial region, resolution and compression ratio. However, this produces multiple copies of the bitstream for the same media file, and causes headaches in media management and wastes valuable server space. A better solution is to put a scalable master bitstream of the compressed image on the server. During image browsing, the user may specify a region of interest (ROI) with a certain spatial and resolution constraint. The browser 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 are arrived. With scalable image compression, it is possible to browse large image quickly and on demand over the Internet, as shown with the Vmedia project [16].

3. THE JPEG 2000 STANARD

We first briefly review the history of the development of the JPEG 2000 standard. In the early 1990s, a number of new image compression algorithms, such as CREW (compression with reversible embedded wavelets) [5] and EZW (embedded zerotree wavelet) [6], emerged to provide not only superior compression performance, but also a new set of features unseen before. Based on industrial demand, the JPEG 2000[1] project was approved as a new work item in 1996. A call for technical contributions was issued in Mar. 1997[10]. The first evaluation is performed in November 1997 in Sydney, Australia, where 24 algorithms were submitted and evaluated. Based on the evaluation, it was decided to create a JPEG 2000 “verification model” (VM) which would lead to a reference implementation for the following standard process. The first VM (VM0) is based on the wavelet/trellis coded quantization (WTCQ) algorithm submitted by SAIC and the University of Arizona (SAIC/UA)[11]. 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. The JPEG 2000 finally became an “International standard” (IS) in December 2000.

JPEG 2000 standardizes the decoder and the bitstream syntax. Nevertheless, information on the encoder implementation is provided to assure a reasonable performance encoder. We choose to describe the JPEG 2000 from the encoder perspective since it can be more easily understood.

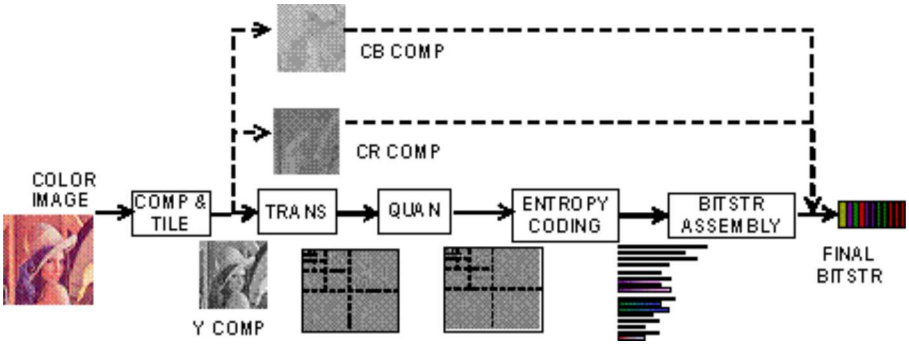

Figure 2 Operation flow of the JPEG 2000 standard.

The operation flow of a typical JPEG 2000 encoder can be shown in Figure 2. The first module is component and tile separation, whose function is to cut the image into manageable chunks and to decorrelate the color components. Huge original images, e.g., aero-photography images, are separated into spatially non-overlapping tiles of equal size. For multi-component (color) images, a component transform is performed to decorrelate the components. For example, a color image with RGB (red, green and blue) components can be transformed to the YCrCb (Luminance, Chrominance red and Chrominance blue) or RCT (reversible component transform) component space. Each tile of each component is then processed separately. The data are first transformed into the wavelet domain, and are then quantized. After that, the quantized coefficients are regrouped to facilitate localized spatial and resolution access. Each subband of the quantized coefficients is 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 into an embedded bitstream with a recorded rate-distortion curve. The embedded bitstream of the code-blocks are assembled into packets, each of which represents a quality increment of one resolution level 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 is one quality increment of the entire image at full resolution. The final JPEG 2000 bitstream may consist of multiple layers.

In the following, we examine the transform, quantization & packet formation, code-block entropy coding and bitstream assembler modules in details.

4. WAVELET TRANSFORM

4.1. Introduction

Most existing high performance image coders in application are transform based coders, and this exists for a very good reason: the transform coder provides good compression performance with reasonable complexity. In the transform coder, the image pixels are converted from the space domain to the transform domain through a linear orthogonal or bi-orthogonal transform. The transform decorrelates the pixels and compacts their energy into a small number of coefficients, results in efficient coding of the transform coefficients. Since most of the energy is compacted into a few large transform coefficients, we may adopt entropy coding scheme that easily locates those coefficients and encodes them. Because the transform coefficients are 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 of an image block can be derived through Karhunen-Loeve (K-L) decomposition. However, the K-L transform lacks fast computation, and the transform is content dependent. It is thus not suited for the compression purposes. Popular transforms used in image coding include block based transform, such as DCT, and wavelet transform. DCT transform of the image block can be quickly computed. It achieves very good energy compaction and coefficient decorrelation. The widely used image compression standard JPEG is a DCT based coding algorithm. However, the DCT is calculated on block of pixels independently, therefore, coding error causes discontinuity between the blocks which leads to annoying blocking artifact. On the contrary, the wavelet transform operates on the entire image, (or a tile of a component in the case of large color image). It offers better energy compaction than the DCT, and no blocking artifact after coding. Moreover, the wavelet transform decomposes the image into an L-level dyadic wavelet pyramid, as shown in Figure 3. The resultant wavelet coefficient can be easily scaled in resolution: by not using the wavelet coefficients at the finest M-levels, we may reconstruct an image that is 2^M times smaller than the original one. The multi-resolution capability of the wavelet transform lends it ideally to scalable image coding.

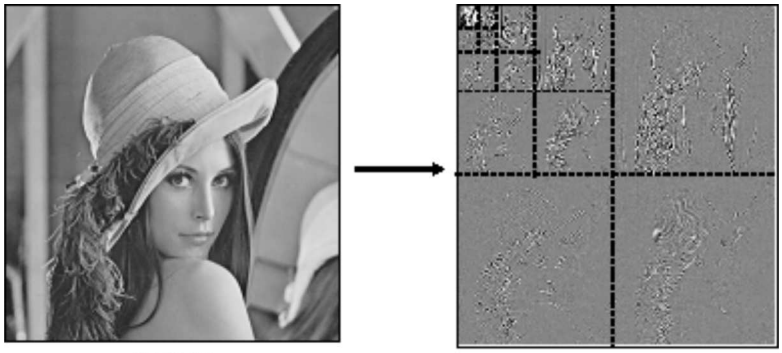

Figure 3 Wavelet transform.

4.2. Wavelet Transform by Lifting.

Wavelet is a signal representation that the low pass coefficients represent slow changing long data segment, and the high pass coefficients represent localized change in short data segment. It provides an elegant framework in which both short term anomaly and long term trend can be analyzed on equal footing. For theory of wavelet and multi-resolution analysis, the reader is referred to [7].

The framework of a one-dimensional wavelet transform can be shown in Figure 4. We decompose the signal *x(n)* through a low and high pass analysis FIR(finite impulse response) filter, and subsample the filtering result by a factor of 2. To reconstruct the original signal, the low and high pass coefficients are upsampled by a factor of 2 and pass through another low and high pass synthesis FIR filter pair. Although IIR (infinite impulse response) filter can also be used, its infinite response leads to infinite data expansion, which is an undesired drawback. For the filter bank in Figure 4, the condition for perfect reconstruction is shown as [8]:

$$\begin{aligned} G'(z)G(z^{-1}) + H'(z)H(z^{-1}) &= 2 \\ G'(z)G(-z^{-1}) + H'(z)H(-z^{-1}) &= 0 \end{aligned}. \qquad (1)$$

Note it is necessary to reverse the time of the analyzing filters, otherwise, it would not be possible to derive a non-delayed, perfectly reconstructed signal. If the conditions for perfect reconstruction are fulfilled then all the aliasing caused by the subsampling will be canceled at the reconstruction.

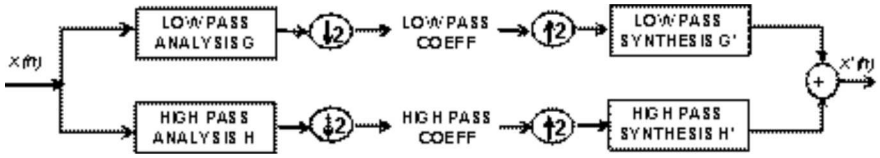

Figure 4 Framework of a one dimensional wavelet transform.

A wavelet transform implemented in the fashion of Figure 4 is called convolutionary implementation. We note that the signal is first convoluted with the filter bank and then subsampled. In other words, only half the samples are kept, with the other half of the filtered samples thrown away. Clearly this is not efficient, and it would be better to do the subsampling before the filtering operation. This leads to an alternative implementation of the wavelet transform termed *lifting* approach. It turns out that all FIR wavelet filters can be factored into lifting step. We refer again to [8] for theoretical details. The default wavelet filters used in the JPEG 2000 is the bi-orthogonal 9-7 wavelet. The lifting step associated with the wavelet can be shown in Figure 5. The original data, $\cdots x_0, x_1, \cdots, x_8 \cdots$, are input from the right. Assuming that the original data are infinite in length, to perform the wavelet transform, we first apply the first stage lifting, which updates the odd index data point $x_1, x_3, \cdots$ as follows:

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

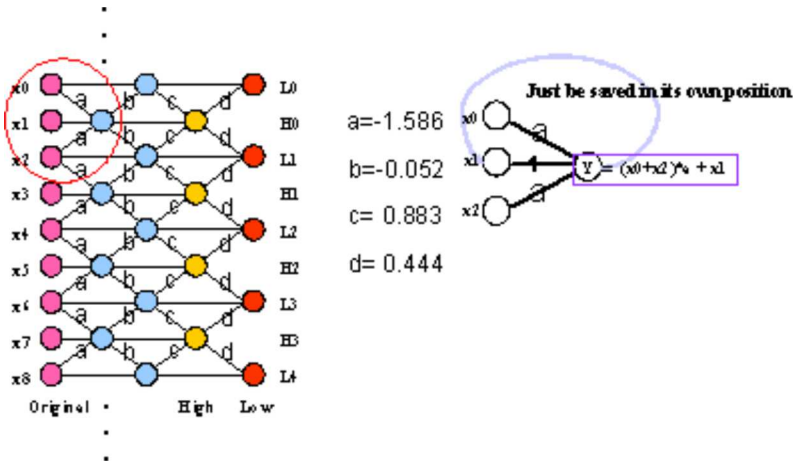

Figure 5 Single-scale one dimension wavelet lifting (bi-orthogonal 9-7 wavelet).

where a and x'_{2n+1} is the first stage lifting parameter and outcome, respectively. After all the odd index data points are calculated, we may perform the second stage lifting:

$$x''_{2n} = x_{2n} + b * (x'_{2n-1} + x'_{2n+1}), \quad (3)$$

where we refer the second stage lifting parameter and outcome as b and x''_{2n}, respectively. The third and fourth stage lifting can be performed in similar fashions:

$$H_n = x'_{2n+1} + c * (x''_{2n} + x''_{2n+2}), \quad (4)$$

$$L_n = x''_{2n} + d * (H_{n-1} + H_n). \quad (5)$$

where H_n and L_n are the resultant high and low pass coefficients. The value of the lifting parameters a, b, c *and* d are shown in Figure 5.

With lifting wavelet, the transform can be easily inversed. Since the basic calculation of the lifting step in Figure 5 is:

$$Y = X + d * (L + R), \quad (6)$$

the inverse can be easily derived as:

$$X = Y + (- d) * (L + R), \quad (7)$$

Therefore, the inverse lifting of the 9-7 bi-orthogonal wavelet can be derived in Figure 6.

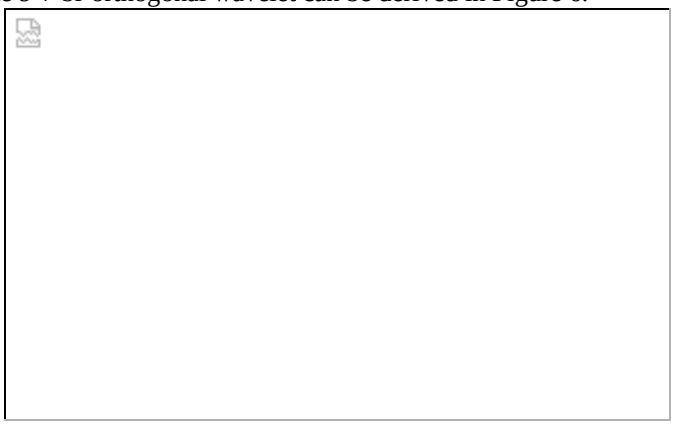

Figure 6 Forward and inverse lifting (9-7 floating wavelet).

Since the actual data in an image transform is finite in length, boundary extension becomes a crucial part of every wavelet decomposition scheme. For symmetrical odd-tap filter (the bi-orthogonal 9-7 wavelet is in this category), symmetrical boundary extension can be used. The data are symmetrically reflected along the boundary points, 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 is shown in Figure 7. Because both the extended data and the lifting structure are symmetrical, all the intermediate and final results of the lifting are also symmetrical with regard to the boundary points. With such an observation, we do not even need to actually extend the data. It is sufficient to double the lifting parameters of the branches that are pointing toward the boundary, as show in the middle of Figure 7. Thus, the boundary extension can be performed without additional computational complexity. Using the principle shown in equation (6) and (7), we may easily derive the inverse lifting structure, as shown in the right of Figure 7. Note again that the parameters of branches that are pointing toward the boundary points are doubled.

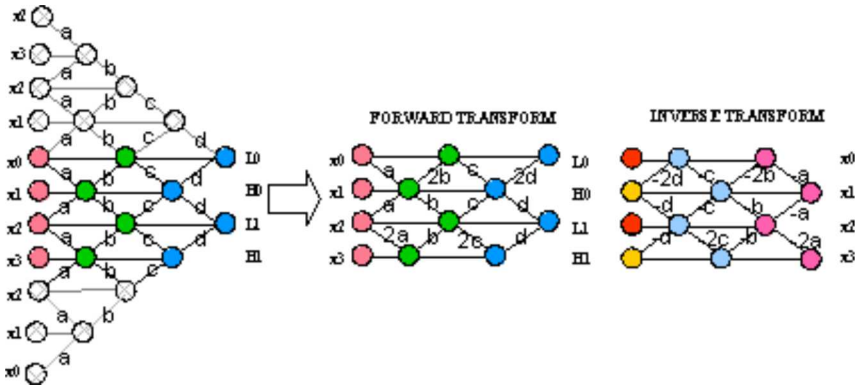

Figure 7 Symmetrical boundary extension of bi-orthogonal 9-7 wavelet.

4.3. Two-Dimensional Wavelet Transform

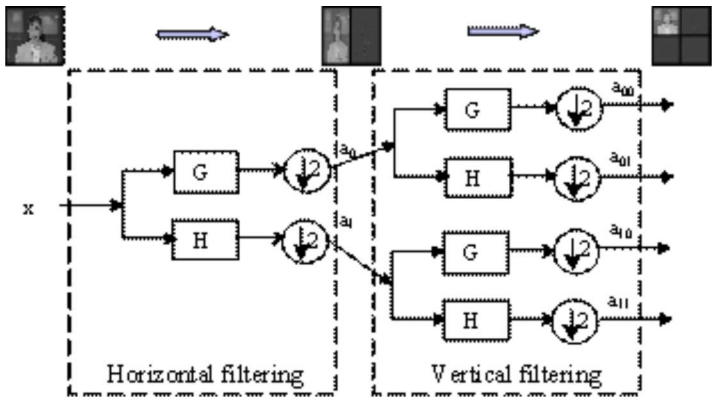

Figure 8 2D wavelet transform.

To apply wavelet transform to a 2D array, such as an image, it is a common practice to apply the wavelet transform in the horizontal and vertical direction separately. The approach is called *2D-separable* wavelet transform. Although it is possible to design a 2D *non-separable* wavelet, which directly uses the 2D filtering and subsampling operation, the computational complexity is greatly increased, while the gain on additional energy compaction is very limited. A sample one scale 2D-separate wavelet transform is shown in Figure 8. The 2D data array of the image is first filtered in the horizontal direction, which results in two subbands, - a horizontal low and a horizontal high pass subband. Each subband then passes through a vertical wavelet filter. The image is thus decomposed into four subbands, - subband 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). Because the wavelet transform is a linear transform, we may switch the order of the horizontal and vertical wavelet filter, and still reach the same effect. A multi-scale dyadic wavelet pyramid shown in Figure 3 can be obtained by further decomposing the subband LL with another 2D wavelet.

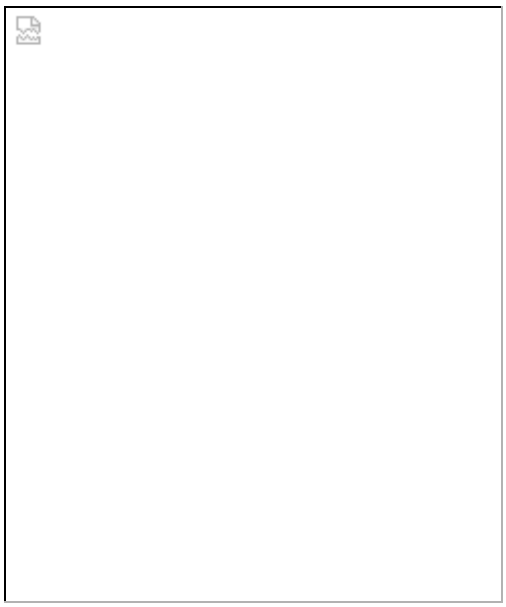

Figure 9 Line based lifting wavelet (bi-orthogonal 9-7 wavelet)

A trick in implementing the 2D wavelet transform is line based lifting. To avoid buffering the entire 2D data array during the vertical wavelet lifting operation, it is possible to compute the vertical low and high pass results one line at a time. The

concept can be shown in Figure 9, with each circle represents an entire line of image. It is described below as:

Step 1. Initialization phase 0.

Three lines of coefficients x_0, x_1 and x_2 are read in. Two lines of lifting operations are performed, and intermediate results x'_1 and x''_0 are generated.

Step 2. Initialization phase 2.

Two additional lines of coefficients x_3 and x_4 are read in. Four lines of lifting operations are performed. The outcome 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 read in two lines of coefficients, perform four lines of lifting operations, and generate one line of low and high pass coefficients.

Step 4. Flushing.

When the bottom of the image is reached, symmetrical boundary extension is again performed to correctly generate the final low and high pass coefficients.

For 9-7 bi-orthogonal wavelet, with line based lifting, only 6 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 & PARTITIONING

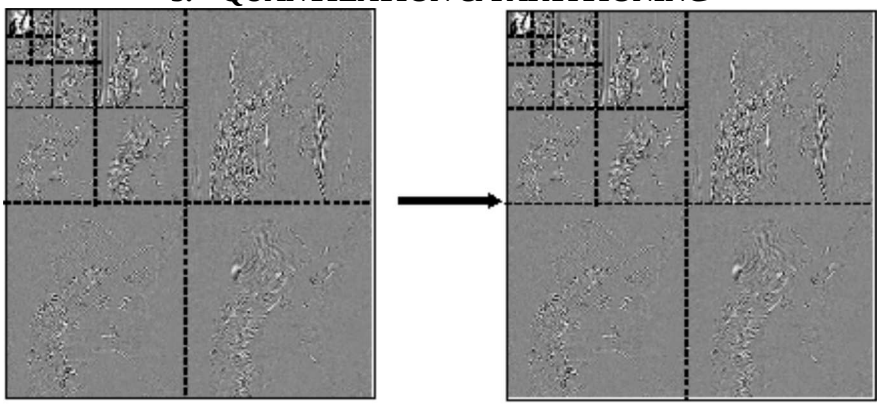

Figure 10 Quantization module.

After the wavelet transform, all wavelet coefficients are uniformly quantized by the following:

$$w_{m,n} = sign(s_{m,n}) \left\lfloor \frac{|s_{m,n}|}{\delta} \right\rfloor, \quad (8)$$

where $s_{m,n}$ is the transform coefficients, $w_{m,n}$ is the quantization result, δ is the quantization step size, *sign(x)* returns the sign of coefficient *x*, and $\lfloor x \rfloor$ operation obtains the largest integer that is less or equal than *x*. The effect of quantization can be shown in Figure 10. The quantization process converts the wavelet coefficients from floating number into integer number. In a non-embedded image coding scheme such as the JPEG, the quantization process determines the allowable distortion of the transform coefficients, as the quantization result is losslessly encoded. However, in JPEG 2000, the quantized coefficients are embedded coded, thus additional distortion can be introduced in the following entropy coding steps. The main functionality of the quantization module is thus to map the coefficients 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 is thus rather fine, e.g., $\delta=1/128$.

The quantized coefficients are partitioned into packets. Each subband is divided into non-overlapping rectangles of equal size. Three rectangles corresponding to the same space location at the three directional subbands HL, LH, HH of each resolution level comprise a packet. 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 operating the entropy coder on 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, thus greatly improves the encoding and decoding speed. In JPEG 2000, the default size of a code-block is 64x64.

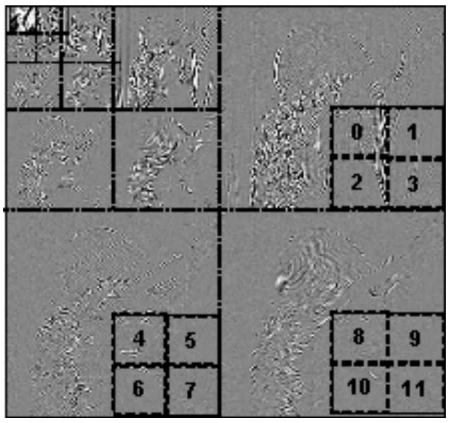

Figure 11 A sample partition and code-blocks.

A sample partition and code-blocks are shown in Figure 11. 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 12 64x64 code-blocks, which are shown as numbered blocks in Figure 11.

6. BLOCK ENTROPY CODING

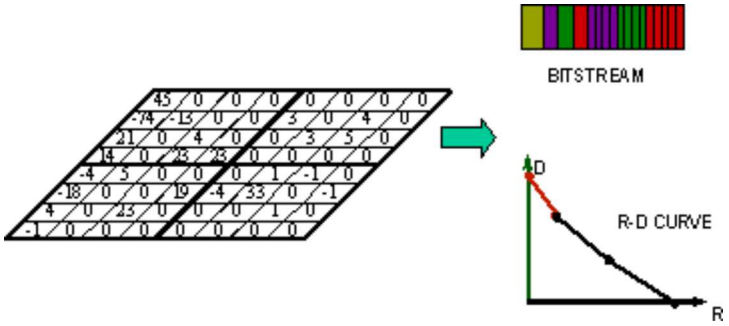

Figure 12 Block entropy coding.

Each code-block is then independently encoded through a sub-bitplane entropy coder. The output bitstream of the entropy coder has the embedding property that the bitstream can be truncated at any point and still be decodable. In JPEG 2000, the output of the block entropy coder not only includes the embedded bitstream, but also includes a rate-distortion (R-D) curve that measures the distortion of the code-block at certain rate points, as shown in Figure 12. The entropy coder also measures 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 sub-bitplane. The coding distortion is obtained by measuring the distortion between the original coefficient and the reconstructed coefficient at the same instance.

To explain the sub-bitplane entropy coder, we examine three key parts of the coder: the coding order, the context, and the arithmetic MQ-coder.

6.1. Embedded coding and the context

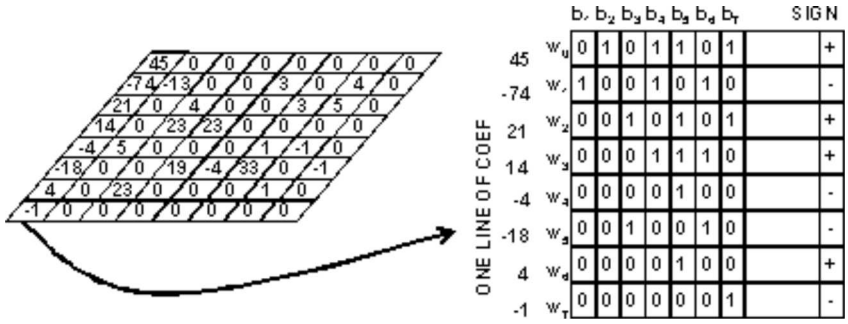

Figure 13 Coefficients and binary representation.

Let each quantized coefficient $w_{m,n}$ be represented in the binary form as:

$$\pm b_1 b_2 \cdots b_n \qquad (9)$$

where b_1 is the most significant bit (MSB), and b_n is the least significant bit (LSB), and ± represents the sign of the coefficient. Let a group of bits at the same significance level of all coefficients be a bitplane. For example, bit b_1 of all coefficients forms the most significance bitplane of the code-block. By coding the more significant bits of all coefficients first, and coding the less significant bits later, the output compressed bitstream is said to have the embedding property, as a lower rate bitstream can be obtained by truncating a higher rate bitstream, which results in a partial decoding of all coefficients. A sample binary representation of the coefficient can be shown in Figure 13. Since representing bits in a 2D block results in a 3D bit array that is very difficult to draw, we only show the binary representation of a column of coefficients as a 2D bit array in Figure 13. However, keep in mind that the true bit array in a code-block is 3D.

The bit array that represents the quantized coefficient consists both bits and sign of the coefficient. Moreover, the bits in the bit array are statistically different. Let b_M be a bit in a coefficient x which is to be encoded. 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 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. We distinguish between significance identification and refinement bits because the significance identification bit has a very high probability of '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 13, the significance identification and the refinement bits are shown with different shades in Figure 14.

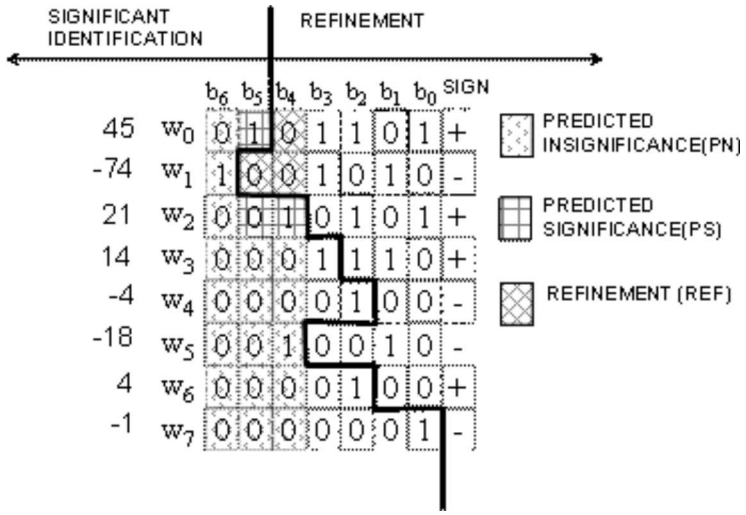

Figure 14 Embedded coding of bit array.

The significant identification bits, refinement bits and signs are not statistically equal among themselves either. 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 anomaly usually causes a cluster of large wavelet coefficients in the neighborhood as well. To account for such statistical difference, we entropy encode the significant identification bits, refinement bits and signs with context, each of which is a number derived from already coded coefficient in the neighborhood of the current coefficient. Such technique is called context adaptive entropy coding, as shown in Figure 15.

Since the context 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 statistic properties, so that the following entropy coder can measure the probability distribution within each context and efficiently compress the bits.

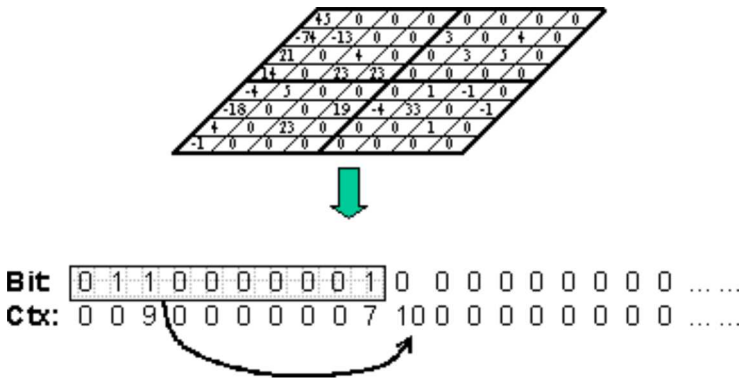

Figure 15 Coding bits and contexts. The context is generated from information that can be derived from already coded bits.

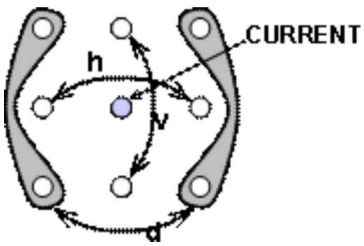

Figure 16 Neighborhood of the current coefficient.

Table 1 Context for the significance identification coding.

LH Subband (also used for 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

We first discuss context for the significant identification coding, which can be tabulated in Table 1. The context is determined in the following way:

Step 1. Neighborhood statistics.

For each bit of the coefficient, the number of significant horizontal, vertical and diagonal neighbors are measured as *h*, *v* and *d*, as shown in Figure 16.

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 through one of the tables shown in Table 1. A total of nine context categories are used in 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.

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 of sign coding can then be calculated according to Table 2.

Table 2 Context and the expected sign for sign coding.

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

The context of the refinement coding bits can be tabulated in Table 3.

Table 3 Context for the refinement bit.

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

6.2. MQ-coder: Context dependent entropy coder.

Using the context, the binary bits and signs in the bit array is separate into class of bits, each of which is assumed to be independently identical distributed (i.i.d). Let the number of classes be *N*, and let there be n_i bits in class *i*, with the probability of the bits to take value '1' be p_i. Shannon's information theory indicates that the entropy of the bitarray is:

$$H = \sum_{i=0}^{N-1} n_i \left[-p_i \log_2 p_i - (1-p_i) \log_2 (1-p_i) \right]. \quad (10)$$

The task of the entropy coder is thus to convert these pairs of bit and context into a compact bitstream representation with length as close to the Shannon limit as possible. Several coders are available for such a task. The coder used in the JPEG 2000 is the MQ-coder [15]. In the following, we focus the discussion on three key aspects of the MQ-coder: the general arithmetic coding theory, fixed point arithmetic implementation and probability estimation.

Figure 17 Input and output of the MQ-coder.

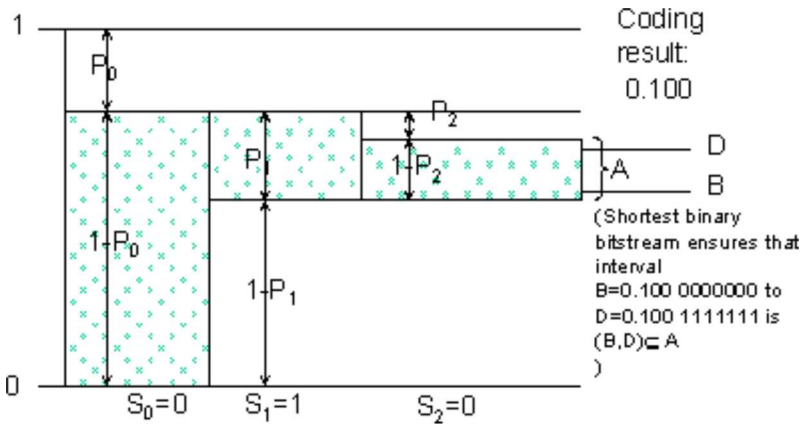

Figure 18 Probability interval subdivision.

6.2.1 General arithmetic coder.

MQ-coder belongs to a category of entropy coders that are referred as the arithmetic coders. The basic theory of the arithmetic coder is based on the recursive probability interval subdivision. Let $S_0S_1S_2\cdots S_n$ 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'. The encoding process can be described as follows:

Step 1. Initialization.

Let the initial probability interval be (0.0, 1.0). 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=0.0* and *A=1.0*.

Step 2. Probability interval subdivision.

The binary symbols $S_0S_1S_2\cdots S_n$ are encoded sequentially. For each symbol S_i, the probability interval *(C, C+A)* is subdivided into two sub-intervals *(C, C+A·(1-P_i))* and *(C+A·(1-P_i), C+A)*. Depending on whether the symbol S_i is '1', one of the two sub-intervals is selected. That is:

$$\begin{cases} C = C, A = A(1-P_i) & S_i = 0 \\ C = C + A\cdot(1-P_i), A = A\cdot P_i & S_i = 1 \end{cases} \quad (11)$$

Step 3. Bitstream output.

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

$$\begin{cases} Upperbound & D = 0.k_1k_2\cdots k_m111\cdots \\ Lowerbound & B = 0.k_1k_2\cdots k_m000\cdots \end{cases} \quad (12)$$

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 arithmetic coding process. The entire binary symbol strings $S_0S_1S_2\cdots S_n$ can thus be recovered from the compressed representation. It can be shown that it is possible to find a final coding bitstream with length:

$$m \le \lceil -\log_2 A \rceil + 2, \quad (13)$$

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

$$H = \sum_{S_0S_1\cdots S_n} -A\log_2 A. \quad (14)$$

The arithmetic coder thus encodes the binary string within 3 bits of its entropy limit, no matter how long the symbol string is. This is very efficient.

6.2.2 Fixed point arithmetic operation.

The generic arithmetic coder assumes infinite precision arithmetic operations, which is unrealistic in the real applications. 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:

$$\begin{cases} C = 1.5\cdot[0.k_1k_2\cdots k_L] + 2^{-L}\cdot 1.5\cdot C_x, \\ A = 2^{-L}\cdot 1.5\cdot A_x, \end{cases} \quad (15)$$

where *L* is a normalization factor determines the magnitude of the interval *A*, A_x and C_x are fixed-point integers representing values between *(0.75,1.5)* and *(0.0, 1.5)*, respectively. Bits $k_1k_2\ldots k_m$ are the output bits that are already been determined (in reality, certain carry over 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 (11) can then be calculated as:

$$\begin{cases} C_x = C_x, A_x = A_x - P_i & S_i = 0 \\ C = C + A_x - P_i, A_x = P_i & S_i = 1 \end{cases}, \quad (16)$$

which can be done quickly without any multiplication operation. The compression performance suffers a little bit, 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 3%, which is well worth the saving in implementation complexity.

6.2.3 Probability estimation.

In the arithmetic coder, it is necessary to estimate the probability of '1' (P_i) of each binary symbol S_i. This is where the context comes into play. As mentioned in Section 6.1, the input binary strings $S_0S_1S_2\cdots S_n$ are accompanied with a context string which classifies the bits into different categories. Within each context, it is assumed that the symbols are independently identically distributed (i.i.d.). We may then estimate the probability of the symbol within each context through observing 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 of symbols to take value '1' in context i through Bayesian estimation as:

$$p_i = \frac{o_i + 1}{n_i + 2}. \quad (17)$$

In the MQ-coder, the 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.

6.3. MQ-coder: Context dependent entropy coder.

Because the embedded bitstream of the code-block may be truncated, the coding order is of paramount importance. It turns out that the optimal coding order is first to encode those bits with the steepest rate-distortion slope, i.e., the largest coding distortion decrease per bit spent [14]. Just as the statistic properties of the bits are different in the bit array, their contribution of the coding distortion decrease per bit is also different.

Let us consider a bit b_i in the ith most significant bitplane, where there are a total of n bitplane. If the bit is a refinement bit, before 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 $(A+2^{n-i}, A+2^{n-i-1})$. Let the bit be equally probable to be '0' and '1', the entropy of the refinement bit is:

$$R_{REF} = 1bit. \quad (18)$$

If we further assume that the value of the coefficient is uniformly distributed in the uncertainty interval, we may calculated the expected distortion before and after the coding as:

$$D_{prev,REF} = \int_{A}^{A+2^{n-i}} (x - A - 2^{n-i-1})^2 dx = \frac{1}{12} 4^{n-i},$$
$$D_{post,REF} = \frac{1}{12} 4^{n-i-1}. \quad (19)$$

It can then be easily derived that the expected rate-distortion slope for the refinement bit is:

$$s_{REF}(i) = \frac{D_{prev,REF} - D_{post,REF}}{R_{REF}} = \frac{\frac{1}{12}4^{n-i} - \frac{1}{12}4^{n-i-1}}{1} = 4^{n-i-2}. \quad (20)$$

If the bit is a significance identification bit, 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 assume that the probability that the coefficient becomes significant is p, the average number of bits to encode the significant identification bit can be calculated as:

$$R_{SIG} = -(1-p)\log_2(1-p) - p\log_2 p + p \cdot 1 = p + H(p). \quad (21)$$

where $H(p)$ is the entropy of the binary symbol with the probability of '1' be p. Note in (21) we account for the one bit which is needed to encode the sign of the coefficient if it becomes significant. We note that if the coefficient is still insignificant, the reconstructed coefficient before and after coding will be both 0, therefore, there is no distortion decrease (coding improvement). The coding distortion only decreases if the coefficient becomes significant, and if we assume that 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 coding distortion decrease as:

$$D_{prev,SIG} - D_{post,SIG} = p \cdot 2.25 \cdot 4^{n-i}. \quad (22)$$

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

$$s_{SIG}(i) = \frac{D_{prev,SIG} - D_{post,SIG}}{R_{SIG}} = \frac{9}{1 + H(p)/p} 4^{n-i-2}. \quad (23)$$

From (20) and (23), we may arrive at the following conclusions:

1. The more significant bit should be encoded earlier.

In fact, within the same coding category (significance identification/refinement), one bitplane significance translates into 4 times more contribution in distortion decrease per coding bit spent.

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

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

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. The entropy coder in the JPEG 2000 thus encodes the bit array 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 sub-bitplanes: the predicted significance (PS), the refinement (REF) and the predicted insignificance (PN). We illustrate the block coding of the coefficients with an example in Figure 19:

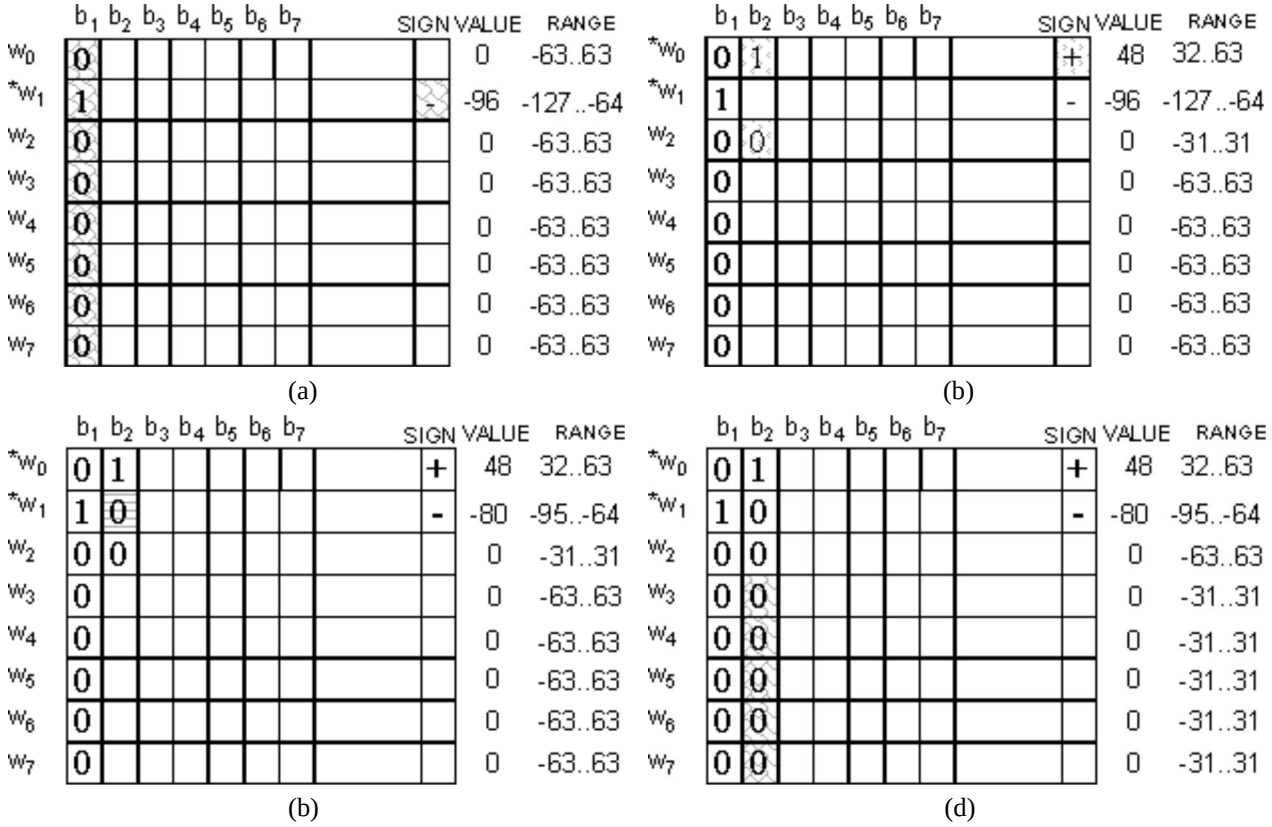

Figure 19 Order of coding: (a) Bitplane $\mathbf{b_1}$, sub-bitplane PN, then Bitplane $\mathbf{b_2}$, sub-bitplanes (b) PS, (c) REF and (d) PN.

Step 1. The most significant bitplane. (PN sub-bitplane of $\mathbf{b_1}$, shown in Figure 19(a))

First, the most significant bitplane is examined and encoded. Since at first, all coefficients are insignificant, all bits in the MSB bitplane all belong to the PN sub-bitplane. Whenever a bit '1' is encountered, which renders the corresponding coefficient non-zero, the sign of the coefficient is encoded immediately afterwards. With the information of the already coded bits 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 19(a). We list the current reconstruction values and the uncertainty ranges of coefficients under columns "value" and "range", respectively. As the coding proceeds, the uncertainty range shrinks, and brings better and better representation for each coefficient.

Step 2. The PS sub-bitplane of $\mathbf{b_2}$ (Figure 19(b))

After all bits in the most significant bitplane have been encoded, the coding proceeds to the PS sub-bitplane of the second most significant bitplane ($\mathbf{b_2}$). The PS sub-bitplane consists of bits of the coefficients that are not significant, but has at least one significant neighbor. The corresponding sample bit-array coding is shown in Figure 19(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 bit '1' 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 sub-bitplane of $\mathbf{b_2}$ (Figure 19(c))

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

Step 4. The PN sub-bitplane of $\mathbf{b_2}$ (Figure 19(d))

Finally, the rest of the bits in the bitplane are encoded in the PN sub-bitplane pass, which consists of the bits of the coefficients that are not significant and have no significant neighbors. Note that since all coefficients are zero at the beginning of the coding, all bits in the most significant bitplane belong to the PN sub-bitplane. Sign is again encoded following bit '1', which turns coefficient into significant.

Steps 2-4 are repeated for the following bitplanes, with the sub-bitplane coding ordered as PS, REF and PN. 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; we can still get an estimate of each coefficient, albeit with a relatively large uncertain range.

7. BITSTREAM ASSEMBLER

The embedded bitstream of the code-blocks are assembled by the bitstream assembler module to form the compressed bitstream of the image. During the block entropy coding, the coder not only produces an embedded bitstream for each code-block *i*, but also records the coding rate R^k_i and distortion D^k_i at the end of each sub-bitplane, where *k* is the index of the sub-bitplane. 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:

$$\begin{cases} \min \sum_i D^{n_i}_i \\ \sum_i R^{n_i}_i \le B \end{cases} \qquad (24)$$

Since there is a discrete number of truncation points n_i, the constraint minimization problem of equation (24) can be solved by distributing bits first to the code-blocks with the steepest distortion per rate spent, as follows:

1) Initialize truncation points $n_i=0$.

At the initialization, no bit segment is included from any code-blocks.

2) For each code block *i*, the maximum possible gain of distortion decrease per rate spent is calculated as:

$$S_i = \max_{k>n_i} \frac{D^{n_i}_i - D^k_i}{R^k_i - R^{n_i}_i}, \qquad (25)$$

We call S_i as 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:

(26)

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

3) The process of step 2) can be repeated 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 $R^{n_i^{new}}_i - R^{n_i}_i$ 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 computational complex. We may speed up the process of the bitstream assembler by first calculating a convex hull of the R-D slope of each code-block *i*, as follows:

1) Set $\mathbf{S}=\{k\}$ which is the set of all truncation points.
2) Set $p=0$,
3) For $k=1, 2, 3, \cdots$,

   If $k \in \mathbf{S}$, set $S^k_i = \frac{D^p_i - D^k_i}{R^k_i - R^p_i}$, if $p \neq 0$ and $S^k_i \ge S^p_i$, point *p* is removed from set **S**, go to step 2)
   otherwise, set $p=k$ and repeat step 3)

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:

$$n_i = \arg\max_k (S^k_i > \lambda). \qquad (27)$$

It is easy to recursively search the minimum slope λ that satisfies the rate inequality in (24). The R-D optimization procedure can be illustrated in Figure 20.

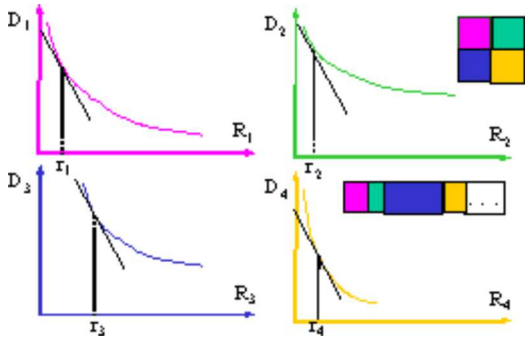

Figure 20 Bitstream assembler: for each R-D slope λ, a truncation point can be found at each code-block. The slope λ should be the minimum slope that the allocated rate for all code-blocks is smaller than the required coding rate *B*.

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, $B^{(1)}$, $B^{(2)}$,$\cdots$, $B^{(n)}$. 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 512x512, 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 truncation point of each code-block $n^{(1)}_i$ 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 $n^{(2)}_i$ can be derived, and the bitstream segment between the first $n^{(1)}_i$ and the second $n^{(2)}_i$ truncation point 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 21.

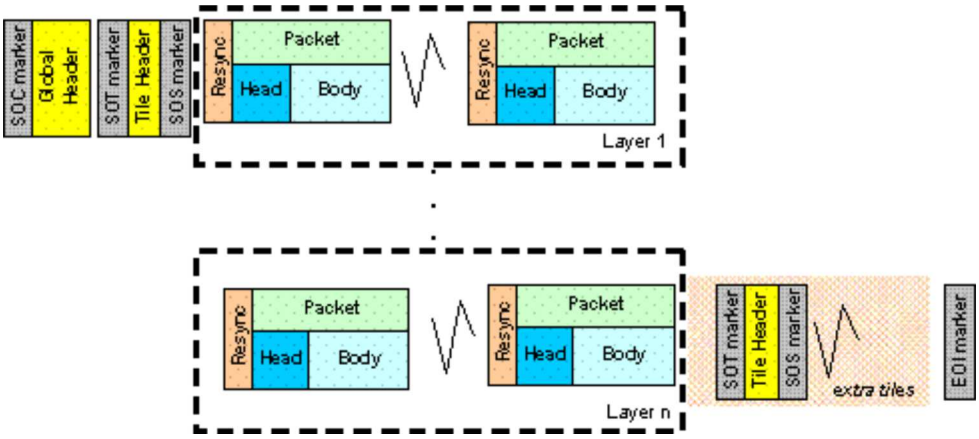

Figure 21 JPEG 2000 bitstream syntax[2].

8. PERFORMANCE OF THE JPEG 2000

We compare the JPEG 2000 image compression standard with the traditional JPEG standard. The test image is the Bike standard image (gray, 2048x2560). Three modes of JPEG 2000 are tested, and compare against two modes of the JPEG standard. The JPEG modes are progressive (P-DCT) and sequential (S-DCT) both with optimized Huffman tables. The JPEG-2000 modes are single layer with the (9,7) wavelet (S-9,7), six layer progressive with the (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 5x3 wavelet. The JPEG progressive mode uses a combination of spectral refinement and successive approximation. We shown the performance comparison in Figure 22.

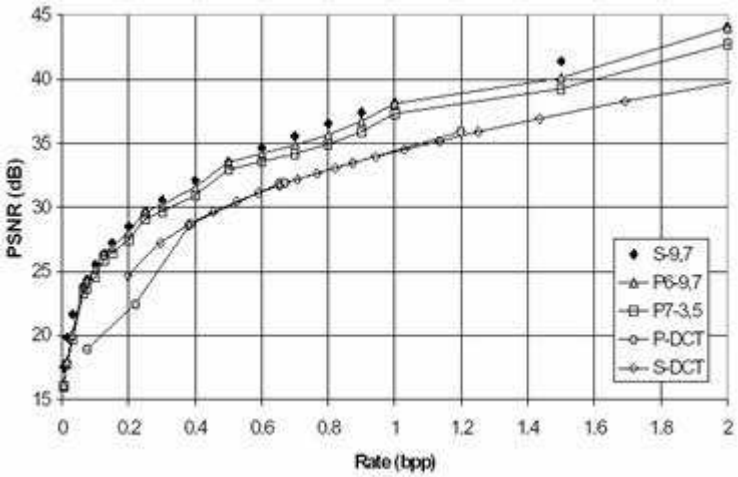

Figure 22 Performance comparison: JPEG 2000 versus JPEG (courtesy of Prof. Marcellin, et. al, [1]).

The JPEG-2000 results are significantly better than the JPEG results for all modes and all bitrates 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 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.

9. REFERENCE

[1] M. W. Marcellin, M. Gormish, A. Bilgin, M. P. Boliek, "An Overview of JPEG2000", *Proc. of the Data Compression Conference*, Snowbird, Utah, March 2000, pp. 523-544.

[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, Mar. 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, Sept. 1992.

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

[6] J. Shapiro, "Embedded image coding using zerotree of wavelet coefficients", *IEEE Trans. On Signal Processing*, Vol. 41, pp. 3445-3462, Dec. 1993.

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

[8] Daubechies, I. and W. Sweldens, "Factoring wavelet transforms into lifting steps", *J. Fourier Anal. Appl.*, Vol. 4, No. 3, 1998.

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

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

[11] 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.

[12] D. Taubman, "High performance scalable image compression with EBCOT", *IEEE Trans. On Image Processing*, Vol. 9, No. 7, pp. 1158-1170, Jul. 2000.

[13] 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.

[14] J. Li and S. Lei, "An embedded still image coder with rate-distortion optimization", *IEEE Trans. On Image Processing*, Vol. 8, no. 7, pp. 913-924, Jul. 1999.

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

[16] J. Li and H. Sun, "A virtual media (Vmedia) access protocol and its application in interactive image browsing", *SPIE, IS&T and ACM SIG Multimedia, Multimedia Computing and Networking 2001 (MMCN'01)*, Vol. 4312, No. 10, San Jose, CA, Jan. 2001.

---

[1] Acronyms are popular in industrial standard groups. Explanation of the acronyms involved are as follows:
JPEG: Joint Photographic Experts Group. This is a group of experts nominated by national standards 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.
ISO: International Standard Organization.
IEC: International Electrotechnical Commission.
JTC: Joint Technical Committee.
SC: Sub Committee.

[2] Explanation of the acronyms:
SOC: start of image (codestream) marker.
SOT: start of tile marker.
SOS: start of scan marker.
EOI: end of image marker.