A Study of the JPEG-2000 Image Compression Standard

by

Clifford Lui

A project submitted to the
Department of Mathematics and Statistics
In conformity with the requirements
For the degree of Master of Science (Engineering)

Queen's University
Kingston, Ontario, Canada
May 2001

Abstract

Image compression is a very essential process in this multimedia computer era, as it keeps the file size of the digital image as low as possible either for hardware storage requirements or fast transmission times. Many graphic compression schemes have been developed over the last decade. The JPEG compression international standard is a very popular image compression scheme due to its low complexity. It was developed by the Joint Photographic Experts Group (JPEG) in 1992 and designed for compressing either color or grayscale images of natural real-world scenes. The Baseline Sequential, Discrete Cosine Transformation (DCT) based mode of operation within the JPEG standard is the only one mode that is widely implemented in many image processing programs. However, as digital imagery equipment became more widely used, the strong need for high performing image compression techniques that offer more sophisticated functionality than the JPEG standard led to the new JPEG-2000 standard. It is based on Discrete Wavelet Transformation (DWT) with arithmetic entropy coding, and it offers many novel features including the extraction of parts of the image for editing without decoding, the focus on regions of interest with sharp visual quality and specified bitrate, and others. The JPEG-2000 compression standard is based on Discrete Wavelet Transformation (DWT) with the arithmetic entropy coding.

This thesis presents the algorithms for the JPEG standard briefly and the JPEG-2000 standard in detail. Several “standard” test images are compressed and reconstructed in both standards in order to compare them visually and objectively.

Acknowledgements

I am very grateful for the useful advice, careful proof-readings, patience, and support of my supervisors, Dr. Fady Alajaji and Dr. Tamas Linder.

I would also like to thank my housemates, Samson Wu and Matthew Shiu, for their support in writing this thesis.

Contents

Chapter 1

Introduction

1.1 Image Compression

When a photo is taken from a digital camera or a graphic is scanned to form a digital image, substantial storage space is required to save it in a file. Also, it is time consuming to transmit it from one place to another because of the large file size. Therefore, the amount of data in the image file must be reduced. This process is called “image compression”.

1.2 Typical Image Compression Encoder and Decoder

Some typical components of an image compression system can also be found in the JPEG and the JPEG-2000 standards. In both algorithms, the following operations are performed in the encoder:

(1) the source image is partitioned into blocks / tiles;

(2) the pixels values are frequency domain transform coded;

(3) the transformed coefficients are quantized;

(4) the resulting symbols are entropy coded.

These common procedures are shown in Figure 1.1. The reverse operations are performed in the decoder of the image compression system as shown in Figure 1.2.

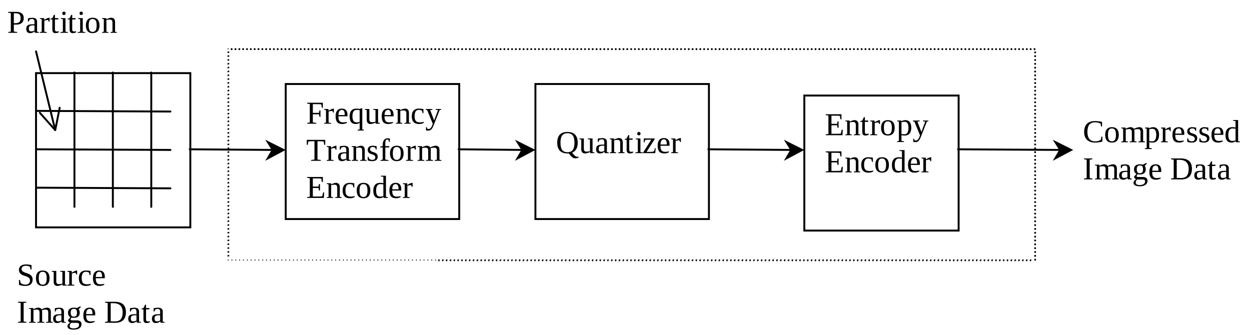


Figure 1.1 Typical image compression encoder

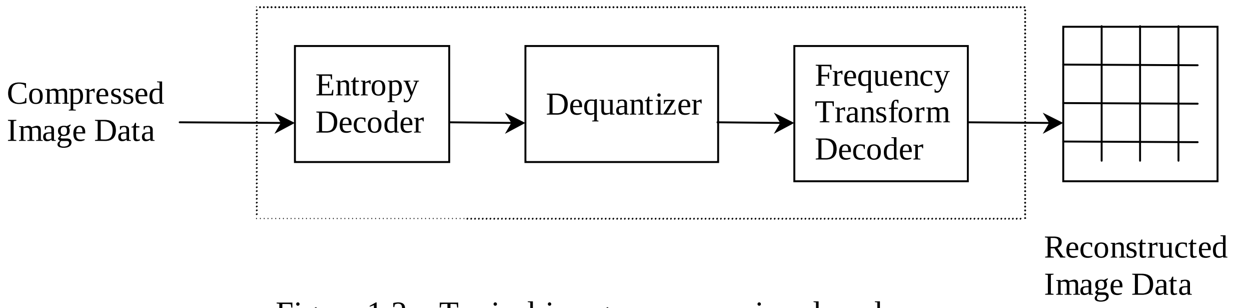


Figure 1.2 Typical image compression decoder

1.3 Frequency Domain Coding

Frequency domain coding is the fundamental part of the two image compression standards that are discussed in this thesis. The purpose of this coding is to decorrelate the information between data. For example, a pixel in the image in red color has a high probability that its immediate neighbour pixels also have a similar color. This behaviour is called "correlation." Removing correlation between the pixels of an image allows more efficient entropy encoding, which is another part of the compression system. Another advantage of frequency domain coding is that the knowledge of the distortion perceived by the image viewer can be used to improve the coder performance. For instance, the low frequency elements from a continuous tone image are more important than the high frequency elements [12], so the quantization step for the high frequency coefficients can be larger. The Discrete Cosine Transformation (DCT) and the Discrete Wavelet Transformation (DWT) are the two frequency domain coding methods adopted by the JPEG and the JPEG-2000 standards, respectively. These transformations decompose the two-dimensional pixel values from the image into basis signals and produce the coefficients of these basis signals as the outputs.

1.4 Quantization of Coefficients

Quantization reduces the precision of the coefficients by dividing them with quantization values, so that less number of bits are required to represent the coefficients. These values are chosen carefully by using knowledge about the human visual system [8]. Quantization is usually the main source for error in the reconstructed image.

1.5 Entropy Coding

Entropy coding is a compression technique that uses the knowledge of the probabilities of all the possible data/symbols within the source image file. If a shorter codeword is assigned to a frequently occurring symbol instead of a rare symbol, the compressed file size will be smaller. The Huffman coding [10] in the JPEG standard has a very simple algorithm while the more complex arithmetic coding [10], [21] in the JPEG-2000 standard achieves 5 to 10 percent more compression rate. One of the reasons is that the JPEG standard uses fixed codewords (see Tables 2.4 and 2.5) for all images while the JPEG-2000 standard uses an adaptive probability estimation process in the arithmetic coding. This estimation process tends to approach the correct probabilities. [10]

1.6 JPEG

JPEG (Joint Photographic Experts Group) is a joint ISO/CCITT committee that developed the JPEG standard in 1992. The JPEG standard is designed to compress continuous-tone still images either in grayscale or in color. This standard allows software implementation on any computer platform with affordable hardware. This is a very important feature that led to the wide use of JPEG throughout the 1990's. The JPEG standard has four different modes of operation, which are: Baseline Sequential encoding, Progressive encoding, Lossless encoding and Hierarchical encoding [20]. Since 1992, when JPEG was released, Baseline Sequential encoding has been the most popular mode because its sophisticated compression method is sufficient for most practical applications. Therefore, we will only discuss this mode in this thesis. For convenience, Baseline Sequential encoding will be denoted as JPEG throughout this thesis.

1.7 JPEG-2000

The goal of the JPEG-2000 is to develop "a new image compression system for all kinds of still images (bi-level, grayscale, color, multi-component) with different characteristics (continuous-tone, text, cartoon, medical, etc), for different imaging models (client/server, real-time transmission, image library archival, limited buffer and bandwidth resources, etc) and preferably within a unified system" [15].

The JPEG-2000 was approved as a new project in 1996. A call for technical contributions was made in March 1997. The resulting compression technologies were evaluated in November 1997. Among the 24 algorithms, the wavelet/trellis coded quantization (WCTQ) algorithm was the winner and was selected as the reference JPEG-2000 algorithm. Its main components are discrete wavelet transformation, trellis coded quantization, and binary arithmetic bitplane coding.

A detailed description of this algorithm can be found in [14]. A list of “core experiments” was performed on this algorithm and other useful techniques in terms of the JPEG-2000 desired features [9]. They were evaluated in terms of complexity and meeting the goals of JPEG-2000. According to the results of these experiments, a “Verification Model” (VM) version 0 was created, which was a reference software of the JPEG-2000 that was used to perform further core experiments. It was updated based on the results of the core experiments that are presented at each JPEG-2000 meeting.

Many additions and modifications were performed on VM 0 after several meetings. VM 2 has the following main improvements: user specified wavelet transformations are allowed; a fixed quantization table is included; no quantization is performed for integer wavelet transformations; several modifications were made to the bitplane coder; rate control is achieved by truncating the bitstream; tiling, region of region coding, error resilience was added [4].

EBCOT (embedded block coding with optimized truncation) was included in VM 3 at the meeting in November 1998 [17]. EBCOT divides each sub-band into rectangular code blocks of coefficients and the bitplane coding is performed on these code blocks independently. The idea of “packet” is also introduced. A packet collects the sub-bitplane data from multiple code blocks in an efficient syntax. Quality “layer” is in turn formed from a group of packets. The packet data, that are not included in previous layers, with the steepest rate-distortion slope are put together in a layer. Optimized truncation is obtained by discarding the least important layers. This scheme is designed to minimize the mean square error with the constraint on bitrate.

In the VM 5, the MQ-coder, submitted by Mitsubishi, was accepted as the arithmetic coder of the JPEG-2000 in March 1999 at the meeting in Korea. This MQ-coder is very similar to the one that is used in the JPEG but this new coder is available on a royalty and free fee basis for ISO standards.

The JPEG-2000 standard has 6 parts at this moment. Part 1 is called the “core coding system”, which describes the specifications of the decoder as a minimal requirement while the specifications of the encoding part are also included only as informative materials to allow further improvements in the encoder implementations. Part 2 is denoted as the extensions of Part 1, which adds more features (user defined wavelet transformation, etc) and sophistication to the core coding system for some advanced users. Part 3 is for the motion JPEG-2000. Part 4 provides a set of compliance testing procedures for the implementations of the coding system in Part 1 as a tool for quality control. Part 5 introduces two free software implementations that perform the compression system for both the encoder and decoder in order to gain wide acceptance of the JPEG-2000. Part 6 defines a file format that stores compound images. Only the contents of Part 1 and Part 5 are discussed in this thesis.

1.8 Thesis Outline

The rest of this thesis is organized in the following way. An overview of the JPEG standard is presented in Chapter 2. A more detailed description of the new JPEG-2000 standard is presented in Chapter 3. Experimental results for comparing the two standards are shown in Chapter 4. In Chapter 5, the results are summarized and the future of JPEG-2000 is discussed.

Chapter 2

JPEG

2.1 Digital Image

Every digital image consists of "component(s)." For example, some color display device's images are composed of three components (Red, Green, and Blue). Printed materials use the CMYK system and its components are Cyan (blue), Magenta, Yellow and blacK. In turn, every component has a rectangular array of pixels. Usually, an uncompressed image uses 8 bits / pixel to specify the grayscale of a color component. Therefore, $2^8 = 256$ grayscale levels are created for each component. If there is only one component, it is called a "grayscale" image. Images, which have two or more color components, are called "color" images.

2.2 Encoder and Decoder Structure of JPEG

The simplified structures of the encoder and the decoder of JPEG are shown in Figure 2.1. Assume that we have a grayscale image for now. Multiple-component images will be discussed in another section. The major processing steps of the encoder are: block division, Forward Discrete Cosine Transformation (FDCT), quantization, and entropy encoding. The role of the decoder is to reverse the steps performed by the encoder.

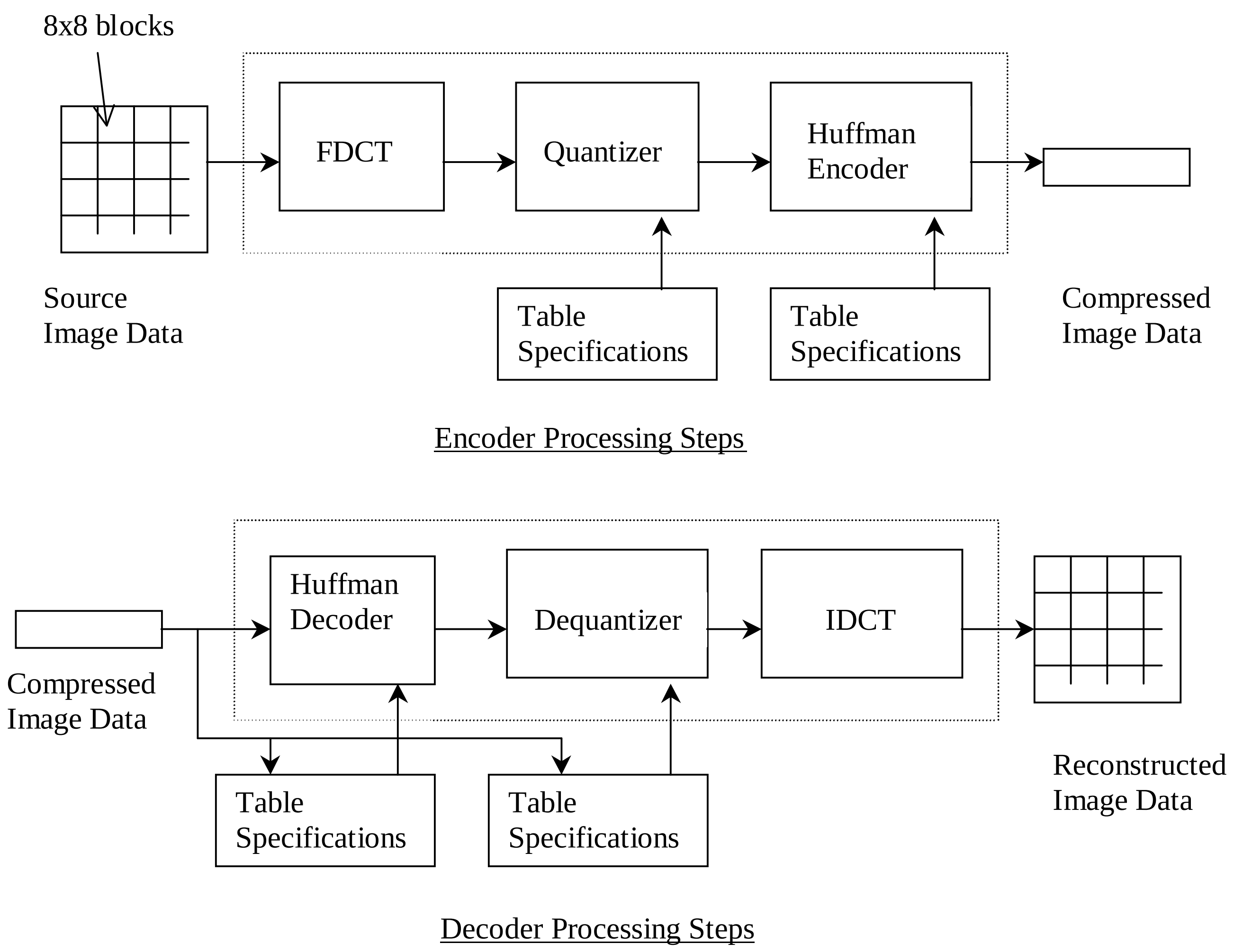


Figure 2.1 Encoder and Decoder Structure of JPEG

2.3 8x8 Blocks

All the pixels in an image are divided into 8x8 sample blocks, except the edge portion. These blocks are ordered according to a “rasterlike” left-to-right, top-to-bottom pattern. (see Figure 2.2). The partition of an image can help to avoid buffering the data for the whole image samples. However, the partition also creates the problem of “blocking artifacts.”

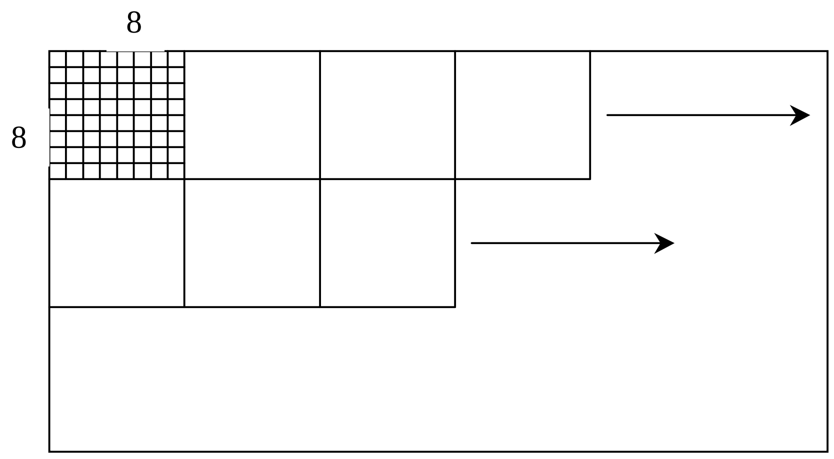


Figure 2.2 “Rasterlike” pattern

2.4 Zero-Shift and Discrete Cosine Transformation

The 8x8 = 64 sample values from each block are shifted from unsigned integers to signed integers ([0, 255] to [-128, 127]). This zero-shift reduces the precision requirements for the DCT calculations. Then, these shifted sample values, f(x,y), are fed to the two-dimensional FDCT (this is created by multiplying two one-dimensional DCTs) according to the following equation. The two-dimensional inverse DCT equation is also provided:

$$F(u,v) = \frac{1}{4}C(u)C(v)\left[\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16}\right] \quad ,$$

$$f(x,y) = \frac{1}{4}\left[\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)C(v)F(u,v)\cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16}\right] \quad ,$$

where $C(u) = \frac{1}{\sqrt{2}}$ if u = 0, and $C(u) = 1$ otherwise (the same is true for the parameter v).

FDCT decomposes the 64-coefficient digital signal into 64 orthogonal basis signals to achieve decorrelation between samples. Each of these basis signals contains one of the 64 unique two-dimensional spatial frequencies. The outputs are denoted as the DCT coefficients, which are the amplitudes for the basis signals. F(0,0) is called “DC coefficient” while the remaining 63 coefficients are called “AC coefficients.”

2.5 Quantization

Quantization reduces the precision of the DCT coefficients, F(u,v), by dividing them with quantization values Q(u,v) and rounding the results to integer values. Dequantization multiplies the quantized coefficient F^{Q}(u,v) with the quantization value Q(u,v) to get the reconstructed coefficient, $F^{Q'}$ (u,v):

$$F^{Q}(u,v) = Integer\_Round\left(\frac{F(u,v)}{Q(u,v)}\right) \quad ,$$
$$F^{Q'}(u,v) = F^{Q}(u,v) * Q(u,v) \quad .$$

The quantization values can be set individually for different spatial frequencies using the criteria based on the visibility of the basis signals. Tables 2.1 and 2.2 give examples for luminance quantization values and chrominance quantization values for the DCT coefficients respectively.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Table 2.1 Luminance quantization table

The luminance value represents the brightness of an image pixel while the chrominance value represents the color of an image pixel. These tables are the results drawn from CCIR-601

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Table 2.2 Chrominance quantization table

experiments by Lohscheller (1984) [8]. For continuous-tone image, the sample values vary gradually from point to point across the image. Therefore, most of the signal energy lies in the lower spatial frequencies, so the quantization values for higher spatial frequencies tend to be large. In practice, a lot of the DCT coefficients have zero or near-zero value, especially for the high spatial frequencies. Therefore, these coefficients usually have a quantized value of zero.

2.6 Huffman Coding

Two types of entropy coding are specified for JPEG. They are Huffman Coding and Arithmetic Coding. Huffman coding has a simpler computation and implementation but the code tables have to be known at the start of entropy coding. Arithmetic coding typically provides 5 to 10% more compression than Huffman coding. However, the particular variant of arithmetic coding specified by the standard is subject to patent [10]. Thus, one must obtain a license to use it. Therefore, most of the software implementations use Huffman Coding.

A similar arithmetic coding technique is also adopted by the new JPEG-2000 standard, so the topic of arithmetic coding is left to be discussed in Chapter 3 and only the Huffman coding is discussed in this section.

2.6.1 Differential Coding and Intermediate Sequence of Symbols

After quantization, the DC coefficients from all blocks are separately encoded from the AC coefficients. The DC coefficient represents the average value of the 64 samples within each block. Thus, strong correlations usually exist between adjacent blocks' DC coefficients. Therefore, they are differentially encoded according to the following equation:

$$\text{DIFF} = \text{DC}_i - \text{PRED}$$

where PRED is the value of the previous block's DC coefficient from the same component. Each DIFF is encoded as "symbol-1" and "symbol-2." Symbol-1 represents the "size" information while symbol-2 represents the sign and amplitude. Size is the number of bits that are used to encode symbol-2. Table 2.3 shows the corresponding size information for DIFF.

Size (symbol 1)	DIFF	Sign and Magnitude (symbol 2)
0	0	--
1	-1, 1	0, 1
2	-3, -2, 2, 3	00, 01, 10, 11
3	-7, …, -4, 4, …, 7	000, …, 011, 100, …, 111
4	-15, …, -8, 8, …, 15	0000, …, 0111, 1000, …, 1111
5	-31, …, -16, 16, …, 31	00000, …, 01111, 10000, …, 11111
6	-63, …, -32, 32, …, 63	000000, …, 011111, 100000, …, 111111
7	-127, …, -64, 64, …, 127	0000000, …, 0111111, 1000000, …, 1111111
8	-255, …, -128, 128, …, 255	Etc
9	-511, …, -256, 256, … 511	Etc
10	-1023, … -512, 512, …, 1023	Etc
11	-2047, …, -1024, 1024, … 2047	Etc

Table 2.3 Huffman coding of DIFF, Sign and Magnitude

If DIFF is positive, symbol-2 represents DIFF as a simple binary number. If it is negative, symbol-2 is “one’s complement” of the amplitude of DIFF in binary number, as shown in Table 2.3.

The quantized AC coefficients are ordered according to the “zigzag” scan in Figure 2.3. This order makes the entropy coding more efficient by placing low-frequency coefficients (likely to be non-zero) before high-frequency coefficients. Then the nonzero AC coefficients are also

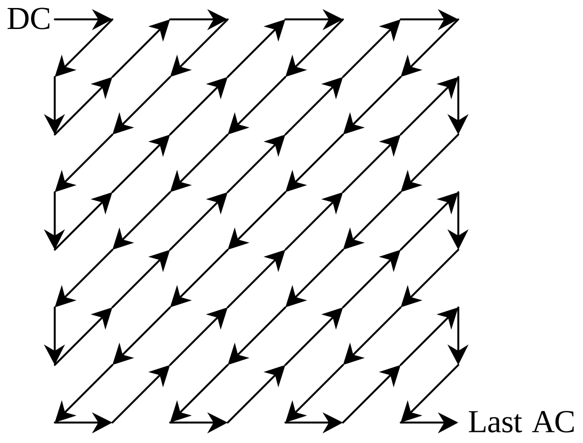


Figure 2.3 Zigzag Sequence

represented by symbol-1 and symbol-2, but symbol-1 represents both the “runlength” (consecutive number) of zero-valued AC coefficients preceding it in the zigzag sequence and the “size” information. Runlength can have a value of 0 to 15. If there are more than 15 consecutive zeros in the sequence, then a symbol-1 of (15, 0) is used to represent 16 consecutive zeros. Up to three consecutive (15, 0) extensions are allowed. If the last run of zeros includes the last AC coefficient, then a special symbol-1, (0,0), meaning EOB (end of block), is appended. The composite “runlength-size” value is (16 x runlength) + size. The way to encode symbol-2 for AC coefficient is the same way that is used to encode that of DIFF. The result from above is the “intermediate sequence of symbols.”

2.6.2 Variable-Length Entropy Coding

The Huffman code assignment is based on a coding tree structure. The tree is organized by a sequence of pairing the two least probable symbols. These two symbols are joined at a node, which is considered as a new symbol. This new symbol's probability is the sum of probabilities of the two joined probable symbols. The codeword is created by assigning 0 either to the upper or lower branches arbitrarily and 1 to the remaining branches of the tree. Then, the bits from these branches are concatenated from the "root" of the tree and traced through the branches back to the "leaf" for each symbol. An example is given below in Figure 2.4.

Symbol	Codeword
a_1	000
a_2	001
a_3	01
a_4	100
a_5	101
a_6	11

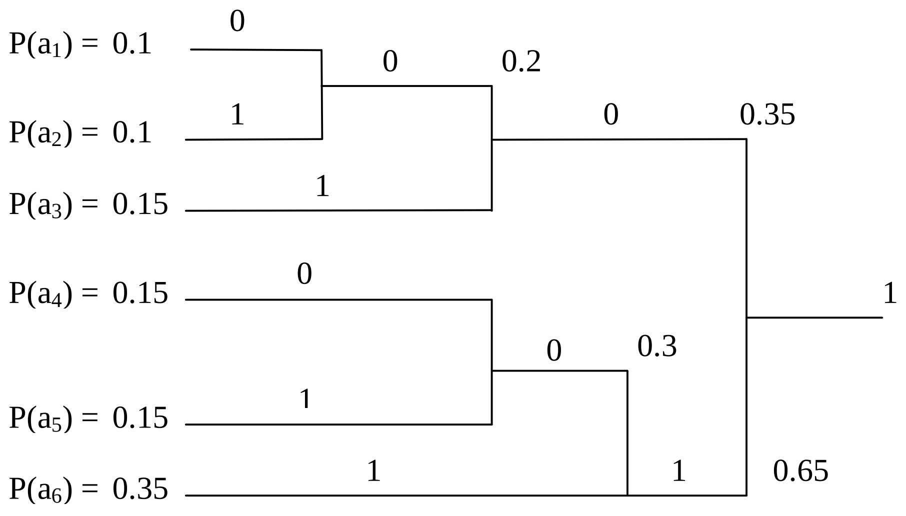


Figure 2.4 Example of Huffman Coding

Only symbol-1 is Huffman encoded with a variable length code. There should be two sets of Huffman tables. One set is for symbol-1 of DIFF and the other set is for symbol-1 of AC coefficients. The Huffman tables for both can be created by counting symbol occurrences for a large group of "typical" images and assigning a different code word to each symbol. Alternatively, these tables can be custom-made for each image separately. Tables 2.4 and 2.5 show the codewords for difference symbol-1 of the AC coefficients and DIFF.

Runlength/Size (symbol-1)	Code Length	Codeword
0/0	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011
1/1	4	1100
1/2	5	11011
1/3	7	1111001
1/4	9	111110110
1/5	11	11111110110
1/6	16	1111111110000100
1/7	16	1111111110000101
1/8	16	1111111110000110
1/9	16	1111111110000111
1/A	16	1111111110001000
2/1	5	11100
2/2	8	11111001
2/3	10	1111110111
2/4	12	111111110100
2/5	16	1111111110001001
2/6	16	1111111110001010
2/7	16	1111111110001011
2/8	16	1111111110001100
2/9	16	1111111110001101
2/A	16	1111111110001110

Table 2.4 Partial Huffman Code for symbol-1 of the AC Coefficients

Size (Symbol-1)	Code Length	Codeword
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Table 2.5 Huffman Code for symbol-1 of DIFF

2.7 Decoding

The decoding procedures perform only the inverse functions of the encoder. They consist of Huffman decoding, ordering the zigzag sequence of AC coefficients, calculation of DC coefficients from DIFF, dequantization, inverse DCT, and inverse of zero-shift from [-128, 127] to [0, 255].

2.8 Multiple-Component Images

The previous sections only discuss the processing of one-component images. For color images, the JPEG standard specifies how multiple components (maximum of 255 components) should be handled as well. A data unit is defined as an 8x8 block of samples. Each component can have its own sampling rate and we denote the dimensions here by x_i horizontal pixels and y_i vertical pixels for i^{th} component. Also, each component has its own relative horizontal and vertical sampling factors, H_i and V_i. The overall image dimensions X and Y are defined as the maximums of x_i and y_i among all the components. These parameters can be expressed according to the following equations:

$$x_i = \left\lceil X \times \frac{H_i}{H_{\max}} \right\rceil \quad ,$$

$$y_i = \left\lceil Y \times \frac{V_i}{V_{\max}} \right\rceil \quad ,$$

where $\lceil \; \rceil$ is the ceiling function

For simplicity, we can consider a three-component (component A, B and C) image with two sets of table specifications. These components and table specifications are multiplexed alternately, as shown in Figure 2.5.

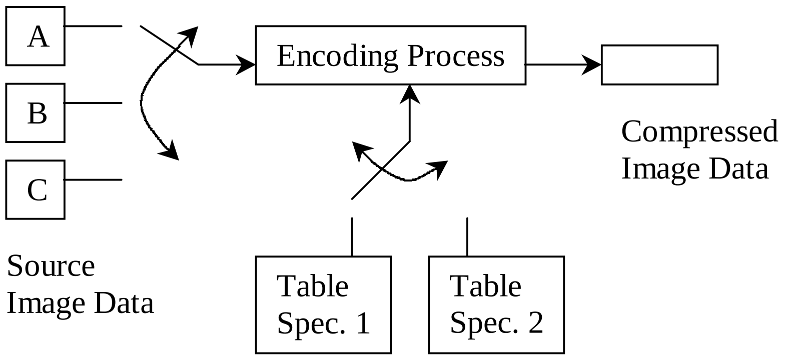
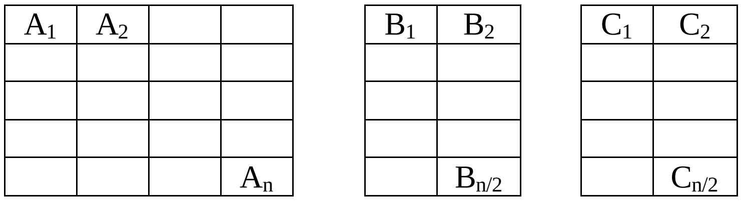


Figure 2.5 Component-interleave and table-switching control

For the non-interleaving mode, encoding is performed for all the image data units in component A before it is performed on other components, and then in turn, all data units of component B is processed before that of component C. On the other hand, interleaving mode compresses a portion of data units from component A, a portion of data units from component B, a portion of data units from component C, and then back to A, etc. For example, if components B and C have half the number of horizontal samples relative to component A, then we can compress two data units from component A, one data unit from component B, and one data unit from component C, as shown in Figure 2.5.



$A_1, A_2, B_1, C_1, A_3, A_4, B_2, C_2, \ldots.., A_{n-1}, A_n, B_{n/2}, C_{n/2}$

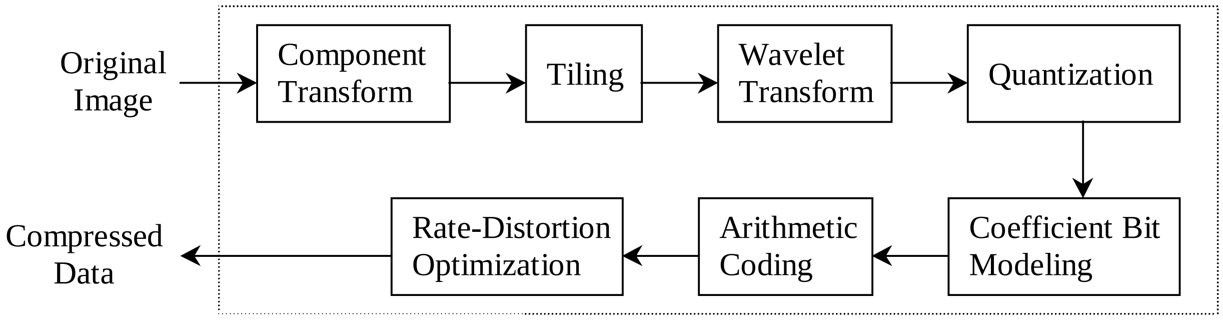
Figure 2.5 Data unit encoding order, interleaved

Chapter 3

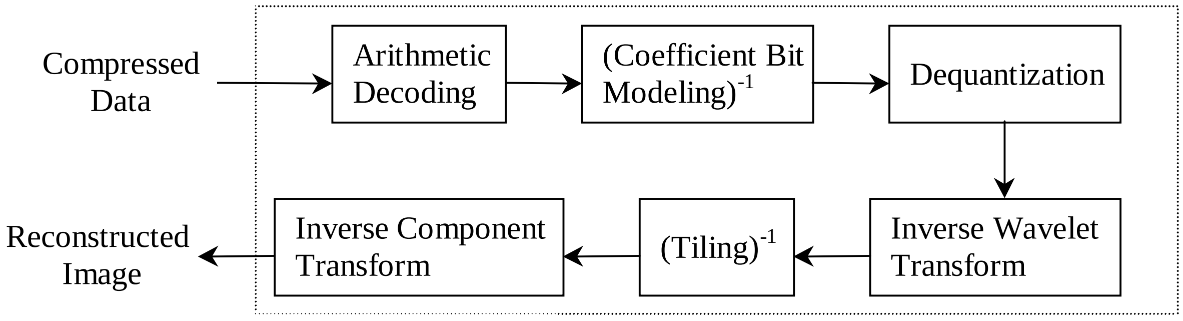
JPEG-2000

3.1 Encoder and Decoder Structures of JPEG-2000

The simplified structures of the encoder and decoder of JPEG-2000 are shown in Figure 3.1. Assume that we have a multiple-component image. The major processing steps of the encoder are: component transformation, tiling, wavelet transformation, quantization, coefficient bit modeling, arithmetic coding, and rate-distortion optimization. The role of the decoder is to reverse the steps performed by the encoder, except the rate-distortion optimization step.



Encoder Processing Steps


Compressed Data
Arithmetic Decoding
(Coefficient Bit Modeling)-1
Dequantization
Inverse Wavelet Transform
(Tiling)-1
Inverse Component Transform
Reconstructed Image

Decoder Processing Steps

Figure 3.1 Encoder and decoder structure of JPEG-2000

3.2 DC Level Shifting

Forward DC level shifting is applied on every sample value I(x,y) that is unsigned in the image according to the following equation. The result is denoted as I'(x,y):

$$I'(x,y) = I(x,y) - 2^{Ssiz^i - 1},$$

where S_{siz}^{i} is the number of bits used to represent the sample value in the ith component before the shifting. For example, if S_{siz}^{i} is 8, I(x,y)'s range is shifted from [0, 255] to [-128, 127]. This is a special case, which is identical to the level shifting of the JPEG standard. Inverse DC level shifting is performed to the reconstructed samples of components that are unsigned only according to the following equation:

$$I(x,y) = I'(x,y) + 2^{Ssiz^i - 1}.$$

This DC level shifting reduces the precision requirements for the wavelet transform calculations.

3.3 Component Transformation

Two types of component transformations are specified in the JPEG-2000 standard [21]. They are the Reversible Component Transformation (RCT) and the Irreversible Component Transformation (ICT).

3.3.1 Reversible Component Transformation (RCT)

Reversible Component Transformation (RCT) should be used with the 5-3 reversible wavelet transformation (Section 3.4). It is a decorrelating transformation that is performed on the first three components of an image. There should be no sub-sampling on these three components and they should have same bit-depth (number of bits to represent a sample value). This transformation is appropriate for both lossy and lossless compression.

The forward RCT is applied to component samples $I_0(x,y)$, $I_1(x,y)$, $I_2(x,y)$, corresponding to the first, second and third components of an image and the outputs are $Y_0(x,y)$, $Y_1(x,y)$ and $Y_2(x,y)$, as shown in the following equations:

$$Y_0(x,y) = \left\lfloor \frac{I_0(x,y) + 2I_1(x,y) + I_2(x,y)}{4} \right\rfloor \quad ,$$

$$Y_1(x,y) = I_2(x,y) - I_1(x,y) \quad ,$$

$$Y_2(x,y) = I_0(x,y) - I_1(x,y) \quad .$$

The corresponding inverse RCT equations are:

$$I_1(x,y) = Y_0(x,y) + \left\lfloor \frac{Y_2(x,y) + Y_1(x,y)}{4} \right\rfloor \quad ,$$

$$I_0(x,y) = Y_2(x,y) + I_1(x,y) \quad ,$$

$$I_2(x,y) = Y_1(x,y) + I_1(x,y) \quad .$$

3.3.2 Irreversible Component Transformation (ICT)

Irreversible Component Transformation (ICT) should only be used with the 9-7 irreversible wavelet transformation. It is a decorrelating transformation that is also performed on the three first components of an image. There should be no sub-sampling on these three components and they should have same bit-depth. This transformation is appropriate for lossy compression only. The forward ICT is applied to component samples $I_0(x,y)$, $I_1(x,y)$, $I_2(x,y)$, corresponding to the first, second and third components and the outputs are $Y_0(x,y)$, $Y_1(x,y)$ and $Y_2(x,y)$, as shown in the following equations:

$$Y_0(x,y) = 0.299 \times I_0(x,y) + 0.587 \times I_1(x,y) + 0.144 \times I_2(x,y) \quad ,$$

$$Y_1(x,y) = -0.16875 \times I_0(x,y) - 0.33126 \times I_1(x,y) + 0.5 \times I_2(x,y) \quad ,$$

$$Y_2(x,y) = 0.5 \times I_0(x,y) - 0.41869 \times I_1(x,y) - 0.08131 \times I_2(x,y) \quad .$$

The corresponding inverse RCT equations are:

$$I_0(x,y) = Y_0(x,y) + 1.402 \times Y_2(x,y) \quad ,$$

$$I_1(x,y) = Y_0(x,y) - 0.34413 \times Y_1(x,y) - 0.71414 \times Y_2(x,y) \quad ,$$

$$I_2(x,y) = Y_0(x,y) + 1.772 \times Y_1(x,y) \quad .$$

3.4 Data Ordering

3.4.1 Data Ordering Scheme

An image is separated into several components if there are more than one component from the image (see Section 2.1). Then, each component is partitioned into non-overlapping tiles to form an array of “tile-components.” In turn, every tile-component is wavelet transformed into 4 sub-bands for every level of the wavelet transformation. Then, each sub-band is divided into a set of code blocks for coefficient bit modeling (see Section 3.8). These processes are summarized in Figure 3.2.

3.4.2 Reference Grid of an Image

A high-resolution grid is used to define most of the structural entities of an image. The parameters that define the grid are shown in Figure 3.3.

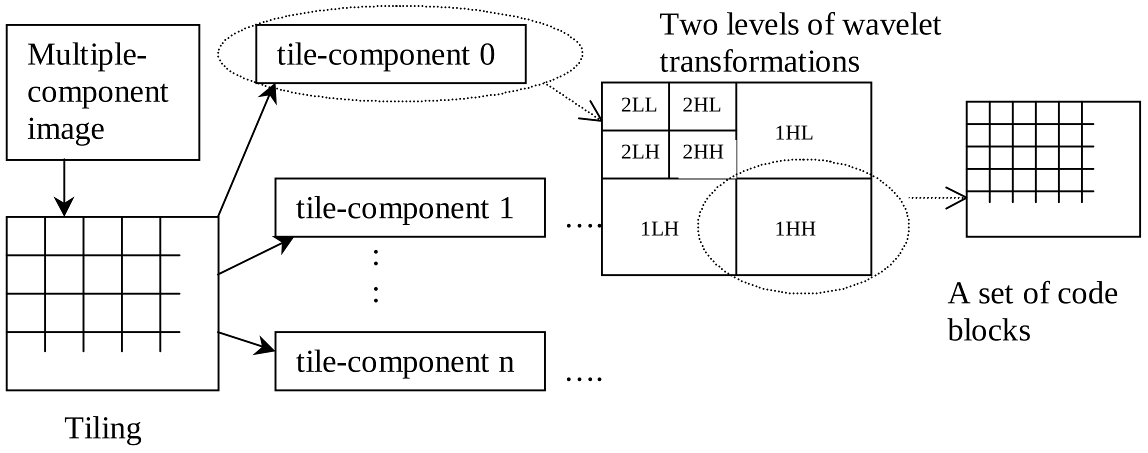


Figure 3.2 Data ordering scheme

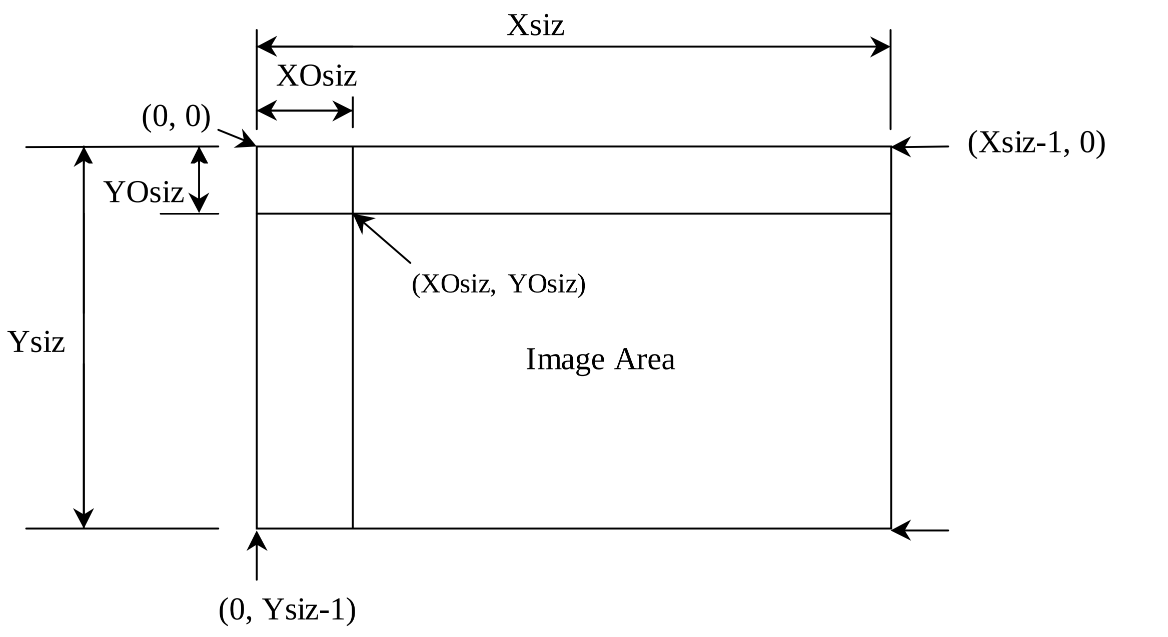


Figure 3.3 Reference Grid

This reference grid is composed of a rectangular grid of sample data points. They are indexed from (0,0) to (Xsiz-1, Ysiz-1). The “image area” is confined by the parameters at the upper left hand corner (XOsiz, YOsiz), and the lower right hand corner (Xsiz-1, Ysiz-1).

3.4.3 Division of an Image into Tiles and Tile-Components

The idea of tiling serves the same purpose as the partition of 8x8 blocks in the JPEG standard. All tiles are handled independently. Therefore, tiling reduces memory requirements because not the entire bitstream is needed to process a portion of the image. Tiling also makes extraction of a region of the image (by specifying the indexes of corresponding tiles) for editing easier. All tiles are rectangular and with the same dimensions, which are specified in the main header (located at the head of a compressed file).

The reference grid is divided into an array of “tiles.” Tiling reduces memory requirements and makes extraction of a region of the image easier. The tile’s dimensions and tiling offsets are defined as (XTsiz, YTsiz) and (XTOsiz, YTOsiz) respectively. Every tile in the image has the same width of XTsiz reference grid points and height of YTsiz reference grid points. The upper left hand corner of the first tile is offset from (0,0) to (XTOsiz, YTOsiz), as shown in Figure 3.4. The tiles are numbered in the “rasterlike” pattern. The values of (XTOsiz, YTOsiz) are constrained by:

$$0 \le XTOsiz \le XOsiz \quad , \qquad 0 \le YTOsiz \le YOsiz \quad .$$

The tile size is constrained in order to ensure that the first tile contains at least one data sample:

$$XTsiz + XTOsiz > XOsiz \quad , \qquad YTsiz + YTOsiz > YOsiz \quad .$$

The number of tiles in the horizontal direction, numXtiles, and in the vertical direction, numYtiles are calculated as follows:

$$numXtiles = \left\lceil \frac{Xsiz - XTOsiz}{XTsiz} \right\rceil \quad , \qquad numYtiles = \left\lceil \frac{Ysiz - YTOsiz}{YTsiz} \right\rceil \quad .$$

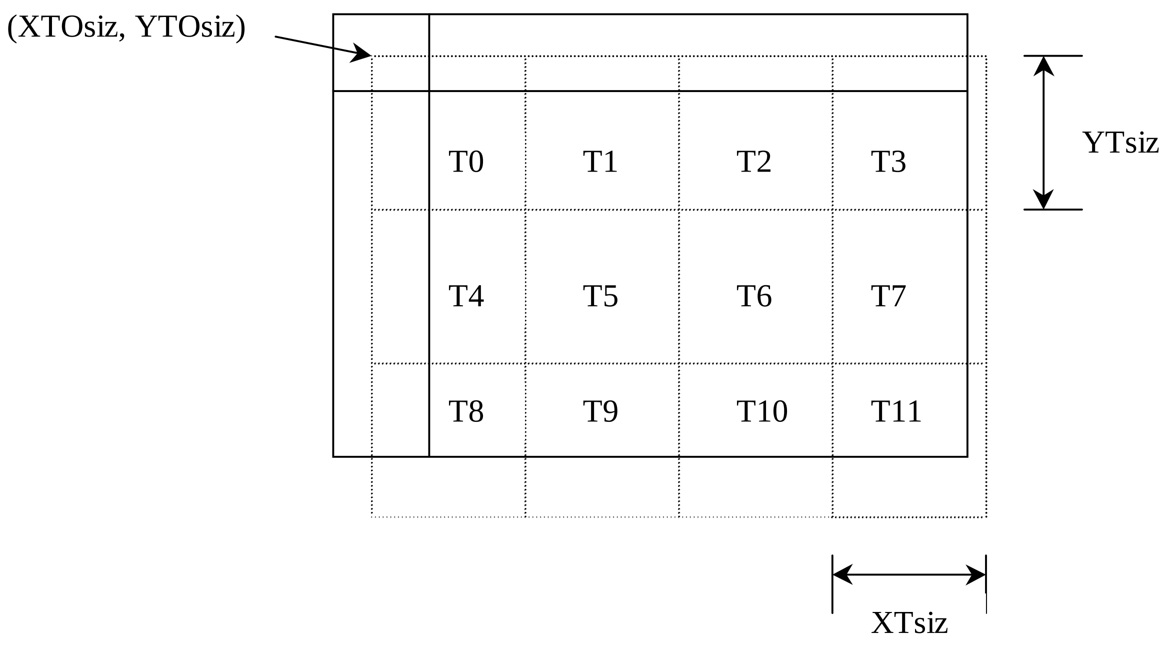


Figure 3.4 Tiling of the reference grid

For the convenience of description, the tiles are numbered in the vertical and horizontal directions. Let p be the horizontal index of a tile, ranging from 0 to numXtiles -1, while q be the vertical index of a tile, ranging from 0 to numYtiles –1. They can be determined by the following equations:

$$p = \mathrm{mod}(t, numXtiles) \quad , \qquad q = \left\lfloor \frac{t}{numXtiles} \right\rfloor \quad ,$$

where t is the index in Figure 3.4.

The coordinates of a tile for a particular (p, q) pair are:

$$\begin{aligned}
tx_0(p,q) &= \max(XTOsiz + p \times XTsiz, XOsiz),\\
ty_0(p,q) &= \max(YTOsiz + q \times YTsiz, YOsiz),\\
tx_1(p,q) &= \min(XTOsiz + (p+1) \times XTsiz, Xsiz),\\
ty_1(p,q) &= \min(YTOsiz + (q+1) \times YTsiz, Ysiz),
\end{aligned}$$

where $tx_0(p,q)$ and $ty_0(p,q)$ are the coordinates of the upper left hand corner of the tile, and $tx_1(p,q) - 1$ and $ty_1(p,q) - 1$ are the coordinates of the lower right hand corner of the tile. The dimensions of that tile are $(tx_1(p,q) - tx_0(p,q), ty_1(p,q) - ty_0(p,q))$.

Each component of the image has its parameter XRsiz(i) and YRsiz(i). The samples of the component i are those samples with index of integer multiples of XRsiz(i) in the horizontal direction and integer multiples of YRsiz(i) in the vertical direction on the reference grid. For the domain of the component i, the coordinates of the upper left hand sample (tcx_0, tcy_0) and the lower right hand sample $(tcx_1 - 1, tcy_1 - 1)$ are defined by:

$$tcx_0 = \left\lceil \frac{tx_0(p,q)}{XRsiz(i)} \right\rceil , \qquad tcx_1 = \left\lceil \frac{tx_1(p,q)}{XRsiz(i)} \right\rceil ,$$
$$tcy_0 = \left\lceil \frac{ty_0(p,q)}{YRsiz(i)} \right\rceil , \qquad tcy_1 = \left\lceil \frac{ty_1(p,q)}{YRsiz(i)} \right\rceil .$$

Thus, the dimensions of the tile-component are $(tcx_1 - tcx_0, tcy_1 - tcy_0)$.

3.4.4 Division of Tile-Component into Resolutions and Sub-bands

Each tile-component's samples are wavelet transformed into N_L decomposition levels (Section 3.5). Then, $N_L + 1$ different resolutions are provided for this tile-component. We denote the resolutions by an index r, ranging from 0 to N_L. r = 0 is the lowest resolution, which is represented by the N_LLL sub-band while $r = N_L$ is the highest resolution, which is reconstructed from the 1LL, 1HL, 1LH and 1HH sub-bands. For a specific resolution r not equal to 0, it is reconstructed from nLL, nHL, nLH, and nHH sub-bands, where n is N_L-r+1. The tile-component samples' coordinates are mapped to a set of new coordinates for a specific r yielding an upper left hand corner's coordinate (trx_0, try_0) and a lower right hand corner's coordinate (trx_1-1, try_1-1) where

$$trx_1 = \left\lceil \frac{tcx_1}{2^{N_L - r}} \right\rceil , \qquad trx_0 = \left\lceil \frac{tcx_0}{2^{N_L - r}} \right\rceil ,$$

$$try_1 = \left\lceil \frac{tcy_1}{2^{N_L - r}} \right\rceil , \qquad try_0 = \left\lceil \frac{tcy_0}{2^{N_L - r}} \right\rceil .$$

Similarly, (tcx_0, tcy_0) and (trx_0, try_0) can be mapped to a specific sub-band, b, with the upper left hand corner's coordinate (tbx_0, tby_0) and the lower right hand corner's coordinate (tbx_1-1, tby_1-1) respectively, where

$$tbx_0 = \left\lceil \frac{tcx_0 - (2^{n_b - 1} \times x0_b)}{2^{n_b}} \right\rceil , \qquad tbx_1 = \left\lceil \frac{tcx_1 - (2^{n_b - 1} \times x0_b)}{2^{n_b}} \right\rceil ,$$

$$tby_0 = \left\lceil \frac{tcy_0 - (2^{n_b - 1} \times y0_b)}{2^{n_b}} \right\rceil , \qquad tby_1 = \left\lceil \frac{tcy_1 - (2^{n_b - 1} \times y0_b)}{2^{n_b}} \right\rceil ,$$

where n_b is the decomposition level of the sub-band b and the values of $x0_b$ and $y0_b$ for different sub-bands are tabulated in Table 3.1.

Sub-band	$x0_b$	$y0_b$
n_bLL	0	0
n_bHL	1	0
n_bLH	0	1
n_bHH	1	1

Table 3.1 Quantities ($x0_b$, $y0_b$) for sub-band b

3.4.5 Division of Resolutions into Precincts

For a particular tile-component and resolution, its samples are divided into precincts, as shown in Figure 3.5. The precinct partition is originated at (0,0). 2^{PPx} and 2^{PPy} are the dimensions of the precinct where PPx and PPy can be different for each tile-component and resolution. The idea of precinct is used to specify the order of appearance of the packets within each precinct in the coded bitstream.

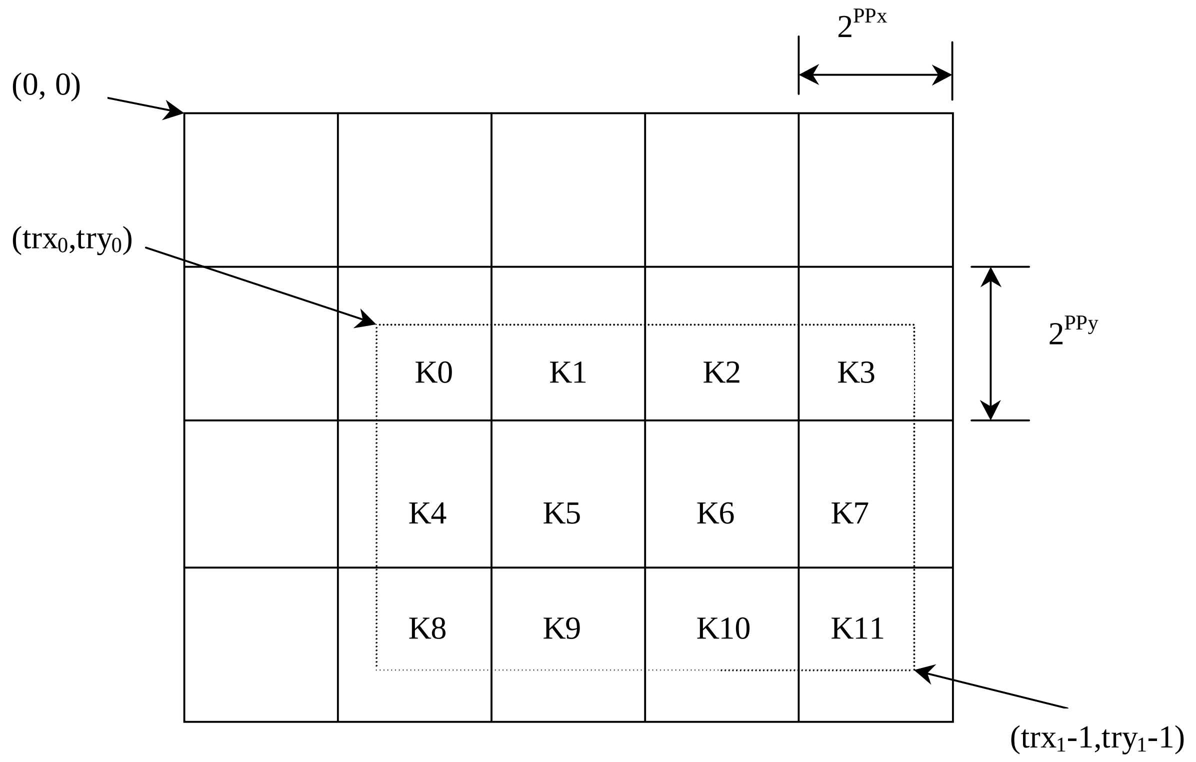


Figure 3.5 Precinct partition

3.4.6 Division of Sub-bands into Code blocks

All sub-band coefficients are divided into code blocks for coefficient modeling and coding. This partitioning reduces the requirements of memory to both the hardware and software implementations. It also provides certain degree of spatial random access to the coded bitstream. Within the same tile-component, the code block's size for each sub-band is determined by xcb and ycb. The width and the height of a code block are $2^{xcb'}$ and $2^{ycb'}$ respectively where

$$xcb' = \begin{cases} \min(xcb, PPx-1) & \text{for r > 0} \\ \min(xcb, PPx) & \text{for r = 0,} \end{cases}$$

$$ycb' = \begin{cases} \min(ycb, PPy-1) & \text{for r > 0} \\ \min(ycb, PPy) & \text{for r = 0.} \end{cases}$$

The code block partition originates from (0,0), as shown in Figure 3.6.

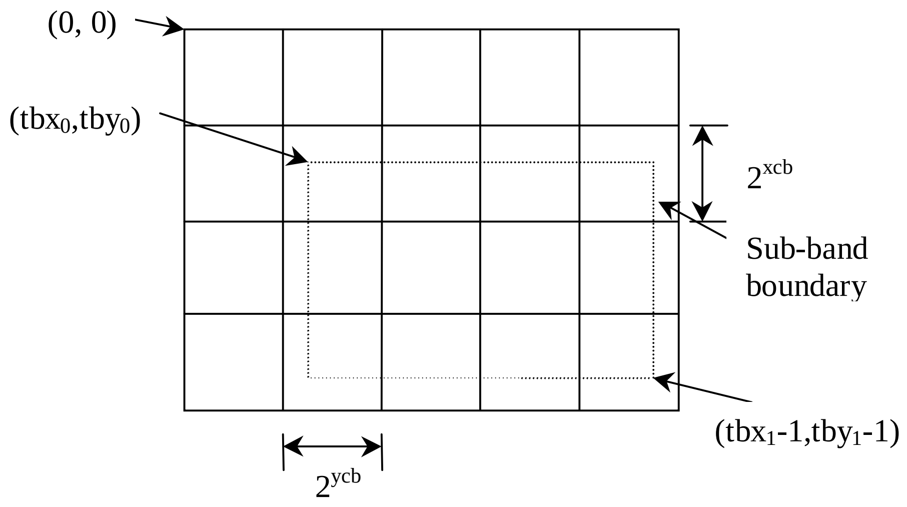


Figure 3.6 Code block partition of a sub-band

Therefore, the precincts are in turn divided into code blocks. For the code blocks that extend beyond the sub-band boundary, only the samples lying within the sub-band boundary are coded.

3.4.7 Division of Coded Data into Layers

The coded data of each code block are spread over a set of layers. Each layer is composed of some number of consecutive bit-plane coding passes (Section 3.7.3) from all code blocks. The number of coding passes is usually different from code block to code block and may be even zero, which results in an empty packet (Section 3.10.2). The layers are indexed from 0 to L-1, where L is the total number of layers in a tile.

3.4.8 Packet

The coded data for a specific precinct of a resolution in a tile-component within a layer is recorded in a contiguous segment called a “packet.” The length of a packet is an integer multiple of 8 bits (one byte). The data in a packet is ordered according to the contribution from sub-band

LL, HL, LH, and HH in that order. This order is obtained from Section 3.5 for wavelet transformation. Within each sub-band, the code block data are ordered in the “rasterlike” pattern within the bounds of the corresponding precinct.

3.4.9 Packet Header Information

The packet headers record the following essential information for the precincts:

(1) Zero length packet, which indicates that whether the packet is empty;

(2) Code block inclusion, indicating which code blocks belong to the packet;

(3) Number of the most significant bit-planes that are “insignificant” (Section 3.7);

(4) Number of the coding passes for each code block within the packet;

(5) Length of the code block data.

These headers are located preceding the packet data.

3.4.9.1 Tag Trees

A tag tree is a way of representing a two-dimensional array of non-negative integers in a hierarchical way. Reduced resolution levels of the two-dimensional array are created successively to form a tree. The minimum integer of the nodes (up to four) on a level is recorded on the node on the next lower level. An example is shown in Figure 3.7. $q_i(m,n)$ is the notation for the value at level i, m^th^ column from the left and n^th^ row from the top. Level 0 is defined as the lowest level.

Each node of every level has an initial “current value” of zero. Assume that there are n levels. The coding starts from the lowest level, which is level 0. If the valve of $q_0(0,0)$ is larger than the current value, a 0 bit is coded and the current values of this node and the nodes above it in the corresponding branch are incremented by one. The above step is repeated until $q_0(0,0)$ is equal to the current value. Then, a 1 bit is coded and the coding moves to the node $q_1(0,0)$ on the next higher level. The above processes are repeated until the node on the highest level n-1 is coded. The other nodes are coded in the same way. However, the nodes that are coded once such as $q_0(0,0)$, $q_1(0,0)$, …, $q_{n-2}(0,0)$ should not be coded again.

In the example of Figure 3.7, $q_3(0,0)$ is coded as 01111. The first two bits, 01, are the code for $q_0(0,0)$. It means that $q_0(0,0)$ is greater than zero and is equal to one. The third bit, 1, is the code for $q_1(0,0)$. The fourth bit, 1, is the code for $q_2(0,0)$ and the last bit, 1, is the code for $q_3(0,0)$. These three 1 bits mean that $q_1(0,0)$, $q_2(0,0)$ and $q_3(0,0)$ have a value of 1. To code $q_3(1,0)$, we do not need to code $q_0(0,0)$, $q_1(0,0)$, $q_2(0,0)$ again. Therefore, its code is 001. It means that $q_3(1,0)$ is greater than 1, 2 and is equal to 3.

1 $q_3(0,0)$	3 $q_3(1,0)$	2 $q_3(2,0)$	3	2	3
2	2	1	4	3	2
2	2	2	2	1	2

a) original array of numbers, level 3

1 $q_2(0,0)$	1 $q_2(1,0)$	2
2	2	1

b) minimum of four (or less) nodes, level 2

1 $q_1(0,0)$	1

c) minimum of four (or less) nodes, level 1

1 $q_0(0,0)$

d) minimum of four (or less) nodes, level 0

Figure 3.7 Example of tag tree representation

3.4.9.2 Zero Length Packet

The first bit in the packet header indicates whether the packet has a length of zero. If this bit is 0, the length is zero. Otherwise, the value of 1 means the packet has a non-zero length. This case is examined in the following sections.

3.4.9.3 Code Block Inclusion

Some code blocks are not included in the corresponding packet since they do not have contributions to the current layer. Therefore, the packet header must contain the information concerning whether a code block within the current precinct boundary is included. Two different ways are specified to signal this information depending on whether the same code block has already been included. For the code blocks that have not been included before, a tag tree for each precinct is used to signal this information. The values of the nodes of this tag tree are the index of the layer in which the code blocks are first included. For the code blocks that have been included before, one bit is used to signal the inclusion information. A 0 bit means that the code block is not included for the current precinct, while a 1 bit means that it is included for the current precinct.

3.4.9.4 Zero Bit-Plane Information

The maximum number of bits, M_b, to represent the coefficients within the code blocks in the sub-band b, is signaled in the JPEG-2000 file main header. However, the actual number of bits that is used is M_b-P, where P is the number of missing most significant bit-planes, whose bits have zero values. For the code block that is included for the first time, the value of P is coded with a separate tag tree for every precinct.

3.4.9.5 Number of Coding Passes

The number of the coding passes for each code block in the packet is identified by the codewords shown in Table 3.2

3.4.9.6 Length of the Data for a Code Block

The lengths of the number of bytes that are contributed by the code blocks are identified in the packet header either by a single codeword segment or multiple codeword

Number of coding passes	Codeword in Packet Header
1	0
2	10
3	1100
4	1101
5	1110
6-36	111100000 – 111111110
37-164	1111111110000000-1111111111111111

Table 3.2 Codewords for the number of coding passes for each code block

segments. The latter case is applied when at least one termination of arithmetic coding happens between coding passes, which are included in the same packet.

For the case of a single codeword segment, the number of bits that is used to represent the number of bytes contributed to a packet by a code block is calculated by:

$$\#\ of\ bits = Lblock + \lfloor log_2(coding\ passes\ added) \rfloor$$

where Lblock is a parameter for each code block in the precinct.

We can see that more coding passes added implies more bits are used. Lblock has an initial value of 3, which can be increased by the "signaling bits" in an accumulative way as needed. The signaling bits precede the number of bytes for a code block in the packet header. A signaling bit of zero means the value of 3 is enough for Lblock. If the signaling bits have k ones followed by a zero, the new value of Lblock is 3 plus k. For example, 44 bytes with 2 coding passes has the code of 110101100 (110 adds two bits, Lblock = 3 + 2 = 5, $\lfloor \mathrm{Log}_2 2 \rfloor = 1$, 5 +1 = 6 bits, $101100_{bin} = 44_{dec}$). Then, the next code block has 134 bytes with 5 coding passes. Its code is 1010000110 (10 adds one bit, Lblock is 5 + 1 = 6, $\lfloor \mathrm{Log}_2 5 \rfloor = 2$, 6 + 2 = 8 bits, $10000110_{bin} = 134_{dec}$).

For the case of multiple codeword segments, let $n_1<n_2<n_3 \ldots <n_K$ be the index of the terminated coding passes included for the code block in the packet. The method that is used in the single codeword segment is repeated for K times consecutively. The first length is the number of bytes from the start of the contribution of the code block in the packet to the end of the coding pass n_1. The "coding passes added" for this length is n_1. The second length is the number of bytes from the end of the coding pass n_1 to the end of the coding pass n_2. The "coding passes added" for this length is $n_2 - n_1$. This process is continued until the end of the coding pass n_K.

3.5 Discrete Wavelet Transformation of Tile-Components

3.5.1 Wavelet Transformation

All finite energy signals with finite duration can be decomposed into a set of basis signals, which are composed of translations and dilations of a simple, oscillatory function called a wavelet $\psi(t)$. $\psi(t)$ has to satisfy two following properties:

$$\int_{-\infty}^{\infty} \psi(t)dt = 0 \quad , \qquad \int_{-\infty}^{\infty} |\psi(t)|^2 dt < \infty \quad .$$

The translation and dilation of $\psi(t)$ can be represented by:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{|a|}}\psi\left(\frac{t-b}{a}\right) \quad ,$$

where b is the translation parameter and a is the dilation parameter.

The value of $1/(|a|^{1/2})$ ensures that the energy of all translations and dilations of $\psi(t)$ are identical. Assume that a is 2^k and b is 2^km. The coefficients of the basis signals d(k, m) are obtained by the convolution of the signals with a low pass filter h(m) and high pass filter g(m) in two separate paths with a downsampling of 2 as the last step (Figure 3.8). c(k,m) are the coefficients of the "scaling function", which in turn can be represented by the translations and dilations of $\psi(t)$ on the lower levels. This process is called decomposition or analysis of the signal. It can be iterated many times to create more decomposition levels.

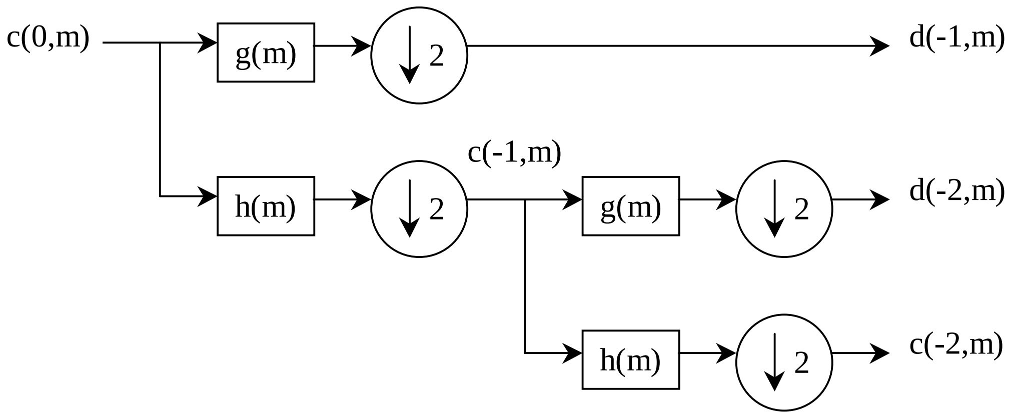


Figure 3.8 A two-level decomposition

The coefficients, c(k,m) and d(k,m), can be used to reconstruct the original signal, which is obtained by adding the results of the convolution of the filters h'(m) and g'(m) with the upsampled version of the coefficients (see Figure 3.9). This process is called reconstruction or synthesis of the signal.

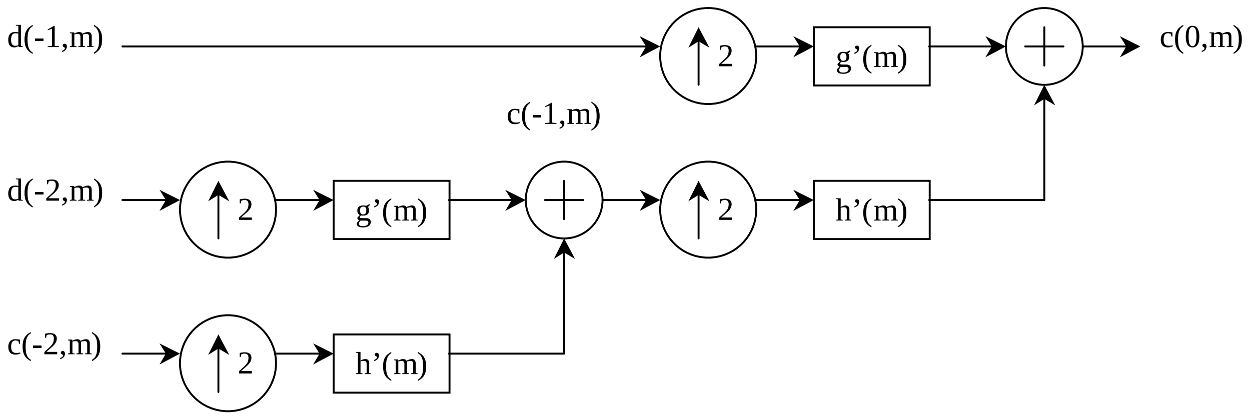


Figure 3.9 A two-level reconstruction

The JPEG-2000 standard specifies two wavelet transformations, which are irreversible Daubechies 9-tap/7-tap filters and reversible 5-tap/3-tap filters. They are chosen for a number of reasons [1]. Both of them have short finite impulse response (FIR) filters, so fast computation can be implemented. 5-tap/3-tap transformation has only two lifting steps, which help to achieve a very low computational complexity. 9-tap/7-tap transformation has the highest values of Peak Signal to Noise Ratio (PSNR) over many test images for low bit rates. Tables 3.3 and 3.4 show the tap values of the analysis and synthesis of the Daubechies 9-tap/7-tap filters. Tables 3.5 and 3.6 show the tap values of the analysis and synthesis of the 5-tap/3-tap filters.

Analysis Filter Tap values		
i	Low Pass Filter h(m)	High Pass Filter g(m)
0	0.6029490182363579	1.115087052456994
±1	0.2668641184428723	-0.5912717631142470
±2	-0.07822326652898785	-0.05754352622849957
±3	-0.01686411844287495	0.09127176311424948
±4	0.02674875741080976	

Table 3.3 Daubechies 9/7 analysis filter tap values

Synthesis Filter Tap values		
i	Low Pass Filter h'(m)	High Pass Filter g'(m)
0	1.115087052456994	0.6029490182363579
±1	0.5912717631142470	-0.2668641184428723
±2	-0.05754352622849957	-0.07822326652898785
±3	-0.09127176311424948	0.01686411844287495
±4		0.02674875741080976

Table 3.4 Daubechies 9/7 synthesis filter tap values

Analysis Filter Tap values		
i	Low Pass Filter h(m)	High Pass Filter g(m)
0	6/8	1
±1	2/8	-1/2
±2	-1/8	

Table 3.5 5/3 analysis filter tap values

Synthesis Filter Tap values		
i	Low Pass Filter h'(m)	High Pass Filter g'(m)
0	6/8	1
±1	2/8	-1/2
±2	-1/8	

Table 3.6 5/3 synthesis filter tap values

3.5.2 2-dimensional Forward Discrete Wavelet Transformation

The 2-dimensional Forward Discrete Wavelet Transformation (FDWT) is performed on every tile-component independently. The number of decomposition levels, N_L, can be different for each tile-component. Figure 3.10 shows one level of decomposition of the tile-component. The two-dimensional array of the tile-component samples are transformed in the vertical direction first and then in the horizontal direction with the same set of filters. The LL sub-band at resolution m is decomposed into four sub-bands called LL (at resolution m-1), HL, LH, and HH for each iteration.

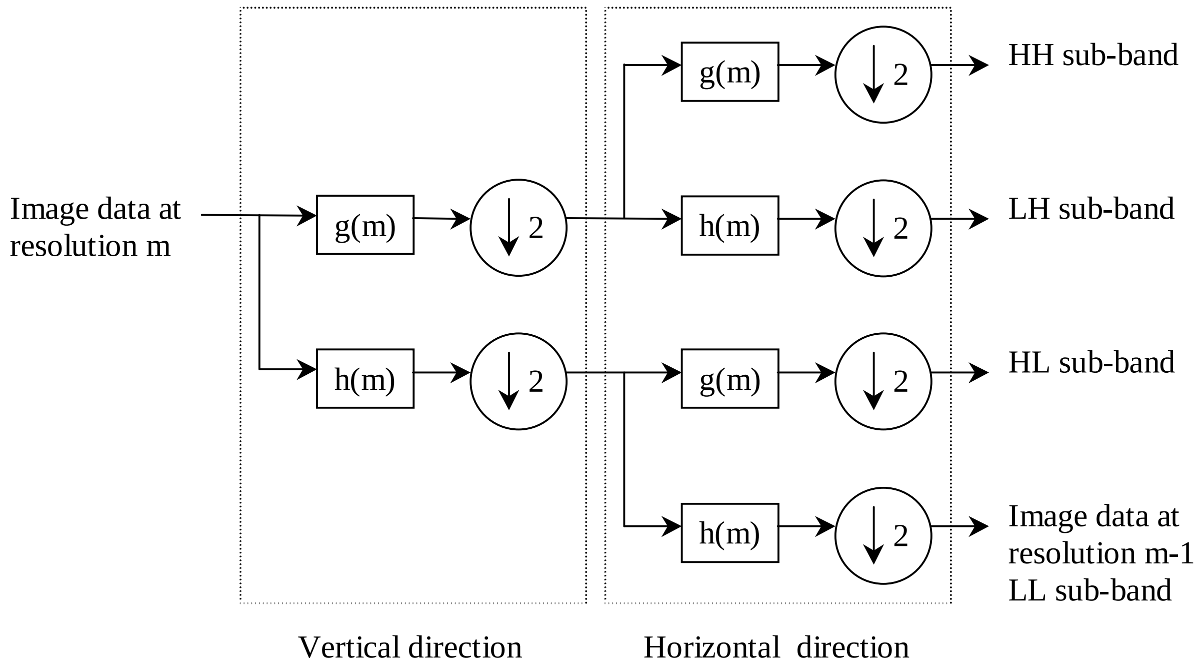


Figure 3.10 One level of decomposition of tile-component

There are $3 \times N_L + 1$ sub-bands for each tile-component. The notation for each sub-band is used in the following way: an index called “lev” corresponds to the level of decomposition and it is followed by either LL, HL, LH, or HH. The order of these sub-bands is:

N_LLL, N_LHL, N_LLH, N_LHH, (N_L-1)HL, (N_L-1)LH, (N_L-1)HH, …, 1HL, 1LH, 1HH.

For the case of $N_L = 2$, the sub-bands can be represented by Figure 3.11.

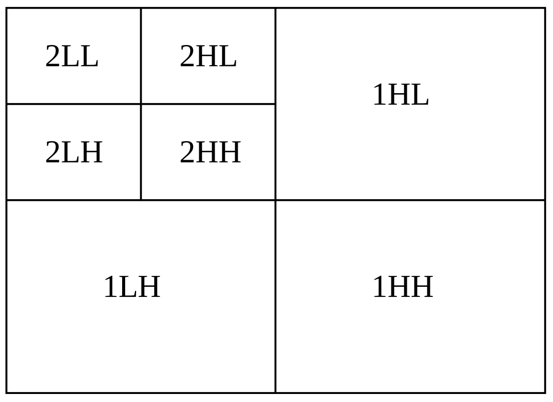


Figure 3.11 Sub-bands representation

The convolution based filtering implementation consists of many dot products (multiplications), so it is not effective for software calculation. A “Lifting” based filtering consists of a sequence of very simple operations [16], so it is chosen for JPEG-2000. For the filtering operations, the odd sample values are updated with a weighted sum of the even sample values and the even sample values are updated with a weighted sum of the odd sample values alternatively. The schemes for the two specified wavelet transformations are shown in the following procedures.

The FDWT starts with an initialization of the variable lev to 0 and sets the sample values of the tile-component I’(x,y) as the input coefficients, $a_{0LL}(u,v)$. The 2D_SD procedure is performed to the levLL sub-band in every level of the decomposition until N_L iterations are reached (see Figure 3.12).

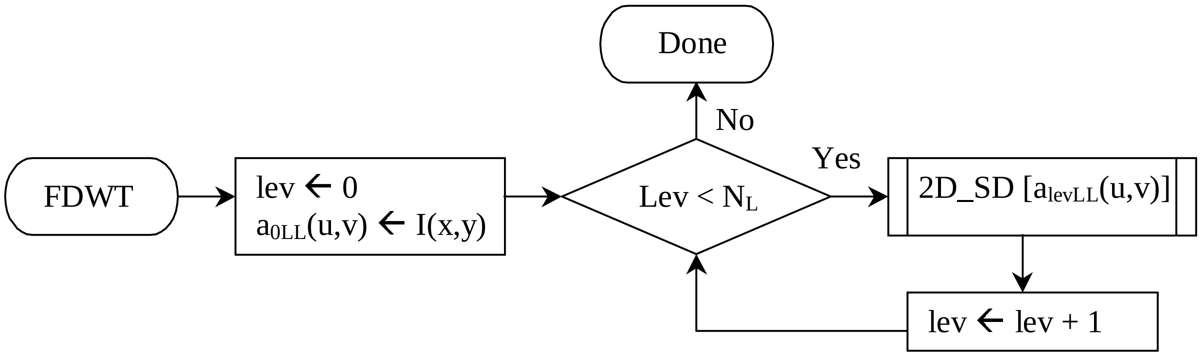


Figure 3.12 The FDWT Procedure

The 2D_SD procedure decomposes the two-dimensional data, $a_{levLL}(u,v)$, into the sub-band coefficients, $a_{(lev+1)LL}(u,v)$, $a_{(lev+1)HL}(u,v)$, $a_{(lev+1)LH}(u,v)$, and $a_{(lev+1)HH}(u,v)$. The total number of the coefficients in the four sub-bands is the same as that of the LL sub-band at one lower level. The coordinates of the upper left hand corner and the lower right hand corner of the tile-component should be supplied as the inputs. This procedure performs the decomposition in the vertical direction first via VER_SD and in the horizontal direction via HOR_SD as the second step. Interleaving the results into the position of four sub-bands by 2D_DEINTERLEAVE is the last step. Figure 3.13 describes the 2D_SD procedure.

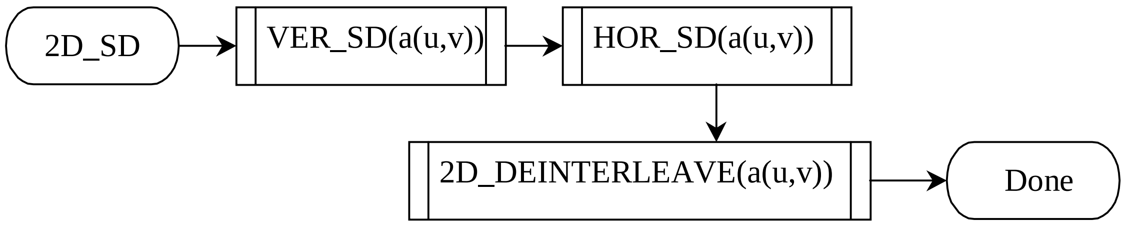


Figure 3.13 The 2D_SD Procedure

The VER_SD procedure performs the decomposition in the vertical direction via 1D_SD for all columns in the tile-component, as shown in Figure 3.14. Let (u_0,v_0) and (u_1,v_1) be the coordinates of the upper left hand corner and the lower right hand corner of the tile-component. These coordinates and $a_{levLL}(u,v)$ are the inputs to this procedure.

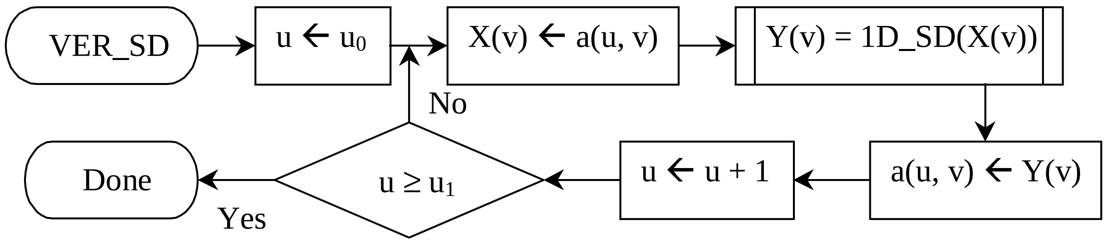


Figure 3.14 The VER_SD Procedure

The HOR_SD procedure performs the decomposition in the horizontal direction via 1D_SD for all rows in the tile-component, as shown in Figure 3.15. The coordinates (u_0,v_0), (u_1,v_1), and the data a(u,v) are the inputs to this procedure.

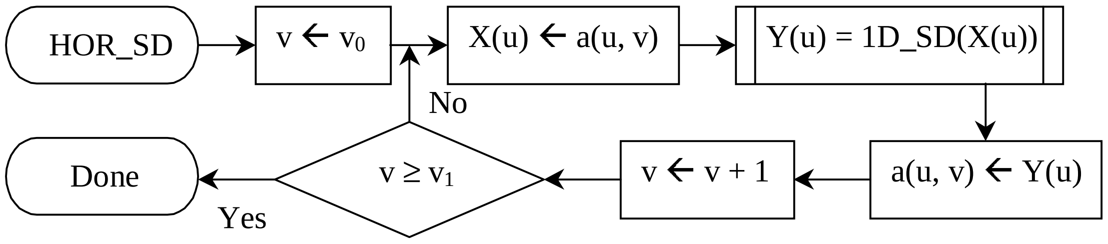


Figure 3.15 The HOR_SD Procedure

The 2D_DEINTERLEAVE procedure arranges the transformed coefficients into their corresponding sub-bands, as shown in Figure 3.16. The coordinates (u_0,v_0), (u_1,v_1), and the data $a(u,v)$ are the inputs to this procedure.

The 1D_SD procedure takes a one-dimensional array of data, X, the index i_0 of the first sample in array X, and the index i_1-1 of the last sample in array X as the inputs and produces a one-dimensional array of data, Y, with the same index as illustrated in Figure 3.17.

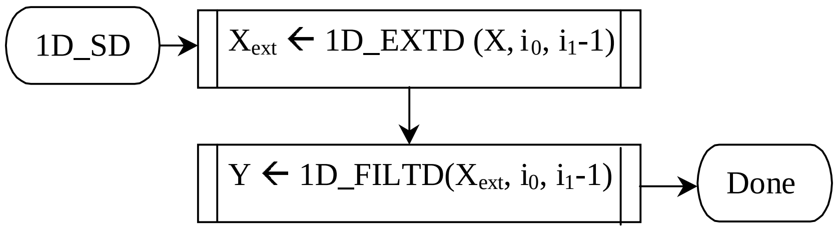


Figure 3.17 The 1D_SD Procedure

The 1D_EXTD procedure is called “periodic symmetric extension.” It extends the signal X beyond its left and right boundaries for the preparation of filtering. The output is denoted as X_{ext}. The boundaries are extended by i_{left} samples to the left and i_{right} samples to the right. The minimum but sufficiently large values of i_{left} and i_{right} for the two transformations are shown in Table 3.7.

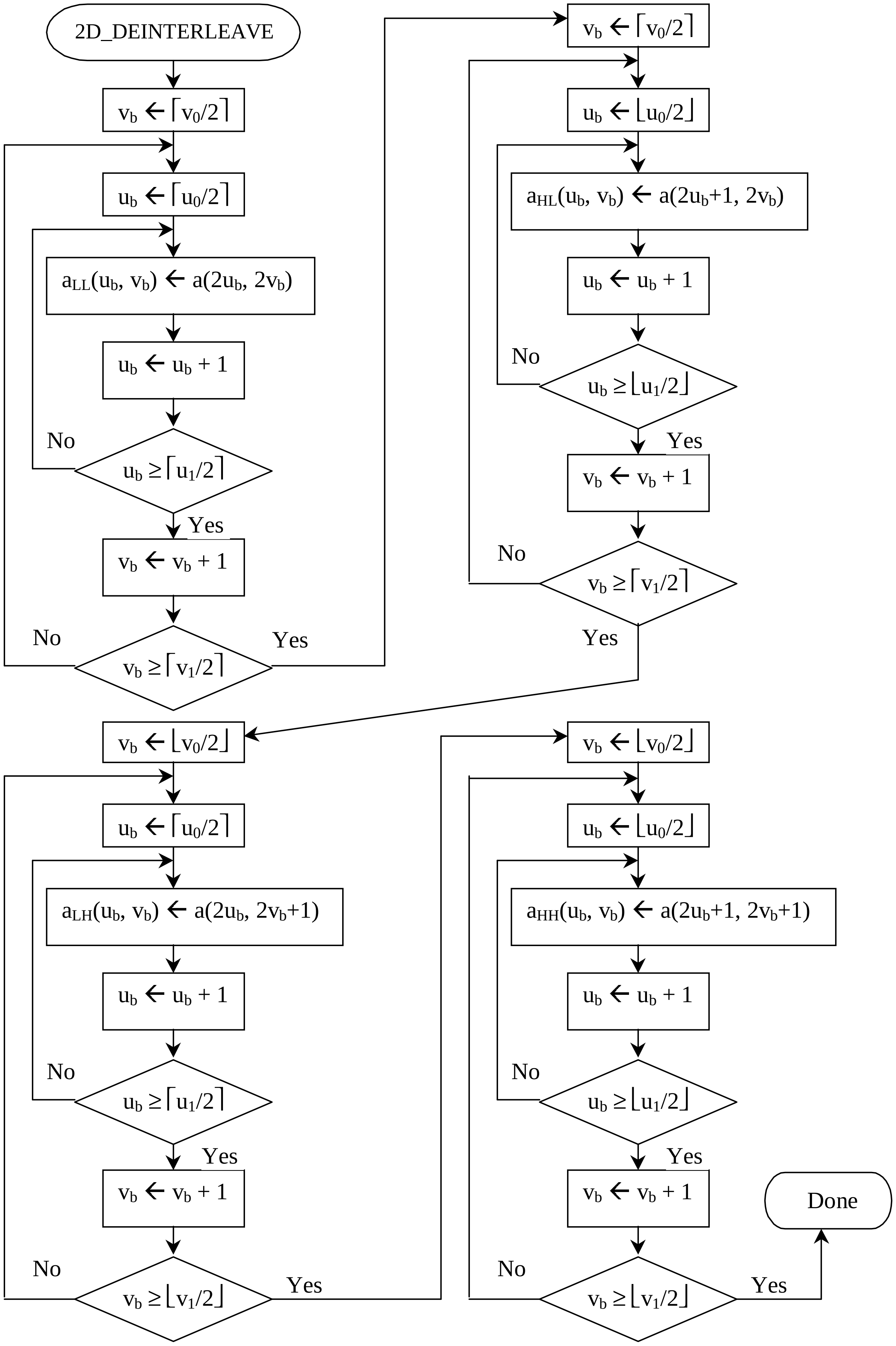

2D_DEINTERLEAVE
$v_b \leftarrow \lceil v_0/2 \rceil$
$u_b \leftarrow \lceil u_0/2 \rceil$
$a_{LL}(u_b, v_b) \leftarrow a(2u_b, 2v_b)$
$u_b \leftarrow u_b + 1$
No
$u_b \geq \lceil u_1/2 \rceil$
Yes
$v_b \leftarrow v_b + 1$
No
$v_b \geq \lceil v_1/2 \rceil$
Yes
$v_b \leftarrow \lceil v_0/2 \rceil$
$u_b \leftarrow \lfloor u_0/2 \rfloor$
$a_{HL}(u_b, v_b) \leftarrow a(2u_b+1, 2v_b)$
$u_b \leftarrow u_b + 1$
No
$u_b \geq \lfloor u_1/2 \rfloor$
Yes
$v_b \leftarrow v_b + 1$
No
$v_b \geq \lceil v_1/2 \rceil$
Yes
$v_b \leftarrow \lfloor v_0/2 \rfloor$
$u_b \leftarrow \lceil u_0/2 \rceil$
$a_{LH}(u_b, v_b) \leftarrow a(2u_b, 2v_b+1)$
$u_b \leftarrow u_b + 1$
No
$u_b \geq \lceil u_1/2 \rceil$
Yes
$v_b \leftarrow v_b + 1$
No
$v_b \geq \lfloor v_1/2 \rfloor$
Yes
$v_b \leftarrow \lfloor v_0/2 \rfloor$
$u_b \leftarrow \lfloor u_0/2 \rfloor$
$a_{HH}(u_b, v_b) \leftarrow a(2u_b+1, 2v_b+1)$
$u_b \leftarrow u_b + 1$
No
$u_b \geq \lfloor u_1/2 \rfloor$
Yes
$v_b \leftarrow v_b + 1$
No
$v_b \geq \lfloor v_1/2 \rfloor$
Yes
Done

Figure 3.16 The 2D_DEINTERLEAVE Procedure

i_0	$i_{left}(5/3)$	$i_{left}(9/7)$
Even	2	4
Odd	1	3

i_1	$i_{right}(5/3)$	$i_{right}(9/7)$
even	2	4
odd	1	3

Table 3.7 Value of i_{left} and i_{right} for extension

Symmetric extension extends the signal with the signal samples obtained by a reflection of the signal centered on the first sample for the left side and on the last sample for the right side, as shown in Figure 3.18. These extensions reduce the blocking effect at the boundaries of the tiles.

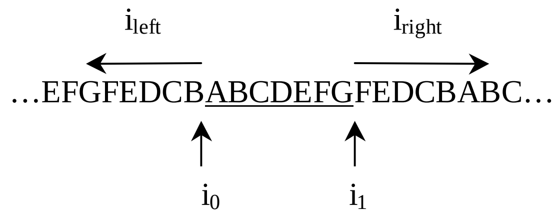


Figure 3.18 Periodic symmetric extension of signal

The 1D_FILTD procedure takes X_{ext}, i_0, and i_1-1 as the inputs and produces Y. The even coefficients of Y are a lowpass downsampled version of X_{ext}, and the odd coefficients of Y are a highpass downsampled version of X_{ext}. This procedure uses the lifting based filtering method, which is a fast algorithm to implement DWT.

For the reversible 5-tap/3-tap filters, a reversible lifting based filtering is implemented. The odd sample values of X_{ext} are updated with a weighted sum of the even sample values and the even sample values of X_{ext} are updated with a weighted sum of the odd sample values alternatively. The odd coefficients of Y are calculated first for all n that satisfy $i_0 - 1 \le 2n + 1 \le i_1 + 1$:

$$Y(2n+1) = X_{ext}(2n+1) - \left\lfloor \frac{X_{ext}(2n) + X_{ext}(2n+2)}{2} \right\rfloor .$$

Then, the even coefficients of Y are calculated for all n that satisfy $i_0 \le 2n \le i_1$:

$$Y(2n) = X_{ext}(2n) - \left\lfloor \frac{X_{ext}(2n-1) + X_{ext}(2n+1) + 2}{4} \right\rfloor \quad .$$

These calculations require a rounding procedure for the intermediate non-integer-valued transform coefficients. The values of Y(k) that satisfy $i_0 \leq k \leq i_1$ are kept as the output.

For the irreversible 9-tap/7-tap filters, an irreversible lifting based filtering is implemented. Four "lifting" steps (1 through 4) and two "scaling" steps (5 and 6) are performed on X_{ext} to produce Y:

$$Y(2n+1) \leftarrow X_{ext}(2n+1) + \{\alpha \times [X_{ext}(2n) + X_{ext}(2n+2)]\} \quad Step1$$

$$Y(2n) \leftarrow X_{ext}(2n) + \{\beta \times [Y(2n-1) + Y(2n+1)]\} \quad Step2$$

$$Y(2n+1) \leftarrow Y(2n+1) + \{\gamma \times [Y(2n) + Y(2n+2)]\} \quad Step3$$

$$Y(2n) \leftarrow Y(2n) + \{\delta \times [Y(2n-1) + Y(2n+1)]\} \quad Step4$$

$$Y(2n+1) \leftarrow -K \times Y(2n+1) \quad Step5$$

$$Y(2n) \leftarrow (1/K) \times Y(2n) \quad Step6$$

where α = -1.586134342, β = -0.052980118, γ= 0.882911075, δ= 0.443506852, and K = 1.230174105.

Step 1 is applied for all n that satisfy $i_0 - 3 \leq 2n + 1 \leq i_1 + 3$. Step 2 is then applied for all n that satisfy $i_0 - 2 \leq 2n \leq i_1 + 2$. Step 3 is then applied on all values of n such that $i_0 - 1 \leq 2n + 1 \leq i_1 + 1$. Step 4 is then applied on all values of n such that $i_0 \leq 2n \leq i_1$. Each of these steps must be performed on the entire tile-component before moving to the next step. Step 5 is performed on all values of n such that $i_0 \leq 2n + 1 \leq i_1$. Step 6 is performed on all values of n such that $i_0 \leq 2n \leq i_1$. The values of Y(k) such that $i_0 \leq k \leq i_1$ are kept as the output.

3.5.3 2-dimensional Inverse Discrete Wavelet Transformation

The 2-dimensional Inverse Discrete Wavelet Transformation (IDWT) reconstructs the DC-level shifted two-dimensional signal I'(x, y) from a set of sub-bands with coefficients $a_b(u_{b,}$ $v_b)$. Figure 3.19 shows one level of reconstruction of the tile-component.

The IDWT starts with an initialization of the variable lev and sets it to N_L. The 2D_SR procedure is performed for every level of lev and is followed by a decrement of lev. The 2D_SR procedure is iterated until lev is equal to zero. The process is finished with the final output, $a_{0LL}(u,v)$ (see Figure 3.20).

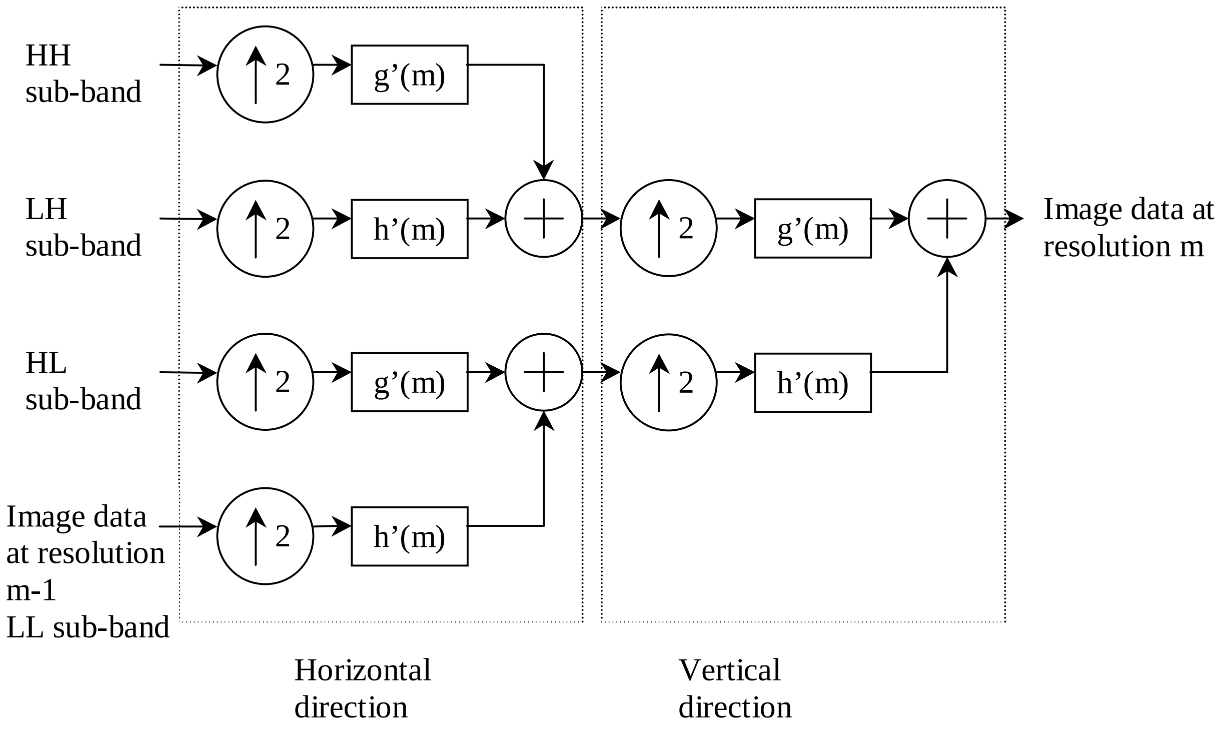
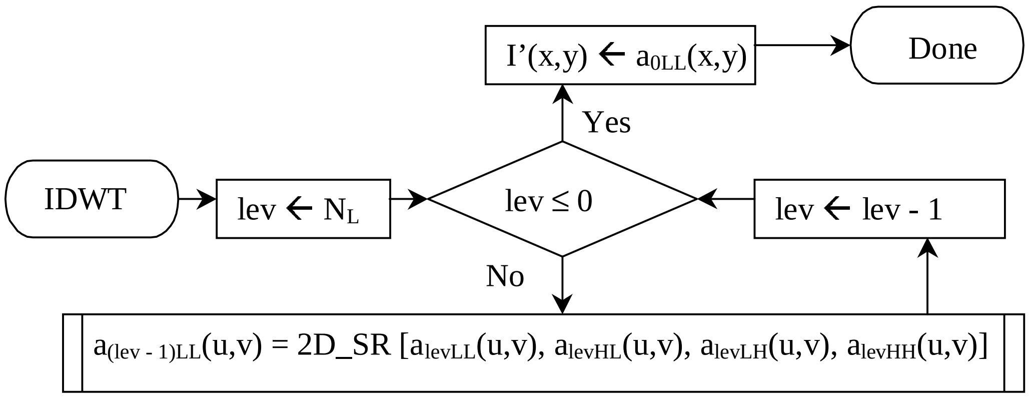


Figure 3.19 One level of reconstruction of tile-component



I'(x,y) ← $a_{0LL}(x,y)$
Done
Yes
IDWT
lev ← N_L
lev ≤ 0
lev ← lev - 1
No
$a_{(lev-1)LL}(u,v)$ = 2D_SR [$a_{levLL}(u,v)$, $a_{levHL}(u,v)$, $a_{levLH}(u,v)$, $a_{levHH}(u,v)$]

Figure 3.20 The IDWT Procedure

The 2D_SR procedure reconstructs the sub-band coefficients of $a_{levLL}(u,v)$, $a_{levHL}(u,v)$, $a_{levLH}(u,v)$, and $a_{levHH}(u,v)$ into $a_{(lev-1)LL}(u,v)$, as shown in Figure 3.21. The first step of this procedure is 2D_INTERLEAVE. It is the inverse of 2D_DEINTERLEAVE, as shown in Figure

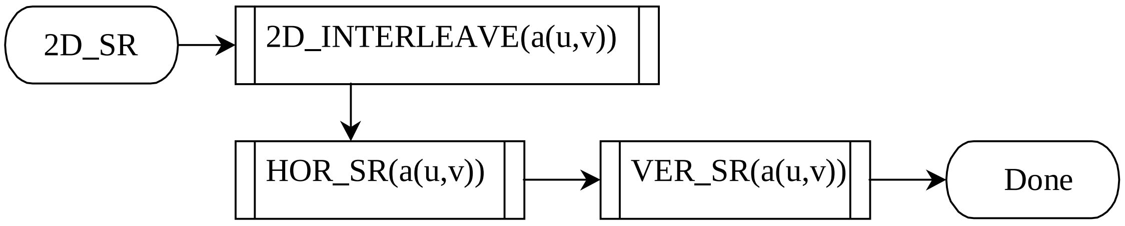


Figure 3.21 The 2D_SR Procedure

3.22. It interleaves the coefficients of four sub-bands to form a new set of a(u,v). Then, the HOR_SR procedure is applied to all rows of a(u,v) to perform the horizontal sub-band recomposition (see Figure 3.23). Let (u_0,v_0) and (u_1,v_1) be the coordinates of the upper left hand corner and the lower right hand corner of a(u,v). They are taken as the inputs to the procedure. The output is stored back into a(u,v). The VER_SR procedure is applied to all columns of a(u,v) to perform the vertical sub-band recomposition (see Figure 3.24). The coordinates (u_0,v_0) and (u_1,v_1) are again taken as the inputs. The output is stored back into a(u,v).

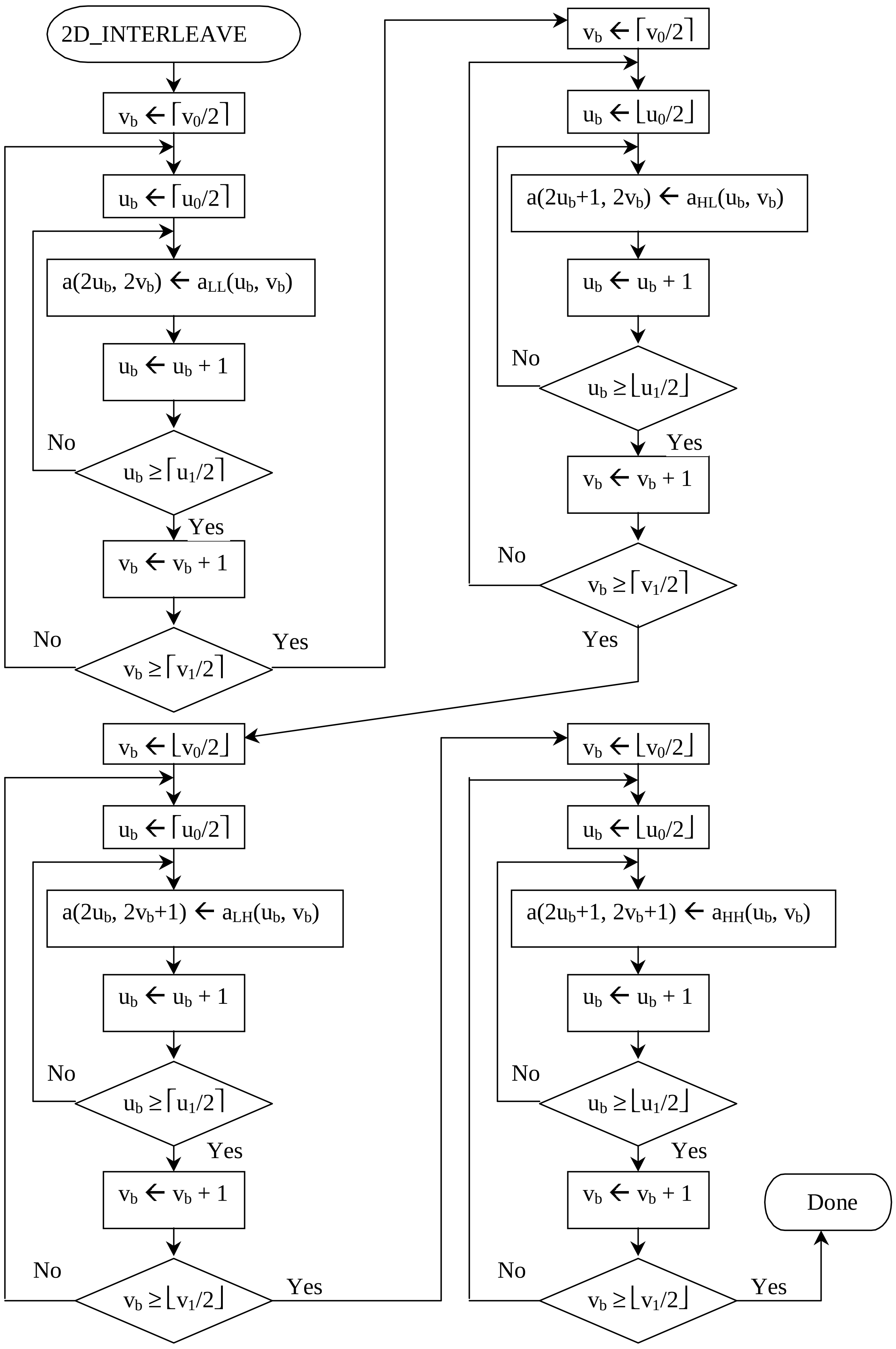

2D_INTERLEAVE
$v_b \leftarrow \lceil v_0/2 \rceil$
$u_b \leftarrow \lceil u_0/2 \rceil$
$a(2u_b, 2v_b) \leftarrow a_{LL}(u_b, v_b)$
$u_b \leftarrow u_b + 1$
No
$u_b \geq \lceil u_1/2 \rceil$
Yes
$v_b \leftarrow v_b + 1$
No
$v_b \geq \lceil v_1/2 \rceil$
Yes
$v_b \leftarrow \lceil v_0/2 \rceil$
$u_b \leftarrow \lfloor u_0/2 \rfloor$
$a(2u_b+1, 2v_b) \leftarrow a_{HL}(u_b, v_b)$
$u_b \leftarrow u_b + 1$
No
$u_b \geq \lfloor u_1/2 \rfloor$
Yes
$v_b \leftarrow v_b + 1$
No
$v_b \geq \lceil v_1/2 \rceil$
Yes
$v_b \leftarrow \lfloor v_0/2 \rfloor$
$u_b \leftarrow \lceil u_0/2 \rceil$
$a(2u_b, 2v_b+1) \leftarrow a_{LH}(u_b, v_b)$
$u_b \leftarrow u_b + 1$
No
$u_b \geq \lceil u_1/2 \rceil$
Yes
$v_b \leftarrow v_b + 1$
No
$v_b \geq \lfloor v_1/2 \rfloor$
Yes
$v_b \leftarrow \lfloor v_0/2 \rfloor$
$u_b \leftarrow \lfloor u_0/2 \rfloor$
$a(2u_b+1, 2v_b+1) \leftarrow a_{HH}(u_b, v_b)$
$u_b \leftarrow u_b + 1$
No
$u_b \geq \lfloor u_1/2 \rfloor$
Yes
$v_b \leftarrow v_b + 1$
No
$v_b \geq \lfloor v_1/2 \rfloor$
Yes
Done

Figure 3.22 The 2D_INTERLEAVE Procedure

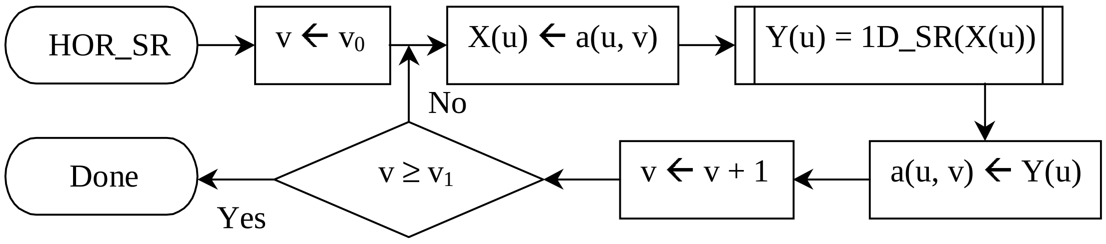


Figure 3.23 The HOR_SR Procedure

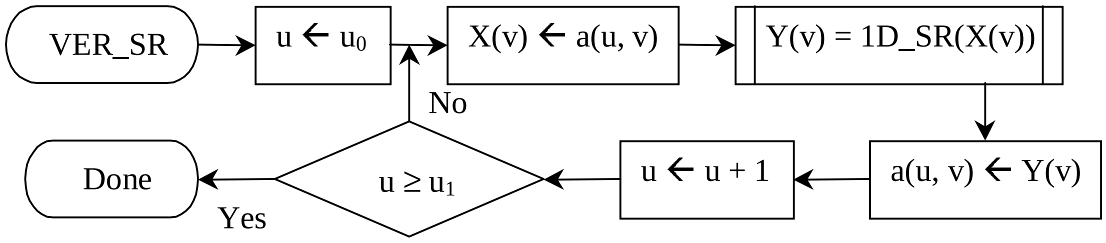


Figure 3.24 The VER_SR Procedure

The 1D_SR procedure takes a one-dimensional array, X, the index i_0 of the first sample in array X, and the index i_1-1 of the last sample in array X as the inputs and produces a one-dimensional array, Y, with the same index (see Figure 3.25).

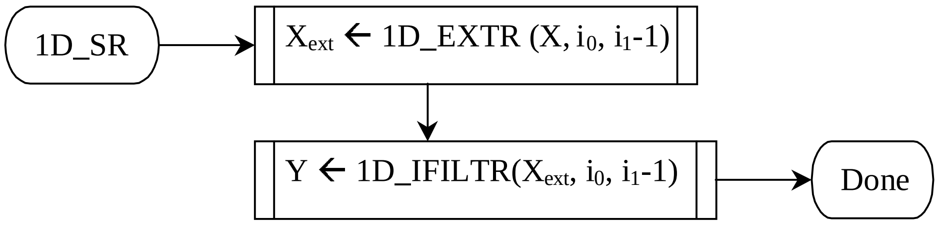

1D_SR
$X_{ext} \leftarrow 1D\_EXTR (X, i_0, i_1-1)$
$Y \leftarrow 1D\_IFILTR(X_{ext}, i_0, i_1-1)$
Done

Figure 3.25 The 1D_SR Procedure

The 1D_EXTR procedure is applied to extend the signal X beyond its boundaries to produce X_{ext}. This procedure is identical to the 1D_EXTD procedure.

The 1D_IFILTR procedure takes the X_{ext}, i_0, and i_1-1 as the inputs and produces Y. This procedure also uses the lifting based filtering.

For the reversible 5-tap/3-tap filters, a reversible lifting based filtering is implemented. The odd sample values of X_{ext} are updated with a weighted sum of the even sample values and the even sample values of X_{ext} are updated with a weighted sum of the odd sample values alternatively. The even coefficients of Y are calculated first for all n that satisfy $i_0 - 1 \leq 2n \leq i_1 - 1$:

$$Y(2n) = X_{ext}(2n) - \left\lfloor \frac{X_{ext}(2n-1) + X_{ext}(2n+1) + 2}{4} \right\rfloor \quad .$$

Then, the odd coefficients of Y are calculated for all n that satisfy $i_0 \leq 2n + 1 \leq i_1$:

$$Y(2n+1) = X_{ext}(2n+1) - \left\lfloor \frac{Y(2n) + Y(2n+2)}{2} \right\rfloor \quad .$$

These calculations require a rounding procedure for the intermediate non-integer-valued transform coefficients.

For the irreversible 9-tap/7-tap filters, an irreversible lifting based filtering is implemented. Two “scaling” steps (1 and 2) and four “lifting” steps (3 through 6) are performed on X_{ext} to produce Y:

$$Y(2n) \leftarrow K \times X_{ext}(2n) \quad Step1$$

$$Y(2n+1) \leftarrow -(1/K) \times X_{ext}(2n+1) \quad Step2$$

$$Y(2n) \leftarrow Y(2n) - \{\delta \times [Y(2n-1) + Y(2n+1)]\} \quad Step3$$

$$Y(2n+1) \leftarrow Y(2n+1) - \{\gamma \times [Y(2n) + Y(2n+2)]\} \quad Step4$$

$$Y(2n) \leftarrow Y(2n) - \{\beta \times [Y(2n-1) + Y(2n+1)]\} \quad Step5$$

$$Y(2n+1) \leftarrow Y(2n+1) - \{\alpha \times [Y(2n) + Y(2n+2)]\} \quad Step6$$

where α = -1.586134342, β = -0.052980118, γ= 0.882911075, δ= 0.443506852, and K = 1.230174105.

Step 1 is applied for all n that satisfy $i_0 - 3 \le 2n \le i_1 + 3$. Step 2 is then applied for all n that satisfy $i_0 - 2 \le 2n + 1 \le i_1 + 2$. Step 3 is then applied on all values of n such that $i_0 - 3 \le 2n \le i_1 + 3$. Step 4 is then applied on all values of n such that $i_0 - 2 \le 2n + 1 \le i_1 + 2$. Step 5 is performed on all values of n such that $i_0 - 1 \le 2n \le i_1 + 1$. Step 6 is performed on all values of n such that $i_0 \le 2n + 1 \le i_1$.

3.6 Quantization

For the 5-tap/3-tap wavelet transformation, no quantization is used to reduce the precision of the coefficients. That means the quantization step is one and the coefficients have integer values. On the other hand, for the 9-tap/7-tap wavelet transformation, each sub-band from a tile-component can have its own quantization step value. The quantization step, Δ_b, for sub-band b is specified by the following equation:

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

where R_b is the nominal dynamic range for sub-band b. It is the sum of the number of bits that are used to represent the original source image tile-component. The exponent/mantissa pairs (ε_b, μ_b) are either signaled for all sub-bands or for the LL sub-band only. In the latter case, the exponent/mantissa pairs (ε_b, μ_b) are determined from the exponent/mantissa pair (ε_0, μ_0) corresponding to the LL sub-band, according to the following equation:

$$(\varepsilon_b, \mu_b) = (\varepsilon_0 + nsd_b - nsd_0, \mu_0)$$

where nsd_b denotes the number of sub-band decomposition levels from the original image tile-component to the sub-band b. Therefore, ε_b for the lower frequency sub-bands tend to be larger and make the quantization steps for these sub-bands to be smaller. Therefore, less distortion is resulted from the quantization error.

Each of the wavelet transformed coefficients, $a_b(u,v)$, of the sub-band b is quantized into $q_b(u,v)$ according to the following equation:

$$q_b(u,v) = sign(a_b(u,v)) \times \left\lfloor \frac{|a_b(u,v)|}{\Delta_b} \right\rfloor \quad .$$

M_b is the expected maximum number of encoded bit-planes for sub-band b. It is calculated by using the following equation:

$$M_b = G + \varepsilon_b - 1,$$

where G is the number of guard bits.
The typical values of G are 1 or 2. The purpose of G is to prevent possible overflow beyond the nominal range of the integer representation of $|q_b(u,v)|$.

The decoder may decide to decode only N_b bit-planes ($M_b > N_b$) for a particular code block due to the embedded nature of the code stream. Therefore, the actual quantization step is $2^{M_b - N_b}$ multiplied by Δ_b for the samples in that code block. Because of the nature of three coding passes for each bit-plane, a truncation of bit stream may also occur between the passes within a bit-plane. Thus, the actual quantization step may be different for different samples even within the same code block if one bit-plane is not completely decoded. However, these quantization step-sizes are always multiples of the reference quantization step by some power of two. Each decoded coefficient, $q'_b(u,v)$, is expressed in a sign magnitude representation and the non decoded bits are set to zero. Then, they are dequantized back into wavelet transform coefficient, $Rq'_b(u,v)$.

For the 9-tap/7-tap wavelet transformation, the following equations are applied.

$$Rq'_b(u,v) = \begin{cases} (q'_b(u,v) + r2^{M_b - N_b(u,v)}) \times \Delta_b & \text{for } q'_b(u,v) > 0 \\ (q'_b(u,v) - r2^{M_b - N_b(u,v)}) \times \Delta_b & \text{for } q'_b(u,v) < 0 \\ 0 & \text{for } q'_b(u,v) = 0, \end{cases}$$

where $N_b(u,v)$ is the number of decoded bit-planes for sample $q'_b(u,v)$ and r is the coefficient reconstruction value. r's value is between 0 and 1 ($0 \le r < 1$). It can be chosen for the best visual or objective quality. The typical value of r is 0.5.

For the 5-tap/3-tap wavelet transformation, the dequantization process is slightly different depending on whether a truncation of bit stream has been made. If no truncation occurs, $Rq'_b(u,v) = q'_b(u,v)$. Otherwise, the following equations are applied:

$$RQ_b(u,v) = \begin{cases} \lfloor (Q_b(u,v) + r2^{M_b - N_b(u,v)}) \rfloor \times \Delta_b & \text{for } Q_b(u,v) > 0 \\ \lfloor (Q_b(u,v) - r2^{M_b - N_b(u,v)}) \rfloor \times \Delta_b & \text{for } Q_b(u,v) < 0 \\ 0 & \text{for } Q_b(u,v) = 0. \end{cases}$$

For 9-tap/7-tap wavelet transformation, no preference of any quantization step-size is specified and different applications may set the step-sizes according to the image's characteristics.

3.7 Coefficient Bit Modeling

After quantization, the coefficients within each code block are separated into bit-planes. These bit-planes are coded starting from the most significant one with at least one non-zero element to the least significant one. A special scan pattern is performed on each bit-plane for each of the three coding passes. Each coefficient bit is coded in exactly one of the three coding passes. They are called significance propagation pass, magnitude refinement pass, and cleanup pass. For each pass, “context value” is determined for the coded bit. These context values and the bit stream are the inputs for the arithmetic coder. EBCOT has a large contribution to the techniques mentioned above [17].

3.7.1 Bit-Plane

The coefficients are represented by sign magnitude binary numbers. A code block consists of a rectangular array of these coefficients. A sequence of binary arrays is formed with one bit from each of the coefficients. The first such array includes the most significant bit from each coefficient. The second array includes the next most significant bit from each coefficient. This process continues until the last array contains the least significant bit from each coefficient. These arrays are called bit-planes. The number of bit-planes that are coded for a code block is M_b-P. (Section 3.4.8.4)

3.7.2 Scan Pattern within Code Block

Starting from the first bit-plane of a code block that contains at least one non-zero element, each bit-plane is scanned in a specified order for each pass. The first four bits of the first column at the top left hand corner of a bit-plane are scanned, then the first four bits of the second column are scanned and so on. After the first four bits of the last column are scanned, the second four bits of the first column are scanned, as shown in Figure 3.26. This scan pattern is continued until the bit at the lower right hand corner is scanned.

1	5	9	13	17	21	25
2	6	10	14	18	22	26
3	7	11	15	19	23	27
4	8	12	16	20	24	28
29	33	…				
30	34	…				
31	35	…				
32	36	…				

Figure 3.26 Scanning order of a bit plane

3.7.3 Coding Passes over the Bit-Planes

Each coefficient in a code block has a binary state variable called “significance state.” It is initialized to 0 and is changed to 1 at the bit-plane where the most significant 1 bit of the coefficient is found (not the sign bit). A “context” vector of a coefficient is defined as a binary vector consisting of the significance states of the 8 surrounding neighbors, as shown in Figure 3.27. If any of these eight neighbors is not lying in the same code block as X, it is considered as

D_0	V_0	D_1
H_0	X	H_1
D_2	V_1	D_3

Figure 3.27 Surrounding neighbors of coefficient X

insignificant in order to achieve the independence of coding between code blocks. Coefficient X can have $2^8 = 256$ different values of context vector. These values are classified into smaller number of context labels according to the rules, which are different for each of the four coding operations. They are significance coding, sign coding, magnitude refinement coding and cleanup coding. These coding operations are performed in the three coding passes for each bit-plane in the following order:

(1) significance and sign coding operations in the significance propagation pass,

(2) magnitude refinement coding in the magnitude refinement pass,

(3) cleanup and sign coding operations in the cleanup pass.

The first bit-plane with at least one non-zero element has a cleanup pass only since there can be no predicted significance or refinement bits. The remaining bit-planes are coded with all three coding passes. Each coefficient bit is coded in exactly one of the three coding passes.

3.7.3.1 Significance Propagation Pass

Empirical evidence suggests that the sample statistics are approximately Markov: the significance state of a sample depends only upon the significance states of its immediate eight neighbors, which are indicated in the context values. The 256 different context values are classified into 9 context labels according to Table 3.8 where $\sum H$ means $H_0 + H_1$. $\sum V$ means $V_0 + V_1$ and $\sum D$ means $D_0 + D_1 + D_2 + D_3$. They are the sum of the significance states. x means that we do not care the value.

LL and LH sub-bands			HL sub-band			HH sub-band		Context Label
$\sum$H	$\sum$V	$\sum$D	$\sum$H	$\sum$V	$\sum$D	$\sum$(H+V)	$\sum$D	
2	x	X	X	2	x	x	≥3	8
1	≥1	X	≥1	1	x	≥1	2	7
1	0	≥1	0	1	≥1	0	2	6
1	0	0	0	1	0	≥2	1	5
0	2	X	2	0	x	1	1	4
0	1	X	1	0	x	0	1	3
0	0	≥2	0	0	≥2	≥2	0	2
0	0	1	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0

Table 3.8 Contexts for the significance propagation pass and cleanup pass

The mapping of context labels depends on which sub-band the code block is in. This mapping can minimize both the model adaptation cost and implementation complexity. The table is constructed by exploring the symmetries in vertical, horizontal, and diagonal directions of the configuration of Figure 3.27. The context label assignment for the LH and HL sub-bands are identical if we exchange the vertical and horizontal directions (transposition) within the code block. The LH sub-band responds strongly to vertically oriented features while the HL sub-band responds strongly to the horizontally oriented features. These behaviors also explain part of the design of Table 3.8.

The bits that were insignificant and have a non-zero context label are included in this pass. Other bits in the same bit-plane are coded either in the magnitude refinement pass or the cleanup pass. The context labels and the bit stream are sent to the arithmetic coder. For the decoding part, the significance state of the coded coefficients in this pass is set to 1 if the decoded bit is 1 and the immediate next bit to be decoded is the sign bit for the coefficient. Otherwise, the significance state remains 0. When we consider the context labels of successive coefficients and coding passes, the most recent significance state for this coefficient is applied.

3.7.3.2 Sign Bit Coding

The context label of a coefficient for sign bit coding is determined through two steps. The first step calculates the vertical and horizontal contributions from V_0, V_1, H_0 and H_1 according to Table 3.9. Each of them can have one of three states: insignificant, significant positive, or significant negative. From Table 3.9, we see that the contribution takes the average of V_0 and V_1 (H_0 and H_1).

V_0 (or H_0)	V_1 (or H_1)	V (or H) contribution
Significant, positive	significant, positive	1
Significant, negative	significant, positive	0
Insignificant	significant, positive	1
Significant, positive	significant, negative	0
Significant, negative	significant, negative	-1
Insignificant	significant, negative	-1
Significant, positive	insignificant	1
Significant, negative	insignificant	-1
Insignificant	insignificant	0

Table 3.9 Vertical (and Horizontal) contributions from four neighbors to the sign context

The second step simplifies the nine permutations of vertical and horizontal contributions to five context labels according to Table 3.10. The context labels and the bit stream are sent to

Horizontal contribution	Vertical contribution	Context label	XOR bit
1	1	13	0
1	0	12	0
1	-1	11	0
0	1	10	0
0	0	9	0
0	-1	10	1
-1	1	11	1
-1	0	12	1
-1	-1	13	1

Table 3.10 Sign contexts from vertical and horizontal contributions

the arithmetic coder. For the decoding part, a bit is returned from the arithmetic decoder. This bit is then logically XORed with the XORbit in the last column of Table 3.10 to produce the sign bit, as shown in the following equation:

$$\textit{Sign bit} = \textit{Arithmetic Decoder (context label, compressed bit stream)} \oplus \textit{XORbit}\ .$$

When XORbit is 1 as for the last four rows of Table 3.10, the returned bit from arithmetic decoder is switched. Sign bit of 1 means negative while 0 means positive.

3.7.3.3 Magnitude Refinement Pass

The bits from coefficients that are already significant (except those that become significant in the immediately proceeding significance propagation pass) are included in this pass. The context label of a coefficient is determined by the summation of the significance states of all eight neighbors and whether the encoded bit is the first refinement bit (the bit immediately after the significance and sign bit), as shown in Table 3.11. Only three context labels are used since very weak correlation exists between any previously encoded bit-plane and the magnitude of the neighboring coefficients.

$\sum H + \sum V + \sum D$	First refinement for this coefficient	Context label
X	False	16
≥1	True	15
0	True	14

Table 3.11 Contexts for the magnitude refinement coding passes

3.7.3.4 Cleanup Pass

If a bit is not coded in the previous two passes, it must be coded in the cleanup pass. The context label for this bit is determined using Table 3.12. A unique single context is also created for run-length. If four contiguous coefficients in the column being scanned are all included in the cleanup pass and the context labels for them are all 0, then the run-length context and the bit stream are given to the arithmetic coder. For the arithmetic decoding part, if the symbol 0 is returned, then all four coefficients remain insignificant. Otherwise, symbol 1 is returned indicating that at least one of the four coefficients is significant. The next two bits returned with the UNIFORM context denote which coefficient is the first one to be found significant, as shown in Table 3.12. The sign bit of that coefficient is handled using the process in Section 3.7.3.2. The remaining coefficients in the column are coded using the process in Section 3.7.3.1. If not all four contiguous coefficients are included in this pass or the context label of any of them is not 0, then they are coded as in the significance propagation pass in Section 3.7.3.1. The scheme for this pass is summarized in Table 3.12. If there are fewer than four rows at the end of the code block, then run-length coding is not applied.

Four contiguous coefficients coded in cleanup pass and all have the 0 context	Symbols returned from run-length context	Four contiguous bits to be coded are zero	Symbols coded with UNIFORM context	Number of coefficients to code
true	0	true	none	none
true	1	False	MSB LSB	
		skip to 1st coefficient sign	0 0	3
		skip to 2nd coefficient sign	0 1	2
		skip to 3rd coefficient sign	1 0	1
		skip to 4th coefficient sign	1 1	0
false	none	x	none	rest of column

Table 3.12 Run-length coder for cleanup passes

3.7.3.5 Example of Coding Passes

Table 3.13 shows an example of coding four coefficients in a column. The coefficients that are not shown in this table are assumed to be zero. This table indicates which bit is included in which pass. The sign bit is indicated by a +/- sign beside the initial 1 bit.

Coding Pass	Coefficient Value			
	10	1	3	-7
Clean-up	1+	0	0	0
Significance		0		
Refinement	0			
Clean-up			0	1-
Significance		0	1+	
Refinement	1			1
Clean-up				
Significance		1+		
Refinement	0		1	1
Clean-up				

Table 3.13 Example of bit-plane coding order

3.7.4 Initialization and Termination

All context labels are initialized or re-initialized with indexes (probabilities) according to Table 3.14. The context labels are re-initialized either at the end of each coding pass or at the end of each code block. The arithmetic coder is terminated either at the end of each coding pass or at the end of each code block.

Context	Initial index from Table 3.18	MPS
UNIFORM	46	0
Run-length	3	0
All zero neighbors (context label 0)	4	0
All other contexts	0	0

Table 3.14 Initial states for all context labels

3.7.5 Error Resilience Segmentation Symbol

A segmentation symbol is optionally coded with the UNIFORM context of the arithmetic coder at the end of each bit-plane. It is applied as an error detection method. The correct arithmetic decoding of this symbol indicates that no error occurs in the corresponding bit-plane.

A segmentation symbol of “1010” should be decoded at the end of each bit-plane. If “1010” is not decoded, bit errors occur for the corresponding bit-plane.

3.7.6 Flow Chart of the Code Block Coding

The steps for coding a bit-plane in a code block can be shown by a flow chart in Figure 3.28. The decisions made are listed in Table 3.15. The processes of sending bits and context labels are listed in Table 3.16.

Decision	Question	Description
D0	Is this the first significance bit-plane for the code block?	Section 3.5.3
D1	Is the current coefficient significant?	Section 3.5.3.1
D2	Is the context label zero?	Section 3.5.3.1
D3	Did the current coefficient just become significant?	Section 3.5.3.1
D4	Are there more coefficients in the significance propagation pass?	
D5	Is the coefficient insignificant?	Section 3.5.3.3
D6	Was the coefficient coded in the last significance propagation pass?	Section 3.5.3.3
D7	Are there more coefficients in the magnitude refinement pass?	
D8	Are four contiguous uncoded coefficients in a column each with a 0 context label?	Section 3.5.3.4
D9	Is the coefficient significant?	Section 3.5.3.4
D10	Are there more coefficients remaining of the four contiguous coefficients?	
D11	Are the four contiguous bits all 0?	Section 3.5.3.4
D12	Are there more coefficients in the cleanup pass?	

Table 3.15 Decisions in the coding passes flow chart

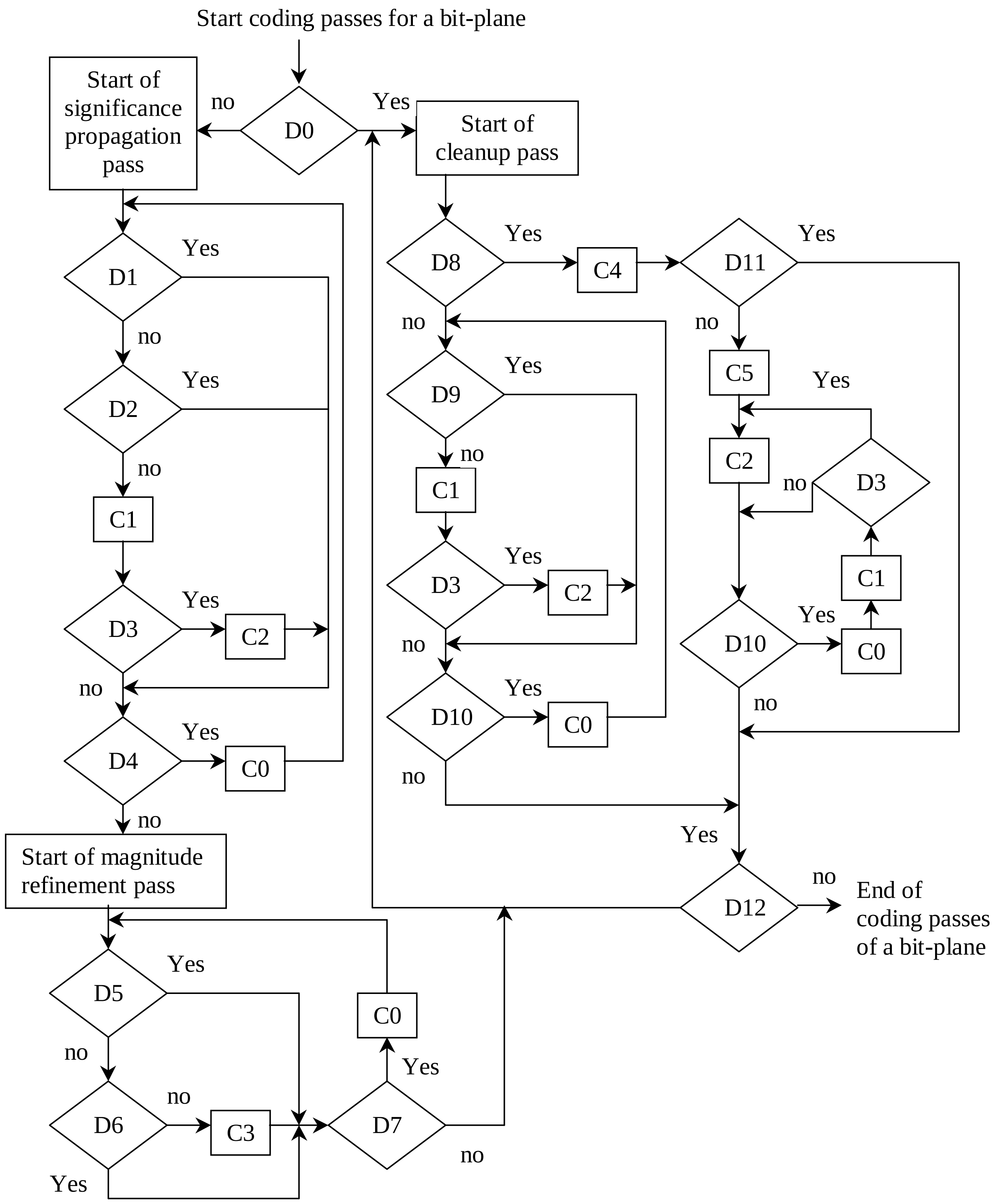


Figure 3.28 Flow chart for all coding passes for a bit-

Code	Coded symbol	Context	Explanation	Description
C0	-	-	Go to next coefficient or column	
C1	Newly significant?	Table 3.7	Code significant bit of current coefficient	Section 3.5.3.1
C2	Sign bit	Table 3.9	Code sign bit of current coefficient	Section 3.5.3.2
C3	Current magnitude Bit	Table 3.10	Code magnitude refinement bit of current coefficient	Section 3.5.3.3
C4	0 1	Run-length context label	Code run-length of four zeros Code run-length not of four zeros	Section 3.5.3.4
C5	00 01 10 11	UNIFORM	1st coefficient is the first significant coefficient 2nd coefficient is the first significant coefficient 3rd coefficient is the first significant coefficient 4th coefficient is the first significant coefficient	Section 3.5.3.4

Table 3.16 Coding in the coding passes flow chart

3.8 Arithmetic Entropy Coder

3.8.1 Basic principles of Arithmetic Coding

In arithmetic coding, all the symbols are ordered on a number line in the interval from 0 to 1 in a pattern that is known to both the encoder and decoder. Each symbol has its subinterval with length that is equal to its probability on the number line. Since the sum of the probabilities of all symbols is 1, the subintervals exactly fill the interval from 0 to 1 on the number line. To code a symbol, a code stream of binary fraction pointing to the subinterval corresponding to the symbol is created. The boundary between two symbols is assigned to the upper subinterval in JPEG-2000. If a symbol occupies the subinterval from 0.5 to 1, a binary fraction, x, from the range of 0.5 to 1 ($0.5 \leq x < 1$) is acceptable as a code stream. The decoder can determine which subinterval is pointed to by the code stream and thus decode the corresponding symbol. The process of subdivision of a subinterval into even smaller subintervals is used to code a sequence of symbols. The length of the subinterval is proportional to the probability of the corresponding sequence of symbols.

3.8.2 Binary Arithmetic Coding

In binary arithmetic coding, only the symbol of 0 and 1 are used. Therefore, a translation of a multi-symbol into a sequence of symbols is required for JPEG-2000. The code stream that is created by the coefficient bit modeling (Section 3.7) for a code block is a multi-symbol that is translated to a sequence of “decisions” (D). The decisions (D) and the context labels (CX) are the inputs for the arithmetic encoder to produce the compressed data. CX provide the probability estimates for D during arithmetic encoding and decoding. The binary fraction pointer, “code string”, is chosen to point to the base (the lower bound) of the probability subinterval.

When a current interval is partitioned into two subintervals, the subinterval for the “more probable symbol” (MPS) is ordered above the subinterval for the “less probable symbol” (LPS). When an MPS is coded, the corresponding LPS subinterval is added to the code string. An MPS symbol can be 0 or 1. This is also true for an LPS. When a D is decoded, the decoder subtracts any subinterval that is added to the code string by the encoder.

Fixed precision integer arithmetic is used for the coding operations. The decimal of 0.75 is represented by a hexadecimal of 8000. For convenience, hexadecimal numbers have 0x as their prefix throughout this thesis. The initial probability interval, A, is kept in the range from 0.75 to 1.5 ($0.75 \le A < 1.5$) by doubling it whenever it falls below 0x8000. The code string register, C, is also doubled whenever A is doubled. This doubling process is called “renormalization.” To avoid C register from overflowing, the high order bits of it is removed periodically and transferred to a buffer.

Keeping A between 0.75 and 1.5 allows an arithmetic approximation to be applied to interval subdivision. The probability estimate of an LPS is denoted by Qe. The actual subintervals for an MPS and LPS are calculated as follow:

$$A - (Qe \times A) = \text{subinterval for an MPS}$$

$$Qe \times A = \text{subinterval for an LPS}$$

Since A is close to 1 at all times, these subintervals can be approximated by

$$A - Qe = \text{subinterval for an MPS}$$

$$Qe = \text{subinterval for an LPS}$$

When an MPS is coded, Qe is added to C and A is reduced to A – Qe. When an LPS is coded, C keeps the same value and A is changed to Qe.

Due to the approximation made above, an LPS subinterval is sometimes longer than the corresponding MPS subinterval. If this occurs, the two subintervals are exchanged. This conditional exchange can only occur when a renormalization is needed.

When a renormalization occurs, the probability estimate is updated for the context label currently being coded. No explicit counting of any symbol is required for this estimation. The probability of renormalization after coding an MPS or LPS provides an approximate symbol counting scheme, which is used to estimate the probabilities of all symbols.

3.8.3 Arithmetic Encoder

The structures of the C and A registers are shown in Table 3.17.

	MSB			LSB
C register	0000 cbbb	bbbb bsss	xxxx xxxx	xxxx xxxx
A register	0000 0000	0000 0000	aaaa aaaa	aaaa aaaa

Table 3.17 Structures of encoder registers

The length of A interval is represented by the “a” bits. The “x” bits are the fractional bits in C. The “s” bits are spacer bits, which provide constraint on carry over and reduce the probability of carry over propagation in the “b” bits. The “b” bits are bits that are removed to the buffer periodically. The “c” bit is carry bit.

The ENCODER in Figure 3.29 initializes the arithmetic encoding by the INITENC procedure. CX and D pairs are read as the inputs to the ENCODE procedure until all pairs are read. The FLUSH procedure outputs the last few bytes of compressed data that are left in C and terminates the arithmetic encoding.

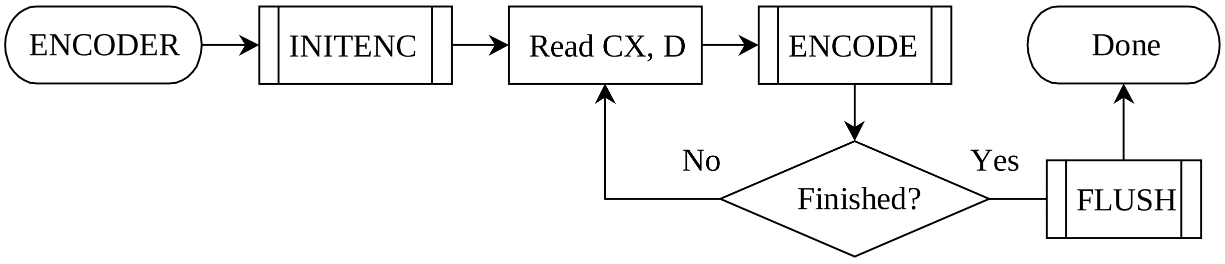


Figure 3.29 Encoder Structure

3.8.3.1 Encoding a Decision (ENCODE)

The ENCODE procedure has two paths. One path is for D equal to 0 and the other one is for D equal to 1. Then a CODE0 or CODE1 procedure is followed respectively (see Figure 3.30).

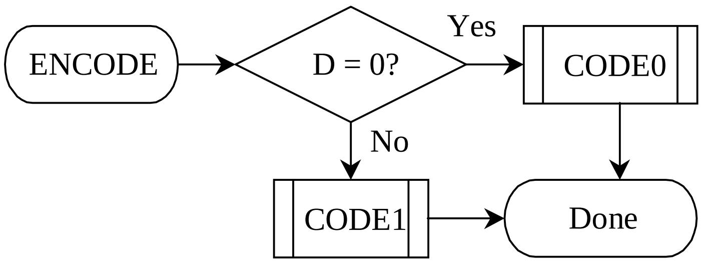


Figure 3.30 ENCODE procedure

3.8.3.2 Encoding a 0 or 1 (CODE0 and CODE1)

For both CODE0 and CODE1 procedures (see Figures 3.31 and 3.32), the decision is either an MPS or LPS. One of the two procedures, CODEMPS and CODELPS, is called appropriately for the decision.

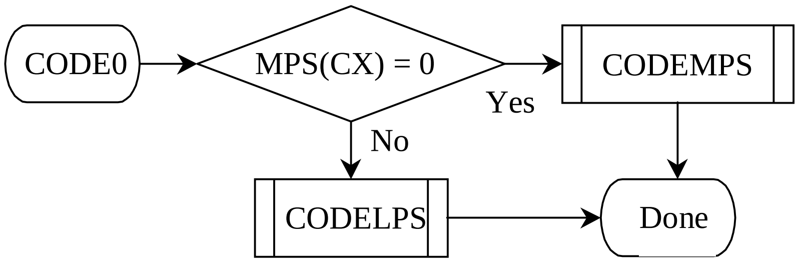


Figure 3.31 CODE0 procedure

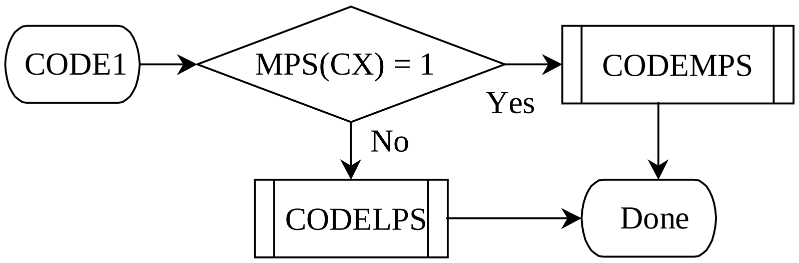


Figure 3.32 CODE1 procedure

CX is the context label that determines the index of probability estimate. The MPS value is initialized as in Table 3.14 and is switched as explained in a later part of this chapter. MPS(CX) is called the sense (0 or 1) of the MPS for context label CX.

3.8.3.3 Encoding an MPS or LPS (CODEMPS and CODELPS)

In the CODELPS procedure (see Figure 3.33), the interval A is usually reduced to Qe(I(CX)), which is the probability estimate of the LPS that is determined from the index I stored for context label CX. However, we have to check whether the subinterval of the MPS (A – Qe(I(CX))) is actually larger than the subinterval of the LPS. If it is not, a conditional exchange is performed. A SWITCH(I(CX)) flag (see Table 3.18) is set when Qe(I(CX)) is greater than 0.5 because the LPS becomes an MPS. This flag switches the sense of the MPS from 0 to 1 or from 1 to 0. A renormalization (RENORME) is always required in this procedure and the probability estimate is updated before the normalization. The next LPS index (NLPS) column in Table 3.18 shows the updated probability estimate index.

In the CODEMPS procedure (see Figure 3.34), the interval A is usually reduced to A - Qe(I(CX)), the subinterval for MPS and Qe(I(CX)) is added to C so that it points to the base of the MPS subinterval. However, if the subinterval of the LPS, Qe(I(CX)), is actually larger than the subinterval of the MPS, a conditional exchange is performed. This conditional exchange

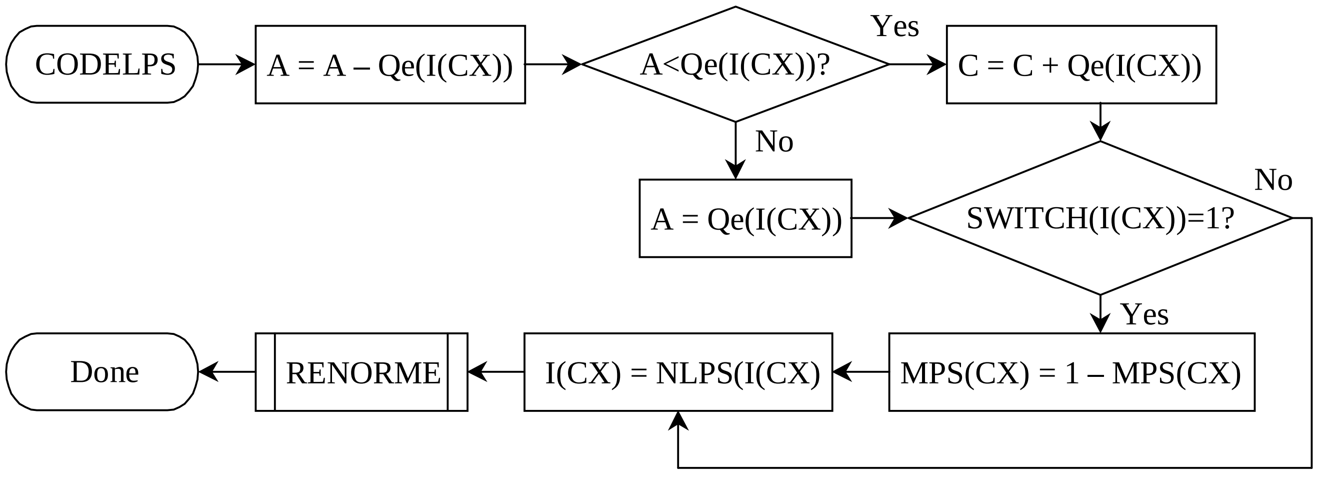


Figure 3.33 CODELPS procedure

cannot occur unless a renormalization is required in this procedure. Therefore, the test for renormalization is performed before the test for conditional exchange is carried out. As in the CODELPS procedure, the probability estimate is updated before the normalization occurs. The next MPS index (NMPS) column in Table 3.18 shows the updated probability estimate index.

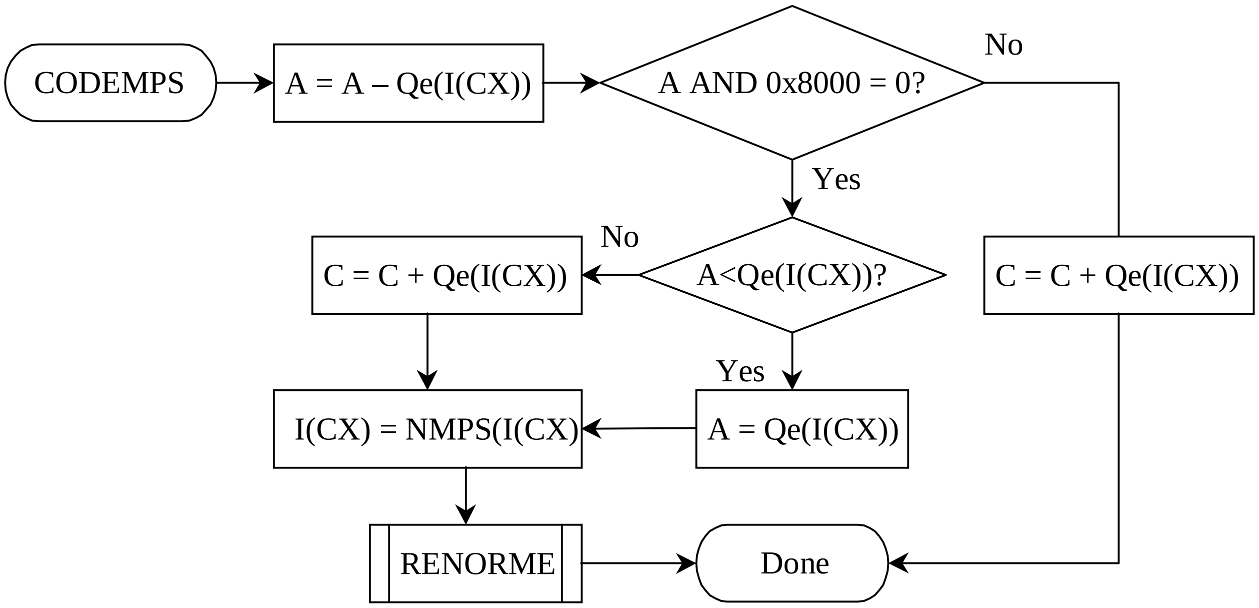


Figure 3.34   CODEMPS procedure

3.8.3.4 Probability Estimation

Table 3.18 shows the Qe values in both hexadecimal integers and decimal fractions. To convert the hexadecimal integer to decimal fraction, the Qe values are multiplied by (¾) * 0x8000.

Renormalization-driven estimation is applied to design Table 3.18. A form of approximate counting before each renormalization is the basic principle of this process. For example, in Figure 3.35, four MPSs are coded and counted before an MPS renormalization is required. The counted symbols are used to update the probability estimate. This estimate

Index	Qe Value		NMPS	NLPS	SWITCH
	hexadecimal	Decimal			
0	0x5601	0.503 937	1	1	1
1	0x3401	0.304 715	2	6	0
2	0x1801	0.140 650	3	9	0
3	0x0AC1	0.063 012	4	12	0
4	0x0521	0.030 053	5	29	0
5	0x0221	0.012 474	38	33	0
6	0x5601	0.503 937	7	6	1
7	0x5401	0.492 218	8	14	0
8	0x4801	0.421 904	9	14	0
9	0x3801	0.328 153	10	14	0
10	0x3001	0.281 277	11	17	0
11	0x2401	0.210 964	12	18	0
12	0x1C01	0.164 088	13	20	0
13	0x1601	0.128 931	29	21	0
14	0x5601	0.503 937	15	14	1
15	0x5401	0.492 218	16	14	0
16	0x5101	0.474 640	17	15	0
17	0x4801	0.421 904	18	16	0
18	0x3801	0.328 153	19	17	0
19	0x3401	0.304 715	20	18	0
20	0x3001	0.281 277	21	19	0
21	0x2801	0.234 401	22	19	0
22	0x2401	0.210 964	23	20	0
23	0x2201	0.199 245	24	21	0
24	0x1C01	0.164 088	25	22	0
25	0x1801	0.140 650	26	23	0
26	0x1601	0.128 931	27	24	0
27	0x1401	0.117 212	28	25	0
28	0x1201	0.105 493	29	26	0
29	0x1101	0.099 634	30	27	0
30	0x0AC1	0.063 012	31	28	0
31	0x09C1	0.057 153	32	29	0
32	0x08A1	0.050 561	33	30	0
33	0x0521	0.030 053	34	31	0
34	0x0441	0.024 926	35	32	0
35	0x02Al	0.015 404	36	33	0
36	0x0221	0.012 474	37	34	0
37	0x0141	0.007 347	38	35	0
38	0x0111	0.006 249	39	36	0
39	0x0085	0.003 044	40	37	0
40	0x0049	0.001 671	41	38	0
41	0x0025	0.000 847	42	39	0
42	0x0015	0.000 481	43	40	0
43	0x0009	0.000 206	44	41	0
44	0x0005	0.000 114	45	42	0
45	0x0001	0.000 023	45	43	0
46	0x5601	0.503 937	46	46	0

Table 3.18 Qe values and probability estimation process

provides a bigger Qe value when an LPS renormalization occurs and a smaller Qe value when an MPS renormalization occurs. The estimation state machine tends to approach the correct probabilities [10]. If the Qe value is too large, an MPS renormalization is more probable than an LPS renormalization. Then, a smaller Qe value is provided after an MPS renormalization. On the

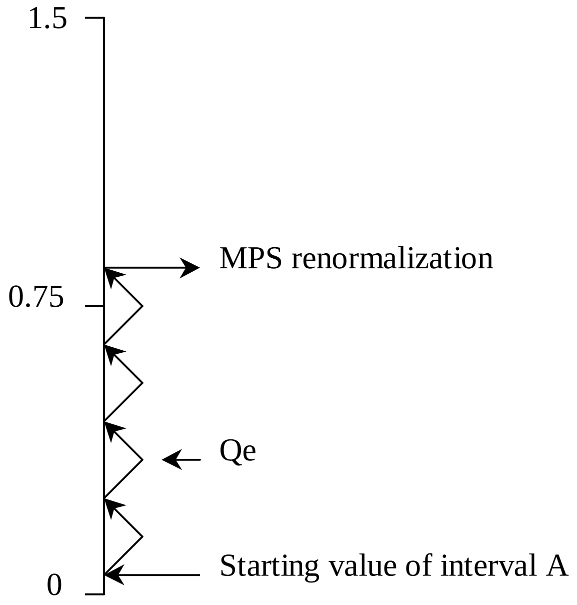


Figure 3.35 Subdivision of the interval A for four MPS

other hand, if the Qe value is too small, an LPS renormalization is more probable than an MPS renormalization. Therefore, a larger Qe value is provided after an LPS renormalization.

3.8.3.5 Renormalization in the Encoder (RENORME)

Both renormalization procedures in the encoder and decoder are very similar. The main difference between them is that in the encoder, it outputs the compressed data and in the decoder, it consumes compressed data. In the RENORME procedure (see Figure 3.36), A and C registers are shifted to the left one bit at a time. The counter CT counts the number of shifts that are

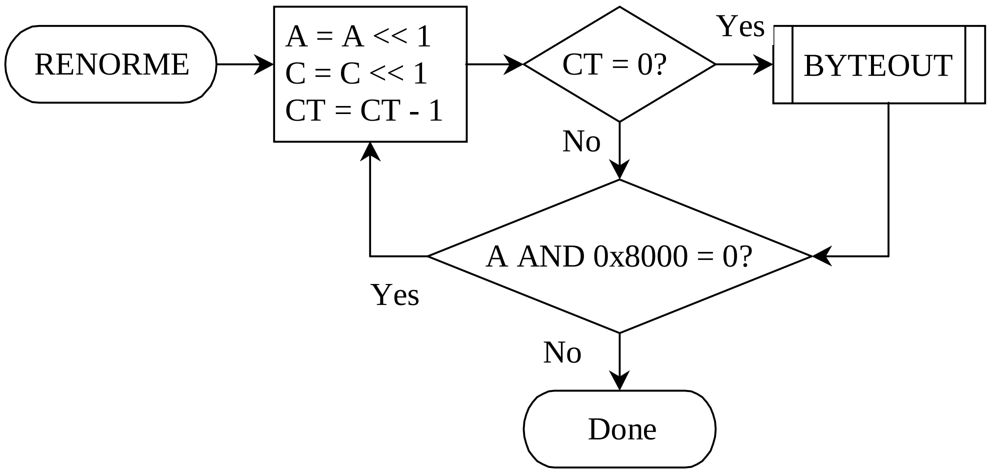


Figure 3.36 RENORME procedure

performed. If CT is counted down to 0, a byte of high order bits is removed from C to buffer by the BYTEOUT procedure. A renormalization is required until A is not less than 0x8000.

3.8.3.6 Compressed Data Output (BYTEOUT)

The BYTEOUT procedure is shown in Figure 3.37. This procedure uses the

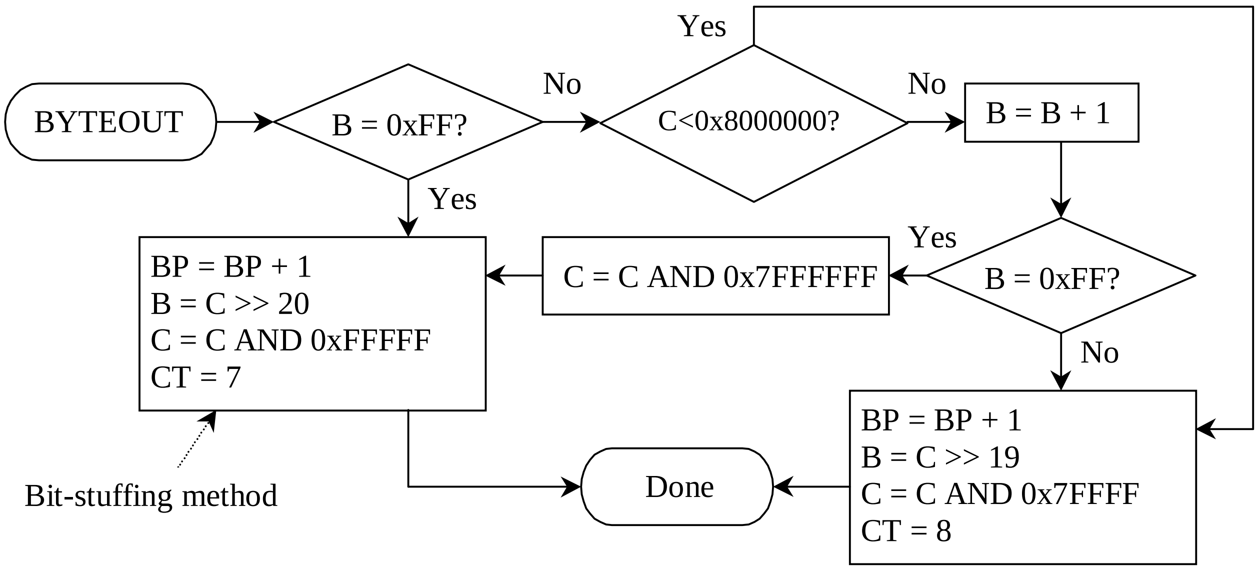


Figure 3.37 BYTEOUT procedure

"bit-stuffing" method to avoid any carry propagation into the completed bytes of compressed data. The conventions used make sure that a carry can only propagate through the byte most recently written to the compressed data buffer.

Many markers are used to signal the characteristics of the final code stream and 0xFF is the first byte of every marker. Therefore, a bit-stuffing method is applied when the most recently written byte is FF (B = 0xFF where B is the byte pointed to by the compressed data buffer pointer BP) to avoid accidental creation of a marker. The bits of "cbbb bbbb" in C are written to the buffer when this bit stuffing method is applied.

If the carry bit "c" is not set (C < 0x8000000), the bits of "bbbb bbbb" are written to the buffer. If "c" is set, a 1 is added to the most recently written byte for carry propagation. If the sum is 0xFF, the bit-stuffing method is also applied and only the bits "0bbb bbbb" are written to the buffer. Otherwise, the bits "bbbb bbbb" are written to the buffer.

3.8.3.7 Initialization of the Encoder (INITENC)

The INITENC procedure sets the values of register A, register C, pointer BP, and counter CT before encoding the code stream (see Figure 3.38). The counter CT is initialized to

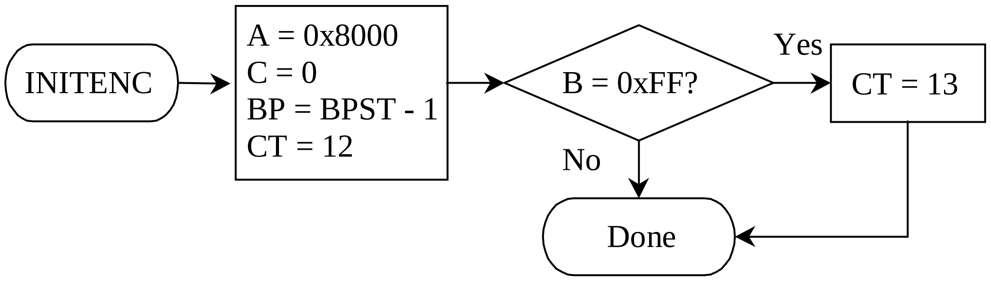


Figure 3.38 INITENC procedure

12 since the spacer bits, “sss”, should also be filled before a byte is removed from C. Pointer BP is set to point to the byte preceding the position BPST where the first byte is written. Therefore, if the preceding byte B is 0xFF, a spurious bit is stuffed and a corresponding increment of CT is needed. Initializations of the MPS and I are mentioned in Section 3.7.4.

3.8.3.8 Termination of Encoding (FLUSH)

The FLUSH procedure is the last step of the arithmetic encoding process and creates the terminating marker at the end of compressed data. The terminating marker prefix, 0xFF, is guaranteed to overlap the final bits of compressed data. Therefore, 0xFF must be read by the arithmetic decoder and interpreted before the decoding is complete. The first step in the FLUSH procedure (see Figure 3.39) is SETBITS. It sets as many bits in C register to 1 as possible, as

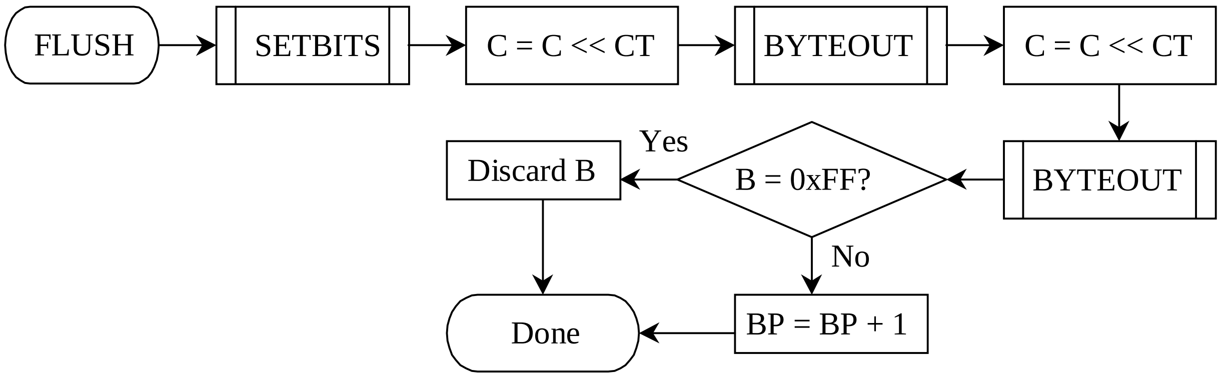


Figure 3.39 FLUSH procedure

shown in Figure 3.40. The exclusive upper bound for the C register is the sum of C register and A register. The 16 least significant bits of C register are set to 1 and the result is compared to the upper bound. If the resulted C register is too large, it is reduced to a value, which is smaller than the upper bound. Then, two bytes of C register are shifted and removed to the buffer. If the last byte is 0xFF, it is discarded.

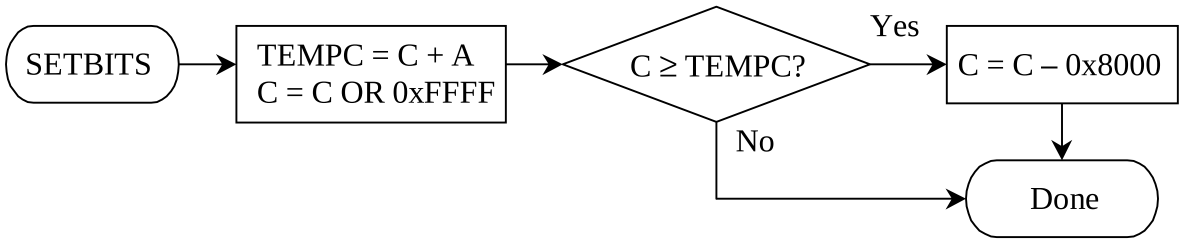


Figure 3.40 SETBITS procedure

3.8.4 Arithmetic Decoder

The inputs of the arithmetic decoder are the compressed data, CD, and the context labels, CX, while the output is the decision, D. The context labels for the encoder and decoder are the same for each given decision.

The structures of the C register and A register for the decoder are shown in Table 3.19.

	MSB	LSB
C high register	xxxx xxxx	xxxx xxxx
C low register	bbbb bbbb	0000 0000
A register	aaaa aaaa	aaaa aaaa

Table 3.19 Structures of decoder registers

The C register is divided into the C low and C high registers. Each renormalization in the C register shifts a bit from the MSB of the C low register to the LSB of the C high register. Only the C high register is used during decoding comparisons. A new byte of compressed data is transferred to the C low register when it is empty. The A register has the same structure as in the encoder conventions.

The first step in the DECODER (see Figure 3.41) is the INITDEC procedure, which initializes the decoding process. Contexts, CX, are read as the input for the DECODE procedure until all of them have been read. The reading of compressed data, CD, is embedded in the

DECODE procedure. The DECODE procedure decodes the binary decision, D, which is either 0 or 1.

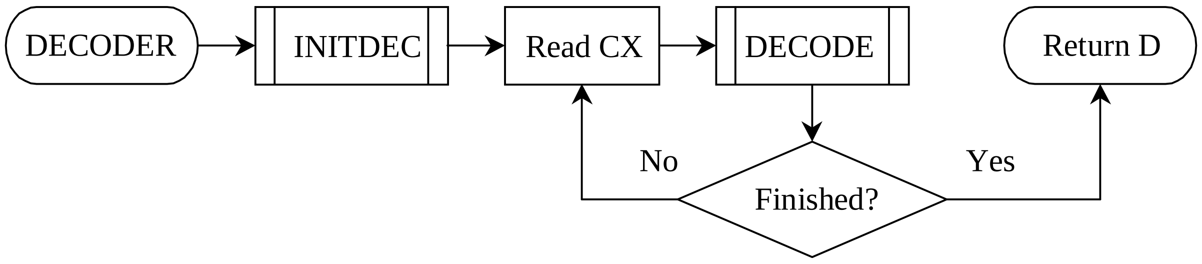


Figure 3.41 DECODER structure

3.8.4.1 Decoding a decision (DECODE)

The DECODE procedure decodes one decision at a time. After decoding a decision, it subtracts any amount from CD that is added to the C register during encoding. The resulted CD is the offset from the base of the current interval to the subinterval allocated to the decisions not yet decoded. The C high register is compared to the length of the LPS subinterval, Qe(I(CX)), for the first part of the procedure (see Figure 3.42). If C high is smaller, an LPS should be

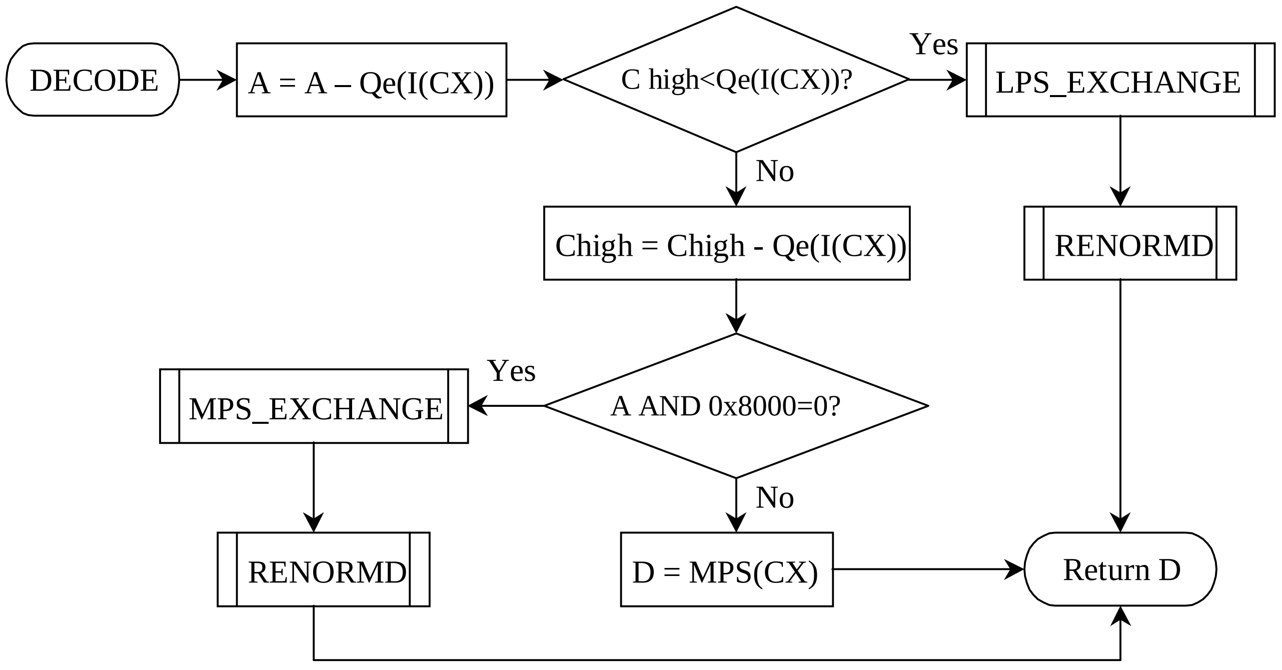


Figure 3.42 DECODE procedure

decoded at most of the times and the LPS_EXCHANGE procedure is used. A renormalization must occur after the LPS_EXCHANGE procedure. If C high is larger, an MPS is usually decoded and C high is reduced by Qe(I(CX)). If A is larger than 0x8000, the MPS(CX) is decoded. On the other hand, if A is smaller than 0x8000, the procedure MPS_EXCHANGE is applied and a renormalization is required.

3.8.4.2 Checking for Conditional Exchange (MPS_EXCHANGE and LPS_EXCHANGE)

A conditional exchange may occur when a renormalization is required. The MPS_EXCHANGE procedure checks whether a conditional exchange is needed, as shown in Figure 3.43. If A (the MPS subinterval) is larger than Qe(I(CX)), which is the subinterval for the

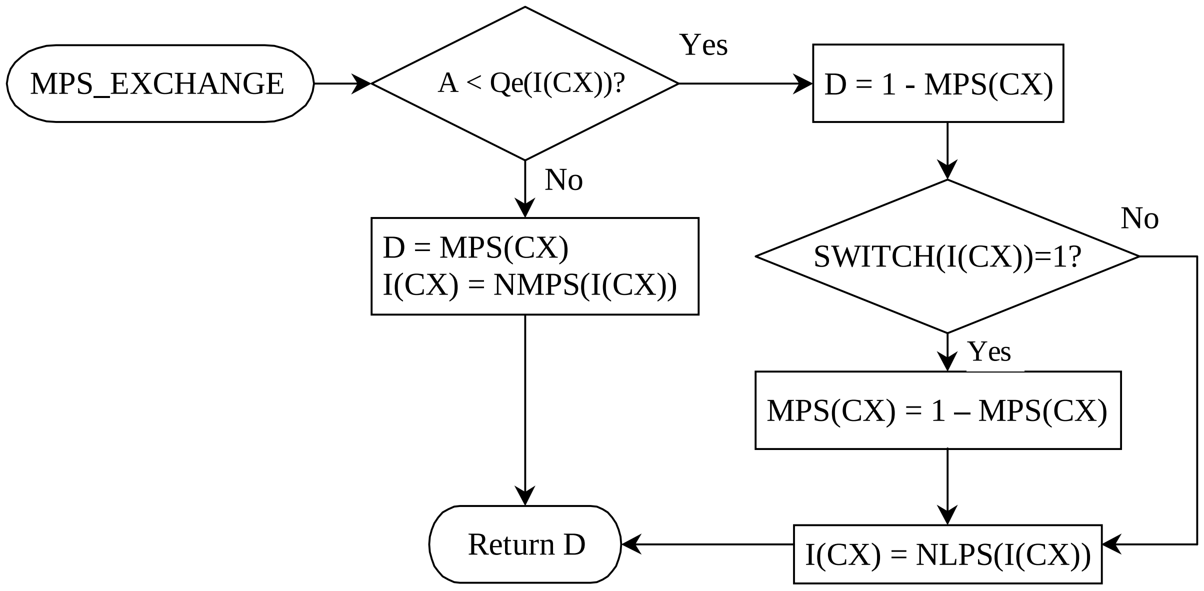


Figure 3.43 MPS_EXCHANGE procedure

LPS, the decoded decision is MPS(CX) as usual and the index for probability estimate of CX is updated from the next MPS index (NMPS) column in Table 3.18. Otherwise, the conditional exchange occurs and the decoded decision is LPS(CX) or 1 – MPS(CX). The MPS sense is switched when the SWITCH column has a 1 for the index I stored for CX in Table 3.18. The index for probability estimate of CX is updated from the next LPS index (NLPS) column in Table 3.18. The probability estimates for the decoder is identical to that of the encoder.

The LPS_EXCHANGE procedure checks whether a conditional exchange is needed, as shown in Figure 3.44. If A (the MPS subinterval) is smaller than Qe(I(CX)), a conditional exchange occurs. Then, A is set to the LPS subinterval, Qe(I(CX)). The decoded decision is MPS(CX) and the index for probability estimate of CX is updated from the next MPS index (NMPS) column in Table 3.18. Otherwise, the decoded decision is the LPS(CX) or 1 – MPS(CX). However, A is still set to Qe(I(CX)). The MPS sense is switched when the SWITCH column has a 1 for the index I stored for CX in Table 3.18. The index for probability estimate of CX is updated from the next LPS index (NLPS) column in Table 3.18.

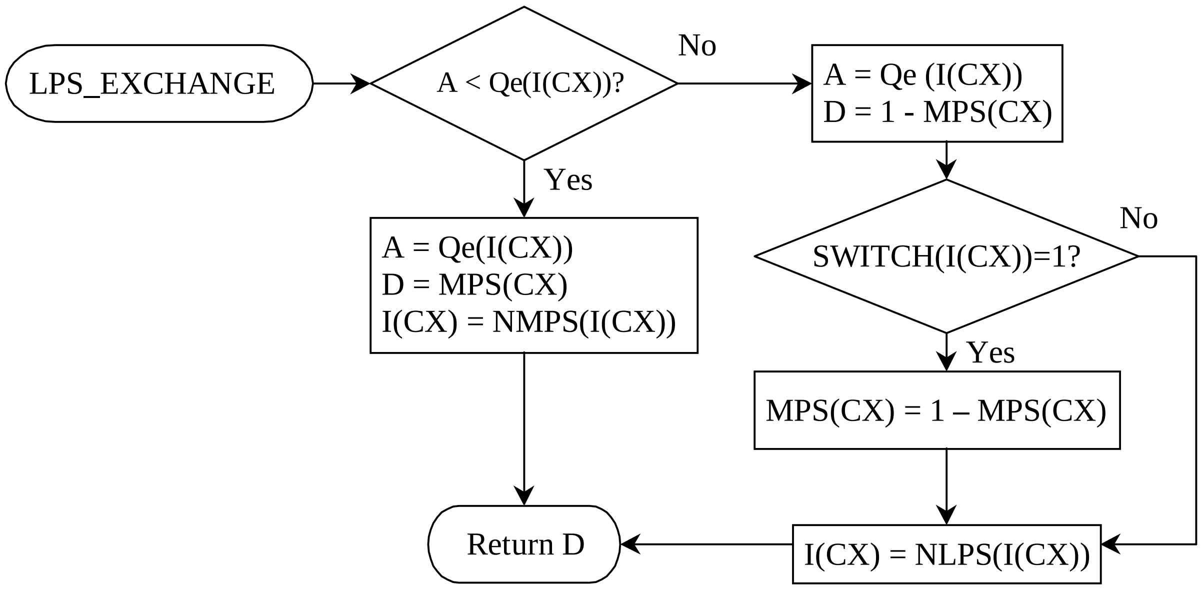


Figure 3.44 LPS_EXCHANGE procedure

3.8.4.3 Renormalization in the Decoder (RENORMD)

The RENORMD procedure (see Figure 3.45) uses the counter CT to determine whether the C low register is empty. If it is empty, the BYTEIN procedure transfers a byte of compressed data to the C low register. The A, C high and C low registers are shifted one bit at a time until A is larger than 0x8000.

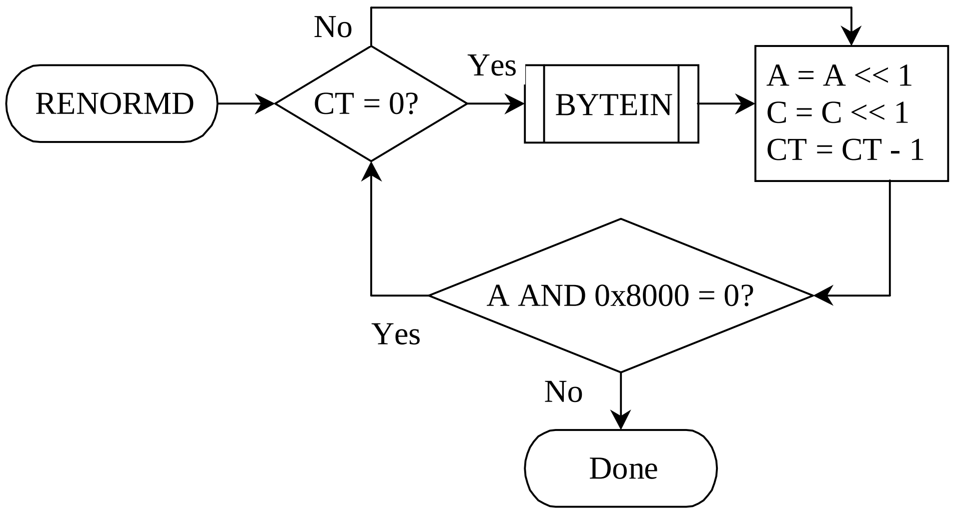


Figure 3.45 RENORMD procedure

3.8.4.4 Compressed Data Input (BYTEIN)

The BYTEIN procedure reads a byte of compressed data, as shown in Figure 3.46. Any compensation for the bit-stuffing method is performed. The terminating marker is also recognized in this procedure wherever the encoder is terminated. The C register in Figure 3.46 is the concatenation of the C high and C low registers. B is the byte pointed by the compressed data buffer pointer BP. B usually is not 0xFF. In that case, BP is incremented by one and a new byte of compressed data is delivered to the high order 8 bits of the C low register. If B is equal to 0xFF, it determines whether it is a marker prefix. If B1 (the byte pointed to by BP + 1) is larger than 8F, it must be the terminating marker code.

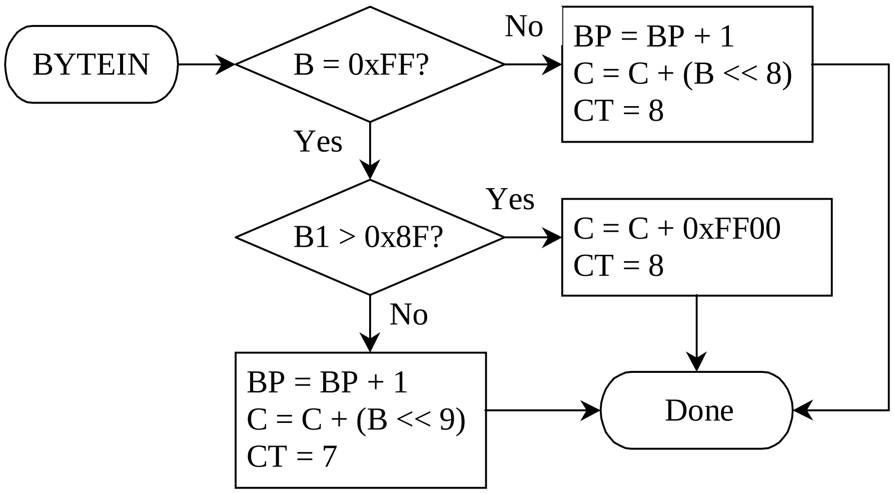


Figure 3.46 BYTEIN procedure

Then, it is appropriately interpreted and terminates the compressed data. 0xFF is added to the C register and the counter CT is set to 8 as the last step of the decoding process. If B1 is not a marker code, BP is incremented to point to the next byte, which contains the stuffed bit. B is shifted one more bit and is added to the C register. Therefore, the stuffed bit is added to the low order bit of the C high register.

3.8.4.5 Initialization of the Decoder (INITDEC)

The INITDEC procedure starts the whole arithmetic decoding process.

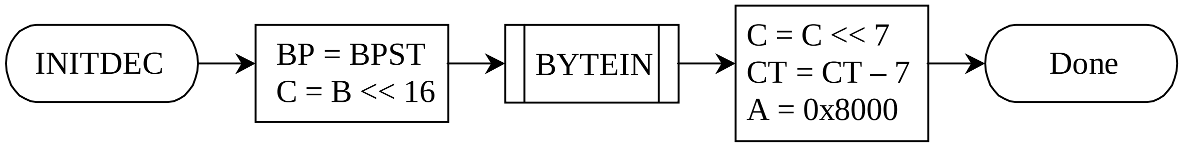


Figure 3.47 INITDEC procedure

BP is set to BPST to point to the first compressed byte. This byte is inserted to the low order byte of the C high register. Another byte is read into the high order byte of the C low register by the

BYTEIN procedure. The C register is then shifted by 7 bits and the counter CT is reduced by 7. The A register is initialized to 0x8000 as in the encoder. (see Figure 3.47)

3.9 Coding Images with Region of Interest

A Region Of Interest (ROI) is a part of an image that is coded earlier than the rest of the image (background) with better quality. The method that is used is Maxshift method.

3.9.1 Maxshift Method

3.9.1.1 Encoding

The encoding process with ROI is almost the same as that without ROI. An ROI mask is formed to indicate which quantized coefficients are within the ROI during the encoding process so that they can be encoded with better quality. The ROI mask is a binary bit map describing these coefficients. (see Section 3.9.2) The coefficients of the background (outside of the ROI mask) are scaled down so that the bits corresponding to the ROI coefficients are placed in higher bit-planes than all the bits for the background coefficients. When the entropy coder encodes the coefficients, the bit-planes for the ROI coefficients are coded before the lower bit-planes for the background coefficients are coded. The scaling value, s, must be chosen so that even the smallest non-zero ROI coefficient is larger than the largest background coefficient. The basic steps of the Maxshift method for the encoding part are:

(1) Create the ROI mask, M(x,y);

(2) Determine the scaling value, s;

(3) Scale down the background coefficients by 2^S;

(4) Store the value s into the code stream.

The number of bit-planes to be coded is increased by s. All the coefficients are entropy coded as usual after these basic steps.

The sufficiently large scaling value, s, is chosen according to the following equation:

$$s \geq max\ (M_b)$$

where max(M_b) is the maximum M_b over all sub-bands for all the background coefficients in any code block in the current component and M_b is the maximum number of encoded bit-planes for the sub-band b.

3.9.1.2 Decoding

After entropy decoding, the coefficients are compared to the threshold value 2^S. If the coefficient is smaller than 2^S, then it is a background coefficient and it is scaled up by 2^S. Otherwise, it belongs to the ROI. The following steps summarize the Maxshift method for the decoding part.

(1) Read the scaling value s;

(2) Compare all coefficients to 2^S and scale up the coefficients that are below 2^S.

3.9.2 Creation of ROI Mask

For simplicity, let us consider a single component image. The ROI mask, M(x,y), is a bit-plane that indicates the coefficients that are needed for the ROI with a 1 and the coefficients that are not needed for the ROI with a 0 as in the following equation:

$$M\ (x, y) = \begin{cases} 1 & \text{Wavelet coefficient (x,y) is needed for ROI} \\ 0 & \text{Wavelet coefficient (x,y) is not needed for ROI.} \end{cases}$$

After each level of the wavelet decomposition, the LL sub-band of the mask is updated row by row and then column by column. The mask indicates which coefficients are required in order to reproduce the coefficients of the previous level mask through the inverse wavelet transformation. Similar process is done for the LH, HL, and HH sub-bands.

3.10 Rate Distortion Optimization

Given a target bit rate such as the number of bits per pixel, distortion must be minimized to give the best quality for the reconstructed image.

3.10.1 Distortion

Distortion is the difference between the original image and the reconstructed image. Many methods can be used to measure distortion. The mean squared error is the most popular one. For a code block B_i, the mean squared error, D_i, is calculated by the following equation:

$$D_i = \sum_{k \in B_i} (s'_i[k] - s_i[k])^2 \quad ,$$

where k is the index for the pixels in the code block B_i. $s_i[k]$ and $s'_i[k]$ are the original pixel value and the reconstructed pixel value at index k in the code block B_i.

3.10.2 Rate Distortion Optimization for Code Block

Let R_{max} be the constrained number of bits specified to store an image. Every image is divided into code blocks. Let $\{B_i\}_{i = 1, 2, \ldots}$ denote the set of code blocks that represents the whole image and each code block B_i has a length of R_i. Therefore, R_{max} has the following relationship with R_i:

$$R_{\max} \geq \sum_i R_i \quad .$$

Let D_i be the distortion incurred from R_i and assume that D_i can be additive [21]. We want to minimize the total distortion D, which is the sum of D_i for each code block B_i:

$$D = \sum_i D_i \quad .$$

For a specific code block B_i, the code stream for it can be truncated to a set of discrete lengths R_i^1, R_i^2, R_i^3,…. These R_i^n are obtained from inclusions of different number of the bit-plane coding passes (Section 3.7.3). Let the distortion resulted for these truncated lengths be D_i^1, D_i^2, D_i^3,…. R_i^n and D_i^n are calculated and temporarily stored with the compressed bit stream during the encoding process.

The final bit stream consists of a number of "layers." Each layer contains the additional contributions (maybe empty for some code blocks) from each code block, as shown in Figure 3.48, so one more layer is a step of improvement in terms of overall image quality. Only 8 code blocks and 3 layers are shown in Figure 3.48 for simplicity. The truncation points of the code block bit streams for each layer are optimal in the rate-distortion sense. It means that the final bit stream is rate-distortion optimal by discarding a whole number of least important layers. If a layer is partially discarded, optimum rate-distortion result is not guaranteed. However, a small departure from the rate-distortion optimal case can be obtained if the number of layers is large.

After the whole image is compressed, a post-processing operation examines each compressed code block and decides which truncation point, R_i^n, should be chosen for each code

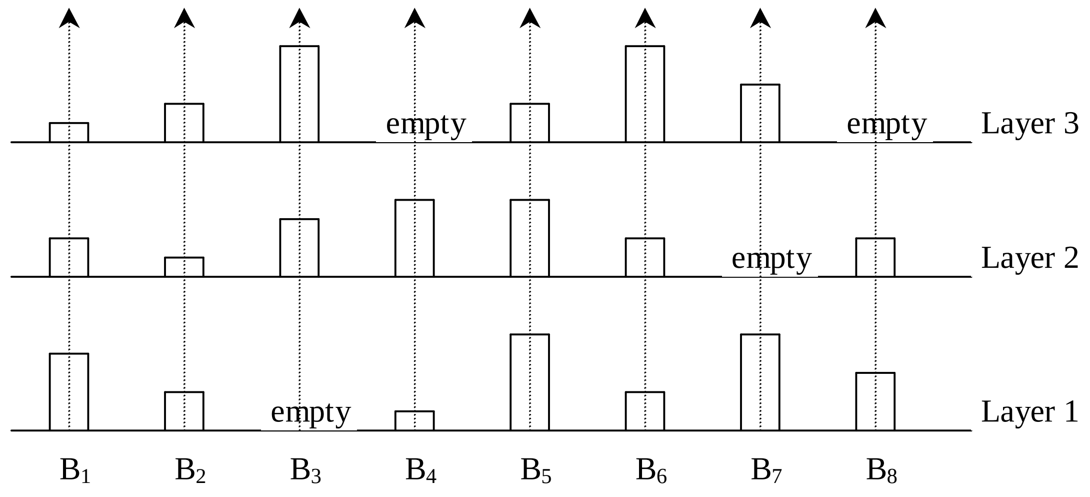


Figure 3.48 Code block bit streams in quality layers

block B_i in order to satisfy the length constraint with the minimum distortion. The problem to be solved is to minimize

$$\sum_i (R_i^n + \lambda D_i^n)$$

where λ can be interpreted as a quality parameter as explained later.

In turn, we can minimize

$$R_i^n + \lambda D_i^n$$

for each individual code block. A simple algorithm to find the optimal n for a given λ is provided as follow:

Set n = 0 (i.e. no information is included)

For k = 1, 2, 3, …

Set $\Delta R_i^k = R_i^k - R_i^n$ and $\Delta D_i^k = D_i^n - D_i^k$

If $\Delta D_i^k / \Delta R_i^k > \lambda^{-1}$, then set n = k, where n is set to the largest k that satisfies $\Delta D_i^k / \Delta R_i^k > \lambda^{-1}$. λ must be adjusted until the sum of R_i is equal to R_{max} (or very close but smaller than R_{max}). λ can be interpreted as a quality parameter, since a larger value of λ means a less severe truncation of the bit stream. λ^{-1} can be identified as a rate-distortion slope threshold. Therefore, different values of λ can be set for different layers.

3.11 Decoding

The decoding procedures perform only the inverse functions of the encoder. They consist of the arithmetic decoding, inverse of coefficient bit modeling, dequantization, inverse wavelet transformation, inverse DC level shifting, and inverse component transformation.

Chapter 4

Experimental Results

4.1 JPEG Compression

For the JPEG compression part, the IrfanView version 3.33 program, which is a popular image viewer and converter for many graphic file formats, is used to convert the original image files into the compressed JPEG files and convert the compressed JPEG files back into the reconstructed image files. The IrfanView program can be downloaded from http://www.ryansimmons.com/users/irfanview/english.htm. The approximate values of bits per pixel (bpp) for the compressed JPEG files can be controlled by a "slider control." The quality level in the "slider control" ranges from 1 (worst quality) to 100 (best quality).

4.2 JPEG-2000 Compression

For the JPEG-2000 compression part, the JJ2000 version 3.2.2 program is chosen to encode the original image files into the compressed J2K files and decode the compressed J2K files back into the reconstructed image files. The JJ2000 program is one of the two reference softwares in the JPEG-2000 standard. It is implemented by the Java programming language. It can be downloaded from [22] solely for research purposes. The values of bpp for the compressed J2K files can be specified in the command line. The detailed information of the JJ2000 program can be obtained from [22].

The default settings of the JJ2000 program are adopted for the experiment except the bitrates in bpp. Some of the important default features are: one tile for the whole image (no tiling), five levels of the wavelet decomposition, adoption of the 9-tap/7-tap wavelet transformation for better performance for visually lossless compression and the irreversible component transformation. A complete listing of the default settings can be found from the downloaded files.

4.3 Compressing and Decompressing of Test Images

Three "standard" single-component test images (Lena, Baboon and Peppers) and a color Lena image are chosen for the experiments. They are stored either in the Portable GrayMap file format (PGM) for the single-component images or in the Portable PixMap file format (PPM) for the color image. All test images have dimensions of 512 x 512. The PGM and PPM file formats are two of the three file formats (the third one is PGX for the grayscale image with arbitrary bit-depth) that are accepted by the JJ2000 program. For the experiments, they are compressed in both the JPEG and JPEG-2000 formats. The specified numbers of (bpp) are 0.0625, 0.1, 0.125, 0.25, 0.5, 1.0, and 2.0 for both standards and the corresponding file sizes in bytes are 2048, 3277, 4096, 8192, 16384, 32768, and 65536 but the exact values of bytes used are shown in Tables 4.1 and 4.2. As one can see from Tables 4.1 and 4.2, the actual file sizes for the four JPEG images for the bitrate of 0.0625 are not shown. It is because they are much larger than the specified 2048 bytes even the worst quality level of 1 in the slider control is applied when the images are saved in the JPEG file format. This is a common behaviour for many other image converter programs since any normal photographic images are totally distorted for the bitrate of 0.0625 in the JPEG file format. Therefore, the bitrate of 0.0625 is not for practical application. On the other hand,

Specified Bit Rate (bpp)	Specified File Size (bytes)	Actual File Size					
		Lena		Baboon		Peppers	
		JPEG	JPEG 2000	JPEG	JPEG 2000	JPEG	JPEG 2000
0.0625	2048	--	2036	--	2018	--	2036
0.1	3277	3521	3259	3280	3204	3313	3194
0.125	4096	4029	4071	3868	4057	3891	4086
0.25	8192	8059	8172	8178	8109	8272	8178
0.5	16384	16297	16229	15843	16214	16333	16357
1.0	32768	32144	32523	32792	32747	32561	32574
2.0	65536	67073	65428	64866	65496	65695	65133

Table 4.1 The actual file sizes of the single-component compressed images

Specified Bit Rate (bpp)	Specified File Size (bytes)	Actual File Size	
		Lena	
		JPEG	JPEG-2000
0.0625	2048	--	2050
0.1	3277	3257	3265
0.125	4096	3796	4098
0.25	8192	8121	8045
0.5	16384	16373	16350
1.0	32768	33137	32759
2.0	65536	67934	65252

Table 4.2 The actual file sizes of the color compressed images

this problem does not appear in JPEG-2000.

For the decoding part, all the compressed files are converted back into the PGM and PPM files using the InfanView and JJ2000 programs. All the original and reconstructed images with bitrates of 0.1 and 0.125 are shown in Figures 4.1, 4.2, 4.3, and 4.4. These two bitrates are chosen since they provide largest difference in quality between the JPEG and JPEG-2000 standards.