IMPLEMENTATION OF POST-COMPRESSION RATE DISTORTION OPTIMIZATION WITHIN EBCOT IN JPEG2000

A Thesis

Submitted to

the Engineering School of the

UNIVERSITY OF DAYTON

In Partial Fulfillment of the Requirements for

The Degree

Master of Science in Electrical Engineering

by

Benjamin T. Fortener, B.S.

UNIVERSITY OF DAYTON

Dayton, Ohio

December 2009

IMPLEMENTATION OF POST-COMPRESSION RATE DISTORTION OPTIMIZATION WITHIN EBCOT IN JPEG2000

APPROVED BY:

Eric J. Balster, Ph.D.
Advisor Committee Chairman
Electrical & Computer Engineering

Frank Scarpino, Ph.D.
Committee Member
Electrical & Computer Engineering

Russell Hardie, Ph.D.
Committee Member
Electrical & Computer Engineering

Malcolm Daniels, Ph.D.
Associate Dean, School of Engineering

Tony Saliba, Ph.D.
Dean, School of Engineering

ABSTRACT

IMPLEMENTATION OF POST-COMPRESSION RATE DISTORTION OPTIMIZATION WITHIN EBCOT IN JPEG2000

Name: Benjamin T. Fortener
University of Dayton, 2009

Advisor: Eric J. Balster

JPEG2000 is the latest standard in image compression with the Joint Photographic Experts Group (JPEG) committee publishing Part 1, second edition of the standard in September 2004. The JPEG2000 compression system improves upon the original JPEG compression standard mostly through the use of a wavelet transform and an Embedded Block Coding with Optimal Truncation (EBCOT) technique that allows for a compression advantage over JPEG of about 30 percent on average. EBCOT depends on an arithmetic entropy coder called the MQ Coder that processes data and outputs the final image bit-stream that is stored in a JPEG2000 file. After the MQ Coder, the bit-stream can be truncated at points defined during the encoding process, and this truncation may be done optimally to minimize the amount of distortion introduced to the decoded image. The method of removing data from the code stream is called the Post-Compression Rate Distortion Optimization (PCRD-opt) algorithm. This thesis details the PCRD-opt algorithm and presents a working implementation that out-performs that of the mainstream reference encoders JasPer and Kakadu.

To my parents, without whose support I would never have gotten to where I am.

ACKNOWLEDGMENTS

- Thank you to Eric Balster for being my advisor and putting up with me and nudging me in the right direction when I would procrastinate too much. That ECE 561 class you taught kind of helped with the wavelet stuff, too.
- Thanks to Dr. Scarpino, for hiring me on to his team and offering me the opportunity to work on what eventually became this thesis. Thank you for being a mentor and someone to offer advice, wisdom, and of course the occasional story that always included the words "hand-waving," "Avogadro's," or both...multiple times.
- Thanks to Kerry Hill and Al Scarpelli for continuing to work so hard to keep our student program going to continue to fund us. We appreciate it very much.
- Thank you to David Lucking, for not taking this topic before I did and always knowing the parts of JPEG2000 I didn't.
- Thanks to Luke Hogrebe for all the Tier 1 knowledge and going first through all of this so I would have an easier time of it.
- Thank you to Frank Fradette for being my entertainment throughout my time here. I mean that in a good way, of course.
- And thank you to David Walker, Thad Marrara, and Nick Vicen for listening to my random, white board scribblings and helping me think through all of this.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

JPEG2000 is the latest standard in image compression with the Joint Photographic Experts Group (JPEG) committee publishing Part 1, second edition of the standard in September 2004. The JPEG2000 compression system improves upon the original JPEG compression standard through the use of a wavelet transform and an Embedded Block Coding with Optimal Truncation (EBCOT) technique that allows for a compression advantage over JPEG of about 10% to 25% for images compressed to about 0.5 to 1 bits per pixel [4]. This improvement is much higher at lower bit rates. JPEG2000's EBCOT algorithm depends upon its Tier 1 portion that contains an arithmetic coder called the MQ Coder. Tier 1 consists of three coding passes that act on bit planes of blocks of wavelet coefficients called code blocks, which are usually 32x32 or 64x64 coefficients in size and are coded independently by Tier 1. The resulting bit stream from the MQ Coder is what makes up the compressed file stream in JPEG2000, and this file stream can be truncated at specified positions within it in order to meet a rate constraint for a compressed image. In order to achieve an optimal code stream truncation, a distortion is calculated for every truncation point. The rates per truncation point and calculated distortions are used in an optimization problem that can be solved using Lagrange multipliers.

Optimal code stream truncation is referred to as Post-Compression Rate Distortion Optimization (PCRD-opt) and is one of the most significant portions of the JPEG2000 encoding scheme, providing a fine-grain inspection of small portions of the stream and how much they contribute to distortion after decoding. Most of the literature regarding PCRD-opt or EBCOT focus on speed or quality improvements and assume a working implementation of the algorithm. This is a problem for custom JPEG2000 encoders that have not implemented an optimal truncation algorithm; the literature defining implementation is sparse. Because the JPEG2000 specification defines only the format of the compressed code stream, implementations of the EBCOT algorithm vary widely. Most implementations in literature use the well-known constrained optimization by method of Lagrange multipliers defined by the standard [cite]. Portions within this algorithm have been modified to increase compression speed or performance at some cost. A few of these methods of modification are explored in researching the PCRD-opt algorithm for this thesis.

In order to forgo calculation of distortions during Tier 1 encoding, Zhu *et al.* [18] assumes minimal distortion contribution from the fractional bits of the normalized version of a coefficient sample. In their proposed algorithm, these bits are disregarded and the quantization median of 0.5 is used in place of the fractional bits. They point out that the majority of the distortion contribution is due to the wavelet energy gain, the bitplane's level, and the number of pixels involved in the distortion calculation. Removing the fractional bits results in a slight decrease of PSNR of the decoded images, but a significant speed increase in the PCRD algorithm as well as an advantage for a hardware implementation.

Yeung *et al.* [17] proposed a "Priority Scanning" method that estimates the order of truncation points by coding passes within a bitplane and by bitplane level. The codeblocks' coding passes are then coded in this order until a rate constraint is met. Using this priority scanning method eliminates the Tier 1 coding portion for data that would be removed afterward by the optimal truncation process. Their algorithm produces the same results as the standard JPEG2000 EBCOT algorithm while reducing the Tier 1 encoding time at a cost of increased complexity.

While many papers have presented alternatives to the PCRD-opt algorithm, the implementation proposed in this paper uses the algorithm defined in [9] in order to get an integrated solution with as little modification as possible to existing code and to adhere to suggestions from the standard. This serves as a proof-of-concept implementation that will allow for further efforts in integrating a PCRD-opt algorithm in a hardware Tier 1 solution in an existing JPEG2000 encoder, henceforth referred to as the UDJPEG2000 Encoder [5],[7],[10]. This thesis presents a detailed look at the EBCOT algorithm with a focus on the PCRD-opt portion and explains the algorithm in detail so as to facilitate its implementation in existing platforms. This paper is organized as follows: following the Introduction, Chapter 2 gives a background on digital imagery in general, and Chapter 3 explains the JPEG2000 encoding process and the benefits of its file format. Then, the significant portions of the JPEG2000 algorithm that are needed to understand what is done in the PCRD algorithm are further detailed in Chapters 4 through 6. The post-compression rate-distortion algorithm used for optimal truncation is explained in Chapter 7 before presenting the results from integrating the algorithm into the UDJPEG2000 Encoder in Chapter

8. The results demonstrate equivalent peak signal-to-noise ratio (PSNR) for a variety of images to the JasPer and Kakadu encoders—reference implementations of the JPEG2000 standard. Finally, conclusions and future work are proposed in Chapter 9.

CHAPTER 2

Digital Imagery

Digital imagery has been a prominent format for imagery for a number of decades. As with other media, imagery and video have been increasingly captured in a digital format. While film may still have the edge for resolution [6], digital cameras are fast becoming industry standard. With all the data that is being captured and stored in digital memory, a big concern becomes storage space. Raw image data from a camera is usually stored as a matrix of pixels for each color component, each pixel having a value representing the intensity of a single sample within that component. For color images, the standard three color planes, red, green, and blue, are all stored. For a grayscale image, there is only one component, the luminance component. A fast-growing field of infrared imagery and images that are created by capturing other wavelengths can also be displayed as grayscale images. The range of values each pixel can take is determined by how many bits are used to store each sample value. With 8 bits, the range is between 0 and $2^8 - 1$, or 255. Typically, black is represented by 0, and the lightest white is represented by 255. The colors between are linear increments of brightness creating a gradient of gray, hence, grayscale. With 8 bits per pixel (one byte per pixel) and 3 color planes per color image, a 10 megapixel raw image has a storage size of 30 million bytes; 30 megabytes for a single image. The problem becomes more evident when considering storing high resolution surveillance

video at 30 frames per second for days, weeks, months, or even years at a time. A solution to this problem lies in data compression.

Traditionally, the most popular format for data compression has been the JPEG format [16], and is prevalent in digital cameras and the Internet. However, the standard JPEG format is somewhat lacking; the discrete cosine transform it performs is done on small, 8x8 blocks of pixels, resulting in severe blocking artifacts at high compression ratios. Another drawback is that lossless JPEG compression (JPEG-LS) uses a completely different algorithm than if performing lossy compression. Most importantly, JPEG's file format is not very flexible for providing some desirable features. The JPEG committee has released a new compression standard called JPEG2000 that addresses these and many other issues. The next section describes the JPEG2000 algorithm and introduces many of the benefits it provides over JPEG.

CHAPTER 3

JPEG2000 Background

JPEG2000 was ratified as an international standard in December 2000 [12] and has since had a second edition published in September 2004 [9]. The JPEG2000 standard improves upon the original JPEG standard in a multitude of ways: JPEG2000 has superior performance at low bit-rates when JPEG artifacts become noticeable. The wavelet transform used in JPEG2000 is applied to the whole image, so the compressed image becomes blurry at very low bit rates as opposed to very blocky with original JPEG. Blurriness in an image is more acceptable to the human visual system than is blockiness [3]. JPEG2000 is one of the only standard image formats that can handle continuous-tone grayscale and color imagery with greater than 16 bits per component per sample, and its compression of bi-level imagery (1-bit per sample) is comparable to other standards specifically designed to compress single bit imagery [12]. The file format for JPEG2000 is one of the greatest improvements over JPEG. It allows for progressive decoding of small parts of the image and of increasing pixel accuracy, meaning that the decoder can display a blurry-looking image almost immediately, and the image becomes clearer as more data is received and decoded. This becomes significant on slow data links and with very large imagery. The JPEG2000 file format can also contain multiple resolutions (image sizes) within one file with no redundant data. With the original JPEG, if more than one resolution of an image is desired, the

original image must be compressed multiple times to multiple files. In this case, the largest resolution image contains a large amount of redundant data already stored in the smaller resolution images. JPEG2000 compresses an image once and stores the multiple resolutions in a single file without the redundant data.

With all the obvious benefits to JPEG2000, some drawbacks exist that have slowed the adoption of the standard in the industry where original JPEG still dominates. The most significant reason is that the computational requirement for the JPEG2000 compression algorithm is more than 30 times as great compared to the current baseline JPEG encoder [1]. This suggests that encoding to a JPEG2000 file takes much longer than encoding to a JPEG file. In some real-time applications, this computation boundary must be overcome before adoption of the technology can take place. Another reason adoption has been slow is that the majority of applications do not inherently support the JPEG2000 file format. Even if the industry were to start using JPEG2000, most standard image file viewers and Internet browsers cannot currently display the JPEG2000 image format.

3.1 Lossy Versus Lossless Compression

Data can either be compressed to a form that can be completely restored to the original (lossless, or reversible, compression), or it can be compressed using a means of removing data so that it cannot be fully restored (lossy, or irreversible, compression). Lossy compression is more typical in most applications because of the much higher compression ratios it can achieve. Lossless compression typically achieves a compression ratio between 1.5:1 and 2.5:1 [2]. In image compression, lossy compression can achieve much higher compression ratios and still preserve visual quality, because

most of the energy content in typical images resides in the lower frequencies. This allows much of the higher frequencies to be compressed more aggressively while preserving the most important visual content. When performing lossless compression, data must be stored as integers; some floating point numbers cannot be stored with infinite precision with a finite number of bits, and performing division may result in irrational numbers. Therefore, if a mathematical operation is performed on data using non-integers, the inverse operation to restore the data may not be able to perfectly restore the data to its original value, thus resulting in a loss of precision.

3.2 Encoding Process

Image compression is a process of decorrelating visual data, removing some of that data in the case of lossy compression, and then coding the data. JPEG2000 follows this procedure while introducing novel methods of coding to achieve a compression efficiency very close to the entropy limit of the data [1]. An image is preprocessed by a level offset, a color transform (for multi-component images), a wavelet transform, and quantization. It is then encoded by JPEG2000's Tier 1 coder, the portion of the JPEG2000 process that achieves entropy coding. For lossy compression, optimal truncation is then performed before assembling the code stream in JPEG2000's Tier 2 portion. The output of Tier 2 is a fully-compliant, JPEG2000 image file. This process flow is shown in Figure 3.1.

3.2.1 Level Offset

When image data is captured, the pixel data is usually stored as unsigned values from 0 to $2^B - 1$, where B is the number of bits used to store each pixel value. This gives the data a mean value of $\frac{2^B-1}{2}$. However, the entropy coder works most

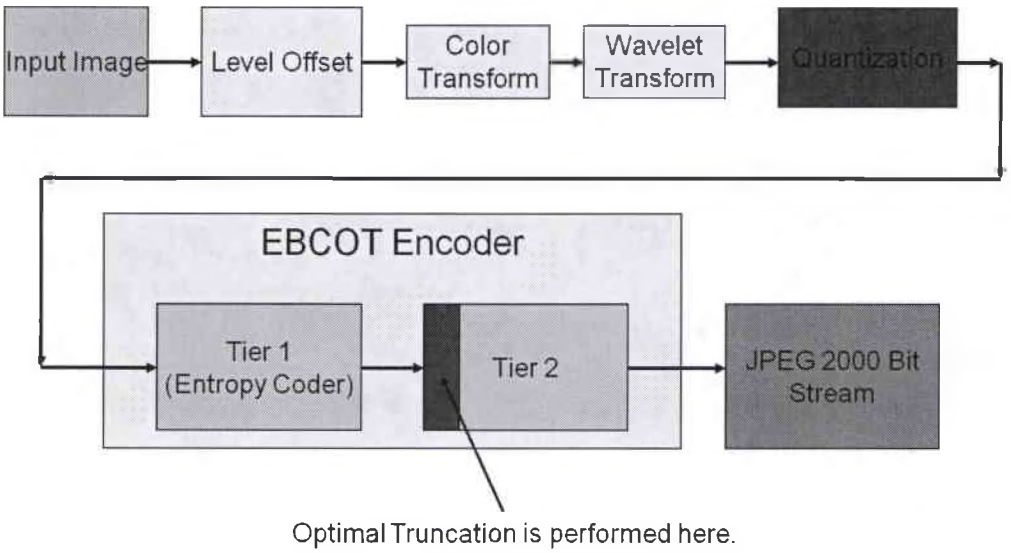

Figure 3.1: A simplified overview of the JPEG2000 encoding process.

efficiently when the mean value of the data is 0. The level offset, then, converts the unsigned data to signed data with values from -2^{B-1} to $2^{B-1}-1$, giving the data a mean value of 0. The data itself is not changed; its representation is just different. The two representations are shown below in Figure 3.2.

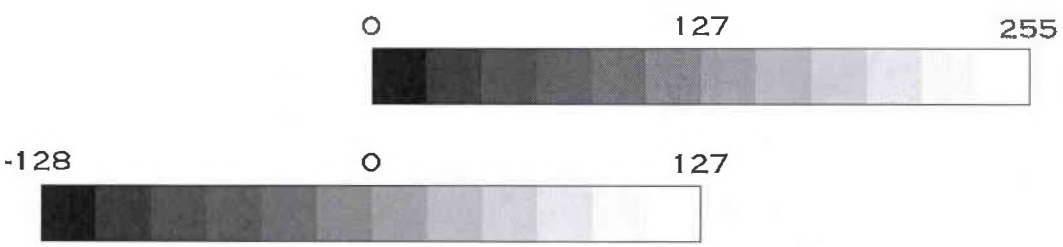

Figure 3.2: Top: unsigned representation of 8-bit, grayscale pixel data. Bottom: signed representation after level offset.

3.2.2 Color Transform

In a color image, pixels are formed typically from three color planes, or color components: red, green, and blue (RGB). Each color component c is stored with a number of bits B_c, with the sum of all three being the total bit depth B. For color images, typically, $B = 24$ and $B_c = 8$.

When pixel data is stored as RGB, the data is highly correlated between color planes, and there is a lot of redundant visual information in all three components. A color transform is used to move the visual information mostly into one component and to decorrelate the data. The transform used takes the signed RGB pixel values and converts them to luminance and chrominance components (YC_rC_b). Y represents the luminance component—the grayscale version of the image—that shows the amount of light intensity of each pixel. The chrominance components C_r and C_b are formed by taking the difference between the red component and the green component ($R - G$) and between the blue component and the green component ($B - G$), respectively. Using the Pythagorean theorem, the green component can be reproduced from the Y C_r and C_b components [15].

For lossless, or reversible, compression, Equations 3.1 and 3.2 are used for the forward and reverse color transforms, respectively, to keep all the data as integers and reconstruct the data exactly.

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} \left\lfloor \frac{R+2G+B}{4} \right\rfloor \\ B - G \\ R - G \end{pmatrix} \tag{3.1}$$

$$\begin{pmatrix} G \\ B \\ R \end{pmatrix} = \begin{pmatrix} Y - \left\lfloor \frac{C_b+C_r}{4} \right\rfloor \\ C_b + G \\ C_r + G \end{pmatrix} \tag{3.2}$$

Equations 3.3 and 3.4 below show the lossy, or irreversible, forward and inverse color transforms, respectively, used in JPEG2000. All these color transform equations—for reversible and irreversible processing—can be found in [12].

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.168736 & -0.331264 & 0.5 \\ 0.5 & -0.418688 & -0.081312 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{3.3}$$

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1.402 \\ 1 & -0.344136 & -0.714136 \\ 1 & 1.772 & 0 \end{pmatrix} \begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} \tag{3.4}$$

Implementing the color transform results in the original, RGB color data to become YC_rC_b data. An example of this is shown with the Baboon image in Figure 3.3.

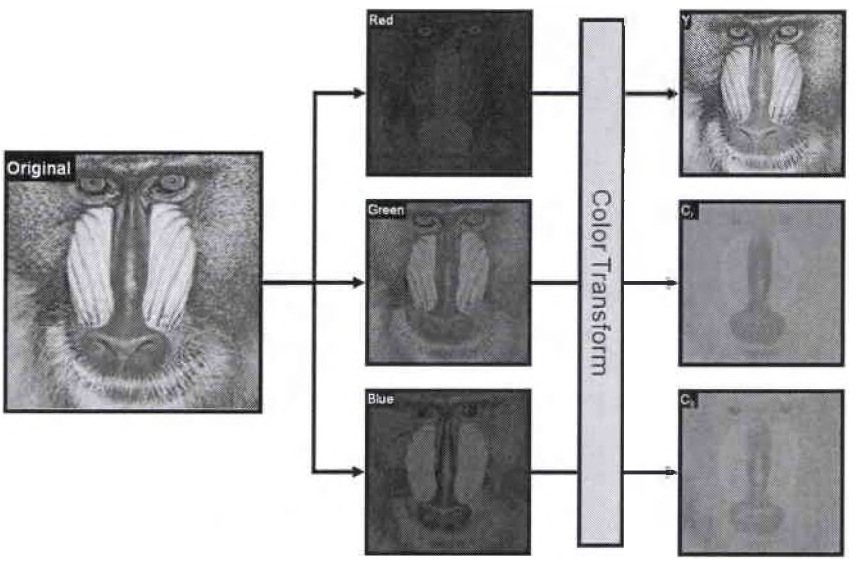

Figure 3.3: Example of the color transform on the Baboon.bmp image.

Another benefit to using the color transform when using lossy compression is that, since a majority of the important data resides in the luminance (Y) component, the

chrominance (C_r and C_b) components can be subsampled in order to further reduce the amount of information needed to be coded without a large loss in quality. A subsampling of 2:1 is typically employed by JPEG and MPEG for the C_b and C_r components [12]. Subsampling by 2 decreases the size of the color components by 4 and thus significantly decreases the amount of data to be coded. An example of this is shown in Figure 3.4.

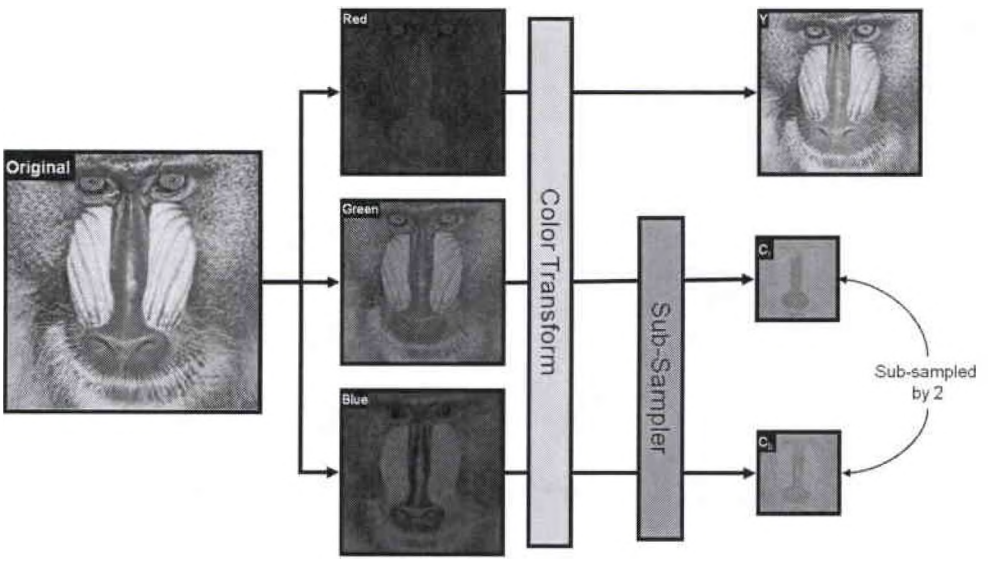

Figure 3.4: Example of subsampling the C_r and C_b color components after the color transform.

3.2.3 Wavelet Transform

After the color transform, the signed pixel values are passed through a discrete wavelet transform to decorrelate the data in the spatial and frequency domains. This process "pushes" the energy in the image into the upper left corner through the use of low- and high-pass filters. These filters process the rows and columns of the image in

sequence to produce four quadrants, each quadrant having been passed through the set of filters in a different order. In order to further decorrelate the data, additional passes may be performed on the quadrant that was passed through the low-pass filter twice. The data resulting from the wavelet transform is no longer pixel values but wavelet coefficients, which can be passed through the inverse wavelet transform to reconstruct the original pixel values. Mathematically, the discrete wavelet transform is completely reversible, or lossless. However, when representing floating point values with a finite number of bits, some precision may be lost resulting in small differences in the reconstructed pixels. If the wavelet is integer-based, the transform is lossless both mathematically and in implementation.

3.2.4 Quantization

Quantization is one of the places in the JPEG2000 process where data is removed in order to achieve a higher compression ratio and is therefore not performed in lossless compression. The process of quantization takes the wavelet coefficients, x[n], from the wavelet transform and divides them by a quantization step size, Δ_b. Dividing the coefficients reduces the number of bits required to represent a coefficient's value, and in turn introduces more zeros into the entropy coder, allowing it to encode code blocks more efficiently. The equation for quantization is

$$\tilde{x}[n] = \left\lfloor \frac{x[n]}{\Delta_b} \right\rfloor \tag{3.5}$$

where $\tilde{x}[n]$ are the quantized wavelet coefficients.

3.2.5 Tier 1

The Tier 1 portion of the encoder processes the quantized wavelet coefficients through three coding passes that determine the type of entropy coding that is performed within the arithmetic coder. Based on statistical modeling of many different images, the three coding passes generate contexts and binary decision values for every bit within an image. The decision bits are encoded using a context-adaptive arithmetic entropy coder called the MQ Coder, and the context information is used to select the estimated probability value from a lookup table, which the MQ Coder uses to create the compressed bit stream [13].

3.2.6 Tier 2

Once a compressed bit stream exists for every codeblock of the image, the JPEG2000 file stream is then assembled. Tier 2 functions as the assembler of all the pieces of the JPEG2000 file. Header information, including the width and height of the image, how many component planes there were, how many wavelet transform levels were performed, as well as information on coding style and other options JPEG2000 supports is included prior to the compressed data stream. The compressed stream, organized into bit streams for each codeblock, can be arranged in a progression scheme. In the JPEG2000 standard, there exists a set of five schemes to order and construct the final code stream that determines ease of access of specific data within the file. Depending upon which scheme is chosen, all the packets for a certain resolution, layer, component, or spatial position may be arranged together. The five progressions defined in the

standard are: Layer–Resolution–Component–Position (LRCP), Resolution–Layer–Component–Position (RLCP), Resolution–Position–Component–Layer (RPCL), Position–Component–Resolution–Layer (PCRL), and finally Component–Position–Resolution–Layer (CPRL). Once all the header information is included and the compressed data organized, the JPEG2000 file is complete.

An example of the structure of a progression scheme is given in Figure 3.5 for the LRCP progression. Note that all the image data in a JPEG2000 file is stored as packets, which contain precincts of compressed codeblock bit streams. A progression defines where a packet for a specific precinct will fall within the file. For the LRCP progression, the highest level is the layer, meaning any packets containing codeblock data that has been included for a layer's specified bitrate will be organized together. In the examples in Figures 3.5, 3.6, and 3.7, there are two layers, and all the data for a single layer is contiguous. The same logic applies for the next level, resolution. Within a specified layer, all the packets of codeblock data for a single resolution are contiguous. Within a resolution, all the data for a single component is contiguous, etc.

The first example, Figure 3.5 shows the required data to parse the entire image. Intuitively, this requires all the data, and is thus shown. In the second example, Figure 3.6 shows the parsing needed to view a grayscale version of the middle resolution of the image. Finally, in the third example, Figure 3.7 shows what is needed to parse a portion of the color image at a lower quality.

The advantage one progression scheme gives over another is evident when specific data is constantly needed to be retrieved from an image or many images. If one is always looking at the bottom-right portion of a stream of images, that bottom-right

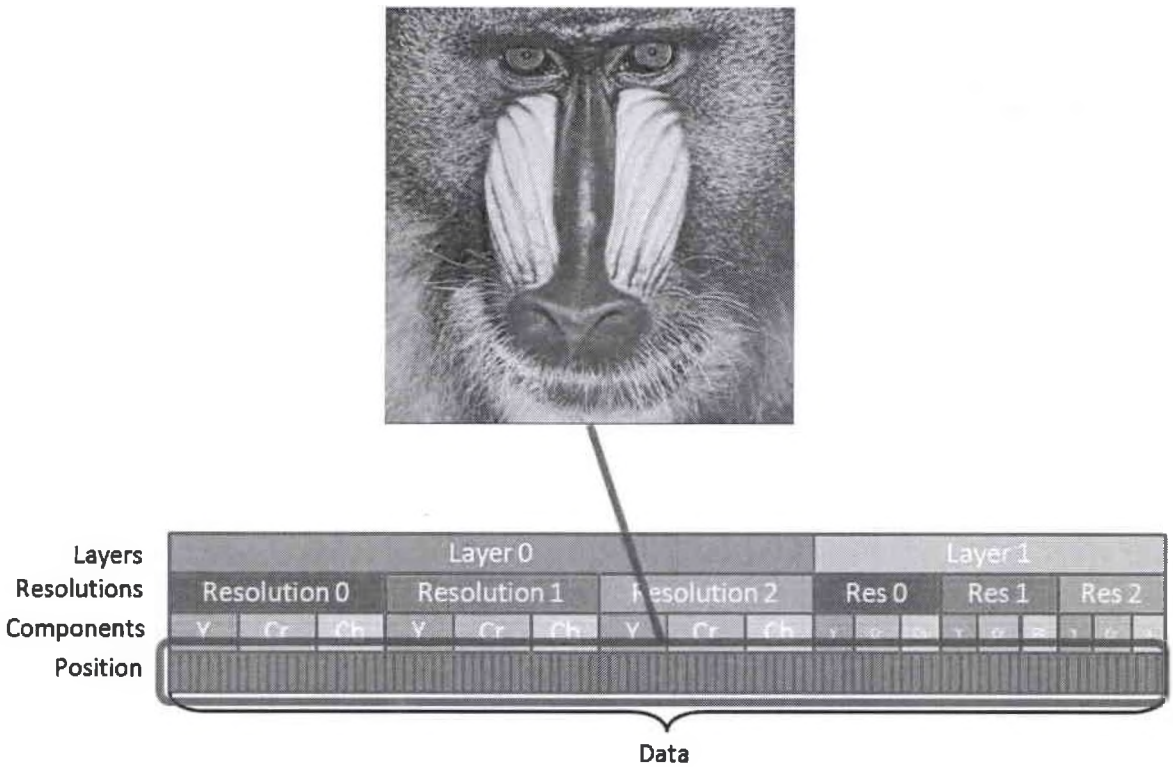

Figure 3.5: An example of the LRCP progression and the data required for the entire image.

portion must continuously be parsed out of the larger file and displayed. Parsing a certain position out of an image is time consuming when using a progression in which position is the lowest level of ordering. In the LRCP example, one must parse out the bottom-right corner for every component for every resolution for every layer. If the PCRL progression scheme was chosen, however, the position would be the highest level order, meaning that bottom-right corner's data is all contiguous.

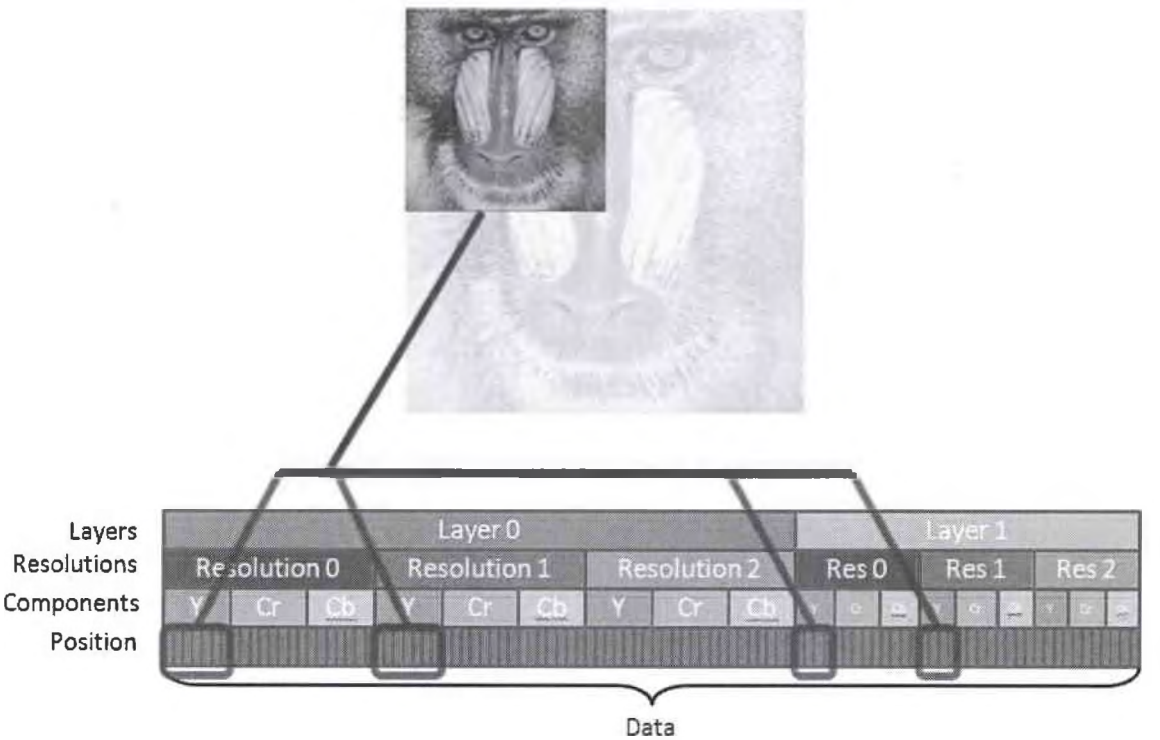

Figure 3.6: An example of the LRCP progression and the data required for a grayscale version of the middle resolution of the image.

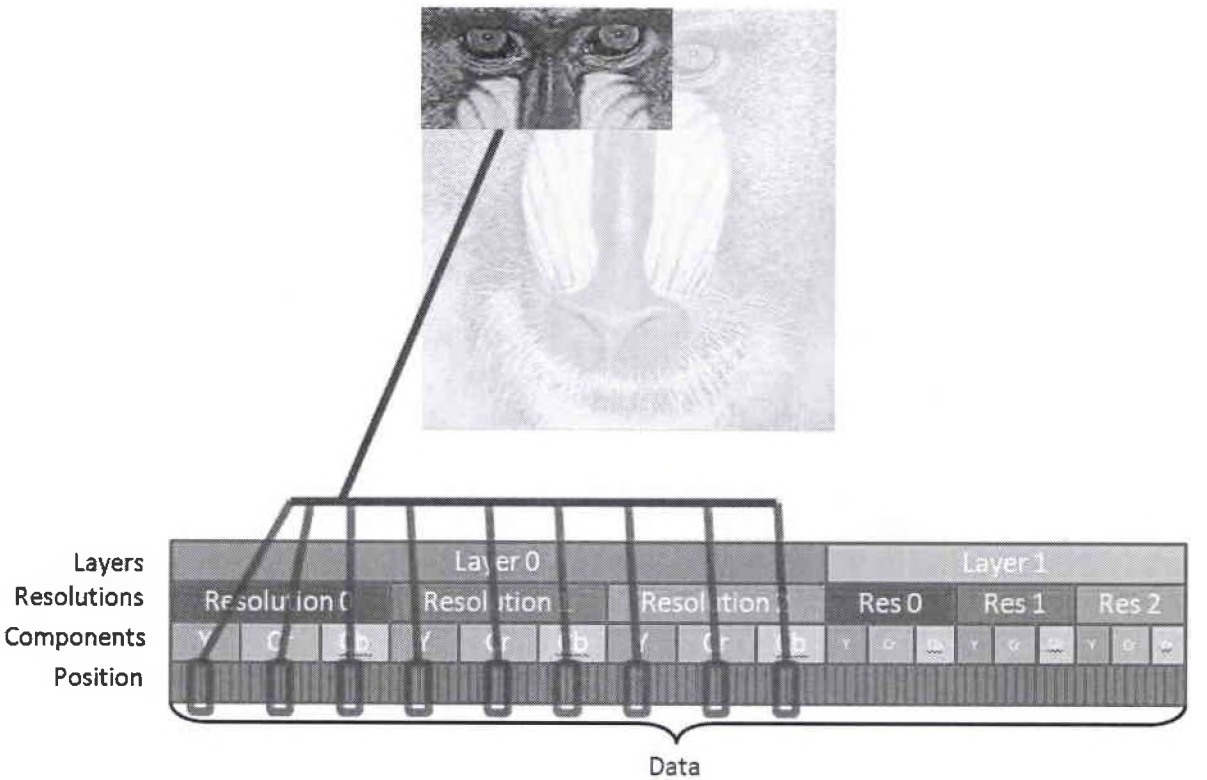

Figure 3.7: An example of the LRCP progression and the data required for just the eye portion of the color image at a lower quality.

CHAPTER 4

Discrete Wavelet Transform

The discrete wavelet transform (DWT) used in the JPEG2000 compression system improves upon the original JPEG's discrete cosine transform (DCT) by performing the transform on the entire image that is to be encoded. The JPEG DCT is performed on 8x8 pixel blocks of an image, causing blocking artifacts at higher compression ratios. The DWT, as implemented in JPEG2000, does not have this problem and only causes the image to lose its higher frequencies when compressed more aggressively, smoothing and blurring it. Blurriness in an image is more acceptable to the human visual system than is blockiness [3]. As the DWT operates on the entire image, it can be performed multiple times at coarser levels as in a filter bank. Figure 4.1 shows a typical cascaded, 3-level, FIR filter bank. A one-dimensional, D-level filter bank would consist of subbands L_D and H_1 through H_D, where D is the number of transform iterations.

The filter bank breaks up the input signal into scaling coefficients and wavelet coefficients at higher levels of coarseness by filtering the output of the low-pass filter recursively. A number of different levels may be used, and Figure 4.1 shows three levels as an example. The h_0 and h_1 represent the low pass and high pass impulse response coefficients for the FIR filter. When these coefficients are convolved with an input signal, the resulting output is a filtered signal. The output of each high- or low-pass filter

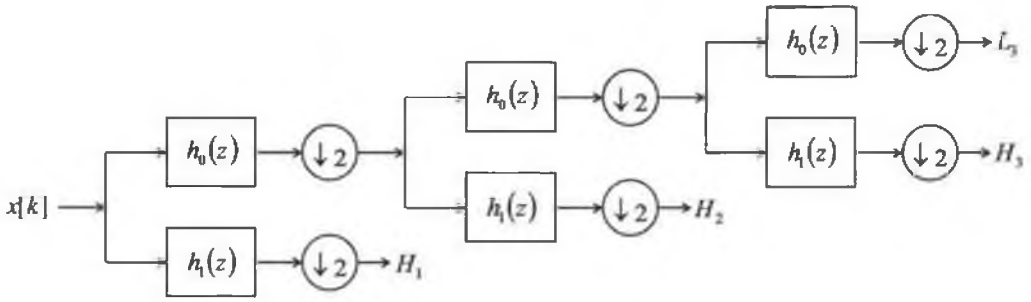

Figure 4.1: A cascaded filter bank, $D = 3$.

is called a subband, denoted b, where $b = \{H_d, L_d\}$ represents each filter the signal was processed by. This design is modified to two dimensions for the rows and columns of an image to create the two-dimensional wavelet transform used in JPEG2000. In the case of two dimensions, b is a two-dimensional subband representing which two filters the signal has been processed by, where $b = \{HH_d, HL_d, LH_d, LL_d\}$. Figure 4.2 shows an example of what the output of a one-level, two-dimensional wavelet transform looks like after processing the baboon image. The subbands are easily visible and are labeled.

4.1 Lifting

As Figure 4.1 shows, the method of implementing the wavelet transform using convolution requires operations on all of the samples by each filter and then downsampling both of the outputs. This is very inefficient, as half of the data is thrown away after processing. A method of factoring the discrete wavelet filter and implementing the transform on even- and odd-numbered samples that have already been downsampled was introduced by W. Sweldons [11]. Downsampling the input signal before filtering allows processing on only the data that is used, making the lifting

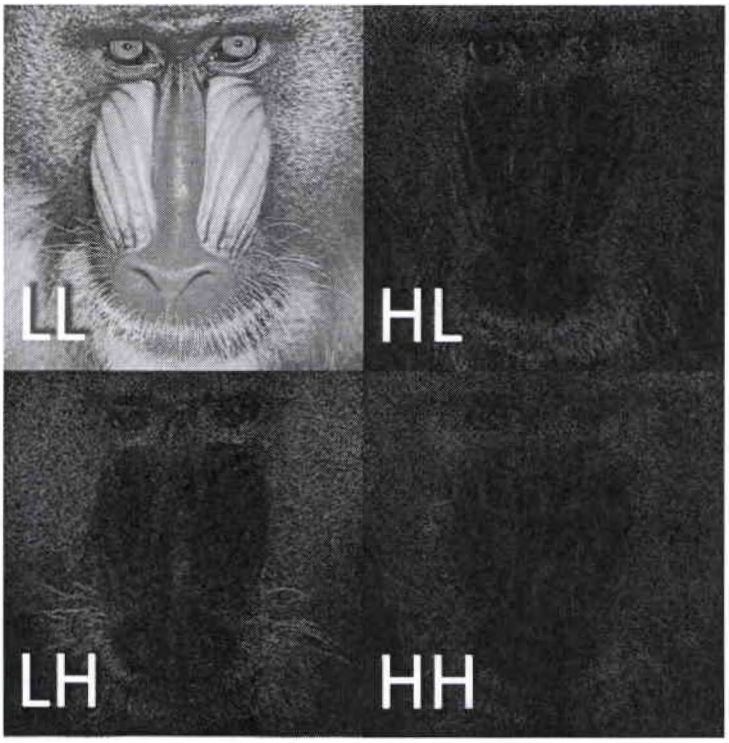

Figure 4.2: Result of a discrete wavelet transform on the baboon image, $D = 1$.

scheme very efficient. Lifting allows for a simpler implementation of symmetric extension for the borders of an image, and also results in a signal size that is exactly the same as the input signal size, and therefore no extra processing is needed to remove boundary artifacts caused by convolution. The JPEG2000 specification calls for implementation of the lossless compression scheme to use lifting to create the wavelet coefficients in order to preserve reversibility [9]. Figure 4.3 shows how the input signal $x[n]$ is split into even and odd samples, $x_e[n]$ and $x_o[n]$, which are passed through the lifting steps separately, effectively downsampling the data before filtering it.

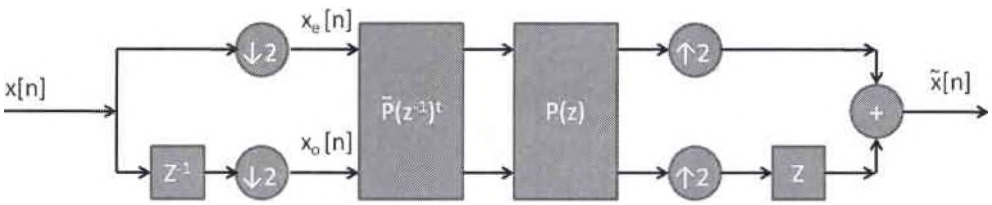

Figure 4.3: Lifting implementation of the discrete wavelet filter process.

4.2 Wavelets In JPEG2000

Two transforms are used for the wavelet transform in JPEG2000; Daubechie's 9/7 is used for irreversible, or lossy, compression, and the LeGall 5/3 uses an integer-to-integer transform to achieve reversible, or lossless, compression. The "loss" associated with the 9/7 transform is only the loss of precision when representing floating point numbers with a finite number of bits. This thesis focuses on the 9/7 wavelet used in irreversible compression, as optimal truncation and the PCRD algorithm remove data from an image in order to provide irreversible compression.

The 9 and 7 in "Daubechie's 9/7" stand for the number of taps in the low- and high-pass analysis filters, respectively. For the analysis, or forward, transform, the low-pass coefficients are denoted h_0 while the high-pass coefficients are h_1. For the synthesis, or inverse, transform, the low-pass coefficients are denoted g_0 and the high-pass coefficients g_1. The coefficients used for the analysis and synthesis transforms in this paper are shown in Tables 4.1 and 4.2 and are defined in the JPEG2000 ISO standard [9].

It should be noted that the coefficients in Tables 4.1 and 4.2 are different than the lifting coefficients used in many implementations to apply the wavelet transform.

Table 4.1: Daubechie's 9/7 Analysis Wavelet Coefficients

z	$h_0(z)$	$h_1(z)$
0	0.602 949 018 236 360	1.115 087 052 457 000
1,-1	0.266 846 118 442 875	-0.591 271 763 114 250
2,-2	-0.078 223 266 528 990	-0.057 543 526 228 500
3,-3	-0.016 864 118 442 875	0.091 271 763 114 250
4,-4	0.026 748 757 410 810	

Table 4.2: Daubechie's 9/7 Synthesis Wavelet Coefficients

z	$g_0(z)$	$g_1(z)$
0	1.115 087 052 457 000	0.602 949 018 236 360
1,-1	0.591 271 763 114 250	-0.266 846 118 442 875
2,-2	-0.057 543 526 228 500	-0.078 223 266 528 990
3,-3	-0.091 271 763 114 250	0.016 864 118 442 875
4,-4		0.026 748 757 410 810

This is because the lifting scheme uses coefficients obtained from factoring the original wavelet coefficients into lifting steps that operate on the even and odd pixels of an row or column. These lifting steps require a different set of coefficients: α, β, γ, and δ, with a scaling factor of K. These lifting coefficients for Daubechie's 9/7 are given in Table 4.3 [5].

Table 4.3: Daubechie's 9/7 *Lifting* Wavelet Coefficients

α	-1.586 134 342
β	-0.052 980 118 54
γ	0.882 911 076 2
δ	0.443 506 852 2
K	1.149 604 398

4.3 Energy Gain from the Transform

As the transform operates on subsequent levels of coefficients, the energy in the subbands has a gain associated with it. Depending upon the filter tap coefficients used and the method of performing the transform, this gain can differ between implementations. The coefficients given in Table 4.1 and the lifting coefficients derived from those and given in Table 4.3 have a unity gain when used to apply the forward wavelet transform. The synthesis coefficients, however, apply the inverse transform, which, in this case, has a non-unity gain. This gain is important to account for when performing quantization or distortion calculations in the PCRD algorithm.

Wavelet gain from a biorthogonal wavelet transform is typically unity for the low-pass filter and 2 for the high-pass filter. In the two-dimensional wavelet transform, the gain is unity for the LL subband, 2 for the LH and HL subbands, each filtered by the high-pass filter once, and 4 for the HH subband, which has been filtered by the high-pass filter twice. The JPEG2000 standard specifies that decoders should expect coefficients from the wavelet transform to have these gains, so if using a forward transform with unity gain, the biorthogonal wavelet transform gains need to be separately applied to the coefficients.

To calculate gains for the forward and inverse wavelet transforms, a reference basis vector, $\mathbf{s}_b$, defined as a vector with elements that are the high- or low-pass coefficients, is used to find the translates of the basis vector, whose L2-norms are the subband gains. The basis vector and its translates are defined in [12] as:

$$\begin{aligned} s_{L_1}[n] &= g_0[n] \\ s_{H_1}[n] &= g_1[n] \end{aligned} \tag{4.1}$$

$$
\begin{aligned}
s_{L_d}[n] &= \sum_k s_{L_{d-1}}[k] g_0[n-2k] \\
s_{H_d}[n] &= \sum_k s_{H_{d-1}}[k] g_0[n-2k]
\end{aligned}
\tag{4.2}
$$

Equation 4.2 is very similar to a conventional down-sampled convolution of the previous reference vector and the low-pass filter tap coefficients. However, the down-sampling occurs in the variable k, which is the index of the summation. Depending on whether n is even or odd, the convolution of the previous reference vector and the even or odd samples of the low-pass filter is calculated. The gain associated with each level of the wavelet transform is calculated using the square of the L2-norm of the appropriate reference vector. Subband gain is shown in Equation 4.3.

$$
G_b = \|\mathbf{s}_b\|^2 \tag{4.3}
$$

These subband gains must be used when calculating distortion for the PCRD algorithm as well as before quantization to account for the gain applied by the inverse transform. Table 4.4 shows the subband gains for 7 wavelet transform levels of the 9/7 wavelet.

Table 4.4: Daubechie's 9/7 Wavelet Subband Gain Factors G_b Used in Distortion Weighting

d	G_{LL_d}	G_{LH_d/HL_d}	G_{HH_d}
1	3.86534814957796	1.02280984035474	0.27064572945149
2	16.9975586827024	3.98802169718592	0.93568243264315
3	70.8608348870272	17.5056459762554	4.32464056533563
4	286.925741944958	72.8615766103804	18.5023808256588
5	1151.50823940491	294.860759763489	75.5034698607394
6	4610.39714362674	1183.17015313397	303.637966026903
7	18447.9631384652	4736.97625448668	1216.33721116798

4.3.1 Distortion Weighting Based on Energy Gain

Energy gain from the synthesis, or inverse, wavelet transform must be taken into account when removing data from the image, as in quantization, or when calculating distortion associated with removing certain data from the image, as in optimal truncation. It is this gain that is the reason for weighting the distortion calculations in the optimal truncation algorithm [12]. Weights are needed in order to correctly determine the distortion associated with truncation and to determine how to properly quantize a certain subband when the inverse wavelet transform applies different amounts of gains to each of those subbands. The wavelet weight is defined by the amount of squared error introduced by a unit error in a transformed coefficient bit. This unit error differs based on which subband the coefficient is in. The reason the weight depends on subband is that the low- and high-pass filters have different gains, so each subband b of the resulting coefficients will have its own gain. Since we are finding distortion introduced after decoding, we use the synthesis vectors associated with each subband in the gain calculation. These synthesis vectors are simply translates of a reference basis vector $\mathbf{s}_b$.

Each subband b has a gain associated with it from the wavelet transform's low- and high-pass filters. Even though the transform may be implemented using the lifting scheme, lifting coefficients are simply found from factoring the low- and high-pass filter coefficients, and the gain is the same for both implementations. This gain is calculated from translations of a reference basis vector $\mathbf{s}_b$ based on how many times a coefficient in subband b has been passed through each filter. For the two-dimensional wavelet transform, the subbands are labeled by level and with which filter has been

applied to create the subband in question. Figure 4.4 shows the labels of these subbands. The gain G_b for each subband b is labeled accordingly.

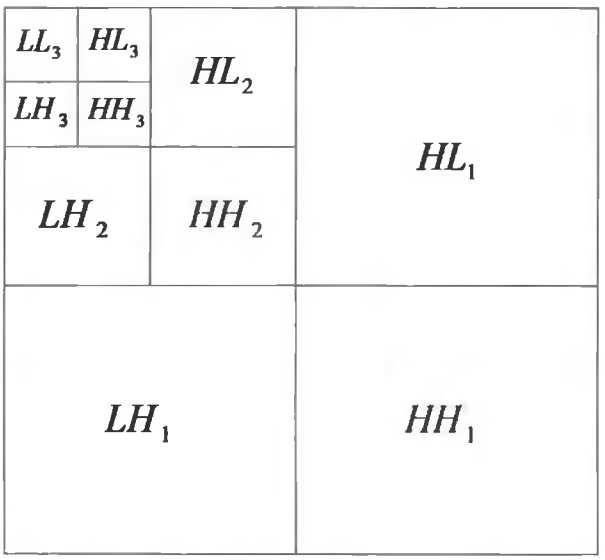

Figure 4.4: Subbands of a 3-level, 2D wavelet transform.

4.3.2 Calculating the Subband Gains

Equations 4.1 and 4.2 show the calculations of the reference vectors used in subband gain calculations. The reference vectors are calculated for the low- and high-pass filters, and are therefore one-dimensional. Since the distortion weighting is applied to a two-dimensional subband, the one-dimensional vectors are multiplicative and are simply multiplied together to achieve the two-dimensional synthesis vectors, $\mathbf{s}_{b_d}$, where $b = \{LL, LH, HL, HH\}$. The energy gain from the wavelet transform is the amount of squared error introduced by a unit error in a transformed coefficient. Therefore, the squared norm of the two-dimensional synthesis vectors is needed to calculate the

energy gain, shown in Equation 4.3. This equation can apply to both one-dimensional and two-dimensional vectors. Equation 4.4 from [12] shows the relationship between the one- and two-dimensional gains and synthesis vectors.

$$\begin{aligned}
\mathbf{s}_{LL_D} &= \mathbf{s}_{L_D}\mathbf{s}_{L_D} \Longrightarrow G_{LL_D} = G_{L_D} \cdot G_{L_D} \\
\mathbf{s}_{HL_d} &= \mathbf{s}_{H_d}\mathbf{s}_{L_d} \Longrightarrow G_{HL_d} = G_{H_d} \cdot G_{L_d} \\
\mathbf{s}_{LH_d} &= \mathbf{s}_{L_d}\mathbf{s}_{H_d} \Longrightarrow G_{LH_d} = G_{L_d} \cdot G_{H_d} \\
\mathbf{s}_{HH_d} &= \mathbf{s}_{H_d}\mathbf{s}_{H_d} \Longrightarrow G_{HH_d} = G_{H_d} \cdot G_{H_d}
\end{aligned} \tag{4.4}$$

CHAPTER 5

Quantization

Quantization is a method of removing data from an image in order to improve Tier 1 coding efficiency. The process of removing data involves dividing the coefficients resulting from the wavelet transform by a value, Δ_b, depending upon the desired compression ratio and a gain factor resulting from the inverse wavelet transform. As the wavelet coefficients are stored in memory with a number of bits per component, B_c, dividing the coefficients will shift their values toward zero and introduce zeros in the most significant bits. These most significant bit planes will then contain a large amount of zeros, which become zero coded within the entropy coder allowing for a much higher compression ratio. Figure 5.1 shows an example of a group of coefficients before and after quantization, and how the division introduces a plane of zeros into the resulting coefficients.

5.1 Quantization as a Means of Removing Data

In JPEG2000, there exist two methods of removing data to gain compression efficiency. Quantization is the coarse—and quicker—method, and optimal truncation via the Post-Compression Rate Distortion algorithm is the more fine-grained—and slower—method. Optimal truncation is generally the more preferred method, because it produces optimal (a minimum) mean-square error between the original image and

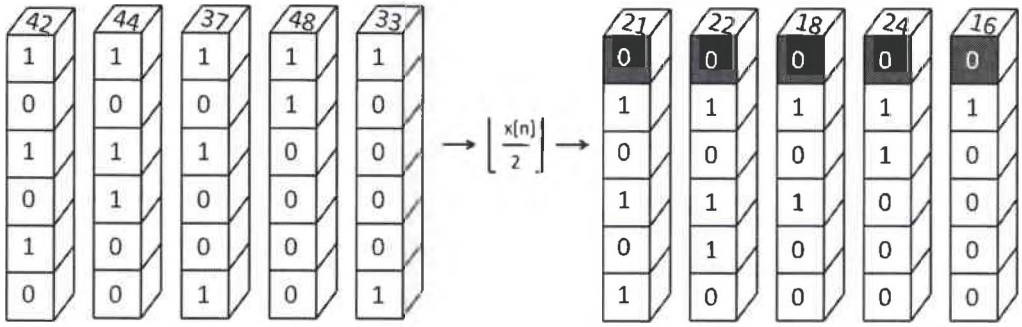

Figure 5.1: Example of binary coefficient values before and after quantization by a Δ_b of 2.

the reconstructed image for a given rate. When removing data with optimal truncation, the default quantization value is typically set to unity to preserve all the coefficient data going into the entropy coder.

In the case of using a non-unity quantization value, however, data is removed uniformly from a subband. Each subband can have a different quantization value, and a derived quantization method is defined to calculate optimal values for each subband based on the gain introduced by the inverse wavelet transform.

5.1.1 Derived Quantization

Derived quantization requires only a single quantization parameter to be given before performing the quantization on the subband data. It relies on the synthesis wavelet subband gain parameter, G_b, calculated in Section 4.3, to derive the resulting quantization steps for each subband. This method effectively scales the desired quantization step by the gain factor applied by the inverse wavelet transform in order to properly divide each subband by the correct step size. Derived quantization provides an optimal quantization of each subband in a mean-squared error sense. An

explanation of synthesis wavelet energy gain applied to a default step size is given in Section 5.2.

5.1.2 Expounded Quantization

Expounded quantization allows one to specify the quantization step parameter for each subband explicitly. Although this does not mean that the synthesis wavelet gain cannot be applied to each explicit step size, the result of expounded quantization will not be as optimal in a mean-squared error sense as derived quantization. Benefits of expounded quantization may be decreased Tier 1 processing time by preserving low frequencies and applying a very high quantization to the larger, high frequency subbands. Explicitly setting subband step sizes also allows for preservation of specific frequency data while quantizing other data in the image.

5.2 Inverse Wavelet Transform Gain in Quantization

When performing the inverse wavelet transform, there is a gain applied to the resulting data. This gain is the same gain used to calculate bit distortions within optimal truncation, the calculations of which are shown in Section 4.3.2. As there is a gain applied to the data by the inverse transform, a uniform division of the coefficients in all subbands results in an effectively different division of each subband. Quantizing the higher frequencies more than the lower ones might be desirable, but the effective amount of quantization is dependent upon the inverse transform gain. Because of this gain, a default quantization step Δ_d that is the desired amount by which to quantize the data must first be scaled according to which subband in which is will be used. The amount by which to scale the default quantization step is given by [12] and shown in Equation 5.1 below, where Δ_d is the default quantization step

size that has the inverse wavelet gain applied to it. Scaling the default step size by the subband-dependent gain G_b produces an array of quantization steps Δ_b that are used on their respective subbands to properly quantize the data.

$$\Delta_b = \frac{\Delta_d}{\sqrt{G_b}} \tag{5.1}$$

CHAPTER 6

Tier 1

JPEG2000's compression performance gains over standard JPEG come mainly from the Tier 1 portion of the encoding process. Tier 1 contains the an entropy coder called the MQ Coder that encodes binary decision bits using most probable symbols based on contexts for each bit. The binary decision bits and contexts are generated for every bit used to store the resulting coefficients from the wavelet transform. This process reduces the amount of bits necessary to store the coefficients, resulting in data compression.

It is important to understand the details of the Tier 1 process, because the PCRD algorithm uses information obtained during Tier 1 coding and the calculations used for distortion are derived from the coding pass operations.

6.1 Lookup Tables

In order to generate the contexts for each bit within a bitplane, the coding passes use statistical lookup tables provided by the JPEG committee in the standard, [9], that use subband information and number of significant neighbors to define which context in the table to use.

6.2 Coding Passes

Tier 1 consists of three coding passes–the significance propagation pass (SPP), the magnitude refinement pass (MRP), and the clean-up pass (CUP)–that scan the data to determine which type of entropy coding will be applied. The coding passes operate on bitplanes of small portions of each subband called codeblocks. Codeblocks are either 32x32 or 64x64 coefficients in dimension, together making up all the subbands that were produced by the wavelet transform. These codeblocks do not cross subband boundaries and thus some codeblocks on the edges of the subbands may be smaller than the set size. Figure 6.1 gives an example of how the codeblocks make up the subbands. As a codeblock is made up of coefficients that have a bit depth B_c, the codeblocks themselves have a depth. The set of bits at one level of the coefficients within a codeblock make up a plane of bits for that codeblock. The codeblocks, then, have B_c bitplanes. The coding passes operate on one bitplane at a time, and each codeblock is encoded by the Tier 1 coding passes independently. The coding passes produce context and binary decision bits for the bits from a bitplane of a component in an image.

6.2.1 Order of Coding Pass Operation

The most significant bitplane of a codeblock is operated on by the coding passes first, followed by the second-most significant bitplane and so on down to the least significant bitplane. All of the bits for each bitplane are encoded by the coding passes, but no bit is encoded by more than one pass. For the most significant bitplane, only the clean-up pass is run, and thus creates the contexts and decision bits for each bit in the most significant bitplane. The subsequent bitplanes are encoded by all three

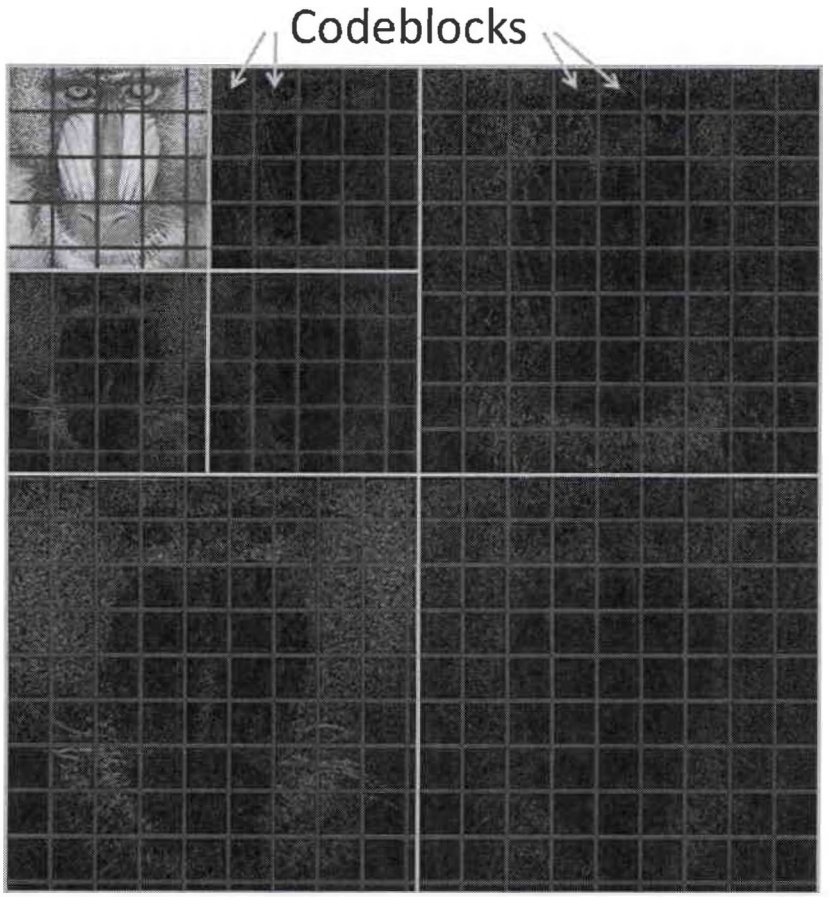

Figure 6.1: Subbands of a two-level wavelet transform, showing codeblock boundaries. Notice the codeblocks do not cross subbands.

coding passes with the significance propagation pass running first, followed by the magnitude refinement pass and finally the clean-up pass. This technique is called fractional bitplane coding, as each coding pass only operates on a fraction of the total number of bits. Once one pass codes a bit from a bitplane, it records that the bit has been coded in a state table, σ'. The coding passes refer to the σ' state table to determine if the bit has already been coded by another pass; if it has, the coding pass skips that bit.

Each pass scans through each bitplane of a codeblock by taking vertical stripes of four coefficients at a time and examining the stripe's neighborhood, a selection of coefficients surrounding the stripe. For the stripes at the edges of a codeblock, zeroes are used for the neighbors where no coefficients exist. An example of the scanning pattern is shown in Figure 6.2, and the zero padding technique is shown in Figure 6.3. In the figures, the current bit being coded is shown in white, and its neighborhood is within the box. Horizontal neighbors are green, vertical neighbors are red, and diagonal neighbors are blue. In the case of Figure 6.3, the bits shown in black are the zero-padded bits at the edges of a codeblock.

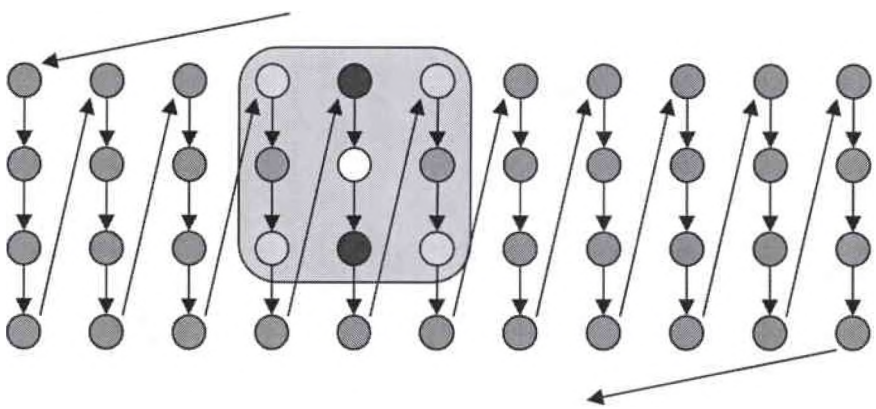

Figure 6.2: Scanning pattern of a coding pass, showing current bit and its neighborhood.

6.2.2 Significance Propagation Pass

The Significance Propagation Pass is the first coding pass to run on all the bitplanes below the most significant bitplane. It scans through the bits, and when it encounters one that is not significant, it assigns that bit as significant by setting a

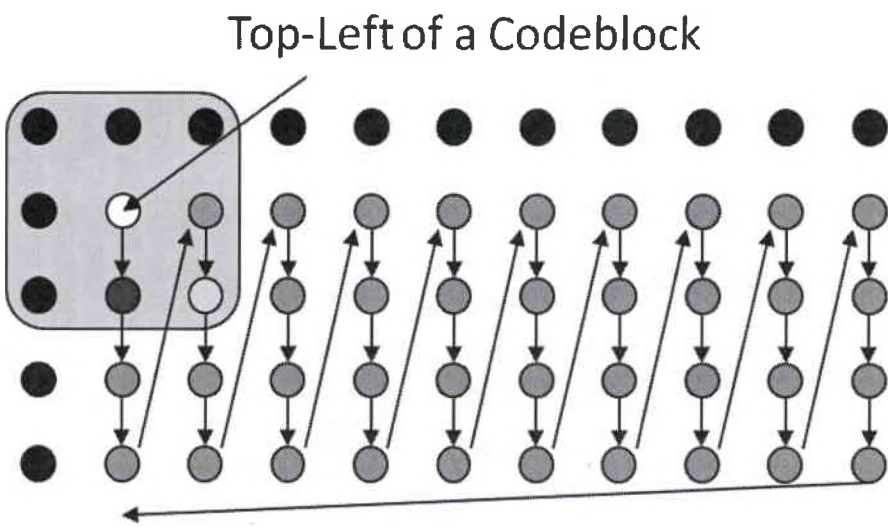

Figure 6.3: Zero-padding technique in a coding pass used at the edges of a codeblock.

corresponding bit within a binary state table, σ, if any of its neighbors are already significant. The σ state table is used in the table lookup for generating contexts. The SPP lookup table is shown below in Figure 6.4. In the lookup tables, the values h, v, and d correspond to the sum of the horizontal, vertical, and diagonal neighboring bits, respectively.

6.2.3 Magnitude Refinement Pass

The Magnitude Refinement Pass (MRP) runs only on coefficients that have already become significant, as indicated by the σ state table. However, the MRP could encounter a bit that has been made significant by the SPP and would thus skip it, as only one coding pass codes a single bit. The coding pass membership state table, σ', ensures this is the case. Once a bit has become significant, all of its bits in the lesser significant bitplanes are coded by the MRP. Contexts for the MRP are generated from

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

Figure 6.4: Lookup table used to produce contexts for the Significance Propagation Pass.

a lookup table depending upon the significance recorded in σ and delayed significance, which is based on the value within σ.

6.2.4 Clean-Up Pass

The clean-up pass acts much like the significant propagation pass, but will perform a run-length coding on a bit whose neighbors all have a significance of zero. This is what allows for a large compression gain when encountering large sections of all zeroes within a bitplane. The clean-up pass runs last on all the bitplanes except the most significant, and will thus code any bit that was not coded by the SPP or the MRP.

6.3 MQ Coder

Contexts and decision bits resulting from the coding passes are passed into the MQ Coder, a context-adaptive binary arithmetic encoder. Using a lookup table, the MQ Coder creates a stream of bits that are output a byte at a time to construct the

final, JPEG2000 data stream. Codeblock bitstreams are embedded, meaning they can be independently truncated and included into the final bitstream. This truncation is what makes optimal truncation and the PCRD algorithm possible by removing data after compression to achieve an optimal distortion per bitrate. In order to determine which portion of each codeblocks' bitstream contains the most important information so that truncation can be made at the optimal place, a knowledge of what happens within the MQ Coder is needed. The distortion calculations that are made on a coding pass basis are explained in Section 7, but the rates (length in bytes) of the bitstream outputs are a direct result of the process that takes place within the MQ Coder.

6.3.1 MQ Coder Internals

Within the MQ Coder, there exist a set of registers which determine the coder's state. As contexts and decision bits are passed into and are encoded by the MQ Coder, the registers are changed based on probability lookup tables, and the coder's state therefore changes as well. The most important registers within the MQ Coder are a 16-bit A register, a 28-bit C register (partitioned into a carry bit, a partial code byte, three spacer bits, and a 16-bit active region), and an 8-bit byte out register. These registers are portrayed in Figure 6.5.

The A register is context-dependent and changes based on the probability lookup table values chosen by the context and decision bit input. The A register then replaces the active region of the C register. It is called the active region because the values are being constantly updated for every new context input. During coding, the active region of the C register gets shifted into the rest of the C register one bit at a time.

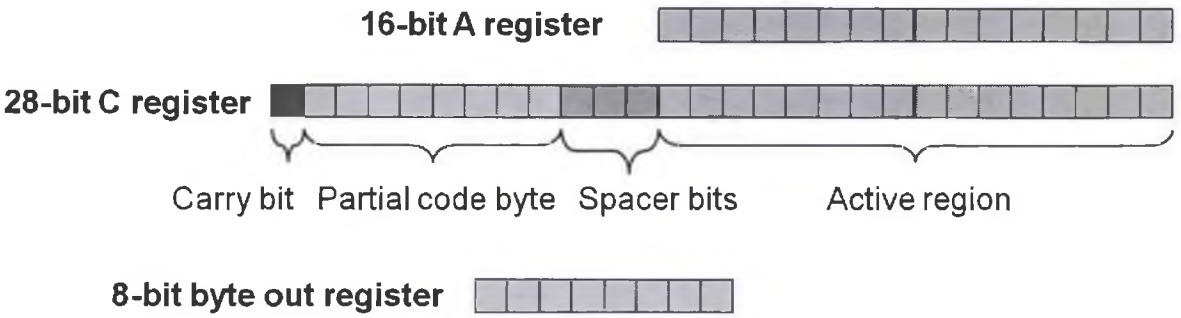

Figure 6.5: Important state registers within the MQ Coder.

When the active portion has been shifted eleven times, the partial code byte has been filled, and the partial code byte is placed into the byte out register. The next time the partial code byte is full, the byte out register is placed onto the end of the codeblock code stream and the byte out register is again replaced with the partial code byte portion of the C register. This process is illustrated in Figure 6.6.

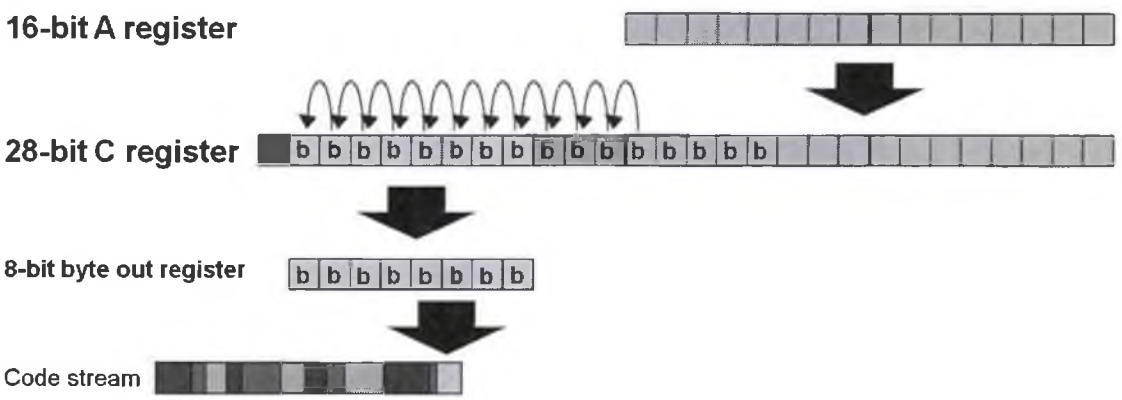

Figure 6.6: The process flow of creating a codeblock bit stream.

6.3.2 End of a Coding Pass

The MQ process is repeated during a single coding pass while that coding pass creates contexts and decision bits for the bits that it encodes within a bitplane. At the end of a single coding pass, the bits that have been shifted into partial code byte register may not completely fill that register. The byte out register may also contain a byte that has yet to be appended to the end of the codeblock bitstream. These bits and the byte out register are left where they are and the next coding pass may begin, meaning that some of the bits attributed to one coding pass may actually belong to the coding pass before it. This must be accounted for when calculating the rate at which a truncation of the code stream may occur so that the decoder can receive all the data from the coding pass it is decoding. In order to preserve this data for use when truncating a codeblock, the MQ Coder's state variables are saved at the end of every coding pass. Figure 6.7 shows how the registers may contain data at the end of a coding pass.

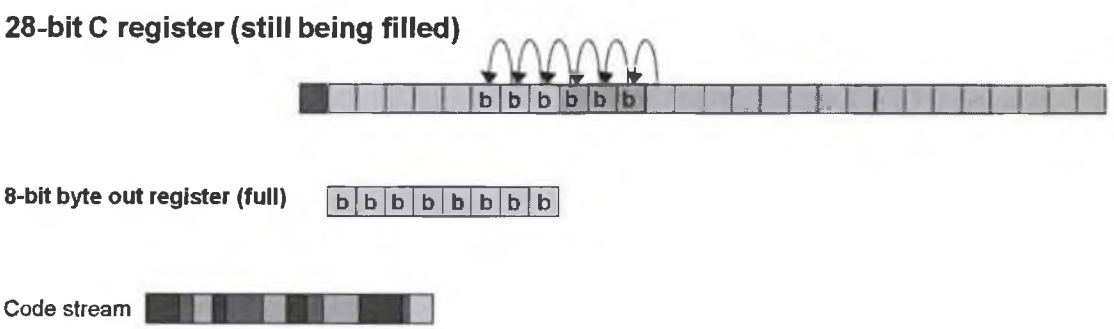

Figure 6.7: End of a single coding pass. When the encoder finishes a single coding pass, some bits may still be left in the byte out register and in the C register. These bits are left in there, and the next coding pass runs.

6.3.3 End of a Codeblock

At the end of a codeblock, the extra data that was leftover from the final coding pass needs to "flushed" onto the end of the code stream so the decoder can fully decode the final coding pass. This flush places the residual bits left in the registers onto the end of the code stream for that codeblock. When a truncation point is chosen for a codeblock by the PCRD algorithm, the state variables that were saved at the end of the coding pass corresponding to the chosen truncation point are restored, and an MQ Coder flush is performed as if the end of the coding pass at which the codeblock was truncated was actually the end of the codeblock during the encoding process. This allows the residual data that was included in the next coding pass output to be included, so that the decoder may correctly decode the entire coding pass. Figure 6.8 shows the flush procedure.

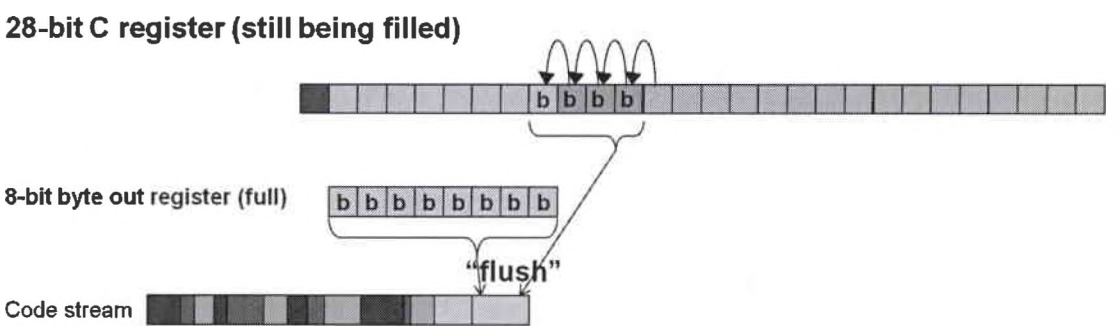

Figure 6.8: End of a codeblock. An MQ Coder flush occurs that pushes the residual bits onto the end of the code stream.

CHAPTER 7

Post-Compression Rate-Distortion Optimization (PCRD-Opt)

Post-Compression Rate Distortion Optimization is the algorithm that implements optimal truncation, the process of truncating codeblock bit streams to achieve a higher compression ratio and an optimal rate per distortion introduced into the decoded image. [9] and [12] describe the optimization algorithm in detail, using the well-known solution to constrained optimization problems of Lagrange multipliers. This section defines the distortions and rates that are used in the optimization problem and details the algorithm used to find optimal truncation points for each codeblock.

7.1 Distortion and Rate

A distortion, D, for a given image is defined as the mean-squared error (MSE) between the decoded wavelet coefficients, $\tilde{q}[n]$, and the original wavelet coefficients, $q[n]$, prior to compression. The equation for distortion is given in Equation 7.1.

$$D = \sum_n \left(\tilde{q}[n] - q[n]\right)^2 \tag{7.1}$$

The rate, R, of an image is the number of bytes used to store it. The goal of the PCRD-Opt algorithm is to find an optimal distortion per a constrained rate for a compressed image. In order to achieve a given rate, the PCRD algorithm finds a set of

truncation points, z_i, for each codeblock. The set z_i consists of individual truncation points, n_c, for each coding pass, c, used to encode that codeblock. These truncation points specify the rates at which a codeblock's code stream may be truncated, thus reducing the overall rate of the image and increasing the resulting distortion after decoding. The optimization problem is given by [9] as: find the set of z_i values which minimizes D subject to the constraint $R \leq R_{max}$, where R_{max} is the max desired file size of the image after compression.

7.1.1 Distortion Optimization

To solve the optimization problem subject to the constraint given, a Lagrange multiplier, λ, is used so that a set of truncation points $\{z_i, \lambda\}$ minimizes

$$D(\lambda) + \lambda R(\lambda) = \sum_i \left(D_i^{(z_i,\lambda)} + \lambda R_i^{(z_i,\lambda)} \right) \tag{7.2}$$

for some $\lambda > 0$. It is then easy to see that the distortion, D, cannot be further reduced without increasing the rate, R. If a value of λ is found such that the set of truncation points, $\{n_i, \lambda\}$, minimizes Equation 7.2, and $R = R_{max}$, then that set of truncation points is a solution to the optimization problem [12]. Typically, because the set of truncation points is discrete, a λ that minimizes Equation 7.2 and yields a rate $R = R_{max}$ will not be found. However, as there are many truncation points and many codeblocks within an image, a λ can typically be found that yields a rate R sufficiently close to R_{max}.

7.1.2 Distortion In EBCOT

Because the truncation points used in the optimal truncation algorithm are defined for each coding pass, the amount of distortion introduced to the decoded image by

removing that coding pass is needed. Each coding pass codes a fraction of the bits in a bitplane and outputs a bit stream. Truncating a coding pass involves removing the bit stream for that coding pass and each coding pass performed after it. In order to determine the amount of distortion introduced by removing a specific coding pass, the distortion for each coding pass must be known. To find distortion associated with a coding pass, the distortion introduced by removing a single bit from a codeblock can be calculate for every bit coded by a coding pass. These distortions are called bit-distortions.

7.1.3 Bit-Distortion Calculations

Bit-distortions are calculated for every bit of a sample once that sample has become significant. Because decoders do not estimate a decoded coefficient using midpoint reconstruction if the coefficient after truncation is zero [9], two equations are needed for bit-distortion estimation: one for when a sample is just becoming significant, and one for a magnitude refinement after a sample has become significant. When the significance propagation pass or the clean up pass is truncated, the entire coefficient's magnitude is truncated, resulting in a value of zero. The reason for this is that the SPP and CUP only code coefficients until they become significant; once they are significant, the magnitude refinement pass codes those coefficients. If an SPP or CUP is truncated, the resulting coefficient is zero. Bit-distortions are denoted $D^p[n]$, where p is the bit-level within a sample, and n is the position within the codeblock. Equation 7.3 shows the distortion calculation for a sample that has just become significant (either by the SPP or the CUP), and Equation 7.4 shows

distortion calculation for a sample whose magnitude is being refined (by the MRP). These equations are defined in the JPEG2000 standard [9].

$$D^p[n] = 2^{2p} G_b \Delta_d^2 [v^p[n]^2 - (v^p[n] - 1.5)^2] \tag{7.3}$$

$$D^p[n] = 2^{2p} G_b \Delta_d^2 [(v^p[n] - 1)^2 - (v^p[n] - 0.5 - v)^2] \tag{7.4}$$

Here, G_b is the wavelet gain calculated in Equation 4.3 for the subband b within which the sample $s[n]$ resides. Δ_d^2 is the error produced by the default quantization step size for subband b. v is the value of the bit at bit-level p, $\in \{0, 1\}$. The bit-distortion $D^p[n]$ also depends on a value $v^p[n]$, which is defined as the normalized difference between the magnitude of sample $s[n]$ and the largest quantization threshold in the previous bit-plane which was not larger than the magnitude [9]. This is just the normalized representation of the sample $s[n]$ at the bit-plane p and below. Because the equations given in the standard are for normalized bit-distortion, the normalized representation is used. That is, for a sample whose 8-bit binary representation is 00010011_2, for bit-plane $p = 4$, $v^p[n] = 1.0011_2 = 1.1875_{10}$. $v^p[n]$ is calculated by

$$v^p[n] = 2^{-p}|s[n]| - 2\left\lfloor \frac{2^{-p}|s[n]|}{2} \right\rfloor \tag{7.5}$$

As seen in Equations 7.3 and 7.4, the bit-distortions are weighted by the energy gain factor calculated from the wavelet basis functions. Any gain that is applied to the signal by the inverse wavelet transform is accounted for by this weight factor when calculating distortion introduced to the decoded image by removing the selected bit. The 2^{2p} accounts for the fact that the distortion is for a single bit at level p. It can be shown that $[v^p[n]^2 - (v^p[n] - 1.5)^2]$ and $[(v^p[n]^2 - 1)^2 - (v^p[n] - 0.5 - v)^2]$ in Equations

7.3 and 7.4 calculate the squared difference between keeping the specified bit and removing it, depending upon which coding pass coded it.

Thus, with the bit level p, the wavelet gain G_b, the step size from quantization Δ, and the difference between keeping a bit and removing it all accounted for, a correct bit-distortion is calculated and recorded for every bit coded in Tier 1. The majority of a coding pass's distortion accumulation comes from the wavelet gain factor and the quantization step size as well as the number of bits coded by that pass [18]. This makes the calculation of the wavelet gain crucial to achieve correct distortion calculations.

7.2 Rate

Size of a codeblock's bit stream is dependent upon the output of the MQ Coder. For every coding pass of a codeblock, some amount of bytes were output by the MQ Coder and added onto the end of the codeblock's bit stream. These rates, along with the state of the MQ Coder at the end of each coding pass, c, are recorded during Tier 1. In determining the rate for a specific truncation point n_c within the set of truncation points z_i, for codeblock i, the bytes included by the *flush* method described in Section 6.3.3 must be accounted for. Given the MQ Coder's state at the end of the coding pass associated with truncation point n_c, a fake flush can be performed which only determines the amount of bytes output by the flush, and that amount can be added to the final rate.

7.3 Rate-Distortion Slopes

Once a rate and distortion is available for every coding pass, a set of rate-distortion slopes, S_i, are calculated by taking the change in distortion between the current truncation point and the previous one divided by the change in rate between the current truncation point and the previous one. This is shown in Equation 7.6.

$$S_i^{n_c} = \frac{D_i^{(n_c)} - D_i^{(n_{c-1})}}{R_i^{(n_c)} - R_i^{(n_{c-1})}} \tag{7.6}$$

The graph of these slopes is a rate-distortion slope graph for a single codeblock, and each point on the graph represents a truncation point, which has an associated rate and distortion. These rate-distortion slopes are used to find the optimal reductions in rate per increases in distortion when truncating a codeblock. Given some value of λ, the set of truncation points that minimizes Equation 7.2 and yields a desired rate, R, can be found by recursively searching through the truncation points for each codeblock and determining how much each one contributes to the overall rate of the image. The sum of all these rates is the final rate, R.

In order to ensure truncation produces an optimal distortion per rate, a constraint is set on the set of truncation points for a single codeblock to force the rate-distortion slopes to form a *convex hull.*

7.3.1 Feasible Truncation Points

Figure 7.1 shows an example of a graph of a single codeblock's rate-distortion slopes and truncation points. As the truncation points' rates decrease, their distortion increases. It can be seen that some truncation points result in a small rate reduction and a much larger distortion increase, where the very next truncation point may

result in a much larger rate reduction for a small distortion increase. The best case truncation point is one that whose rate is significantly smaller than the one before it and whose distortion is as close to the one before it. This means that the truncation point's slope would be close to zero.

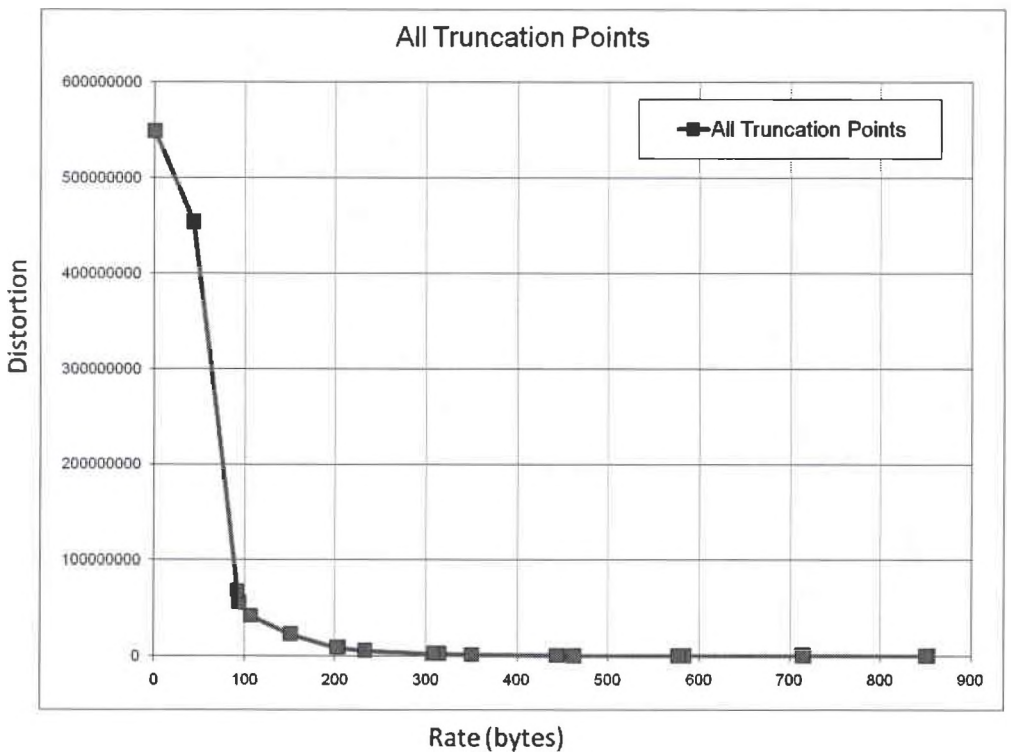

Figure 7.1: All truncation points for a codeblock in an image.

If the slopes are followed "up the graph" (meaning starting from the largest rate and working toward a smaller rate), some truncation points lie inside of a strictly *convex hull* formed by the slopes. Figure 7.2 shows an example of a rate-distortion slope graph that labels the invalid truncation points to be removed, forming a strictly convex hull. Rate-distortion slopes that form a strictly convex hull are desired because

for every truncation point selected on the graph, the rate reduction per distortion increase is better than the one before it. This places a limit on the number of valid—or feasible—truncation points, which significantly simplifies finding the truncation point that is optimal for a specific codeblock. The invalid truncation points are removed from the set of possible truncation points, and are not included when selecting the optimal truncation points.

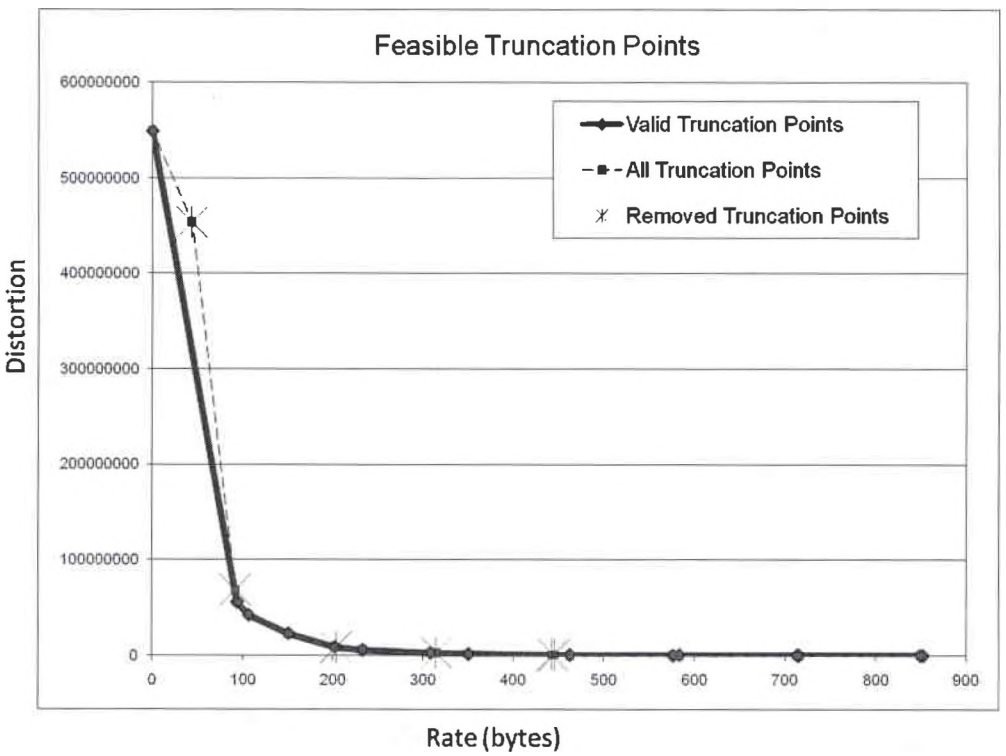

Figure 7.2: Feasible truncation points for a codeblock in an image.

7.4 Optimal Truncation

When truncating a codeblock at a point n_c, the resulting bit stream for that codeblock is now the end of the codeblock. The MQ Coder flush must be performed on that coding pass as if it was the end of the codeblock, which is why the MQ Coder's state was saved for the end of each coding pass during Tier 1. When an optimal truncation point is selected for a codeblock, the MQ Code flush is performed for that coding pass, adding the final bytes of that coding pass onto the bit stream. The rate for that codeblock is reduced and the number of coding passes included in the final codeblock bit stream is recorded so that it can be placed in the JPEG2000 file's header information so that the decoder can properly decode the truncated codeblocks.

In performing the codeblock truncation, the Lagrange method is typically used to find the optimal truncation points and is what is defined in both [9] and [12]. A simple recursive method may be applied, however, but is more processing intensive.

7.4.1 Using the Lagrange Method

In order to find the optimal truncation points used to truncate the codeblock code streams, the λ value selected is used as an inverse codeblock quality parameter. The smaller the λ, the larger the rates of each codeblock can be in Equation 7.2, and thus the larger the final rate, R, will be. After selecting a λ value and finding the optimal truncation point for each codeblock, which minimizes $D_i^{(z_i)} + \lambda R_i^{(z_i)}$, the overall rate, R, will be the summation of all the rates of the truncated codeblocks. If the resulting rate is larger than R_{max}, the λ value can be increased and the algorithm repeated.

In order to increase the speed of the algorithm, a bisection method may be used to halve a working interval defined by a λ^{min} and a λ^{max}, between which the optimal λ resides [12].

7.4.2 Using the Recursive Method

Since the invalid truncation points were removed from the rate-distortion slope graph, each subsequent slope is more negative than the previous. In order to find the first coding pass that should be removed from a codeblock, all the slope values for the final coding pass may be compared, and the one closest to zero (the optimal distortion reduction per rate decrease) can be truncated. This process is repeated until the summation of the rates of each codeblock is less than or equal to the desired rate, R_{max}.

CHAPTER 8

Results

Implementation of the Post-Compression Rate-Distortion algorithm within the EBCOT encoding scheme resulted in a marked improvement in compression ratio versus visual quality of the decoded image compared to the derived quantization method described in 5.1.1. In comparisons against mainstream JPEG2000 encoders, the implementation described in Section 7 matched performance and at some compression ratios outperformed that of the JasPer encoder and the Kakadu encoder.

A variety of images were used in comparison testing. Smaller images (with dimensions in the range of 256x256 to 512x512) commonly used in signal and image processing were taken from the USC Signal and Image Processing Institute's online database, [14]. These are shown in Figure 8.1 below. Larger images are becoming more common for image processing testing as resolutions of standard formats increase, and three images from the Standard Color Image Data, [8], as well as an image commonly used for compression testing given in [12] were used for high resolution image testing and are shown in Figure 8.2. Table 8.1 lists all the images used and their sizes and bit depths.

8.1 PSNR Results Comparisons

Because the PCRD algorithm is based on finding an optimal distortion per a given rate, and distortion is defined as the mean-squared error (MSE) between the

Figure 8.1: Small images used for PCRD algorithm performance comparisons.

decoded wavelet coefficients and the original wavelet coefficients, measuring algorithm performance is typically done using peak signal-to-noise ratio (PSNR). PSNR takes the max pixel value in an image—for 8 bit images, this is 255—and divides it by the root MSE. PSNR, measured in decibels, is then calculated by

$$PSNR = 20 \cdot \log_{10}\left(\frac{255}{\sqrt{MSE}}\right) \tag{8.1}$$

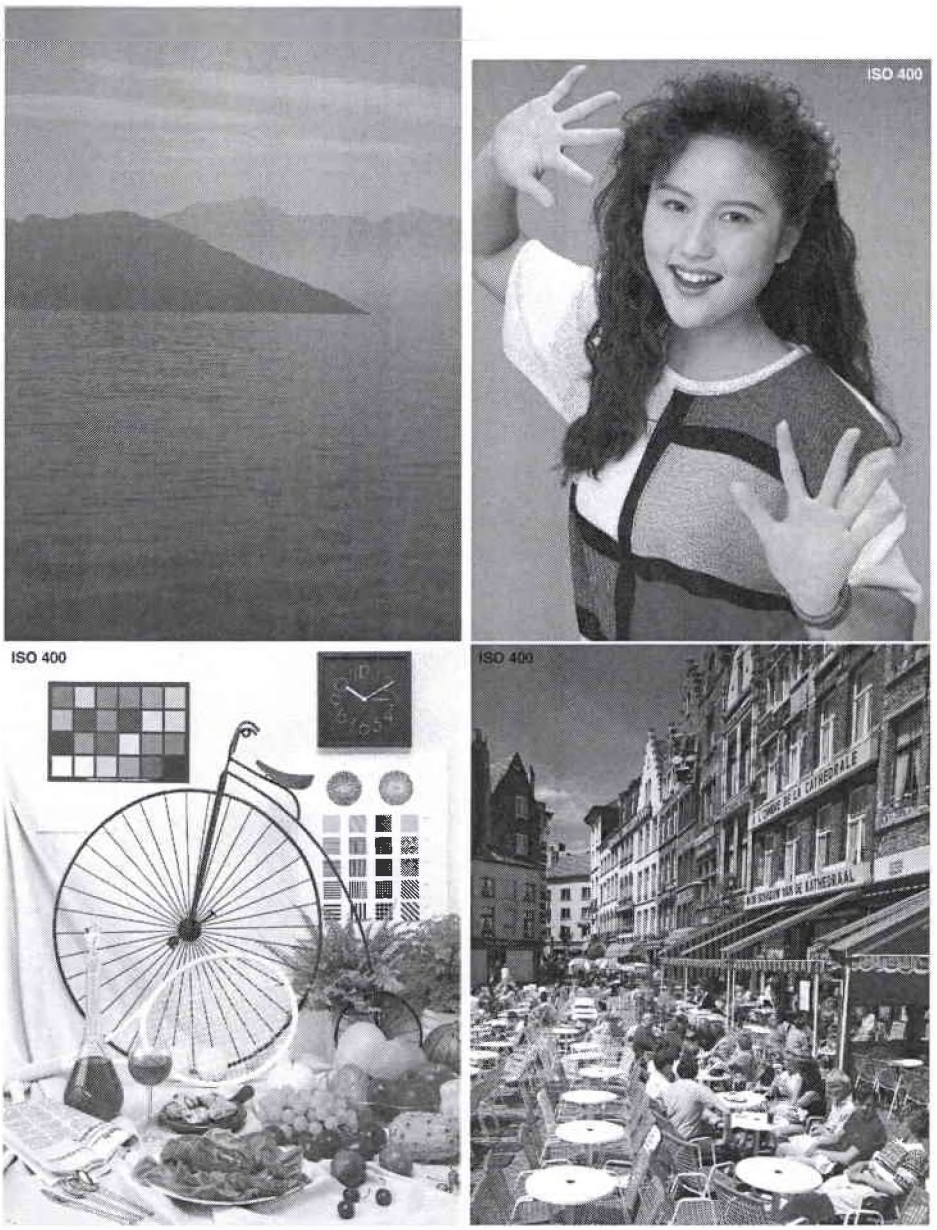

Figure 8.2: Large images used for PCRD algorithm performance comparisons.

The comparison of image quality using PSNR after decoding the image should not be confused with the calculation of distortion during Tier 1, which is part of the encoding process. Distortion measurements used in the PCRD algorithm implemented

Table 8.1: Image Information for Images Used in Results Testing

Image	Width x Height	Bit Depth (bpp)	File Size (bytes)
Cameraman.bmp	256x256	8	65,536
Peppers.bmp	512x512	8	262,144
Baboon.bmp	512x512	8	262,144
Lena.bmp	512x512	8	262,144
Goldhill.bmp	512x512	8	262,144
Barbara.bmp	512x512	8	262,144
Boat.bmp	512x512	8	262,144
Water.bmp	1465x1999	8	2,928,535
Woman.bmp	2048x2560	8	5,242,880
Bike.bmp	2048x2560	8	5,242,880
Cafe.bmp	2048x2560	8	5,242,880

during encoding represent the squared difference between wavelet coefficients, which is not the same as squared error of a decoded image and the original. The reason for this is because of the transform done on the data before taking distortion measurements. It should also be noted that minimum mean-squared error and maximum PSNR does not necessarily mean optimal visual quality. Mean-squared error is a standard means by which to measure difference of a decoded image from the original, but it is possible for an image that has a larger MSE to appear visibly closer to the original, and much of this has to do with the human visual system and its sensitivity to certain frequency ranges. The human eye's contrast sensitivity function (CSF) is well known and can be used to weight certain spatial frequencies higher when calculating distortion [12]. JPEG2000 provides a set of visual weights to modify bit-distortion calculations in order to achieve a better visual quality image, while possibly lowering the resulting peak signal-to-noise ratio [9].

8.2 Encoder Comparisons

Tests were performed by compressing the images listed in Table 8.1 with the UD-JPEG2000 Encoder using quantization only and using optimal truncation with the proposed PCRD algorithm, with the Kakadu encoder, and with the JasPer encoder. Images were compressed multiple times each at increasing compression ratios to show performance at multiple bitrates. Each compression engine was set to use the following JPEG2000 parameters: 32x32 codeblocks, 5 wavelet levels, and the 9/7 wavelet for lossy compression. The graphs produced display PSNR versus compressed file size in bytes (rate).

Because the results of the following tests are very similar, the graphs and tables are given to show uniform performance across a variety of images. Explanation and a summary of these results is given for the Peppers image in Figure 8.3 and its table, Table 8.2, and applies to Figures 8.4 through 8.13 and Tables 8.3 through 8.12.

It should be noted that the JasPer encoder would not produce a compressed image with a rate of 4 bits per pixel and higher. In order to achieve this, the UDJPEG2000 Encoder scales the quantization steps down by 2, increasing the resulting coefficients from quantization, thus preserving more precision bits going into the Tier 1 process. It would appear that the Kakadu encoder also does this, as it achieves PSNR values on par with the UDJPEG2000 Encoder for a rate of 4 bits per pixel and above.

8.2.1 Peppers

Figure 8.3 shows PSNR results for the four encoding methods used on the Peppers image. It can be seen from the graph that the proposed PCRD algorithm with optimal truncation outperforms that of the reference encoders as well as quantization at high

bitrates. This performance difference is minimized as compression ratio increases and the resulting file size gets smaller. At very low file sizes, the proposed PCRD algorithm performs very closely to that of the reference encoders JasPer and Kakadu. Table 8.2 lists each file size and PSNR, showing how little the difference in PSNR is at the higher compression ratios. Bolded values in the tables show results when one encoder produced a smaller rate than the others while maintaining a higher PSNR. As mentioned above, the results for the rest of the images in Figures 8.4 through 8.13 and Tables 8.3 through 8.12 follow those shown for the Peppers image in Figure 8.3 and Table 8.2.

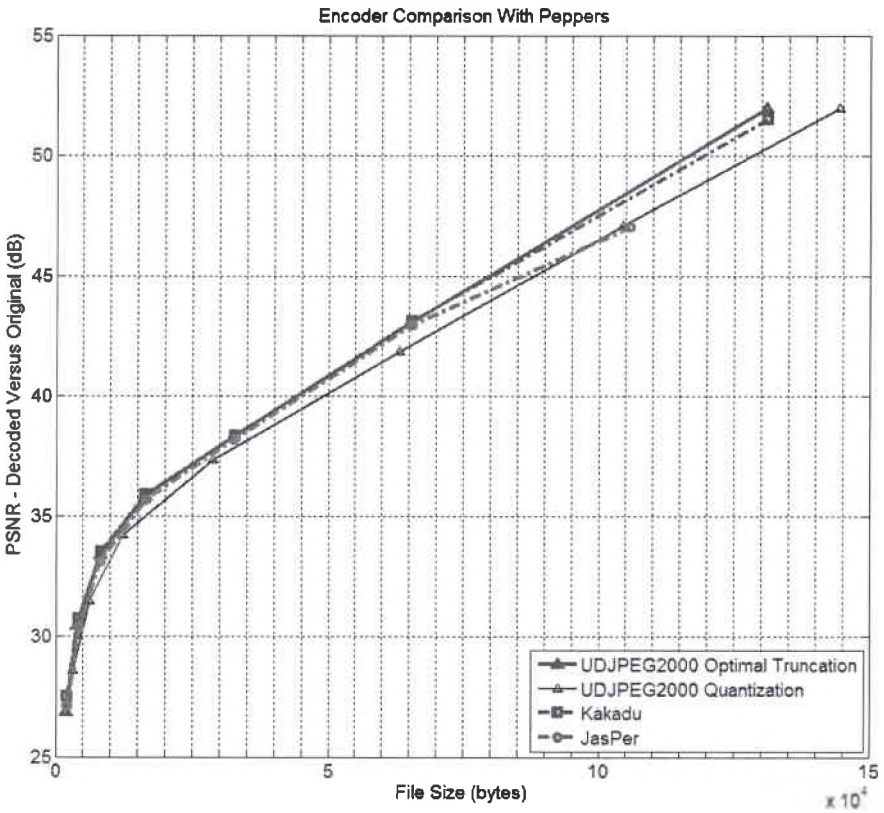

Figure 8.3: Encoder comparison results for the Peppers image.

Table 8.2: File Sizes and PSNR Values for the Encoded Peppers Image

UD-Quant		**UD-PCRD**		**Kakadu**		**JasPer**	
Size	PSNR	Size	PSNR	Size	PSNR	Size	PSNR
144,278	51.944	130,897	51.954	131,022	51.494	105,713	47.036
104,399	47.088	65,231	43.024	65,516	43.099	65,513	42.961
63,179	41.834	32,544	38.317	32,810	38.365	32,758	38.212
28,809	37.331	16,350	35.877	16,429	35.906	16,495	35.704
12,213	34.235	8,079	33.425	8,269	33.525	8,121	33.117
6,163	31.496	3,937	30.47	4,164	30.731	4,160	30.366
3,169	28.59	1,839	26.854	2,122	27.52	2,084	27.093

8.2.2 Baboon

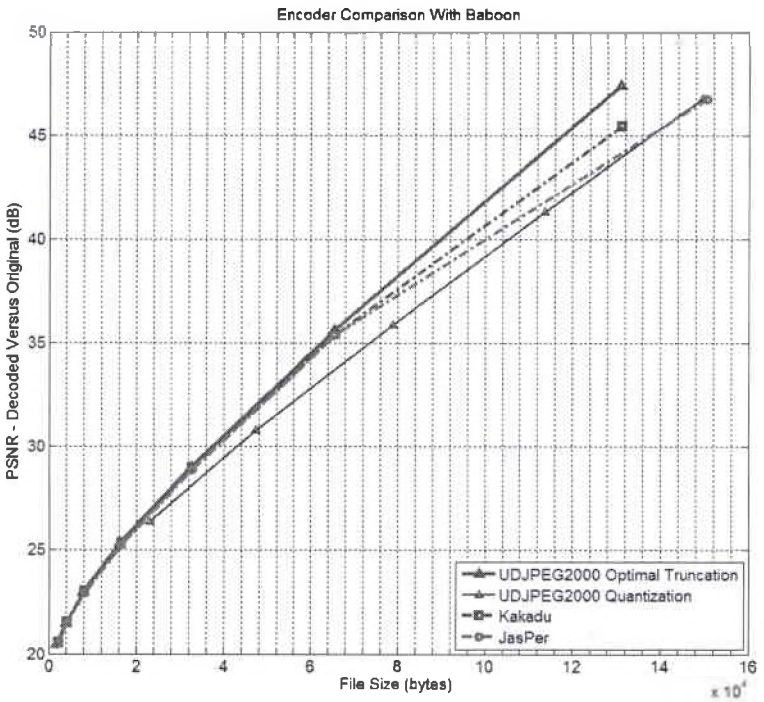

Figure 8.4: Encoder comparison results for the Baboon image.

Table 8.3: File Sizes and PSNR Values for the Encoded Baboon Image

UD-Quant		**UD-PCRD**		**Kakadu**		**JasPer**	
Size	PSNR	Size	PSNR	Size	PSNR	Size	PSNR
149,516	46.799	**130,957**	**47.399**	131,157	45.444	150,467	46.736
113,710	41.299	**65,451**	**35.611**	65,605	35.42	65,519	35.357
78,851	35.867	**32,559**	**29.006**	32,850	28.997	32,760	28.875
47,393	30.803	16,130	25.342	16,457	25.35	16,497	25.26
23,024	26.357	7,830	22.959	8,177	23.004	8,101	22.915
–	–	3,949	21.458	4,119	21.524	4,175	21.485
–	–	1,879	20.462	2,136	20.564	2,088	20.474

8.2.3 Lena

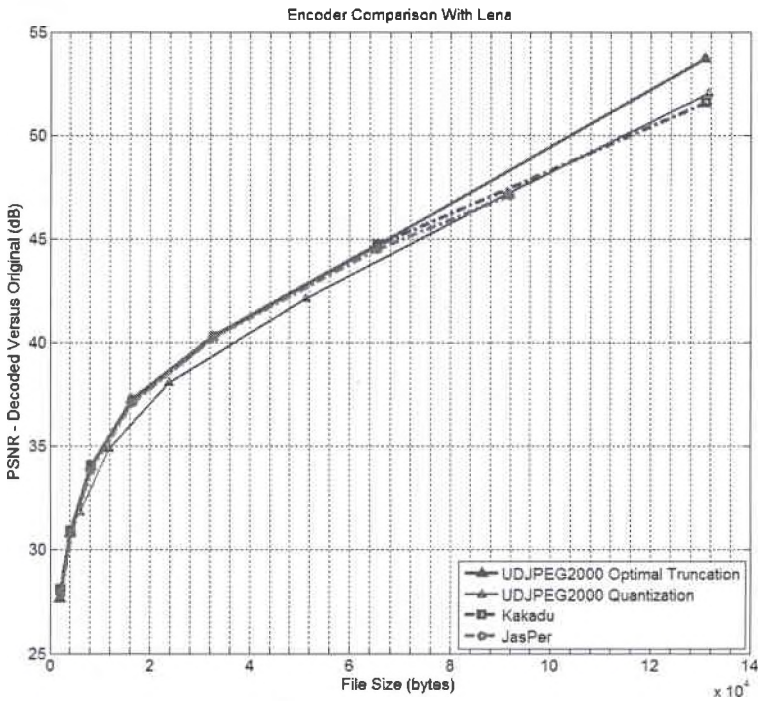

Figure 8.5: Encoder comparison results for the Lena image.

Table 8.4: File Sizes and PSNR Values for the Encoded Lena Image

UD-Quant		**UD-PCRD**		**Kakadu**		**JasPer**	
Size	PSNR	Size	PSNR	Size	PSNR	Size	PSNR
131,623	52.015	130,825	53.723	131,083	51.584	91,978	47.103
91,041	47.143	**65,437**	**44.724**	65,557	44.705	65,523	44.511
51,298	42.135	**32,605**	**40.302**	32,839	40.297	32,732	40.228
23,846	38.084	**16,335**	**37.239**	16,463	37.192	16,513	37.108
11,772	34.902	8,069	34.041	8,274	34.051	8,117	33.804
5,918	31.797	3,948	30.782	4,148	30.909	4,194	30.808
–	–	1,848	27.63	2,129	28.079	2,095	27.851

8.2.4 Cameraman

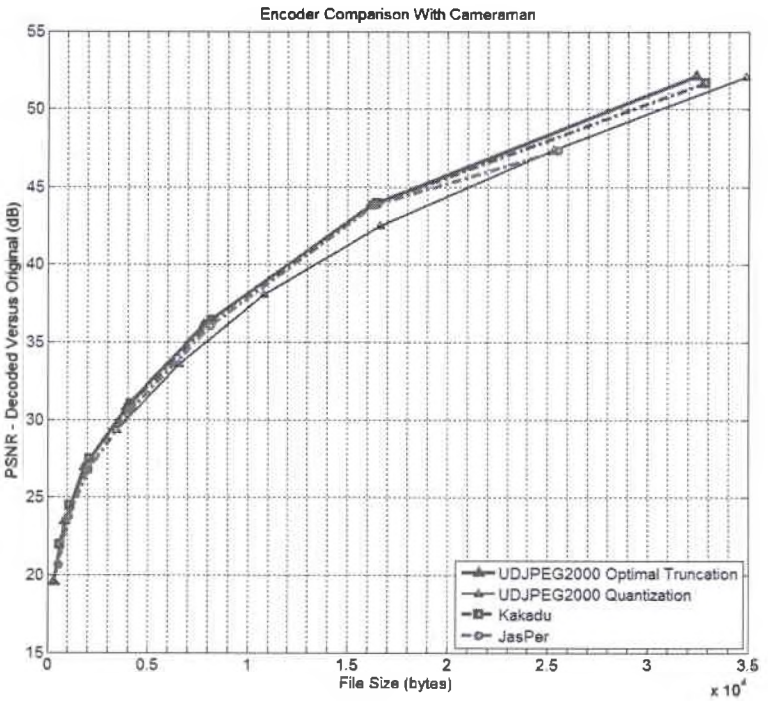

Figure 8.6: Encoder comparison results for the Cameraman image.

Table 8.5: File Sizes and PSNR Values for the Encoded Cameraman Image

UD-Quant		**UD-PCRD**		**Kakadu**		**JasPer**	
Size	PSNR	Size	PSNR	Size	PSNR	Size	PSNR
44,428	54.29	32,406	54.29	32,831	51.7	25,506	47.337
34,852	52.089	**16,137**	**54.29**	16,460	43.932	16,358	43.844
25,258	47.399	**7,835**	**52.143**	8,230	36.412	8,144	36.016
16,604	42.502	**3,938**	**43.807**	4,179	31.028	4,116	30.606
10,827	38.052	**1,879**	**36.125**	2,131	27.445	2,011	26.786
6,589	33.575	**842**	**30.834**	1,100	24.476	1,043	23.778
3,463	29.336	**328**	**26.997**	597	21.966	517	20.666

8.2.5 Gold Hill

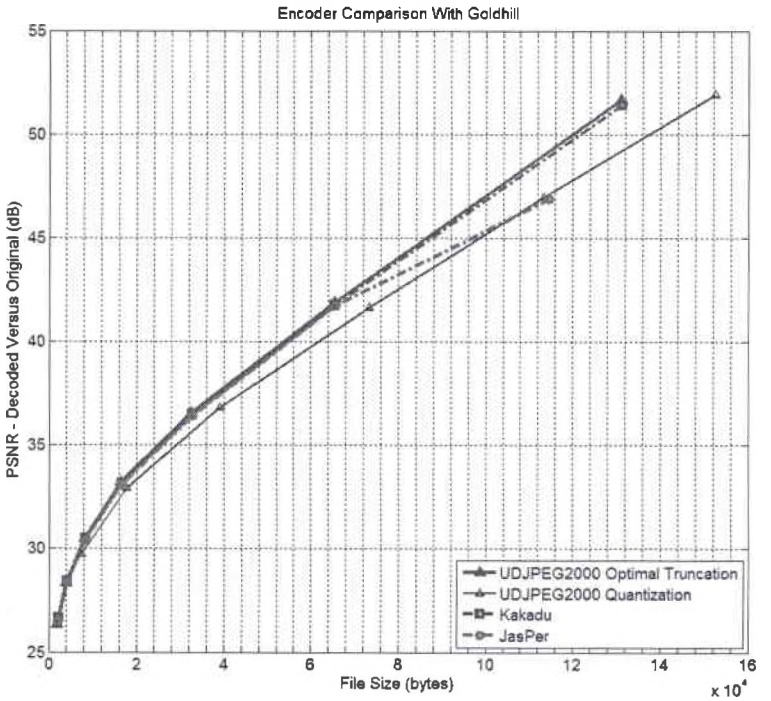

Figure 8.7: Encoder comparison results for the Gold Hill image.

Table 8.6: File Sizes and PSNR Values for the Encoded Gold Hill Image

UD-Quant		**UD-PCRD**		**Kakadu**		**JasPer**	
Size	PSNR	Size	PSNR	Size	PSNR	Size	PSNR
152,499	51.903	130,966	51.652	131,137	51.405	114,602	46.891
113,265	46.94	65,546	41.895	65,534	41.759	65,524	41.711
73,300	41.617	**32,680**	**36.563**	32,823	36.477	32,744	36.394
39,133	36.806	**16,293**	**33.212**	16,448	33.153	16,491	33.078
17,762	32.913	8,082	30.521	8,276	30.523	8,119	30.389
7,336	29.744	3,925	28.321	4,154	28.426	4,182	28.36
–	–	1,846	26.314	2,149	26.67	2,080	26.48

8.2.6 Barbara

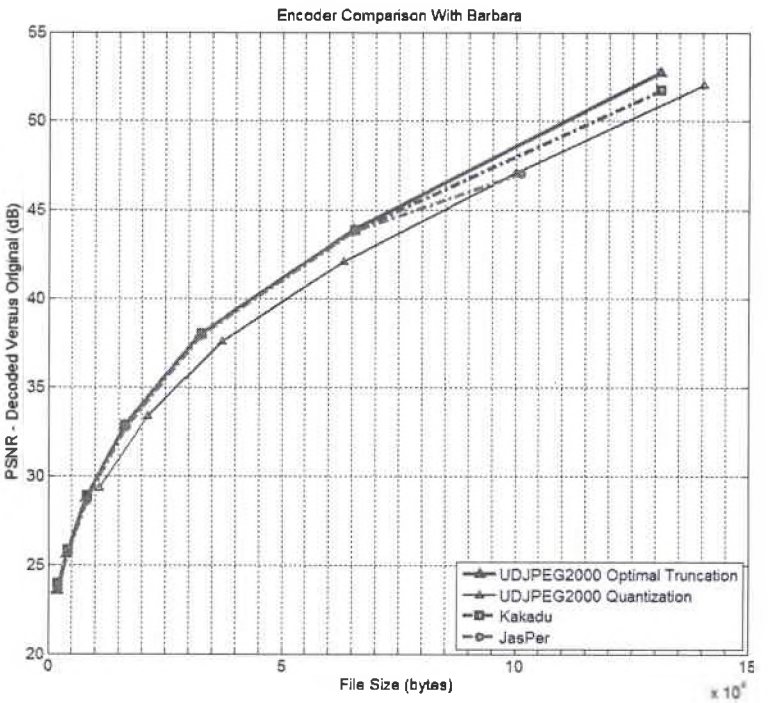

Figure 8.8: Encoder comparison results for the Barbara image.

Table 8.7: File Sizes and PSNR Values for the Encoded Barbara Image

UD-Quant		**UD-PCRD**		**Kakadu**		**JasPer**	
Size	PSNR	Size	PSNR	Size	PSNR	Size	PSNR
140,183	51.992	130,896	52.738	131,114	51.746	101,217	47.038
100,167	47.093	**65,366**	**43.903**	65,594	43.866	65,504	43.817
63,212	42.087	32,612	38.015	32,822	38.048	32,750	37.947
37,172	37.616	16,287	32.85	16,408	32.855	16,514	32.769
21,209	33.393	8,033	28.751	8,275	28.887	8,123	28.609
10,758	29.354	3,877	25.609	4,147	25.838	4,194	25.738
—	—	1,843	23.568	2,143	23.945	2,028	23.668

8.2.7 Boat

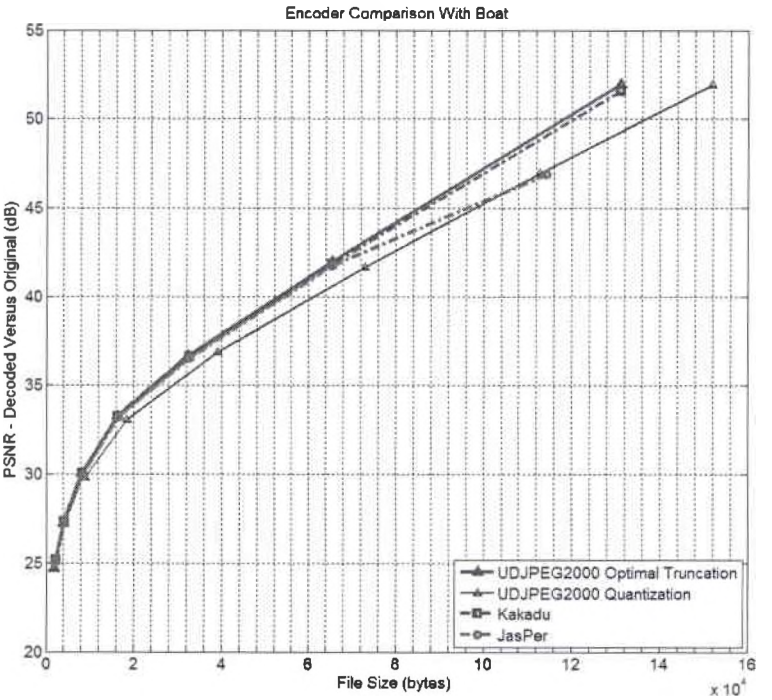

Figure 8.9: Encoder comparison results for the Boat image.

Table 8.8: File Sizes and PSNR Values for the Encoded Boat Image

UD-Quant		**UD-PCRD**		**Kakadu**		**JasPer**	
Size	PSNR	Size	PSNR	Size	PSNR	Size	PSNR
151,936	51.94	131,075	51.947	131,126	51.572	114,263	46.921
112,692	46.973	65,542	41.999	65,560	41.832	65,494	41.755
72,890	41.659	**32,478**	**36.674**	32,822	36.649	32,733	36.56
39,042	36.857	**16,307**	**33.307**	16,387	33.249	16,515	33.197
18,389	33.065	8,041	30.048	8,266	30.066	8,110	29.854
8,889	29.8	3,947	27.255	4,137	27.318	4,189	27.22
–	–	1,764	24.718	2,129	25.172	2,095	25.019

8.2.8 Water

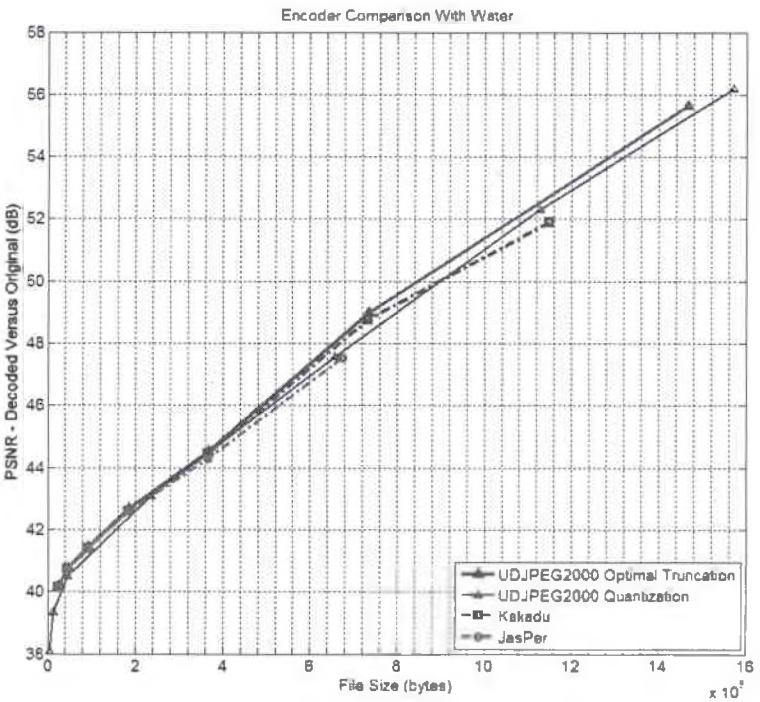

Figure 8.10: Encoder comparison results for the Water image.

Table 8.9: File Sizes and PSNR Values for the Encoded Water Image

UD-Quant		**UD-PCRD**		**Kakadu**		**JasPer**	
Size	PSNR	Size	PSNR	Size	PSNR	Size	PSNR
1,571,125	56.181	1,466,611	55.629	1,147,244	51.913	674,364	47.541
1,127,004	52.31	734,270	48.984	732,198	48.757	674,364	47.541
656,267	47.565	367,447	44.535	366,116	44.453	365,341	44.302
236,080	43.097	184,149	42.679	183,089	42.599	184,484	42.607
44,344	40.501	92,125	41.445	91,535	41.41	90,740	41.393
12,644	39.331	46,167	40.757	45,768	40.732	46,846	40.747
5,282	38.063	**22,980**	**40.159**	23,142	40.147	23,409	40.146

8.2.9 Woman

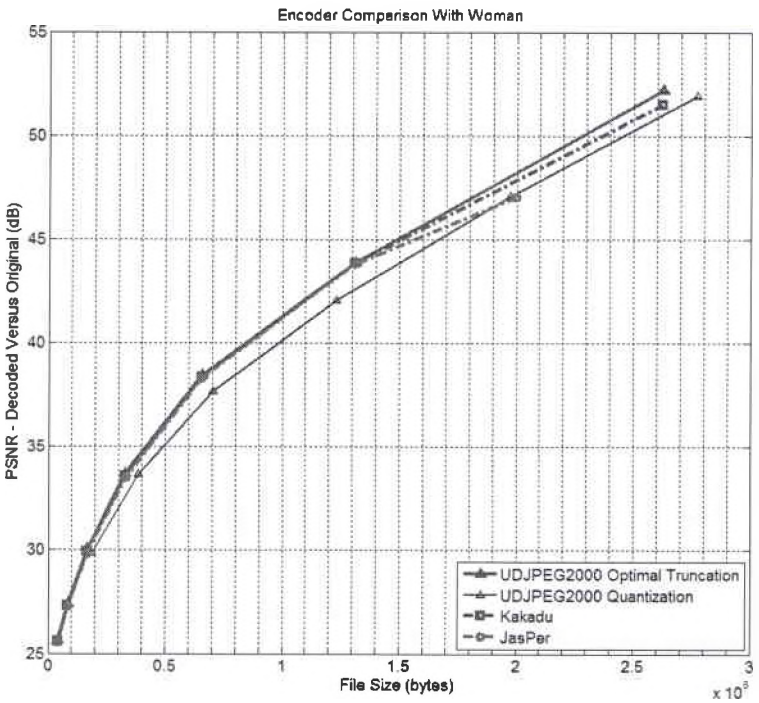

Figure 8.11: Encoder comparison results for the Woman image.

Table 8.10: File Sizes and PSNR Values for the Encoded Woman Image

UD-Quant		**UD-PCRD**		**Kakadu**		**JasPer**	
Size	PSNR	Size	PSNR	Size	PSNR	Size	PSNR
2,769,554	51.928	2,625,791	52.187	2,621,516	51.493	1,999,661	47.02
1,973,844	47.071	1,314,648	43.899	1,310,718	43.849	1,309,673	43.818
1,230,933	42.05	658,132	38.432	655,437	38.343	655,275	38.275
702,950	37.672	329,746	33.677	327,759	33.555	330,299	33.51
386,853	33.649	165,081	30.044	163,908	29.95	162,512	29.823
185,218	29.836	82,616	27.386	81,997	27.325	83,869	27.317
—	—	**41,112**	**25.619**	41,310	25.593	41,937	25.569

8.2.10 Bike

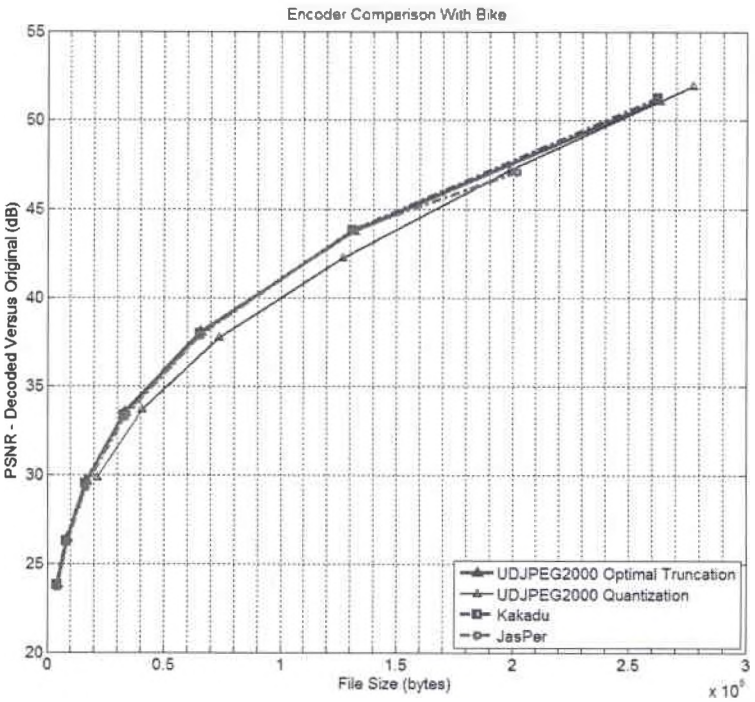

Figure 8.12: Encoder comparison results for the Bike image.

Table 8.11: File Sizes and PSNR Values for the Encoded Bike Image

UD-Quant		**UD-PCRD**		**Kakadu**		**JasPer**	
Size	PSNR	Size	PSNR	Size	PSNR	Size	PSNR
2,772,755	51.908	2,625,777	51.111	2,621,511	51.243	2,015,244	47.112
1,987,833	47.19	1,314,652	43.741	1,310,711	43.779	**1,310,341**	**43.816**
1,267,638	42.207	658,716	38.065	655,435	37.963	655,353	37.877
736,357	37.722	330,332	33.557	327,765	33.394	330,278	33.29
406,113	33.666	165,628	29.686	163,909	29.523	162,508	29.312
211,548	29.864	82,867	26.427	81,995	26.3	83,861	26.229
–	–	41,446	23.839	41,370	23.776	41,918	23.679

8.2.11 Cafe

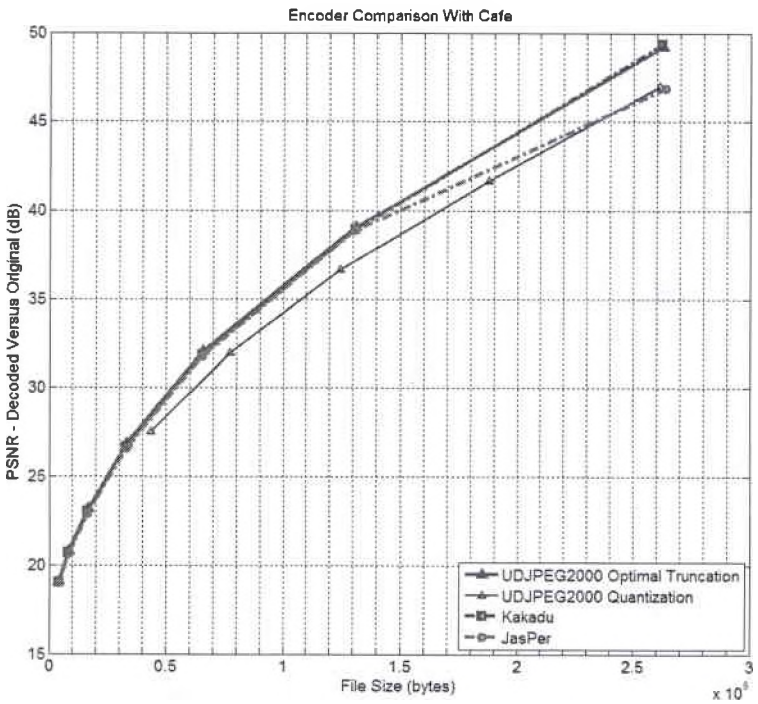

Figure 8.13: Encoder comparison results for the Cafe image.

Table 8.12: File Sizes and PSNR Values for the Encoded Cafe Image

UD-Quant		**UD-PCRD**		**Kakadu**		**JasPer**	
Size	PSNR	Size	PSNR	Size	PSNR	Size	PSNR
2,610,516	46.97	2,625,855	49.207	2,621,500	49.341	2,637,100	46.884
1,875,294	41.691	1,314,565	39.086	1,310,783	38.947	1,310,600	38.879
1,245,804	36.661	658,542	32.093	655,414	31.898	655,337	31.784
773,164	31.957	329,834	26.871	327,722	26.671	330,283	26.597
435,931	27.547	165,202	23.201	163,912	23.065	162,529	22.926
–	–	82,568	20.794	81,991	20.705	83,821	20.681
–	–	**41,229**	**19.072**	41,360	19.025	41,928	18.991

8.3 PCRD Algorithm Processing Intensity Versus Quantization

In order to find how much complexity and overhead including optimal truncation with the PCRD algorithm to the UDJPEG2000 Encoder added, the number of coding passes performed in Tier 1 is recorded when using optimal truncation and compared to that number when using quantization. Uniform quantization effectively removes entire bitplanes from codeblocks, which then do not have to be coded in Tier 1. As Tier 1 takes 70% to 80% of the encoding processing time, quantization can significantly speed up the overall process [7]. Tables 8.13 through 8.23 display the number of coding passes processed when using optimal truncation and when using quantization for the images listed in Table 8.1 for multiple compressed rates. These results show that, when using optimal truncation, the number of coding passes processed is always the total number of coding passes needed to process every bit plane. This is straightforward, as optimal truncation encodes all the bit planes, and then truncates the resulting bitstreams after coding. Quantization, on the other hand, removes data prior to the Tier 1 process, which then only codes the bit planes that have not been removed.

Since adding optimal truncation adds a significant amount of time to the encoding process—as indicated by the increased number of coding passes processed in Tier 1—some work may need to be done to decrease this processing cost while keeping the benefits of optimal truncation. Some ideas that may be included in future versions of the UDJPEG2000 Encoder are suggested in the following section.

Table 8.13: Number of Coding Passes Processed for the Peppers Image

UD-Quant		UD-PCRD	
Size	Coding Passes	Size	Coding Passes
144278	1321	130897	1531
104399	1111	65231	1531
63179	901	32544	1531
28809	691	16350	1531
12213	481	8079	1531
6163	297	3937	1531
3169	166	1839	1531

Table 8.14: Number of Coding Passes Processed for the Baboon Image

UD-Quant		UD-PCRD	
Size	Coding Passes	Size	Coding Passes
144278	1321	130897	1531
104399	1111	65231	1531
63179	901	32544	1531
28809	691	16350	1531
12213	481	8079	1531
6163	297	3937	1531
3169	166	1839	1531

Table 8.15: Number of Coding Passes Processed for the Lena Image

UD-Quant		UD-PCRD	
Size	Coding Passes	Size	Coding Passes
131623	1225	130825	1435
91041	1015	65437	1435
51298	805	32605	1435
23846	595	16335	1435
11772	395	8069	1435
5918	234	3948	1435
–	–	1848	1435

Table 8.16: Number of Coding Passes Processed for the Cameraman Image

UD-Quant		UD-PCRD	
Size	Coding Passes	Size	Coding Passes
34852	595	32406	670
25258	520	16137	670
16604	445	7835	670
10827	370	3938	670
6589	295	1879	670
3463	220	842	670
–	–	328	670

Table 8.17: Number of Coding Passes Processed for the Gold Hill Image

UD-Quant		UD-PCRD	
Size	Coding Passes	Size	Coding Passes
152499	1249	130966	1459
113265	1039	65546	1459
73300	829	32680	1459
39133	619	16293	1459
17762	415	8082	1459
7336	246	3925	1459
–	–	1846	1459

Table 8.18: Number of Coding Passes Processed for the Barbara Image

UD-Quant		UD-PCRD	
Size	Coding Passes	Size	Coding Passes
140183	1276	130896	1486
100167	1066	65366	1486
63212	856	32612	1486
37172	646	16287	1486
21209	448	8033	1486
10758	288	3877	1486
–	–	1843	1486

Table 8.19: Number of Coding Passes Processed for the Boat Image

UD-Quant		UD-PCRD	
Size	Coding Passes	Size	Coding Passes
151936	1267	131075	1477
112692	1057	65542	1477
72890	847	32478	1477
39042	637	16307	1477
18389	433	8041	1477
8889	272	3947	1477
–	–	1764	1477

Table 8.20: Number of Coding Passes Processed for the Water Image

UD-Quant		UD-PCRD	
Size	Coding Passes	Size	Coding Passes
1571125	10318	1466611	10318
1127004	8002	734270	10318
656267	5686	367447	10318
236080	3370	184149	10318
44344	1490	92125	10318
12644	518	46167	10318
5282	161	22980	10318

Table 8.21: Number of Coding Passes Processed for the Woman Image

UD-Quant		UD-PCRD	
Size	Coding Passes	Size	Coding Passes
2769554	21266	2625791	25124
1973844	17408	1314648	25124
1230933	13550	658132	25124
702950	9818	329746	25124
386853	6539	165081	25124
185218	3839	82616	25124
–	–	41112	25124

Table 8.22: Number of Coding Passes Processed for the Bike Image

UD-Quant		UD-PCRD	
Size	Coding Passes	Size	Coding Passes
2772755	24323	2625777	28181
1987833	20465	1314652	28181
1267638	16607	658716	28181
736357	12763	330332	28181
406113	9054	165628	28181
211548	5733	82867	28181
–	–	41446	28181

Table 8.23: Number of Coding Passes Processed for the Cafe Image

UD-Quant		UD-PCRD	
Size	Coding Passes	Size	Coding Passes
2610516	22016	2625855	29732
1875294	18158	1314565	29732
1245804	14314	658542	29732
773164	10575	329834	29732
435931	7025	165202	29732
–	–	82568	29732
–	–	41229	29732

CHAPTER 9

Conclusions and Future Work

This thesis presents the theory behind the PCRD-Opt algorithm and shows the results of integrating the PCRD-Opt algorithm into the UDJPEG2000 Encoder. Implementation of the proposed PCRD algorithm resulted in a significant improvement over compression using uniform derived quantization. When compared to available reference encoders, the UDJPEG2000 Encoder outperforms JasPer in general and produces results very similar to that of Kakadu for all bit rates.

Some possible future work regarding the UDJPEG2000 Encoder and the PCRD-Opt algorithm specifically is:

- Implementation of the PCRD algorithm adds encoding time, which may not be acceptable in some real-time applications. In order to decrease the amount of time spent in the Tier 1 process, coding passes may be removed prior by quantization.

- Using quantization as the sole remover of data, however, negates the purpose of JPEG2000's optimal truncation and post-compression rate-distortion implementation. It may be possible to combine these with a controller that would monitor how many bit planes were removed by the optimal truncation algorithm in the previous image, and then remove close to that many bit planes

using quantization for the next image. Tier 1 time would be decreased, and the optimal truncation algorithm could still provide the same benefit it does without quantization.

- Along with a controller, the process of finding the rates and distortions and the optimal truncation points is not a trivial task. The recursive method outlined in Section 7.4.2 may take a maximum of C^2 comparisons, where C is the total number of coding passes for an entire image. The Lagrange method described in Section 7.4.1 may be faster if implemented, but depends upon the constraints set for λ Some work could be done to find an optimal solution for this.

- A good method for improving the speed of optimal truncation is an FPGA implementation of the algorithm. Specifically targeted hardware can easily outperform a software implementation, as software must be allowed CPU time by the operating system, which may swap the process out of the run state many times before it finishes. Hardware can run in parallel and can always be doing the calculations needed for optimal truncation. Parts of the algorithm that lend themselves to an FPGA implementation are discussed in [12] and could be included in future development plans for the UDJPEG2000 Encoder.

- Another possible method to speed up encoding time is to implement the wavelet transform and quantization using integer computation. Floating point multiplies take a significant amount more time than integer multiplies and are much more difficult to implement in a hardware solution. By scaling up the floating point values and then truncating them to integer values, some precision is able

to be saved while still performing integer computations. This could provide a significant speed increase for a very small sacrifice in quality.

BIBLIOGRAPHY

[1] Acharya, Tinku and Ping-Sing Tsai. *JPEG2000: Standard For Image Compression. Concepts, Algorithms, and VLSI Architectures.* John Wiley and Sons, Inc., 2005.

[2] Arps, Ronald B. and Truong, Thomas K. *Comparison of International Standards for Lossless Image Compression*, pages 113–123. Morgan Kaugmann Publishers Inc., San Francisco, CA, USA, 2001.

[3] Balster, Eric J. *Video Compression and Rate Control Methods Based on the Wavelet Transform.* PhD thesis, The Ohio State University, 2004.

[4] Christopoulos, C. A. and Ebrahimi, T. and Skodras, A. N. JPEG2000: The New Still Picture Compression Standard. In *MULTIMEDIA '00: Proceedings of the 2000 ACM Workshops on Multimedia*, pages 45–49. ACM, 2000.

[5] Flaherty, M. A Study of the Design and Real-time Implementation of a Semi-Generic, Integer-to-Integer Discrete Wavelet Transform. Master's thesis, University of Dayton, 2006.

[6] Frazer, G. W., Fournier, R. A., Trofymow, J. A., Hall, R. J. A Comparison of Digital and Film Fisheye Photography for Analysis of Forest Canopy Structure and Gap Light Transmission.

[7] Hogrebe, L. A Parallel Architecture of JPEG2000 Tier I Encoding for FPGA Implementation. Master's thesis, University of Dayton, 2009.

[8] International Organization for Standardization, Geneva, Switzerland. Graphic Technology - Prepress Digital Data Exchange - Part 2: XYZ/sRGB Encoded Standard Colour Image Data (XYZ/SCID). Technical report, ISO/IEC, 2004.

[9] ISO/IEC 15444-1:2004(E). *"JPEG 2000 Image Coding System: Core Coding System"*, September 2004.

[10] Mundy, D. The Study and HDL Implementation of the JPEG2000 MQ Coder". Master's thesis, University of Dayton, 2007.

[11] Sweldons, W. The Lifting Scheme: A New Philosophy in Biorthogonal Wavelet Constructions. In *Proc. SPIE-Wavelet Applications in Signal and Image Processing III*, volume 2569, 1995.

[12] Taubman, D. and M. W. Marcellin. *JPEG2000. Image Compression Fundamentals, Standards, and Practice.* Kluwer Academic Publishers, 2002.

[13] Vicen, N. A Resolution Based Bit-Sream Parser for the JPEG-2000 Encoding Standard. Master's thesis, University of Dayton, 2009.

[14] Weber, Allan. USC Signal and Image Processing Institute Database. `http://sipi.usc.edu/database/`.

[15] Weise, Marcus and Diana Weynand. *How Video Works.* Focal Press, second edition, 2007.

[16] Wiggins, R. H., Davidson, H. C., Harnsberger, H. R., Lauman, J. R. Image File Formats: Past, Present, and Future. *Radiographics*, 21:789–798, 2001.

[17] Yeung, Y. M., O. C. Au, and A. Chang. An Efficient Optimal Rate Control Scheme for JPEG2000 Image Coding. In *Proc. 2003 International Conference on Image Processing*, volume 3, pages III–761–4 vol.2, September 2003.

[18] Zhu, Yue-xin, Nan-ning Zheng, Jing Zhang, and Zong-ze Wu. Approximate Treatment for Calculation of the Rate-Distortion Slope in EBCOT. 2005.