

2009

A resolution based bit-stream parser for the JPEG-2000 encoding standard

Nicholas P. Vicen
University of Dayton

Follow this and additional works at: https://ecommons.udayton.edu/graduate_theses

Recommended Citation

Vicen, Nicholas P., "A resolution based bit-stream parser for the JPEG-2000 encoding standard" (2009).
Graduate Theses and Dissertations. 6138.
https://ecommons.udayton.edu/graduate_theses/6138

This Thesis is brought to you for free and open access by the Theses and Dissertations at eCommons. It has been accepted for inclusion in Graduate Theses and Dissertations by an authorized administrator of eCommons. For more information, please contact mschlangen1@udayton.edu, ecommons@udayton.edu.

A RESOLUTION BASED BIT-STREAM PARSER FOR THE
JPEG-2000 ENCODING STANDARD

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

Nicholas P. Vicen

UNIVERSITY OF DAYTON

Dayton, OH

August 2009


A RESOLUTION BASED BIT-STREAM PARSER FOR THE JPEG-2000
ENCODING STANDARD

APPROVED BY:


HFG -

Please Do Not
Remove This Note!

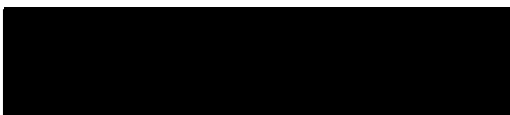
Original




Frank A. Scarpino, Ph.D.
Advisor Committee Chairman
Electrical & Computer Engineering



Eric Balster, Ph.D.
Committee Member
Electrical & Computer Engineering



John Weber, Ph.D.
Committee Member
Electrical & Computer Engineering



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



Joseph E. Saliba, Ph.D., P.E.
Dean, School of Engineering

ABSTRACT

A RESOLUTION BASED BIT-STREAM PARSER FOR THE JPEG-2000 ENCODING STANDARD

Name: Nicholas P. Vicen
University of Dayton, 2009

Advisor: Dr. Frank A. Scarpino

This thesis topic is inspired by the work performed at the Air Force Research Laboratory (AFRL) located in Wright Patterson Air Force Base (WPAFB). Exploring ways to improve compressed image quality led to an investigation of a server/client architecture to display multiple JPEG-2000 images at different levels of resolution. JPEG-2000 is an open standard of compression created by the Joint Pictures Expert Group as a successor to standard JPEG [1]. One of its many features is the ability to embed multiple resolution levels within a single compressed image, referred to as resolution scalability. Resolution scalability allows a controller to tailor the size and quality of the image for the particular viewing area, i.e. the size of the viewing screen/medium. While a JPEG-2000 file contains these embedded resolution levels, a routine is needed to extract each individual level of resolution after the image has been encoded using the JPEG-2000 standard while using the JPEG-2000 Internet Protocol (JPIP). This thesis discusses the development of a bit-stream parser to facilitate

this task.

ACKNOWLEDGEMENTS

To Dr. Frank Scarpino, Dr John Weber, Dr. Eric Balster and David Mundy for giving me the inspiration for this thesis topic.

To Thaddeus Marrara, Jim Hayes, Brett McNerney, and Frank Fradette, my fellow graduate students who helped out in various capacities.

To Kerry Hill and Al Scarpelli for funding my research.

And finally to my parents for supporting me in all my career choices.

TABLE OF CONTENTS

	Page
Approval	ii
Abstract	iii
Acknowledgments	iv
List of Illustrations	vii
Chapters:	
1. Introduction	1
1.1 Why Compression is Necessary	1
1.2 The JPEG-2000 Compression Summary	4
1.3 The JPEG-2000 Compression Standard	7
1.4 The Preprocessing Phase	9
1.4.1 The Tile Partitioner	10
1.4.2 Level Offset	10
1.4.3 Color Transform	11
1.4.4 Discrete Wavelet Transform	12
1.5.2 Quantization	16
1.5 Compression	18
1.5.1 Tier I Coding	19
1.5.1.1 Significance Propagation Pass	22

	Page
1.5.1.2 Magnitude Refinement Pass	23
1.5.1.3 Clean-up Pass	24
1.5.1.4 Sign Coding.	24
1.5.2 Rate Control and Optimal Truncation.	25
1.5.3 Tier II Coding	26
1.5.3.1 Packets, Progressions, and Precincts . . .	27
2. JPIP	30
2.1 Introduction	30
2.2 JPIP Advantages	31
2.3 JPEG-2000 Code-stream Elements	31
2.4 JPIP Server-Client Interaction Overview	32
2.4.1 Data-bin Allocation	34
2.4.2 JPIP Messages and Streams.	34
3. Bit-Stream Resolution Scalable Parser	36
3.1 Introduction.	36
3.2 Tier II Progressions.	36
3.2.1 Progression by Resolution	37
3.3 The Bit-stream Resolution Parser.	38
3.3.1 Headers and Marker Segments	39
3.3.2 Construction of the Code-stream	41
3.3.3 Image and Tile Size (SIZ) Marker Segment	43
3.3.4 Coding Style Default (COD) marker Segment.	44

3.3.5	Resolution boundary and Precinct Calculation	46
3.3.6	Code-block and Precinct Calculation.	47
3.3.7	Start of Packet (SOP) Marker Segment.	48
3.4	UDP Transmission and Precinct Modifications.	50
3.4.1	Precinct Modifications for Resilient Codestreams. .	50
3.4.2	Error Detection during UDP Transmission	51
4.	Results.	52
4.1	Test Parameters.	52
4.2	Progression Order Testing	52
4.3	Resolution Parser Testing	53
4.4	JPIP Integration Testing	58
4.5	Summary of Results	61
5.	Conclusion and Future Works	62
5.1	Conclusion	62
5.2	Future Work	63
	References	64

LIST OF ILLUSTRATIONS

Figure	Page
1.1 MPEG-2 Frame Illustration	4
1.2 Tiling Illustration	6
1.3 JPEG2000 Encoding Algorithm Block Diagram	9
1.4 Single Dimensional Filter Bank	13
1.5 Two Dimensional Filter Bank	14
1.6 Three Levels of decomposition in 2D DWT	15
1.7 First layer of decomposition visual	15
1.8 Second layer of decomposition visual	16
1.9 Graphical Representation of Dead-zone Scalar Quantization	17
1.10 Bit-plane Illustration	19
1.11 EBCOT Scan Pattern.	20
1.12 Tier-1 Overview	22
1.13 Rate-Distortion Information to Quality Layers	28
1.13 JPEG-2000 Packet Construction	29
2.1 JPIP Server/Client Architecture.	32
3.1 Progression by Resolution	38
3.2 Construction of the Code-stream	41
3.3 Construction of tile-part header.	43
3.4 Image and Tile Size (SIZ) Marker Segment	44

3.5	Coding Style Default (COD) Marker Segment	45
3.6	Resolution Boundary Calculation	47
3.5	COD Marker Segment RLCP Modification Verification	53
4.3.1	Resolution Parser at Resolution Level Four.	54
4.3.2	Resolution Parser at Resolution Level Three	55
4.3.3	Resolution Parser at Resolution Level Two.	56
4.3.4	Resolution Parser at Resolution Level One.	57
4.3.5	Resolution Parser at Resolution Level Zero.	58
4.4.1	Full Image at the Lowest Resolution	59
4.4.2	Higher Resolution Image	60
4.4.3	Highest Resolution Image	60
5.1	Images with increasing Resolution Levels	62

Introduction

While exploring ways to display images at the utmost quality while constrained to limited bandwidth requirements led to an investigation of a server/client architecture to display multiple JPEG-2000 images at different levels of resolution. JPEG-2000 is an open standard of compression created by the Joint Pictures Expert Group as a successor to standard JPEG [1]. One of its many features is the ability to embed multiple resolution levels within a single compressed image, referred to as resolution scalability. Resolution scalability allows a controller to tailor the size and quality of the image for the particular viewing area. The controller researched is referred to as the JPEG-2000 Internet Protocol (JPIP). JPIP is a server/client interface that allows for an efficient and effective way of interacting with JPEG-2000 images. JPIP supports resolution scalability but has no inherent way of extracting the resolution level requested. However, a JPEG-2000 file contains these embedded resolution levels. Therefore a method is needed to extract each individual level of resolution after the image has been encoded.

This thesis presents a JPEG-2000 resolution parser for enabling JPIP in a real-time environment. Chapter one lays the foundation of compression and specifically the JPEG-2000 standard. Chapter two explains the JPIP extension to JPEG-2000. Chapter three discusses the design and implementation of the resolution parser. Chapter four shows the results of the design. Finally, chapter five voices the conclusion and future works inspired by this thesis.

CHAPTER 1

JPEG-2000 Standard

The JPEG-2000 standard is widely considered today to be the de-facto compression standard with surveillance imaging systems. The National Imagery Transmission Format Standard (NITFS) has named JPEG-2000 the recommended image compression technique for all branches of the government. It features superior quality at very low bit-rates (high compression rates) compared to standard JPEG as well as allowing random code-stream access which allows multiple resolutions and/or quality layers to be extracted from a single JPEG-2000 compressed image [1]. Other features illustrate why the JPEG-2000 standard is so widely accepted today.

1.1 Why Compression Is Necessary

Compression techniques allow one to retain image quality while utilizing existing communication infrastructure. As data requirements increase to provide enhanced image quality, the existing bandwidth becomes a constraint. Determining how to lower the data requirements for transmission and recreate the enhanced image afterwards without sacrificing response time is the challenge addressed by compression.

There are two types of compression, lossless and lossy. Lossless compression allows for the perfect reconstruction of individual pixels during decompression. Lossy compression can provides much higher compression rates at the expense of potential data loss.

Images contain a number of pixels. Each pixel typically contains a byte (8-bits) of information per component, ranging from 0-255. A component is a color plane with the value of the pixel representing different intensities of said color. A standard color image contains three components (red, green, and blue), while a grayscale image includes only one.

For example, the new 1080p video standard produces images that are 1920 pixels wide by 1080 pixels high at 60 frames per second. The total number of bytes that are necessary to transmit this information is:

$$\begin{aligned} \text{Total bytes} &= (\# \text{ of pixels wide}) \mathbf{1920} \times & (1.1) \\ &(\# \text{ of pixels high}) \mathbf{1080} \times \\ &(\# \text{ of frames/sec}) \mathbf{60} \times \end{aligned}$$

$$\begin{array}{rcl}
 \text{(bytes per pixel)} & 1 & \times \\
 \text{(\# of color planes)} & 3 & \times \\
 \hline
 & \mathbf{373,248,000 \text{ bytes}} &
 \end{array}$$

Four hundred megabytes per second is a considerable amount of data. These bytes must then fit within the same bandwidth (channel capacity) designed for the previous 480i standard that produced only 30 megabytes of data a second. As it is much more cost effective to compress a video stream then change every cable/satellite channel across the world. Therefore, a compression technique is required to compress this data by a factor of roughly ten.

The most common form of video compression for this high definition format is the MPEG-2 standard. MPEG-2 contains three types of frames: inner frames, predictive frames, and back predictive frames as seen in Figure 1.1. A frame is merely one picture or still image of a movie. Inner frames are encoded as standard JPEG images and interlaced at periodic intervals within the video stream. Using the inner frames as a reference, the predictive and back predictive frames are constructed as the difference between previous frames. Thus, instead of keeping all the image data for each frame, only the differing information between frames is saved.

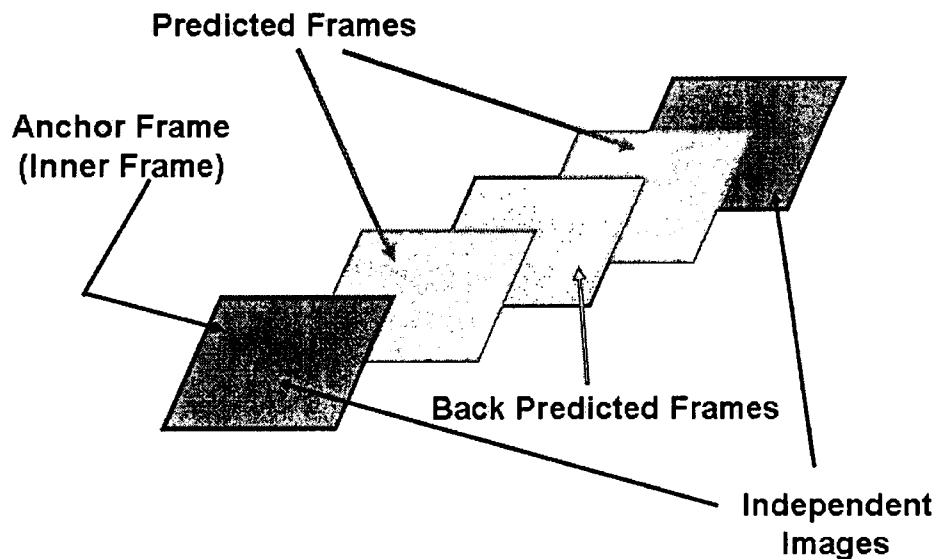


Figure 1.1: MPEG-2 Frame Illustration

1.2 The JPEG-2000 Compression Summary

In December 2000, the International Organization for Standardization (ISO) announced the standardization of the JPEG-2000 still image compression scheme [1]. JPEG-2000 provides many improvements over its predecessor standard JPEG in the following areas:

- (1) *Superior quality at low bit-rates:* JPEG-2000 images offer sharper visual image quality at very low bit-rates (less than 0.1 bit per pixel). On average, JPEG-2000 offers 30% more efficient compression than its JPEG predecessor [1].
- (2) *Larger dynamic range:* JPEG-2000 allows for a much wider range of bits per pixel (1 to 38 bits) for each color component than standard JPEG (1 to 8 bits) [1].

- (3) *Larger images*: Standard JPEG is constrained to the three color components. Conversely, JPEG-2000 can support up to 2^{14} components. JPEG-2000 also supports image sizes of $(2^{32} - 1)$ by $(2^{32} - 1)$ [1].
- (4) *Both lossless and lossy compression options*: JPEG-2000 supports both reversible and irreversible compression within the single unified standard [1].
- (5) *Progressive transmission*: The JPEG-2000 code-stream can be organized by pixel accuracy, by resolution level, or by spatial region. This progressive nature allows for the display of partial images as compressed bits are incrementally received by the decoder [1].
- (6) *Random Code-stream access*: The JPEG-2000 code-stream is easily accessed at random points allowing post-compression manipulation of images [1].
- (7) *Region of Interest Coding*: The user may require a certain region of the image to be encoded with a higher quality compared to the remaining part of the image. This is applicable in facial recognition and target tracking applications [1].
- (8) *Hierarchical algorithmic structure*: During compression, a JPEG-2000 image is partitioned into smaller, independent images. This encourages parallel processing [1].

These additional features and added flexibility result in a much more computationally intense and time consuming algorithm than original JPEG. In fact, it takes on the average of 30 times longer to compress identical images using JPEG-2000 versus JPEG [5]. But, JPEG-2000's hierarchical algorithmic structure allows for multiple tiles, which decompresses the original image into smaller images to be processed independently, thus decreasing the encoding time as seen in Figure 1.2.

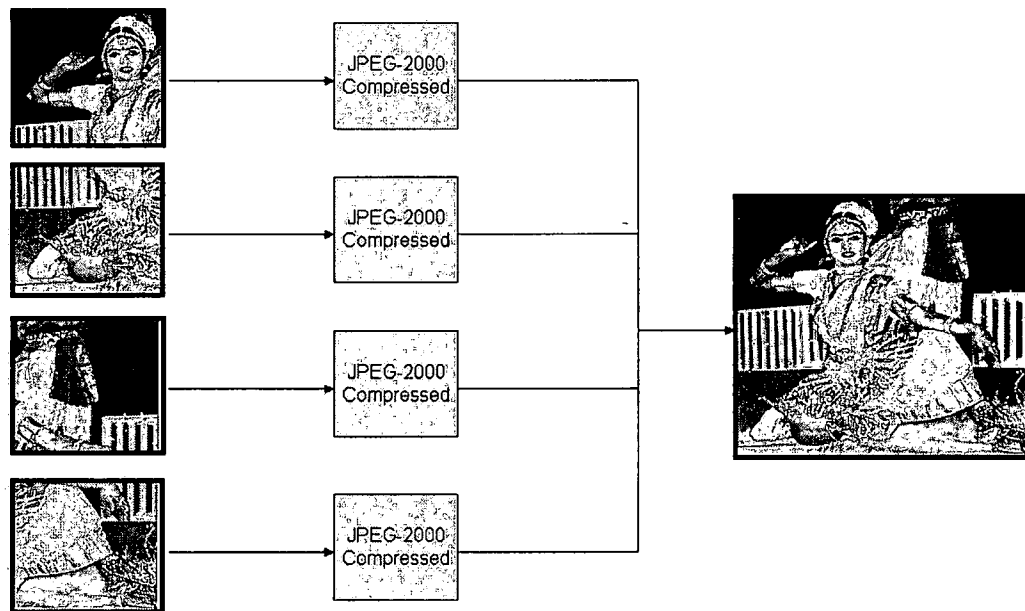


Figure 1.2: Tiling Illustration

Figure 1.2 illustrates how a singular image is partitioned into four separate tiles. The four smaller images (each is a quarter of the size of the original) are encoded concurrently and then recombined into the full image. Tiling is explained in more detail in Section 1.4.2. This simple example demonstrates the power of the inherent parallelism of JPEG-

2000. By merely dividing the image into four tiles, the total encoding time is reduced by a fourth assuming a parallel processing architecture.

1.3 The JPEG-2000 Compression Standard

The JPEG-2000 standard contains twelve parts which are as follows [7]:

- Part 1—The *Core Coding System* is defined to specify the basic algorithmic features and code-stream syntax for JPEG-2000. [1]
- Part 2—The *extensions* [2] to Part 1 of the core coding system are lengthened to allow for:
 - Alternate wavelet decomposition, quantization, and Region of Interest (ROI) extended forms
 - JPX, a new file format, to support extended color spaces, animation, and multiple quality layers
 - Metadata set for photographic imagery
- Part 3—*Motion JPEG2000* [3] file format (MJ2) support is added to encode image sequences with the JPEG2000 core for motion video.
- Part 4—*Conformance Testing* [4] specifies procedures for encoding/decoding compliance testing.
- Part 5—*Reference Software* [5] is provided to validate and test JPEG2000 systems.
- Part 6—The *Compound Image File Format* [6] (JPM) is specified to facilitate the storing of large compound images.

- Part 7—Abandoned.
- Part 8—*Secure JPEG2000* (JPSEC) asserts security protocols such as encryption and watermarking for JPEG2000 applications.
- Part 9—*Interactivity Tools, API's, and Protocols* [12] (JPIP) are defined for efficient exchange of JPEG2000 images, videos, and related metadata.
- Part 10—*3D and Floating Point Data* (JP3D) is concerned with the coding of three-dimensional data, thus extending JPEG2000 from planar to volumetric images.
- Part 11—*JPEG2000 Wireless* (JPWL) standardizes tools and methods to achieve efficient transmission of JPEG2000 imagery over an error-prone wireless network.
- Part 12—*ISO Base Media File Format* is a joint JPEG2000 and MPEG-4 initiative to create a general format for timed sequences of media data for future applications.

The majority of the discussion will focus on Parts 1 and 9 of the standard, the core coding system and the JPIP communication protocol. The core coding system is divided into three phases called (1) preprocessing, (2) compression, and (3) bit-stream formation. A block diagram of the JPEG2000 encoding algorithm can be seen below in Figure 3 [7].

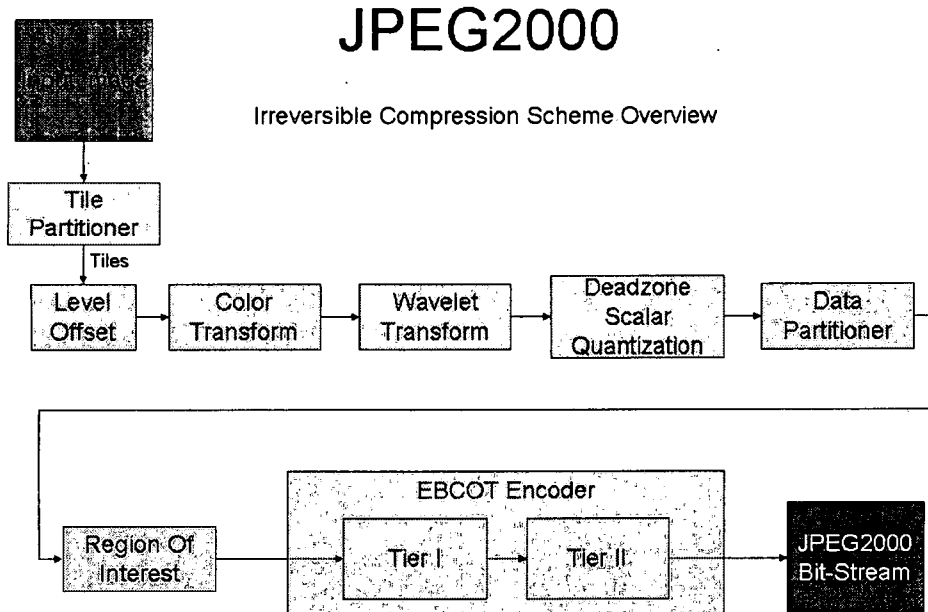


Figure 1.3: JPEG2000 Encoding Algorithm Block Diagram

1.4 Preprocessing Phase

Phase 1 consists of manipulating and transforming the data to allow efficient encoding during the compression phase.

After the image is decomposed into smaller independent images (tiles), the data is transformed into a more efficient format for encoding.

Image pixels are converted from the spatial to transform domain by an orthogonal transform. Effectively, the transform de-correlates the pixels and represents the information in a condensed number of coefficients designed specifically for the entropy encoder scheme used.

Image transforms are all derived through the Karhunen-Loeve decomposition [8]. But, the Karhunen-Loeve transform is computationally intensive and content dependent so it is not ideal for compression

purposes. Blocked based transforms, such as the Discrete Cosine Transform (DCT) and wavelets are the popular choices for compression techniques. The JPEG compression standard uses the DCT algorithm as it is much faster than the Karhunen-Loeve algorithm. However, since the DCT is performed on smaller blocks of pixels independently, border and blocking artifacts can occur. Consequently JPEG-2000 chose the wavelet-based approach as it operates on the entire image removing the DCT produced artifacts.

The pre-processing phase consists of six major functions all of which are optional except the wavelet transform and data partitioner: *Tile Partitioner, Level Offset, Color Transform, Wavelet Transform, Deadzone Scalar Quantization, and Data Partitioner.*

1.4.1 Tile Partitioner

Tiling allows for an efficient mechanism to partition a large input image into a number of rectangular non-overlapping blocks. Each block is called a *tile*. The tiles can be of any arbitrary size, but are typically chosen to be powers of two. Each tile is processed and compressed independently. Tiling also allows for random spatial access of an image and is an important function of the JPIP protocol.

1.4.2 Level Offset

The level offset, also called a DC level shift in other literatures, ensures that the image pixel values are centered about zero. This helps improve the compression efficiency as the closer a value is to zero, the fewer number of bits it takes to represent the sample leading to a more efficient compression.

The level offset represents unsigned integers as signed values. The formula for producing a signed value, $S(n)$, from an unsigned number, $U(n)$, is as follows, where bitdepth refers to the precision (number of bits) of each pixel [7]:

$$S(n) = U(n) - 2^{BitDepth-1} \quad (1.4.2.1)$$

Assuming the bit-depth as 8, to convert the unsigned integer 100 into its signed representation produces [7]:

$$S(n) = 100 - 2^{8-1} = 100 - 128 = -28 \quad (1.4.2.2)$$

In this example, the value -28 can be represented using seven bits (six bits plus a sign bit) while eight bits are needed for 100.

1.4.3 Color Transform

Color images are composed of three components called color planes (red, green, and blue). By exploiting color plane redundancy by reducing the correlation between these components, the color transform improves the overall compression rate. There are two types of color transforms in the JPEG2000 standard, reversible and irreversible.

The reversible color transform (RCT) is used for *lossless* compression. It allows the individual pixels to be reconstructed identically with the inverse of the reversible color transform. The forward and inverse RCT functions are seen below [7].

$$\begin{pmatrix} Y_r \\ U_r \\ V_r \end{pmatrix} = \begin{pmatrix} \frac{R + 2G + B}{4} \\ B - G \\ R - G \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} G \\ R \\ B \end{pmatrix} = \begin{pmatrix} Y_r - \frac{U_r + V_r}{4} \\ V_r + G \\ U_r + G \end{pmatrix} \quad (1.4.3)$$

The Irreversible Color Transformation (ICT) is used in *lossy* compression systems and is often referred to as the luminance-chrominance color transformation. It transforms the data into its luminance and chrominance components. The Y component is the luminance (intensity of light on each pixel) of the image. Cb and Cr are the two chrominance (color information) components. The transformation matrices for the forward ICT is seen below.

$$\begin{pmatrix} x_Y[n] \\ x_{C_b}[n] \\ x_{C_r}[n] \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.168736 & -0.331264 & 0.5 \\ 0.5 & -0.418688 & -0.081312 \end{pmatrix} \begin{pmatrix} x_R[n] \\ x_G[n] \\ x_B[n] \end{pmatrix}$$

1.4.4 The Discrete Wavelet Transform

There are two wavelet transforms JPEG-2000 recommends. For lossless compression, the LeGall wavelet ((5, 3) filter) is used. Lossy compression utilizes the Daubechies wavelet ((9, 7) filter).

The Discrete Wavelet Transform (DWT) separates a signal into its low and high frequency components. The DWT is implemented in JPEG-2000 by dual Finite Impulse Response (FIR) filters in a filter bank structure. In the forward DWT, both a low-pass filter (h) and a high-pass filter (g) independently filter the input signal (x) into two output streams. These output streams are then sub-sampled to produce the low-pass (y_L) and high-pass (y_H) sub-band outputs as seen in Figure 1.4 [8].

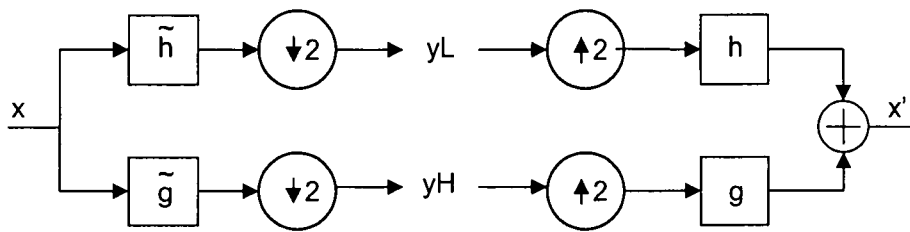


Figure 1.4: Single Dimensional Filter Bank

During the inverse transform computation, both y_L and y_H are up-sampled and filtered by low-pass (h) and high-pass (g) filters respectively. Then they are added together to obtain the reconstructed signal (x') as shown in Figure 1.4.

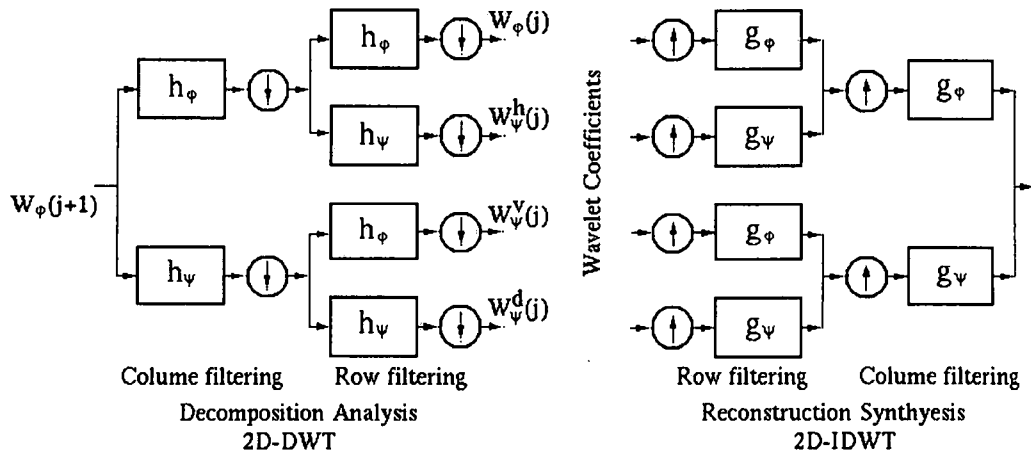
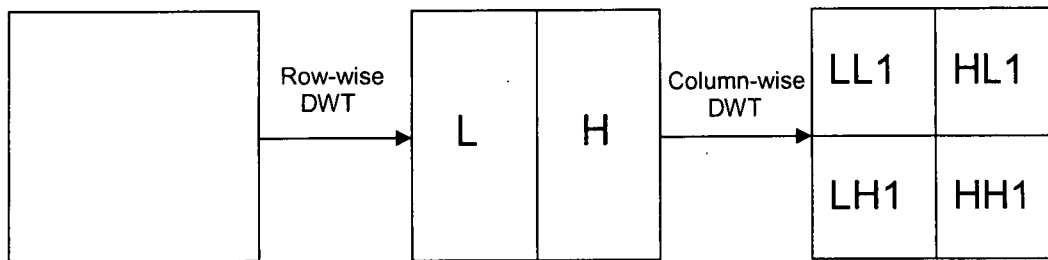


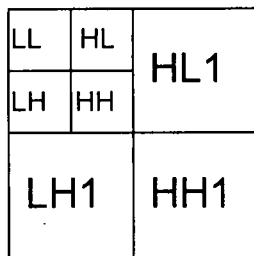
Figure 1.5: Two Dimensional Filter Bank

For multi-resolution wavelet decomposition, the low-pass sub-band (yL) becomes the input to the next filter bank effectively decomposing the low frequency information further. Each iteration of this process produces a new resolution level and transforms the image into a dyadic wavelet pyramid of N levels, where N is the number of iterations. A two level DWT decomposition and reconstruction can be seen in Figure 1.5.

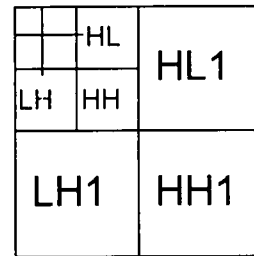
Since two dimensional wavelet filters are separable functions, a two dimensional DWT can be obtained by first applying the single dimensional DWT to each row (to produce Low-pass and High-pass sub-bands in each row) and then to each column as illustrated in Figure 1.6. In the first level of decomposition, four sub-bands LL1, LH1, HL1 and HH1 are obtained. By repeating the wavelet decomposition in the LL1 sub-band, the LL2, LH2, HL2 and HH2 are produced as shown in Figure 1.6. An illustration of the wavelet transform is shown in Figure 1.7 for one resolution level and Figure 1.8 shows a two resolution wavelet decomposition [8].



(a) First Level of Decomposition



(b) Second Level of Decomposition



(c) Third Level of Decomposition

Figure 1.6: Three Levels of decomposition in 2D DWT

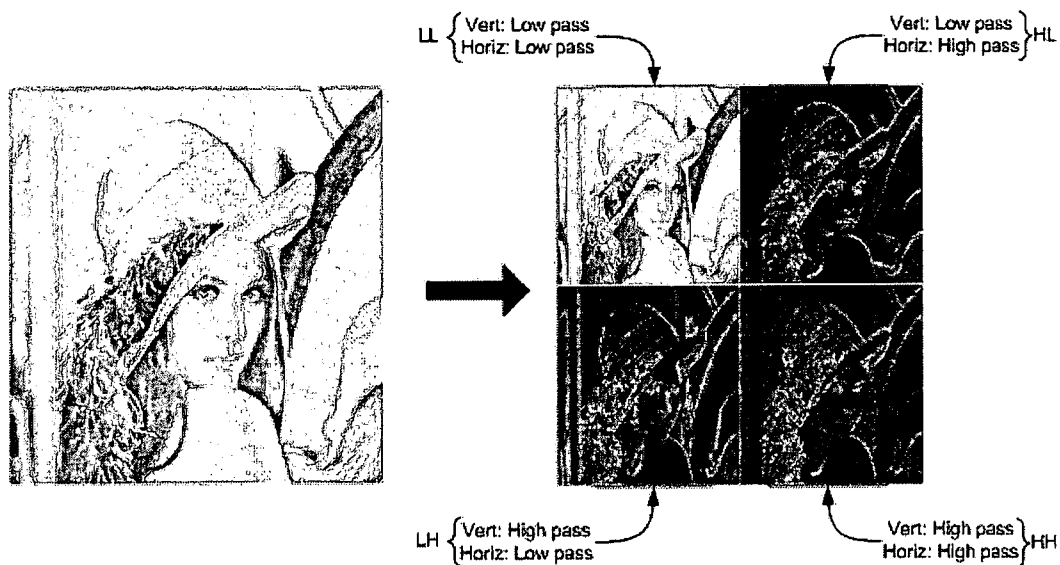


Figure 1.7: First layer of decomposition visual

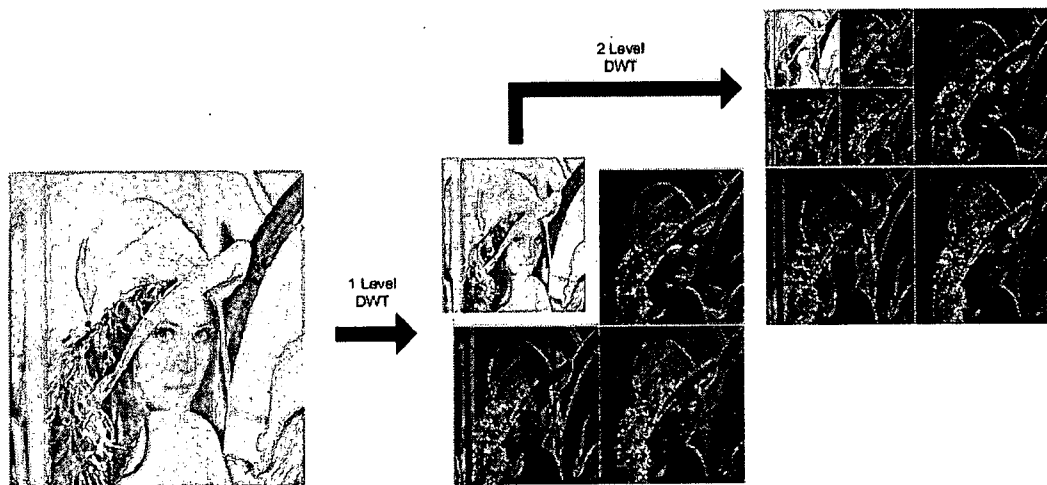


Figure 1.8: Second layer of decomposition visual

1.4.5 Quantization

After the DWT, all the sub-bands are quantized in lossy compression mode in order to reduce the precision of the sub-bands to aid in achieving compression. Quantization is not performed in the case of lossless encoding. JPEG-2000 utilizes uniform scalar quantization with dead-zone about the origin. In a dead-zone scalar quantizer with step-size Δ_b , the width of the dead-zone (i.e., the central quantization bin around the origin) is $2\Delta_b$ as shown in Figure 1.9 below.

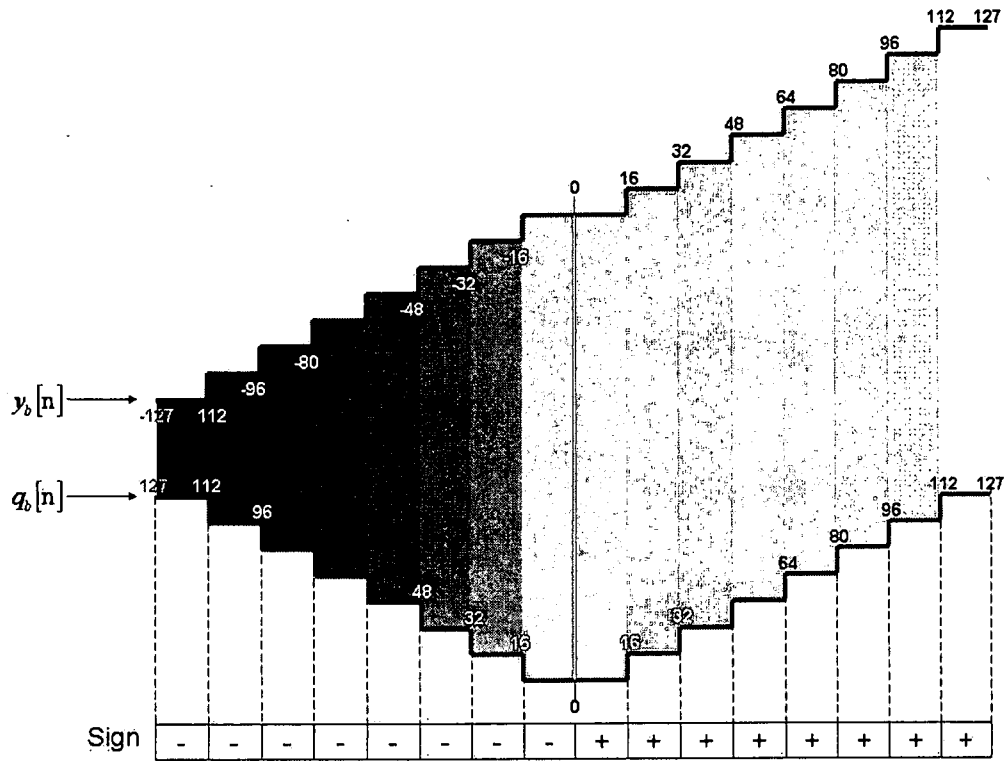


Figure 1.9: Graphical Representation of Dead-zone Scalar Quantization

The standard supports separate quantization step sizes for each sub-band. The quantization step size (Δ_b) for a sub-band (b) is calculated based on the dynamic range of the sub-band values. The formula of uniform scalar quantization with a dead-zone is [11]

$$q_b(i, j) = \text{sign}(y_b(i, j)) \left\lceil \frac{|y_b(i, j)|}{\Delta_b} \right\rceil, \quad (3)$$

where $y_b(i, j)$ is a DWT coefficient in sub-band b and Δ_b is the quantization step-size for the sub-band b . All the resulting quantized DWT coefficients $q_b(i, j)$ are signed integers.

All the computations up to the quantization step are carried out in a two's complement form. After the quantization, the quantized DWT coefficients are converted into sign-magnitude representation prior to entropy coding because of the inherent characteristics of the entropy encoding process.

The quantization step-sizes (Δ_b) for each sub-band b are calculated using an exponent (epsilon) and mantissa (mu) pair. The formula for calculating these values are seen in equation 4.

$$\Delta_b = 2^{-\epsilon_b} \left(1 + \frac{\mu_b}{2^{11}} \right) \quad (4)$$

The exponent and mantissa pair is used to represent the quantization level in the code-stream.

1.5 Compression

After the preprocessing phase, each of the tiles is further decomposed into smaller non-overlapping partitions called code-blocks. Typical sizes for code-blocks are 64 by 64 bits. The size of the code-blocks allows them to reside in CPU cache memory greatly reducing encoding and decoding time. Also, since the code-blocks are independently encoded, it allows parallel encoding of code-blocks. The EBCOT (Embedded Block Coding with Optimal Truncation) entropy encoding block is broken into two parts, Tier I and Tier II. Tier I encodes code-blocks while Tier II organizes the compressed code-stream.

1.5.1 Tier I Coding

The Tier I coding block is the arithmetic encoding portion of the EBCOT block. The code-block is separated into bit-planes. A bit-plane refers to the bits of the code-block at a specific depth or significance. For example the most significant bit of every pixel in the code-block is its own bit-plane. By encoding the more significant bits of a sample first followed by the lesser bits in a descending order, the output compressed stream has the embedding property. This occurs because a partial decoding of all coefficients produces a lower rate bit-stream than a full decode step.

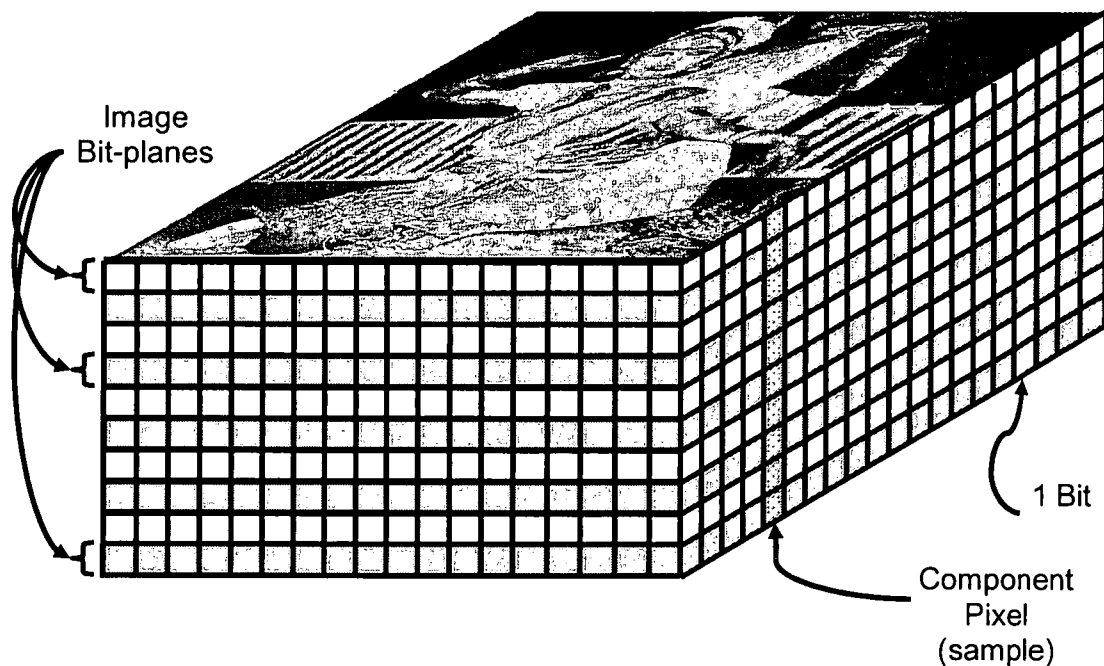


Figure 1.10: Bit-plane Illustration

Each bit-plane is scanned in a particular scan pattern as shown in Figure 1.11. The scan pattern can be divided into sections (or stripes),

each with four consecutive rows starting from the first row of a code-block. The scan proceeds top to bottom, left to right, until all the elements of a code-block are encoded or decoded. The highlighted green box is called a neighborhood and refers to the eight surrounding samples of the current bit (colored in white in this example).

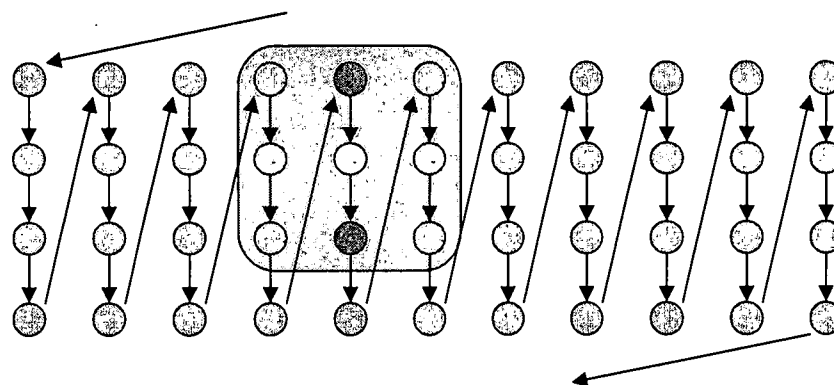


Figure 1.11: EBCOT Scan Pattern

Each bit in a bit-plane is encoded by a fractional bit-plane coding technique to generate intermediate data in the form of a context and a binary decision value for each position. The binary decision values generated by the EBCOT are encoded using a context adaptive binary arithmetic coder, called the MQ-coder. The context information generated by EBCOT is used to select the estimated probability value from a lookup table. The MQ-coder generates the compressed bit-stream from the lookup table. The Tier I uses a predefined lookup table with 47 entries to create only 19 possible different contexts for each bit type. This facilitates quick adaptation in the MQ-coder and produces compact bit-streams[9].

EBCOT encodes each bit in one of the following three coding passes in order.

- *Significant propagation pass (SPP)*: During SPP, a bit is coded if its location is not significant, but at least one sample in its neighborhood is significant. Significant means that the bit is the most significant bit of the sample value [8].
- *Magnitude refinement pass (MRP)*: All the bits that have not been coded in SPP, but became significant in a previous bit-plane are coded in this pass [8].
- *Cleanup pass (CUP)*: All the bits that have not been coded in either SPP or MRP are coded in this pass. CUP also performs a form of run-length coding to efficiently code a string of zeros [8].

In EBCOT, the coding passes perform one or more of the four possible coding operations to generate the context and decision information to be subsequently entropy encoded by the MQ-coder. The four coding operations are zero coding (ZC), sign coding (SC), magnitude refinement coding (MRC), and run-length coding (RLC). The coding operations require the use of three different state variables σ , σ' , and η . In addition to these state variables, the EBCOT requires two more arrays of a code-block size. They are sign array (\hat{A}) and magnitude array (\hat{A}). The sign array is used to store the sign bits of the elements of the code-block. The magnitude array stores the unsigned integers of the code-block elements. A block diagram of the Tier I processing chain can be seen below in Figure 1.12 [7].

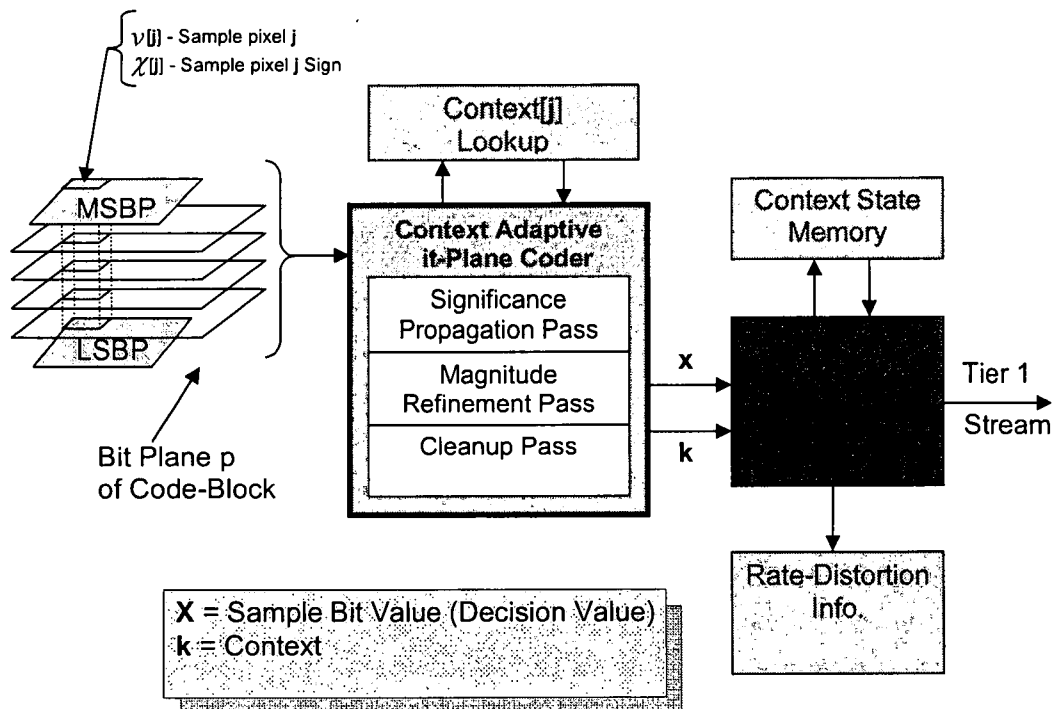


Figure 1.12: Tier-1 Overview

1.5.1.1 Significance Propagation Pass

The significance propagation pass operates on a pixel whose bits have not already been deemed significant. Operating on a bit at a time, this pass determines if any pixel in the current bit's neighborhood is significant. If any of the neighboring pixels are significant, the current pixel bit is encoded with the MQ-coder and is made significant by setting $\sigma \leftarrow 1$. Thus, significance propagates outward from pixels that are already significant. In addition, the coding pass membership state variable σ is set to 1 indicating that the pixel bit has already been encoded by the

significance propagation pass and should not be encoded by the magnitude refinement pass.

The context value is calculated based on the current sub-band the pixel resides and the number of significant pixels in the neighborhood.

Table 1 defines the context values for each sub-band.

Table 1: Context Values for SPP Pass

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	3	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

1.5.1.2 Magnitude Refinement Pass

Bits in the pixel that are already significant and not encoded by the significance propagation pass are encoded in the magnitude refinement pass. Since the MPP always follows the SPP, it is possible that a pixel is significant but its bit in the current bit-plane should not be encoded by the magnitude refinement pass. This is the purpose of the coding pass membership state variable σ . This state variable keeps track of when a bit is encoded by the SPP. The pixel's encoded magnitude value is then refined. The magnitude refinement pass also forms context information.

This context value is based on 2 different values. The first is significance value which is formed by the SPP, and the delayed significance (based on the value of σ). These two values in conjunction with a lookup table form the context value of the MRP.

1.5.1.3 Cleanup Pass

The cleanup pass (CUP) follows the SPP and MRP passes and encompasses two different coding paths. The first path is similar to the SPP. The second path performs a run-length coding scheme when the significance of the entire neighborhood is zero. When run length coding occurs, the length of the run is measured while the bit-value is equal to zero. This greatly improves performance in regions of zeros in each bit-plane. The run length is passed to the MQ coder. In addition, any bit not encoded in either the significance propagation or the magnitude refinement pass is encoded in the cleanup pass.

1.5.1.4 Sign Coding

Recall that after quantization, the pixel values are represented as sign and magnitude components. The sign therefore must be encoded at some point. This encoding takes place as soon as a pixel becomes significant. Thus, a pixel can have its sign encoded in either the SPP or the CUP.

The sign encoding context formation is based on the significance and sign value of the neighboring vertical and horizontal pixels. An intermediate value known as the horizontal and vertical sign bias functions are computed as

$$\begin{aligned}\chi^h[j] &= \chi[j_1, j_2 - 1]\sigma[j_1, j_2 - 1] + \chi[j_1, j_2 + 1]\sigma[j_1, j_2 + 1] \\ \chi^v[j] &= \chi[j_1 - 1, j_2]\sigma[j_1 - 1, j_2] + \chi[j_1 + 1, j_2]\sigma[j_1 + 1, j_2]\end{aligned}\quad (7)$$

where χ is the pixel sign value $\chi = (-1, 1)$ and σ is the significance of the pixel $\sigma \in [0, 1]$. These intermediate values are then truncated to the range $[-1, 1]$ with the functions

$$\begin{aligned}\bar{\chi}^h[j] &= \text{sign}(\chi^h[j])\min\{1, |\chi^h[j]|\} \\ \bar{\chi}^v[j] &= \text{sign}(\chi^v[j])\min\{1, |\chi^v[j]|\}\end{aligned}\quad (8)$$

The context is then determined from the horizontal and vertical sign values using a lookup table and sent to the MQ coder for processing.

1.5.2 Rate Control and Optimal Truncation

Rate control is a process by which the bit-rates are allocated in each code-block in order to achieve the overall target encoding bit-rate for the whole image while minimizing the distortion (errors) introduced in the reconstructed image due to quantization and truncation of codes to achieve the desired compression rate.

The JPEG2000 encoder generates a number of independent bit-streams by encoding the code-blocks. Accordingly a rate-distortion optimization algorithm generates the truncation points for these bit-streams. The truncation points are then analyzed to determine the optimal way to minimize the distortion corresponding to a target bit rate. Since this occurs on a code-block level, it is a much finer rate control than controlling it through the quantization step size because the step size. The step size can only be controlled on a sub-band basis.

1.5.3 Tier-2 Coding

After the compressed bits for each code-block are generated by Tier I coding, the Tier II coding engine efficiently represents the layer and block summary information for each code-block. A layer consists of consecutive bit-plane coding passes from each code-block in a tile, including all the sub-bands of all the components in the tile. The block summary information consists of length of compressed code words of the code-block, the most significant magnitude bit-plane at which any sample in the code-block is nonzero, as well as the truncation point between the bit-stream layers, among others. The decoder receives this information in an encoded manner in the form of two tag trees. This encoding helps to represent this information in a very compact form, without incurring too much overhead in the final compressed file. The encoding process is

popularly known as Tag Tree coding. For more information on tag tree coding, refer to [11].

1.5.3.1 Packets, Progressions, and Precincts

JPEG 2000 provides five different progressions or "scalabilities" in which an image can be encoded. Each progression has different attributes which may be desirable across a range of applications. Each of these progressions produce "quality layers" which contains a portion of the image's overall data. When all of the quality layers are used to reconstruct an image, the resulting image is of the highest quality available with respect to how the image was encoded. These quality layers can be scalable in several different ways, called progressions.

There are four progressions defined by the JPEG-2000 standard and they are broken into three categories: (1) resolution scalable, (2) signal to noise ratio scalable, (3) component scalable. Each progression uses a different technique to determine what information goes into each quality layer. For example, signal to noise ratio uses the rate-distortion information to determine what portions of a code-block's code-stream go into each quality layer. Figure 1.12 shows this process graphically below.

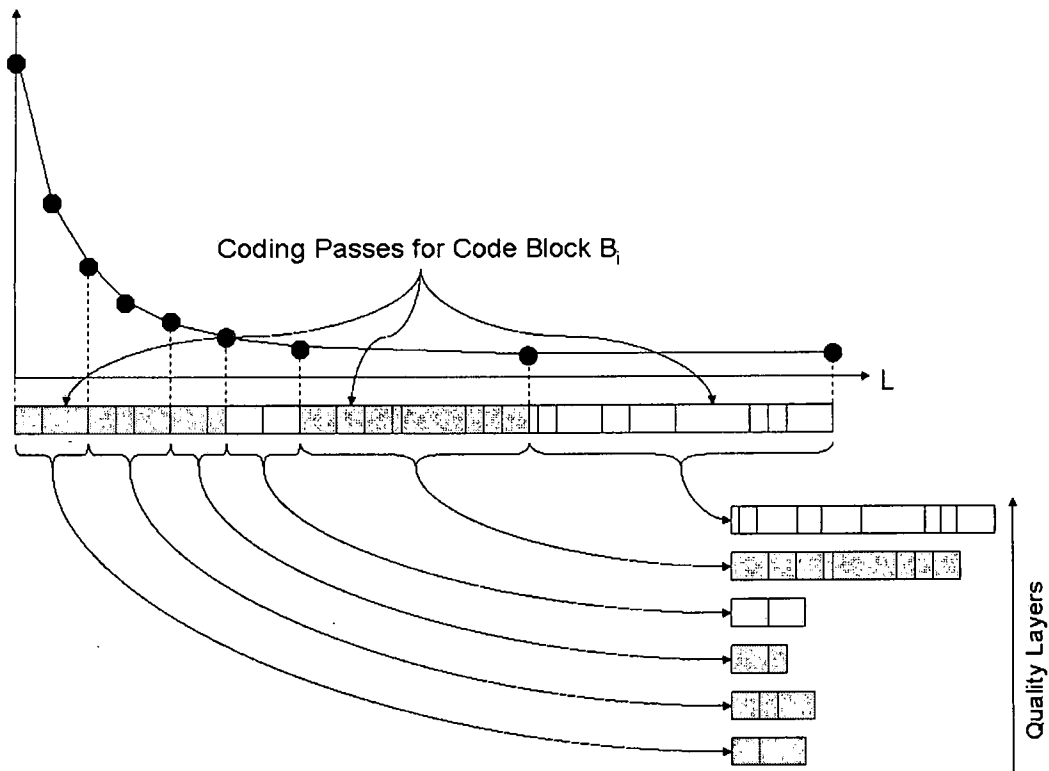


Figure 1.12: Rate-Distortion Information to Quality Layers

As another example, resolution scalability uses the resolution level boundaries as defined by the wavelet transform to determine the quality layers. Recall that each successive resolution level of the wavelet transform contains twice the resolution of the previous.

Precincts contain at least one code-block from the same spatial area of each sub-band in a wavelet resolution level. Each of these precincts is used to form one packet per quality layer. Therefore, for each precinct, there exists one packet per quality layer. Each sub-band precinct contains its own header information within the packet header, which appears before any of the image information (which is provided by the MQ

coder) in the code stream. This allows for random extraction of certain parts of the image that is suitable for an application.

The order in which the sub-band precincts appear in the code-stream is always in the well defined order of HL, LH, HH. The packet structure is shown graphically in Figure 1.13 below.

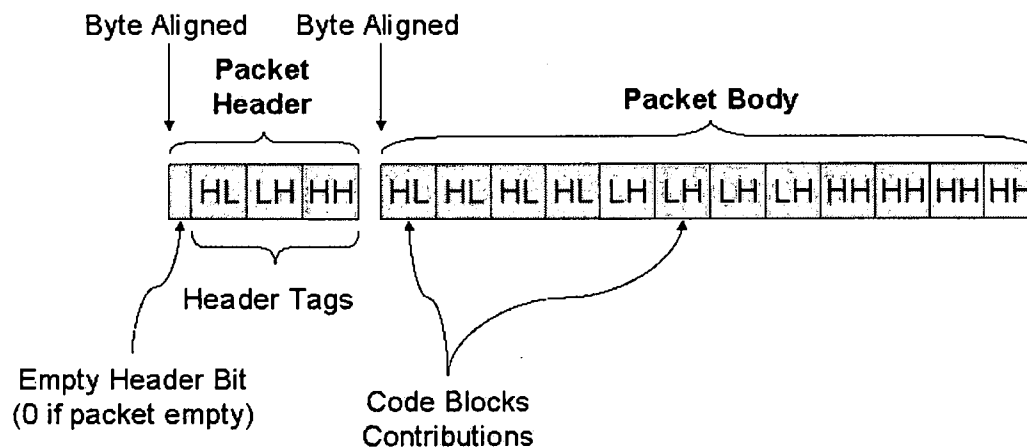


Figure 1.13: JPEG-2000 Packet Construction

These packets are then continuously placed in a JPEG-2000 code-stream depending upon the desired progression.

Chapter 2

JPIP

2.1 Introduction

The ISO/IEC Joint Technical Committee of Photographic Experts (JPEG) developed an international standard for interactivity with JPEG-2000 code-streams and files. This standard is known as JPIP (JPEG-2000 Internet Protocol), and is Part 9 of the standard mentioned in section 1.3. The purpose of JPIP is to standardize a means of interacting with JPEG-2000 based data in an efficient and effective manner [12].

Recall the many features of the JPEG2000 image compression standard that are in support of interactive access to large images. Chief amongst these are resolution scalability, quality scalability, spatial random access, and highly efficient compression. However, the compression standard itself describes only the code-stream syntax suitable for storing the compressed data in a file. An ideal application is an intelligent browsing client that could access appropriate regions of interest from the

compressed JPEG-2000 file. Such an approach allows existing HTTP servers to be used.

2.2 JPIP Advantages

With the JPIP protocol, a client does not directly access the compressed file. Instead, it requests a portion of the image as its window of interest (WOI). The WOI approach is a more efficient service for interactive imaging, with fewer round-trip delays, and the opportunity for servers to efficiently manage their resources [12]. JPIP requests contain the form of identifying the client's spatial region of interest and resolution. The server then determines the optimum sequence of response elements in order to efficiently process the request, so as to minimize bandwidth while optimizing image quality to the client. For interactive applications, multiple JPIP requests can be issued and efficiently cached within an interactive browsing session, allowing the server to avoid the redundant requests. The JPIP protocol has features which commend its use for collaborative remote image interaction, for providing services on unreliable transports (e.g., wireless transmission), for flexible and domain-specific delivery of meta-data, and for applications which involve low data-rate networks [12].

2.3 JPEG-2000 Code-Stream Elements

Recall from chapter 1, that JPEG2000 is based on the Discrete Wavelet Transform (DWT), together with Embedded Block Coding with Optimized Truncation (EBCOT). The DWT analysis decomposes the image into sub-bands. Each sub-band is partitioned into square blocks, known as code-blocks. Each code-block is independently coded into a finely embedded bit-stream and can be truncated at any point resulting in a much finer granularity in bit-rates than quantizing on the sub-band level. Each block of each sub-band may be independently truncated to any desired length. This process is known as optimal truncation.

2.4 JPIP Client-server Interaction Overview

Figure 2.1 illustrates the typical interaction between a client application and a remote server. Normally the client application is a graphical user interface (GUI) and is driven by an interactive user. We use the term "window of interest" to refer to the user's current spatial region and resolution. Typically, the interactive user pans (changes the spatial region) or zooms (changes the resolution or scale) the focus window.

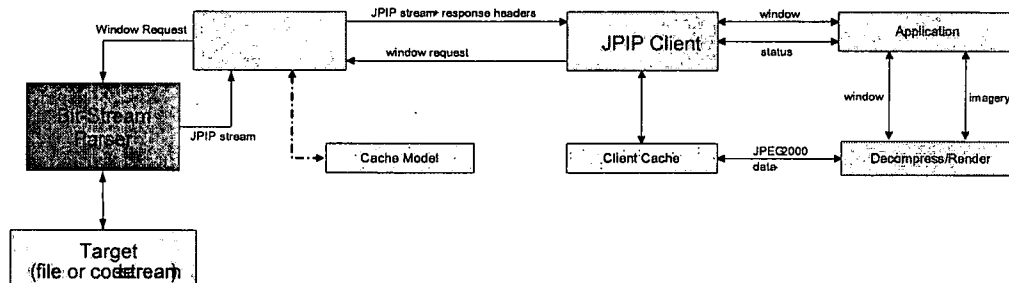


Figure 2.1: JPIP Server/Client Architecture

The Server and Client boxes are the HTTP interfaces between the two systems. The client makes a WOI request via the HTTP protocols and the server then processes that request in an efficient manner. It first determines the resolution and spatial region of the WOI. The Bit-Stream Parser (highlighted in red) loads the appropriate image and determines the specified resolution to return to the client. An important aspect of Figure 2.1 is the separation of the image decompression/rendering process from client-server communication. The inherent scalability of JPEG2000 means that an image can be decoded either on the server or client side of the communication. This is advantageous if one wants all the processing to occur on the server for instance and then distribute decompressed images to smaller, less robust clients. In fact, it is often helpful to render a larger region of the image than just the WOI to provide an interactive user with navigation context [10]. As the client receives more data from the server, the contents in memory grow and decompressing is repeated to progressively refine and display the image in increasing layers of resolution.

The actual communication between client and server consists of request/response pairs. The request identifies the WOI via its geometric attributes (x and y coordinates with respect to the upper left hand corner of the canvas). This has several benefits: 1) JPIP requests can be compact and intuitive, facilitating their inclusion as URL's in HTML pages; 2) JPIP

requests can be used to extract an appropriate image region from a non-JPEG2000 file (e.g. a server may offer a trans-coding service); 3) the focus window expresses an end-user's ultimate interests, rather than a client's interpretation of those interests in terms of JPEG2000 elements allowing the server to determine how best to respond to the request [12].

2.4.1 Data-bin Allocation

JPIP defines a means for partitioning the information within any JPEG2000 file into a collection of "data-bins." Data-bins are typically transmitted and cached in linear fashion which is determined from the progression order of the encoded file.

JPIP defines two partitioning schemes for JPEG2000 code-streams, either precincts or tiles [12]. In both cases, the code-stream's main header is assigned its own data-bin followed by either one data bin for the entire tile (tile partitioning) or a data bin for each precinct within the tile (precinct partitioning). The tile-based approach offers server simplicity at the expense of reduced flexibility as spatial accessibility relies exclusively on tiles. The precinct approach is more complicated but allows for much greater spatial accessibility. The resolution parser operates on the precinct data bin approach.

2.4.2 JPIP Messages and Streams

As mentioned above, the entire contents of a JPEG2000 file are partitioned into data-bins, including code-stream and meta-data. To describe the server's response data, JPIP defines a new media type called a "JPIP stream." There are two different stream types, tile-oriented data-bin ipt-stream and precinct based ipp-stream.

A JPIP stream consists of a sequence of JPIP messages, each of which contains a range of bytes from the contents of a single data-bin [12]. Each message has its own byte-aligned header which compactly identifies the data-bin class (meta data, code-stream main header, tile header, precinct or tile), a unique identifier for the data-bin within its class and the location and length of the range of bytes which may be found in the message body [12].

A JPIP message request contains a means for identifying the target file and the WOI chosen by the user. A typical request could look like:

"target=images/test.jp2&fsiz=10000,9000&rsiz=600,400&roff=2000,3000".

This request is for the file "images/test.jp2", at a resolution whose full image size is 10k by 9k, within a WOI of size 600 by 400, at a location of 2000 pixels from the left and 3000 pixels from the top. In order to process this request, the server needs a resolution parser in order to return test.jp2 at the required resolution from the client [14].

CHAPTER 3

Bit-Stream Resolution Scalable Parser

3.1 Introduction

During work with the Air Force, a need arose for resolution scalability to be added to the JPEG-2000 encoder. As briefly mentioned in Section 2.3.1, resolution scalability allows a single JPEG-2000 compressed image to contain multiple resolutions. To implement resolution scalability, a modification is made to the encoder during the Tier II code-stream formation described in Section 3.2 below. A JPIP server then is modified in order to distribute the imagery to multiple clients with the desired resolution using the bit-stream resolution scalable parser. Also since the data is transmitted via the Universal Datagram Protocol (UDP), the size of the precincts are modified to allow for a more consistent and reliable data-stream.

3.2 Tier II Progressions

As mentioned briefly in the JPEG-2000 background information in Chapter 1, the JPEG-2000 embedded block coding scheme contains two tiers, I and II. Tier I defines an arithmetic encoding scheme that encodes the partitioned code-blocks into independent embedded bit-streams using a context based arithmetic encoder as mentioned in Section 1.5. Tier II reorders these code-blocks into the final JPEG-2000 bit-stream.

However, the code-blocks can be reordered in five different ways, called progressions. These five orders are as follows [1]:

1. Layer—Resolution—Component—Position (LRCP)
2. Resolution—Layer—Component—Position (RLCP)
3. Resolution—Position—Component—Layer (RPCL)
4. Position—Component—Resolution—Layer (PCRL)
5. Component—Position—Resolution—Layer (CPRL)

The progression chosen defines the order at which the code-blocks are placed in the bit-stream. The left most term defines the type of progression used. Therefore, numbers 2 and 3 are called progression by resolution.

3.2.1 Progression by Resolution

Progression by resolution orders the code-stream packets by individual resolution levels allowing for easier random code-stream access by the resolution parser. A JPEG-2000 encoder is modified to change the progression from a quality (LRCP) to a resolution (RLCP) progression.

As the code-stream is being built, the packets are ordered by resolution levels with the lowest level appearing first in the code-stream. Each additional level produces an increase in the resolution. This is shown in Figure 3.1. In this example, packets are inserted into the code-stream from the upper left hand corner to the bottom right.

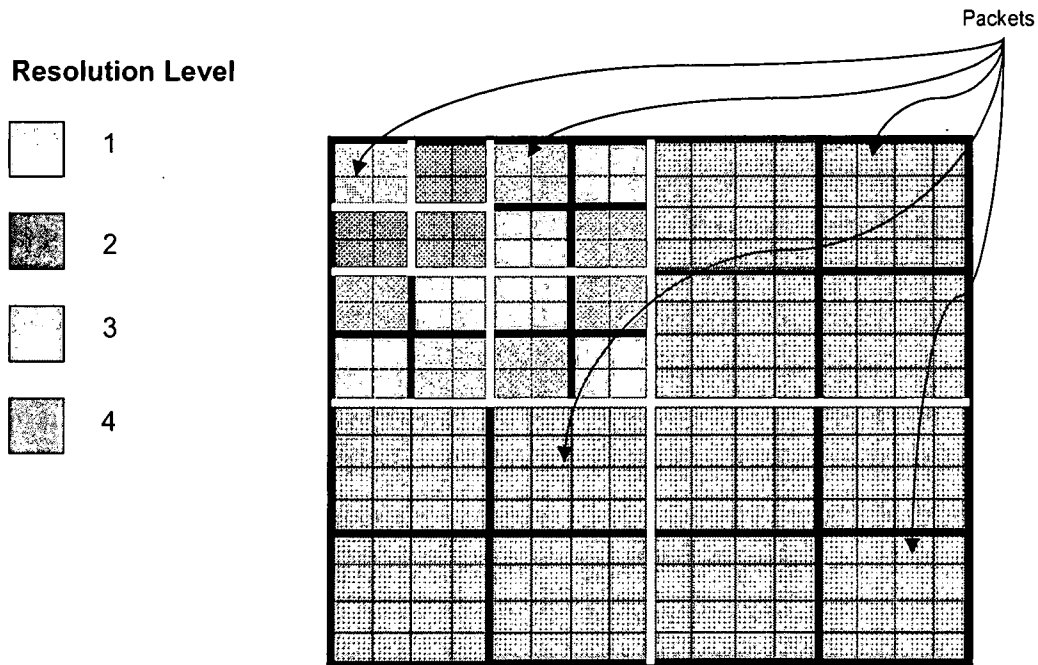


Figure 3.1: Progression by Resolution

3.3 The Bit-stream Resolution Parser

After the JPEG-2000 encoder modifications are made, the bit-stream resolution parser is developed. The parser uses the code-stream header information to determine the resolution bounds (the amount of precincts per resolution) in order to extract the desired resolution from the code-stream. The parser acts as a simple decoder that produces the desired resolution image.

Recall from Chapter 2 that each JPEG-2000 file contains a main header that informs the decoder of the image's attributes. The main header contains marker segments used to separate types of information. In particular, two marker segments, the SIZ and COD, are required to effectively parse the image into resolutions.

3.3.1 Headers and Marker Segments

JPEG-2000 uses marker segments to delimit and signal the characteristics of the codestream. Headers are collections of markers and marker segments. There are two types of headers in this specification. The main header is found at the beginning of the codestream. The tile-part headers are found at the beginning of each tile-part. Some markers and marker segments are restricted to only one of the two types of headers while others can be found in either.

Six types of markers and marker segments are used: delimiting, fixed information, functional, in bit stream, pointer, and informational. Delimiting marker and marker segments must be used to frame the headers and the data. Fixed information marker segments give required information about an image. The location of these marker segments is specified. Functional marker segments are used to describe the coding functions used. In bit stream markers and marker segments are used for error resilience. Pointer marker segments point to specific offsets in the bit stream. Informational marker segments provide ancillary information.

Table 2 below specifies the syntax and information marker segments contain.

Table 2: List of Marker Segments

	Name	Code
Delimiting marker segments		
Start of codestream	SOC	0xFF4F
Start of tile-part	SOT	0xFF90
Start of data	SOD	0xFF93
End of codestream	EOC	0xFFD9
Fixed information marker segments		
Image and tile size	SIZ	0xFF51
Functional marker segments		
Coding style default	COD	0xFF52
Coding style component	COC	0xFF53
Region-of-interest	RGN	0xFF5E
Quantization default	QCD	0xFF5C
Quantization component	QCC	0xFF5D
Progression order default	POD	0xFF5F
Pointer marker segments		
Tile-part lengths, main header	TLM	0xFF55
Packet length, main header	PLM	0xFF57
Packet length, tile-part header	PLT	0xFF58
Packed packet headers, main header	PPM	0xFF60
Packed packet headers, tile-part header	PPT	0xFF61
In bit stream marker segments		
Start of packet	SOP	0xFF91
End of packet header	EPH	0xFF92

Information marker segments		
Comment and extension	CME	0xFF64

3.3.2 Construction of the codestream

The codestream is constructed with the marker segments described in Section 3.3.1. A main header is required for the image and must appear first in the codestream followed by tile-part headers which precedes the data of each tile. An illustration of the ordering can be seen below in Figure 3.2.

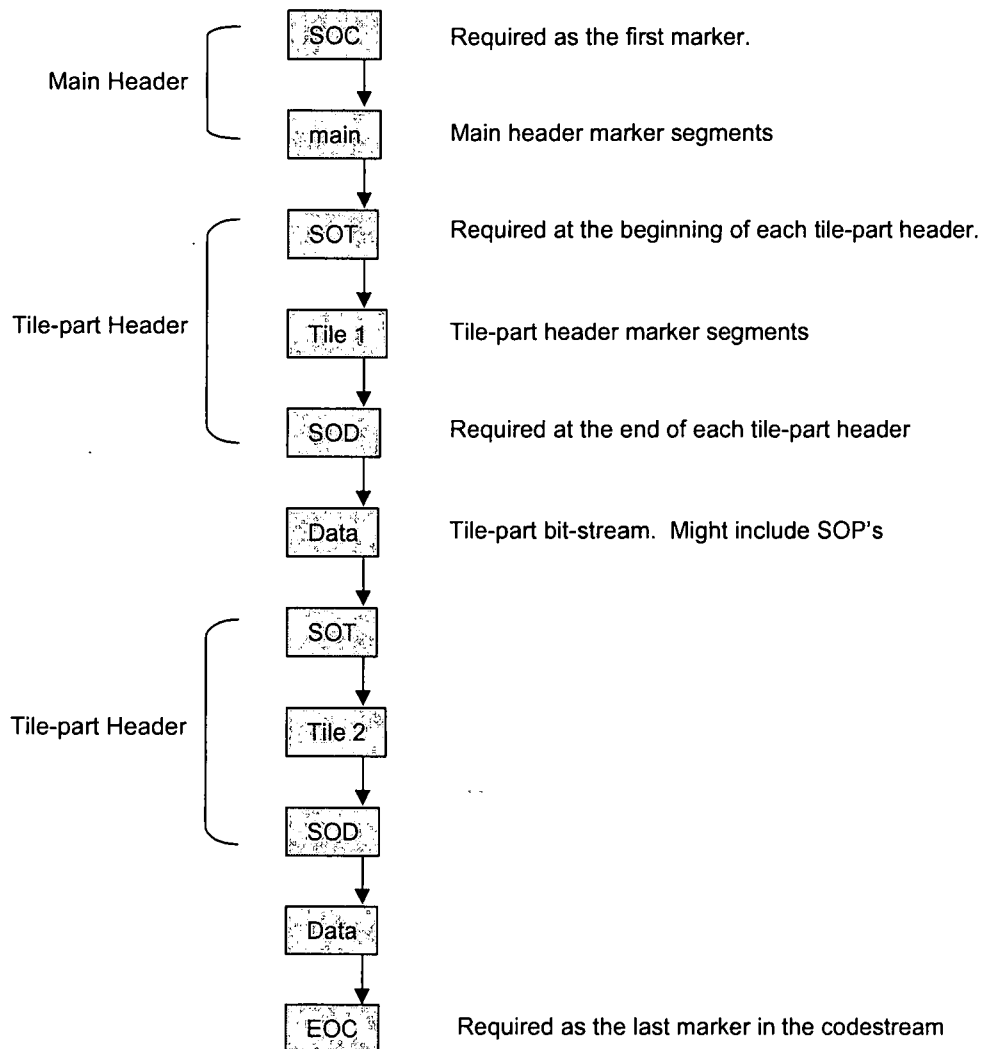


Figure 3.2: Construction of the Codestream

The tile-part header has a specific ordering and optional marker segments which can be seen in Figure 3.3. Each tile has its own header to allow independent coding and quantization characteristics to exist between tiles. In other words, each tile can be coded to have dissimilar quantization parameters, resolution levels, components, etc.

For example, the COD marker segment is required in the main header. If all components in all the tiles are coded the same way, this is all that is required. If there is one component that is coded differently than the others (for example the luminance component of an image composed of luminance and chrominance components) then the COC can denote that in the main header. If one or more components are coded differently in different tiles, then the COD and COC are used in a similar manner to denote this in the tile-part headers.

The POD marker, which controls the progression order, may similarly appear in the main header and creates the default progression order to traverse all the tiles. However, a tile can have a different progression order if its tile-part header POD marker segment exists.

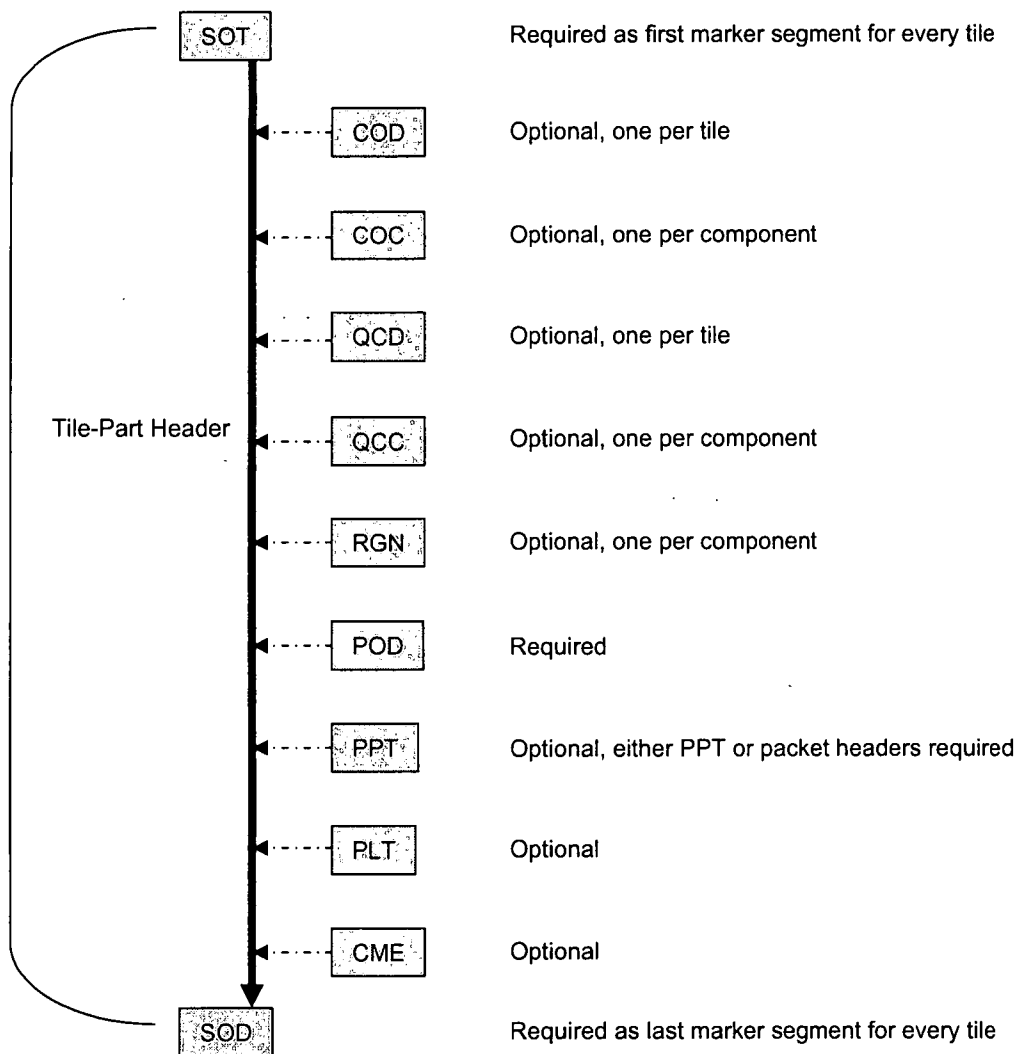


Figure 3.3: Construction of tile-part header

3.3.3 Image and Tile Size (SIZ) Marker Segment

The SIZ marker segment is the first marker in the main header and is denoted by the hexadecimal value FF51. This marker contains the height, width, anchor points, and number of components of the image and tile. Figure 3.4 shows the ordering of the SIZ marker segment.

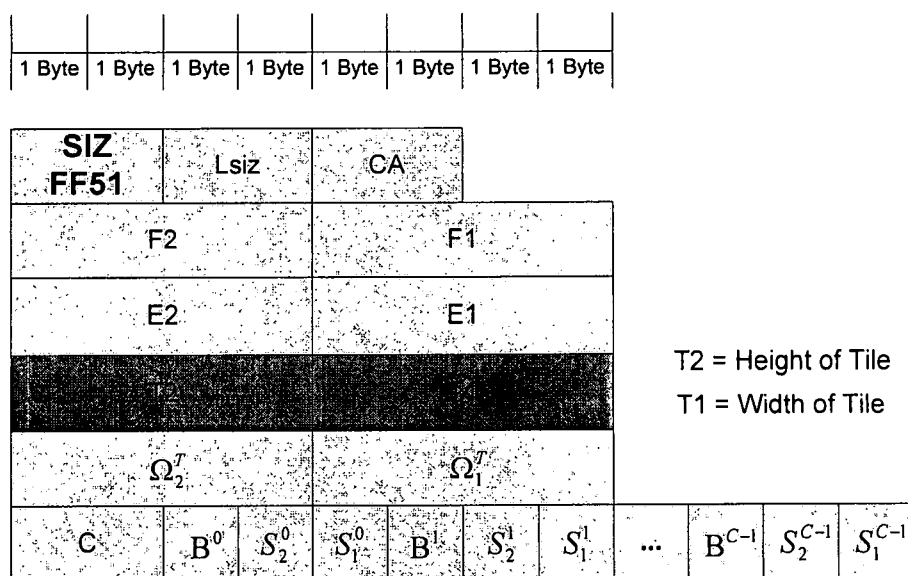


Figure 3.4: Image and Tile Size (SIZ) Marker Segment

The red highlighted portion of Figure 3.5, T2 and T1, denotes the height and width of the tile respectively. T2 and T1 are each four byte values. The height and width are required for the resolution level dimensions. Equation 3.3 shows how the resolution level is calculated from the height and width of the tile.

$$\frac{T_2}{2^{r-1}}, \frac{T_1}{2^{r-1}}, r = \text{the resolution level} \quad (3.3)$$

3.3.4 Coding Style Default (COD) Marker Segment

The COD marker segment provides the default coding style parameters. These include: progression order, number of quality layers and wavelet transform levels, size of the code-blocks and precincts, and

the wavelet transform used. Figure 3.5 illustrates the COD marker segment.

COD FF52	Lcod		CS		Λ_i	MC
	E_2^{CB}	E_1^{CB}	MS	WT		

$D_{t,c}$: Transform levels O_p : Progression Order

E_P^0 : 4 MSB's height exponent of precincts of resolution 0

E_P : 4 LSB's width exponent of precincts of resolution 0

Figure 3.5: Coding Style Default (COD) Marker Segment

The important values to ascertain from this marker segment are highlighted once again in red.

$D_{t,c}$ is the number of transform levels employed by the wavelet.

There are $D_{t,c} + 1$ resolution levels for each component. A value of zero means the wavelet transform is not performed.

The O_p parameter specifies one of the five progression orders that was used by the encoder to arrange the code-stream. It is a one-byte unsigned integer with values 0, 1, 2, 3, and 4 as seen below:

0: Layer-Resolution-Component-Position (LRCP)

1: Resolution-Layer-Component-Position (RLCP)

2: Resolution-Position-Component-Layer (RPCL)

3: Position-Component-Resolution-Layer (PCRL)

4: Component-Position-Resolution-Layer (CPRL)

The progression order determines how the parser will traverse the code-stream to extract the correct precincts of the desired resolution.

$E_p^{D,c}$ denotes the height and width of the precincts at resolution level D . Since each resolution level can have different precinct sizes, an $E_p^{D,c}$ parameter exists for each level. $E_p^{D,c}$ is a single byte value with the four most significant bits expressing the exponent of the height and the four least significant bits the width. Equation 3.4 below shows how the height ($P_1^{t,c,r}$) and width ($P_2^{t,c,r}$) are calculated.

$$P_1^{t,c,r} = 2^{E_{p,1}^r} \quad P_2^{t,c,r} = 2^{E_{p,2}^r} \quad (3.4)$$

3.3.5 Resolution Boundary and Precinct Calculation

The height and width of the precincts in conjunction with the tile size determine the resolution boundaries for the code-stream packets. The resolution height H_r and width W_r are dependent upon the tile height T_2 and width T_1 . To determine the size of resolution r , it is simply:

$$H_r = \frac{T_2}{2^{r-1}} \quad W_r = \frac{T_1}{2^{r-1}} \quad (3.5)$$

After the resolution sizes are calculated, the boundaries of the sub-bands within each resolution follow naturally. Figure 3.6 illustrates how the sub-bands are calculated. Recall the wavelet transform separates the image into four domains as determined by the high pass and low pass characteristics from the filtered results.

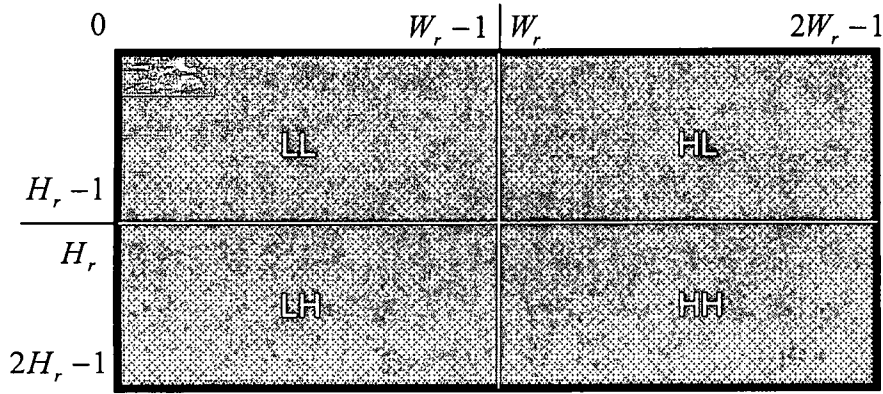


Figure 3.6: Resolution Boundary Calculation

3.3.6 Code-block and Precinct Calculation

Once the precinct boundaries have been established, the number of code-blocks per resolution level can be calculated. The number of code-blocks per sub-band is defined as:

$$CB_{SB} = \text{ceil}\left(\frac{SB_W}{CB_W}\right) * \text{ceil}\left(\frac{SB_H}{CB_H}\right) \quad 3.3.3$$

where SB_W and SB_H are the sub-band heights and widths respectively, and CB_W and CB_H are the code-block heights and widths.

The number of code-blocks per sub-band can differ for the same resolution. This occurs if the resolution boundaries are not divisible by two. Then, to compute the number of code-blocks per resolution level, simply sum each the number of code-blocks of the three sub-bands (HL, LH, and HH).

When Tier-II forms the code-stream precincts, multiple code-blocks can be contained within each precinct. Recall, code-blocks from the HL, LH, and HH sub-bands form a single precinct.

3.3.7 Start of Packet (SOP) Marker Segment

The Start of Packet (SOP) marker segment is optional and may appear immediately before each packet in the code-stream when indicated in a Coding Style Default (COD) marker segment. The SOP marker segment is useful for code-stream parsing and/or error resilient decoding. The value (0xFF91) of the SOP marker is easily searchable allowing for random code-stream access. Since the SOP marker (0xFF91) exceeds the maximum allowed value of any two consecutive bytes (0xFF8F), it can be used to detect errors arising while parsing for packet headers. Error detection and code-stream resilience accommodations for the size of the packets are described in the next section.

For the bit-stream resolution parser, the packet headers are used to determine resolution boundaries from the calculations performed in the previous sections. Recall that there exists one packet per precinct per quality layer. Therefore once the number of precincts per resolution is known, extracting the packets corresponding to those precincts is a trivial task.

3.4 UDP Transmission and Precinct Modifications

UDP is an internet protocol (IP) that does not require prior communications to set up transmission channels or data paths allowing for applications to broadcast or multicast data streams to multiple clients concurrently. However, it does not have implicit hand-shaking between clients which fails to guarantee reliability, ordering, or integrity of the data stream. Thus, UDP is an unreliable service and packets may arrive out of order, duplicated, or go missing without notice. It also does not include error checking or correction. Therefore, in order to use the UDP IP, modifications had to be made to the size of the packets in order to make error detection and correction as seamless as possible [13].

3.4.1 Precinct Modifications for Resilient Code-streams

NITFS recommended method for resolution parsing is to make the precinct the same size as the entire resolution level. The packet contains exactly one precinct per resolution layer making it relatively easy to extract the desired resolution while reducing the code-stream length as fewer SOP packer headers are located in the code-stream. The result is fewer but much larger packets.

While this is practical in most situations, problems can occur when using UDP to transmit the data. JPEG-2000 images are encoded to six resolution levels in most cases. The six levels correspond to six packets.

UDP is an unreliable transmission system. Obviously with so few packets, it is that much more important each one is transmitted reliably.

A solution is to make each precinct the size of a code-block. Since code-blocks are coded independently, errors will not propagate beyond the corrupted code-block. With the precinct modifications, now each packet corresponds to a precinct the size of a code-block. This finer granularity allows for a more resilient code-stream. Errors now are contained within 32 x 32 bytes of a code-stream, not within packets that could be as large as the entire image.

3.4.2 Error Detection during UDP Transmission

As UDP has no error detection, a simple way is to compare the number of packet headers in the code-stream to the total number of packets. Upon the detection of such errors, the presence of subsequent SOP segments enables resynchronization at a well-defined packet boundary, where processing can resume. As the total number of packets expected is already calculated, merely counting the SOP marker segments is sufficient. The small packet sizes also lessen the bandwidth requirements to retransmit corrupted data packet or packets.

CHAPTER 4

Results

4.1 Test Parameters

In order to verify the functionality of the resolution parser, three independent tests were performed for accuracy: the progression ordering, resolution parsing and JPIP integration. The images are then decoded using three independent JPEG-2000 encoders to validate each coding stream. The decoders utilized are Irfanview, KaKadu and OpenJPEG and are set to the most resilient mode in order to guarantee code-stream precision.

4.2 Progression Order Testing

Recall from section 3.2 on Progression Ordering that the JPEG-2000 encoder is modified from the LRCP to RLCP progression. This modification occurs in the Tier-II of the EBCOT encoding scheme. The Coding Style Default (COD) marker segment is also changed to reflect the reordering of the precincts. The RLCP progression takes the value of one in the O_p parameter of the COD marker segment (as opposed to zero using the old LRCP progression). Figure 4.1 below shows the main

header of an encoded JPEG-2000 file with the appropriate COD marker segment.

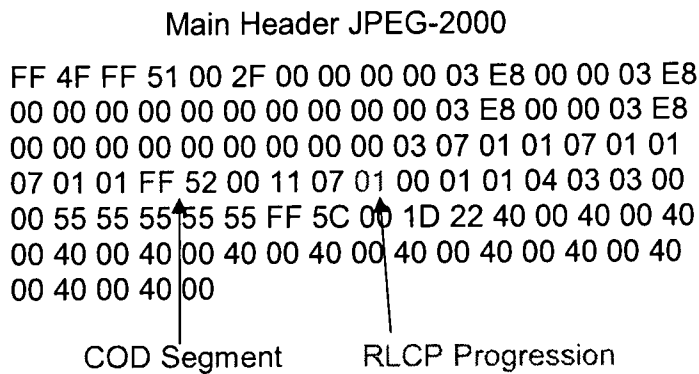


Figure 4.1: COD Marker Segment RLCP Modification Verification

4.3 Resolution Parsing Testing

The re-ordering of the precincts is verified in this section although the modifications to the code-stream occur during the encoding process and could be shown in the previous section. If the precincts are encoded in an incorrect way, the resolution parser would fail so it reasons both can be tested simultaneously.

A simple Graphical User Interface (GUI) is written to perform testing. The program opens an encoded JPEG-2000 file and trans-codes the image using the resolution parser to produce the desired resolution. It then decodes the image using one of the three decoders mentioned in section 4.1 to display the image at the resolution selected. The GUI application and results are seen in Figures 4.3.1 through 4.3.5. The

encoded image has a total of five resolution levels, and all five are tested.

The KaKadu decoder was used in these figures for display purposes.

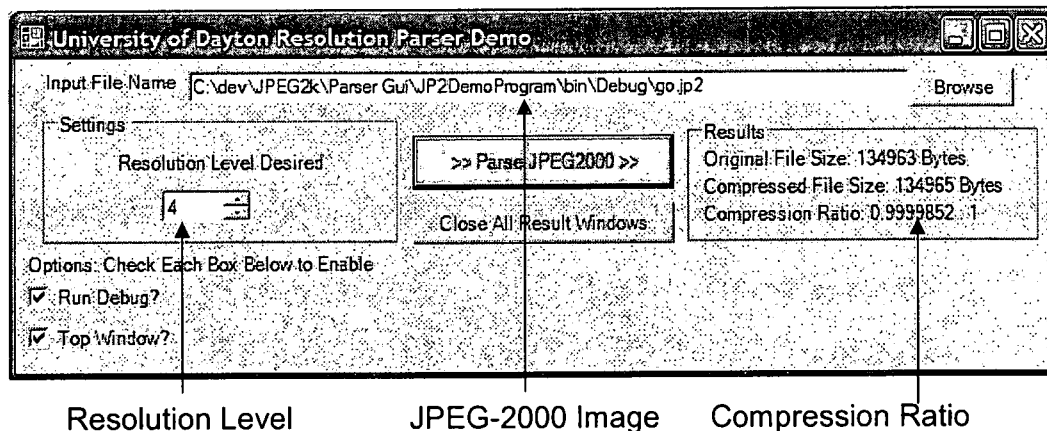
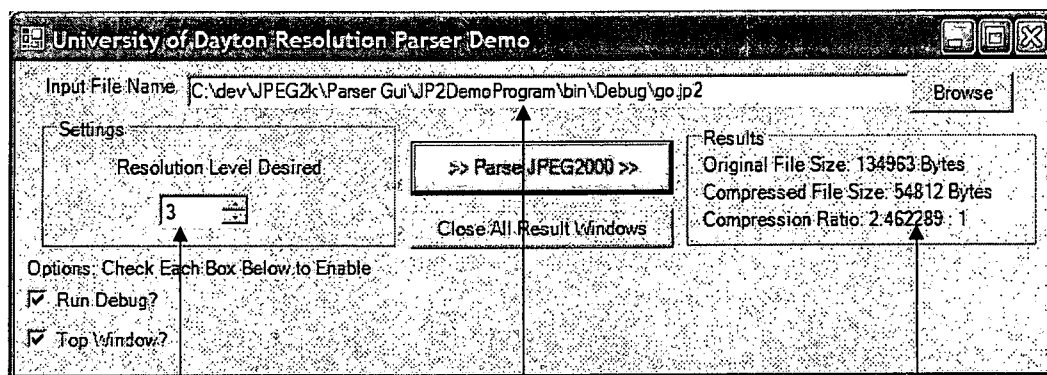


Figure 4.3.1: Resolution Parser at Resolution Level Four



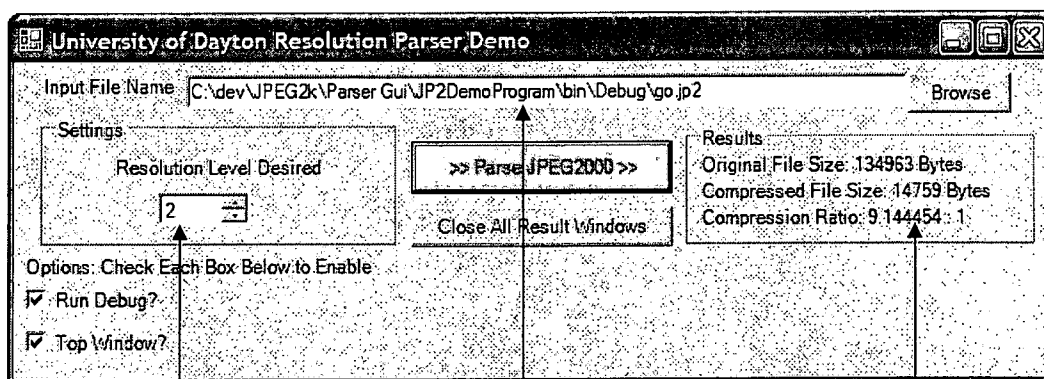
Resolution Level

JPEG-2000 Image

Compression Ratio



Figure 4.3.2: Resolution Parser at Resolution Level Three



Resolution Level

JPEG-2000 Image

Compression Ratio

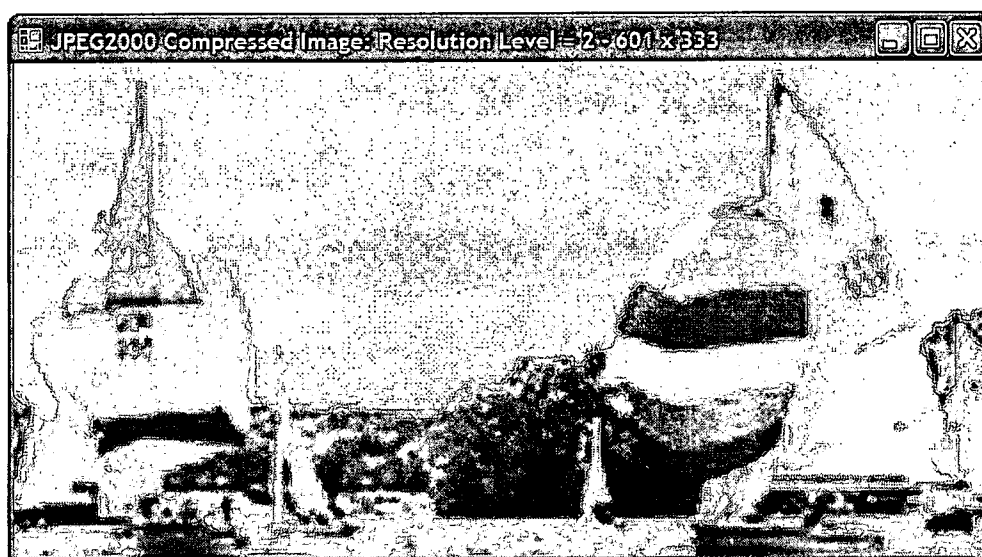
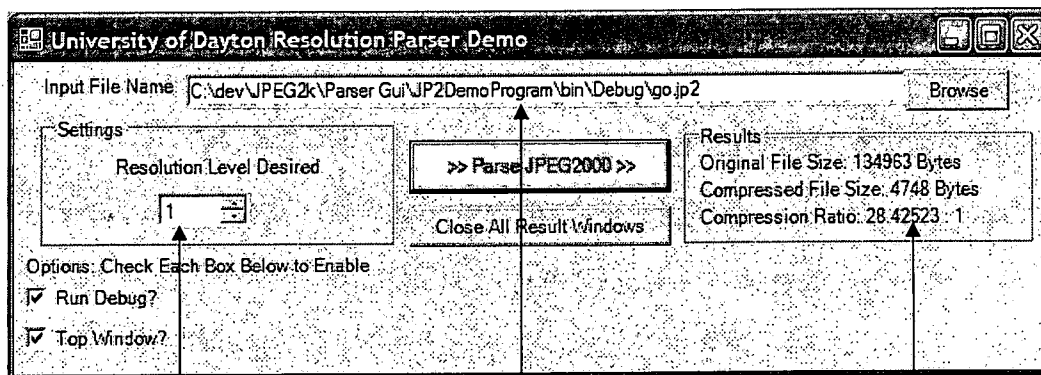


Figure 4.3.3: Resolution Parser at Resolution Level Two



Resolution Level

JPEG-2000 Image

Compression Ratio

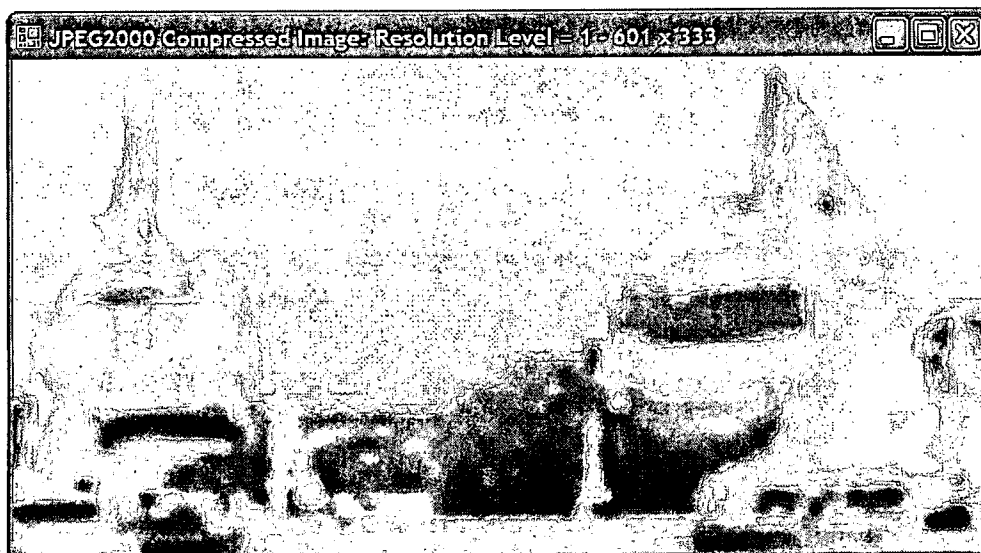
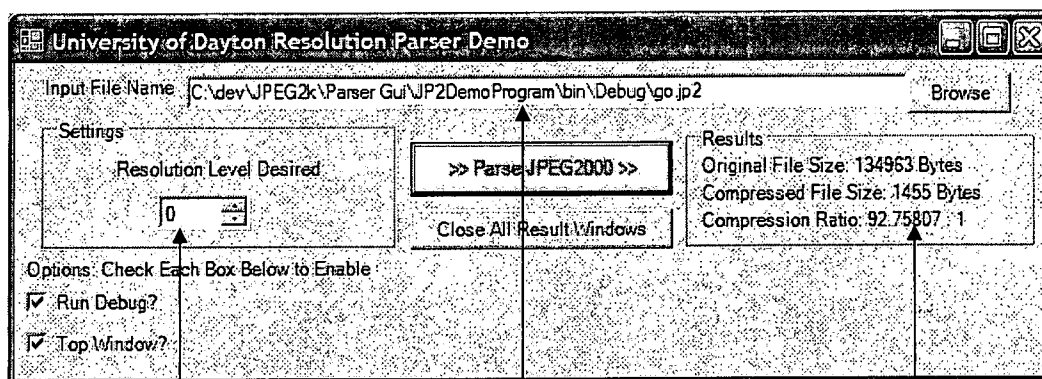


Figure 4.3.4: Resolution Parser at Resolution Level One



Resolution Level

JPEG-2000 Image

Compression Ratio



Figure 4.3.5: Resolution Parser at Resolution Level Zero

4.4 JPIP Integration Testing

The resolution parser is integrated into the JPIP standard. Using C#, a simple client is written in order to display the images at the highest resolution possible based on the user's screen resolution. The client sends the HTTP JPIP message to the JPIP server. The server parses the desired resolution and returns the valid image to the client for display.

Figure 4.4.1 shows a full image displayed at the lowest resolution so it can fit onto a 24 inch monitor. Figure 4.4.2 shows a zoomed in view of two of the buildings at an increased resolution. Figure 4.4.3 shows the targeted building at the highest quality and resolution possible.



Figure 4.4.1: Full Image at the Lowest Resolution

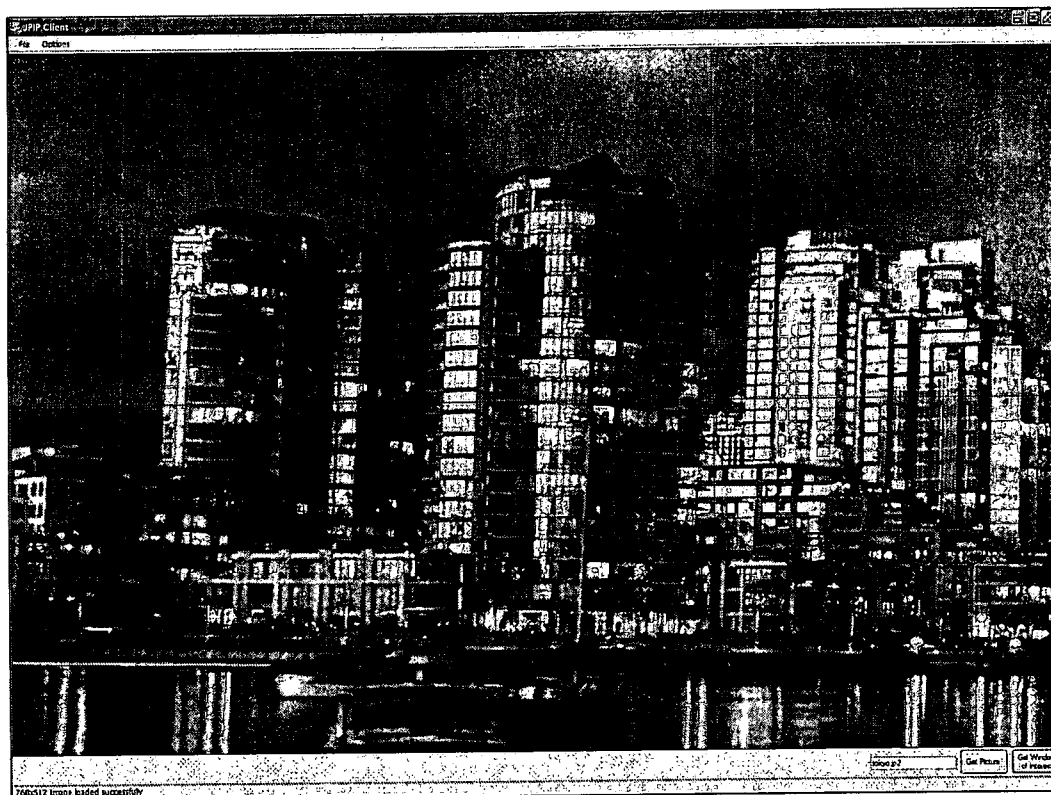


Figure 4.4.2: Higher Resolution Image

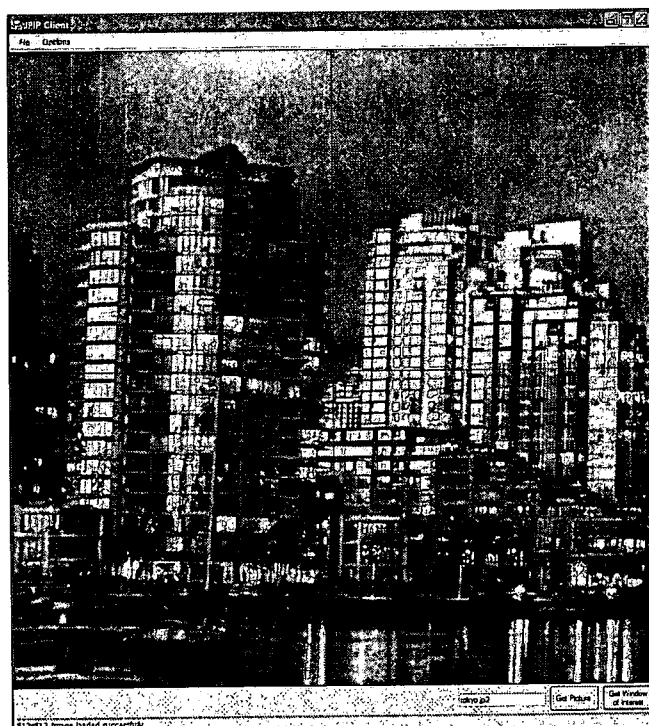


Figure 4.4.3: Highest Resolution Image

4.5 Summary of Results

The three tests prove successful. The progression mode is correctly changed from LRCP to RLCP and the resolution parser correctly extracted the desired resolution levels from the encoded code-streams as three decoders could each recognize and decode the parsed images. The parser then is integrated into a JPIP server/client interface to produce the desired resolution levels for regions of interest in encoded imagery.

CHAPTER 5

Conclusion and Future Works

5.1 Conclusion

As Figure 5.1 shows, the final product of the parser is an image with the maximum quality to fit the resolution and bandwidth requirements of the user. Many resolutions are embedded within a single JPEG-2000 code-stream allowing for dynamic resolution choices for multiple scenarios. The resolution control is handled by the JPIP server and allows for the same image to be sent to different clients at different resolutions. This application fits the need for an adaptable real-time imaging system. It adjusts to mobile users in the field with small PDA's to special surveillance ground stations with immense processing servers, while still maintaining the protocols of the JPIP standard. Also a novel approach for error detection and resilience is added to the encoder to accommodate broadcasting transmissions and insure fewer packets drops.

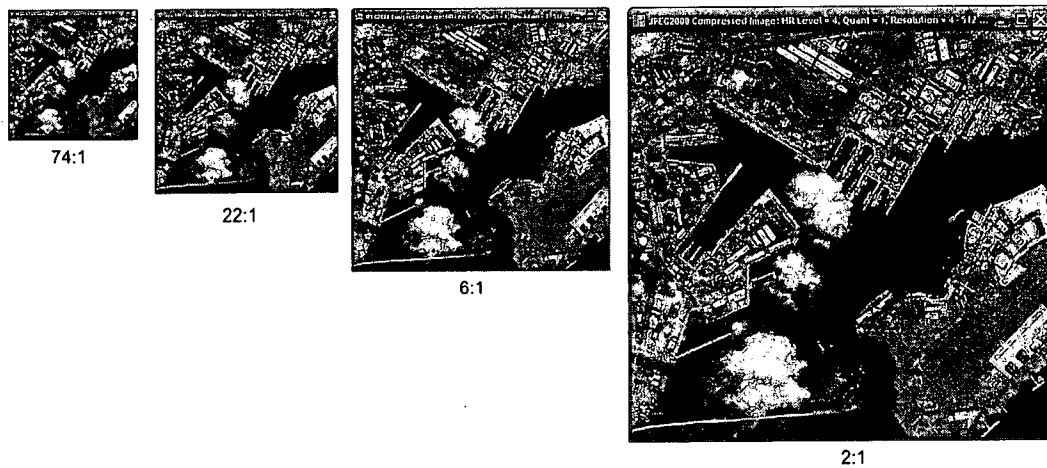


Figure 5.1: Increasing Resolution Levels with Compression Rates

5.2 Future Work

One main modification that could be added to the resolution parser to make it more robust and allow for an even further dynamic system is a spatial parser. In addition to resolution parsing, a spatial component addition would further increase the compression gain and allow for improved region of interest selection. As of now, the parser is limited to tile boundaries for spatial extraction of resolution levels.

References

- [1] ISO/IEC 15444-1, "Information Technology – JPEG2000 Image Coding System – Part 1: Core Coding System," 2000.
- [2] ISO/IEC 15444-2, "Information Technology – JPEG2000 Image Coding System: Extensions," 2004.
- [3] ISO/IEC 15444-3, "Information Technology – JPEG2000 Image Coding System: Motion JPEG2000," 2003.
- [4] ISO/IEC 15444-4, "Information Technology – JPEG2000 Image Coding System: Conformance testing," 2004.
- [5] ISO/IEC 15444-5, "Information Technology – JPEG2000 Image Coding System – Part 5: Reference Software," 2003.
- [6] ISO/IEC 15444-6, "Information Technology – JPEG2000 Image Coding System – Part 6: Compound Image File Format," 2003.
- [7] David S. Taubman and Michael W. Marcellin. *JPEG2000 Image Compression Fundamentals, Standards and Practice*. Kluwer Academic Publishers, Boston, Mass., 2002.
- [8] Michael P. Flaherty. 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, 300 College Park, Dayton, OH 45440, December 2006.
- [9] David B. Mundy. The Study and HDL Implementation of the JPEG 2000 MQ Coder. Master's thesis, University of Dayton, 300 College Park, Dayton, OH 45440, May 2007.
- [10] Tinku Archarya. VLSI Algorithms and Architectures for JPEG2000.
- [11] Michael W. Marcellin et al. An Overview of Quantization in JPEG2000. 20 April 2001.
- [12] David taubman and Robert Prandolini. Architecture, Philosophy and Performance of JPIP: Internet Protocol Standard for JPEG2000.

[13] Jin Li. *Image Compression – the Mechanics of the JPEG 2000*. Microsoft Research, Signal Processing. One Microsoft Way, Redmond, WA 98052.

[14] J. Postel. *User Datagram Protocol*. The Internet Society, August 1980.

R002594893

51
182