An Overview of JPEG-2000

Michael W. Marcellin[1], Michael J. Gormish[2], Ali Bilgin[1], Martin P. Boliek[2]

This paper appeared in *Proc. of IEEE Data Compression Conference,* pp. 523-541, 2000. When JPEG 2000 Part I went from CD to FCD the term "packet partition location" was changed to "precinct."

Abstract

JPEG-2000 is an emerging standard for still image compression. This paper provides a brief history of the JPEG-2000 standardization process, an overview of the standard, and some description of the capabilities provided by the standard. Part I of the JPEG-2000 standard specifies the minimum compliant decoder, while Part II describes optional, value-added extensions. Although the standard specifies only the decoder and bitstream syntax, in this paper we describe JPEG-2000 from the point of view of encoding. We take this approach, as we believe it is more amenable to a compact description more easily understood by most readers.

1 Introduction

As digital imagery becomes more commonplace and of higher quality, there is the need to manipulate more and more data. Thus, image compression must not only reduce the necessary storage and bandwidth requirements, but also allow extraction for editing, processing, and targeting particular devices and applications. The JPEG-2000 image compression system has a rate-distortion advantage over the original JPEG. More importantly, it also allows extraction of different resolutions, pixel fidelities, regions of interest, components, and more, all from a single compressed bitstream. This allows an application to manipulate or transmit only the essential information for any target device from any JPEG 2000 compressed source image. JPEG-2000 has a long list of features, a subset of which are:

- State-of-the-art low bit-rate compression performance
- Progressive transmission by quality, resolution, component, or spatial locality
- Lossy and lossless compression (with lossless decompression available naturally through all types of progression)
- Random (spatial) access to the bitstream
- Pan and zoom (with decompression of only a subset of the compressed data)
- Compressed domain processing (e.g., rotation and cropping)
- Region of interest coding by progression
- Limited memory implementations.

The JPEG-2000 project was motivated by Ricoh's submission of the CREW algorithm [1,2] to an earlier standardization effort for lossless and near-lossless compression (now known as JPEG-LS). Although LOCO-I [3] was ultimately selected as the basis for JPEG-LS, it was recognized that CREW provided a rich set of features

[1] Department of ECE, University of Arizona, Tucson, AZ, {marcellin,bilgin}@ece.arizona.edu
[2] Ricoh Silicon Valley, Menlo Park, CA, {gormish,boliek}@rsv.ricoh.com

worthy of a new standardization effort. Based on a proposal authored largely by Martin Boliek [4], JPEG-2000 was approved as a new work item in 1996, and Boliek was named as the project editor. Many of the ideas in JPEG-2000 are inspired by the work of [1, 5, 6, 7]. Also in 1996, Dr. Daniel Lee of Hewlett-Packard was named as the Convener of ISO/IEC JTC1/SC29/WG1 (the Working Group charged with the development of JPEG-2000, hereinafter referred to as simply WG1).

2 The JPEG-2000 Development Process

A Call for Technical Contributions was issued in March 1997 [8], requesting compression technologies be submitted to an evaluation during the November 1997 WG1 meeting in Sydney, Australia. Further, WG1 released a CD-ROM containing 40 test images to be processed and submitted for evaluation. For the evaluations, it was stipulated that compressed bitstreams and decompressed imagery be submitted for six different bitrates (ranging from 0.0625 to 2.0 bits per pixel (bpp)) and for lossless encoding. Eastman Kodak computed quantitative metrics for all images and bit rates, and conducted a subjective evaluation of 18 of the images (of various modalities) at three bit-rates in Sydney using evaluators from among the WG1 meeting attendees. The imagery from 24 algorithms was evaluated by ranking the perceived image quality of hard-copy prints.

Although the performance of the top third of the submitted algorithms were statistically close in the Sydney evaluation, the wavelet/trellis coded quantization (WTCQ) algorithm, submitted by SAIC and the University of Arizona (SAIC/UA), ranked first overall in both the subjective and objective evaluations. In the subjective evaluation, WTCQ ranked first (averaged over the entire set of evaluated imagery) at 0.25 and 0.125 bpp, and second at 0.0625 bpp. In terms of RMS error averaged over all images, WTCQ ranked first at each of the six bitrates. Based on these results, WTCQ was selected as the reference JPEG-2000 algorithm at the conclusion of the meeting. It was further decided that a series of "core experiments" would be conducted to evaluate WTCQ and other techniques in terms of the JPEG-2000 desired features and in terms of algorithm complexity.

Results from the first round of core experiments were presented at the March 1998 WG1 meeting in Geneva. Based on these experiments, it was decided to create a JPEG-2000 "Verification Model" (VM) which would lead to a reference implementation of JPEG-2000. The VM would be the software in which future rounds of core experiments would be conducted, and the VM would be updated after each JPEG-2000 meeting based on the results of core experiments. SAIC was appointed to develop and maintain the VM software with Michael Marcellin as the head of the VM Ad Hoc Group. Eric Majani (Canon-France) and Charis Christopoulos (Ericsson-Sweden) were also named as co-editors of the standard at that time. Results from round 1 core experiments were selected to modify WTCQ into the first release of the VM (VM 0).

2.1 The WTCQ Algorithm

The basic ingredients of the WTCQ algorithm are: the discrete wavelet transform, TCQ [9, 10] (using step sizes chosen via a Lagrangian rate allocation procedure), and binary arithmetic bitplane coding. The *embedding principle* [5, 6, 1, 7, 11, 12, 13], asserts the encoded bitstream should be ordered in a way that maximally reduces MSE

per bit transmitted. In WTCQ embedding is provided by the bitplane coding similar to that of [1]. The bitplane coding operates on TCQ indices (trellis quantized wavelet coefficients) in a way that enables successive refinement. This is accomplished by sending bitplanes in decreasing order from most- to least-significant. To exploit spatial correlations within bitplanes, spatial context models are used. In general, the context can be chosen within a subband and across subbands. The WTCQ bitplane coder avoids the use of inter-subband contexts to maximize flexibility in scalable decoding, and to facilitate parallel implementation. WTCQ also includes a "binary mode," a classification of coefficients, multiple decompositions (dyadic, packet, and others), and difference images to provide lossless compression. A more complete description of WTCQ can be found in [14].

2.2 VM 0 – VM 2

Additions and modifications to VM 0 continued over several meetings, with refinements contributed by many WG1 members. VM 2.0 supported user specified floating point and integer transforms, as well as user specified decompositions (dyadic, uniform, etc.). As a simpler alternative to the Lagrangian rate allocation, a fixed quantization table ("Q-table") was included. This is analogous to the current JPEG standard [15]. When a Q-table is used, precise rate control can still be obtained by truncating the (embedded) bitstream. In addition to TCQ, scalar quantization was included in VM 2.

For integer wavelets, scalar quantization with step size 1 was employed (i.e., no quantization), which allowed progression to lossless in the manner of CREW or SPIHT [16] (using the S+P transform). Rate control for integer wavelets was accomplished by embedding, and lossless compression was available naturally from the fully decoded embedded bitstream. Other features, such as tiling, region of interest coding/decoding (University of Maryland, Mitsubishi, and Ericsson), error resilience (Motorola-Switzerland, TI, Sarnoff, UBC), approximate wavelet transforms with limited spatial support (Motorola-Australia, Canon-France) were added to the VM, often from other original contributions to the Sydney meeting. For complete description of these and other technologies see [17].

Along with the additions described above, several refinements were made to the bitplane coder. The major changes were the de-interleaving of bitplanes and improvements to the context modeling. Within a given bitplane of each subband, the bits were "de-interleaved" into three "sub-bitplanes" of the following types: 1) bits predicted to be newly "significant," 2) "refinement" bits, and 3) bits predicted to remain "insignificant." The idea of sub-bitplanes was first presented in [13] for use with Golomb coding, and is motivated by rate-distortion concerns [11, 12]. It is desirable to have the bits with the steepest rate-distortion slopes appear first in an embedded bitstream.

The de-interleaving employed in VM 2 was adapted from [13] for use with arithmetic coding, and did not use the parent index to predict significance. Thus, the VM 2 bitplane coder has no inter-subband dependencies such as those used in [13] and in the zerotree based schemes of [5, 7]. This allows for a certain amount of parallelism and enables the type of progression present in the encoded bitstream to be changed without decoding (see the description of parsing in Section 5).

As in VM 0, all coding was carried out using context dependent binary arithmetic coding. The particular arithmetic coder employed is described in [18]. It should be noted that, when encoding a particular bit, neither significance prediction, nor context modeling stages can use any information that would not be available at the decoder when that bit needs to be decoded. Thus, for those wavelet coefficients that are non-causal with respect to the scan pattern, only information from more significant bitplanes is used.

2.3 VM 3 – VM 5

At the November 1998 WG1 meeting in Los Angeles, David Taubman (then at Hewlett-Packard) presented EBCOT (embedded block coding with optimized truncation) [19, 20]. EBCOT included the idea of dividing each subband into rectangular blocks of coefficients and performing the bitplane coding independently on these "code-blocks" (rather than entire subbands as in previous VMs). This partitioning reduces memory requirements in both hardware and software implementations, as well as providing a certain degree of (spatial) random access to the bitstream. EBCOT also included an efficient syntax for forming the sub-bitplane data of multiple code-blocks into "packets," which taken together form quality "layers."

Tremendous flexibility in the formation of packets and layers is left to the implementer of an encoder. The default policy in the VM encoder is to place in each layer the sub-bitplanes (among all sub-bitplanes not yet included in previous layers) with steepest rate-distortion slope (as estimated in the encoder). This policy aims to minimize the MSE at each point in the embedded bitstream and improves the MSE performance over a simple "round robin" ordering as implemented in VM 2. Other policies have been explored as well. One particularly interesting policy modifies the distortion estimates of each sub-bitplane consistent with visual masking properties. Thus, code-blocks in regions where more distortion can be tolerated (visually) are de-emphasized in the bitstream formation. Even when this masking policy is employed, progressive transmission eventually results in lossless decompression (when integer wavelets are employed). The policy has little effect on the ultimate lossless file size (bitrate), but can have dramatic impact on the visual quality for partial (embedded) decoding at lower rates.

EBCOT was adopted for inclusion in VM 3 at the Los Angeles meeting. Taubman re-implemented the entire VM in an object-oriented manner at that time. In subsequent VM's, the block coder was refined to include only 3 passes (EBCOT used 4) similar to those of VM 2. Subsequent VM's were also modified with more "hardware friendly" context modeling, and scan pattern (within code-blocks) [21].

At the March 1999 WG1 meeting in Korea, the MQ-coder (submitted by Mitsubishi) was adopted as the arithmetic coder for JPEG-2000. This coder is functionally similar to the QM-coder available as an option in the original JPEG standard. The MQ-coder has some useful bitstream creation properties, is used in the JBIG-2 standard, and should be available on a royalty and fee free basis for ISO standards. In fact, one goal of WG1 has been the creation of a Part I which could be used entirely on a royalty and fee free basis. It is felt this is essential for the standard to gain wide acceptance as an interchange format (witness the large difference in utilization of JPEG with Huffman coding and JPEG with arithmetic coding).

At the same time as changes were being made to the internal coding algorithms, the syntax wrapping the compressed data was developed. This syntax is made up of a

sequence of markers, compatible with those of the original JPEG [15], with features added by Ricoh and Aerospace Corporation to allow the identification of relevant portions of the compressed data.

While the bitstream syntax provides all the data necessary for the decoder to recreate the input pixel array, applications often require additional information not present in the bitstream. One Annex of the JPEG-2000 standard contains an optional minimal file format to include information such as the color space of the "pixels" and intellectual property (copyright) information for the image. Hopefully the inclusion of this annex will prevent the proliferation of proprietary file formats that happened with the original JPEG. This optional file format is extensible and Part II will define storage of many additional types of "metadata."

3 Final Standardization

The document describing the JPEG-2000 Part I decoder reached "Committee Draft" (CD) status in December 1999. Although technical changes are still possible, they now require support of a "national body." In April 2000 the draft may obtain "Final Committee Draft" (FCD) form, and if work proceeds at the maximum possible rate under ISO rules, "Final Draft International Standard" (FDIS) in August 2000, and finally JPEG-2000 may become an "International Standard" (IS) in December 2000. Part II is scheduled to be eight months behind Part I, becoming an International Standard in July 2001.

It is worth noting that the standard specifies only the decoder and bitstream syntax. Although informative descriptions of some encoding functions will be provided in the text of the standard, there are no requirements that the encoder perform compression in any prescribed manner. This leaves room for future innovations in encoder implementations.

For the purpose of interchange it is important to have a standard with a limited number of options, so decoders in browsers, printers, cameras, or PDAs can be counted on to implement all the options and an encoded image will be displayable by all devices. For this reason some choices have been limited in the standard. Part I, therefore, will describe the minimal decoder required for JPEG-2000, which should be used to provide maximum interchange. However, there are applications for image compression where interchange is less important than other requirements (e.g., ability to handle a particular type of data). Part II will consist of optional "value added" technologies, not required of all implementations. Of course, images encoded with Part II technologies usually will not be decodable by Part I decoders. Table 1 lists the various components of the compression system and the extensions likely for Part II. For example, Part I will require one floating-point wavelet (9,7), and one integer wavelet (3,5), while Part II will allow multiple wavelets including "user defined."

Other items which are important for the adoption of JPEG-2000 did not fit properly in either the "minimum decoder" of Part I, or the "extensions" of Part II. Motion JPEG has been a commonly used method of editing high quality video (e.g., in production studios) without the existence of an ISO standard. Part III of JPEG-2000 will be "Motion JPEG-2000." Part II of the original JPEG was a set of compliance tests to ensure quality implementation of the standard. WG1 plans to provide JPEG-2000 compliance tests in Part IV. Finally, the key to success of the JPEG-2000 standard may well be the

availability of high quality free software. The JJ2000 group (Cannon France, Ericsson, EPFL) has produced a Java implementation for standard promotion, and UBC has announced the intention to release C software. WG1 has started a Part V of the standard for encouraging development of free software.

Table 1: Division of the Standard Between Part I and Part II.

Technology	Part I	Part II
Bitstream	Fixed and variable length markers.	New markers can be skipped by a Part I decoder.
File format	Optional. Provide intellectual property (e.g. copyright) information, color or tone-space for image, general method of including metadata.	Allow metadata to be interleaved with coded data. Define types of metadata.
Arithmetic Coder	MQ-coder.	Same?
Coefficient Modeling	Independent coding of fixed size blocks within subbands. Division of coefficients into 3 sub-bitplanes. Grouping of sub-bitplanes into "layers."	Special models for binary or graphic data?
Quantization	Scalar quantizer with dead-zone, truncation of code-blocks.	Trellis Coded Quantization.
Transformation	Low complexity (5,3) and high performance Daubechies (9,7). Mallat decomposition.	Many more filters, perhaps "user-defined" filters. Packet and other decompositions.
Component decorrelation	Reversible component transform (RCT), YCrCb transform.	Arbitrary point transform or reversible wavelet transform across components.
Error Resilience	Resynchronization markers.	Fixed length entropy coder, repeated headers.
Bit-stream Ordering	Progressive by tile-part, then SNR, or resolution, or component.	Out of order tile-parts.

4 JPEG-2000 Coding Engine

4.1 Tiles and Component Transforms

In what follows, we provide a description of the JPEG-2000 coding engine. Our goal is to illuminate the key concepts at a sufficient level to impart a fundamental understanding of the algorithm without dwelling too much on details. In the standard, an image can consist of multiple components (e.g., RGB) each possibly subsampled by a different factor. Conceptually, the first algorithmic step is to divide the image into rectangular, non-overlapping tiles on a regular grid. Arbitrary tile sizes are allowed, up to and including the entire image (i.e., no tiles). Components with different subsampling factors are tiled with respect to a high resolution grid, which ensures spatial consistency of the resulting tile-components. Each tile of a component must be of the same size, with the exception of tiles around the border (all four sides) of the image.

When encoding an image having multiple components such as RGB, a point-wise decorrelating transform may be applied across the components. Two transforms are

defined in Part I of the standard: 1) the YCrCb transform commonly used with original JPEG images, and 2) the Reversible Component Transform (RCT) which provides similar decorrelation, but allows lossless reconstruction of all components [22]. After this transform all components are treated independently (although different quantization is possible with each component, as well as joint rate allocation across components). For the sake of simplicity, we now describe the JPEG-2000 algorithm with respect to a single tile of a single component (e.g., gray level) image.

4.2 Partitions, Transforms, and Quantization

Given a tile, an L-level dyadic (pyramidal) wavelet transform is performed using either the (9,7) floating point wavelet [23], or the (5,3) integer wavelet [24]. Progression is possible with either wavelet but the (5,3) must be used if it is desired to progress to a lossless representation. Although we describe the algorithm here in terms of processing on an entire tile, more memory efficient implementations are possible using sliding-window [25] or block-based transform techniques [26, 27].

From an L-level transform it is natural to reconstruct images at L+1 different "sizes," or "resolutions." We refer to the lowest frequency subband (LFS) as resolution 0, and the original image as resolution L. The LFS is also referred to as the *resolution-level 0 subband*. The three subbands needed to augment resolution j into resolution j+1 are referred to collectively as *resolution-level j+1 subbands*.

After transformation, all wavelet coefficients are subjected to uniform scalar quantization employing a fixed dead-zone about the origin. This is accomplished by dividing the magnitude of each coefficient by a quantization step size and rounding down. One quantization step size is allowed per subband. These step sizes can be chosen in a way to achieve a given level of "quality" (as in many implementations of JPEG), or perhaps in some iterative fashion, to achieve a fixed rate. The default behavior of the VM is to quantize each coefficient rather finely, and rely on subsequent truncation of embedded bitstreams to achieve precise rate control. The standard places no requirement on the method used to select quantization step sizes. When the integer wavelet transform is employed, the quantization step size is essentially set to 1.0 (i.e., no quantization). In this case, precise rate control (or even fixed quality) is achieved through truncation of embedded bitstreams.

After quantization, each subband is subjected to a "packet partition." This packet partition divides each subband into regular non-overlapping rectangles. Three spatially consistent rectangles (one from each subband at a given resolution level) comprise a packet partition location. The packet partition provides a medium-grain level of spatial locality in the bitstream for the purpose of memory efficient implementations, streaming, and (spatial) random access to the bitstream, at a finer granularity than that provided by tiles. Finally, code-blocks are obtained by dividing each packet partition location into regular non-overlapping rectangles. The code-blocks are then the fundamental entities for the purpose of entropy coding.

To recap, an image is divided into tiles and each tile is transformed. The subbands (of a tile) are divided into packet partition locations. Finally, each packet partition location is divided into code-blocks. This situation is illustrated in Figure 1. This figure depicts a packet partition of the subbands at resolution level 2 (of a 3-level dyadic wavelet transform of one tile). Also shown is the division of one packet partition location into twelve code-blocks.

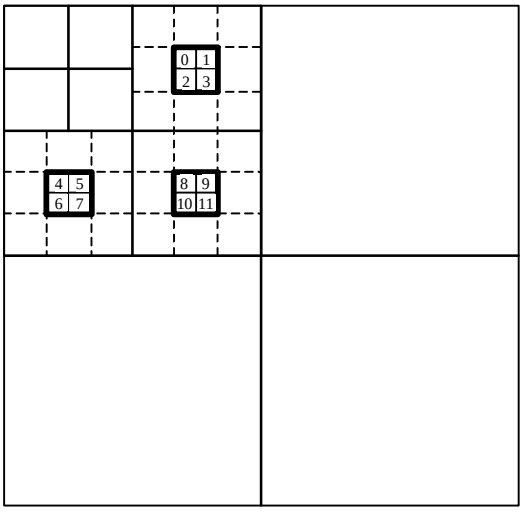

Figure 1: Twelve code-blocks of one packet partition location at resolution level 2 of a 3-level dyadic wavelet transform. The packet partition location is emphasized by heavy lines.

4.3 Block Coding

Entropy coding is performed independently on each code-block. This coding is carried out as context-dependent, binary, arithmetic coding of bitplanes. Consider a quantized code-block to be an array of integers in sign-magnitude representation, then consider a sequence of binary arrays with one bit from each coefficient. The first such array contains the most significant bit (MSB) of all the magnitudes. The second array contains the next MSB of all the magnitudes, continuing in this fashion until the final array which consists of the least significant bits of all the magnitudes. These binary arrays are referred to as bitplanes.

The number of bitplanes in a given code-block (starting from the MSB) which are identically zero is signaled as side information, as described later. So, starting from the first bitplane having at least a single 1, each bitplane is encoded in three passes (referred to as sub-bitplanes). The scan pattern followed for the coding of bitplanes, within each code-block (in all subbands), is shown in Figure 2. This scan pattern is basically a column-wise raster within stripes of height four. At the end of each stripe, scanning continues at the beginning (top-left) of the next stripe, until an entire bitplane (of a code-block) has been scanned.

The prescribed scan is followed in each of the three coding passes. The decision as to which pass a given bit is coded in is made based on the "significance" of that bit's location and the significance of neighboring locations. A location is considered significant if a 1 has been coded for that location (quantized coefficient) in the *current or previous* bitplanes.

The first pass in a new bitplane is called the significance propagation pass. A bit is coded in this pass if its location is not significant, but at least one of its eight-connected neighbors is significant. If a bit is coded in this pass, and the value of that bit is 1, its location is marked as significant for the purpose of coding subsequent bits in the *current*

and subsequent bitplanes. Also, the sign bit is coded immediately after the 1 bit just coded. The second pass is the magnitude refinement pass. In this pass, all bits from locations that became significant in a *previous* bitplane are coded. The third and final pass is the clean-up pass, which takes care of any bits not coded in the first two passes.

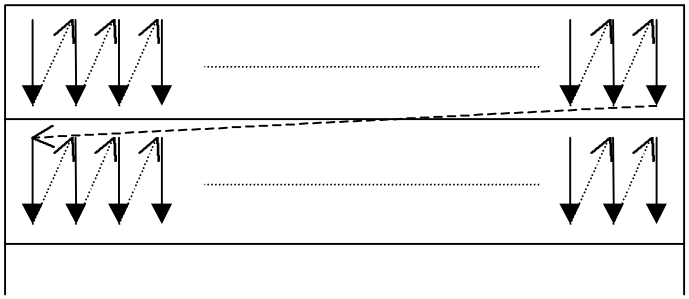

Figure 2: Scan pattern for bitplane coding.

Table 2 shows an example of the coding order for the quantized coefficients of one 4-sample column in the scan. This example assumes all neighbors not included in the table are identically zero, and indicates in which pass each bit is coded. As mentioned above, the sign bit is coded after the initial 1 bit and is indicated in the table by the + or – sign. Note that the very first pass in a new block is always a clean-up pass because there can be no predicted significant, or refinement bits.

Table 2: Example of Sub-Bitplane Coding Order.

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

All coding is done using context dependent binary arithmetic coding. The arithmetic coder employed is the MQ-coder as specified in the JBIG-2 standard [28]. The coding for the first and third passes is identical, with the exception that run coding is sometimes employed in the third pass. Run coding occurs when all four locations in a column of the scan are insignificant and each has only insignificant neighbors. A single bit is then coded to indicate whether the column is identically zero or not. If not, the length of the zero run (0 to 3) is coded, reverting to the "normal" bit-by-bit coding for the location immediately following the 1 that terminated the zero run. The sign and magnitude refinement bits are also coded using contexts designed specifically for that purpose.

For brevity, the computation to determine each context is not included here. However, unlike JBIG or JBIG-2 which use thousands of contexts, JPEG-2000 uses no more than nine contexts to code any given type of bit (i.e., significance, refinement, etc.). This allows extremely rapid probability adaptation and decreases the cost of independently coded segments.

Before leaving this section, we mention a few issues regarding the arithmetic coding. The context models are always reinitialized at the beginning of each code-block. Similarly, the arithmetic codeword is always terminated at the end of each code-block (i.e., once, at the end of the last sub-bitplane). The best performance is obtained when these are the only reinitializations/terminations. It is allowable however, to reset/terminate at the beginning/end of every sub-bitplane within a code-block. This frequent reset/termination, plus optionally restricting context formation to include data from only the current and previous "scan-stripes" is sufficient to enable parallel encoding of all sub-bitplanes within a code-block (of course, parallel encoding of the code-blocks themselves is always possible). Reset/termination strategies can also impact the error resilience of the decoder. Finally, "selective arithmetic coder bypass" can be used to significantly reduce the number of symbols arithmetically coded. In this mode, the third coding pass of every bitplane employs arithmetic coding, as before. However, after the fourth bitplane is coded, the first and second passes are included as raw (uncompressed) data. For natural imagery, all of these modifications produce a surprisingly small loss in compression efficiency. For other imagery types (graphics, compound documents, etc.) significant losses can be observed.

4.4 Packets and Layers

The compressed bitstreams associated with some number of sub-bitplanes from each code-block in a packet partition location are collected together to form the body of a "packet." The body of a packet is preceded by a packet header. The packet header contains: block inclusion information for each block in the packet (some blocks will have no coded data in any given packet); the number of completely zero bitplanes for each block; the number of sub-bitplanes included for each code-block; and the number of bytes used to store the coded sub-bitplanes of each block. It should be noted that the header information is coded in an efficient and embedded manner itself. The data contained in a packet header supplements data obtained from previous packet headers (within the same packet partition location) in a way to just enable decoding of the current packet. A discussion of this process is beyond the scope of this paper, for more details see [20].

Figure 3 depicts one packet for the packet partition location illustrated in Figure 1. Note that each of the twelve code-blocks can contribute a different number of sub-bitplanes (possibly zero) to the packet, and empty packet bodies are allowed.

Packet Header	n_0 sub-bitplanes from code-block 0	n_1 sub-bitplanes from code-block 1	.....	n_{11} sub-bitplanes from code-block 11

Figure 3: The composition of one packet for the packet partition location of Figure 1.

A packet can be interpreted as one quality increment for one resolution level at one spatial location (packet partition locations correspond roughly to spatial locations). A "layer" is then a collection of packets: one from each packet partition location of each resolution level. A layer then can be interpreted as one quality increment for the entire image at full resolution.

As noted above, there is no restriction on the number of sub-bitplanes contributed by each code-block to a given packet (layer). Thus, an encoder can format packets for a variety of purposes. For instance, consider the case when progression and the features provided by the packet partition are not of interest. The packet partitions can be set larger than the subbands (turned off), and all sub-bitplanes from all blocks can be included in a single packet per resolution layer. This provides the most efficient compression performance, as the packet header information is minimized under this scenario.

On the other hand, if progression by quality (embedding) is desired, a very small number of sub-bitplanes can be included in each packet. The current VM supports a generic scalable setting which includes approximately 50 layers. In this case, on average, less than 1 sub-bitplane per code-block contribute to each packet. The strategy employed by the VM (many others are possible) to form packets in the 50 layer case is based on rate distortion theory. Each packet is constructed to include all sub-bitplanes with (estimated) rate-distortion slope above a given threshold. This threshold is adjusted to achieve the desired size (bit-rate) for the aggregate of all packets within the layer under construction. This provides very fine-grained quality (rate) progression at the expense of some additional overhead due to the (numerous) packet headers. Nevertheless, the VM provides start-of-the-art compression performance even with 50 layers.

5 JPEG-2000 Bitstream

JPEG-2000 provides better rate-distortion performance, for any given rate, than the original JPEG standard. However, the largest improvements are observed at very high and very low bitrates. The improvements in the "near visually lossless" realm are more modest (approximately 20%). Thus, widespread adoption of the new standard will likely be based on the JPEG-2000 feature set. While JPEG provided different methods of generating progressive bitstreams, with JPEG-2000 the progression is simply a matter of the order the compressed bytes are stored in a file. Furthermore, the progression can be changed, additional quantization can be done, or a server can respond only with the data desired by a client, all without decoding the bitstream.

5.1 Progression

There are four basic dimensions of progression in the JPEG-2000 bitstream: resolution, quality, spatial location, and component. Different types of progression are achieved by the ordering of packets within the bitstream. Although tiles provide an important mechanism for spatial progression, we assume in what follows (for simplicity) that the image consists of a single tile. Each packet is then associated with one component (say i), one layer (j), one resolution level (k), and one packet partition location (m). A bitstream for a color image having the usual type of progression by SNR (embedded) can be constructed by writing the packets using four nested loops. The innermost loop is partition location, followed by resolution level, followed by

component, with the outermost loop being by layer. For progressive by resolution, the order of nesting could be by partition location, layer, component, and resolution level. Another interesting progression results from making the outermost loop in the nesting "by component". The progression can then be by SNR or resolution for a gray scale image, with color information being added last. Similarly, spatial progression (or streaming) can be achieved by placing the packet partition location outmost in the nesting.

Finally, we note that the progression type can be changed at various places within the bitstream. For example, it is possible to progress by SNR at a given (reduced) resolution, then change to progression by SNR at a higher resolution. The packets included in the bitstream will then be those needed in order for the higher resolution subbands to "catch up" to the current layer of the lower resolution image. This change in progression allows an icon to be displayed first, then a screen resolution image, and finally if needed a print resolution image. With a typical 5 level transform, a 1024 by 1024 pixel print resolution image can provide a 256 by 256 screen resolution image, or a 32 by 32 icon. Progression by layer at each resolution allows the best possible image to be displayed at each resolution while receiving data over a slow connection.

As discussed previously, each layer provides more bits of some of the wavelet coefficients. The role of layers in providing progression by SNR has been detailed above. However, layering is a much more powerful concept. The layers need not be designed specifically for optimal SNR progression. For example, JPEG-2000 does not explicitly define a method of subsampling color components as JPEG does (JPEG provides subsampling on color components as a means to reduce computational complexity, and because it provides quantization the human visual system is unlikely to notice). A JPEG-2000 encoder, could place all the high frequency bands of the color components in the last layer. Discarding the last layer, would then have the same effect as subsampling in JPEG. A decoder which did not receive high frequency subbands could use a simplified transform to save computational complexity. Layers are of course much more general than subsampling. For images with significant color edges, some bits of the color coefficients might be saved in earlier layers.

5.2 Parsing

Even though a JPEG-2000 bitstream can be stored in any reasonable desired order, it can of course, only exist in one order at a time. However, because the coded data within packets are identical regardless of the progression type chosen, it is trivial to change the order, or to extract any required data from the bitstream.

The JPEG-2000 bitstream contains markers which identify the progression type of the bitstream. Other markers may be written which store the length of every packet in the bitstream. To change a bitstream from progressive by resolution to progressive by SNR, a parser can read all the markers, change the type of progression in the markers, write the lengths of the packets out in the new order, and write the packets themselves out in the new order. There is no need to run the MQ-coder, the context model, or even decode the block inclusion information. The complexity is only slightly higher than a pure copy operation.

Likewise, when sending a color image to a grayscale printer, there is no point in sending color information. A parser can read the markers from a 3 component file, and write markers for a one component file, and discard all packets containing color

components. Similarly, while editing, a compressed image might be stored at 2 bpp or even losslessly. If 2000 images are to be distributed on a CD-ROM, the layers contributing the least to quality can be discarded across the image set, until the required size is reached. Fifty layers provide enough information to extract almost any desired bitrate at any desired resolution.

5.3 Spatial Accessibility

All of the operations described in the previous section as "parsing" from one file to another file, could be performed on a server in response to requests, and the "parsed" bitstream could be sent out over a serial line instead of writing a new file. However, in addition to the whole image operations described previously, a client may wish to obtain compressed data for only a particular spatial portion of an image.

If the regions of interest (ROI) are known in advance, i.e. at encode time, JPEG-2000 provides additional methods of providing greater image quality in the foreground vs. the background. First, all of the code-blocks which contain coefficients affecting the ROI can be identified, and the bitplanes of those coefficients can be stored in higher layers relative to other coefficients. Thus a layer progressive bitstream can naturally send the ROI with higher quality (earlier in the bitstream) than the background. It should be noted that fully lossless encoding of the entire image is still possible (with no loss in compression efficiency over the case without ROI's) when the (5,3) integer wavelet is employed.

In addition, an explicit ROI can be defined and those coefficients which affect the ROI can be shifted and coded as if they were in their own set of bitplanes. For an encoder, this allows individual coefficients to be enhanced rather than entire code-blocks (which must have the same set of sub-bitplanes included in each code-block without explicit ROI). The decoder does not need to calculate which coefficients have been shifted, it simply detects those coefficients which have bitplanes shifted by the encoder and shifts them down to the level of the other coefficients before the inverse wavelet transform. Fully lossless encoding is still possible, but with some loss in compression efficiency.

If the regions of interest are not known at encode time, there are still several methods for a "smart" server to provide exactly the right data to a client requesting a specific region. The simplest method to provide access to spatial regions of the image (which are not known at encode time) is for the encoder to tile the image. Since tiling divides the image spatially any region desired by the client will lie within one or more tiles. Tiles as small as 64 by 64 are useable although tiles this small increase the bitrate noticeably. Tiles over 256 by 256 samples have almost no compression performance impact (but offer less flexible access for small regions). All of the parsing operations described previously on the whole image can selectively be applied to specific tiles. Other tiles could be discarded, or transmitted at a much lower quality (more of the data could be parsed). The bitstream contains the length of each tile (always in the tile header and optionally grouped in the main header), so it is always possible to locate the desired tiles with minimal complexity. Similarly, packet partitions can be extracted from the bitstream for spatial access. The length information is still stored in the tile header, and the data corresponding to a packet partition location are easily extracted. However, due to the filter impulse response lengths, care must taken to extract all data required to decode the region of interest.

Finer grain access is possible by parsing individual code-blocks. As in the case of packet partition locations, it is necessary to determine which code-blocks affect which pixel locations (a single pixel can effect four different code-blocks within each subband and each resolution and each component). The correct packets containing these code-blocks can be determined from the progression order information. Finally, the location of the compressed data for the code-blocks can be determined by decoding the packet headers. All of this is substantially more difficult than identifying entire tiles of interest, but substantially easier than operating the arithmetic coder and context model to decode the data.

5.4 Image Editing and Compression

All uncompressed tiled image formats allow regions of an image to be edited, and only those tiles affected need to be rewritten to disk. With compression the compressed size of an edited tile can change. Because of the flexibility in quantization in JPEG-2000 it is possible to truncate an edited tile to fit in the previous size. Alternatively, Part II will allow out of order tiles within the bitstream so an edited tile could be rewritten at the end of the bitstream

The main header of a JPEG-2000 bitstream of course contains the width and height of the image, but it also contains a horizontal and vertical offset for the start of the image. This allows the image to be cropped (to a sub-rectangle of the original) without requiring a forward and inverse wavelet transform for recompression. In fact, all tiles inside the newly cropped image need not be changed at all, and tiles on the edge of the new image need only have the code-blocks on the edges recoded, and new tile headers and packet headers written to the bitstream (no wavelet transform).

JPEG-2000 Part I allows 90, 180, and 270 degree rotations, and horizontal and vertical flips of an image. These geometric manipulations can be performed without inverse or forward wavelet transform. However, all code-blocks need to be re-coded in the wavelet domain. Part II will allow the same transformations to be simply flagged in the bitstream, and left for the decoder to perform as each code block is decompressed.

Finally, the integer nature of the (5,3) wavelet allows an image or portion of an image to be compressed multiple times with the same quantization with no additional loss. Unfortunately, this is only true if the decompressed sample values are not clipped when they fall outside the full dynamic range (e.g., 0 to 255 for 8 bit images). If the original image did not use the full dynamic range (for example 8 bit images using only 32 to 220), then this is not an issue. If clipping occurs, the cycle of clipping and quantization can cause successive loss with each re-compression.

6 Performance

Figure 4 provides rate-distortion performance for two different JPEG modes, and three different JPEG-2000 modes for the bike image (grayscale, 2048 by 2560) from the SCID test set. The JPEG modes are progressive (P-DCT) and sequential (S-DCT) both with optimized Huffman tables. The JPEG-2000 modes are single layer with the (9,7) wavelet (S-9,7), six layer progressive with the (9,7) wavelet (P6-9,7), and 7 layer progressive with the (3,5) wavelet (P7-3,5). The JPEG-2000 progressive modes have been optimized for 0.0625, 0.125, 0.25, 0.5, 1.0, 2.0 bpp and lossless for the 5x3 wavelet. The JPEG progressive mode uses a combination of spectral refinement and successive approximation.

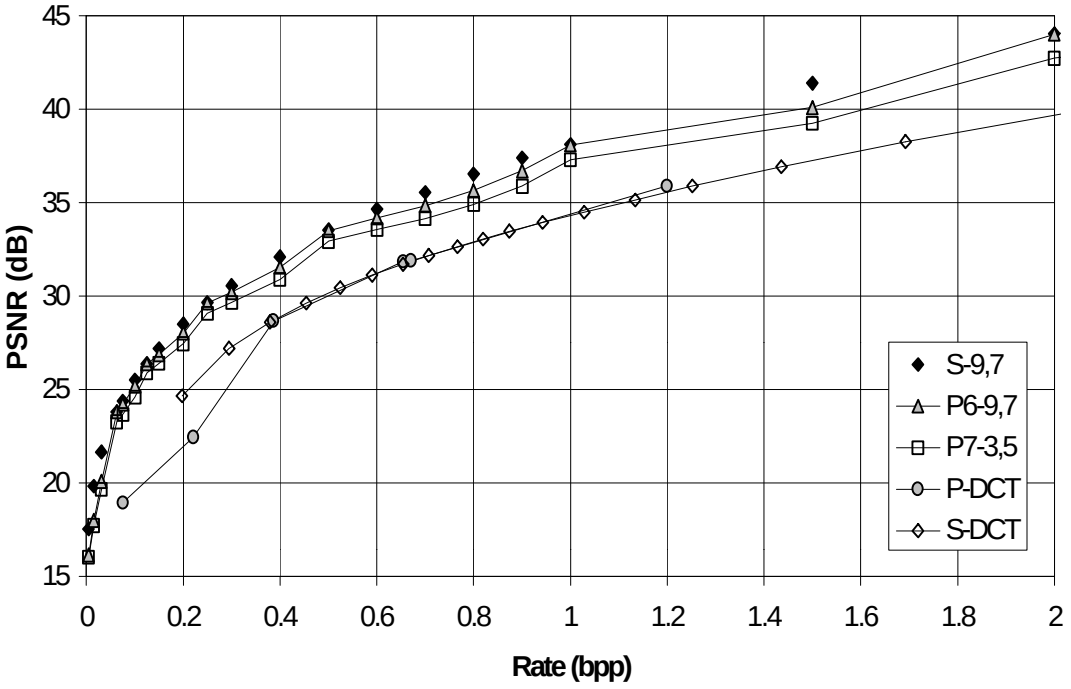

Figure 4: Rate-distortion performance for JPEG and JPEG 2000 on the SCID bike image.

The JPEG-2000 results are significantly better than the JPEG results for all modes and all bitrates on this image. Typically JPEG-2000 provides only a few dB improvement from 0.5 to 1.0 bpp but substantial improvement below 0.25 bpp and above 1.5 bpp. It should be noted that there are images for which JPEG performance is very close to the (3,5) wavelet performance (at least between 0.5 and 1.5 bpp). It should also be noted that the progression in JPEG was not optimized for this image, while the JPEG-2000 progressive modes are optimized for the image. However, this is a key advantage of the progressive JPEG-2000 over progressive JPEG. With progressive JPEG the DCT coefficients remain unchanged, but the encoding of those coefficients in any scan depend on the previous stages, and the number of bits/coefficients coded in each stage. It is thus extremely difficult to optimize over all the progression possibilities. For JPEG-2000 the coded data bits do not change regardless of the method of progression or number of stages used (The packet headers do change, but this is a second order effect). Thus it is relatively easy to select the desired progression, for example by adding sub-bitplanes which improve the R-D the most until the desired rate is achieved.

With JPEG-2000 the progressive performance is almost identical to the single layer performance at the rates for which the progression was optimized. Once again, this is because the coded data bits do not change. The slight difference is due solely to the increased signaling cost for the additional layers (which changes the packet headers). It is possible to provide "generic rate scalability" by using upwards of fifty layers. In this case the "scallops" in the progressive curve disappear, but the overhead increases, so the curve is always lower than the single layer points.

Although JPEG-2000 provides significantly lower distortion for the same bitrate, the computational complexity is significantly higher. Current JPEG-2000 software implementations run roughly a factor of three slower than optimized JPEG codecs. Speed of JPEG-2000 code should increase over time with implementation optimization, but the multi-pass bitplane context model and arithmetic entropy coder will prevent any software implementation from reaching the speed JPEG obtains with the DCT and Huffman coder.

JPEG-2000 also requires more memory than sequential JPEG, but not as much as might be expected. For conceptually simple implementations, encoders and decoders buffer entire code-blocks, typically 64 by 64 for entropy coding. However, block based, or sliding window implementations of the wavelet transform allow operation on just a few code-blocks at a time. For highly optimized, pipelined, parallel implementations, entropy coding can proceed without buffering of code-blocks. Short and wide codeblocks (say 4 by 512) can also be employed to limit the memory requirements of the overall system when sliding window wavelet transforms are employed.

Progressive JPEG-2000 can actually use less memory than progressive JPEG (although at additional computational cost). For progressive JPEG decompression, typically an entire coefficient buffer the size of the image is kept, coefficients are updated as data is decoded and the inverse DCT is performed to update the screen. JPEG-2000 implementations can keep just the compressed data in memory and augment the compressed data with new data, then decode a code-block, and perform the inverse wavelet transform.

Table 3: Lossless performance of JPEG, JPEG-LS, and JPEG-2000.

Method	Aerial2	Bike	Barbara	Cmpnd1
JPEG	5.589	4.980	5.663	2.478
JPEG-LS	5.286	4.356	4.863	1.242
JPEG-2000 (50 layers)	5.467	4.562	4.823	2.166
JPEG-2000 (One layer)	5.441	4.541	4.783	2.138

Table 3 shows the lossless performance of JPEG, JPEG-LS, and JPEG-2000. JPEG uses a predictor and Huffman coding (no DCT). In each case the best of all predictors has been used, and Huffman tables have been optimized. For primarily continuous-tone imagery as in the Aerial2, Bike, and Barbara images, JPEG-2000 is close to JPEG-LS, and substantially better than JPEG lossless. For images with text and graphics (2/3 of the Cmpnd1 image contains only rendered text), JPEG-LS provides almost a factor of two gain over JPEG lossless and JPEG-2000. Of course, the entire feature set is available for even losslessly compressed JPEG-2000 imagery, while the other two algorithms can provide only lossless raster-based decompression (for each tile). Hopefully, Part II of JPEG-2000 will improve performance on compound imagery.

Table 4 shows the PSNR obtained with JPEG-2000 on the Bike image with various encoding modes at 1.0 and 0.25 bpp. For comparison the image was first encoded with 512 by 512 tiles, 64 by 64 code-blocks, the (3,5) wavelet, and 7 rate-distortion optimized layers from 0.0625, to 2.0 bpp, and lossless. The remaining lines in the table show slight modifications from this reference. Using 128 by 128 tiles has a noticeable affect, especially at lower bitrates. But the 512 by 512 tiles had little impact over no tiles at all.

Reducing the code-block size also has a noticeable impact on compression, but it does not vary with bitrate. The (9,7) wavelet and non-SNR-progressive (one layer) encodings provide a significant performance increase. Providing complete rate scalability (50 layers) has a slight cost. Finally, values for sequential JPEG with optimized Huffman tables are in the last line.

Table 4: Performance Effects of Encoder Options on Bike Image.

Encode Method	**PSNR (dB)**	
	1.0 bpp	0.25 bpp
Reference	37.27	29.00
128 by 128 Tiles	36.81	28.16
No Tiles	37.31	29.08
32 by 32 Code-blocks	37.10	28.86
50 Layers	37.17	28.81
One Layer	37.73	29.19
(9,7) Wavelet	38.05	29.55
JPEG	34.37	27.21

7 Conclusion

The definition of JPEG-2000 is of course the standard. ISO sells copies of the specification but only after the "International Standard" stage is reached. Drafts of the standard and much more information are available by joining the WG1 committee or the appropriate national body responsible for sending delegates. Hopefully, free software will soon be available so the features can be tested by anyone.

JPEG-2000 is unlikely to replace JPEG in low complexity applications at bitrates in the range where JPEG performs well. However, for applications requiring either higher quality or lower bitrates, or any of the features provided, JPEG-2000 should be a welcome standard.

References

1. A. Zandi, J. D. Allen, E. L. Schwartz, and M. Boliek, "CREW: Compression with reversible embedded wavelets," *Proc. of IEEE Data Compression Conference*, Snowbird, Utah, pp. 212-221, March 1995.
2. M. Boliek, M. Gormish, E. L. Schwartz, and A. F. Keith, "Decoding compression with reversible embedded wavelets (CREW) codestreams," *Journal of Electronic Imaging*, vol. 7, no. 3, pp. 402-209, July 1998.
3. M. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS," submitted *to IEEE Trans. on Image Proc*.
4. M. Boliek, "New work item proposal: JPEG2000 image coding system," *ISO/IEC JTC1/SC 29/WG1 N390*, June 1996.
5. J. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. on Sig. Proc*., vol. 41, no. 12, pp. 3445-3462, Dec. 1993.
6. D. Taubman and A. Zakhor, "Multirate 3-D subband coding of video," *IEEE Trans.*

on Image Proc., vol. 3, no. 5, pp. 572-588, Sept. 1994.
7. A. Said and W. A. Pearlman, "A new fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. on Circuits and Systems for Video Tech*., vol. 6, no. 3, pp. 243-250, June 1996.
8. "Call for contributions for JPEG 2000 (JTC 1.29.14, 15444): image coding system," *ISO/IEC JTC1/SC 29/WG1 N505*, March 1997.
9. M. W. Marcellin and T. R. Fischer, "Trellis coded quantization of memoryless and Gauss-Markov sources," *IEEE Trans. on Commun*., vol. 38, no. 1, pp. 82-93, Jan 1990.
10. J. H. Kasner, M. W. Marcellin, B. R. Hunt, "Universal trellis coded quantization," *IEEE Trans. on Image Proc.*, vol. 8, no. 12, pp. 1677-1687, Dec. 1999.
11. J. Li, P. Cheng, and C.-C. J. Kuo, "On the improvements of embedded zerotree wavelet (EZW) coding," in *Proc. SPIE, Vis. Comm. and Image Proc*., vol. 2601, pp. 1490-1501, Taipei, Taiwan, May 1995.
12. J. Li and S. Lei, "Rate-distortion optimized embedding," *Proc. of Picture Coding Symposium*, pp. 201-206, 1997.
13. E. Ordentlich, M. Weinberger, and G. Seroussi, "A low-complexity modeling approach for embedded coding of wavelet coefficients," *Proc. of IEEE Data Compression Conf*., pp. 408-417, 1998.
14. P. J. Sementilli, A. Bilgin, J. H. Kasner, M. W. Marcellin, "Wavelet TCQ: submission to JPEG-2000," *Proc. of SPIE, Appl. of Digital Image Proc*., pp. 2-12, July 1998.
15. W. B. Pennebaker and J. L. Mitchell, *JPEG still image data compression standard*, Van Nostrand Reinhold, New York, 1993.
16. A. Said and W. A. Pearlman, "An image multiresolution representation for lossless and lossy compression," *IEEE Trans. Image Proc*., vol. 5, no. 9, pp. 1303-1310, Sept. 1996.
17. C. Christopoulos, "JPEG-2000 verification model 2.0 (technical description)," *ISO/IEC JTC 1/SC 29/WG 1 N988*, Oct. 1998.
18. D. Speck, "New options in radix-255 arithmetic coder," *ISO/IEC JTC1/SC 29/WG1 N482R*, March 1997.
19. D. Taubman, "Report on coding experiment codeff22: EBCOT (Embedded block coding with optimized truncation)," *ISO/IEC JTC1/SC 29/WG1 N1020R*, October 1998.
20. D. Taubman, "High performance scalable image compression with EBCOT," to appear in *IEEE Trans. on Image Proc*.
21. SAIC/UA, HP/UNSW, HP, WSU, Kodak, "Report on core experiments codEff4, codEff5 and codEff7: reduced complexity entropy coding," *ISO/IEC JTC1/SC 29/WG1 N1312*, June 1999.
22. M. J. Gormish, E. L. Schwartz, A. Keith, M. Boliek, and A. Zandi, "Lossless and nearly lossless compression for high quality images," *Proc. of SPIE, Very High Resolution and Quality Imaging II*, pp. 62-70, February 1997.
23. M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image coding using wavelet transform," *IEEE Trans. on Image Proc*., vol. 1, no. 2, pp 205-220, Apr. 1992.
24. R. Calderbank, I. Daubechies, W. Sweldens, and B.-L. Yeo, "Wavelet transforms that map integers to integers," *Journal of Appl. and Comp. Harmon. Analy*., vol. 5, pp.

332-369, 1998.
25. Chrysafis and A. Ortega, “Line based, reduced memory, wavelet image compression,” *Proc. of IEEE Data Compression Conf.*, pp. 398-407, 1998.
26. Motorola Australia and UBC, “Report on core experiment codeff1: complexity reduction of SSWT,” *ISO/IEC JTC1/SC 29/WG1 N1003*, November 1998.
27. Canon Research France, Motorola Australia and Hewlett Packard, “Report on Core Experiment CodEff11: Reduced overlap in SSWT,” *ISO/IEC JTC1/SC 29/WG1 N1189*, March 1999.
28. “Information technology - coded representation of picture and audio information - lossy/lossless coding of bi-level images,” 14492 Final Committee Draft, *ISO/IEC JTC1/SC 29/WG1 N1359*, July 1999.