JPEG2000: HIGHLY SCALABLE IMAGE COMPRESSION

Michael W. Marcellin and Ali Bilgin

Department of Electrical and Computer Engineering, The University of Arizona, Tucson, AZ 85721.
{mwm,bilgin}@ece.arizona.edu http://www-spacl.ece.arizona.edu

ABSTRACT

JPEG2000 is the latest ISO/IEC image compression standard. It is distinct from previous standards in that it is more than an input-output filter. JPEG2000 enables decompression of many image products from a single compressed file and offers several opportunities for compressed domain processing. This paper describes the feature set of the JPEG2000 compression standard, followed by a high level description of the algorithm.

1. INTRODUCTION

Previous image compression systems and/or standards have been used primarily as input-output filters within applications. That is, as an image was written to (or read from) a disk it was compressed or decompressed, largely as a storage function. Additionally, all decisions as to image quality and/or compression ratio were made at compression time. At decompression time, only the (single) image quality, size, resolution and spatial extent envisioned by the compressor was available to the decompressor.

For example, with JPEG baseline (sequential mode), an image is compressed using a particular quantization table. This essentially determines the quality that will be achieved at decompression time. Lower (or higher) quality decompressions are not available to the decompressor. Similarly, if JPEG lossless, or JPEG-LS are employed for compression, only lossless decompression is available, and high compression ratios are impossible.

Notable exceptions to these rigid structures exist. In fact, the original JPEG standard has four "modes" of operation: sequential, progressive, hierarchical, and lossless. However, these four modes rely on distinctly different technologies, and result in distinctly different codestreams. While certain interactions (say, between progressive and hierarchical) are allowed according to the standard document, this has never been exploited (to the best of the authors' knowledge) in a commercial system.

JPEG2000 creates a framework where the image compression system acts more like an image processing system than a simple input-output storage filter. The decision on several key compression parameters such as quality or resolution can be delayed until after the creation of the codestream, and several different image products to be extracted from a single codestream.

This paper is organized as follows. In Section 2, we introduce the JPEG2000 feature set. In Section 3, we describe the JPEG2000 algorithm. Finally, we present results on the performance of JPEG 2000 in Section 4.

2. JPEG2000 FEATURES

JPEG2000 brings a new paradigm to image compression standards [1]. The benefits of all four JPEG modes are tightly integrated in JPEG2000. The compressor decides maximum image quality (up to and including lossless). Also chosen at compression time, is maximum resolution (or size). Any image quality or size can be decompressed from the resulting codestream, up to and including the maximums chosen at encode time.

For example, suppose an image is compressed losslessly at full size. Suppose further that the resulting file is of size B_0 (bytes). It is then possible to extract B_1 bytes from the file, ($B_1 < B_0$) and decompress those B_1 bytes to obtain a lossy decompressed image. Similarly, it is possible to extract B_2 bytes from the file and decompress to obtain a reduced resolution image. In addition to the quality scalability and resolution scalability, JPEG2000 codestreams support spatial random access. There are several mechanisms to retrieve and decompress data from the codestream corresponding to arbitrary spatial regions of an image. The different mechanisms yield different granularity of access, at varying levels of difficulty.

This random access extends to color components as well. Specifically, the black and white (grayscale) component can be extracted from a color image. As above, this can be done region by region with varying qualities and resolutions.

It is important to note that in every case discussed above, it is possible to locate, extract, and decode only the bytes required to decode the desired image product. It is not necessary to decode the entire codestream and/or image. In many cases, the bytes extracted and decoded are identical to those obtained if only the desired image products were compressed in the first place.

2.1. Compressed Domain Image Processing/Editing

Any of the image products discussed above can be extracted to create a new JPEG2000 compliant codestream without a decompress/recompress cycle. In addition to the elegance and computational savings, compression noise "build-up" that occurs in most compression schemes when repetitive compress/decompress cycles are utilized can be avoided.

In addition to reduced quality and reduced resolutions, compressed domain image cropping is possible. Cropping in the compressed domain is accomplished by accessing the compressed data associated with a given spatial region (using random codestream access, as discussed above) and

0-7695-1062-0/01 $10.00 © 2001 IEEE

rewriting it as a compliant codestream. Some special processing is required around the cropped image borders.

Geometric manipulations are also supported in the (partially) compressed domain. Image rotations of 90, 180 and 270 degrees are possible. Image flipping (top-to-bottom and/or left-to-right) can also be performed. These procedures cannot be carried out entirely compressed. Some transcoding of arithmetically coded data is required, but an inverse/forward transform cycle is unnecessary.

2.2. Progression

Many types of progressive transmission are supported by JPEG2000. As mentioned previously, progressive transmission is highly desirable when receiving imagery over slow communication links. As more data are received, the rendition of the displayed imagery improves in some fashion. JPEG 2000 supports progression in four dimensions: Quality, Resolution, Spatial Location, and Component.

The first dimension of progressivity in JPEG2000 is quality. As more data are received, image quality is improved. It should be noted that the image quality improves remarkably quickly with JPEG 2000. An image is typically recognizable after only about 0.05 bits/pixel have been received. For a 320 × 240 pixel image, this corresponds to only 480 bytes of received data. With only 0.25 bits/pixel (2,400 bytes) received, most major compression artifacts disappear. To achieve quality corresponding to no visual distortion, between 0.75 and 1.0 bits per pixel are usually required. Demanding applications may sometimes require up to 2.0 bits/pixel or even truly lossless decompression (e.g., medical applications). We remark here again, that all qualities up to and including lossless (equivalently, all bitrates, or all compression ratios) are contained within a single compressed codestream. Improving quality is a simple matter of decoding more bits.

The second dimension of progressivity in JPEG2000 is resolution. In this type of progression, the first few bytes are used to form a small "thumbnail" of the image. As more bytes are received, the resolution (or size) of the image increases by factors of 2 (on each side). Eventually, the full size image is obtained.

The third dimension of progressivity in JPEG2000 is spatial location. With this type of progression, imagery is received in a "stripe," or "scan" based fashion, from top-to-bottom. This type of progression is particularly useful for certain types of low memory printers and scanners.

The fourth and final dimension of progressivity is by component. JPEG2000 supports images with up to 16384 components. Most images with more than 4 components are from scientific instruments (e.g., LANDSAT). More typically, images are 1 component (grayscale), 3 components (e.g., RGB, YUV, etc.), or 4 components (CMYK). Overlay components containing text or graphics are also common. Component progression controls the order in which the data corresponding to different components is decoded. With progressive by component, the grayscale version of an image might first be decoded, followed by color information, followed by overlaid annotations, text, etc.

The four dimensions of progressivity are very powerful and can be changed (nearly at will) throughout the codestream. Since only the data required by the viewer needs to be transmitted, the "effective compression ratio" experienced by the client can be many times greater then the actual compression ratio as measured from the file size at the server.

3. THE JPEG2000 ALGORITHM

In this section, we provide a high level description of the JPEG 2000 algorithm. Although the standard specifies only the decoder and codestream syntax, this paper focuses on the description of a representative encoder, since this enables a more readable explanation of the algorithm. We note that we will discuss the algorithm as it applies to Part I of the standard. Part I describes the minimal decoder required for JPEG2000, which should be used to provide maximum interchange. "Value-added" technologies that are not required of all implementations are described in Part II.

3.1. Tiles and Component Transforms

In JPEG2000, an image is defined as a collection of two-dimensional rectangular arrays of samples. Each of these arrays is called an image component.Typical examples include RGB images that have three components, and CMYK images that have four components. Components need not have the same number of samples. The image resides on a high resolution grid. This grid is the reference for all geometric structures in JPEG2000.

The first step in JPEG2000 is to divide the image into non-overlapping rectangular tiles. The array of samples from one component that fall within a tile is called a *tile-component*. The primary benefits of tiles are that they provide a simple vehicle for limiting implementation memory requirements and for spatial random access. They can also be useful for segmenting compound imagery, as the coding parameters can be changed from tile to tile. As the tile grid is rectangular and regular, options for segmentation are rather restricted.

The primary disadvantage of tiles is blocking artifacts. Since each tile is compressed independently of all other tiles, visible artifacts can occur at tile boundaries. For high bit rates and large tile sizes, these artifacts are generally invisible.

When multiple component images are being encoded, one of two optional component transforms can be applied to the first three components. These transforms decorrelate the components, and increase the compression efficiency.

The first component transform is called the reversible color transform (RCT) and is only used in conjunction with the reversible wavelet transform discussed in the following section. The second transform is the irreversible color transform (ICT) and is only used in conjunction with the irreversible wavelet transform. Although both transforms are invertible, in the mathematical sense, the RCT maps integer color components to integer transformed color components, and is perfectly invertible using only finite (low) precision arithmetic. Conversely, ICT employs floating point arithmetic, and, in general, would require infinite precision arithmetic to guarantee perfect inversion.

3.2. The Wavelet Transform

JPEG2000 (Part 1) supports two choices for the wavelet filters. The so called (9,7) wavelet transform has floating point impulse responses of lengths 9 and 7. This transform is known as the irreversible transform, and is useful for high performance lossy compression. The so called (5,3) wavelet transform is implemented using integer arithmetic. Careful rounding of (both intermediate results and the final wavelet coefficients) is performed during filtering. The resulting transform is reversible, enabling lossless (in addition to lossy) compression. For a given (lossy) compression ratio, the image quality obtained with the (9,7) transform is generally superior to that obtained with the (5,3) transform. However, the performance of the (5,3) transform is still quite good.

Each resolution of a tile-component is partitioned into *precincts*. Precincts behave much like tiles, but in the wavelet domain. The precinct size can be chosen independently by resolution, however each side of a precinct must be a power of 2 in size. Figure 1 shows a precinct partition for a single resolution.

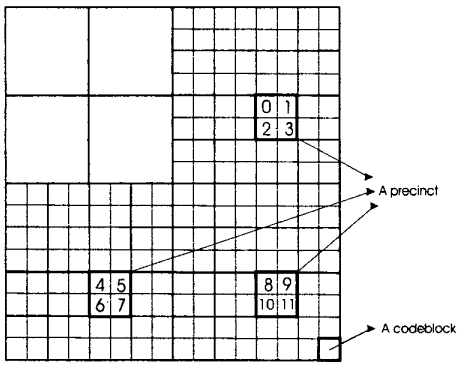

Figure 1: Partitioning of wavelet subbands.

Precincts are another ingredient to low memory implementations in the absence of tiles. Compressed data from a precinct are grouped together to form a *packet*. Before a packet header can be created, all compressed data from the corresponding precinct must be available. Thus, only the compressed data for a precinct must be buffered, rather than that of an entire tile (or image in the absence of tiles).

In addition to the memory benefits of precincts, they also provide a method of spatial random access. The granularity of this method is finer than that for tiles, but courser than that for *codeblocks* (see next paragraph). Also, since precincts are formed in the wavelet transform domain, there is no "break" in the transform at precinct boundaries (as there is at tile boundaries). Thus, precincts do not cause block (tile) artifacts.

Codeblocks form the smallest geometric structure of JPEG2000. Initial quantization and bitplane coding are performed on codeblocks. Codeblocks are formed by partitioning subbands. Since the precinct size (resolution dependent) and codeblock size (resolution independent) are both powers of 2, the two partitions are forced to "line up." Thus, it is reasonable to view the codeblocks as partitions of the precincts (rather than of the subbands). If the codeblock size exceeds the precinct size in any subband, the codeblocks are forced to obey precinct boundaries. Effectively then the codeblock width or height (or both) is reduced to that of the precinct. The partitioning of wavelet subbands into codeblocks is illustrated in Figure 1 as well.

As discussed earlier, codeblocks help enable low memory implementation by limiting the amount of (uncompressed) wavelet data that must be buffered before it can be quantized and compressed. This is in contrast to precincts, which limit the amount of compressed data that must be buffered before packets can be formed. Another benefit of codeblocks is that they provide fine grain random access to spatial regions.

3.3. Quantization and Bitplane Coding

Quantization is a process by which wavelet coefficients are reduced in precision to increase compression. For a given wavelet coefficient z and quantizer step size Δ, a signed integer q given by

$$q = Q(z) = \text{sign}(z) \left\lfloor \frac{|z|}{\Delta} \right\rfloor$$

is used to indicate in which interval z lies. This index is coded using techniques described later in this section. Given q, the decoder must estimate the value of z as

$$\hat{z} = \overline{Q^{-1}}(q) = \begin{cases} 0 & q = 0 \\ \text{sign}(q)\,(|q| + \delta)\,\Delta & q \neq 0 \end{cases}$$

where δ is a user selectable parameter (typically $\delta = 1/2$).

This particular type of quantization is known as deadzone uniform scalar quantization, and is key to the quality progression feature of JPEG2000. The reason for this is that such quantizers are "embedded." That is, the quantization index of every quantization (with step size Δ) has embedded within it, the index of every quantization with step size $2^p\Delta$, $p = 0, 1, 2, \ldots$ Thus, if z is quantized with step size Δ to get q, and the p Least Significant Bits (LSBs) of q are missing (not decoded yet), then the appropriate dequantizer is

$$\hat{z} = \overline{Q^{-1}}(q^{(p)}) = \begin{cases} 0 & q^{(p)} = 0 \\ \text{sign}(q^{(p)}) \left(\left|q^{(p)}\right| + \delta\right) 2^p\Delta & q^{(p)} \neq 0 \end{cases}$$

In this case, $\hat{z}$ is identical to what it would have been if the step size had been $2^p\Delta$ in the first place.

For irreversible wavelets, a different step size can be chosen for each subband. These step sizes are substantially equivalent to Q-table values from JPEG and can be chosen to meet differing needs. For reversible wavelets, a step size of $\Delta = 1$ is used. This results in no quantization at all unless one or more LSBs of the (integer) wavelet coefficients are omitted. In this case, the equivalent induced step size is $2^p\Delta = 2^p$.

Entropy coding is performed independently on each codeblock. This coding is carried out as context-dependent, binary, arithmetic coding of bitplanes. The arithmetic coder employed is the MQ-coder as specified in the JBIG-2 standard. For brevity, the computation to determine each context is not included here.

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 LSBs 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 coding passes).

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 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.

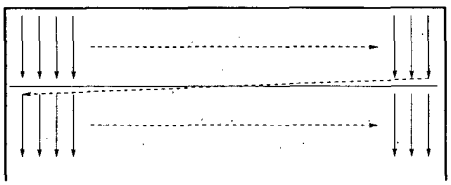

Figure 2: Scan pattern for bitplane coding.

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.

All bitplane coding is done using context dependent binary arithmetic coding 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.

3.4. Packets and Layers

Packets are the fundamental unit used for construction of the codestream. A packet contains the compressed bytes from some number of coding passes from each codeblock in one precinct of one tile-component. The number of coding passes can vary from block to block. Any number of passes (including zero) is legal. In fact, zero passes from every codeblock is legal. In this case the packet must still be constructed as an "empty packet." An empty packet consists of a packet header, but no packet body.

A packet consists of a packet header followed by a packet body. The packet body contains m_i coding passes from codeblock i in order $i = 0, 1, 2, \ldots$, and m_i can be any integer including 0.

The packet header contains the information necessary to decode the packet. It includes a flag denoting whether the packet is empty or not. If not, it also includes: codeblock inclusion information (whether $m_i = 0$, or $m_i > 0$ for each codeblock i); the number of completely zero bitplanes (zero MSBs) for each codeblock; the number of coding passes m_i for each included codeblock; and the number of compressed bytes included for 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 (for the same precinct) in a way to just enable decoding of the current packet. For example, the number of leading zero MSBs for a particular codeblock is included only in the first packet that contains a contribution from that codeblock.

It is worth reiterating that the bitplane coding of a codeblock is completely independent of any other codeblock. Similarly, the header coding for a packet in a particular precinct is independent of any other precinct.

A layer is a collection of packets from one tile. Specifically, a layer consists of one packet from each precinct of each resolution of each tile-component of one tile.

The first packet of a particular precinct contains some number of coding passes from each codeblock in the precinct. These coding passes represent some (varying) number of MSBs for each quantized wavelet coefficient index in each codeblock of the precinct. Similarly, the following packets contain compressed data for more and more coding passes of bitplanes of quantized wavelet coefficient indices, thereby reducing the number of missing LSBs (p) in the quantizer indices $q^{(p)}$. We note here that at the end of a given layer l, the number of missing LSBs (p) can be vastly different from codeblock to codeblock. However, within a single codeblock, (p) cannot differ from coefficient to coefficient by more than 1.

From this discussion, it is now understood that a packet provides one quality increment for one spatial region (precinct), at one resolution of one tile-component. Since a layer comprises one packet from each precinct of each resolution of each tile-component, a layer is then understood as one quality increment for an entire tile.

It should also be clear that the quality increment need not be consistent throughout the tile. Since the number of coding passes in a layer (via packets) can vary codeblock by codeblock, there is complete flexibility in "where and by how much" the quality is improved by a given layer. For example, a layer might improve one spatial region of a reduced resolution version of only the grayscale (luminance) component.

4. PERFORMANCE

Figure 3 provides rate-distortion performance for two different JPEG modes, and three different JPEG2000 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 JPEG2000 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 JPEG2000 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.

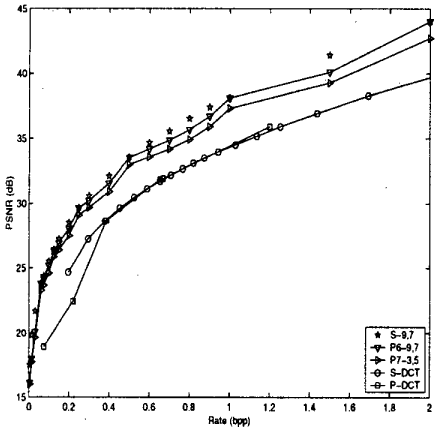

Figure 3: Rate-distortion performances of JPEG and JPEG2000 on the bike image.

The JPEG2000 results are significantly better than the JPEG results for all modes and all bitrates on this image. Typically JPEG2000 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 the progression in JPEG was not optimized for this image, while the JPEG2000 progressive modes are optimized for the image. However, the ability to perform such optimization in a simple manner is a key advantage of JPEG2000 over progressive JPEG.

With JPEG2000 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 JPEG2000 provides significantly lower distortion for the same bitrate, the computational complexity is significantly higher. Current JPEG2000 software implementations run roughly a factor of three slower than optimized JPEG codecs. Speed of JPEG2000 code should increase over time with implementation optimization.

JPEG2000 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.

Table 1: Lossless performance of JPEG, JPEG-LS, and JPEG2000.

Method	Images			
	Aerial2	Bike	Barbara	Cmpnd1
JPEG	5.589	4.980	5.663	2.478
JPEG-LS	5.286	4.356	4.863	1.242
JPEG2000 (50 Layers)	5.467	4.562	4.823	2.166

Table 1 shows the lossless performance of JPEG, JPEG-LS, and JPEG2000. 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, JPEG2000 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 JPEG2000. Of course, the entire feature set is available for even losslessly compressed JPEG2000 imagery, while the other two algorithms can provide only lossless raster-based decompression (for each tile).

5. REFERENCES

[1] ISO/IEC 15444-1, "JPEG2000 image coding system", Dec. 2000.