Processing JPEG-Compressed Images and Documents

Ricardo L. de Queiroz, *Member, IEEE*

***Abstract*— As the Joint Photographic Experts Group (JPEG) has become an international standard for image compression, we present techniques that allow the processing of an image in the "JPEG-compressed" domain. The goal is to reduce memory requirements while increasing speed by avoiding decompression and space domain operations. In each case, an effort is made to implement the minimum number of JPEG basic operations. Techniques are presented for scaling, previewing, rotating, mirroring, cropping, recompressing, and segmenting JPEG-compressed data. While most of the results apply to any image, we focus on scanned documents as our primary image source.**

***Index Terms*—Compression, image processing, JPEG.**

I. Introduction

SINCE images were brought to the desktop by more powerful computer hardware, image compression has become a popular necessity. The resolution of scanners, monitors, and printers has steeply increased in the past few years, outpacing the decline in memory prices. We are particularly concerned with images manipulated in the printing business, which involves the scanning and printing of documents and pictures. A typical scan of an 8.5×11 inches page at 600 pixels/in (ppi) generates roughly 30 MB of data per color separation (assuming one byte per color component per pixel). The same amount of data may be necessary to print a full page at a high addressability printer at the same resolution. From this example, we can appreciate the potential savings that might be provided by compression.

Typical lossless (or reversible) coders can barely attain a compression ratio of $2:1$ for most images. Thus, users often cope with lossy algorithms, which tend to slowly degrade the image quality in exchange for more aggressive compression ratios. The recommendation ISO DIS 10 918-1 known as JPEG Joint Photographic Experts Group (JPEG) has become an international standard for lossy compression of still images. Although there are more effective (and complex) compression methods, the recent decline in memory prices makes the JPEG trade-off (complexity versus compression) an ideal candidate for storage format. A comprehensive discussion of the standard, along with the recommendation text itself, can be found in [1]. JPEG has several modes of operation. For simplicity we concentrate on the most popular mode, known as baseline JPEG, although several results may also apply to other modes of operation. We employ the term *JPEG* to designate baseline JPEG, unless otherwise stated.

The objective of this paper is to present techniques that allow the processing of JPEG-compressed data without decompressing it, i.e., operations are performed in the "JPEG-compressed" domain. The meaning of "JPEG-compressed" domain also deserves some clarification. JPEG compression is performed by a series of operations: transform, quantization, zigzag scanning, differential pulse code modulation (DPCM), and entropy coding. Decompression is accomplished by performing inverse steps in an inverse order. We assume the data is only available in compressed format. Therefore, the first operations to be applied to the data are part of the decompression routine, and an effort is made to perform as few operations as possible. In some cases, only partial entropy decoding is needed, while in others we use most of the data in transform domain. So, most operations whose outputs are also JPEG-compressed images use the following steps: partial block decompression, fast processing, partial block compression. Each operation described in this paper can be alternatively implemented in the trivial way: decompressing, processing, and recompressing the image. The motivation for processing compressed images is because it saves memory and/or improves speed. Other authors have also addressed the topic of processing/analyzing JPEG compressed data. See, for example, [3]–[7] for further results in this topic.

Section II gives an overview of JPEG. Section III presents a technique to very quickly extract a preview image from the JPEG data and to scale the image size. Section IV introduces the concept of "cost maps" and show how they can be used for cropping an image and to ease the processing in the compressed domain. Rotation, transposition, mirroring, etc. are discussed in Section V. Image segmentation is discussed in Section VI, where we use a technique based on the cost maps. Section VII is concerned with recompression of the already compressed data. We discuss some techniques and suggest improvements and shortcuts. Finally, the concluding remarks are presented in Section VIII.

II. JPEG Overview

As discussed, JPEG is implemented through a sequence of operations as in Fig. 1. Let us ignore file format related aspects (such as header, byte stuffing, etc. [1]). The image is divided into blocks of 8×8 pixels. Blocks are grouped into minimum coding units (MCU). If the image dimensions are not integer multiples of the dimensions of an MCU, the image may be

Manuscript received January 7, 1997; revised March 5, 1998. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Thrasyvoulos N. Pappas.

The author is with Xerox Corporation, Webster, NY 14580 USA (e-mail: queiroz@wrc.xerox.com).

Publisher Item Identifier S 1057-7149(98)08715-6.

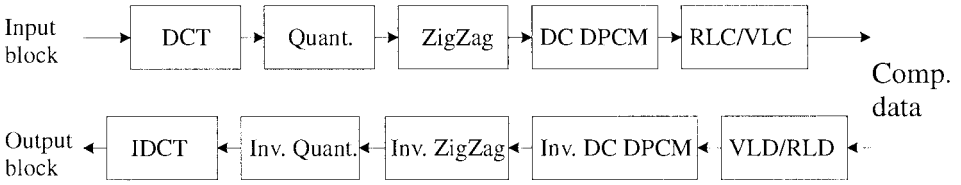


Fig. 1. JPEG basic operations.

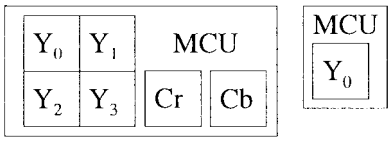


Fig. 2. Grouping blocks in an MCU for typical luminance-chrominace data, and for a monochrome image.

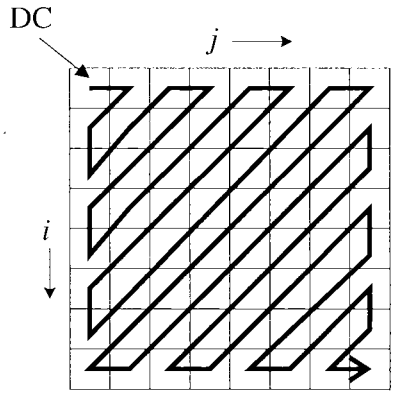


Fig. 3. Zigzag scanning order for an 8 × 8 block.

padded (perhaps with zeroes) until fitting the desired size. The real image size is conveyed in the header so that the decoder can appropriately crop the resulting image. If the image has a number of color separations, it is frequently converted to a luminance/chrominance/chrominance color space [1] such as YCrCb, YUV, CIELab, etc., and the chrominance signals are commonly downsampled by a factor of two in each direction. A typical MCU is illustrated in Fig. 2, containing four blocks from the luminance separation and one block from each of the chrominance separations. If the image is monochrome, the MCU has a single block. For simplicity, assume that an MCU contains just one block so that we can ignore MCU in this discussion, since grouping blocks does not affect the results in this paper.

The basic building blocks for JPEG compression are shown in Fig. 1. The block is transformed using the discrete cosine transform (DCT) [1], [2]. The DCT of an M-point sequence is defined by the $M \times M$ matrix $\mathbf{D}_M$ whose entries are

$$d_{ij}^M = \sqrt{\frac{2}{M}} k_i \cos\left(\frac{(2j+1)i\pi}{2M}\right) \tag{1}$$

where $k_0 = 1$ and $k_i = 1/\sqrt{2}$, for $1 \leq i \leq M-1$. Let the block of 8×8 pixels be denoted by $\mathbf{X}$, while the transformed block is denoted by $\mathbf{Y}$. The separable transform and its inverse are performed as

$$\mathbf{Y} = \mathbf{D}_8 \mathbf{X} \mathbf{D}_8^T \quad \mathbf{X} = \mathbf{D}_8^T \mathbf{Y} \mathbf{D}_8. \tag{2}$$

The DCT matrices are highly structured and have fast implementation algorithms [2].

The quantization step involves simple unbounded uniform quantizers. Each of the 64 transformed coefficients (y_{ij}) in a block is applied to a uniform quantizer with step size q_{ij} generating a quantized number c_{ij}. Quantization and inverse quantization in JPEG are defined as

$$c_{ij} = \text{round}(y_{ij}/q_{ij}) \quad \hat{y}_{ij} = c_{ij} q_{ij}. \tag{3}$$

The step sizes are stored in a table that is transmitted along with the compressed data.[1] Example (default) tables for luminance and chrominance are given in the JPEG draft.

The two-dimensional (2-D) array c_{ij} is converted into a one-dimensional (1-D) vector $zz(n)$ through zigzag scanning, which organizes the data into a vector following the path shown in Fig. 3. The DC component (c_{00}) is the first element of the vector and is replaced by the difference between itself and the DC component of the previous block,[2] and this difference is represented in $zz(0)$. The inverse operations are performed at the decoder side (from $zz(n)$ to c_{ij}).

This resulting vector is lossless encoded using a combination of run-length coding (RLC) and variable-length coding (VLC). Both the DC and the AC coefficients are divided into category, offset, and sign. Let the number be represented in magnitude and sign so that its magnitude is represented as a binary string of fixed length. For example: 00...001XXXXX. There are a number of zeros before the first nonzero bit followed by a number of bits that can be either zero or one. The order of the last nonzero digit (from right to left) is the category of the number (SSSS), which in the example is 6. There are, thus, SSSS-1 offset bits to completely characterize the magnitude given the category. If the number is 0, we define $\text{SSSS} = 0$. $zz(0)$ is encoded by sending a variable length code representing SSSS [1], followed by one bit for sign (1 if positive), and, SSSS-1 bits for the offset. If $\text{SSSS} = 1$, offset is not sent, and if $\text{SSSS} = 0$, the sign is not sent either. The remaining 63 elements of $zz(n)$ are encoded through a combination of the *run of zero-valued elements before the first nonzero element* (RRRR) and of the element itself. Again, the coefficient is broken apart into sign, category and offset. RRRR and SSSS are combined into one 8-b symbol. If the run is larger than 16, a special symbol is used (ZRL). The AC

[1] In case multiples images are transmitted, there is a provision to transmit all the tables upfront [1].

[2] JPEG defines variants for multiple-blocks MCU and when using restart markers [1].

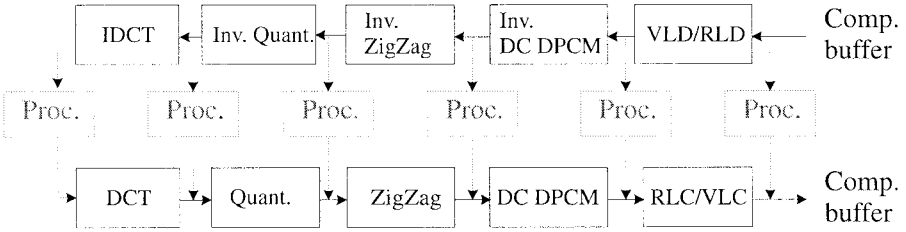


Fig. 4. Processing a compressed buffer. The goal is to perform as few JPEG operations as possible before and after the processing stage.

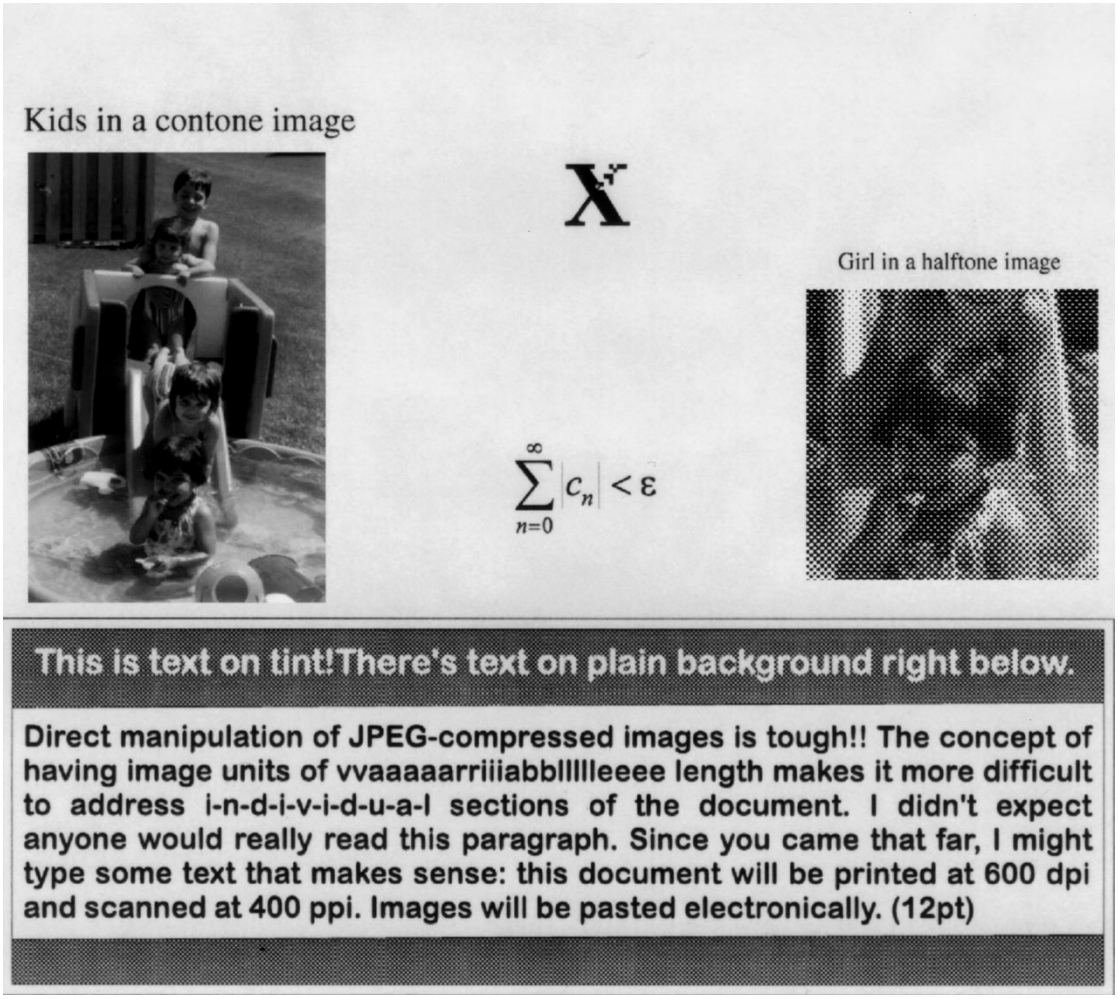


Fig. 5. Test image after compression at a ratio of 10.7 : 1.

coefficients are encoded by sending the variable length code representing the RRRR/SSSS symbol followed by one sign bit (1 if positive), and by SSSS-1 offset bits. If all coefficients are zero until the end of the vector an EOB symbol is encoded instead. Both EOB and ZRL symbols are encoded using the same table as RRRR/SSSS. The decoder decodes symbols, reads signs and offsets, and reconstructs the vector.

Definitions of VLC tables are beyond the scope of this paper. However, JPEG specifies example (default) tables and also provides algorithms for image specific table optimization. As with the quantizer tables, VLC table information is provided to the decoder. A preferred way to encode and decode the (RRRR/SSSS or SSSS) symbols is by using a look-up table (LUT). Each symbol has 8 b, and each codeword might use anything between one and 16 bits. So, the encoder may use an up-to-256-entry LUT containing the codeword and its length, while the decoder might use a 64K-entry LUT containing the decoded symbol and the codeword length. At the decoder, the length information may be used to shift a 16-b register after the symbol is decoded and to make it ready for the next input data. In between symbols, of course, one may extract offset and sign bits from the register and shift it appropriately.

The goal of this paper is not to compress an image but to operate on an already compressed buffer. Fig. 4 illustrates the process, where the compressed data is (partially) decompressed and processed. The result is, then, recompressed into an

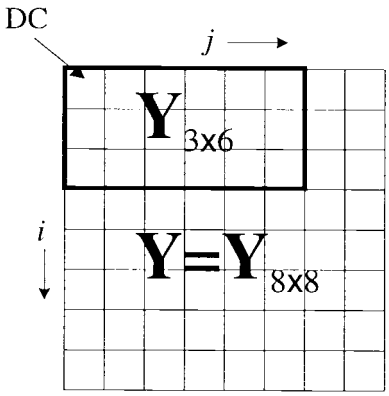


Fig. 6. Subblock of $m \times n$ DCT coefficients $(m = 3, n = 6)$.

output buffer. In the figure, there are several options for where to place a *shortcut* processing stage and bypass the remaining JPEG operations. An attempt is made to implement the minimum number of JPEG stages.

Throughout the paper we use a test document which was compressed using JPEG's default luminance quantizer table scaled by 1.25 achieving a compression ratio of 10.7 : 1. The decompressed test image is shown in Fig. 5. It contains regions with halftones, text and continuous tone (contone) images. The image was originally obtained by scanning a document at 400 ppi and pasting images electronically, before compression. The image resolution is 2424 $\times$ 2824 pixels.

III. Previewing and Resizing

Decimation and interpolation can be performed in the DCT domain [2]. Given that the DCT is a filterbank [8], one can borrow the filtering characteristics of the basis functions (filters) and perform filtering by simply discarding subbands or padding zeros. By weighting coefficients in the DCT domain, one will not get an equivalent linear time-invarying filter. In fact, the effective filter is a linear periodically time-varying (LPTV) system [8]. A discussion on the resulting LPTV filter and its efficacy for image downsampling can be found in [9]. By simply discarding or adding zero subbands one might get adequate and fast results using the DCT [9]–[13].

Suppose one has a compressed buffer and wants to quickly extract a downsampled image for previewing purposes. Starting with the 8 $\times$ 8 blocks inherent of JPEG, the inverse transform is given by (2). Let us denote the lower-frequency $m \times n$ coefficients of $\mathbf{Y}$ by $\mathbf{Y}_{m\times n}$, as in the example in Fig. 6, i.e., $\mathbf{Y}_{m\times n}$ contains y_{ij} for $0 \le i < m$ and $0 \le j < n$. A good preview approximation for an image at $m/8 \times n/8$ times the original resolution can be obtained by extracting a block of $m \times n$ pixels $(\mathbf{X}_{m\times n})$ for every JPEG block. This can be accomplished as:

$$\mathbf{X}_{m\times n} = \frac{\sqrt{mn}}{8}\mathbf{D}_m^T \mathbf{Y}_{m\times n}\mathbf{D}_n. \tag{4}$$

In other words, we perform a scaled inverse $m \times n$ DCT over $\mathbf{Y}_{m\times n}$. Two cases of interest are $m = n = 1$ and $m = n = 2$. The first is the trivial case of extracting the DC component $(c_{00} * q_{00})$, enough said. The second one, occurs for example

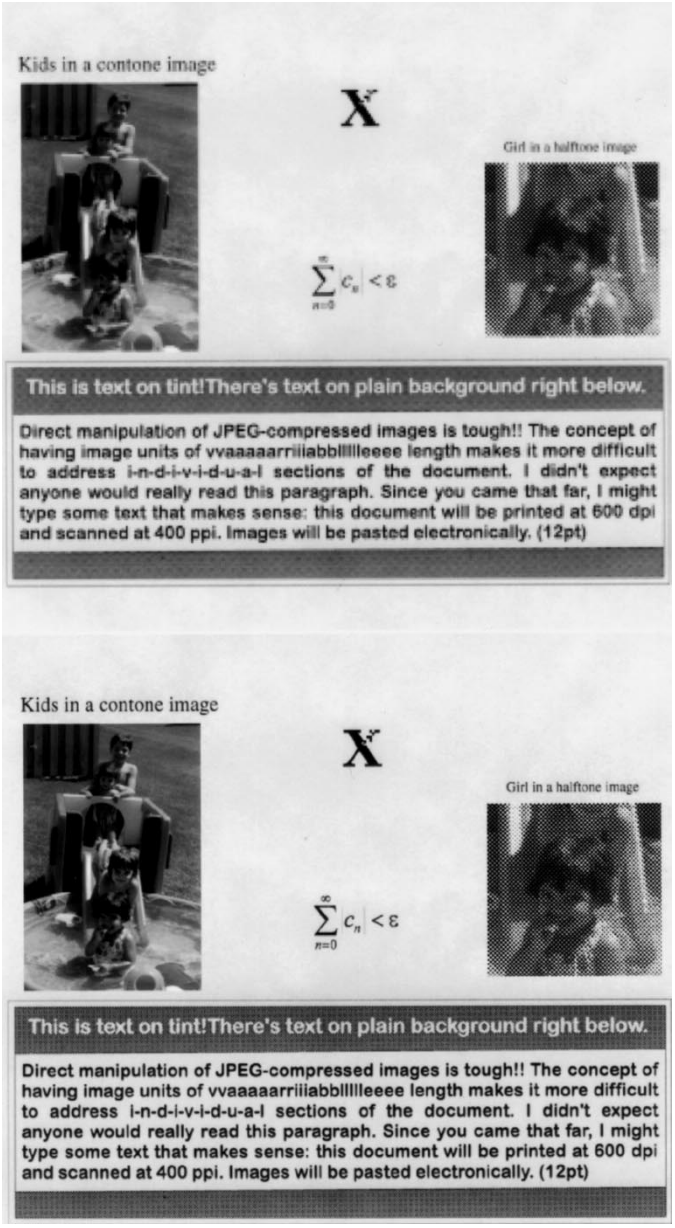


Fig. 7. Preview images extracted from the compressed bitstream for $m = n = 1$ (DC only) and $m = n = 2$.

when one scans a document at 300 ppi, compresses it, and wants to preview the image in a typical 75 ppi monitor. In this case,

$$\begin{bmatrix} x_{00} & x_{01} \\ x_{10} & x_{11} \end{bmatrix} = \frac{1}{8}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} y_{00} & y_{01} \\ y_{10} & y_{11} \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{5}$$

$$= \frac{1}{8}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} zz'(0)q_{00} & zz(1)q_{01} \\ zz(2)q_{10} & zz(4)q_{11} \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{6}$$

where $zz'(0)$ is the quantized DC coefficient (DPCM decoded). Note that the above method just requires four integer multiplications, eight additions, plus scaling. The scaling by a power of two of an integer can be done by binary shift. Note that even $zz(n)$ can be calculated faster because only the relevant samples are to be decoded. After the last relevant sample has been assigned, one may skip the rest of the block by

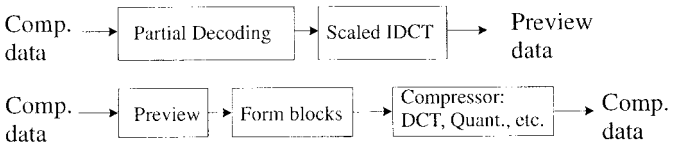


Fig. 8. Block diagram for resizing and previewing.

decoding the RRRR/SSSS symbols and shifting out the next SSSS samples (without bothering with the reconstruction of the coefficient amplitude and sign). Preview images extracted from the compressed data are shown in Fig. 7.

Other values of m and n apply as well and there is a fast DCT implementation for virtually any size. Noninteger relations can be used and, for this, one might start with the closest values of m, n and resize the decompressed image. At least it saves the computation of the full inverse DCT.

For upsampling the image, a block is formed as

$$\mathbf{Y}_{m\times n} = \begin{bmatrix} \mathbf{Y} & \mathbf{0}_{8\times n-8} \\ \mathbf{0}_{m-8\times 8} & \mathbf{0}_{m-8\times n-8} \end{bmatrix} \tag{7}$$

and the inverse transform is performed as in (4), i.e. an inverse $m \times n$ DCT.

Now, suppose one wants to resize the image and recompress it in a different size. Regrettably, there is no easy way to do it for a general scaling factor. This is because once the image is recompressed, the block size is dictated by JPEG as having 8×8 samples. By resizing the input blocks, we are forced to gather pixels from a plurality of input blocks to form a single 8×8 output block. The first step is basically the previewing process as shown in Fig. 8. The resulting samples are grouped into 8×8 blocks further applying the basic JPEG compression steps. In this case, savings only come from the previewing process replacing the 8×8 inverse DCT. Simplifications are obtained if m or n are both multiples of 8 or are either 1, 2, or 4. In this case, the original block is scaled into an integer number of new blocks, or several original blocks are scaled and grouped to form a single new block. These constraints largely simplify the process of forming new blocks (Fig. 8) but do not affect the other operations (previewing and compression).

IV. The Encoding Cost Map

The cost in an encoding process is defined as the number of bits required to encode a particular symbol or collection of symbols. In the JPEG context, we focus on the cost of encoding one block. An encoding cost map (ECM) can be formed where each entry in the map is related to the cost of each associated block in the document. The ECM is one key feature in processing compressed images and will be used in the next sections. By organizing ECM entries as pixels of an image, the ECM corresponding to the test image is shown in Fig. 9. The ECM has two basic properties.

First, the most important obstacle to perform any fast processing over JPEG-compressed data resides on the fact that one cannot determine where, in the compressed data stream, a block begins before decoding the preceding bitstream. If we could store or derive the ECM, we would be able to easily address individual blocks, enabling operations such as cropping, segmentation, rotation, etc. Thus, ECM provides addressability for the compressed data.

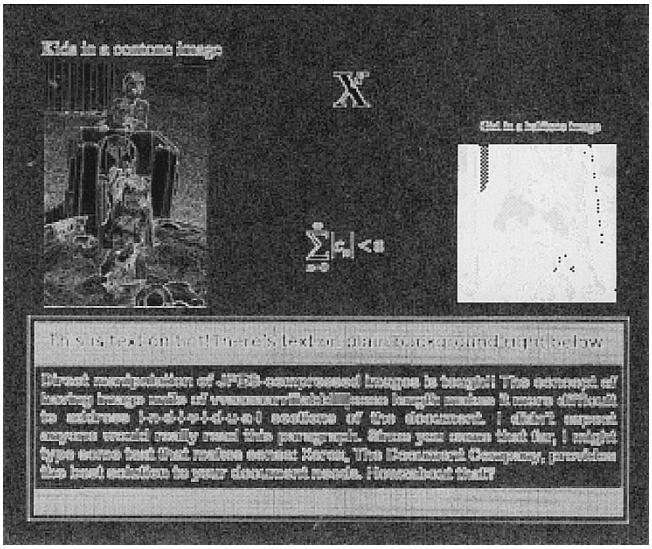


Fig. 9. ECM of the test image.

Second, the ECM conveys information pertaining to the activity of a block. Because of JPEG's compression strategy, smooth areas generate low ECM entries, while edges generate high ECM entries. Using the ECM we take advantage of the JPEG computation for modeling edges and such by only measuring the degree of success or failure to compress a particular block. Thus, the ECM provides edge activity information.

While those two key properties will be used in this paper, a more complete discussion of ECM's properties can be found in [20].

V. Side Information and Editing

In some cases, it might be advantageous to store side-information along with the compressed data, comprising the ECM and auxiliary data. The amount of side-information is variable and may depend on the compression rate and on the desired browsing speed. For moderate compression ratios (say $12:1$), the side information can be placed into JPEG[3] without significantly increasing the file size (compression ratio would drop to about $10:1$). This is a small price paid to allow image manipulation in the compressed domain. The alternative is to derive the side information at the decompression side by spending computation time. One example of side information for this purpose is: 1) the ECM encoded using any simple format,[4] 2) the sum of the entries of the ECM in each row, encoded using any simple format; 3) a decimated DC map, i.e., a map with N DC coefficients per row of blocks. If $N = 0$ the decimated DC map is not stored at all. Each row of blocks is segmented into N sections and each entry in the map is the DC coefficient of the leftmost block of each section. The decimated DC map can be encoded using any simple format. An illustration of this side information is given in Fig. 10.

[3] For example, using a comment or application field [1].

[4] ECM may include byte stuffing [1] or the data can be considered prefiltered to remove stuffed bytes;

Given the ECM and the sum of each row of it, one can easily seek any block in the compressed data. Actually, one can easily crop a region of the image. In Fig. 10, the sum of entries of the ECM in a row ($\mathrm{RL}[i]$) gives the total number of bits used to encode each block row and the ECM entries $\mathrm{ECM}[i,j]$ give the amount of bits used to encode each block. The DC map entry for row i and section k is denoted by $\mathrm{DC}[i,k]$. Assume one wants to crop the region comprising blocks *block*$[i,j]$ for *TOP* $\leq i \leq$ *BOTTOM*, *LEFT* $\leq i \leq$ *RIGHT*. Let the *LEFT* block of each row belong to section k and the *RIGHT* block belong to section l. Set $D_{\mathrm{last}} = 0$.

To crop a region and place it into a buffer, we can use the following steps: Skip *TOP* rows (row 0 trough row *TOP* $-$ 1). Perform the following steps for $i =$ *TOP* through $i =$ *BOTTOM*. Skip all blocks in row i until getting to the first block of section k. Let $D = \mathrm{DC}[i,k]$. For each block, until reaching the *LEFT* block, decode the DC coefficient, accumulate this value onto D, and skip the rest of the block. Decode the DC coefficient and accumulate this value onto D. Write $D - D_{\mathrm{last}}$ as a JPEG DC value to the buffer. If the original DC value of *block*$[i,$*LEFT*$]$ used t bits to be encoded, write the next $(\mathrm{ECM}[i,$*LEFT*$]-t)$ bits and all the bits used for the next *RIGHT-LEFT* blocks to the buffer. Set D_{last} as the actual DC value of the *RIGHT* block. This value is found in the same way, by accumulating the DC values (differences) of blocks in section l onto $\mathrm{DC}[i,l]$ until reaching block *RIGHT*. If i is not *BOTTOM* skip all blocks to the end of the row and repeat the process for the next i.

The final data in the buffer will correspond to the JPEG compressed data which would result from cropping the decompressed image and recompressing it (respecting the minimum unit of 8 $\times$ 8-pixels block).

There are alternative methods to manipulate JPEG compressed data. In one extreme, we can use the tools provided by JPEG, which are the restart markers. These markers are uniquely identifiable and placed periodically in the bitstream (period of an integer number of blocks). So, if one sets the restart marker to b blocks, one can search the data looking only for the restart markers (and counting them) in order to advance the file pointer to an specific position in the image. On the other hand, we can derive pointers to the compressed data blocks (instead of the ECM) and do some tricks to keep track of the DC coefficients [4]. We believe the proposed method might lie somewhere in between the two alternatives.

VI. Rotation and Mirroring

Rotation and mirroring of documents are frequent operations in the printing business. Simple operations such as rotation by 90° (and multiples of 90°), transposition and mirroring are usually done in the spatial domain. If the image is compressed, these operations would demand the decompression of the image for spatial rotation. Digital document processing would greatly benefit from the ability to rotate and mirror compressed images because of savings in storage and computation.

More general rotation and shearing are possible in the DCT domain [7]. In this case, the data blocks are no longer independent and that would force us to retransform the image.

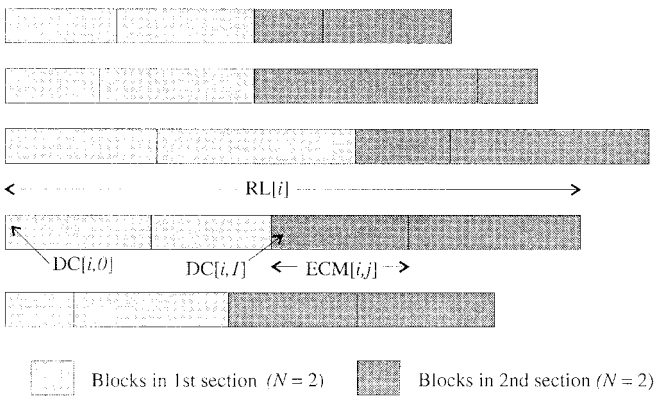


Fig. 10. Illustration of the compressed bitstream and its relation to the ECM and DC map entries. The ECM entry $\mathrm{ECM}[i,j]$ tells the size in bits of the jth block of the ith row of blocks. The sum of the ECM entries in the ith row (i.e., $\mathrm{RL}[i]$) is actually the length in bits of the codes composing the ith row. Each row is divided into N sections. The DC map entry $\mathrm{DC}[i,k]$ gives the actual DC value of the first block of the kth section.

It is unclear if it is simpler than it is to decompress the image and to apply a fast rotation in space domain. In any case, we concentrate in the subset of mirroring, transposition, and rotation by 90°, $-$90°, and 180° and show how they can be easily implemented in the compressed domain.

A. Intrablock Operations

Let $\mathbf{X}_{90}$, $\mathbf{X}_{-90}$, $\mathbf{X}_{180}$ be the image block $\mathbf{X}$ rotated by 90°, $-$90°, and 180° degrees, respectively. Also, let $\mathbf{X}_{\mathrm{VM}}$ and $\mathbf{X}_{\mathrm{HM}}$ be the blocks which are a vertical and horizontal mirrors, respectively, of $\mathbf{X}$. We denote by $\mathbf{Y}_{90}$, $\mathbf{Y}_{-90}$, $\mathbf{Y}_{180}$, $\mathbf{Y}_{\mathrm{VM}}$ and $\mathbf{Y}_{\mathrm{HM}}$ the transformed coefficients of $\mathbf{X}_{90}$, $\mathbf{X}_{-90}$, $\mathbf{X}_{180}$, $\mathbf{X}_{\mathrm{VM}}$ and $\mathbf{X}_{\mathrm{HM}}$, respectively (i.e. $\mathbf{Y}_{90} = \mathbf{D}_8\mathbf{X}_{90}\mathbf{D}_8^T$, etc.). Also, let $\mathbf{J}$ be the 8 $\times$ 8 reversing matrix defined as

$$\mathbf{J} = \begin{bmatrix} 0 & & \cdots & 0 & 1 \\ & & & 1 & 0 \\ \vdots & & & & \vdots \\ 0 & 1 & & & \\ 1 & 0 & \cdots & & 0 \end{bmatrix}$$

and let $\mathbf{V} = \mathrm{diag}\{1,-1,1,-1,1,-1,1,-1\}$. Then, one can check that the DCT matrix has the following property:

$$\mathbf{D}_8 = \mathbf{V}\mathbf{D}_8\mathbf{J} \tag{8}$$

i.e., half of its rows are even-symmetric and half of them are odd-symmetric. The following relations are also true:

$$\begin{aligned} &\mathbf{X}_{90} = \mathbf{J}\mathbf{X}^T \quad \mathbf{X}_{-90} = \mathbf{X}^T\mathbf{J} \quad \mathbf{X}_{180} = \mathbf{J}\mathbf{X}\mathbf{J} \\ &\mathbf{X}_{\mathrm{HM}} = \mathbf{X}\mathbf{J} \quad \mathbf{X}_{\mathrm{VM}} = \mathbf{J}\mathbf{X}. \end{aligned} \tag{9}$$

Hence, putting all these results together we get, after some algebraic manipulation, to

$$\begin{aligned} &\mathbf{Y}_{90} = \mathbf{V}\mathbf{Y}^T \quad \mathbf{Y}_{-90} = \mathbf{Y}^T\mathbf{V} \quad \mathbf{Y}_{180} = \mathbf{V}\mathbf{Y}\mathbf{V} \\ &\mathbf{Y}_{\mathrm{HM}} = \mathbf{Y}\mathbf{V} \quad \mathbf{Y}_{\mathrm{VM}} = \mathbf{V}\mathbf{Y} \end{aligned} \tag{10}$$

which are simple relations to rotate and mirror the block in the DCT domain. Remember that $\mathbf{VY}$ means to change the sign of the coefficients of every odd-numbered row of $\mathbf{Y}$, while $\mathbf{YV}$

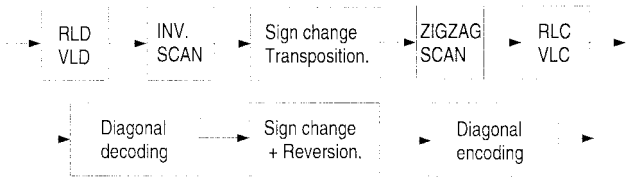


Fig. 11. Flow graph for block mirroring and rotation. At the top is the basic algorithm requiring decoding and inverse zigzag prior to the operations of changing sign and transposing. This algorithm makes it easier to apply existing JPEG building blocks. At the bottom is the faster algorithm which decodes one diagonal of quantized coefficients at a time, performing reversal and sign inversion.

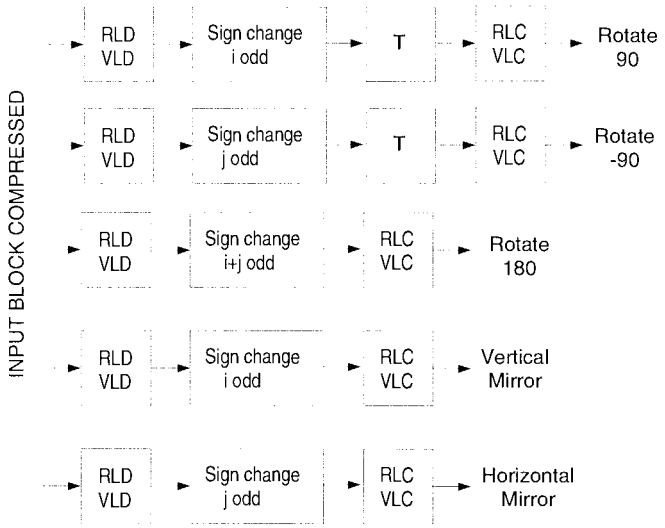


Fig. 12. Flow graph for each operation of block mirroring and rotation. T means transposition.

means the same for the columns of $\mathbf{Y}$. These operations are trivial and would spare us from having to perform an inverse DCT to rotate the block in space domain.

The DC coefficient is not affected by these operations and sign change is independent of the quantization. Thus, these operations can be applied directly to the decoded coefficients without involving DPCM and quantization. The resulting flow-graph for rotation-mirroring of one block of compressed data is shown on the top of Fig. 11.

The sequence of VLD, inverse zig-zag scanning, rotation, zig-zag scanning, plus VLC, can be further simplified. We can decode data relative to one diagonal (according to the zig-zag path) of the block at a time. Having the coefficients of one diagonal in hands, we can change the necessary coefficient signs (according to which operation is desired), reverse the short sequence of coefficients (if transposition is needed), and encode the coefficients again (see Fig. 11).

As a remark, the operation in diagonal data can be tricky. The diagonals are not independent because of the counting of zero coefficients. When one reverses the coefficients in the diagonal, the counting may change too. This can be resolved by storing the run-of-zeros counting before opening one diagonal, calculating the new counting after the data is reversed and storing this number for the next diagonal operation.

Each operation is illustrated in Fig. 12. Note that all operations can be hardcoded avoiding the zigzag scanning and its inverse. For example, sign inversion of all c_{ij} for i odd (i.e., reversing the sign of every odd-numbered row) is equivalent to change the sign of $zz(n)$ for $n \in \{2, 4, 7, 9, 11, 13, 16, 18, 20, 22, 24, 26, 29, 31, 33, 35, 36, 38, 40, 44, 46, 48, 49, 51, 53, 55, 57, 58, 60, 62, 63\}$. The same can be done for reversing the signs of the odd-numbered columns.

B. Interblock Operations

Assume an image is composed by scan lines and each scan line is composed by several bit-strings of different sizes, instead of fixed-bit-number pixels, i.e., an image made of variable-length pixels. Assume the file containing the variable length pixels is stored sequentially. Rotation and mirroring are possible if we know how many bits are spent to encode each pixel (i.e., if we derive or pre store the ECM). Alternatively, we can also store the offset to reach the first bit of each pixel. In both cases we can always readily address individual pixels in the image.

Let the images (composed by variable size pixels) before and after the rotation and mirroring be labeled "A" and "B," respectively. To generate image "B" it will be necessary to address nonsequential positions inside image "A." This can be done with the aid of 32-b pointers and the map with the number of bits spent to encode each pixel. Each pointer stores the number of bits to seek in image "A" in order to reach the first bit of the pixel to be placed in image "B". Since we know how many bits were used to encode each pixel we can easily move the pointer n pixels to the left or to the right. Therefore, we avoid having to recalculate offsets every time we want to address one single pixel and, at the same time, we avoid storing a long offset pointer for every pixel in the image. The pointers are positioned in image "A" at the beginning of the rotation (mirroring) operation pointing to the first pixels that will be written in image "B." As each pixel is written, the pointers are changed to address the next block. Of course, the initial position and movement of these pointers depend on which of the five operations described here will be applied. Now, we replace the notion of *variable size pixel* by *block of compressed data*.

C. Image Rotation and Mirroring

The rotation or mirroring are performed in the following steps.

- Read compressed data, decode and reencode each block. As the blocks are being decoded they are rotated (mirrored) using the fast method described.
- The DC, which is a difference between the actual DC and the DC of the previous block is replaced by the actual DC before writing back each block. Skip this step for all but the first block in a row in case of vertical mirroring.
- Store the length in bits of each block (ECM) in a separate array.
- Perform interblock rotation on the blocks already internally rotated (mirrored) placing them in their final memory location. Write blocks in their definitive order:

from left to right and from top to bottom of the image after rotation (mirroring). The initial positioning and movement of pointers for the interblock rotation are gathered from the array with the length in bits for each block, which was created in the previous step.

- Rewrite the DC coefficient of the blocks (as the difference between the actual DC of a block and that of a previous one) as the blocks are written. Skip this step for all but the first block in a row in case of vertical mirroring.

The intermediate buffer is stored compressed. The process is composed by simple pointer operations and fast intrablock rotation. Therefore, this method largely saves storage and computation.

VII. Segmentation

We address the segmentation of an image into specific regions such as those containing halftones, text, continuous-tone pictures, etc., without decompressing and buffering the whole image. The idea is to browse the compressed data to extract the necessary information in order to perform the segmentation. Applications include: selective postprocessing, rendering hints, and recompression. In selective postprocessing, one can filter background, use antiblocking filtering in contone images and antiringing filtering near text and halftone edges. Rendering hints are common practice in the printing business, since text and graphics are rendered differently than contone images. Recompression will be discussed later on.

Various segmentation and classification approaches have been investigated and most of them are based on examining the decompressed image and perhaps extracting high-frequency information (edges) to localize letters, graphics, etc. Examples are [14]–[19]: 1) decompressing the image and applying segmentation in space domain, 2) segmenting the DC map, 3) using the AC energy (sum of the square or of the absolute values of the AC coefficients in a block) map. The first implies full JPEG decompression, high computation and large storage space. Also, blurring, ringing, and blocking artifacts can complicate the segmentation. The DC component alone implies a large loss in resolution within a simple framework. The AC energy is more complex but captures intrablock activity information. We do not address any of these but introduce a new method combining the information regarding high-frequency contents of the AC energy method with the simplicity of the DC map method.

We use the ECM for segmentation because it is simple to compute, results in a reduced image, and captures intrablock activity. The only JPEG stage necessary is the VLD, which does not even need to be fully implemented. If the ECM is sent as side-information, no computation is necessary. Several different blocks may generate the same ECM entry and the pure histogram analysis of the ECM may not suffice for any robust segmentation algorithm. Regions with very dense concentration of bright pixels in the ECM may indicate a halftone, while text letters may have its borders delimited by bright ECM pixels, etc. The ECM corresponding to our test image is shown in Fig. 9, where entries were converted to pixel brightness. Halftones and tint areas are characterized by large concentration of bright pixels. Edge of letters in text areas are bright, thus forming a sparser distribution of bright pixels. Contone regions contain midrange pixels with sparse bright areas. The background is basically a dark (nonuniform) area. A more sophisticated variation combines the ECM data with the DC coefficients map. For example, background detection is easily made by selecting blocks with low ECM and large DC magnitude.

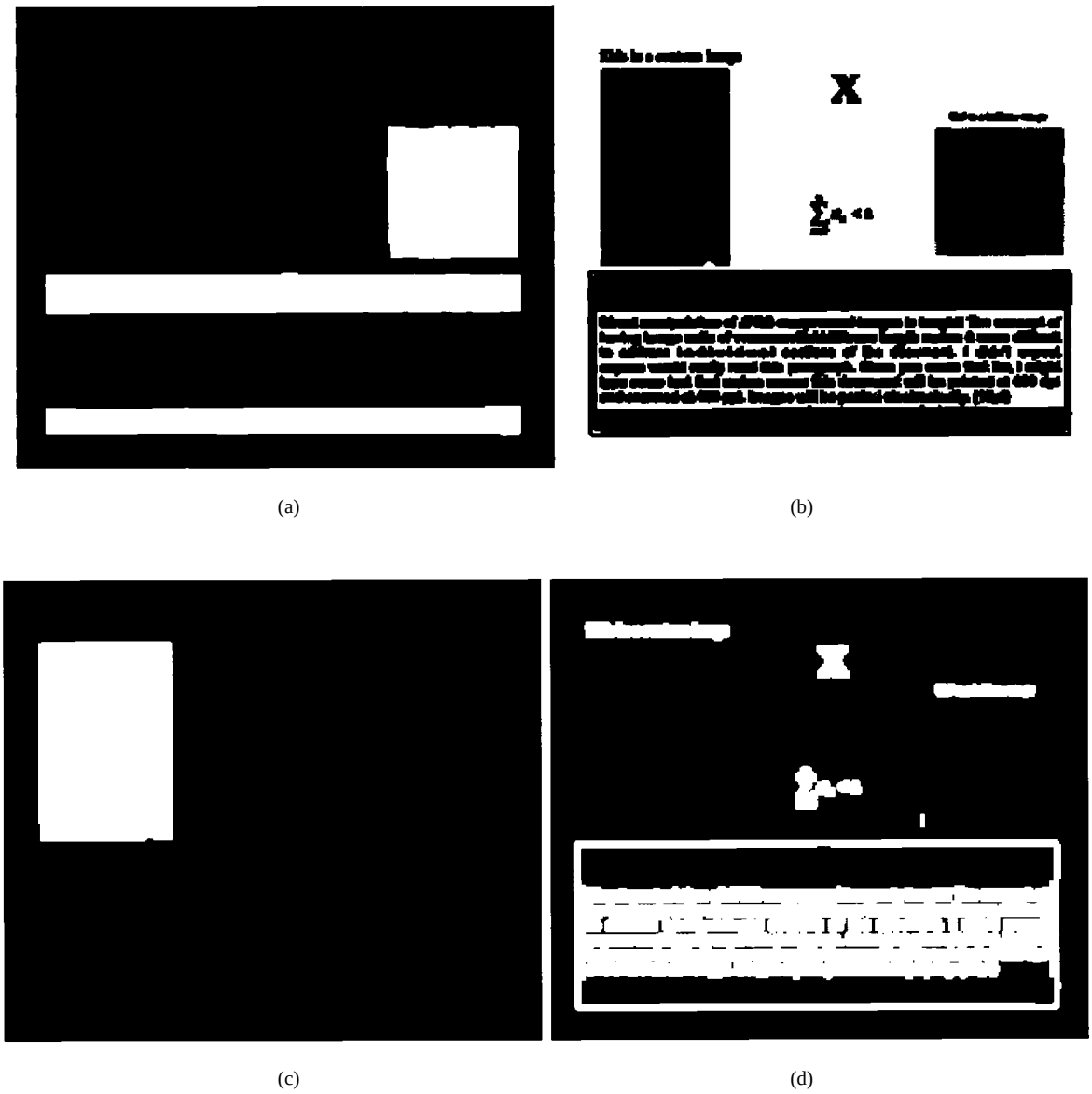
A more complete discussion of ECM-based segmentation technique can be found in [20]. A segmentation algorithm has to take into account classes of images (and their statistics) as well as the quantizer table used, since both will alter the appearance of the ECM data and affect the output. We refrain from getting into such an extensive discussion here by just presenting an outline of a simple segmentation algorithm starting with the images shown in Figs. 9 and 7. We intend to illustrate the main concepts. An example of a simple nonadaptive segmentation algorithm based on pixel oriented operations is now presented.

In order to detect halftone one might look for large concentrations of very bright pixels in the ECM. In this case, for example, we can filter the ECM and threshold the output. For a 3×3 averaging filter and threshold of 100 we get an initial binary mask to work. We have to separate halftones from the text letter edges. The difference resides on the fact that halftones and tints are large and dense patches in the mask, while letters and graphics are separated by background. So, we can process the mask with binary morphological operators [21]. First, we close the small gaps there might be in the halftone patches using a small structuring element (e.g., 5×5). This also fuses the text. We now apply one of our assumptions: the halftone patches are much larger than text letters. For 12-point letters, text can be eliminated by an erosive operation, while the tint and halftone areas are just reduced. Therefore, we can directly apply a 5×5 element in an opening operation (erosion followed by dilation). Finally, we dilate the image a little only to provide safety margins as well as more uniform boundary regions. The resulting mask is shown in Fig. 13(a), which indicates the areas of possible halftones and tints.

Background can be efficiently detected by combining the ECM with the DC image.[5] Bright DC pixels whose ECM value is low are often part of the background. We detect background by filtering the ECM and the DC image before comparing the results to thresholds. Using a 3×3 averaging filter, the resulting mask shown in Fig. 13(b) indicates blocks whose filtered DC value is brighter than 220 and filtered ECM value is below 60.

We now delete the background and halftone areas from the original ECM and construct a mask S with the remaining entries. The task is reduced to the separation of contone regions from text and graphics. Again, we assume contone regions are much larger than the letter dimensions, i.e., contone is a large dense patch while the text is not. Hence, we perform an opening operation with a large 9×9 structuring element, followed by some dilation in order to provide a safety margin as well as uniform boundaries. The resulting mask is shown in Fig. 13(c).

[5] DC values are scaled (divided by eight) to fit in the range 0–255.



(a) (b)

(c) (d)

Fig. 13. Segmentation masks. (a) Halftone. (b) Background. (c) Contone. (d) Text and graphics.

We delete the contone regions from mask S and the remaining mask is the mask for text and graphics which can be optionally dilated a little for safety and to yield more uniform boundaries. The resulting mask indicating the regions of text and graphics is shown in Fig. 13(d). Further details in this algorithm are found in [20].

VIII. Recompression

In some cases, the document may undergo a light compression where it suffers virtually invisible distortion. For storage for longer periods, one may find it suitable to further compress the document using JPEG. Given that the document has already been compressed we address the recompression of an image without fully decompressing it. For this we can apply several methods and we discuss 1) simple requantization of the DCT coefficients, 2) thresholding [23], and 3) background replacement.

A. Requantization

If each block is decompressed and recompressed, the quantized coefficients c_{ij} are scaled by the respective quantizer step ($\hat{y}_{ij} = c_{ij}q_{ij} = \text{round}(y_{ij}/q_{ij})q_{ij}$). Inverse DCT is not necessary because the reconstructed block will be transformed again anyway. Let the new quantizer steps be p_{ij}. Hence, the reconstructed coefficients are $u_{ij} = \text{round}(c_{ij}q_{ij}/p_{ij})p_{ij}$. This might cause some problems if p is not a multiple of q. For example, let $q_{ij} = 2$, $p_{ij} = 3$, and $y_{ij} = 4.9$. So, $c_{ij} = 2$ and $y_{ij} - \hat{y}_{ij} = 0.9$. If we have used a larger quantizer step such as $q_{ij} = 3$, the quantizer error would be 1.1, instead. However, if we requantize the coefficients using $p_{ij} = 3$, then $u_{ij} = 3$ and the accumulated error is 1.9, which is larger than using $q_{ij} = 3$. This is just a reminder that rounding errors may accumulate. There is not much that can be done in this sense, except to better estimate the reconstructed coefficient. We attempt to do so by assuming that for input signals with

typical image statistics, the DCT coefficients are reasonably modeled by a Laplacian probability density function (pdf) [2], as

$$f(x) = \frac{\alpha}{2} e^{-\alpha|x|} \tag{11}$$

which is a zero-mean pdf with variance $2\alpha^2$. If the Laplacian-modeled variable is quantized using uniform step sizes, the only information available to the receiver is that the original coefficient was in the interval $\gamma - t \leq y_{ij} \leq \gamma + t$, where $\gamma = c_{ij}q_{ij}$ and $t = q_{ij}/2$. The trivial solution suggested in JPEG [1] is to reconstruct the coefficient in the center of the interval as $\hat{y}_{ij} = \gamma$, which simplifies implementation. We can assume positive values without loss of generality. The optimal reconstruction lies in the centroid of the pdf for the interval [22], i.e.

$$\gamma' = \frac{\int_{\gamma-t}^{\gamma+t} \lambda f(\lambda)\, d\lambda}{\int_{\gamma-t}^{\gamma+t} f(\lambda)\, d\lambda} = \gamma + \frac{1}{\alpha} - t \coth(\alpha t). \tag{12}$$

Note that it means a constant bias toward origin: $\delta = \gamma - \gamma' = t \coth(\alpha t) - \frac{1}{\alpha}$. For different coefficients (different step sizes and variances) we have

$$\delta_{ij} = q_{ij} \coth(\alpha_{ij} q_{ij}/2) - \frac{1}{\alpha_{ij}}. \tag{13}$$

Given the coefficient variances (σ_{ij}^2), we can estimate the α parameters as $\alpha_{ij} = \sqrt{2}/\sigma ij$. The variances can be easily calculated for a given image model [2] or can be estimated from one or several images. Therefore, we can precalculate all δ_{ij} and subtract the coefficients magnitude by δ_{ij} as

$$\hat{y}_{ij} = c_{ij}\ q_{ij} - \text{sign}[c_{ij}]\delta_{ij} \tag{14}$$

where sign$[0] = 0$. For example, computing the variances directly from the test image in Fig. 5, and applying the default luminance quantizer table we get the δ_{ij} table (rounded to integer) as

$$\begin{array}{cccccccc} 0 & 0 & 0 & 1 & 6 & 10 & 16 & 25 \\ 0 & 0 & 1 & 1 & 10 & 14 & 24 & 11 \\ 0 & 0 & 1 & 3 & 18 & 20 & 27 & 24 \\ 1 & 1 & 3 & 4 & 24 & 33 & 34 & 27 \\ 4 & 7 & 16 & 27 & 33 & 54 & 51 & 38 \\ 3 & 7 & 18 & 22 & 40 & 45 & 53 & 43 \\ 11 & 24 & 32 & 38 & 51 & 57 & 57 & 49 \\ 31 & 27 & 45 & 44 & 56 & 47 & 50 & 46. \end{array}$$

B. Thresholding

Thresholding is a technique intended to provide spatial adaptivity to the quantization in JPEG [23], [24]. JPEG is called an adaptive coder because it fixes the quantizer steps in such a way that different coefficients demand different bit-rates. However, the quantization process is static because the same quantizer step (q_{ij}) is applied to all coefficients c_{ij}; therefore, the same maximum distortion is allowed to all c_{ij}. Since we cannot change the quantizer entries on a block by block basis,[6] thresholding arises as a technique to examine each quantized coefficient and to decide if it is worthwhile (in a rate-distortion sense) to keep this coefficient or not. If it is decided to throw away the coefficient, it is set to zero (thresholded). The thresholding procedure can provide adaptation in a block basis, can improve signal-to-noise ratios (SNR's) by 1 dB [23] and is perfectly compatible with JPEG because the decoder does not know the thresholded coefficient ever existed. The reader is referred to [23] for in-depth analysis of the method and its complete description. Also, in [24] it is combined with quantizer and VLC optimization for maximum JPEG performance that rivals much more sophisticated coders. We present here a simplification of the method in [23] for speed purposes. If compression speed is not a concern, we strongly recommend implementing the method described in [23] and [24].

In the simplified approach, we look at each and every nonzero quantized coefficient in a block. For a nonzero coefficient $zz(n)$, assume the next nonzero coefficient in the vector order is $zz(l)$ at index l. For simplicity let $l - n < 16$, to avoid ZRL symbols. Let $zz(n)$ have RRRR_1 zeros before it and category SSSS_1, while $zz(l)$ has $\text{RRRR}_2 = l - n - 1$ zeros before it and category SSSS_2.

- The cost of sending $zz(n)$ is: b_1 b to encode RRRR_1/SSSS_1, plus SSSS_1 b (offset and sign—see Section II).
- The cost of sending $zz(l)$ is: b_2 b to encode RRRR_2/SSSS_2, plus SSSS_2 b.
- If $zz(n)$ is set to zero, the cost of sending $zz(l)$ is the cost of sending a number with category SSSS_2 but with $\text{RRRR}_1 + \text{RRRR}_2 + 1$ zero samples in front of it (b_3 b), plus SSSS_2 b for offset and sign. b_3 includes the possible cost of sending ZRL symbols.

Focusing on $zz(n)$, the cost $R(n)$ of keeping the coefficient (not thresholding) is the cost of sending $zz(n)$ and $zz(l)$ minus the cost of sending $zz(l)$ if $zz(n) = 0$. $R(n \mid zz(n) \neq 0) = b_1 + b_2 - b_3 + \text{SSSS}_1$. The benefit of not thresholding $zz(n)$ is a decrease in reconstruction distortion. Assuming a weighted mean-square-error measure, and using the decompressed image without thresholding as a reference, this distortion is $w(n)|zz(n)q(n)|^2$ since the DCT is orthogonal, where $q(n)$ are the q_{ij} scanned into zigzag form. The COST/BENEFIT (rate/distortion) ratio is

$$\nu(n) = \frac{b_1 + b_2 - b_3 + \text{SSSS}_1}{w(n)|q(n)zz(n)|^2}. \tag{15}$$

$\nu(n)$ is compared to a threshold τ and we set $zz(n) = 0$ whenever $\nu(n) > \tau$.

If the run of zeros amounts to more than 16, one has to include the cost of ZRL in the above discussion, which is a trivial task. This simplified technique by itself can improve SNR in single compression by a proper choice of τ. For recompression, it is preferable to combine thresholding with requantization.

[6] Recent JPEG extensions (JPEG Part 3—ISO DIS 10 918-3) adds an alternative quantizer table scaling factor for each block, in order to allow MPEG compatibility. However, all step sizes are scaled by the same amount [25].

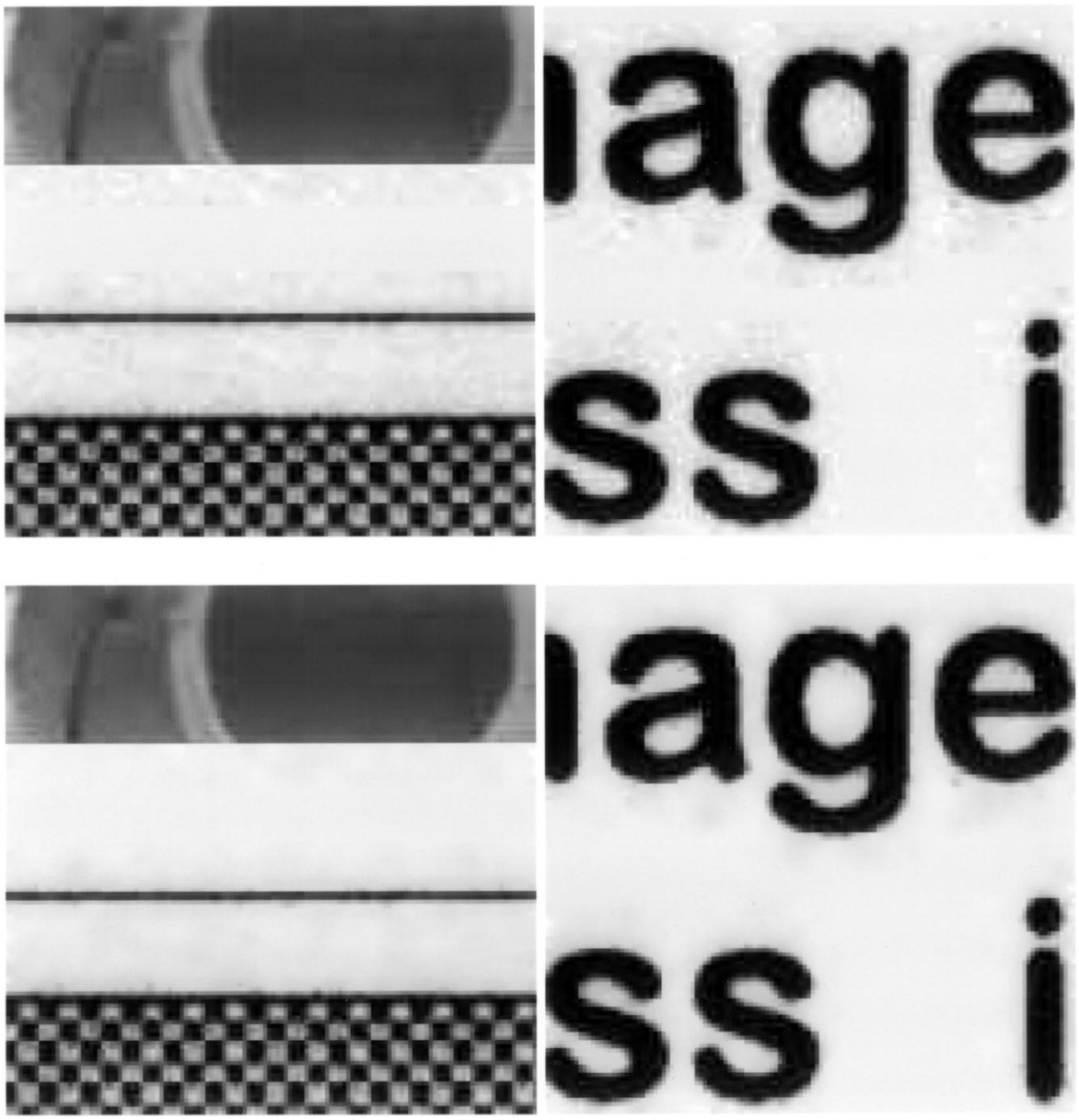


Fig. 14. Stitching problems at background boundaries. The reconstructed images are processed by a σ-filter that is tuned to process pixels above a luminance threshold close to the background region. Top: parts of the reconstructed image. Bottom: same as above after processing.

C. Background Removal

Another approach for recompression is to eliminate redundant information in the document image. For example, the background (paper) is not as smooth as one might expect and is composed by bright ECM pixels distributed among dark ones. Thus, several blocks spend an excessive number of bits to encode the background.

If we use the background detection method previously described, we can modify the compressed data to contain smooth flat blocks on the background, hence, spending fewer bits. In a horizontal run of consecutive background blocks, the first block is assigned to an average value. The difference between this value and the DC of the block right before the run is encoded, followed by an EOB. After that, all blocks are encoded using a precomputed symbol meaning "null DC difference" followed by EOB. Using default luminance VLC tables, this sequence is 001 010. The average value of the first block in the run can be: the block's own DC value, average of DC values of background blocks, or a predefined value.

Using our test image, by substituting the background blocks by flat patches of brightness 235 (value found in average for the particular scanner settings and paper used), we increased the compression ratio from 10.68 to 13.86. In other words, by eliminating the background texture information we shrunk the compressed file by almost a quarter (about 23%). In any case, it is likely there will be *stitching* problems, which are discontinuity artifacts present at the borders between two different regions that were put together artificially. In our case, this happens between the original and the artificial background regions, since the original background data is present around other objects such as text, images, etc. Fig. 14 shows two segments of the test image after replacing the background

by constant data. To clean the respective images, we used a σ-filter in the regions surrounding the detected background region. A σ-filter is an average window which just includes neighbor samples that are within $\pm\sigma$ gray levels from the center-pixel level. Furthermore, the filter is just turned on if the center pixel is brighter than a predefined threshold, and is located close to the background regions. In the example shown in Fig. 14, $\sigma = 32$, the neighborhood is 5×5, and the threshold was set to 200.

IX. Conclusion

We presented techniques for processing compressed images. We used simple algorithms for scaling, previewing, cropping, rotating, mirroring, segmenting, and recompressing compressed images. Just algorithms which can lead to substantive reductions in memory requirements and complexity were presented.

We tried to avoid discussing obvious processing derived from the fact that DCT is a linear transform. For example, it is obvious that the DCT of a linear combination of input blocks will result in the same linear combination of the respective transformed blocks. That account for fading, mixing, etc. Also, it is simple to control image brightness by modifying the DC coefficient. However, this method may fail for sharp edges. Too see this, imagine that a flat block and a sharp edge can have the same DC coefficient. If the change in brightness is not linear, dark and bright pixels should change differently, and that will not happen if one only operates on the DC. In other words, brightness modifications may work for smooth blocks, but can fail otherwise and create jagged edges. In any case, these cases were covered elsewhere.

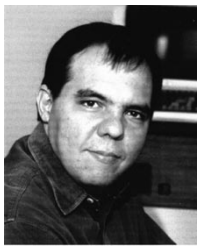
The goal is to save computation, therefore, to reduce costs and increase speed, while reducing buffer (memory) costs. The main savings come from the fact that one might be able to use a small buffer to keep the image in compressed format. Furthermore, the DCT is a relatively expensive process, and by cutting JPEG basic steps we are also able to cut processing time. It is evident that processing compressed data may be a key to digital document processing, facilitating the manipulation and processing of high-resolution images at a reasonable cost. The suite of available algorithms continuously grows. Operations such as halftoning, color correction, etc., will be presented in a future opportunity.

Acknowledgment

The author wishes to thank R. Eschbach and P. Fleckenstein for discussions related to this paper. Dr. Eschbach introduced the author to the topic of processing compressed images, and the use of ECM for segmentation is found in more detail in a joint paper [20].

References

[1] W. B. Pennebaker and J. L. Mitchell, *JPEG: Still Image Compression Standard*. New York: Van Nostrand, 1993.
[2] K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, Applications*. New York: Academic, 1990.
[3] B. C. Smith and L. A. Rowe, "Algorithms for manipulating compressed images," *IEEE Trans. Comput. Graph. Appl.*, vol. 13, Sept. 1993.
[4] R. F. Miller and S. M. Blonstein, "Compressed image virtual editing system," U.S. Patent 5 327 248, July 1994.
[5] M. Chen and Z. Shae. "Video mixing technique using JPEG compressed data," U.S. Patent 5 257 113, Oct. 1993.
[6] M. Shneier and M. Abdel-Mottaleb, "Exploiting the JPEG compression scheme for image retrieval," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 18, pp. 849–853, Aug. 1996.
[7] B. Shen and I. K. Sethi, "Scanline algorithms in the JPEG DCT compressed domain," *J. Electron. Imag.*, vol. 5, pp. 182–190, Apr. 1996.
[8] P. P. Vaidyanathan, *Multirate Systems and Filter Banks*. Englewood Cliffs, NJ: Prentice-Hall, 1993.
[9] R. L. de Queiroz and R. Eschbach, "Fast downscaled inverses for images compressed with M-channel lapped transforms," *IEEE Trans. Image Processing*, vol. 6, pp. 794–807, June 1997.
[10] J. M. Adant *et al.*, "Block operations in digital signal processing with applications to TV coding," *Signal Process.*, vol. 13, pp. 285–397, Dec. 1987.
[11] K. N. Ngan, "Experiments on 2D decimation in time and orthogonal transform domains," *Signal Process.*, vol. 11, pp. 249–263, Oct. 1986.
[12] B. Natarajan and B. Vasudev, "A fast approximate algorithm for scaling down digital images in the DCT domain," in *Proc. IEEE Int. Conf. on Image Processing*, Washington, DC, 1995, vol. II, pp. 241–243.
[13] S. Martucci, "Image resizing in the DCT domain," in *Proc. IEEE Int. Conf. on Image Processing*, Washington, DC, 1995, vol. II, pp. 244–247.
[14] H. T. Fung and K. J. Parker, "Segmentation of scanned documents for efficient compression," in *Proc. SPIE: Visual Communications and Image Processing*, Orlando, FL, 1996, vol. 2727, pp. 701–712.
[15] S. N. Srihari, "Document image understanding," in *Proc. Int. Symp. Circuits and Systems*, 1986, pp. 87–96.
[16] T. Pavlidis and J. Zhou, "Page segmentation and classification," *CVGIP: Graph. Models Image Process.*, vol. 54, pp. 484–496, Nov. 1992.
[17] D. Dunn, T. Weldon, and W. Higgins, "Extracting halftones from printed documents using texture analysis," in *Proc. Int. Conf. Image Processing*, Lausanne, Switzerland, 1996, vol. II, pp. 225–228.
[18] K. Murata, "Image data compression and expansion apparatus, and image area discrimination apparatus," U.S. Patent 5 535 013, July 1996.
[19] Z. Fan, "Segmentation-based JPEG image artifacts reduction," U.S. Patent 5,495,538, Feb., 1996.
[20] R. de Queiroz and R. Eschbach, "Fast segmentation of JPEG-compressed documents," *J. Electron. Imag.*, vol. 7, pp. 367–377, Apr. 1998.
[21] E. R. Dougherty, *An Introduction to Morphological Image Processing*, vol. TT9. Bellingham, WA: SPIE Press, 1992.
[22] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Boston, MA: Kluwer, 1992.
[23] K. Ramchandran and M. Vetterli, "Rate-distortion optimal fast thresholding with complete JPEG-MPEG decoder compatibility," *IEEE Trans. Image Processing*, vol. 3, pp. 700–704, Sept. 1994.
[24] M. Crouse and K. Ramchandran, "Joint thresholding and quantizer selection for decoder compatible baseline JPEG," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing*, Detroit, MI, 1995, vol. 4, pp. 2991–2994.
[25] D. Lee, "JPEG: New enhancements and future prospects," in *Proc. IS&T's 48th Annu. Conf., Track I—Imaging on the Information Superhighway*, 1995, pp. 58–62.



Ricardo L. de Queiroz (S'89–M'95) received the B.S. degree from Universidade de Brasilia, Brazil, in 1987, the M.S. degree from Universidade Estadual de Campinas, Brazil, in 1990, and the Ph.D. degree from University of Texas at Arlington, in 1994, all in electrical engineering.

From 1990 to 1991, he was a Research Associate with the DSP research group at Universidade de Brasilia. In 1994, he became a Teaching Assistant in the Electrical Engineering Department, University of Texas at Arlington. He joined Xerox Corporation, Webster, NY, in August 1994, where he is currently a Member of the Research Staff at the Color and Digital Imaging Systems Laboratory. His research interests are multirate signal processing and filterbanks, image and signal compression, color imaging, and processing of compressed images.

Dr. de Queiroz is currently Vice-Chair of the Rochester Chapter of the IEEE Signal Processing Society. He received the Academic Excellence Award in 1993 from the Electrical Engineering Department, University of Texas at Arlington.