PROCESSING JPEG-COMPRESSED IMAGES

Ricardo L. de Queiroz

Xerox Corporation
800 Phillips Rd, M/S 128-27E, Webster, NY 14580
queiroz@wrc.xerox.com

ABSTRACT

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. 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.

1. INTRODUCTION

We are particularly concerned with images manipulated in the printing business, which involves scanning and printing documents and pictures at a large resolution. These documents are often compressed. The recommendation ISO DIS 10918-1 known as JPEG (Joint Picture Expert Group) has become a *de facto* standard for lossy compression of still images. 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. We also assume monochrome images (single block MCU) and ignore header formats to simplify the presentation.

The objective of this paper is to present techniques that allows image processing in the "JPEG-compressed" domain. JPEG compression is performed by a series of operations: transform, quantization, zigzag scanning, 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 (see Fig. 1). 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. The motivation is to save memory and/or to improve speed. Other authors have also addressed the topic of processing/analyzing JPEG compressed data. See for example [2]–[6] for further results in this topic.

Due to space limitations fine details of the presentation were omitted, which can be found in [7].

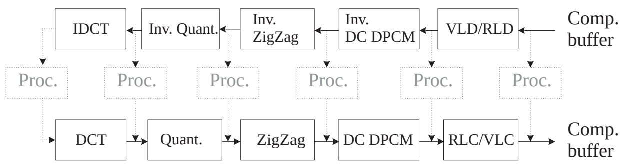

Figure 1: Processing a compressed buffer. The goal is to perform as few JPEG operations as possible before and after the processing stage.

2. PREVIEWING AND RESIZING

We briefly discuss the previewing method and leave further references and discussions on the subject to [8]. Algorithmic details are also given in [7].

The idea is to quickly extract a downsampled image from a compressed buffer for previewing purposes. Let us denote the lower-frequency $m \times n$ coefficients of a DCT transformed block as $\mathbf{Y}_{m\times n}$. 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:

$$\mathbf{X}_{m\times n} = \frac{\sqrt{mn}}{8} \mathbf{D}_m^T \mathbf{Y}_{m\times n} \mathbf{D}_n. \qquad (1)$$

where $\mathbf{D}_n$ is the $n \times n$ DCT matrix. In other words, we perform a scaled inverse $m \times n$ DCT over $\mathbf{Y}_{m\times n}$. To compute $\mathbf{Y}_{m\times n}$ 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 decoding the RRRR/SSSS symbols and shifting out the next SSSS samples (without bothering with coefficient amplitude and sign) [1]. For non-integer relations, 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, $\mathbf{Y}_{m\times n}$ is formed by padding the block with zero samples.

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, since blocks in JPEG have fixed size. The first step is basically the previewing process as shown in Fig. 2. 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

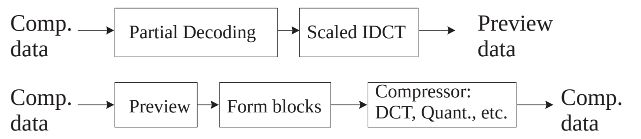

Figure 2: Block diagram for resizing and previewing.

to form a single new block. These constraints simplify the process of forming new blocks (Fig. 2) but do not affect the other operations (previewing and compression).

3. COST MAPS AND SIDE INFORMATION

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 (or MCU). 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 most important obstacle to perform any fast processing over JPEG-compressed data resides on its sequential compression mode. Since the number of bits spent to encode each block is variable, one cannot decode a block before decoding the preceding bit-stream (let us ignore restart markers [1] for now). If we store or derive the ECM, we can easily address individual blocks, enabling operations such as cropping, segmentation, rotation, etc. In some cases, it might be advantageous to store side-information along with the compressed data, comprising the ECM and auxiliary data. For moderate compression, the side information (placed into JPEG in a comment or application field) would not significantly decrease the compression ratio. The alternative is to derive the side information at the decompression side by spending computation time.

One example of side information (which may be encoded using any simple format) is: the ECM (may include byte stuffing [1] or they can be removed as pre-processing); the sum of the entries of the ECM in each row; and a horizontally decimated DC map, i.e., a map with N DC coefficients per row of blocks. An illustration of this side information is given in Fig. 3. A more detailed description of side information can be found in [7], while a better discussion of the ECM properties can be found in [9].

As an application exampe, one can easily crop a region of the image. In Fig. 3, the sum of entries of the ECM in a row (RL[i]) gives the total number of bits used to encode each block row and the ECM entries 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 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 $D_{last} = 0$.

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 =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.

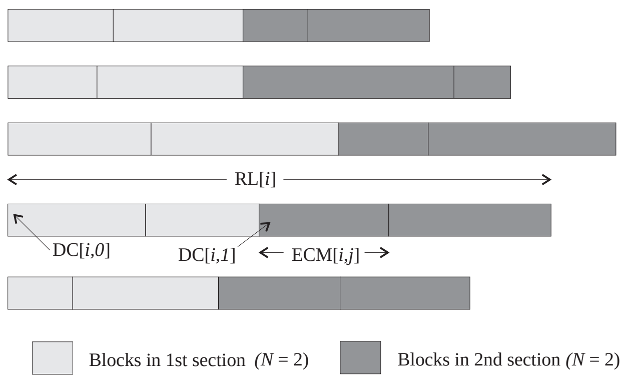

Figure 3: On side information for block addressing.

Decode the DC coefficient and accumulate this value onto D. Write $D - D_{last}$ 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 (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 $DC[i,l]$ 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 8×8-pixels block).

There are alternative methods to manipulate JPEG compressed data. For example, using restart markers [1] or as in [3]. We believe the proposed method might lie somewhere in between the two alternatives.

4. ROTATION AND MIRRORING

Rotation and mirroring of documents are frequent operations in the printing business (usually done in the spatial domain). We concentrate in mirroring, transposition, and rotation by 90, -90, and 180 degrees and show how those operations can be easily implemented in the compressed domain. More general rotation and shearing is possible in the DCT domain [6], although it is unclear if it is simpler than it is to decompress the image and to apply a fast rotation in space domain.

Let $\mathbf{X}_\theta$, be the image block $\mathbf{X}$ rotated by θ degrees, and let $\mathbf{X}_{VM}$ and $\mathbf{X}_{HM}$ be vertical and horizontal mirrors, respectively. We use the same notation for the transformed block $\mathbf{Y}$. Also, let $\mathbf{J}$ be the 8×8 reversing matrix and let $\mathbf{V} = diag\{1,-1,1,-1,1,-1,1,-1\}$. Then, one can check that the DCT matrix has the following property:

$$\mathbf{D}_8 = \mathbf{V}\mathbf{D}_8\mathbf{J} \tag{2}$$

The following relations are also true: $\mathbf{X}_{90} = \mathbf{J}\mathbf{X}^T$, $\mathbf{X}_{-90} = \mathbf{X}^T\mathbf{J}$, $\mathbf{X}_{180} = \mathbf{J}\mathbf{X}\mathbf{J}$, $\mathbf{X}_{HM} = \mathbf{X}\mathbf{J}$, $\mathbf{X}_{VM} = \mathbf{J}\mathbf{X}$. Hence, we get, after some algebraic manipulation, to:

$$\begin{gathered}\mathbf{Y}_{90} = \mathbf{V}\mathbf{Y}^T \quad \mathbf{Y}_{-90} = \mathbf{Y}^T\mathbf{V} \quad \mathbf{Y}_{180} = \mathbf{V}\mathbf{Y}\mathbf{V} \\ \mathbf{Y}_{HM} = \mathbf{Y}\mathbf{V} \quad \mathbf{Y}_{VM} = \mathbf{V}\mathbf{Y}\end{gathered} \tag{3}$$

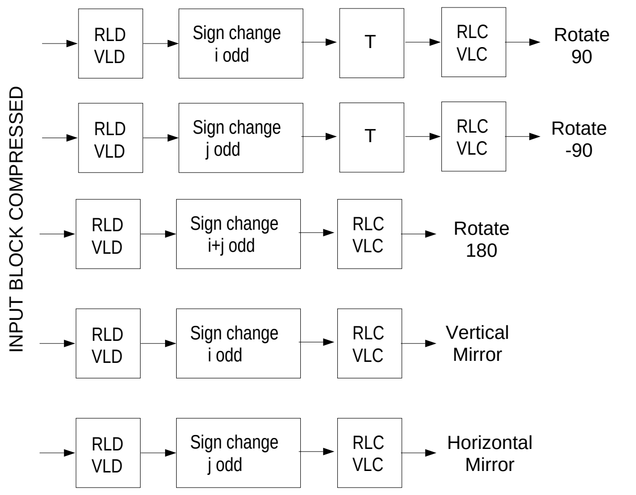

Figure 4: Flow graph for each operation of block mirroring and rotation. T means transposition.

which are simple relations to rotate and mirror the block in the DCT domain. Note that these trivial operations just involve sign changes of few coefficients and simple data reordering, while sparing us from performing space domain rotation. A basic diagram is shown in Fig. 4 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 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. Further details can be found in [7].

Assume an image is composed by variable-length pixels. Rotation and mirroring are possible if we know how many bits are spent to encode each pixel (i.e. if we derive or store the ECM). For rotation, it is simple to address non-sequential positions inside such an image with the aid of 32-bit pointers and the map with the number of bits spent to encode each pixel. After a pixel is copied, the pointers are modified to point to the next pixel to be processed. The analogy between variable-length pixel and an encoded block yields the following overall algorithm. Read compressed data, decode and re-encode each block. As the blocks are being decoded they are rotated (mirrored) using the fast method described. The 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 inter-block 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 inter-block 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 intra-block rotation. Therefore, this method largely saves storage and computation.

5. SEGMENTATION

We can perform segmentation of a document into specific regions such as those containing halftones, text, continuous-tone pictures, etc., without decompressing and buffering the whole image. For this, we browse the compressed data to extract the ECM on which segmentation is performed. With knowledge of the ECM, as discussed, regions can be cropped or replaced without decompressing the image. For details in this algorithm please see [9].

6. 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.

6.1. Requantization

If each block is decompressed and recompressed, the quantized coefficients c_{ij} are scaled by the respective quantizer step q_{ij} and requantized with new quantizer steps p_{ij}. This might cause some problems if p is not a multiple of q, i.e. rounding errors may accumulate. There is not much that can be done in this sense, except to better estimate the reconstructed coefficient. As JPEG uses uniform step sizes, the only information available to the receiver is that the original coefficient was in the interval $\gamma - t \le y_{ij} \le \gamma + t$, where $\gamma = c_{ij}q_{ij}$ and $t = q_{ij}/2$. The trivial solution 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 and model DCT coefficients by a Laplacian probability density function (PDF) whose decaying parameter is α. Then, optimal reconstruction value lies in the centroid of the PDF for the interval [10], i.e.

$$\gamma' = \frac{\displaystyle\int_{\gamma-t}^{\gamma+t} \lambda f(\lambda) d\lambda}{\displaystyle\int_{\gamma-t}^{\gamma+t} f(\lambda) d\lambda} = \gamma + \frac{1}{\alpha} - t\ coth(\alpha t). \tag{4}$$

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

$$\delta_{ij} = q_{ij}\ coth(\alpha_{ij} q_{ij}/2) - \frac{1}{\alpha_{ij}}. \tag{5}$$

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 or can be estimated from one or several images. Therefore, we can pre-calculate all δ_{ij} and subtract the coefficients magnitude by δ_{ij} as

$$\hat{y}_{ij} = c_{ij}\ q_{ij} - sign[c_{ij}]\delta_{ij} \tag{6}$$

where $sign[0] = 0$.

6.2. Quick (local) Thresholding

Thresholding is a technique intended to provide spatial adaptivity to the quantization in JPEG [11],[12]. since we cannot change the quantizer entries on a block by block basis[1]. Thresholding examines each quantized coefficient (in a rate-distortion sense) and if it is decided discard the coefficient, it is set to zero (thresholded). We present here a simplification of the method in [11] for speed purposes. If compression speed is not a concern we recommend the methods in [11],[12].

In the simplified approach, we look at each and every non-zero quantized coefficient in a block. Let $zz(n)$ be a non-zero coefficient in the zigzag-arranged vector whose next non-zero coefficient appears at index l. Assume the RRRR/SSSS symbol for $zz(n)$ spends b_1 bits to be encoded, so that $zz(n)$ costs $b_1 + SSSS$ bits [1]. Assume the RRRR/SSSS symbol for $zz(l)$ spends b_2 bits to be encoded. If $zz(n)$ is set to zero the cost of sending the RRRR/SSSS symbol for $zz(l)$ usually increases because of the extra run of zeros. Assume that, if $zz(n)$ is set to zero, the cost of the RRRR/SSSS symbol for $zz(l)$ is b_3. If S is SSSS for $zz(n)$, the net cost $R(n)$ of encoding $zz(n)$ is: $R(n|zz(n) \neq 0) = b_1 + b_2 - b_3 + S$. The benefit of not thresholding $zz(n)$ is the decrease in distortion given by the information conveyed in $zz(n)$. Assuming a mean-square-error (MSE) measure, and using the decompressed image without thresholding as a reference, this distortion is $|zz(n)|^2$ since the DCT is orthogonal. If a weighted MSE is used, the distortion can be measured as $w(n)|zz(n)|^2$. The local COST/BENEFIT (rate/distortion) ratio is

$$\nu(n) = \frac{b_1 + b_2 - b_3 + \mathrm{SSSS}_1}{w(n)|zz(n)|^2} \tag{7}$$

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

6.3. 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 (see [9]) 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 pre-computed symbol meaning "null DC difference" followed by EOB. Using default luminance VLC tables, this sequence is 001010. 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. In test documents (with reasonable amount of text and graphics) background replacement typically increased the compression ratio by more than 20% (can be as high as 50%!). Stitching problems present at the borders between artificial and natural background can be easily removed using a simple non-linear filter [9].

[1]Recent JPEG extensions (JPEG Part 3 - ISO DIS 10918-3) adds an alternative quantizer table scaling factor for each block. However, all step sizes are scaled by the same amount.

7. CONCLUSIONS

We discussed techniques for processing compressed images, using simple algorithms for scaling, previewing, cropping, rotating, mirroring, segmenting, and recompressing compressed images. Those algorithms can lead to substantive reductions in memory requirements and complexity were presented. Due to space limitations much off the details were left to the full-paper version [7] and segmentation issues were left to a companion paper [9].

We tried to avoid discussing obvious processing derived from the fact that DCT is a linear transform. That account for fading, mixing, brighteness control, etc.

8. REFERENCES

[1] W. B. Pennebaker and J. L. Mitchell, *JPEG: Still Image Compression Standard,"* New York, NY: Van Nostrand Reinhold, 1993.

[2] B. C. Smith and L. A. Rowe, "Algorithms for manipulating compressed images," *IEEE Trans. Computer Graphics and Applications,* vol. 13, Sep. 1993.

[3] U. S. Patent 5,327,248, *Compressed Image Virtual Editing System,* R. F. Miller, S. M. Blonstein.

[4] U. S. Patent 5,257,113, *Video Mixing Technique Using JPEG Compressed Data,* M. Chen, and Z. Shae.

[5] M. Shneier and M. Abdel-Mottaleb, "Exploiting the JPEG compression scheme for image retrieval," *IEEE Trans. on PAMI,* Vol. 18, pp. 849–853, Aug. 1996.

[6] B. Shen and I. K. Sethi, "Scanline algorithms in the JPEG DCT compressed domain," *J. of Electronic Imaging,* vol. 5 (2), pp. 182–190, Apr. 1996.

[7] R. de Queiroz, "Processing JPEG-Compressed Images," under review for *IEEE Trans. on Image Processing.*

[8] R. L. de Queiroz and R. Eschbach, "Fast downscaled inverses for images compressed with M-channel lapped transforms," *IEEE Trans. on Image Processing,* Vol. 6, pp. 794–807, June, 1997.

[9] R. de Queiroz and R. Eschbach, "Segmentation of compressed documents," in this proceedings.

[10] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression,* Kluwer Academic, 1992.

[11] K. Ramchandran and M. Vetterli, "Rate-distortion optimal fast thresholding with complete JPEG-MPEG decoder compatibility," *IEEE Trans. on Image Processing,* vol. 3, pp. 700–704, Sep. 1994.

[12] M. Crouse and K. Ramchandran, "Joint threhoslding and quantizer selection for decoder compatible baseline JPEG," *Proc. ICASSP* vol. 4, pp. 2991–2994, 1995.