

Color Correcting JPEG Compressed Images

R.Victor Klassen
Raja Balasubramanian
Ricardo de Queiroz
Color and Digital Imaging Systems
Xerox Corporation

Abstract

Many high end printing systems store and/or transmit compressed images to save bandwidth, memory, and disk space. Prior to printing, the system will decompress and then color correct the image to compensate for the characteristics of a particular printer. In the conventional architecture, the decompression step converts the image from some encoded format (e.g. JPEG with Huffman encoding) into the spatial domain. In this domain, the color image may be stored in some standard device independent color space such as IJG/CCIR YCrCb, SMPTE RGB or Kodak Photo Ycc. The ensuing color correction transforms colors from this space to device coordinates. Our proposed technique uses information about the JPEG compressed format to gain efficiency in the color transformation.

Introduction

The use of color in an imaging system increases the storage, memory, and bandwidth by a factor of at least three over the use of grayscale. Image compression therefore becomes critical, particularly in high volume printing applications. To this end, JPEG is probably the most widely used compression standard [1]. The compression is often applied in a device independent color encoding; hence, the image must be color corrected to device coordinates after decompression. The color correction step often involves the use of 3-D lookup tables (LUT) with 3-D interpolation [2]. For large color intensive images, the cost of such a method of color correction can be significant.

Our approach involves parsing the color correction problem into two processes, applying an expensive process to a low resolution version of the image before decompression, and an inexpensive operation to the entire image after decompression. The ratio of resolutions controls the tradeoff between quality and speed.

The next two sections provide the background to the method: first a review of JPEG compression and decompression, and then a description of color correction, especially as we have parsed the problem in order to gain efficiency. The following section describes our method. We close with a discussion of speedup results and image quality effects.

JPEG Compression and Decompression

JPEG is a widely used standard for lossy image compression. In its most popular mode (baseline), the compression employs a sequence of operations which exploit statistical and visual redundancies through signal analysis in a transform domain. We only describe it in brief.

Compression begins by dividing the image into

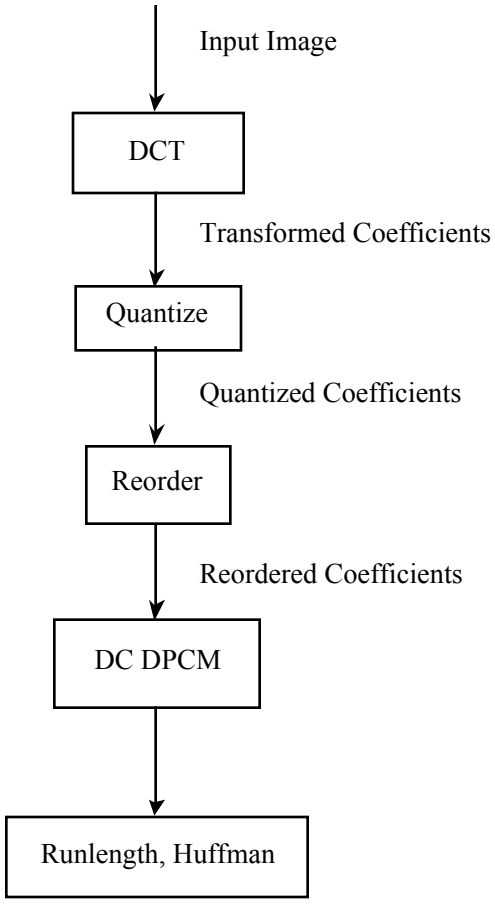

Figure 1. Basic blocks of JPEG compression.

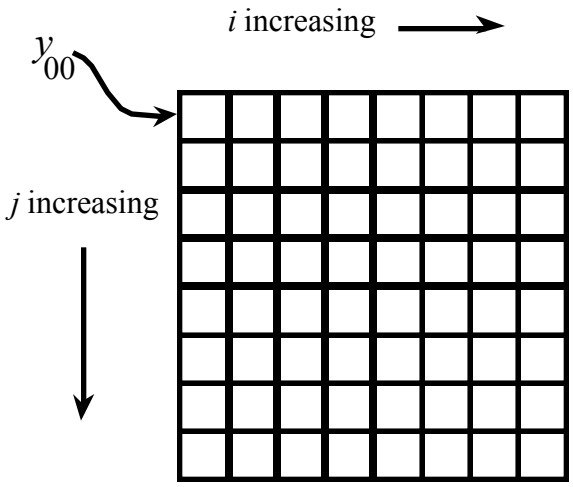

Figure 2. Layout of a transformed block. Spatial frequency increases with increasing i and j.

8×8 blocks, which are treated almost independently. Figure 1 shows the steps in compressing a single block. The first step, labeled DCT, transforms the block into the spatial frequency using the discrete cosine transform. In such a transform, the DC component y_{00}, represents the average value over the block, and 63 AC components y_{ij} represent higher frequency bands, as shown Figure 2. This figure shows components in the horizontal and vertical directions ordered from low to high frequency in left to right, and top to bottom order, respectively.

The second step quantizes transformed samples y_{ij} (that is, scales them using 64 static *quantizers,* discarding fractional parts). Other than numerical inaccuracies in the DCT, this is the only part of the process that discards information.

The third step re-orders the coefficients as shown in Figure 3 for more efficient encoding. Re-ordering follows a zig-zag path intended to group the low frequency components of both dimensions toward the beginning of the block.

The final two steps provide the compression itself. The DC term is subtracted from the DC term of the previous block and the difference is stored. The AC terms are stored with zeroes run length encoded. Fast coding techniques such as Huffman coding are applied to the data preconditioned in this way. The data that is most visually significant is grouped at the beginning of the encoding for each block.

In order to fully decompress a block the encoding steps must be reversed. To decompress part of the block components, such as the lower-frequency N×N portion, we use lower cost versions of each of the steps. We recover only as much information about the high order coefficients as we need in order to construct the N×N coefficients we desire. Once the reduced set of DCT components is identified, we use

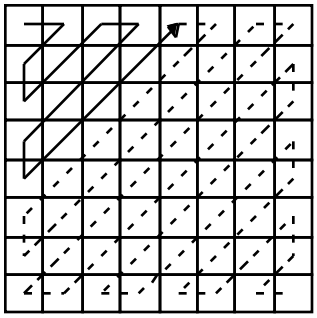

Figure 3. Zigzag re-ordering. Coefficients are re-ordered to follow the pattern indicated.

a simpler inverse DCT to recover an approximate block of the image. (See [3] for details on reduced inverse DCTs).

Color correction

Correcting for RGB devices

RGB devices include monitors (cathode-ray tube based, and liquid crystal display based), film recorders, and some printers that present an RGB interface.

We separate the color correction problem into two phases. First, we account for any mismatch between the image and device primaries, and interactions among the device primaries. Second, we linearize the individual primaries to some measured standard. The first phase of the problem requires at least a 3×3 matrix multiplication, and possibly a full three dimensional table lookup, followed by interpolation [2, 4]. The second phase involves using tone reproduction curves (TRCs), implemented as one-dimensional table lookups, and is therefore cheaper than the first phase. Normally, if the first phase involves a 3 dimensional table lookup with interpolation, the second phase can be combined into the same table lookup, yielding the entire result. For reasons that will become evident, we choose to keep the two phases separate.

Given image data in a device independent luminance-chrominance form, such as YC_rC_b, the first phase of color correction converts these coordinates to a device dependent luminance chrominance space, $Y'C_1'C_2'$. Note that the relation between YC_rC_b and RGB need not be kept between $Y'C_1'C_2'$ and the device dependent RGB space (i.e. R′G′B′). In other words, we can change the luminance-chrominance color space at no cost because we use table look-ups to convert and conversion can be done off-line. The advantage of choosing a suitable luminance-chrominance space is to reduce operations when converting data to R′G′B′, as compared to the conversion YC_rC_b-RGB as described in IJG/CCIR.

The disadvantage is that we may end up with data in different color spaces for the same DCT block. We will discuss this topic later on. The device luminance-chrominance space is defined by a simple transformation from device RGB (or CMY):

$$Y' = 3\,R'/8 + G'/2 + B'/8 \quad (1)$$
$$C_1' = B' - Y'$$
$$C_2' = R' - Y'$$

The inverse transform from device $Y'C_1'C_2'$ to device RGB is simply:

$$R' = C_2' + Y \quad (2)$$
$$G' = Y - 3\,C_2'/4 - C_1'/4$$
$$= Y - C_2' + (C_2' - C_1')/4$$
$$B' = C_1' + Y,$$

which involves a total of 5 additions and one shift. Linearization TRC's are then applied to the device RGB values prior to output.

Correcting for CMYK Devices

For four-colorant printers, color correction comprises a transformation from a device independent 3-tuple to printer CMYK. Again, the transformation can be divided into two parts.

Phase I:

This phase converts the device independent 3-tuple (*e.g.* colorimetric RGB) into a device dependent 3-tuple with the same color coordinate orientation (*e.g.* printer RGB). This transformation, which takes into account various inter-colorant interactions (*e.g.* unwanted spectral absorptions of the colorants), is typically implemented by a 3-D lookup table (LUT) [4]. The LUT comprises a 3-D grid of nodes in colorimetric RGB space. Output printer RGB values are precomputed and stored at these node locations. Given an arbitrary colorimetric RGB input, the transformation is obtained as follows: 1) retrieve the rectangular cell enclosing the input point; 2) extract the printer RGB values at appropriate vertices of the cell; 3) calculate the printer RGB corresponding to the given input colorimetric RGB by applying some type of 3-D interpolation among the printer RGB values at the cell vertices. This first phase is typically the more computationally intensive phase.

Phase II:

This phase largely consists of simpler 1-dimensional transformations. Examples of these are undercolor removal (UCR) and gray component replacement (GCR), that convert printer RGB to printer CMYK [4]; and individual linearization of each of the C, M, Y, K colorants using TRCs.

Figure 4 summarizes the two phases. Often, both phases are combined into a single LUT that maps the device independent 3-tuple directly to CMYK.

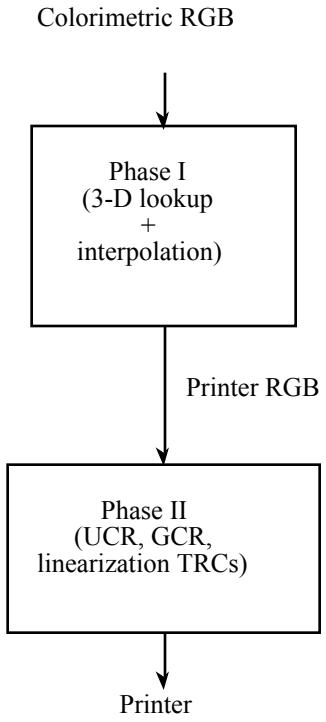

Figure 4. The two phases of color correction for CMYK devices

In conventional applications, the image is first decompressed, and then the color correction transform is applied, as shown in Figure 5.

Fast color correction

We will describe the CMYK case; the RGB is simpler, and easy to derive from the description of the

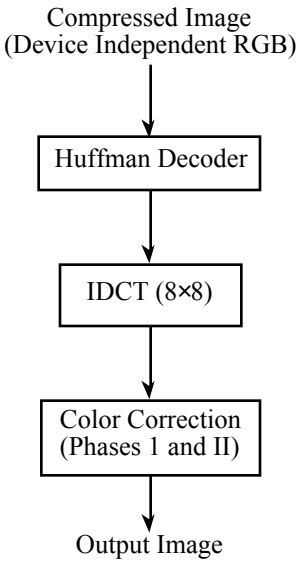

Figure 5. Conventional approach where decompression is followed by color correction

CMYK case. We assume that images are stored in colorimetric (device independent) RGB space; and JPEG compressed format. We exploit the fact that the human visual system is most sensitive to errors at low spatial frequencies. (This is especially true of chromatic errors.) We thus propose an efficient color correction scheme that performs the more expensive (Phase I) correction only on a sub-image corresponding to the first few low order DCT coefficients. The simpler (Phase II) color correction is then performed on the entire image. The steps are summarized below:

1. Compute the N×N subblock of low order DCT components, N < 8;
2. Perform the N×N inverse DCT, and obtain a reduced block of an RGB image of reduced resolution;
3. Apply Phase I color correction to the pixels of the N×N reduced block by 3-D lookup and interpolation, yielding an image in printer RGB coordinates;
4. Perform a forward N×N DCT on the reduced block and place the resulting DCT components into their respective positions in the original 8×8 DCT block;
5. Perform an 8×8 inverse DCT to the resulting block to reconstruct the image at full resolution.
6. Apply Phase II color correction to all image pixels of the resulting image.

These steps are summarized in Figure 6. Note that since the N×N subblock of partially corrected pixels is re-inserted into the 8×8 block in step 4, it is important that Phase I correction preserves the orientation of color space (*i.e.* colorimetric RGB to device RGB; colorimetric YC_1C_2 to device YC_1C_2; etc).

In the RGB case discussed above, we use colorimetric YC_rC_b and device YC_1C_2. As YC_1C_2 is just a simplification of YC_rC_b we expect the two color orientations to be very close. The DCT block right before inverse DCT contains low-frequency components from device YC_1C_2 but high-frequency components from colorimetric YC_rC_b. As the difference between colorimetric and device YC_1C_2 (or YC_rC_b) is largely more significant than the differences between colorimetric YC_1C_2 and colorimetric YC_rC_b the slightly different high-frequency components (whenever they are present) do not impact the quality of the color reproduction.

Results and Summary

Results indicate significant computational savings over the conventional approach (*i.e.* full decompression, followed by RGB to CMYK by 3-D LUT and interpolation for all image pixels), with little loss in image quality. The quality-cost trade-off clearly depends on the number of low order DCT coefficients (*i.e.* N^2) to which Phase I color correction is applied. We carried out experiments with N = 1, 2, 3. When N = 1, the computational savings are a factor of 64; and the image quality may be acceptable for some markets. When N = 3, there is some computation savings; and the image quality is very close to that achieved with conventional decompression and color correction. The computational savings from the various algorithms are summarized in Table 1.

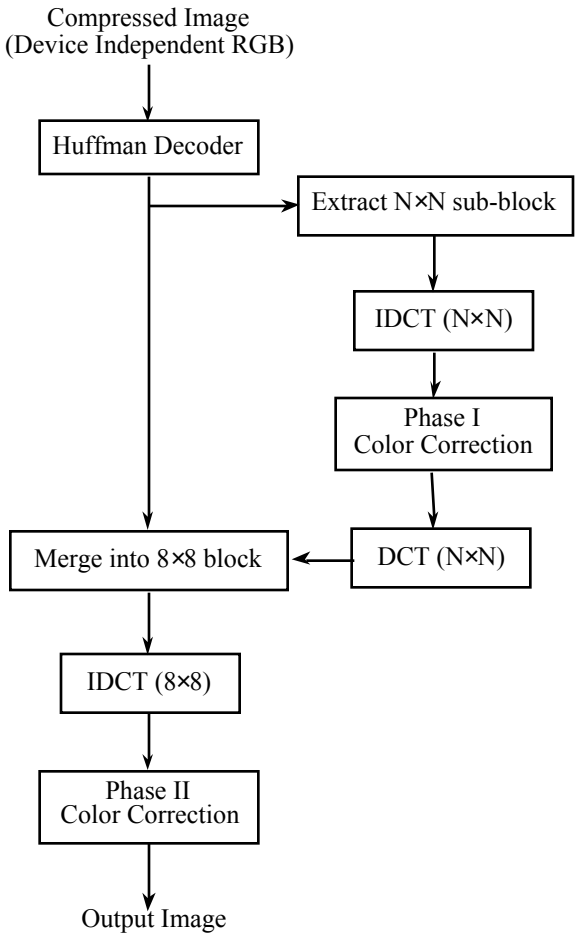

Figure 6. Proposed scheme of combining color correction with JPEG decompression

Operation	N=1	N=2	N=3
Lookups	40.9	37.2	29.7
Comparisons	18.8	13.8	6.3
Additions	48.8	43.3	33.6
Multiplications	98.8	95.3	89.5
Shifts	46.9	37.5	34.4

Table 1: Percentage savings of proposed approach (Figure 6) over conventional approach (Figure 5).

References

1. W. Pennebaker and J. Mitchell, *JPEG: still image data compression standard*, Van Nostrand Reinhold 1993.
2. H. R. Kang, *Color Technology for Electronic Imaging Devices*, SPIE optical engineering press, Bellingham WA, 1997.
3. 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.
4. J. A. C. Yule, *Principles of Color Reproduction*, John Wiley & Sons, New York, 1967