The JPEG Image Compression Algorithm

John W. O'Brien (obrienjw@colorado.edu)

***Abstract*— One of the most common methods of representing graphical information in a digital form is by spatially sampling the color components of an image—red, green, and blue for example. These digitized samples can be placed in a long sequence for the purpose of storage and transmission. For any reasonably detailed image, this string of samples can become very long. As of this writing, a typical digital camera will produce images in excess of three million samples. Consequently, various methods are employed to translate this representation into a compact form. A particular approach to *image data compression*—known as the JPEG algorithm—is examined herein.**

The basis for the JPEG algorithm is the Discrete Cosine Transform (DCT) which extracts spatial frequency information from the spatial amplitude samples. These frequency components are then quantized to eliminate the visual data from the image that is least perceptually apparent, thereby reducing the amount of information that must be stored. Finally, the redundant properties of the quantized frequency samples are exploited through Huffman coding to produce the compressed representation. Each of these steps is reversible to the extent that an acceptable approximation of the original space-amplitude samples can be reconstructed from the compressed form. This paper examines each step in the compression and decompression sequence, with special emphasis on the DCT.

***Index Terms*— compression, jpeg, discrete cosine transform, orthonormal basis transformation, vector projection**

I. Introduction

A. Overview

The objective of JPEG image compression is simple: to store the data necessary to reconstruct a digital image using as little space as possible while maintaining enough visual detail so that storing the image is actually worthwhile. It should be noted early that, unlike compression techniques used with other types of data—like financial records, or word processing documents—JPEG is a lossy algorithm, meaning that visual information is selectively discarded in order to improve the compression ratio. Once discarded, this information can not be recovered, but it is selected to minimize the effect on the perception of someone viewing the reconstructed image. In order to determine which information can be removed, and to perform the removal in a straight forward fashion, the image data is converted from a space-intensity form to a space-frequency form.

B. History

In the late 1980's and early 1990's, a joint committee, known as the Joint Photographic Experts Group (JPEG), of the International Standards Organization (ISO) and the Comité Consultatif International Téléphonique et Télégraphique (CCITT) developed and established the first international compression standard for continuous-tone images. A good summary of their work can be found in [1]. The standard they produced [2] has since become ubiquitous on the Internet in the field of digital photography. It has also been extended and adapted for use in the encoding and compression of digital videos.

C. Parallels

In general, people do not develop an intuition about spatial frequency to the same degree as they do about temporal frequency. The idea that a time signal has frequency components, and that the signal can be analyzed and modified on the basis of these components, should be familiar to anyone who plays or listens to music. In that case the time signal is the varying air pressure that manifests sound, and the frequency components are the different pitches that we can identify as notes. In fact, we characterize notes by describing their relative fundamental frequency: low or high. Even on a more subtle level, it is commonly understood that there is a connection between the way something sounds and the relative contributions of its different frequency components.

On the other hand, fewer people would be immediately comfortable discussing the spatial frequency characteristics of something, and might have difficulty envisioning how an image would look if you "turned up" the high frequency elements of it. However, a basic, qualitative grasp of spatial frequency in images is not hard. In an image, low frequency elements represent smooth, gradual changes in color and brightness, whereas higher frequency contributions are present when an image contains sharp contrast and fine detail.

To get a general idea of the approach taken by the JPEG algorithm, let us think briefly about a simple audio signal. A typical human ear can detect audio signals with frequencies as low as $20Hz$ and as high as $20kHz$; a factor of a thousand. However, most human speech is confined within a range of $100Hz$ to $8kHz$; less than a factor of a hundred. This is the reason that the audio of a telephone conversation is encoded at a rate of $64kbps$ while one channel of an audio CD takes over $700kbps$. Many of the frequencies unnecessary for the transfer of voice information have been removed from the telephone data stream, but the CD retains data representing frequencies outside that range in order to facilitate richer, fuller reproduction of tonal "information."

D. Technical details

A digitized sample is known as a *picture element* or *pixel* for short. Each pixel is most commonly represented by three 8-bit unsigned integers. The three values correspond to the intensity of three color components[1] (e.g. red, green, and blue) on a scale of 0 to 255. JPEG compression operates on each color component separately, so the discussion that follows will deal only with 8-bit pixels. It may be useful for the reader to visualize grayscale images (like a black and white photograph), wherein each pixel can be stored as an 8-bit value indicating the amount of whiteness, instead of amounts of redness, greenness, and blueness.

There are several variants—modes of operation—of the JPEG algorithm included in the standard. We are principally concerned with the Baseline DCT Sequential mode. This compression method takes place in four stages, which will be described in detail shortly.

1) The uncompressed source data is separated into 8×8 blocks of pixels. 128 is subtracted from the value of each pixel so that the new effective range is from -128 to 127.
2) Each block is transformed into an 8×8 block of frequency coefficients.
3) These coefficients are quantized.
4) An entropy encoder is applied to the quantized coefficients.

II. Model development and mathematical formulation

A. Definitions

An uncompressed image is stored as a two dimensional array of pixels with width W and height H. We will use the convention that pixels are indexed starting with $(0, 0)$ in the upper left corner of the image.

B. The Discrete Cosine Transform (DCT)

This transformation can be developed starting with the Fourier series, extending to the Fourier transform, switching from continuous to discrete, and finally using a cosine instead of a complex exponential for the basis. The details of that progression will not be reproduced here because they are thoroughly covered elsewhere [3]. Instead, it will be most useful to investigate the DCT in the context of vector spaces, projections, and matrices.

1) Projections: We will begin by reviewing simple projections in $\mathbb{R}^3$, with the Euclidean dot product. Inspiration is taken from Ch. 5 of [4]. Given an orthonormal basis we can find the projection of any vector $\mathbf{v} \in \mathbb{R}^3$ onto each of the basis vectors, thereby giving a set of coefficients which can be used to reproduce $\mathbf{v}$ as a linear combination of the basis vectors. What does this mean? Well, a vector under the standard basis of $\mathbf{e_1}$, $\mathbf{e_2}$, and $\mathbf{e_3}$ (known to some people as $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$, and $\hat{\mathbf{k}}$) gives the number of units along the three principal axes; x, y, and z. For instance, the vector $(3, 5, 7)^\intercal$ means that you move three units along the x-axis, five units parallel to the y-axis, and finally seven units parallel to the z-axis. The standard basis is implied. However, if it was agreed that we are to use a different set of basis vectors, then that same vector would represent a different point in space. For example, let us twist the standard basis counterclockwise about the z-axis by $45°$. The new basis vectors are then given by

$$\mathbf{v_1} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix}, \mathbf{v_2} = \begin{pmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix}, \mathbf{v_3} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}. \tag{1}$$

Note that one of the vectors did not change. With this set the same set of coefficients $(3, 5, 7)^\intercal$ now means to move three units along the line $x = y$, five units parallel to the line $x = -y$, and seven units parallel to the z-axis. Under the standard basis, this location would be represented as $(-2/\sqrt{2}, 8/\sqrt{2}, 7)^\intercal$.

In the preceding example, we observed how one set of coefficients can represent two different points in space depending on the basis used. Next, and more relevant to the DCT, we show how a single point can be represented by two different sets of coefficients.

Let

$$\begin{aligned} \mathbf{v} &= \begin{pmatrix} -\frac{4}{\sqrt{2}} \\ \frac{4}{\sqrt{2}} \\ 8 \end{pmatrix} \\ &= c_1\mathbf{v_1} + c_2\mathbf{v_2} + c_3\mathbf{v_3}. \end{aligned} \tag{2}$$

Since the new basis defined in (1) is orthonormal, we can find the coefficients c_i by taking the inner product of $\mathbf{v}$ with each basis vector in turn. To show why this works, here is the full computation for the first coefficient, beginning with a substitution from (2) and then using the properties of bilinearity (inner products in general) and of orthonormality (these basis vectors in particular):

$$\begin{aligned} \langle \mathbf{v}, \mathbf{v_1} \rangle &= \langle c_1\mathbf{v_1} + c_2\mathbf{v_2} + c_3\mathbf{v_3}, \mathbf{v_1} \rangle \\ &= c_1\langle \mathbf{v_1}, \mathbf{v_1} \rangle + c_2\langle \mathbf{v_2}, \mathbf{v_1} \rangle + c_3\langle \mathbf{v_3}, \mathbf{v_1} \rangle \\ &= c_1(1) + c_2(0) + c_3(0) \\ &= c_1 \end{aligned} \tag{3}$$

We form a new vector $\mathbf{w}$ with these coefficients,

$$\mathbf{w} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 0 \\ -4 \\ 8 \end{pmatrix}. \tag{4}$$

The reader might wish to apply (1) and (2) to (4) to verify this result.

2) Extending and generalizing: It might not be immediately apparent why this concept is especially useful in the application of image compression. To begin to uncover the true utility of this approach, let us examine more closely the above example. Notice that the original vector $\mathbf{v}$ has three, non-zero coefficients, while the transformed vector $\mathbf{w}$ has a coefficient that is zero. In this relatively simple situation (vectors in $\mathbb{R}^3$) sending one of the coefficients to zero is no great feat. What if, instead of a three-dimensional space, we

[1]It turns out that the JPEG algorithm prefers a color space known as YCrCb. These three components are called *luminance* (Y) and *chrominance* (Cr and Cb). Luminance basically means "brightness" while chrominance has to do with the difference between a color and a chosen reference color of the same luminous intensity. The reference colors for YCrCb are red and blue as the subscripts suggest.

were working in a higher-dimensional space? It might be the case that several, even most, of the coefficients in a vector are transformed to zero. That would introduce the possibility of representing the vector in a compact way. If you think about it, we do something like this all the time: an expression like $k = \{1, 2, 3, \ldots, 99\}$ really means that k is the first ninety-nine integers. The "…,99" is a six character shorthand for ninety-six of the members of k, which would otherwise have taken one hundred eighty six digits and ninety five commas (two hundred eighty one more characters). By adjusting the representation we are able to hold a lot of information in a small space. However, this alternate representation depends heavily on special attributes of the information being represented.

3) Images as vectors: Now we take this idea and apply it to the task of image compression. Despite the fact that any given sub-pixel (e.g. the green component) in an image can take on any integer on the interval $[0, 255]$, the relation to that pixel and its neighbors is typically not that arbitrary. Pick up any photograph and look for patterns in the variation of color and brightness across the image. Depending on the particular image you may see large patches with very little, or very gradual, variation. Other areas of the image will have larger fluctuations from one point to an adjacent point, or fluctuations that are spaced more closely, but they might form some repeating pattern. There are many combinations as well, but these patterns represent a property of the image that we call spatial frequency content. Just as a passage of music with bass notes has low temporal frequency components, an image with slow changes in tone has low spatial frequency components. The next step is to quantify these frequency components.

4) Intensity to frequency: Consider a line of eight adjacent pixels. Think of it as some part of one of the rows or columns from an image, or just think of it as an independent bunch of pixels that decided to hang out together and form a line. Now we treat this row of pixels like a vector; a list of coefficients indicating how much of the appropriate color component is present at that spatial location. Each basis vector in this model would be just like the standard Euclidean basis in geometry, and have the unit value in one of the eight positions with zeros in the rest. The unit value, in this case, is equivalent to a 255th of the maximum possible intensity. A minor adjustment at this stage, as mentioned at the end of (§I-D), is to use a signed integer representation of the pixels, subtracting 128 from each intensity sample so that they now fall on the interval [-128,127]. Later on when we decompress a JPEG file, the last step in the decoding sequence will be to add 127 back to every sample.

Now we select an alternate basis that will exploit the patterns that are generally found in groups of pixels like this. Since we are interested in the spatial frequency content of the group, we choose basis vectors that are parameterized in a frequency variable: those produced by cosine functions.

The basis vectors of the DCT are given by the function

$$\begin{aligned}
\mathbf{d}_{\boldsymbol{\Omega}}[t] &= \frac{C(\Omega)}{2}\cos\left(\frac{(2t+1)\Omega\pi}{16}\right) \\
C(\Omega) &= \begin{cases} \frac{1}{\sqrt{2}} & \Omega = 0 \\ 1 & \text{else} \end{cases} \\
\text{for } t &= \{0, 1, \ldots, 7\} \\
\Omega &= \{0, 1, \ldots, 7\},
\end{aligned} \tag{5}$$

where t is the coefficient index, and Ω identifies one of the eight basis vectors by the frequency used to generate it. This definition is chosen to give the basis set the property of orthonormality[2]. Fig. 1 shows what each of these vectors would look like on an intensity versus position plot. The highest values would correspond to the brightest pixels while the lowest (largest negative) values are the darkest pixels.

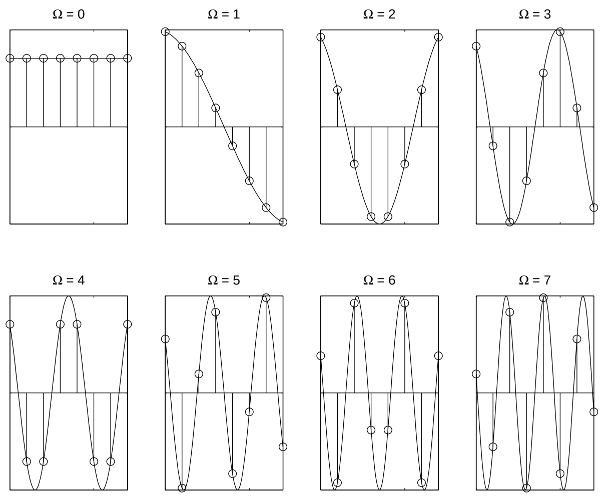

Fig. 1. A largely qualitative view of the DCT basis vectors.

If we now apply the method described above in (§II-B.1), we can express the pixel vector as a linear combination of the sinusoidal basis vectors defined in (5). The remarkable feature of the transformed vector is that many of the high frequency coefficients are often zero or close to zero. This happens because continuous tone digital images typically have little or no high frequency spatial variation. Look again at the $\Omega = 7$ basis vector in Fig. 1. The corresponding pixel values would alternate between very bright and very dark pixels across the whole row. Given that computer monitors have pixels that are about $0.3mm$ wide, one may readily imagine that an image with a drastically different color every fraction of an inch would be unpleasant to look at.

In a group of eight frequency coefficients it would not be out of the question to expect three or four of them to be zero, and certainly four or five could be very close to zero. If a coefficient is nearly zero, it means that the contribution of the corresponding frequency component is small, perhaps even sufficiently insignificant that it would not be missed if it were to be discarded.

[2]Orthonormality can be proved for this set of vectors by simply testing every pair under the dot product. There are 24 pairs, each requiring 8 multiplications and 7 additions for the dot product. A mere 360 arithmetic operations.

5) Spatial frequency in two directions: There is one more extension of the DCT that is important for the JPEG algorithm. Since images are represented as 2-D arrays of pixels, it is advantageous to incorporate a quantitative measure of both vertical and horizontal spatial frequency, not just one or the other. The discussion in (§II-B.4) dealt only with a line of pixels. While an image can certainly be separated into many such lines, and transformed accordingly, it is better to transform blocks (sub images, so to speak).

6) Definition of the DCT: The 2-D DCT used for JPEG image compression is defined as follows[3]. We use the general form of the DCT basis vectors from (5).

$$F(v,u) = \sum_{x=0}^{7}\sum_{y=0}^{7} p(y,x)\mathbf{d_u}[x]\mathbf{d_v}[y] \tag{6}$$

where $F(v,u)$ is the frequency coefficient with vertical frequency v and horizontal frequency u, and $p(y,x)$ gives the value of pixel in row y and column x of the block.

This definition can be framed in terms of matrix operations. Let

$$F = \begin{bmatrix} f_{00} & \cdots & f_{07} \\ \vdots & \ddots & \\ f_{70} & & f_{77} \end{bmatrix} \qquad f_{vu} = F(v,u), \tag{7}$$

$$P = \begin{bmatrix} p_{00} & \cdots & p_{07} \\ \vdots & \ddots & \\ p_{70} & & p_{77} \end{bmatrix} \qquad p_{yx} = p(y,x), \tag{8}$$

and

$$D = \begin{bmatrix} d_{00} & \cdots & d_{07} \\ \vdots & \ddots & \\ d_{70} & & d_{77} \end{bmatrix} \qquad d_{\Omega t} = \mathbf{d_\Omega}[t]. \tag{9}$$

That is, the matrices F and P are direct analogs to the digital storage of frequency coefficients and pixels respectively, while D has the DCT basis vectors as its rows. By using the element-wise definitions of (7), (8), and (9) to substitute in (6) we see a slightly different form of the DCT equation.

$$f_{vu} = \sum_{x=0}^{7}\sum_{y=0}^{7} p_{yx} d_{ux} d_{vy} \tag{10}$$

By rearranging factors and grouping appropriately[4] we are able to recognize (10) as the element-wise definition of matrix multiplication:

$$F = DPD^{\intercal} \tag{11}$$

[3]The term “two-dimensional (2-D)” in this context refers to the shape of the 8×8 pixel block, not the number of basis vectors of the DCT.

[4]Recall also that the subscripts of a matrix element can be reversed by taking the transpose of the matrix.

7) The inverse DCT (IDCT): Until now, this discussion has been exclusively concerned with the compression of an image. Let us not overlook the fact that a compression scheme is of no use unless the process can be reversed and the information decompressed. We now have all we need to prove that the DCT is reversible. The proof is very nice: The matrix D has full rank because its rows are a basis. Therefore D and its transpose are invertible. Since the rows of D are orthonormal, D is an orthogonal matrix, so $D^{\intercal} = D^{-1}$. The conclusion of the proof gives us an equation for the IDCT.

$$\begin{aligned} D^{-1}F(D^{\intercal})^{-1} &= D^{-1}DPD^{\intercal}(D^{\intercal})^{-1} \\ &= P \\ \Rightarrow P &= D^{\intercal}FD \end{aligned} \tag{12}$$

If we take the transpose of (12) we find that we can perform the IDCT by using the DCT to operate on $F^{\intercal}$ and then taking the transpose of the result.

As (12) shows, the DCT transform and its inverse will theoretically result in no loss of image detail. In practice, all values are stored with some finite precision, and the cosine functions are not ideal, so it is possible to introduce some small error in the image reconstruction just by transforming data back and forth from intensity to frequency. However, this loss is insignificant compared to the loss that is intentionally introduced during the next stage of the algorithm.

C. *Quantization*

After a block of pixels has been transformed to frequency coefficients, it is quantized. This is the stage of the algorithm when information is discarded. Quantization means that a larger unit step size is selected for each element in the 8×8 block and the sample at that block is forced to the nearest multiple of that step size. For example, if a frequency coefficient after the DCT was 47.98, and the corresponding quantization value was 28, the quantized frequency coefficient would be 2. Note that any frequency coefficient with a value less than half of the corresponding quantization value will be forced to zero.

The quantization step can be formally defined in this way:

$$g_{vu} = \mathrm{N}\left(\frac{f_{vu}}{q_{vu}}\right) \tag{13}$$

Where $N(x)$ chooses the nearest integer value, and q_{vu} is the 8-bit unsigned integer quantization value.

The quantization table is the primary input parameter to a JPEG compressor. There are approximately 1.3×10^154 valid quantization tables. Many are of little use because high quantization values cause too much visual information to be lost, ruining the image, and because it is inconsistent with the common properties of images to keep information about high frequency components while removing information about low frequency components. The point is that the quality scale of 1 to 10 available through the user interface of most JPEG compressors comes nowhere near the level of control that is supported by the JPEG algorithm.

More than one quantization table can be used for a single image (up to four). This is useful, for example, because our eyes are less sensitive to variations in blue chrominance than to luminance, so certain color components can be more heavily quantized with less effect on our perception of the image. Each quantization table that is used must be stored in the compressed file.

D. Intermediate coding

After the transform and the quantization, an intermediate coding strategy is applied to the blocks of frequency coefficients. Each 8×8 block is mapped into a 1-D array 64 elements long using a zig-zag pattern. This choice of mapping is designed to place coefficients with similar overall frequency (the combined horizontal and vertical frequency, approximately speaking) in adjacent locations in the linear array. Because the non-zero coefficients tend to cluster in the upper-left corner of the block, the zig-zag pattern tends to result in a linear array terminated by a long string of consecutive zeros.

The second part of the intermediate coding is a mixture of run-length encoding (RLE) and variable length integer (VLI) encoding. The RLE technique is applied only to runs of zeros, and the VLU coding is applied to all non-zero coefficients. Each element that is produced by this combined coding has this form: (run,size,value). Run indicates the number of zeros that precede this non-zero entry and is represented by a 4-bit unsigned integer. Size is also a 4-bit integer and indicates the number of bits required by the value field. Value itself is the value of the quantized frequency coefficient. There is a small difference in how the lowest frequency terms (DC) are handled, but it is not necessary to discuss it for the purpose of this paper.

E. Entropy coding

The most common entropy coding method used by the JPEG algorithm is Huffman coding. Arithmetic coding is also used, and produces slight smaller files, but is more complex to implement. In essence, entropy coding attempts to further reduce the number of bits necessary for storing each element from the intermediate coded array based on the probability that a certain element will appear. A discussion of the probabilistic interpretation of information is worthy of its own paper, but a brief example is included below which should give the reader a good idea of how the technique works. A stream of Huffman coded bits must be accompanied by a code table, or key, for decoding the stream. The JPEG file format supports using multiple tables (up to 2 for the Baseline DCT mode), which means that the compressor can select two distinct probability density functions and choose then match the distribution of a given block to one or the other to give the best compression.

In essence, the DCT puts the data in a form that can take best advantage of the compaction afforded by the intermediate and entropy coding. The latter steps are where the data compression actually happens.

F. Decompression

To decompress an encoded JPEG image each of the above steps are reversed in reverse order.

1) Use the stored Huffman table to decode strings of bits into a 64-element array;
2) reform the array into an 8×8 block;
3) multiply each element in the block by a corresponding quantization value;
4) perform the IDCT on the dequantized block;
5) and add 128 to each of the de-transformed pixels to recover an approximation of the original image.

III. Numerical work and examples

To exercise the theory that the first sections of this paper developed, we will now choose an example image, select a representative block of pixels, and perform all of the steps outlined above.

See Fig. 2 for the source image we will be working with[5], where the selected block is shown in a magnified region.

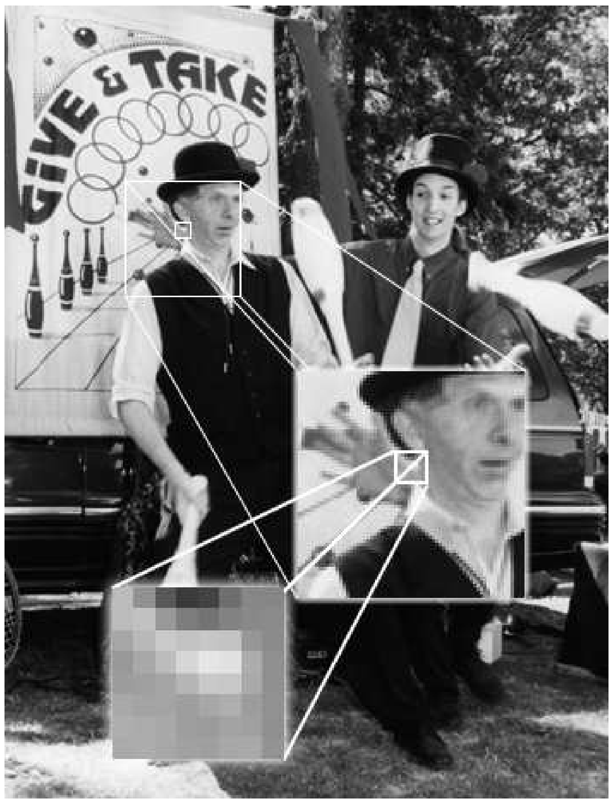

Fig. 2. A representative block of pixels.

The MatLab code used to perform most of the following computations can be found in Appendix I.

A. Intensity samples

The selected block of pixels has these values:

[5]Photograph courtesy of The Give & Take Jugglers.

$$P_0 = \begin{bmatrix} 188 & 145 & 88 & 58 & 67 & 110 & 134 & 134 \\ 187 & 193 & 152 & 125 & 115 & 130 & 137 & 139 \\ 166 & 184 & 201 & 194 & 198 & 195 & 151 & 139 \\ 152 & 168 & 188 & 214 & 229 & 225 & 172 & 143 \\ 156 & 159 & 165 & 181 & 201 & 199 & 169 & 144 \\ 168 & 163 & 164 & 158 & 167 & 174 & 156 & 145 \\ 174 & 171 & 169 & 158 & 155 & 163 & 160 & 144 \\ 170 & 172 & 167 & 161 & 159 & 167 & 169 & 147 \end{bmatrix} \quad (14)$$

The visual-numerical analog can be seen, for instance, by noting the association between the lowest value in the matrix—58, or 22.7% white, at location (0,3)—and the dark pixel in the top row of the block, and between the highest value—229, or 89.8% white, at location (3,4)—and the nearly white pixel near the center of the block.

B. Level shift

The sample range is shifted to be zero-centered by subtracting 128:

$$P_0' = P_0 - (128)I = \begin{bmatrix} 60 & 17 & -40 & -70 & -61 & -18 & 6 & 6 \\ 59 & 65 & 24 & -3 & -13 & 2 & 9 & 11 \\ 38 & 56 & 73 & 66 & 70 & 67 & 23 & 11 \\ 24 & 40 & 60 & 86 & 101 & 97 & 44 & 15 \\ 28 & 31 & 37 & 53 & 73 & 71 & 41 & 16 \\ 40 & 35 & 36 & 30 & 39 & 46 & 28 & 17 \\ 46 & 43 & 41 & 30 & 27 & 35 & 32 & 16 \\ 42 & 44 & 39 & 33 & 31 & 39 & 41 & 19 \end{bmatrix} \quad (15)$$

C. Forward DCT

Switching to a floating point representation we apply (11) to P_0' to find the frequency coefficients.

$$F_0 = \mathrm{DCT}[P_0'] = \begin{bmatrix} 263.00 & 46.69 & -10.84 & 45.62 & -28.00 & -4.91 & 5.84 & -5.57 \\ -67.22 & 24.13 & 56.18 & 6.96 & -2.32 & -8.65 & 0.87 & -5.86 \\ -119.71 & 32.57 & 129.33 & -7.86 & -8.71 & 10.05 & -8.83 & 1.19 \\ -87.58 & -7.34 & 71.10 & 16.89 & 4.75 & 4.86 & -1.32 & 2.37 \\ -11.75 & -33.87 & -3.76 & 23.69 & 4.75 & 5.40 & -3.85 & -0.44 \\ 3.79 & -10.86 & -13.13 & 8.78 & 1.96 & 4.01 & 6.98 & 0.26 \\ -1.56 & -6.20 & -7.08 & 4.27 & 1.17 & -0.42 & 8.42 & 2.77 \\ -2.73 & -0.94 & -1.93 & 1.05 & 0.31 & 0.95 & 2.29 & 3.48 \end{bmatrix} \quad (16)$$

D. Quantization

The quantization table we will use is chosen from an example in [1]. Note that the values in the upper-left corner of the matrix, which correspond to DC and low-frequency components, are fairly low (10-20), whereas the values in the lower-right region of the matrix, which are used to quantize the high frequency components, are much higher (80-120).

$$Q_0 = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \quad (17)$$

Element-wise division $F_0[v,u]/Q_0[v,u]$ and rounding gives the block of quantized coefficients.

$$G_0 = \begin{bmatrix} 16 & 4 & -1 & 3 & -1 & 0 & 0 & 0 \\ -6 & 2 & 4 & 0 & 0 & 0 & 0 & 0 \\ -9 & 3 & 8 & 0 & 0 & 0 & 0 & 0 \\ -6 & 0 & 3 & 1 & 0 & 0 & 0 & 0 \\ -1 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (18)$$

E. Zig-zag sequence

Most computers store 2-D arrays in memory in a row-wise sequence. That is, all elements from the top row are stored consecutively, followed by all elements from the next row, and so forth. For the purpose of JPEG compress, on the other hand, it is desirable to have as many of the zero coefficients next to each other as possible to maximize the compression available with the RLE and Huffman coding. Consequently, the 1-D representation chosen for this 2-D array starts in the upper left corner and sweeps back and forth diagonally. The linear representation of G_0 would then be

$$G_0 = \{16, 4, -6, -9, 2, -1, 3, 4, 3, -6, -1, 0, 8, \\ 0, -1, 0, 0, 0, 3, -2, 0, 0, 0, 0, 1, 0, \ldots, 0\} \quad (19)$$

F. Intermediate coding

Though the principal thrust of this paper is not entropy coding, it is worthwhile to glance at an example. A general sense of what is happening can be gained without extensive analysis of the theoretical background of the approach.

1) Representing values: Commonly, the elements of G_0 would be stored in the computer as fixed-length binary integers. Since they are so small, this choice would "waste" many bits since a small number can be represented with a small number of bits. The biggest number in this array can be specified with five bits (10000). To exploit this fact, a set of variable length integer (VLI) codes are specified by the JPEG standard. The ones that will be used for this example are given in Table I.

value	code	size
-9	0110	4
-6	001	3
-2	01	2
-1	0	1
1	1	1
2	10	2
3	11	2
4	100	3
8	1000	4
16	10000	5

TABLE I
VARIABLE LENGTH INTEGERS USED IN THE EXAMPLE OF (§III-F.1).

2) Intermediate and entropy coding: The block G_0 will be entropy encoded starting at the first element and stepping through it to the last element.[6] Each of the corresponding sequence of coded pieces of information will have the form (run,size,value). As per the brief discussion in (§II-D), the intermediate code of G_0 would be

$$\begin{aligned}G_0 = \{&(0,5), 10000, (0,3), 100, (0,3), 001, (0,4), 0110,\\ &(0,2), 10, (0,1), 0, (0,2), 11, (0,3), 100, (0,2), 11,\\ &(0,3), 001, (0,1), 0, (1,4), 1000, (1,1), 0, (3,2), 11,\\ &(0,2), 01, (4,1), 1, (0,0)\}\end{aligned} \quad (20)$$

Each (run,size) pair are the zero-run and value size, while the numbers that follow are the VLI values from Table I. The block is terminated with the pair (0,0) indicating that there are no more non-zero elements.

It is the (run,size) pairs that receive the Huffman coding. Table II shows all of the pairs that appear in 20, as well as how many times they appear in this block.

Huffman codes have the following properties:

1) No code is a prefix for any other code (e.g. if 010 is valid, then 010... is not)
2) Codes that are more likely to occur have shorter path lengths typically

Since we are only concerned for the moment with data from a single block, it is possible to develop a Huffman table that is beautifully suited to *this* block. The choice is shown in Table II. Substituting from that table into (20) gives this string of bits:

[6]DC coefficients are actually treated differently than AC coefficients. The values are stored as differences relative to previous blocks, and a different coding table is used. However, this detail will be neglected for the purpose of this example.

#	pair	code
4	(0,2)	00
4	(0,3)	01
2	(0,1)	100
1	(0,0)	101
1	(0,4)	11000
1	(0,5)	11001
1	(1,1)	11010
1	(1,4)	11011
1	(3,2)	11100
1	(4,1)	11101

TABLE II
A LIST OF (RUN,SIZE) PAIRS FROM (20), NUMBER OF OCCURRENCES, AND HUFFMAN CODES.

$$\begin{aligned}G_0 = \{&\\ &1100110000011000100\\ &1110000110001010000\\ &0110110000110100110\\ &0011011100011010011\\ &100110001111011101\\ &\}\end{aligned} \quad (21)$$

That is 94 bits. The original block of intensity samples would have been 4096 bits long in memory. Of course, this is not counting the storage space required to store the quantization table(s) and the Huffman coding table(s), but they are shared among many blocks so the storage expense per block is negligible. Assuming that we have already taken care of storing the Huffman and quantization tables, the compression ratio for this block is 2.3%, or about 0.18 bits per pixel.

G. Decompression

If the inverse of this process is applied to the compressed data, a block of pixels P_1 will be generated. This block is intended to be a reconstruction of the original P_0, acknowledging that there will be some difference due to the loss and finite precision of the system. The

$$P_1 = \begin{bmatrix} 187 & 146 & 88 & 53 & 63 & 100 & 127 & 134\\ 190 & 175 & 148 & 126 & 125 & 135 & 137 & 131\\ 171 & 185 & 197 & 198 & 194 & 183 & 160 & 139\\ 142 & 165 & 193 & 212 & 220 & 210 & 180 & 150\\ 147 & 155 & 167 & 184 & 202 & 203 & 177 & 146\\ 176 & 168 & 158 & 158 & 172 & 179 & 160 & 134\\ 183 & 175 & 162 & 154 & 159 & 165 & 155 & 138\\ 164 & 167 & 164 & 158 & 159 & 166 & 163 & 154 \end{bmatrix} \quad (22)$$

To see how much these samples vary from the original we will also compute a percent error block. Note that most pixels have changed by 1-5% and none more than 10%. There are even some that have been reconstructed exactly.

$$E = \begin{bmatrix} -0.5 & 0.7 & 0.0 & -8.6 & -6.0 & -9.1 & -5.2 & 0.0 \\ 1.6 & -9.3 & -2.6 & 0.8 & 8.7 & 3.8 & 0.0 & -5.8 \\ 3.0 & 0.5 & -2.0 & 2.1 & -2.0 & -6.2 & 6.0 & 0.0 \\ -6.6 & -1.8 & 2.7 & -0.9 & -3.9 & -6.7 & 4.7 & 4.9 \\ -5.8 & -2.5 & 1.2 & 1.7 & 0.5 & 2.0 & 4.7 & 1.4 \\ 4.8 & 3.1 & -3.7 & 0.0 & 3.0 & 2.9 & 2.6 & -7.6 \\ 5.2 & 2.3 & -4.1 & -2.5 & 2.6 & 1.2 & -3.1 & -4.2 \\ -3.5 & -2.9 & -1.8 & -1.9 & 0.0 & -0.6 & -3.6 & 4.8 \end{bmatrix} \qquad (23)$$

IV. Conclusion

This examination of the JPEG compression algorithm, in particular the DCT operation, has demonstrated both the qualitative concepts behind the technique as well as the quantitative processes that are used to apply them.

The important results that can be drawn from this work include an understanding of the scope of the technique. Performing a DCT on blocks in an image gives easily compressible data because of the content of most images, not because of an inherent constraint of the DCT. The other important result is the framing of the DCT in terms of matrix operations. It is the first step in developing computational methods for computing the transform quickly.

Study that could logically follow from this work would be: 1) an exploration of Huffman coding in the context of probability and information theory; 2) a review of the other modes of operation of the JPEG algorithm (Progressive, Hierarchical); and 3) applications of the DCT or similar transforms to the compression and manipulation of other kinds of data (like audio).

V. Acknowledgment

The author wishes to recognize and thank Dr. Anne Dougherty for her thoughtfulness and high-standards.

References

[1] G. K. Wallace, “The jpeg still picture compression standard,” *Communications of the ACM*, Apr. 1991.

[2] *Recommendation T.81*, International Telecommunication Union (ITU) Std., Sept. 1992, joint Photographic Expert Group (JPEG). [Online]. Available: http://www.w3.org/Graphics/JPEG/itu-t81.pdf

[3] C. L. Phillips, J. M. Parr, and E. A. Riskin, *Signals, Systems, and Transforms*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2003.

[4] P. J. Olver and C. Shakiban, *Applied Linear Algebra*. Upper Saddle River, NJ: Prentice Hall, 2006.

Appendix I
MatLab code

```
% FILE: jpeg.m
% DATE: 2005-12-01
% DESCRIPTION: Example code to demonstrate
% the JPEG compression algorithm

% Block of pixel values
P0 = [188 145  88  58  67 110 134 134;
      187 193 152 125 115 130 137 139;
      166 184 201 194 198 195 151 139;
      152 168 188 214 229 225 172 143;
      156 159 165 181 201 199 169 144;
      168 163 164 158 167 174 156 145;
      174 171 169 158 155 163 160 144;
      170 172 167 161 159 167 169 147];

% Quantization table
quant = [
16 11 10 16  24  40  51  61;
12 12 14 19  26  58  60  55;
14 13 16 24  40  57  69  56;
14 17 22 29  51  87  80  62;
18 22 37 56  68 109 103  77;
24 35 55 64  81 104 113  92;
49 64 78 87 103 121 120 101;
72 92 95 98 112 100 103  99];

% Perform level shift
   P = P0 - 128;
% Construct elementary row operation matrix
% to normalize the DC basis vector
   S = eye(8)/2;
   S(1,1) = 1/2/sqrt(2);
% DCT basis vectors
   D = zeros(8,8);
   for t = [0:7]
      for w = [0:7]
         D(w+1,t+1) = cos((2*t+1)*w*pi/16);
      end
   end
   D = S * D;
% Perform forward DCT
   F = D * P * D’;
% Perform quantization of coefficients
   Q = round(F./quant);
% Dequantize
   F1 = quant.*Q;
% Perform IDCT
   P1 = D’ * F1 * D;
% Level shift up
   P1 = P1 + 128;
% Integer round
   P1 = round(P1);
% Compute precent error
% to one decimal place
   E = round((P1 - P0)./P0*1000)/10;
```