A Gaussian Derivative Based Version of JPEG for Image Compression and Decompression

Alexander P. Morgan, Layne T. Watson, *Fellow, IEEE*, and Richard A. Young

***Abstract*—The compression and decompression of continuous-tone images is important in document management and transmission systems. This paper considers an alternative image representation scheme, based on Gaussian derivatives, to the standard discrete cosine transformation (DCT), within a JPEG framework. Depending on the computer arithmetic hardware used, the approach developed here might yield a compression/decompression technique twice as fast as the DCT and of (essentially) equal quality.**

***Index Terms*—Gaussian derivatives, image compression, JPEG.**

I. Introduction

The current standard for continuous-tone still image compression and decompression is JPEG [13]. We outline this method in Section II. Several highly polished and optimized commercial implementations are currently available. However, there is always the desire to make compression/decompression faster and/or make the resulting decompressed images of higher visual quality and/or make the compressed images take less storage space. Other approaches have been proposed, including methods based on wavelets [2] and fractals. This is an area of much active research and commercial interest, particularly when extended to moving images. We cannot begin to survey the field here, but mention some standard references such as [4], [12], [13], [14].

The point of this research is to replace the discrete cosine transformation (DCT) of the image by a representation based on Gaussian derivatives, but otherwise to follow the JPEG scheme. The goal is to obtain faster decompression times, while keeping other figures of merit unchanged or improving them. The motivation for using Gaussian derivatives is given in Section III. Following the JPEG paradigm may be disadvantageous, and the experimental results suggest that a more adaptive scheme might work better. In Section IV we compare via numerical experiments the DCT with three representations based on the Gaussian derivatives. Two are clearly inferior to the DCT, and the third representation is competitive in image quality, and potentially twice as fast as the DCT, depending on the computer arithmetic hardware used.

Manuscript received , 1995; revised , 1995. The work of Watson was supported in part by Department of Energy grant DE-FG05-88ER25068.

A. P. Morgan is with the Mathematics Department, General Motors Research, 30500 Mound Road, Box 9055, Warren, MI 48090-9055.

L. T. Watson is with the Departments of Computer Science and Mathematics, Virginia Polytechnic Institute & State University, Blacksburg, VA 24061-0106.

R. A. Young is with the Analytic Processes Department, General Motors Research, 30500 Mound Road, Box 9055, Warren, MI 48090-9055.

II. JPEG

Here we will briefly outline the JPEG scheme, omitting many details. See [13] for a full description or pages 394–405 of [4] for a nice discussion with a fully worked out example.

A continuous tone image is generally in color, which implies three separation images to represent the one in color. Essentially, each of the separation images is a monochrome image which can be separately compressed. Thus, we focus on a single monochrome image. This image is presented by a rectangular matrix of integers, each integer giving the image intensity at its corresponding pixel. The integers range from 0 (black) to 255 (white). Thus, a 160×160 pixel image is presented by a 160×160 matrix of integers, and assuming each integer is represented by 8 bits, the uncompressed image takes $160 \times 160 \times 8 = 204800$ bits to represent. Since JPEG can typically obtain a 25 to 1 compression ratio, the compressed image might take only 8192 bits, and the advantage for storage of large numbers of images is obvious.

JPEG is "lossy" in that the decompressed image is not exactly the same as the original. In a 25 to 1 compression, the differences are noticeable but acceptable in many cases. "Dithering" the image is sometimes used to make a nicer display. The compression ratios can be tuned to produce better reconstructed images, but then the compressed images require more storage space. There are also "lossless" compression schemes, but they do not come close to offering the same degree of compression.

JPEG first partitions the image matrix into 8×8 blocks. Each 8×8 block represents 64 pixels, and in the JPEG paradigm almost all of the compression of the image is achieved by representing each 8×8 block with fewer than its original 1536 bits. (A color pixel is represented by 8×3 bits.) Some compression is achieved by differencing the DC terms (average pixel intensities over the blocks) from block to block. Any benefit that might be realized by considering larger blocks, blocks of varying sizes, or overlapping blocks is forfeited in this scheme.

So we focus now on a single 8×8 block. In the JPEG scheme, this block is represented in the frequency domain by taking its DCT, which replaces the 64 image-intensity integers by 64 integers representing the transform coefficients. A "quantization matrix" (q-matrix) is now used in conjunction with "Huffman coding" to reduce the number of bits representing these coefficients to a relatively small number. The basic idea is that (in general) there will be only a few large coefficients and the rest will be small enough to set to zero. The coefficients too large to be set to zero will be given a more compact representation by "rounding" their values to "click stops", for example, going from 0 to 255 by 5s rather than allowing every intermediate integer. Then these numbers can be given a still more compact representation via Huffman coding, essentially by representing more commonly occurring values by fewer bits than less commonly occurring ones. (See the cited references for more details.)

The q-matrix, Q_0, is a fixed 8×8 matrix of integers. Denote an 8×8 image block by I. The associated 8×8 transformed image block I_T is divided term-by-term by the corresponding entries of Q_0 and the results rounded to the nearest click stop. This "quantized" matrix I_{TQ} is stored, after it is encoded using Huffman coding. To reconstruct the original block I, the entries of I_{TQ} are multiplied term-by-term by the corresponding entries of Q_0 and then the inverse discrete cosine transform is applied, giving $\tilde{I}$. If there were no rounding when I_{TQ} is created, the reconstructed $\tilde{I}$ would be the same as the original I. In practice, the two differ. The JPEG compression-decompression process functions as a low bandpass filter on the image.

A typical Q_0 might be an integer multiple of a matrix such as

$$Q = \begin{bmatrix} 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \end{bmatrix}$$

from [12], page 366. The effect of dividing by such a Q_0 would be to favor the lower frequencies (i.e., those represented in the upper left corner of the matrix). The multiplier might be any positive integer up to, say, 25. The larger the multiplier, the fewer bits saved, the bigger the compression, the more degraded the decompressed image. Some types of images can be compressed more than others and still be usable. This is a function of the intended use of the image also. Images for medical, legal, or space (NASA, LANDSAT) applications typically must be lossless. Quantization matrices actually used by JPEG tend to be more finely tuned than the above. Here is

$$Q_1 = \begin{bmatrix} 1 & 1 & 5 & 6 & 14 & 15 & 27 & 28 \\ 2 & 4 & 7 & 13 & 16 & 26 & 29 & 42 \\ 3 & 8 & 12 & 17 & 25 & 30 & 41 & 43 \\ 9 & 11 & 18 & 24 & 31 & 40 & 44 & 53 \\ 10 & 19 & 23 & 32 & 39 & 45 & 52 & 54 \\ 20 & 22 & 33 & 38 & 46 & 51 & 55 & 60 \\ 21 & 34 & 37 & 47 & 50 & 56 & 59 & 61 \\ 35 & 36 & 48 & 49 & 57 & 58 & 62 & 63 \end{bmatrix}$$

from [4], page 385, and

$$Q_2 = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

from [13], page 37, both of which we used in our experiments (see Section IV).

It is clear that the choice of a quantization matrix and multiplier will greatly influence the performance of the method. When we replace the DCT with another transform, one of our main technical challenges is to choose an appropriate corresponding quantization matrix.

The speed to compress and decompress images is an important characteristic of any compression/decompression scheme. Most of the time in JPEG is taken up by the DCT and its inverse. Numerically, this can be measured by the number of multiplications and additions required per 8×8 block, which has been reduced to about 176 multiplications and 464 additions for the DCT [8]. Other considerations, such as arranging for some of the computation to be done in scaled fixed-point arithmetic with a minimum of shifts, or taking advantage of fast computer memory (caches) and parallel processors when available, make it complicated to estimate the speed of the method in comparison with alternatives. However, fast efficient implementations exploit these and other such hardware characteristics (cf. Appendix 1).

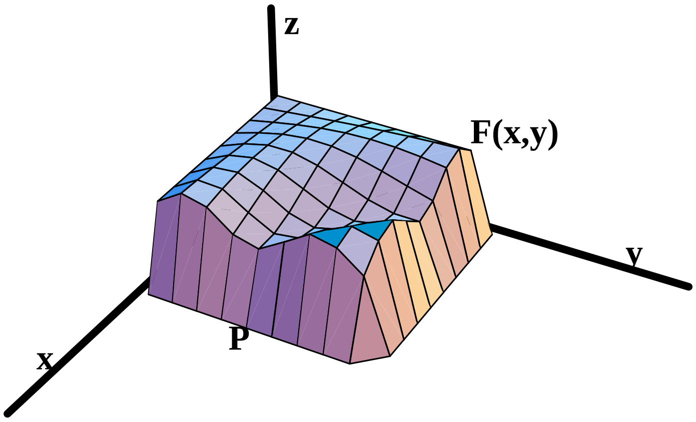


Figure 1. $F(x, y) \geq 0$ on the image P and $F(x, y) = 0$ outside of P.

III. Gaussian Derivatives

In this section we will go into more detail than the previous section, whose subject already has complete references. The main justification for using a Gaussian derivative basis is biological. Our purpose is to represent images to be displayed for human viewing. The biological data suggests that especially efficient compression ratios may be obtained by using basis functions from a specific class of functional shapes known to approximate those used by primate visual neurons (see [16], [17], [18]). The conjecture is that matching the representation of the visual data to that of primate vision systems will improve the efficiency of transfer, minimize storage requirements, and allow more rapid compression and decompression. Intuitively, the argument is that this approach should be optimal with respect to these measures of efficiency, because the reconstructed images are specifically tuned to the human vision system.

A. A Standard Framework for Image Representation

We shall consider the (monochrome) image intensity function $F(x, y)$ defined in a rectangular region P in the plane (Figure 1), and let $0 \leq F(x, y) \leq 255$. The typical tacit assumption is that F is piecewise smooth, and that the rectangular array of integers I, called the *image*, is obtained from quantized, noisy samples of F at the pixel centers. We shall focus on representing F over a small square window T inside P, where in the final implementation we shall cover P by such windows, thereby producing a collection of local representations for the image.

It will be useful to define a fixed square S about the origin, a translation of the window T, and define $f(x, y)$ over S to be the linear translate of F over T (see Figure 2). We begin by considering f over S in Subsection A.1. Then, in Subsection A.2, we extend the discussion to F over T.

A.1. Basis Functions Centered at the Origin in the Plane

We draw on the standard Hilbert space theory of orthonormal bases [15]. We state the problem in the continuous domain here and discuss the discrete version later. The problem is: Given a function $f(x, y)$ defined for points (x, y) in the plane, express f in terms of an infinite collection of "basis functions".

We begin by specifying a *weight function* $w(x, y) \geq 0$ for $(x, y) \in R^2$. The choice of a weight function will be significant in what follows, and in Subsection C our three alternative "methods"

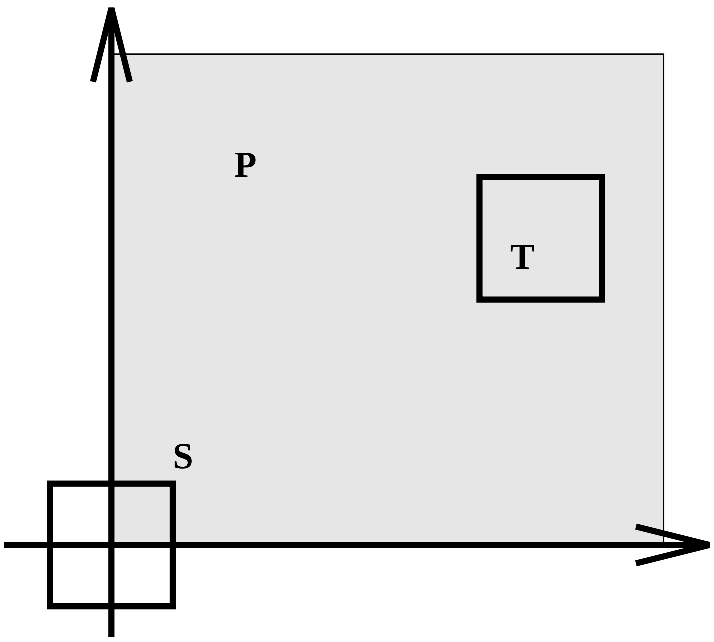


Figure 2. Square window T in P with standard template S symmetric about the origin. F over T becomes f over S via a linear transformation. The basis functions $\{\beta_{mn}\}$ are naturally centered at the origin.

will differ from each other in their three alternative choices for w. Assume that w is "integrable," by which we mean that its integral over R^2 is well defined and finite [15].

Let $f_1(x, y)$ and $f_2(x, y)$ be real measurable functions defined for all $(x, y) \in R^2$. Define the inner product of f_1 and f_2 with respect to the weight w to be the integral

$$\langle f_1, f_2 \rangle = \int_{R^2} f_1(x, y) f_2(x, y) w(x, y)\, dx dy,$$

assuming it exists and is finite. $\langle f_1, f_2 \rangle$ is finite if both $\langle f_1, f_1 \rangle$ and $\langle f_2, f_2 \rangle$ are finite, by the Schwarz inequality [15]. We say that f is *square-integrable* if $\langle f, f \rangle$ is finite, and the set of all such square-integrable f is called a Hilbert space.

Suppose we are given a set of real square-integrable functions $\beta_{mn}(x, y)$ for $m = 0, 1, 2, \ldots; n = 0, 1, 2, \ldots$ defined over R^2. (The Greek letter β is meant to suggest "basis.") By definition this collection of functions is *orthogonal* when

$$\langle \beta_{mn}, \beta_{pq} \rangle = 0$$

for $m \neq p$ or $n \neq q$, $m, n, p, q = 0, 1, 2, \ldots$ If, in addition,

$$\langle \beta_{mn}, \beta_{mn} \rangle = 1$$

for $m, n = 0, 1, 2, \ldots$, then the collection is called *orthonormal*. Note that the meaning of orthogonal and orthonormal depends on the choice of w.

Let $\{\beta_{mn}\}$ be orthogonal, and suppose now that we wish to approximate $f(x, y)$ over R^2 by a sum of the form

$$\sum_{m=0}^{M-1} \sum_{n=0}^{N-1} c_{mn} \beta_{mn}(x, y),$$

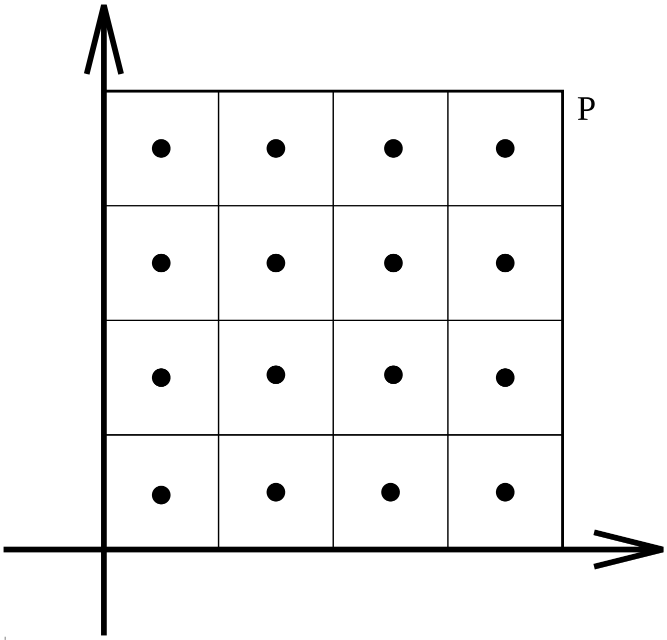


Figure 3. Q is the set of center points of the square nonoverlapping windows over each of which F will be represented by an infinite series expansion. T in Figure 2 stands for one of these squares.

where the c_{mn} are constants and N and M are given indices, in such a way that the mean-square error

$$e_{MN}^2 = \int_{R^2} \left(f(x,y) - \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} c_{mn}\beta_{mn}(x,y) \right)^2 w(x,y)dxdy$$

is minimized. From the classical theory of Fourier series, the constants c_{mn} that minimize e_{MN}^2 are given by

$$c_{mn} = \frac{\langle f, \beta_{mn} \rangle}{\langle \beta_{mn}, \beta_{mn} \rangle},$$

and are known as Fourier coefficients.

Let us note for emphasis that the Fourier coefficients c_{mn} are defined in terms of the inner product, and therefore depend on the weight w.

A set of functions β_{mn} for $m, n = 0, 1, 2, \ldots$ is called *complete* if for every square integrable function f we have

$$\lim_{M,N\to\infty} \min e_{MN}^2 = 0.$$

Roughly speaking, $\{\beta_{mn}\}$ is a complete set if f can be approximated arbitrarily closely by a finite linear combination of the β_{mn}. A complete orthonormal set of functions is called an *orthonormal basis.* It is our intention to work only with complete orthonormal bases, although we have not provided a formal proof of completeness for the orthonormal bases we present in Subsection C.

A.2. Representing Image Intensity Functions

We visualize an image domain P as a closed rectangular subset of the plane, partitioned into square windows further partitioned into square pixels, with an *intensity function* $F(x, y)$ giving grayscale values for $(x, y) \in P$ (i.e., $F(x, y)$ is a measurable function taking values from 0 to 255 for $(x, y) \in P$ and $F(x, y) = 0$ for $(x, y) \notin P$). See Figures 1, 2, and 3. Let Q be the set of window centers in the image domain P. We shall represent F in a square window T centered at a given point in Q by expanding $\chi_T(x, y)F(x, y)$ in terms of an orthonormal basis $\{\beta_{mn}\}$ as discussed above, where

$$\chi_T(x, y) = \begin{cases} 1, & (x, y) \in T, \\ 0, & (x, y) \notin T, \end{cases}$$

is the characteristic function of the square T. Thus, for each point in Q, we shall have (conceptually) an infinite sequence of coefficients $\{c_{mn}\}$ that represents F in the square window centered at that point. The dimensions of the square windows and the number of basis functions are fixed, but the coefficients c_{mn} may be different for each point in Q. Note that the basis functions are naturally centered at the origin of the plane, so that what we actually do is translate F over a window T to the origin and represent this translated F over a standard square window S centered at the origin. To recover F over the original window T in P we can merely translate it back. We use this collection of local representations of the image to reconstruct the image for viewing, piecing together the patchwork of image windows.

Precisely, let $T \subset P$ be the window centered at $(x_0, y_0) \in Q$, so that the linear transformation $L(x, y) \equiv (x - x_0, y - y_0)$ maps T one-to-one onto S with $L(x_0, y_0) = (0, 0)$. (See Figure 2.) Define $f(x, y) \equiv F(x + x_0, y + y_0)\chi_S(x, y)$, the translate of $F\chi_T$ over T to S. Then, using the orthonormal basis $\{\beta_{mn}(x, y)\}$, we have

$$f(x, y) = \sum_{m=0}^{\infty} \sum_{n=0}^{\infty} c_{mn}\beta_{mn}(x, y),$$

where

$$\begin{aligned} c_{mn} &= \int_{R^2} f(x, y)\beta_{mn}(x, y)w(x, y)\, dxdy \\ &= \int_S F(x + x_0, y + y_0)\beta_{mn}(x, y)w(x, y)\, dxdy \\ &= \int_T F(x, y)\beta_{mn}(x - x_0, y - y_0)w(x - x_0, y - y_0)\, dxdy, \end{aligned}$$

and finite truncations of this series give the best mean-square fit by the β_{mn} to F restricted to T translated to the origin.

To apply the above to image representation problems (in particular, compression/decompression) we must:

(1) Find good choices of w and β. Here we are compelled to justify "better" choices by both theoretical arguments and experimental comparisons. If a physical interpretation or practical justification of w is not available, then $w \equiv 1$ is the logical choice. Note that given w and orthonormal β_{mn}, we may define $\beta'_{mn}(x, y) = \sqrt{w(x, y)}\beta_{mn}(x, y)$ where therefore β'_{mn} is orthonormal with respect to the unity weight.

(2) Convert the continuous model to a discrete model. In practice, we shall not have F but rather a discrete noisy sample I of F. Also, we anticipate a digital implementation using integer arithmetic.

(3) Determine a strategy (perhaps an adaptive algorithm) for truncating the above series so that only a small number of the c_{mn} will be needed. Reducing the number of coefficients c_{mn} needed (with as little degradation of the reconstructed image as possible) is the essence of effective image compression.

B. Some Results from Primate Vision Experiments

Define the basic Gaussian

$$G_0(x) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-x^2/2\sigma^2}, \tag{1}$$

where σ is a positive constant, and the derivatives of the Gaussian

$$G_n(x) = \frac{d^n}{dx^n} G_0(x), \qquad \text{for } n = 1, 2, \ldots \tag{2}$$

The collection of functions (1) with (2) is sometimes referred to in the literature via the initials GD; for example, a GD-model for image representation would be a model using (1) and (2). The σ is used to tune the spatial scaling of the representation.

Various experiments have been carried out with primates that identify GD functions at several levels of the vision system. At the most basic level, the GD functions themselves have been identified by measuring the system's response to point sources. Several studies have been carried out that confirm these functional shapes up to $n = 4$. In the next subsection we note that the GD functions are orthogonal with the reciprocal Gaussian $1/G_0(x)$ as the weight function.

Further studies confirm that, if $F(x)$ represents the intensity function of an image, then the convolution

$$\int_{-\infty}^{\infty} F(x-y) G_n(y) dy \tag{3}$$

can be measured from cells at points in the visual cortex of the brain of a primate viewing the image represented by $F(x)$. Experiments have been conducted establishing one-, two-, and three-dimensional versions of this result, i.e., viewing long-bar images where a single variable x is sufficient, ordinary planar (x, y) images, and moving images where the variables are thus (x, y) and t. See [16] for a detailed summary. These biological observations suggest that orthonormal bases of functions closely related to (1), (2), and (3) will be naturally efficient for the representation of images. This is based on two lines of reasoning:

- that biologically evolved vision systems are near optimal (through the rigors of natural selection), and
- that matching the computer representation of images to our own vision systems will be most efficient for human viewing (as opposed to more synthetic, analytically derived forms of image analysis and representation).

C. Three Alternative GD-based Orthonormal Bases

Our choices of alternative weight and basis functions have the useful property of being separable in x and y, if the σs in x and y are equal. Thus, we can define them in one variable and extend the definitions to two variables by taking products: $\alpha(x)$ generates $\beta(x,y) \equiv \alpha(x)\alpha(y)$ and $v(x)$ generates $w(x,y) = v(x)v(y)$. We will make this explicit below, defining the three alternative (α, v) pairs from which the three (β, w) pairs can be defined.

Hermite polynomials will be useful, so we begin with them. The Hermite polynomial $H_n(x)$ of degree n is defined by

$$H_n(x) = (-1)^n G_n(x)/G_0(x). \tag{4}$$

The Hermite polynomials are orthogonal with respect to the weight G_0, satisfying [6]

$$\int_{-\infty}^{\infty} H_n(y)H_m(y)G_0(y)dy = \begin{cases} 0 & \text{if } m \neq n, \\ \dfrac{n!}{\sigma^{2n}} & \text{if } m = n. \end{cases} \tag{5}$$

Let us define $H_n^*(x) = \frac{\sigma^n}{\sqrt{n!}} H_n(x)$, which is thus the normalized version of H_n with respect to the inner product defined by the weight $G_0(x)$. We also normalize $G_n(x)$ via the definition $G_n^*(x) = \frac{\sigma^n}{\sqrt{n!}} G_n(x)$. It is now routine to show that, with the following choices for $w(x,y) = v(x)v(y)$ and $\beta_{mn}(x,y) = \alpha_m(x)\alpha_n(y)$, the $\beta_{mn}(x,y)$ are orthonormal bases with respect to the inner products defined by the corresponding $w(x,y)$:

No.	$v(x)$	$\alpha_n(x)$
1	$G_0(x)$	$H_n^*(x)$
2	$1/G_0(x)$	$G_n^*(x)$
3	1	$\frac{G_n^*(x)}{\sqrt{G_0(x)}}$

In the language of signal processing, the *decoder* is $\alpha_n(x)$ and the *encoder* is $v(x)\alpha_n(x)$. Note that

$$G_n^* = (-1)^n G_0(x) H_n^*(x) \quad \text{and} \quad \frac{G_n^*(x)}{\sqrt{G_0(x)}} = (-1)^n \sqrt{G_0(x)} H_n^*(x).$$

Each of these three choices of weights and bases generate coefficients c_{mn} by a formula similar to the GD equation (3). In the light of Section B, this provides the connection to the biological evidence which we want to exploit. Any of these three choices might be justified as a model. Choice 1 is used in [5] and [6]. Choice 2 is suggested by the point-source experimental data noted in Section B. Choice 3, which also comes up in quantum physics, has the advantage of being more balanced than the other two, the unity weight being attractive in that it obviates the need for a physical interpretation of w. In our experiments, Choice 3 worked best.

D. The Krawtchouk Polynomials

We can find a collection of *discretely orthogonal* polynomials that converge to the Hermite polynomials as the number of discrete points goes to infinity. We use the (discrete) binomial coefficient $\binom{M}{x}$ to approximate the Gaussian. Then the three bases described in Subsection C can be approximated by substituting these discretely orthogonal polynomials for the continuously orthogonal Hermite polynomials.

The main reason for taking this approach is computational efficiency: the Krawtchouk polynomials can be constructed very efficiently, as shown below and in Appendix 1.

The reference [10] and the classic reference [1] give the basic facts about the Krawtchouk polynomials as a "discrete Hermite transform." Given integer $M > 0$, the Krawtchouk polynomials are defined by

$$K_n(x) \equiv \frac{1}{\sqrt{\binom{M}{n}}} \sum_{k=0}^{n} (-1)^{n-k} \binom{M-x}{n-k} \binom{x}{k}$$

for $n = 0$ to M, where the $\binom{a}{b}$ are binomial coefficients. Then $K_n\left(x + \frac{M}{2}\right)$ approximates

$$\frac{1}{\sqrt{2^n n!}} H_n\left(\frac{x}{\sqrt{M/2}}\right)$$

as $M \to \infty$ ((61) from [10]). The "discrete binomial" function ((58) from [10]) we use is

$$B_0(x) = \frac{1}{2^M}\binom{M}{x},$$

which is actually defined for all real x if the binomial coefficient is represented in terms of gamma functions [1]. $B_0(x + M/2)$ approximates $G_0(x)$ (with $\sigma = \sqrt{M/2}$) as $M \to \infty$. The Krawtchouk polynomials are (discretely) orthonormal with weight B_0 since

$$\sum_{x=0}^{M} K_n(x) K_m(x) B_0(x) = \begin{cases} 0, & \text{if } n \neq m, \\ 1, & \text{if } n = m. \end{cases}$$

The matrices $K^{(p)}$ given in Table 1 are defined by $K^{(p)}(i,j) = K_i(j)$ for $p = M + 1$. In signal processing terminology these $K^{(p)}$ are "masks." Appendix 1 details how to construct them efficiently.

The discrete versions of the three types of bases given in Subsection C are then

No.	$v(x)$	$\alpha_n(x)$
1	$B_0(x)$	$K_n(x)$
2	$1/B_0(x)$	$B_0(x)K_n(x)$
3	1	$\sqrt{B_0(x)}K_n(x)$

Adhering to the JPEG model, we plan to use 8×8 image blocks, so this sets $M = 7$. In terms of the convergence results cited above, this is a relatively small value of M. In other words, the $K_n(x)$ do not necessarily approximate the $H_n(x)$ very closely for this value of M, and in fact there is significant deviation for the larger values of n. Although the discrete orthogonality of the K_n makes them a reasonable discrete basis, the biological justifications in Subsection B are based on the Gaussian derivatives specifically, and the substitution of the Krawtchouk polynomials may not be as appropriate.

IV. Numerical Experiments

We ran numerical experiments to test the DCT against the three Krawtchouk variants, which we call KT1, KT2, and KT3, corresponding to numbers 1, 2, and 3 in the above table, respectively. These tests were designed to determine the relative quality of the decompressed images at comparable levels of compression, without regard to issues of speed of compression and decompression. We used a number of monochrome images for testing, and we shall report here on a representative sample of the much larger collection of tests we actually did. Since the test criterion was "quality" of the decompressed images, there is a subjective aspect to these tests. However, our purpose here is only to determine feasibility of the new approaches compared to the DCT rather than adjudicate refined comparisons. Thus, we believe that our conclusions are sufficiently clear so as to not require sophisticated measures of quality consensus (say, by surveying panels of viewers).

A. Methods

Our test environment was the Matlab package running on a Macintosh II computer. Matlab has a sufficiently complete image processing toolbox for this work, and the Macintosh is a very convenient interactive computing environment.

As a preliminary step, we subtracted the average of each 8×8 block from each element in the block and applied all transforms to the result. Naturally, this average must be saved along with the (quantized, coded) coefficients for later decompression. This step is necessary since preliminary work showed that a constant grey-level enters in a nonuniform way into the even coefficients of the approximating Krawtchouk series, which can produce distorted picture output under compression conditions.

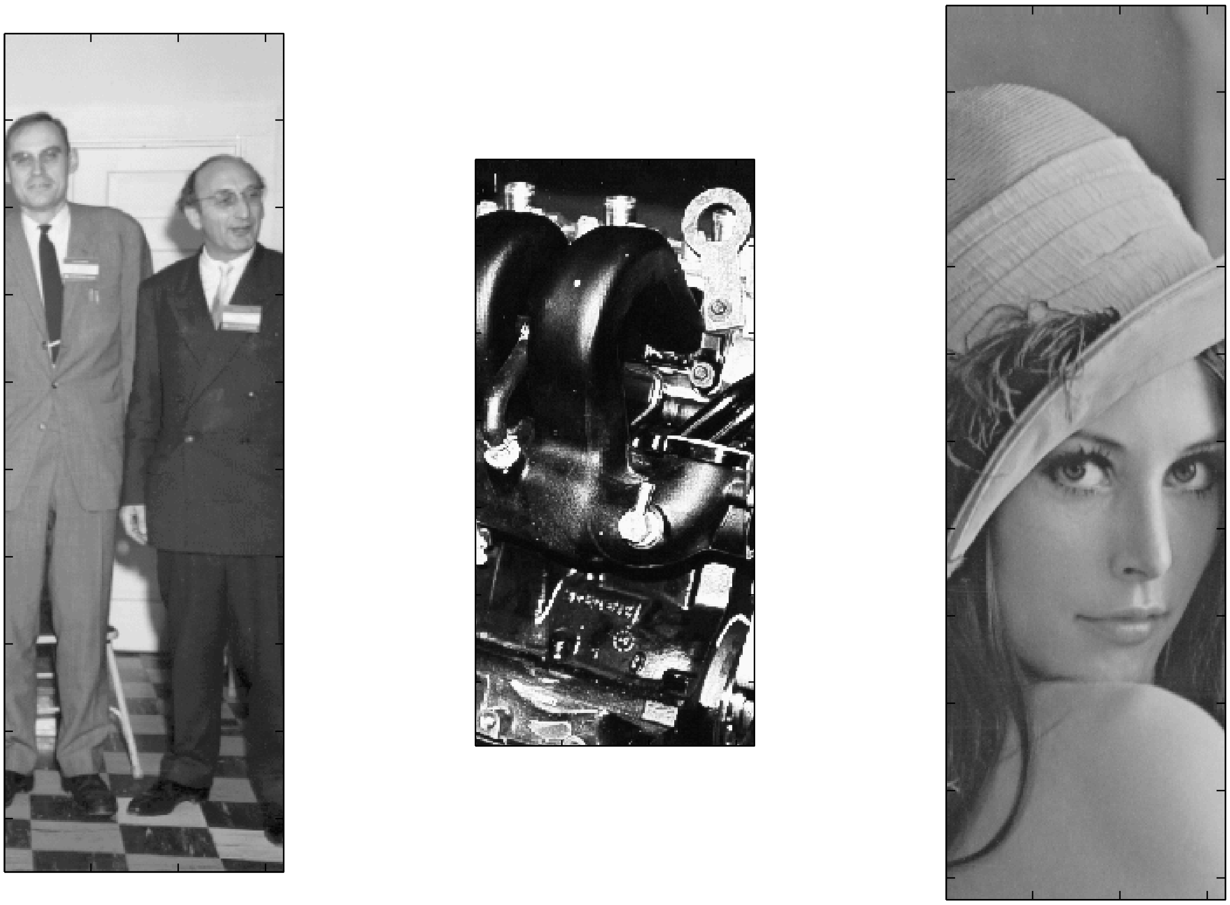
We addressed the issue of choice of q-matrices in two ways:

- we chose the same q-matrix and used it across all methods (with the scale factors described in the next paragraph), and
- we customized a q-matrix for each method from the magnitudes of coefficients on a set of test images.

By looking at the magnitudes of coefficients on test images, we determined that the order of magnitude of coefficients of the DCT and KT3 were comparable, whereas those of KT1 were about 1/10 and of KT2 100 times that of the DCT, respectively. Therefore, we modified the JPEG procedure of dividing the q-matrix into the coefficient matrix, by first multiplying the coefficient matrix by 10 or 1/100 for the KT1 or KT2, respectively. When the compressed coefficients are decompressed, this scaling is undone by multiplying by 1/10 or 100 after multiplying by the q-matrix, for KT1 or KT2, respectively. Sample coefficient magnitudes for the four methods are given in Appendix 2.

The procedure we used for customizing q-matrices was to construct averages of coefficients over all 8×8 blocks in a test image, then invert the terms of these matrices and normalize by the largest element absolute value of the original matrix. (If there is a zero entry, it is changed to 1.) Finally, these matrix elements are rounded to the nearest integer. This replaces the largest element of the original average matrix by 1 (or -1) and smaller elements by larger integers. We did this for three images (Figure 5) and averaged the resulting q-matrices to get our final q-matrices for the DCT and each KT type. (See Appendix 2). The logic of this construction is that the larger the average coefficient, the more significant it is, and the closer to 1 we want the corresponding value of the q-matrix. We also did experiments using the standard deviation in place of the magnitude, the logic here being that we want less granularity in the coefficients that are varying the most. These q-matrices did not differ significantly from those based on magnitudes.

For noncustomized q-matrices, we tried two standard matrices tuned to JPEG and the DCT, cited in Section II as Q_1 and Q_2.



1 2 3

Figure 5. Test Images 1, 2, and 3 for creating q-matrices. (See Appendix 2.)

B. Results

Q_2 seemed to work better than Q_1 in most of our tests, so the results cited below use either Q_2 or customized q-matrices. Preliminary tests established that KT2 did not work well at all, and thus only KT1 and KT3 were tested further. Figure 6 shows comparisons between DCT, KT1, and KT3, using either Q_2 or customized q-matrices as developed in Appendix 2, on Test Image 4 (Figure 6g shows the original image). Here we use the notation from Appendix 2:

$$\begin{aligned} Q_{\mathrm{dct}} &= \mathrm{AVG(Q(DCT(Images1,2,3)))}, \\ Q_{\mathrm{kt1}} &= \mathrm{AVG(Q(KT1(Images1,2,3)))}, \\ Q_{\mathrm{kt3}} &= \mathrm{AVG(Q(KT3(Images1,2,3)))}, \end{aligned}$$

where the right-hand-sides are defined in Appendix 2. The DCT tended to give a softer image (as in low bandpass filtering) than KT1 and KT3, which gave more detail but a more "grainy" image. Compare, for example, Figure 6d to 6e and 6f. Further, KT1 suffered from edge effects that seem to follow from the significant damping of the weight function used to construct the coefficients, which in turn generated an enhanced sensitivity to the q-matrix-induced perturbations of the reconstructed image. For example, compare Figure 6b to 6a and 6c. Reasonably comparable results by the DCT and KT3 could be obtained at the compression ratio of 15, e.g., Figures 6a and 6c. However, other tests favored the smoothing effect of the DCT (Figures 6d and 6f).

Figure 7 shows comparisons between DCT, KT1, and KT3, using either Q_2 or customized q-matrices, on Test Image 5. Figure 7 follows a pattern similar to Figure 6. The KT1 images (Figures 7b and 7e) had more block-edge effects, while the DCT and KT3 images were about the same. Compare Figure 7a to 7c and 7d to 7f. However, the DCT images 7a, 7d seemed just a little "smoother," compared to the speckle in the KT images 7c, 7f, whereas some detail (e.g., the windshield wiper in Figures 7d and 7f) was sharper in the KT images than in the DCT images.

We conjecture that the source of the speckle in the KT3 images is the Gaussian derivative basis' tendency to accentuate edge effects (a desirable quality for biological vision systems), and the Hermite or Krawtchouk bases being ill suited to a block transform scheme. Overlapping windows and varying the scale factor σ might produce better approximations to $f(x, y)$, but then the compression and decompression become much more complicated. To test whether the KT3 images could indeed be improved, we applied the MATLAB Wiener filter to Figures 6c, 6f, 7c, 7f, giving Figures 8a–d respectively. The filtered KT3 images are nearly indistinguishable from the DCT images. Whether a filtered Krawtchouk polynomial based block transform, still cheaper than the DCT, can be developed is a question for future research.

C. Conclusions

This paper documents a theoretical and experimental investigation of using image transforms based on Gaussian derivatives rather than the discrete cosine transform for image compression and decompression in a JPEG context. The qualitative effects of the DCT reconstruction seem less offensive over a wider ranger of images, but the KT1 and KT3 might be acceptable or even favored for certain technical contexts where slightly more detail would be preferred to a smooth image. A Wiener filtered KT3 is nearly indistinguishable from the DCT, but the cost of filtering or adaptive overlapping windows with Krawtchouk bases remains open. The results suggest that acceptable image quality at comparable compression ratios to the DCT may be obtained, and that the speed of decompression for the GD-based approach might be on the order of twice as fast as the DCT on computer platforms where integer addition is a factor of 10 faster than integer multiplication and floating point arithmetic.

REFERENCES

[1] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions*, Dover, New York, 1964.

[2] M. Barlaud (Ed.), *Wavelets in Image Communication*, Elsevier, 1994.

[3] N. Jayant, J. Johnson, and R. Safranek, "Signal Compression Based on Models of Human Perception," *Proc. IEEE*, vol. 81, pp. 1385–1422, 1993.

[4] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, Addison-Wesley Publishing Company, 1992.

[5] M. Hashimoto and J. Sklansky, "Edge Detection by Estimation of Multiple-Order Derivatives," in *Proc. IEEE Computer Vision and Pattern Recognition Conf.*, 1983, pp. 318–325.

[6] M. Hashimoto and J. Sklansky, "Multiple-Order Derivatives for Detecting Local Image Characteristics," *Computer Vision, Graphics, and Image Processing*, vol. 39, pp. 28–55, 1987.

[7] R. M. Lesperance and R. A. Young, "A Motion Analyzer for Real-Time Vision," GMR-7865, GMR Tech. Rep., Jan. 20, 1993.

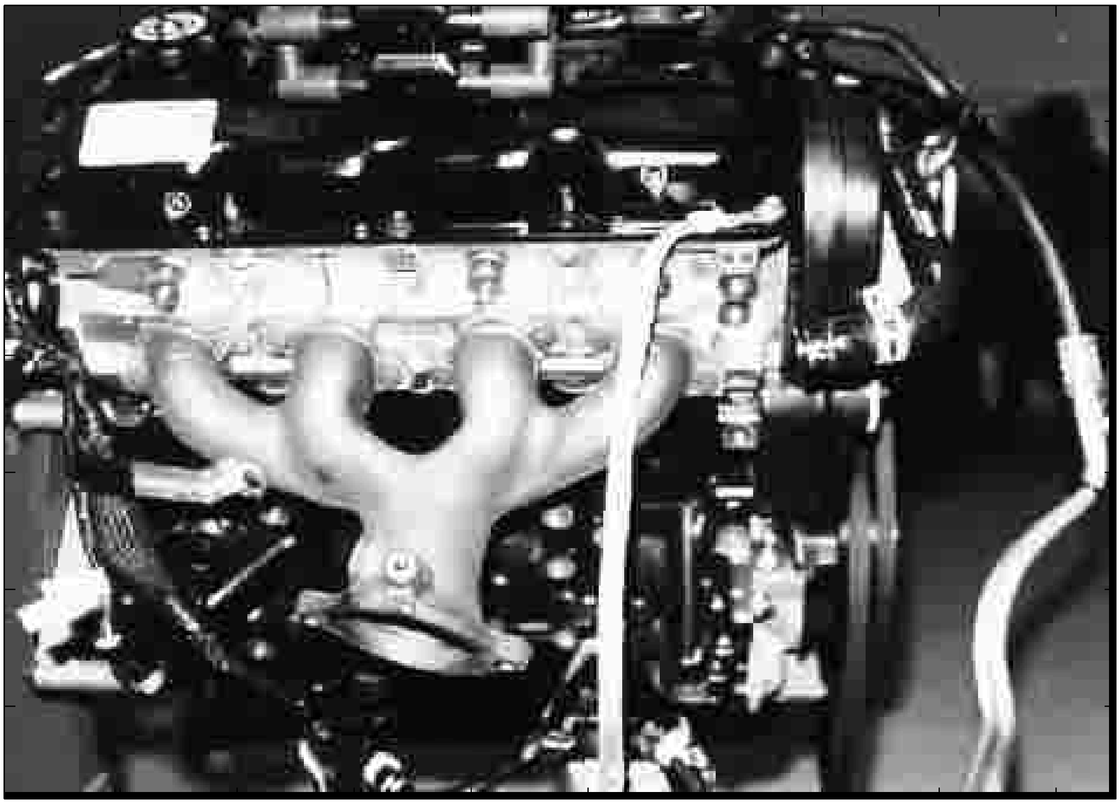


Figure 6a. Image 4, $Q = 5 * Q_2$, DCT, compression ratio = 16.

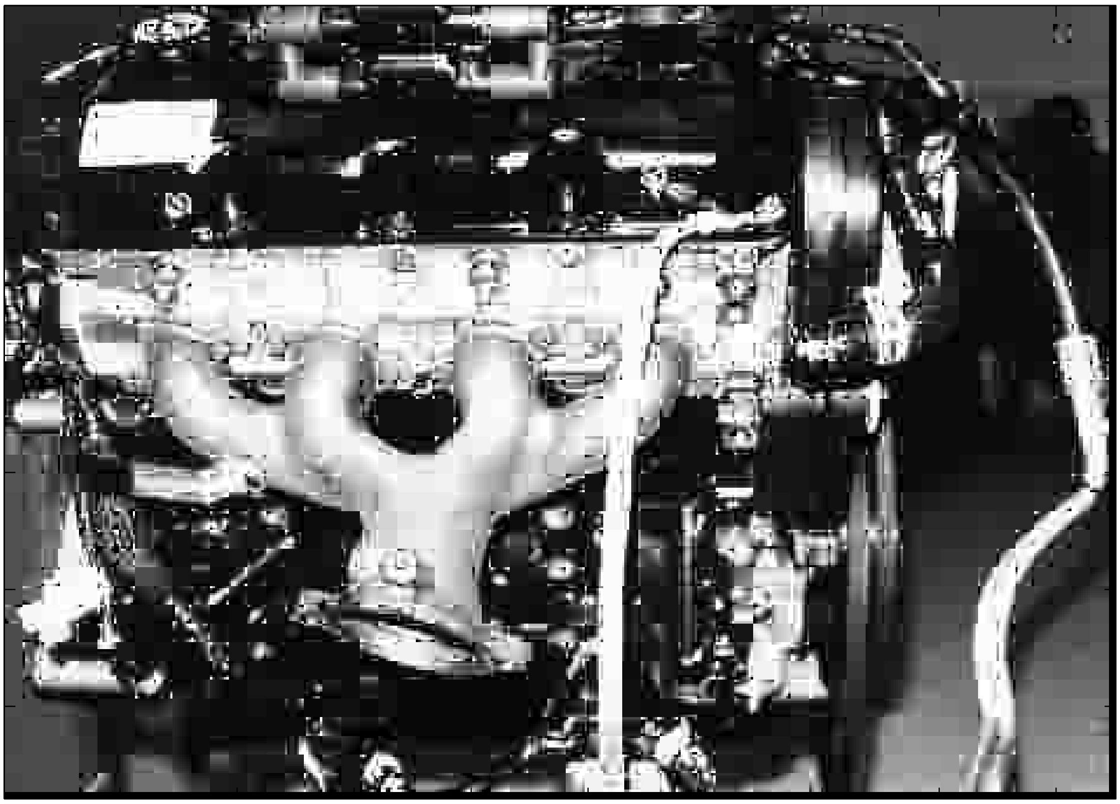


Figure 6b. Image 4, $Q = 5 * Q_2$, KT1, compression ratio = 17.

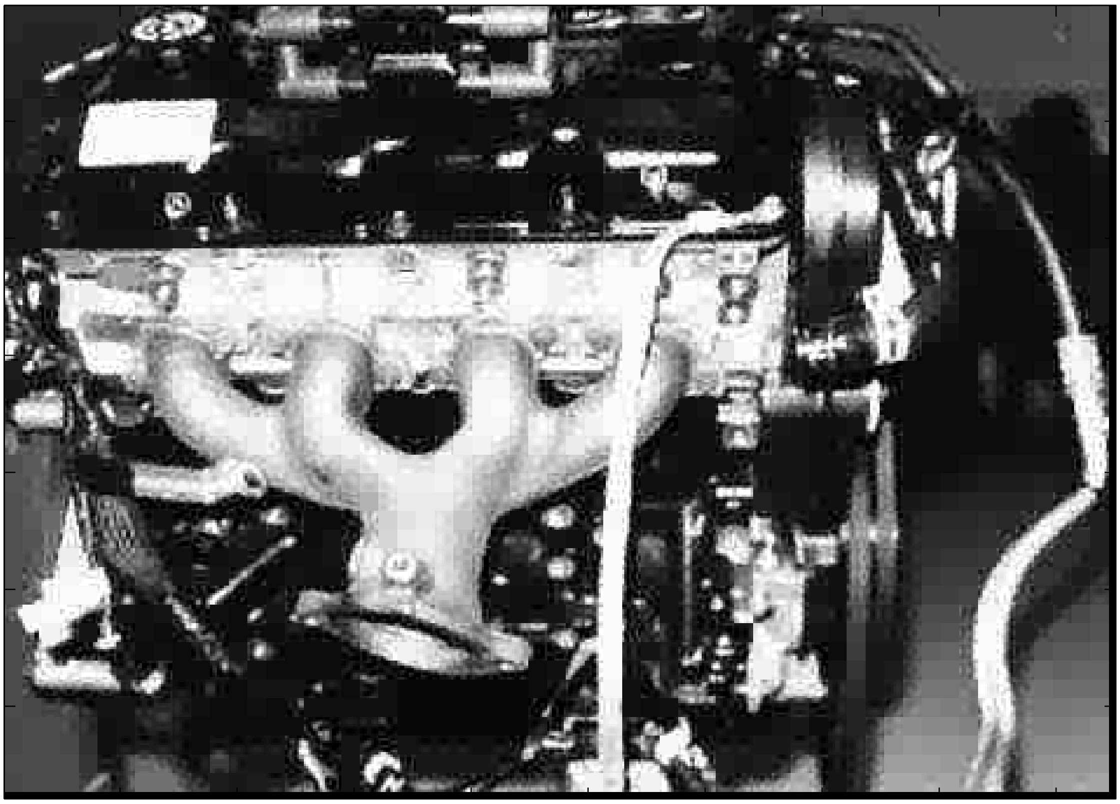


Figure 6c. Image 4, $Q = 5 * Q_2$, KT3, compression ratio = 14.

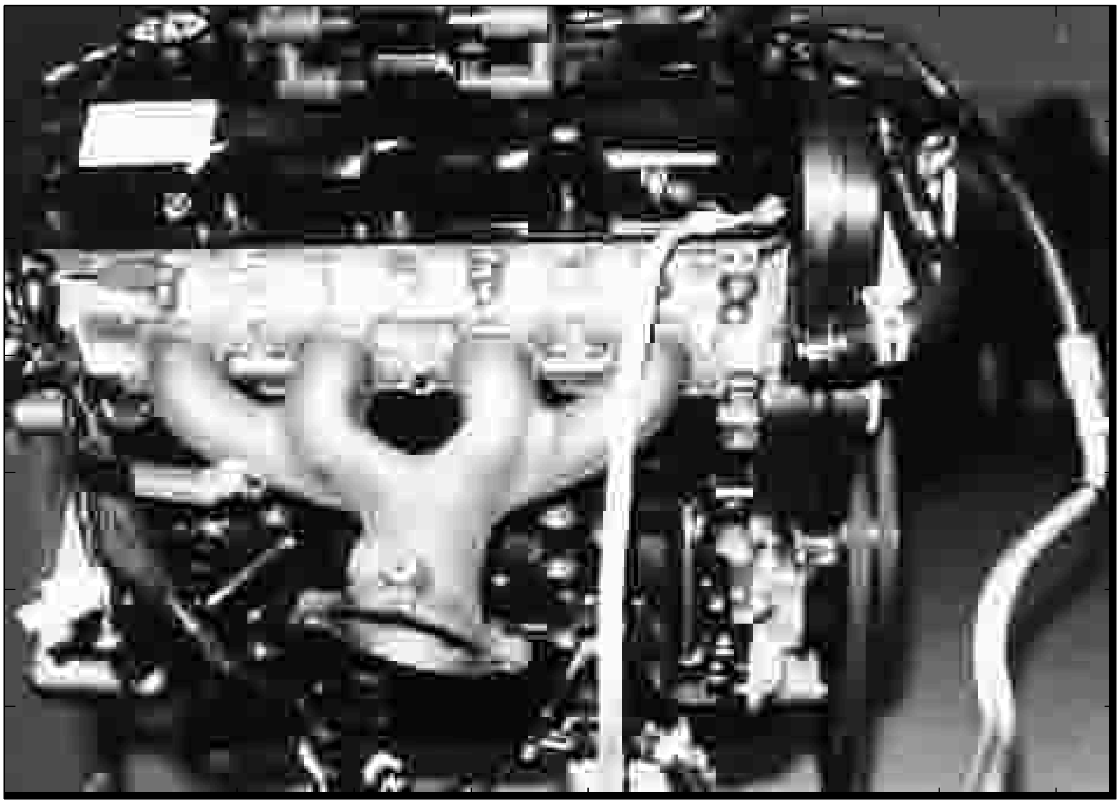


Figure 6d. Image 4, $Q = 50 * Q_{\mathrm{dct}}$, DCT, compression ratio = 21.

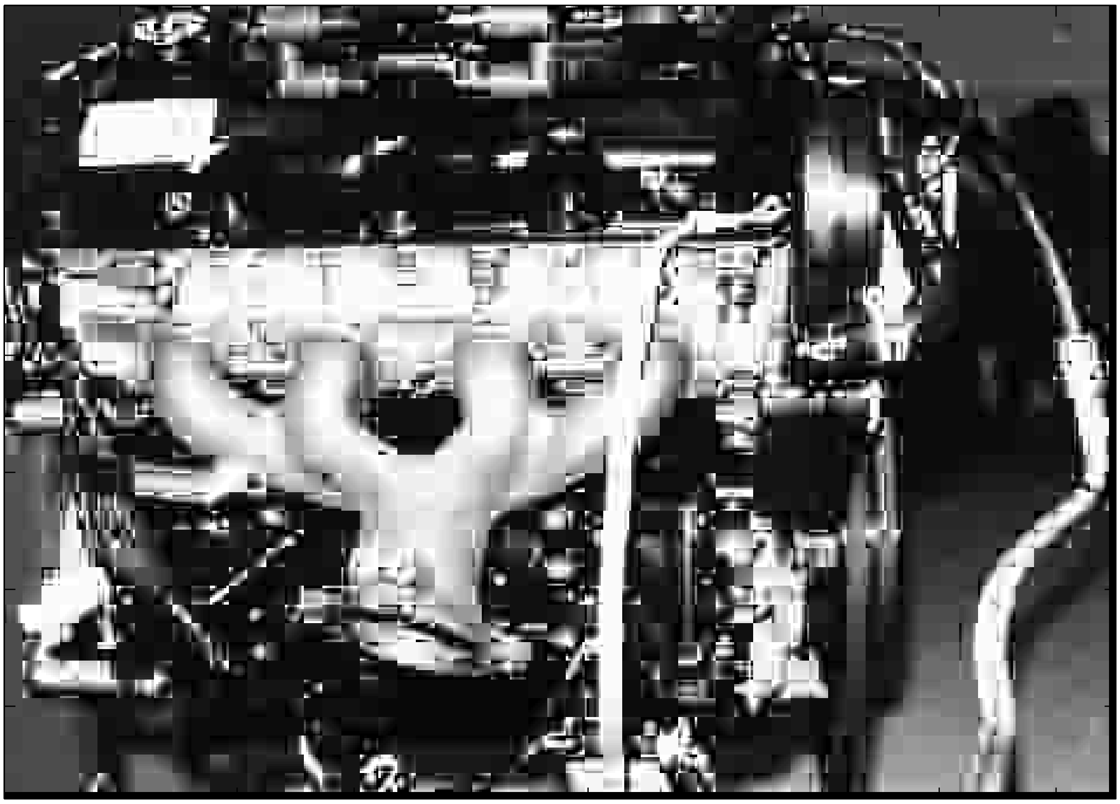


Figure 6e. Image 4, $Q = 50 * Q_{\mathrm{kt1}}$, KT1, compression ratio = 22.

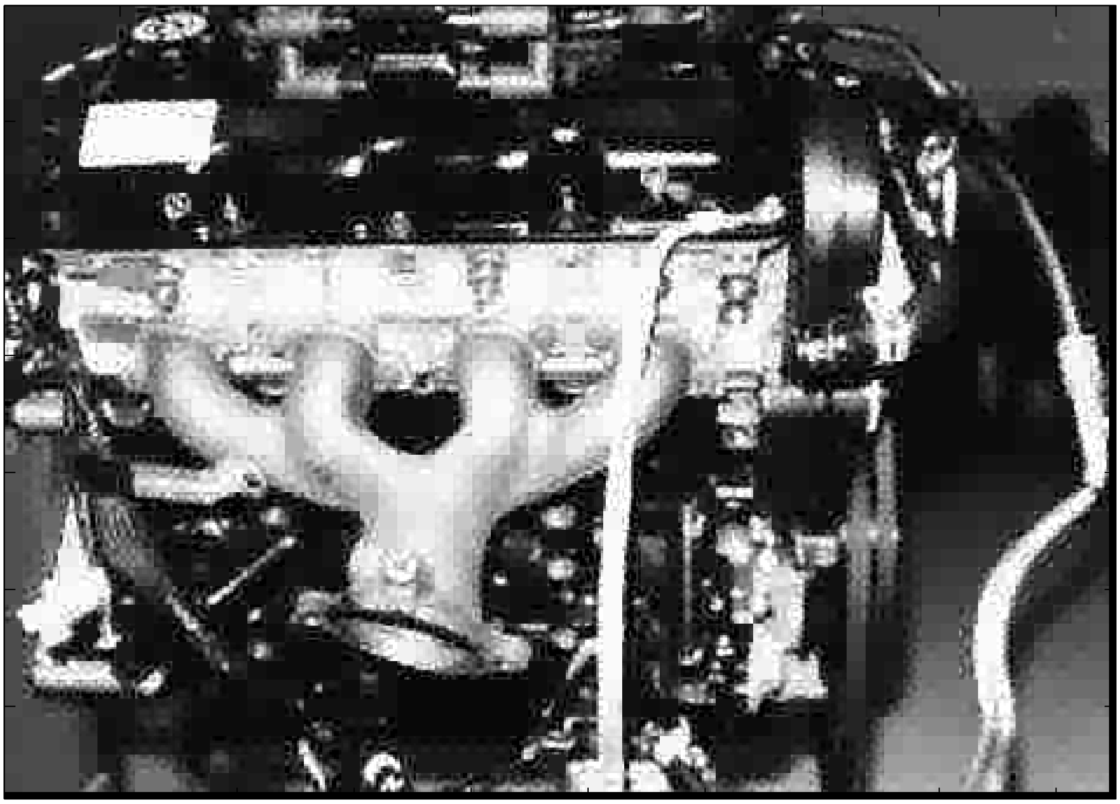


Figure 6f. Image 4, $Q = 50 * Q_{\mathrm{kt3}}$, KT3, compression ratio = 17.

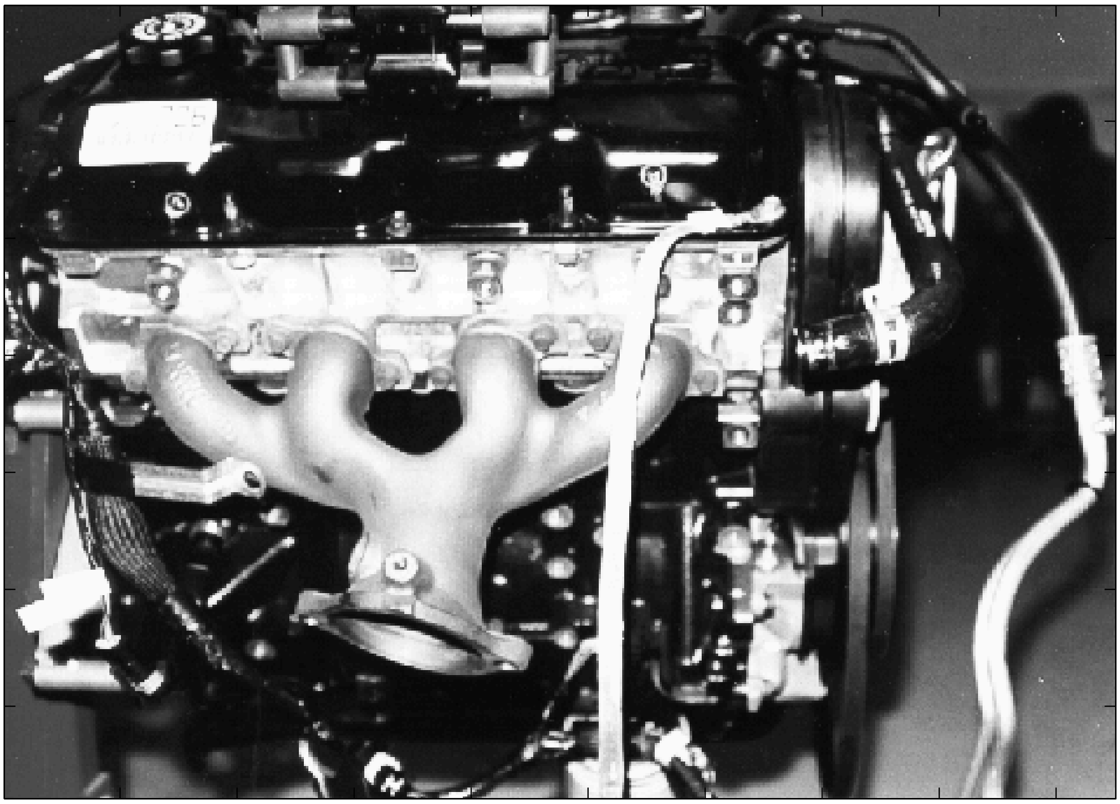


Figure 6g. Original Image 4.

Figure 7a. Image 5, $Q = 10 * Q_2$, DCT, compression ratio = 30.

Figure 7b. Image 5, $Q = 10 * Q_2$, KT1, compression ratio = 31.

Figure 7c. Image 5, $Q = 10 * Q_2$, KT3, compression ratio = 28.

Figure 7d. Image 5, $Q = 40 * Q_{\mathrm{dct}}$, DCT, compression ratio = 27.

Figure 7e. Image 5, $Q = 40 * Q_{\mathrm{kt1}}$, KT1, compression ratio = 25.

Figure 7f. Image 5, $Q = 40 * Q_{\mathrm{kt3}}$, KT3, compression ratio = 22.

Figure 7g. Original Image 5.

[8] C. Loeffler, A. Ligtenberg, and G. S. Moschytz, "Practical fast 1-D DCT algorithms with 11 multiplications," in *Proc. Int. Conf. on Acoustics, Speech, and Signal Processing*, 1989, pp. 988–991.

[9] S. G. Mallat, "Multiresolution approach in wavelets in computer vision," in *Wavelets: Time-frequency Methods in Phase Space*, 2nd ed., J. M. Combes, A. Grossman, and Ph. Tchamitchian, Eds., Proc. Int. Conf., Marseille, France, Dec. 14–18, 1987, Springer-Verlag, pp. 313–327.

[10] J. B. Martens, "The Hermite Transform—Theory," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 38, pp. 1595–1606, 1990.

[11] J. B. Martens, "The Hermite Transform—Applications," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 38, pp. 1607–1618, 1990.

[12] Mark Nelson, *The Data Compression Book*, M & T Books, 1992.

[13] William B. Pennebaker and Joan L. Mitchell, *JPEG Still Image Data Compression Standard*, Van Nostrand Reinhold, 1993.

[14] A. Rosenfeld and A. Kak, *Digital Picture Processing, Vol. 1*, Academic Press, Boston, 1982.

[15] H. L. Royden, *Real Analysis*, Macmillan, London, 1968.

[16] R. A. Young, "The Gaussian Derivative Theory of Spatial Vision: Analysis of Cortical Cell Receptive Field Line-Weighting Profiles," GMR-4920, 1985.

[17] R. A. Young, "The Gaussian Derivative Model for Machine Vision: Visual Cortex Simulation," GMR-5323, 1986.

[18] R. A. Young, "Oh Say Can You See? The Physiology of Vision," *SPIE Proc. Electronic Imaging*, San Jose, CA, vol. 1453, pp. 92–123, 1991.

[19] R. A. Young and R. M. Lesperance, "A physiological model of motion analysis for machine vision," *SPIE Proc.*, San Jose, CA, vol. 1913, pp. 48–123, 1993.

Appendix 1: Fast Krawtchouk Decompression

Compression and decompression using the DCT or KT 2-D transforms reduces to a sequence of 1-D transforms via the property of *separability*, as described in [4], page 94. It is generally assumed that such transforms will be implemented in this way, after the 1-D transform has been carefully optimized. However, we should note that optimal speed performance might be obtained using a different strategy, depending on a number of factors related to the computer environment. Also, the assumption in creating optimized software that integer arithmetic is faster than floating point is not true for some new (e.g., RISC) machines, so that optimal arrangements of the algorithmic implementation for an Intel 80386-based machine might be suboptimal for a PowerPC 601-based machine. Pipelining and memory cache opportunities are also important considerations.

Having said all that, we shall show how to generate the 1-D KT decompression using 81 integer additions and no multiplications, except that a floating-point normalization is needed for each KT variant, which is not in the inner loop of the computation. This compares well with the best reported operation counts for the DCT, namely 11 multiplications and 29 additions [8]. On an Intel 80486-based machine, an integer multiplication takes about 10 times as many clock counts as an integer add. Thus we may compare 139 addition-equivalents for the DCT with 81 for the KT. However, we emphasize that the actual opportunities for optimization will depend on the details of the computer architecture and software.

The computational strategy we shall use here is expressed typically by phrases like "the KT can be implemented by a cascade of very simple filters" (e.g., [10], page 1604). However, we shall present our argument in simple terms without reference to filter theory.

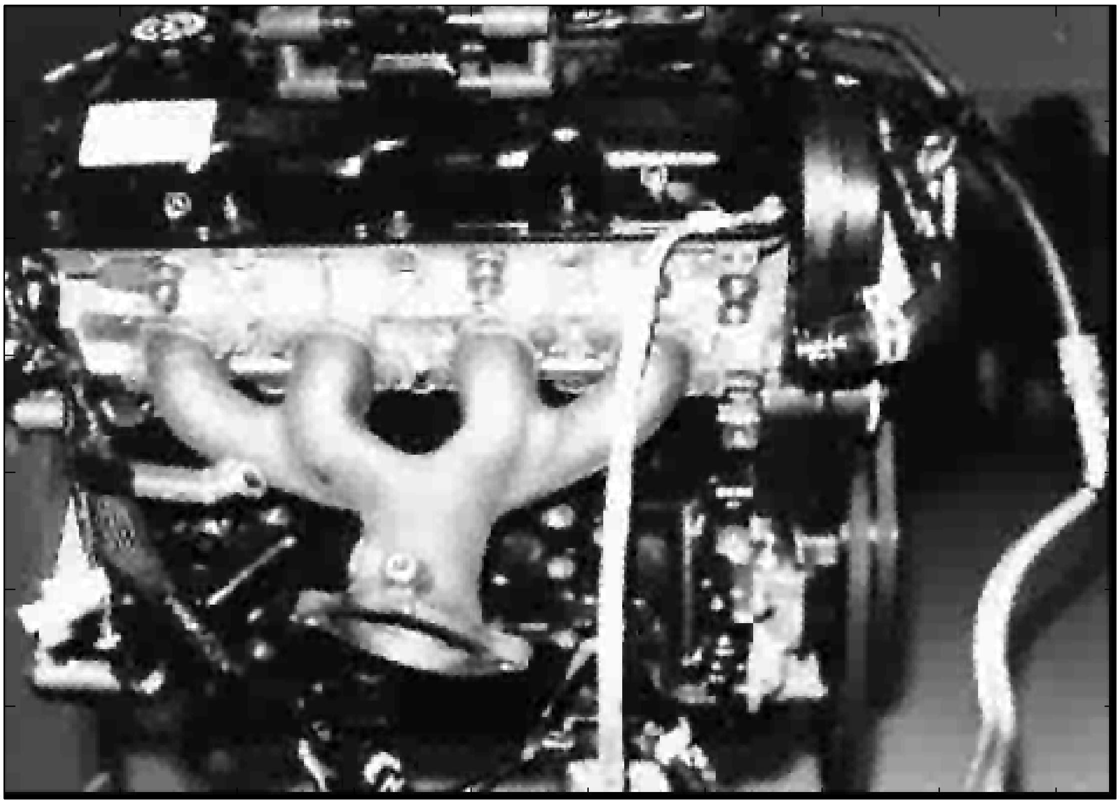


Figure 8a. Wiener filter applied to the KT3 image in Figure 6c.

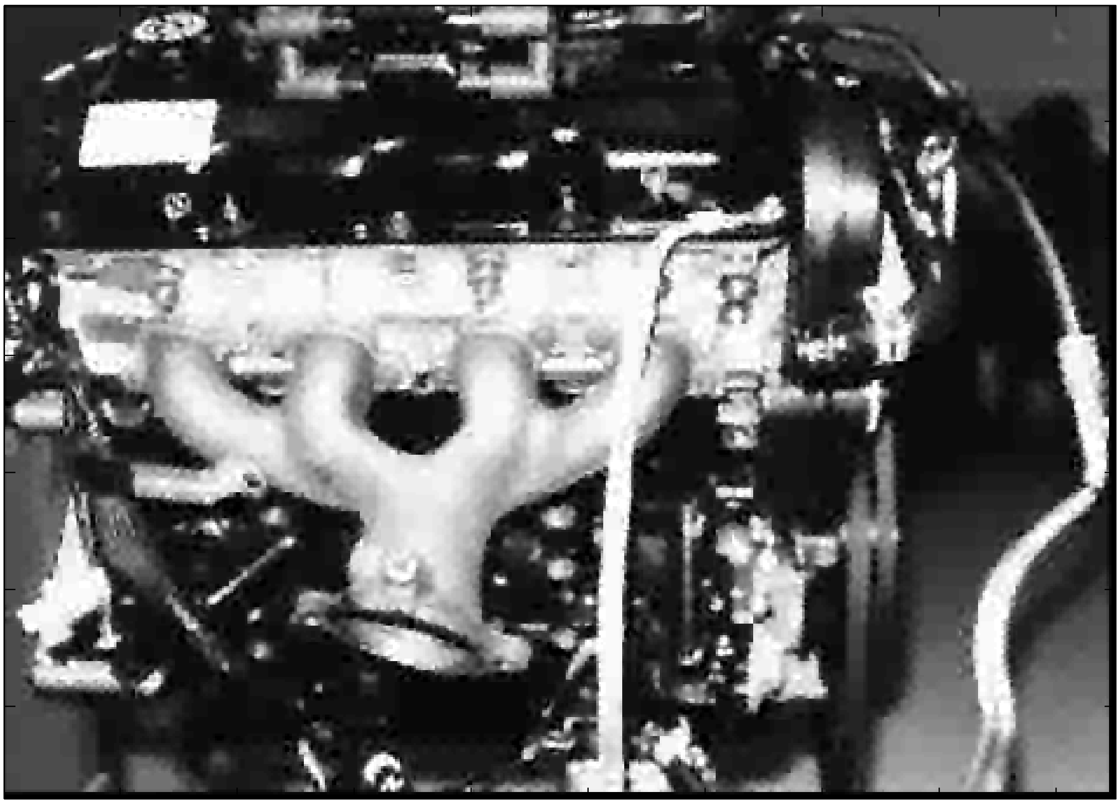


Figure 8b. Wiener filter applied to the KT3 image in Figure 6f.

Figure 8c. Wiener filter applied to the KT3 image in Figure 7c.

Figure 8d. Wiener filter applied to the KT3 image in Figure 7f.

"Decompressing" means converting coefficients into intensity values. Specifically, given a vector of coefficients $r = [r_0, r_1, \ldots, r_7]$, we want to compute the vector matrix product $r \cdot K^{(8)}$, where $K^{(i)}$ denotes the ith Krawtchouk matrix, given by formulas in Subsection IIID and listed for $i = 2$ to 8 in Table 1.

It turns out, however, that we have a recursive relationship: Let

$$\begin{gathered}
[a_0^8, a_1^8, \ldots, a_7^8] \equiv r \cdot K^{(8)}, \\
r_{p,q} \equiv [r_p, \ldots, r_q] \text{ for } 0 \leq p < q \leq 7, \\
[a_0^7, a_1^7, \ldots, a_6^7] \equiv r_{0,6} \cdot K^{(7)}, \\
[b_0^7, b_1^7, \ldots, b_6^7] \equiv r_{1,7} \cdot K^{(7)}.
\end{gathered}$$

Then

$$\begin{aligned}
a_0^8 &= a_0^7 - b_0^7, & a_4^8 &= a_4^7 - b_4^7, \\
a_1^8 &= a_0^7 + b_0^7, & a_5^8 &= a_4^7 + b_4^7, \\
a_2^8 &= a_2^7 - b_2^7, & a_6^8 &= a_6^7 - b_6^7, \\
a_3^8 &= a_2^7 + b_2^7, & a_7^8 &= a_6^7 + b_6^7.
\end{aligned}$$

We see that, having generated a_0^7, a_2^7, a_4^7, a_6^7, b_0^7, b_2^7, b_4^7, b_6^7, we can generate the a_i^8 for $i = 0$ to 7 at the cost of 8 additions. This recursion continues: the a^7, b^7 terms can be generated from the a^6, b^6, and c^6 terms, using only additions, and so on. Each set of recursive formulas can be confirmed from the matrix definitions. By counting the additions at each stage, we arrive at the final operation count. For completeness, we will list the relevant formulas and counts below.

Let

$$\begin{gathered}
[a_0^6, a_1^6, \ldots, a_5^6] \equiv r_{0,5} \cdot K^{(6)}, \\
[b_0^6, b_1^6, \ldots, b_5^6] \equiv r_{1,6} \cdot K^{(6)}, \\
[c_0^6, c_1^6, \ldots, c_5^6] \equiv r_{2,7} \cdot K^{(6)}.
\end{gathered}$$

Then

$$\begin{aligned}
a_0^7 &= a_0^6 - b_0^6, & b_0^7 &= b_0^6 - c_0^6, \\
a_2^7 &= a_2^6 - b_2^6, & b_2^7 &= b_2^6 - c_2^6, \\
a_4^7 &= a_4^6 - b_4^6, & b_4^7 &= b_4^6 - c_4^6, \\
a_6^7 &= a_5^6 + b_5^6, & b_6^7 &= b_5^6 + c_5^6,
\end{aligned}$$

whose addition count is 8.

Let

$$\begin{gathered}
[a_0^5, a_1^5, \ldots, a_4^5] \equiv r_{0,4} \cdot K^{(5)}, \\
[b_0^5, b_1^5, \ldots, b_4^5] \equiv r_{1,5} \cdot K^{(5)}, \\
[c_0^5, c_1^5, \ldots, c_4^5] \equiv r_{2,6} \cdot K^{(5)}, \\
[d_0^5, d_1^5, \ldots, d_4^5] \equiv r_{3,7} \cdot K^{(5)}.
\end{gathered}$$

Then

$$
\begin{array}{lll}
a_0^6 = a_0^5 - b_0^5, & b_0^6 = b_0^5 - c_0^5, & c_0^6 = c_0^5 - d_0^5, \\
a_2^6 = a_2^5 - b_2^5, & b_2^6 = b_2^5 - c_2^5, & c_2^6 = c_2^5 - d_2^5, \\
a_4^6 = a_4^5 - b_4^5, & b_4^6 = b_4^5 - c_4^5, & c_4^6 = c_4^5 - d_4^5, \\
a_5^6 = a_4^5 + b_4^5, & b_5^6 = b_4^5 + c_4^5, & c_5^6 = c_4^5 + d_4^5,
\end{array}
$$

whose addition count is 12.

Let

$$
\begin{aligned}
[a_0^4, a_1^4, \ldots, a_3^4] &\equiv r_{0,3} \cdot K^{(4)}, \\
[b_0^4, b_1^4, \ldots, b_3^4] &\equiv r_{1,4} \cdot K^{(4)}, \\
[c_0^4, c_1^4, \ldots, c_3^4] &\equiv r_{2,5} \cdot K^{(4)}, \\
[d_0^4, d_1^4, \ldots, d_3^4] &\equiv r_{3,6} \cdot K^{(4)}, \\
[e_0^4, e_1^4, \ldots, e_3^4] &\equiv r_{4,7} \cdot K^{(4)}.
\end{aligned}
$$

Then

$$
\begin{array}{ll}
a_0^5 = a_0^4 - b_0^4, & c_0^5 = c_0^4 - d_0^4, \\
a_2^5 = a_2^4 - b_2^4, & c_2^5 = c_2^4 - d_2^4, \\
a_4^5 = a_3^4 + b_3^4, & c_4^5 = c_3^4 + d_3^4, \\
b_0^5 = b_0^4 - c_0^4, & d_0^5 = d_0^4 - e_0^4, \\
b_2^5 = b_2^4 - c_2^4, & d_2^5 = d_2^4 - e_2^4, \\
b_4^5 = b_3^4 + c_3^4, & d_4^5 = d_3^4 + e_3^4,
\end{array}
$$

whose addition count is 12.

Let

$$
\begin{array}{ll}
[a_0^3, a_1^3, a_2^3] \equiv r_{0,2} \cdot K^{(3)}, & [d_0^3, d_1^3, d_2^3] \equiv r_{3,5} \cdot K^{(3)}, \\
[b_0^3, b_1^3, b_2^3] \equiv r_{1,3} \cdot K^{(3)}, & [e_0^3, e_1^3, e_2^3] \equiv r_{4,6} \cdot K^{(3)}, \\
[c_0^3, c_1^3, c_2^3] \equiv r_{2,4} \cdot K^{(3)}, & [f_0^3, f_1^3, f_2^3] \equiv r_{5,7} \cdot K^{(3)}.
\end{array}
$$

Then

$$
\begin{array}{lll}
a_0^4 = a_0^3 - b_0^3, & c_0^4 = c_0^3 - d_0^3, & \\
a_2^4 = a_2^3 - b_2^3, & c_2^4 = c_2^3 - d_2^3, & e_0^4 = e_0^3 - f_0^3, \\
a_3^4 = a_2^3 + b_2^3, & c_3^4 = c_2^3 + d_2^3, & e_2^4 = e_2^3 - f_2^3, \\
b_0^4 = b_0^3 - c_0^3, & d_0^4 = d_0^3 - e_0^3, & e_3^4 = e_2^3 + f_2^3, \\
b_2^4 = b_2^3 - c_2^3, & d_2^4 = d_2^3 - e_2^3, & \\
b_3^4 = b_2^3 + c_2^3, & d_3^4 = d_2^3 + e_2^3, &
\end{array}
$$

whose addition count is 15.

Let

$$
\begin{aligned}
[a_0^2, a_1^2] &\equiv r_{0,1} \cdot K^{(2)} = [r_0 - r_1, r_0 + r_1], \\
[b_0^2, b_1^2] &\equiv r_{1,2} \cdot K^{(2)} = [r_1 - r_2, r_1 + r_2], \\
[c_0^2, c_1^2] &\equiv r_{2,3} \cdot K^{(2)} = [r_2 - r_3, r_2 + r_3], \\
[d_0^2, d_1^2] &\equiv r_{3,4} \cdot K^{(2)} = [r_3 - r_4, r_3 + r_4], \\
[e_0^2, e_1^2] &\equiv r_{4,5} \cdot K^{(2)} = [r_4 - r_5, r_4 + r_5], \\
[f_0^2, f_1^2] &\equiv r_{5,6} \cdot K^{(2)} = [r_5 - r_6, r_5 + r_6], \\
[g_0^2, g_1^2] &\equiv r_{6,7} \cdot K^{(2)} = [r_6 - r_7, r_6 + r_7].
\end{aligned}
$$

Then

$$
\begin{aligned}
a_0^3 &= a_0^2 - b_0^2, & d_0^3 &= d_0^2 - e_0^2, \\
a_2^3 &= a_1^2 - b_1^2, & d_2^3 &= d_1^2 - e_1^2, \\
b_0^3 &= b_0^2 - c_0^2, & e_0^3 &= e_0^2 - f_0^2, \\
b_2^3 &= b_1^2 - c_1^2, & e_2^3 &= e_1^2 - f_1^2, \\
c_0^3 &= c_0^2 - d_0^2, & f_0^3 &= f_0^2 - g_0^2, \\
c_2^3 &= c_1^2 - d_1^2, & f_2^3 &= f_1^2 - g_1^2,
\end{aligned}
$$

whose addition count is $12 + 14$ to generate the superscript 2 terms.

The total count of additions is 81.

Appendix 2: Q-matrix Generation

This appendix gives sample coefficient matrices and their associated q-matrices. It also includes average q-matrices over the 8×8 blocks of Images 1–3. It does this for each of the DCT, KT1, KT2, and KT3 transforms. See Section III for more details. Note that KT2 is multiplied by 10 so that there would be less loss after rounding.

$$\mathrm{DCT(Image2)} = \begin{bmatrix} 0 & 105 & 57 & 33 & 24 & 19 & 16 & 15 \\ 106 & 62 & 40 & 29 & 21 & 18 & 16 & 13 \\ 55 & 38 & 28 & 24 & 18 & 15 & 13 & 11 \\ 33 & 27 & 23 & 17 & 14 & 12 & 11 & 10 \\ 23 & 22 & 17 & 13 & 12 & 11 & 9 & 9 \\ 18 & 17 & 15 & 12 & 11 & 10 & 9 & 8 \\ 17 & 14 & 13 & 10 & 10 & 9 & 8 & 7 \\ 13 & 11 & 11 & 10 & 9 & 7 & 7 & 6 \end{bmatrix}.$$

$$\mathrm{Q(DCT(Image2))} = \begin{bmatrix} 1 & 1 & 2 & 3 & 5 & 6 & 7 & 7 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 2 & 3 & 4 & 4 & 6 & 7 & 8 & 10 \\ 3 & 4 & 5 & 6 & 8 & 9 & 10 & 11 \\ 5 & 5 & 6 & 8 & 9 & 10 & 12 & 12 \\ 6 & 6 & 7 & 9 & 10 & 11 & 12 & 14 \\ 6 & 8 & 8 & 10 & 11 & 12 & 14 & 16 \\ 8 & 9 & 9 & 11 & 12 & 14 & 15 & 18 \end{bmatrix}.$$

$$\mathrm{AVG(Q(DCT(Images1,2,3)))} = \begin{bmatrix} 1 & 1 & 2 & 4 & 6 & 8 & 11 & 13 \\ 1 & 2 & 3 & 5 & 7 & 10 & 12 & 14 \\ 3 & 4 & 5 & 6 & 9 & 12 & 14 & 17 \\ 5 & 6 & 7 & 9 & 12 & 14 & 17 & 18 \\ 7 & 9 & 10 & 12 & 14 & 17 & 19 & 20 \\ 10 & 12 & 12 & 15 & 16 & 18 & 20 & 21 \\ 12 & 15 & 16 & 18 & 19 & 21 & 22 & 24 \\ 14 & 17 & 18 & 20 & 21 & 22 & 23 & 25 \end{bmatrix}.$$

$$10 * \mathrm{KT1(Image2)} = \begin{bmatrix} 99 & 94 & 53 & 35 & 27 & 24 & 24 & 27 \\ 92 & 42 & 29 & 21 & 18 & 16 & 17 & 19 \\ 51 & 29 & 19 & 16 & 13 & 12 & 12 & 14 \\ 35 & 20 & 16 & 12 & 11 & 10 & 11 & 12 \\ 26 & 17 & 13 & 10 & 10 & 9 & 9 & 11 \\ 24 & 15 & 12 & 11 & 9 & 9 & 9 & 11 \\ 23 & 14 & 12 & 9 & 10 & 9 & 9 & 10 \\ 25 & 16 & 14 & 12 & 11 & 10 & 12 & 12 \end{bmatrix}.$$

$$\mathrm{Q}(10 * \mathrm{KT1(Image2)}) = \begin{bmatrix} 1 & 1 & 2 & 3 & 4 & 4 & 4 & 4 \\ 1 & 2 & 3 & 5 & 6 & 6 & 6 & 5 \\ 2 & 3 & 5 & 6 & 8 & 8 & 8 & 7 \\ 3 & 5 & 6 & 8 & 9 & 10 & 9 & 8 \\ 4 & 6 & 8 & 10 & 10 & 12 & 11 & 9 \\ 4 & 7 & 8 & 9 & 11 & 12 & 11 & 9 \\ 4 & 7 & 8 & 11 & 10 & 11 & 11 & 10 \\ 4 & 6 & 7 & 8 & 9 & 10 & 9 & 8 \end{bmatrix}.$$

$$\mathrm{AVG}(\mathrm{Q}(10*\mathrm{KT1}(\mathrm{Images}1,2,3))) = \begin{bmatrix} 1 & 1 & 2 & 3 & 5 & 5 & 6 & 6 \\ 1 & 3 & 4 & 6 & 8 & 9 & 10 & 9 \\ 3 & 5 & 7 & 9 & 11 & 12 & 12 & 11 \\ 4 & 7 & 9 & 11 & 13 & 14 & 15 & 12 \\ 5 & 9 & 11 & 13 & 15 & 16 & 16 & 13 \\ 6 & 10 & 12 & 14 & 16 & 17 & 17 & 13 \\ 7 & 11 & 13 & 15 & 16 & 17 & 16 & 13 \\ 6 & 10 & 12 & 13 & 14 & 14 & 13 & 10 \end{bmatrix}.$$

$$\mathrm{KT2}(\mathrm{Image}2) = \begin{bmatrix} 0 & 1461 & 895 & 1974 & 929 & 1099 & 373 & 219 \\ 1472 & 1459 & 2312 & 1859 & 1840 & 1074 & 673 & 249 \\ 875 & 2307 & 2044 & 3045 & 1928 & 1772 & 763 & 366 \\ 1960 & 1936 & 3063 & 2432 & 2485 & 1423 & 907 & 324 \\ 906 & 1901 & 1881 & 2537 & 1739 & 1488 & 691 & 324 \\ 1074 & 1117 & 1710 & 1418 & 1449 & 869 & 547 & 212 \\ 357 & 684 & 740 & 905 & 690 & 548 & 286 & 138 \\ 200 & 238 & 352 & 319 & 316 & 210 & 138 & 61 \end{bmatrix}.$$

$$\mathrm{Q}(\mathrm{KT2}(\mathrm{Image}2)) = \begin{bmatrix} 1 & 2 & 3 & 2 & 3 & 3 & 8 & 14 \\ 2 & 2 & 1 & 2 & 2 & 3 & 5 & 12 \\ 4 & 1 & 1 & 1 & 2 & 2 & 4 & 8 \\ 2 & 2 & 1 & 1 & 1 & 2 & 3 & 9 \\ 3 & 2 & 2 & 1 & 2 & 2 & 4 & 9 \\ 3 & 3 & 2 & 2 & 2 & 4 & 6 & 14 \\ 9 & 4 & 4 & 3 & 4 & 6 & 11 & 22 \\ 15 & 13 & 9 & 10 & 10 & 15 & 22 & 50 \end{bmatrix}.$$

$$\mathrm{AVG}(\mathrm{Q}(\mathrm{KT2}(\mathrm{Images1,2,3}))) = \begin{bmatrix} 1 & 2 & 4 & 2 & 4 & 3 & 10 & 15 \\ 3 & 2 & 2 & 2 & 2 & 4 & 6 & 17 \\ 5 & 1 & 2 & 1 & 2 & 2 & 5 & 9 \\ 2 & 2 & 1 & 2 & 2 & 3 & 4 & 13 \\ 4 & 2 & 2 & 1 & 2 & 2 & 5 & 11 \\ 4 & 4 & 2 & 3 & 3 & 5 & 7 & 20 \\ 12 & 5 & 5 & 4 & 5 & 6 & 14 & 28 \\ 18 & 18 & 11 & 13 & 12 & 21 & 30 & 71 \end{bmatrix}.$$

$$\mathrm{KT3}(\mathrm{Image}2) = \begin{bmatrix} 39 & 86 & 45 & 55 & 27 & 23 & 16 & 14 \\ 86 & 53 & 49 & 34 & 25 & 19 & 16 & 14 \\ 44 & 50 & 34 & 31 & 22 & 17 & 13 & 12 \\ 54 & 35 & 32 & 22 & 18 & 13 & 12 & 10 \\ 27 & 27 & 20 & 18 & 14 & 11 & 10 & 9 \\ 22 & 18 & 15 & 13 & 12 & 10 & 8 & 7 \\ 16 & 15 & 13 & 11 & 10 & 9 & 8 & 7 \\ 13 & 11 & 11 & 10 & 9 & 8 & 7 & 6 \end{bmatrix}.$$

$$
\mathrm{Q(KT3(Image2))} = \begin{bmatrix} 2 & 1 & 2 & 2 & 3 & 4 & 5 & 6 \\ 1 & 2 & 2 & 3 & 3 & 5 & 5 & 6 \\ 2 & 2 & 3 & 3 & 4 & 5 & 6 & 7 \\ 2 & 2 & 3 & 4 & 5 & 7 & 7 & 9 \\ 3 & 3 & 4 & 5 & 6 & 8 & 9 & 10 \\ 4 & 5 & 6 & 6 & 7 & 9 & 10 & 12 \\ 5 & 6 & 7 & 8 & 8 & 10 & 11 & 13 \\ 7 & 8 & 8 & 9 & 10 & 11 & 12 & 15 \end{bmatrix}.
$$

$$
\mathrm{AVG(Q(KT3(Images1,2,3)))} = \begin{bmatrix} 3 & 1 & 2 & 2 & 4 & 4 & 8 & 10 \\ 1 & 2 & 2 & 3 & 4 & 7 & 9 & 11 \\ 3 & 2 & 3 & 3 & 5 & 7 & 10 & 12 \\ 2 & 3 & 4 & 5 & 7 & 10 & 12 & 15 \\ 4 & 4 & 6 & 6 & 9 & 11 & 15 & 16 \\ 5 & 8 & 8 & 10 & 12 & 15 & 16 & 18 \\ 9 & 10 & 11 & 13 & 14 & 17 & 17 & 19 \\ 12 & 14 & 14 & 16 & 17 & 18 & 18 & 20 \end{bmatrix}.
$$

Table 1. Krawtchouk matrices ("masks") for $p = M + 1 = 2$ to 8.

$$K^{(2)} = \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix},$$

$$K^{(3)} = \begin{bmatrix} 1 & 1 & 1 \\ -2 & 0 & 2 \\ 1 & -1 & 1 \end{bmatrix},$$

$$K^{(4)} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -3 & -1 & 1 & 3 \\ 3 & -1 & -1 & 3 \\ -1 & 1 & -1 & 1 \end{bmatrix},$$

$$K^{(5)} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -4 & -2 & 0 & 2 & 4 \\ 6 & 0 & -2 & 0 & 6 \\ -4 & 2 & 0 & -2 & 4 \\ 1 & -1 & 1 & -1 & 1 \end{bmatrix},$$

$$K^{(6)} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ -5 & -3 & -1 & 1 & 3 & 5 \\ 10 & 2 & -2 & -2 & 2 & 10 \\ -10 & 2 & 2 & -2 & -2 & 10 \\ 5 & -3 & 1 & 1 & -3 & 5 \\ -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix},$$

$$K^{(7)} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -6 & -4 & -2 & 0 & 2 & 4 & 6 \\ 15 & 5 & -1 & -3 & -1 & 5 & 15 \\ -20 & 0 & 4 & 0 & -4 & 0 & 20 \\ 15 & -5 & -1 & 3 & -1 & -5 & 15 \\ -6 & 4 & -2 & 0 & 2 & -4 & 6 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix},$$

$$K^{(8)} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ -7 & -5 & -3 & -1 & 1 & 3 & 5 & 7 \\ 21 & 9 & 1 & -3 & -3 & 1 & 9 & 21 \\ -35 & -5 & 5 & 3 & -3 & -5 & 5 & 35 \\ 35 & -5 & -5 & 3 & 3 & -5 & -5 & 35 \\ -21 & 9 & -1 & -3 & 3 & 1 & -9 & 21 \\ 7 & -5 & 3 & -1 & -1 & 3 & -5 & 7 \\ -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 \end{bmatrix}.$$