What's the Deal with the DCT?

James F. Blinn, *California Institute of Technology*

The discrete cosine transform is the basis of most image compression techniques today. Why is it better than the discrete Fourier transform?

Image compression is all the rage these days, and many popular techniques are based on a gimmick called the discrete cosine transform (DCT). It shows up in JPEG (Joint Photographic Experts Group) and MPEG (Motion Picture Experts Group) compression schemes, in techniques for picture phone transmission, and in all four of the proposals the FCC is considering as high definition television standards to replace NTSC. Why is the DCT so wonderful, and in particular, why is it better than the easier-to-compute discrete Fourier transform (DFT)? In this column, I'm going to play around with both these transforms, review their properties, and show an interesting way of thinking about them.

What are we talking about?

A function is a little machine that turns numbers into other numbers. You feed a number in, you get another number out. A transform is a somewhat bigger machine that turns functions into other functions. For example, the Fourier transform takes a continuous function of time (like a sound wave) or space (like an image), which we'll call $s(x)$, and turns it into a continuous function of frequency, $F(f)$. The *discrete* Fourier transform takes s_x, a list of samples of a function (indexed by discrete time or space steps) and changes it into a list of samples of another function (indexed by discrete frequency steps), F_f.

The general DFT and DCT equations have lots of variables and subscripts that make them a bit confusing to understand. Fortunately, I can simplify this a bit. All of the above image compression techniques divide an image into 8×8 pixel blocks and perform a 2D, eight-element DCT on the blocks. For this reason we only need to discuss eight-element transforms. This lets me explicitly write out some formulas instead of getting buried in subscripts.

For the purposes of this column, I will describe all frequencies in units of "cycles per eight samples." Thus a frequency of 1 will be a sine or cosine wave with one cycle across the eight samples. According to the sampling theorem, we can't expect to represent anything higher than a frequency of 4, commonly called the Nyquist rate.

The discrete Fourier transform

Let's start by considering only one-dimensional transforms. We'll first look at an eight-element DFT. The official definition is

$$F_f = \sum_{x=0}^{7} s_x e^{-2\pi i f x/8}$$

This takes a list of eight pixel values s_x and generates a list of eight complex numbers F_f. (Note that the i in the above is not an index, it's the square root of –1.) A more edifying way of writing this is in terms of trigonometric functions. We can separate the real and imaginary parts of the result as follows:

$$F_f = \sum_{x=0}^{7} s_x \cos\left(\frac{2\pi f x}{8}\right) - i\sum_{x=0}^{7} s_x \sin\left(\frac{2\pi f x}{8}\right)$$

It'll be useful to give names to the separate real and imaginary parts of the F values, so

$$F_{Rf} = \sum_{x=0}^{7} s_x \cos\left(\frac{2\pi f x}{8}\right)$$

and

$$F_{If} = \sum_{x=0}^{7} s_x \sin\left(\frac{2\pi f x}{8}\right)$$

Now, let's admire these equations a bit. See the summation? See the product of two values under the sum? What does this remind us of? Hmm . . . Yes! It's a matrix product. We can write the Fourier transform as a matrix of cosines and sines times a vector of samples s_x. What's more, since we only evaluate the sine and cosine at integer multiples of $2\pi/8 = 45°$, the actual values in the sine and cosine matrices are particularly simple. Let's give a name to $\sin(45°) = \cos(45°) = \sqrt{2}/2 \equiv r$. Explicitly calculating the cosines, we can write the real part of the transform as

$$\begin{bmatrix} F_{R0} \\ F_{R1} \\ F_{R2} \\ F_{R3} \\ F_{R4} \\ F_{R5} \\ F_{R6} \\ F_{R7} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & r & 0 & -r & -1 & -r & 0 & r \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 1 & -r & 0 & r & -1 & r & 0 & -r \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -r & 0 & r & -1 & r & 0 & -r \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 1 & r & 0 & -r & -1 & -r & 0 & r \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{bmatrix}$$

Giving roman letter names to vectors and boldface names to

matrices, we can condense this to

$$F_R = \mathbf{R}s$$

I really wanted to write the matrix equation transposed, as a row vector times the matrix. That would make the condensed version look more like the summation version. But two eight-element row vectors don't fit on the page very well, so we'll have to use the column-vector notation.

Anyway, now let's write the imaginary part of the DFT as a matrix

$$\begin{bmatrix} F_{I0} \\ F_{I1} \\ F_{I2} \\ F_{I3} \\ F_{I4} \\ F_{I5} \\ F_{I6} \\ F_{I7} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & r & 1 & r & 0 & -r & -1 & -r \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\ 0 & r & -1 & r & 0 & -r & 1 & -r \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -r & 1 & -r & 0 & r & -1 & r \\ 0 & -1 & 0 & 1 & 0 & -1 & 0 & 1 \\ 0 & -r & -1 & -r & 0 & r & 1 & r \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{bmatrix}$$

Naming the vectors and matrix, we get

$$F_I = \mathbf{S}s$$

Let's pause to observe some patterns in the elements of the above two matrices. First, note that they are both symmetric and are both singular (because of repeated rows). Each row of **R** is a cosine function digitized at some sample rate, while each row of **S** is a sine function digitized at the same rate. I will call a row of such a matrix, plotted against its index, a *basis function.*

A pair of corresponding rows from **R** and **S**, when dotted with the column vector of samples, gives the real and imaginary part of the frequency response for that frequency: Row 0 (the top row) gives the amount of frequency 0 (also called the DC component), row 1 gives the amount of frequency 1, and so forth.

Note that there are lots of common terms in the arithmetic for computing the matrix products. For example, the product rs_1 shows up in rows 1, 3, 5, and 7. The fast Fourier transform is simply an organized way of exploiting these common computations.

The inverse DFT

The inverse transform is defined as

$$s_x = \frac{1}{8}\sum_{f=0}^{7} F_f \cos\left(\frac{2\pi fx}{8}\right) + i\frac{1}{8}\sum_{f=0}^{7} F_f \sin\left(\frac{2\pi fx}{8}\right)$$

The reason this works can be easily seen in terms of matrix products. The forward transform is the matrix

$$\mathbf{R} - i\mathbf{S}$$

The matrix form of the inverse is

$$\frac{1}{8}(\boldsymbol{R} + i\boldsymbol{S})$$

Multiplying these two matrices gives

$$\frac{1}{8}((\boldsymbol{RR} + \boldsymbol{SS}) + i(\boldsymbol{SR} - \boldsymbol{RS}))$$

The basis functions for the Fourier transform (the rows of the matrix) are *orthogonal.* That means that, if you take the dot product of any two different rows, you get zero. This makes the product of **R** times **R** and of **S** times **S** easy to calculate (recall that they're symmetrical). The result is quite pretty:

$$\mathbf{RR} = \begin{bmatrix} 8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 4 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 4 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 0 & 4 & 0 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 4 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 & 4 \end{bmatrix}$$

The square of the **S** matrix is

$$\mathbf{SS} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 & -4 \\ 0 & 0 & 4 & 0 & 0 & 0 & -4 & 0 \\ 0 & 0 & 0 & 4 & 0 & -4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -4 & 0 & 4 & 0 & 0 \\ 0 & 0 & -4 & 0 & 0 & 0 & 4 & 0 \\ 0 & -4 & 0 & 0 & 0 & 0 & 0 & 4 \end{bmatrix}$$

Note that some rows of **R** and **S** are identical, otherwise their squares would be diagonal matrices. Anyway, add these matrices, divide by 8 and, voilà, an identity matrix.

Now how about the imaginary component? Well, the product of **R** and **S** is all zeroes because you always get different rows when picking one each from **R** and **S**.

A more compact form

The eight-element DFT takes a list of eight numbers and produces 16 numbers (eight real and eight imaginary coefficients). This seems to violate the law of conservation of information. We expect to start with eight numbers into the DFT and get eight independent numbers out. And that's really what happens. Note that, in matrix **R**, row 3 equals row 5, row 2 equals row 6, and row 1 equals row 7 (we start numbering from zero). In other words, rows 5, 6, and 7 give redundant information; there are only 5 independent rows. Likewise, in matrix **S**, row 0 and row 4 are really boring, row 3 equals minus row 5, row 2 equals minus row 6, and row 1 equals minus row 7; so only rows 1, 2, and 3 give interesting information.

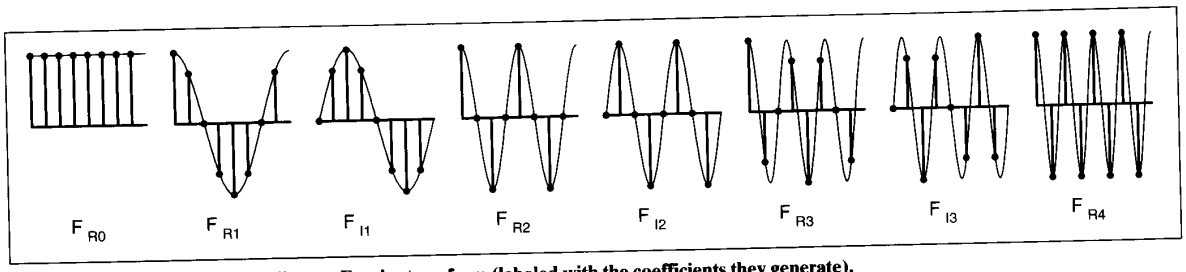


Figure 1. Basis functions for the discrete Fourier transform (labeled with the coefficients they generate).

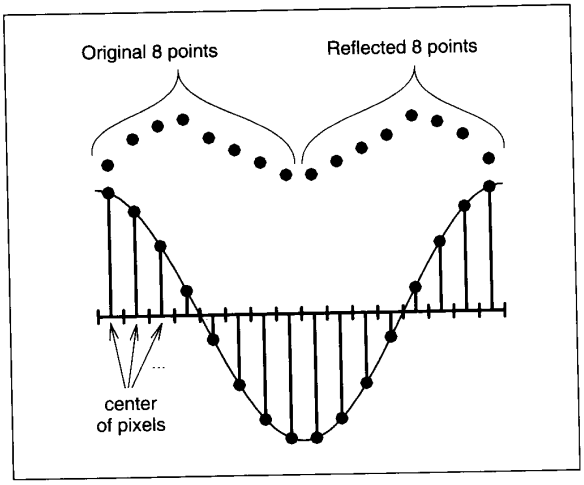
Let's formulate a transformation matrix that just generates the interesting information from the DFT. We can define a new output vector containing just the unique elements of F: F_0, the real and imaginary parts of F_1, F_2, and F_3, and the real part of F_4. This gives us

$$\begin{bmatrix} F_{R0} \\ F_{R1} \\ F_{I1} \\ F_{R2} \\ F_{I2} \\ F_{R3} \\ F_{I3} \\ F_{R4} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & r & 0 & -r & -1 & -r & 0 & r \\ 0 & r & 1 & r & 0 & -r & -1 & -r \\ 1 & 0 & -1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & -1 \\ 1 & -r & 0 & r & -1 & r & 0 & -r \\ 0 & r & -1 & r & 0 & -r & 1 & -r \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{bmatrix}$$

Let's call this gigantic matrix **F**. Row 0 gives the DC component. Rows 1 and 2 are digitized cosine and sine at frequency 1. Rows 3 and 4 are digitized cosine and sine at frequency 2. Rows 5 and 6 are digitized cosine and sine of frequency 3. Row 7 is a cosine of frequency 4. Figure 1 shows a plot of all eight of these basis functions. In these plots, I overlaid the curve of the continuous sines and cosines for reference.

We can also think of the results of this transform in terms of magnitudes and phase angles. We get a magnitude for frequencies 0, 1, 2, 3, and 4, and we get phase angles for frequencies 1, 2, and 3. The phase of frequencies 0 and 4 are implicitly zero.

Figure 2. Centering of the discrete cosine transform basis function $G_{1/2}$.



There's something special about this matrix. Can you find it? It's based on the orthogonality property of the basis vectors. Orthogonality implies that multiplying **F** times the transpose of **F** gives you something that's almost an identity matrix. In particular, you get a diagonal matrix with the diagonal elements having values (8, 4, 4, 4, 4, 4, 4, 8). I'll let you ponder this a bit, while I proceed to the DCT.

The discrete cosine transform

As I've indicated, the discrete Fourier transform generates a complex valued result. We use the discrete *cosine* transform as a way of doing frequency analysis without needing complex numbers. There are two ways of thinking about how the DCT does this: We either modify the signal or we modify the basis functions.

The signal modification approach works as follows. First, note that all imaginary components of the Fourier transform come from multiplication by a digitized sine wave (an antisymmetric function) at some frequency. Specifically, each row of matrix **S** is antisymmetric about column 4. If the signal happened to be symmetric, that is, if $s_1 = s_7$, $s_2 = s_6$, and $s_3 = s_5$, then all the terms would cancel out and all the imaginary components would be zero. The DCT builds on this concept with a few modifications. An eight-element DCT generates an artificial symmetric 16-element signal by appending a reversed copy of the signal to itself. The DCT then performs a variant of a 16-element DFT on the symmetric signal. The variation, as illustrated in Figure 2, centers the 16-element signal within the cosine or sine wave. (Of course, since the signal is symmetric, we don't have to do the left-hand and right-hand arithmetic twice.) The result is eight unique nonzero terms from the cosines and eight zero terms from the sines.

In the basis function interpretation, we simply use basis functions that are only cosines, but we also allow half integer frequencies. That is, instead of using frequencies 0, 1, 2, 3, and 4, we use frequencies 0, 0.5, 1, 1.5, 2, 2.5, 3, and 3.5. Again, for symmetry, we don't start digitizing the cosine basis functions at zero degrees, but at an offset of one half the pixel spacing.

This all boils down to an official definition of the DCT as

$$G_{u/2} = \frac{1}{2} K_u \sum_{x=0}^{7} s_x \cos\left(\frac{(2x+1)u\pi}{16}\right)$$

where $K_0 = \sqrt{2}/2$ and $K_u = 1$ for $u \neq 0$.

Now let's discuss some of the details of this definition. First, note the somewhat funny "subscripting" of the output elements, $G_{u/2}$. I did this to label each output coefficient with the frequency it stands for. I'll tell you the reason for the K_u factor in a minute.

Table 1. Values of c_j.

	Angle		Value			
	radians	degrees	precise	numeric	polynomial	recursive
c_0	0	0	1	1	1	1
c_1	$\pi/16$	11.25	$\frac{1}{2}\sqrt{2+\sqrt{2+\sqrt{2}}}$	0.9807853	c_1	c_1
c_2	$2\pi/16$	22.50	$\frac{1}{2}\sqrt{2+\sqrt{2}}$	0.9238794	$2c_1^2-1$	$2c_1c_1-c_0$
c_3	$3\pi/16$	33.75	complicated	0.8314695	$4c_1^3-3c_1$	$2c_1c_2-c_1$
c_4	$4\pi/16$	45	$\frac{1}{2}\sqrt{2}$	0.7071065	$8c_1^4-8c_1^2+1$	$2c_1c_3-c_2$
c_5	$5\pi/16$	56.25	complicated	0.5555699	$16c_1^5-20c_1^3+5c_1$	$2c_1c_4-c_3$
c_6	$6\pi/16$	67.50	$\frac{1}{2}\sqrt{2-\sqrt{2}}$	0.3826829	$32c_1^6-48c_1^4+18c_1^2-1$	$2c_1c_5-c_4$
c_7	$7\pi/16$	78.75	$\frac{1}{2}\sqrt{2-\sqrt{2+\sqrt{2}}}$	0.1950897	$64c_1^7-112c_1^5+56c_1^3-7c_1$	$2c_1c_6-c_5$

The DCT formula only uses samples of the cosine function spaced at integer multiples of $\pi/16$, so we can see what is going on a bit more easily by defining the function

$$c_j = \cos\left(\frac{j\pi}{16}\right)$$

You really need to know only the values from the first quadrant of the cosine function, $c_0 \ldots c_7$, since you can get the cosine of all other angles by shifting and reflecting them back to the first quadrant. I've listed these eight values in Table 1. Since they're important, I cataloged several ways to compute them. Some of these might help speed up the computation, but I'm not sure.

So, finally writing the definition of the DCT as a matrix gives us

$$\begin{bmatrix} G_0 \\ G_{1/2} \\ G_1 \\ G_{3/2} \\ G_2 \\ G_{5/2} \\ G_3 \\ G_{7/2} \end{bmatrix} = \frac{1}{2}\begin{bmatrix} c_4 & c_4 & c_4 & c_4 & c_4 & c_4 & c_4 & c_4 \\ c_1 & c_3 & c_5 & c_7 & -c_7 & -c_5 & -c_3 & -c_1 \\ c_2 & c_6 & -c_6 & -c_2 & -c_2 & -c_6 & c_6 & c_2 \\ c_3 & -c_7 & -c_1 & -c_5 & c_5 & c_1 & c_7 & -c_3 \\ c_4 & -c_4 & -c_4 & c_4 & c_4 & -c_4 & -c_4 & c_4 \\ c_5 & -c_1 & c_7 & c_3 & -c_3 & -c_7 & c_1 & -c_5 \\ c_6 & -c_2 & c_2 & -c_6 & -c_6 & c_2 & -c_2 & c_6 \\ c_7 & -c_5 & c_3 & -c_1 & c_1 & -c_3 & c_5 & -c_7 \end{bmatrix}\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{bmatrix}$$

We'll name the vector and matrix as follows:

$$G = \mathbf{C}s$$

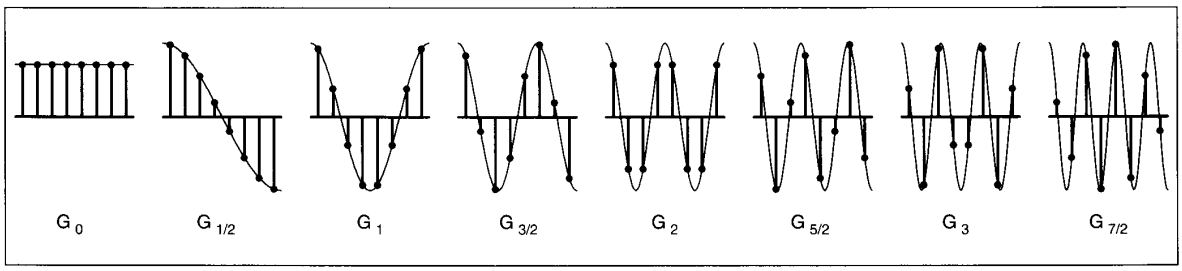
A few notes: The factor 1/2 is built into the matrix **C**. I left it outside for neatness when I wrote the explicit matrix. Also, note that, according to the formula, the top row of the matrix should be a row of the values K_0c_0. Since $K_0 = c_4$ and $c_0 = 1$, I just used c_4 for the top row.

Perusing the matrix, we can see various patterns. For example, if we ignore signs and treat the top row c_4 as a disguised c_0, we see that each column contains each value of c_j exactly once. Each row of the matrix is a digitized cosine of frequencies 0, 0.5, 1, 1.5, 2, 2.5, 3, and 3.5. There is no wave at the Nyquist rate of 4. Note that row 1 has one sign change, row 2 has two sign changes, and so forth. A plot of these basis functions appears in Figure 3.

This matrix **C** is not symmetric, but it does have the orthogonality property. In addition, each row of the matrix dotted with itself gives the value 1. (We invented the factor of K_0 to make this true for the top row too.) In other words, matrix **F** times the transpose of matrix **F** gives an identity matrix; the transpose is the inverse. And what does this mean?

> The DCT matrix is a rotation matrix in eight-dimensional space.

Figure 3. Basis functions for the discrete cosine transform (labeled with the coefficients they generate).



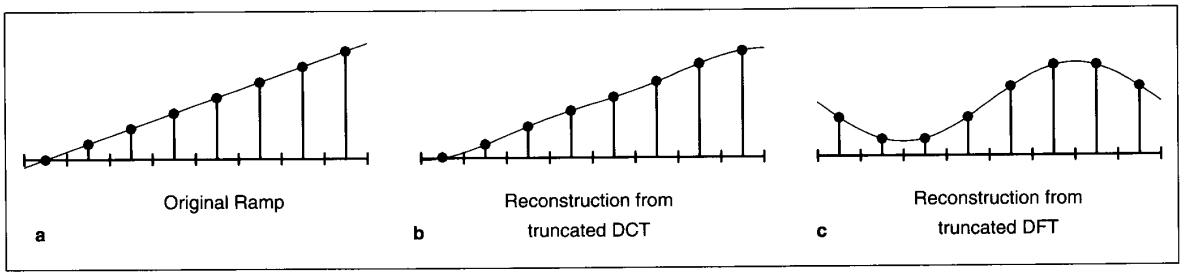


Figure 4. A comparison of the accuracy of truncated discrete cosine transforms and discrete Fourier transforms.

Among other things, this means that the Euclidean length of a sample vector remains unchanged after rotation. So

$$\sum G_{j/2}^2 = \sum s_j^2$$

This quantity is sometimes referred to as the energy of the signal. The nice thing about both the DFT and DCT is that they pack more of the energy into the lower order coefficients, so we don't lose energy by dropping some of the high-order coefficients.

Comparing the DFT with the DCT

Our compact form of the DFT—the matrix **F**—was almost a rotation matrix. We can modify it a bit to make it exactly a rotation by scaling the top and bottom rows by $\sqrt{1/8}$ and the inside rows by $\sqrt{1/4}$. We would then have to define some scaled frequency components like $\tilde{F}_{R0} = \sqrt{1/8}\,F_{R0}$, $\tilde{F}_{R1} = \sqrt{1/4}\,F_{R1}$, and so forth, as a result of the transform. I wont explicitly rewrite the matrix equation for this, but I will use the pure rotation version in the example below.

The main point of comparing the DFT with the DCT is the amount of arithmetic necessary. Note that, with the DFT, all the multiplications involve the rather simple numbers 0, 1, –1, r, and $-r$. In fact, you can perform the compact DFT with only four multiplications of r by s_1, s_3, s_5, and s_7, and a handful of additions and subtractions. On the other hand, examining the DCT matrix product reveals that we must calculate all possible permutations of products of c_j with s_x. That's sixty four multiplications.

Why is the DCT better?

The DCT is harder to compute, so there must be some good reason for using it. I believe the reason can be simply stated: We need fewer DCT coefficients than DFT coefficients to get a good approximation to a typical signal. The whole reason for performing a transform is that we expect that the higher frequency coefficients are small in magnitude and can be more crudely quantized than the low-frequency coefficients.

For example, consider a simple ramp function, shown in Figure 4a. The DCT coefficients and the DFT coefficients appear in Table 2. Now suppose, for compression purposes, we keep only the three largest coefficients of each of these transforms and then perform the inverse transform. The result appears in Table 3 and is plotted in Figures 4b and 4c. You can see that the DCT result is much better. This is basically because the DFT approximation is trying to model a repeated ramp, or sawtooth wave. It has to devote a lot of high-frequency coefficients to approximate the (actually spurious) discontinuities from one copy of the wave to the next. The DCT is effectively operating on a triangle wave, one formed from the ramp alternating with its reflection. There is no discontinuity and thus less need for high frequency coefficients to model one. The DCT is thus better

Table 2. Transforms of ramp function.

s_x	DCT	DFT
0	356.38	356.38
36	–213.92	–72.00
72	0.00	–173.82
108	–24.24	–72.00
144	0.00	–72.00
180	–7.23	–72.00
216	0.00	–29.82
252	–1.83	–50.91

Table 3. Accuracy of truncated transforms.

Truncated DCT	IDCT	Truncated DFT	IDFT
356.38	2.19	356.38	90.00
–213.92	31.95	–72.00	39.09
0.00	73.46	–173.82	39.09
–24.24	110.11	0.00	90.00
0.00	141.89	0.00	162.00
0.00	178.54	0.00	212.91
0.00	220.05	0.00	212.91
0.00	249.81	0.00	162.00

equipped to model finite-length pieces of a function (eight samples worth in our examples) that have fairly different values on their right and left sides. In fact, you can do approximate linear interpolation between any two endpoint values of a pixel row by using just the two lowest-frequency DCT basis functions.

The 2D DCT

Image compression uses a 2D DCT applied to blocks of 8 × 8 pixels. The 2D DCT consists of taking a one-dimensional DCT of each column of pixels followed by a one-dimensional DCT of each row of the result. As a result, this gives us an 8 × 8 matrix **G**. In matrix terms this is simply

$$\mathbf{G} = \mathbf{F}\mathbf{s}\mathbf{F}^t$$

where each term in the above equation is an 8 × 8 matrix.

Why is this interesting?

DCT-based image compression takes advantage of the fact that most images don't have much energy in the high-frequency coefficients. Thinking of the DCT as an eight-dimensional rotation gives us another way to look at this. Consider a row of eight pixels as a point in eight-dimensional space. Various rows of eight pixels from various images would make a cluster of points in this space. For typical images, this cluster is not uniformly dispersed; it is squashed fairly flat in some directions. The eight-element DCT rotates this cluster so the flat directions line up along some of the coordinate axes. You can then get away with representing the points of the cluster (each of which represents a row of eight pixels) in a lower dimensional space. The DCT doesn't always find the flattest direction, though. In fact, analyzing images for the direction (in eight-space) where the cluster is flattest corresponds to looking for other sets of basis vectors to use for a transformation. This is the idea behind what is called the Karhunen-Loéve transform. The DCT comes pretty close to the Karhunen-Loéve, though.

Finally, the representation of the DCT or DFT as a matrix and the inverse transform as a matrix inversion bolsters my far-reaching claim:

> All problems in computer graphics can be solved with a matrix inversion. ❑

Further DCT reading

R.J. Clarke, *Transform Coding of Images*, Academic Press, San Diego, 1985, Ch. 3, "Orthogonal Transforms for Image Coding."

G.K. Wallace, "The JPEG Still Picture Compression Standard," *Comm. ACM*, Vol. 34, No. 4, April 1991, pp. 31-44.

Erratum

In my last column (May 1993, pp. 75-80), "A Trip Down the Graphics Pipeline: The Homogeneous Perspective Transform," reference 2 was incorrect. The correct reference is

> R.F. Riesenfeld, "Homogeneous Coordinates and Projective Planes in Computer Graphics," IEEE CG&A, Vol. 1, No. 1, Jan. 1981, pp. 50-55.