

## Supplement to Chapter 2 of *The Science of Digital Media* – Digital Image Representation

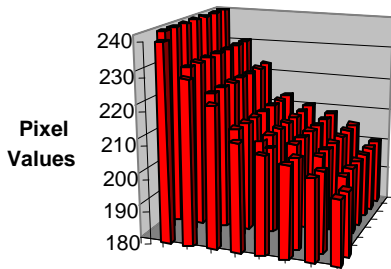
### Programming Exercise – Digital Imaging > Discrete Cosine Transform<sup>1</sup>

---

#### Introduction:

The discrete cosine transform (DCT) is used in JPEG, MPEG, and other compression methods. The transform itself does not actually compress the image. Rather, it converts the pixel values from the *spatial domain* to the *frequency domain*. Once the image has been converted to the frequency domain, the high-frequency components can be coarsely quantized without perceptible deterioration of image quality.

In converting the image from the spatial domain to the frequency domain, the DCT transforms an array of pixel values into an equal-size array of frequency components, which also represent the image.



For example, imagine that the graph to the left represents the grayscale values for an 8 x 8 square of pixels in an image. (Generalizing to RGB entails treating each of the color channels in a similar manner.) These values can be pictured as a curve. While they vary horizontally, they stay constant vertically.

The DCT works by decomposing this waveform into a sum of cosine waves of different frequencies – which are called the frequency components.

The equation for the DCT is as follows:

$$F(u, v) = \sum_{r=0}^{M-1} \sum_{s=0}^{N-1} \frac{2C(u)C(v)}{\sqrt{MN}} f(r, s) \cos\left(\frac{(2r+1)u\pi}{2M}\right) \cos\left(\frac{(2s+1)v\pi}{2N}\right)$$

where  $C(\delta) = \frac{\sqrt{2}}{2}$  if  $\delta = 0$  otherwise  $C(\delta) = 1$

---

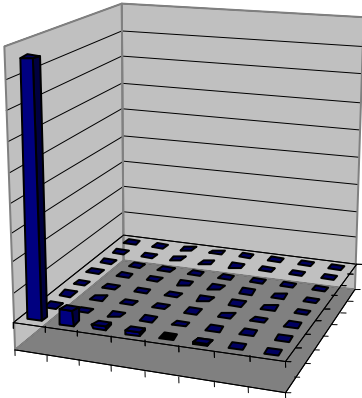
<sup>1</sup>This material is based on work supported by the National Science Foundation under Grant No. DUE-0340969. This programming assignment was written by Todd Martin and Jennifer Burg.

Each element of the array  $F(u, v)$  is a summation in which all elements of the original array  $f(r, s)$  are multiplied by a cosine wave of a certain frequency. This frequency depends on  $u$  and  $v$ , so each element of the transformed array represents a different frequency.

The DCT is invertible. That is, you can begin with the image data in the frequency domain and compute the image data in the spatial domain. This is done with the following equation:

$$f(r, s) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} \frac{2C(u)C(v)}{\sqrt{MN}} F(u, v) \cos\left(\frac{(2r+1)u\pi}{2M}\right) \cos\left(\frac{(2s+1)v\pi}{2N}\right)$$

where  $C(\delta) = \frac{\sqrt{2}}{2}$  if  $\delta = 0$  otherwise  $C(\delta) = 1$



The result of the DCT when performed on the example above is given by the graph to the left. The values represented on the graph are the coefficients of the frequency components, *not* pixel values. The value in the left corner, indicated by the green arrow, is called the *DC component*, while the other values, indicated by the red arrow, are called the *AC components*.

This terminology, borrowed from electrical engineering, refers to the fact that the DC component is an average of all of the pixels in the original image, multiplied by a constant, while the AC components represent the amplitudes of each of the frequency components.

The frequencies in the graph increase from left to right along the horizontal axis and front to back along the vertical axis. This can be deduced from the DCT equation. As  $u$  and  $v$  increase, so do the coefficients of the cosine values – and the corresponding frequencies. As you can see, most of the higher-frequency components in this image are 0.

This is why the DCT is essential to many compression methods. The human eye cannot detect high-frequency components of images – which represent very fast changes in color (over a short space). Thus, many compression methods like JPEG use a quantization matrix that more coarsely quantizes the high-frequency components, using fewer bits to store their values and

rounding many of them to 0. This, combined with the run-length encoding that can be performed on the 0s, is used to compress the image.

## **Instructions:**

### **The Assignment**

Using the programming language of your choice, implement the discrete cosine transform.

Your program should take as input a file containing an array of integer values representing the brightness of the image pixels. Prompt the user for the name of the input file. The image file should have the width and height of the image as integer values at the head of the file.

Design your DCT algorithm to handle pixel arrays that are  $8 \times 8$ .

Prompt the user for the name of the encoded image file, and save this as the output file. Be sure to output the height and width (in that order) before the data so that the inverse transform can be applied later.

Put the actions above in a loop in your main function, asking the user if he or she wants to transform another image.

### **Before Writing the Program**

To run your program, you will need input files, so create these first or ask your instructor if he or she plans to create them for you. The image files should contain an array of integers and a header of two integers giving the width and height of the image.

### **Ideas for Further Experimentation and Analysis**

- Implement the inverse discrete cosine transform (IDCT) algorithm. Compare the original file with the file you get after transforming and then performing the inverse.
- Alter your DCT implementation to allow arrays of dimensions that are *not*  $8 \times 8$ . To do this, you must pad your array with 0s to make the height and width multiples of 8, then split it into  $8 \times 8$  blocks to compute the transform.
- Try the DCT on different arrays. What happens when all of the integers are the same? What about when they are all different?
- Think about how the results of your DCT algorithm lend themselves to compression.