IMAGE COMPRESSION USING FOURIER TRANSFORMS

Kevin Cherry

May 2, 2008

Math 4325

Compression is a technique for storing files in less space than would normally be required. This in general, has two major purposes: making file transfer over various protocols (ftp, http, etc.) quicker and for saving storage space on hard drives. The normal downsides, however, are that images might not be able to be decompressed exactly as they were originally (lossy vs. lossless which will be discussed later), the compressing and decompressing process takes time (a time/space tradeoff as is often found in computers and computing technologies), and compatibility as one or more file types need to be made for each of the compression techniques.

The most common image compression formats include GIF, JPG (which includes .jpg, .jpeg, .jpe, .jp2, .j2c, with containers: .jif, .jfif, and .jfi) and PNG. Picking the correct compression format for an image depends on the type of image you want to compress. The GIF format works ideally for images with less than 256 distinct colors and those composed of basic geometrical shapes and cartoon-like drawings. Both PNG and GIF support transparency by selecting an unused color and marking that as the transparent color. JPG works best on photographic images and/or those with lots of distinct colors and details. GIF and PNG are both lossless compression methods meaning that these formats can be made to hold all data necessary to be decompressed into an exact replica of the original image. JPG, however, is considered lossy and even on the highest settings; it is incapable of restoring the exact original image.

How compression works varies greatly. GIF compression in its simplest form is nothing more than noting the repetition of adjacent pixels of the same color. For example, if we had an image whose top line of pixels were all white, and the width of the

image was 100, instead of storing the hex value #FFFFFF 100 times, we could just write 100#FFFFFF, meaning that the next 100 pixels are white. This is why simple images are ideal for GIF compression, but complex ones (where the pixels change colors frequently) aren't as effective. JPG compression is a lot more complex and involves the use of Discrete Cosine Transforms (DCT) and Discrete Fourier Transforms (DFT). I will cover these in detail a little later on, but first it is important to view the steps that take place before this is used. The best way to understand JPG compression is to go step-by-step through the methods used.

The first part of JPG compression is known as color space transformation. It was found that the human eye can detect somewhat minor changes in the brightness of an image however small changes in color aren't as noticeable. The color format YCbCr, in which Y stands for brightness, Cb stands for something known as chrominance blue, and Cr is for chrominance red, is used in place of other formats such as RGB or HSV. Since the human eye is less accepting of brightness changes, the Y value doesn't get touched. Instead Cb and Cr are downsampled, that is, spatial resolution is reduced in these values. Next the entire image is split into 8x8 pixel blocks and compression continues on each block independently.

This is where Discrete Fourier Transform (DFT) and Discrete Cosine Transform (DCT) come in. From my understanding you can use either one, however DCT is more recommended. I will start with an explanation on using DFT on an image. In **FIGURE 1** a random 2x2 image was given as f. Normally this would be a 8x8 made from the block splitting stage earlier, but for sake of simplicity, this 2x2 example will work. The color values come from the fact that 256 permutations are possible with 8 bits (where each bit

is a 1 or 0) and since the computer works in chunks of bits called bytes, and since 8 bits is a byte, each color can be represented using 1 byte. In the case of f, 255 represents pure white and 0 represents pure black. The numbers in between represent the amount of gray in the color with the higher values being lighter. Using the analysis equation, we are able to get the Fourier Transform of this image as shown in the equations that follow **FIGURE 1**. Using DCTs are a little more complicated. The first thing we must do is to subtract 128 (half of 256 as 256 is the current maximum for all values) from each of the original values. This changes the values to be centered on zero as opposed to all non-negative values. Next we apply the formula found in **FIGURE 3** to get G (the Fourier Transform of g, our original 8x8 block). Since this could yield fractional values, we need to round each value to the nearest integer. In the result the top left value is known as our DC coefficient and all other values in the block are known as AC coefficients.

The next step is known as quantization. Using a common quantization matrix like the one in **FIGURE 4**, we apply the formula given in **FIGURE 4** to obtain B. This step allows for the values found in G (which is now B) to be smaller and therefore require fewer bits to be stored by the computer. This step is mostly responsible for JPG not being able to represent the image exactly and is the cause of artifacts in the image that show up at lower qualities. This is due to the rounding that takes place.

The final step is called entropy coding. For this step we first arrange the numbers sequentially from the matrix using a "zigzag" pattern as shown in **FIGURE 5**. After this special code words are used to stand for certain patterns in the list of numbers. For example, EOB is used to mean that the rest of the list of numbers contains only zeros.

Using these code words instead of writing out the actual numbers saves a good bit (no pun intended) of space.

Just how good the compressed image appears depends on a quality/space tradeoff. The higher the quality, the less compressed the image is. The lower the quality, the more compressed but more inaccurate the image is. Due to its lossy nature, JPG will never exactly represent the uncompressed image, however in most images it can get quite close even at qualities of about 70%. The main degradation of GIF images comes into play when you decrease the number of colors GIF is allowed to use to display the image. The smaller the color palette is the more detail is lost.

JPG compression wouldn't be possible without the use of Fourier and Cosine Transforms. Because of these, it is one of the best compression methods for images. Without compression, there is a good chance image use among web pages would not be as common. Compression allows us to send data even over a slower connection in a feasible amount of time and to make better use of our hard drive space.

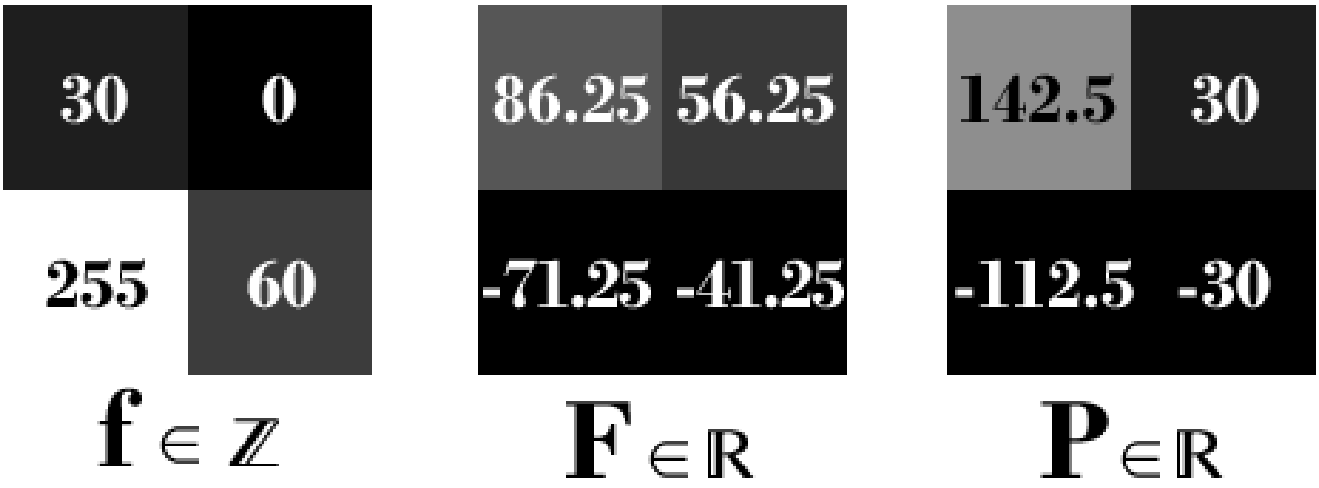

FIGURE 1

DFT values for a sample 2x2 image given by f.

$$f(a,b) = \frac{1}{N^2}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} F(k,l)e^{2\pi i(\frac{ka}{N}+\frac{lb}{N})} \qquad Synthesis\ Equation$$

$$F(k,l) = \frac{1}{N}\sum_{b=0}^{N-1} P(k,b)e^{-2\pi i\frac{lb}{N}} \qquad Analysis\ Equation$$

$$where\ P(k,b) = \frac{1}{N}\sum_{a=0}^{N-1} f(a,b)e^{-2\pi i\frac{ka}{N}}$$

Note: The analysis equation is normally broken up like this for computational efficiency. This reduces time complexity as we can calculate P(k,b) first and then simply look up those values when computing F(k,l). This turns the equation into two separate loops as opposed to two nested loops.

Work:

$$P(0,0) = \frac{1}{2}\sum_{a=0}^{1} f(a,0) = \frac{1}{2}(30+255) = 142.5$$

$$P(0,1) = \frac{1}{2}\sum_{a=0}^{1} f(a,1) = \frac{1}{2}(0+60) = 30$$

$$P(1,0) = \frac{1}{2}\sum_{a=0}^{1} f(a,0)e^{-\pi i a} = \frac{1}{2}\left(30+255e^{-\pi i}\right) = 15+127.5\,e^{-\pi i} = -112.5$$

$$P(1,1) = \frac{1}{2}\sum_{a=0}^{1} f(a,1)e^{-\pi i a} = \frac{1}{2}\left(0+60e^{-\pi i}\right) = 30\,e^{-\pi i} = -30$$

$$F(0,0) = \frac{1}{2}\sum_{b=0}^{1} P(0,b) = \frac{1}{2}(142.5+30) = 86.25$$

$$F(0,1) = \frac{1}{2}\sum_{b=0}^{1} P(0,b)e^{-\pi i b} = \frac{1}{2}\left(142.5+30e^{-\pi i}\right) = 56.25$$

$$F(1,0) = \frac{1}{2}\sum_{b=0}^{1} P(1,b) = \frac{1}{2}(-112.5-30) = -71.25$$

$$F(1,1) = \frac{1}{2}\sum_{b=0}^{1} P(1,b)e^{-\pi i b} = \frac{1}{2}\left(-112.5-30e^{-\pi i}\right) = -41.25$$

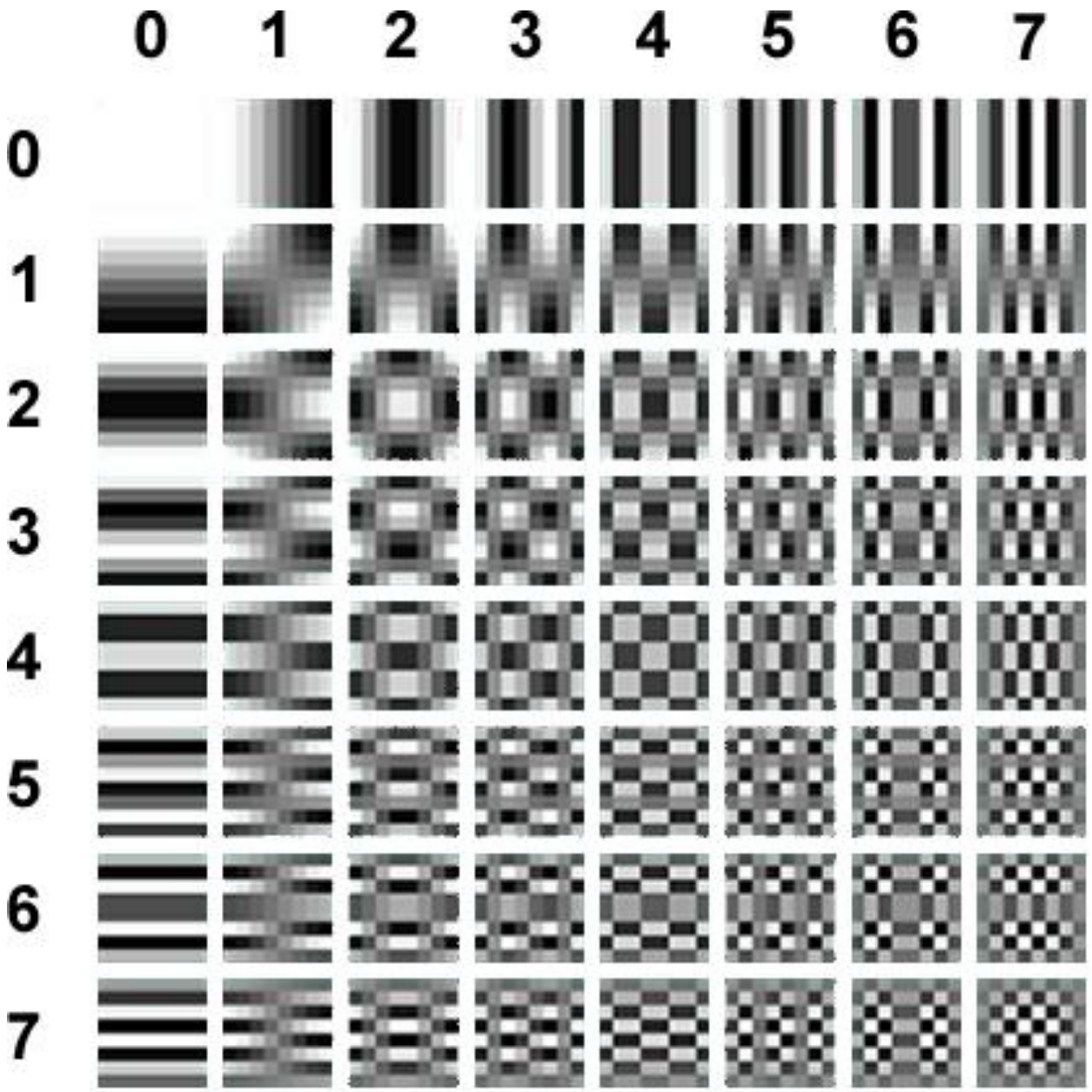

FIGURE 2

DCT Transforms used for finding horizontal and vertical spatial frequency, i.e. the measure of how often the structure repeats per unit distance, values for u and v respectively. In this case the unit distance is one square in this 8x8 grid.

$$G_{u,v} = \alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} g_{x,y}\cos\left[\frac{\pi}{8}\left(x+\frac{1}{2}\right)u\right]\cos\left[\frac{\pi}{8}\left(y+\frac{1}{2}\right)v\right]$$

FIGURE 3

where:
$u = horizontal\ spatial\ frequency\ of$ **FIGURE 2**
$v = vertical\ spatial\ frequency\ of$ **FIGURE 2**

$$\alpha_p(n) = \begin{cases} \sqrt{\frac{1}{8}}, & n = 0 \\ \sqrt{\frac{2}{8}}, & else \end{cases}$$

$g_{x,y} = pixel\ value\ at\ coordinates\ x, y$
$G_{x,y} = DCT\ coefficient\ at\ coordinates\ x, y$

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

FIGURE 4

Common quantization matrix used for scaling values of G. This uses the formula:

$$B_{j,k} = round\left(\frac{G_{j,k}}{Q_{j,k}}\right) \quad for\ 0 \leq j,k\ \leq 7$$

Where: Q is a common quantization matrix like **FIGURE 4**.

G is our Transform from the previous step.

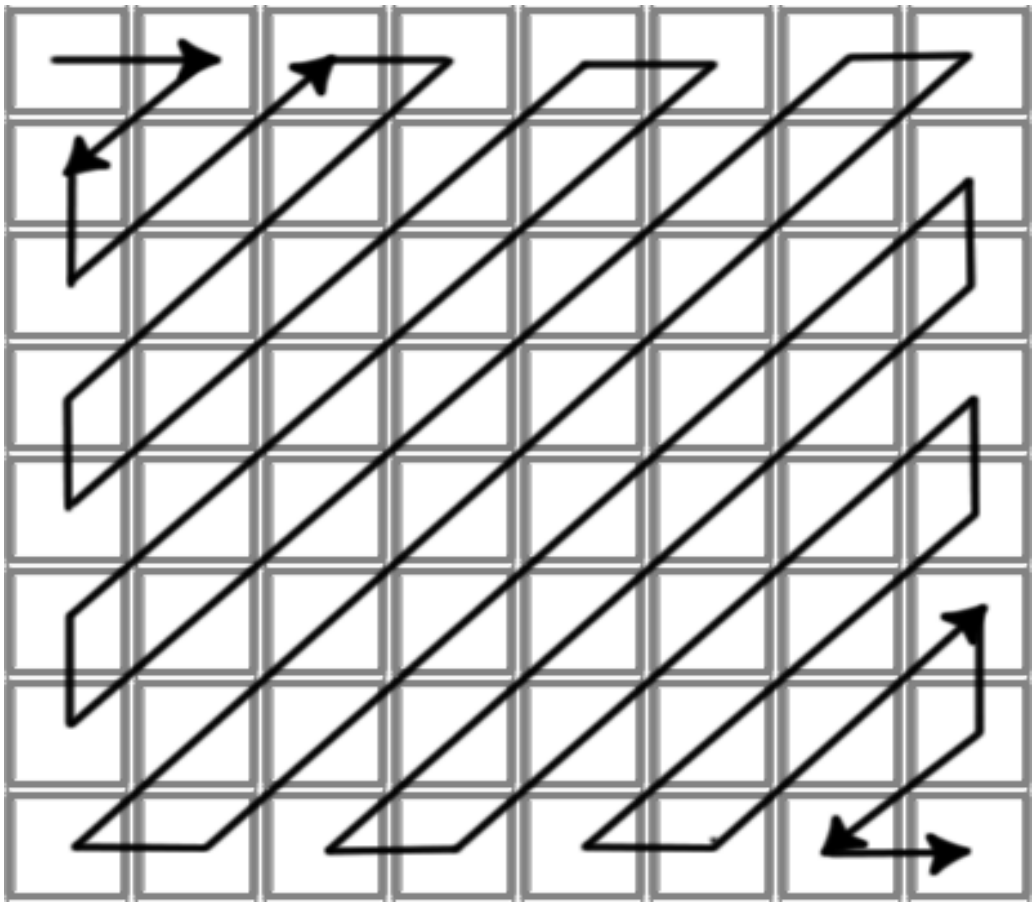

FIGURE 5

"Zigzag" sequence for entropy coding phase.

Bibliography

http://homepages.inf.ed.ac.uk/rbf/HIPR2/fourier.htm

http://www.faqs.org/faqs/jpeg-faq/part1/

http://www.pagetutor.com/image_compression/index.html

http://www.pagetutor.com/about_gifs/index.html

http://en.wikipedia.org/wiki/JPEG#Discrete_cosine_transform

http://blogs.msdn.com/devdev/archive/2006/04/12/575384.aspx