Image Compression Using Transformations

ELEC 539 PROJECT REPORT: PROJECT I

BACH Group
ANDY DORAN and BRENT M. HENDRICKS and CHERRY WANG and HUIPIN ZHANG

Department of Electrical and Computer Engineering
Rice University, Houston, Texas 77251-1892, USA
rabin@rice.edu brentmh@rice.edu ccwang1@rice.edu hpzhang@rice.edu

April 10, 1999

1 Introduction

Image compression is an important technique used to assist in data storage and transmission. The idea is to represent an image with a smaller amount of information while still being able to recover the original image. There are two types of compression, lossless and lossy. Lossless coding tries to take advantage of redundancies in the image data to reduce the data size of the image. The original image can be decompressed perfectly. Lossy compression allows for error between the original image and the decompressed image in order to achieve significant gains in the amount of compression. There are many methods to try to get greater compression ratios while maintaining a small amount of error. In this paper we will explore several methods of transform coding.

Transform coding takes the original image data into a new domain to try to take advantage of the properties of the transform. For instance, if the transform takes correlated image data into a domain where the data is uncorrelated, that might be an advantage for compression the image. We used the Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT) to explore the gains of transform coding. The DCT exploits the energy compaction of the transform to put most of the image information in a small number of bits. Thus, the majority of bits can be quantized to smaller bit sizes in order to gain compression. The Haar DWT localizes sums and differences between adjacent to try to localize the edge information of an image and reduce the bits size of the majority of bits. A higher order Daubechies wavelet transform will try to localize the edges of an image as well.

Once the images were transformed into their new domain, we used three different types of bit allocation algorithms to take advantage of the domain properties. As stated above, the algorithms will assign more bits to the higher energy image components and less to the lower energy components. The individual methods we chose will be explained in sections 2.1, 2.2 and 4.

The objective of this paper is to explore the three compression methods and compare their compression rates (in Bits Per Pixel) to the Picture Signal to Noise Ratio (PSNR) in order to see witch method has the best compression for a given error.

2 Transform coding

2.1 Discrete Cosine Transform

The DCT is fast to calculate and is optimal for images that are highly correlated. Unlike the Fourier Transform, the DCT is all real, which means that for an $N \times N$ image, we will also have $N \times N$ coefficients. Like the Fourier Transform, the DCT provides a 1-to-1 mapping. Therefore, to get back the original image, we simply take the IDCT of the our transformed image. We can find the DCT by using the following formula:

$$F(u,v) = \frac{1}{4}C(u)C(v)\sum_{i=0}^{7}\sum_{i=0}^{7} f(i,j)\cos(\frac{(2i+1)u\pi}{16})\cos(\frac{(2j+1)v\pi}{16}),$$

where $C(0) = 1/\sqrt{2}$ and $C(u) = 1$ for $u \neq 0$. DCT images typically look like the following:

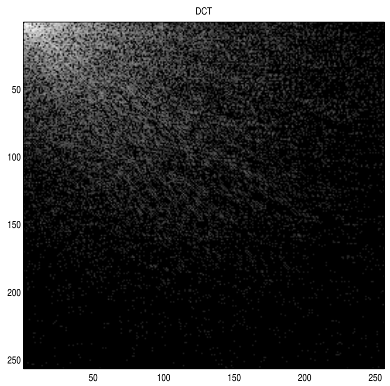


Figure 1: DCT image.

The bright regions correspond to big coefficients and the black regions correspond to small coefficients. If we then divide the image into a number of small blocks, we can allocate bits for each block, where the blocks on the top left corner would be encoded with many bits, and the ones at the bottom right corner would be coded with few or no bits. The allocation of encoding bits is distributed by means of a mask. We divide the DCT into 16x16 blocks and all the coefficients of the block is encoded with the same number of bits.

2.2 Wavelet Transforms

The Haar Wavelet Transform performs sum and difference calculations on the input image data. The actual transform is as follows:

$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

This transform can be applied to adjacent horizontal points (paired off in twos), then adjacent vertical cells. The transform is then run again on the upper left quadrant of the new image. The process can be repeated until you get down to a two-by-two image in the upper left quadrant, although it turns out that a minimum block size of 16-by-16 or 32-by-32 is ideal for compression. The new image is the same size as the previous image, but most of the information is located in the upper left smallest block. This block contains the normalized average of the original image, scaled to a smaller block size. The rest of the image contains a mix of the addition and subtraction information. The idea is that most of the adjacent pixels of the original picture have close to the same value as each other, so storing the difference between them requires a small number of bits. The actual bit allocation algorithm will be discussed in the next section.

Once we have assigned the number of bits we want to represent each pixel, we can use the same transform to invert our data and return to the original image. Note that the Haar wavelet transform is equal to its inverse. You can view an example of a transformed image in the "Sample Images" section of the web page.

3 Quantization

3.1 DCT

Images are typically coded with a maximum of 8 bits. Due to the energy compaction offered by the DCT, the big coefficients are really big and will take more than 8 bits to code. If we look at the raw transform, the largest values are typically in the +/- 20,000 range. That would take 16 bits to encode! What we need to do is to scale down the values so that we can encode them using 8 bits, i.e. values from 0 to 256. But since we have to account for the negative values also, we really only have values in the range of -128 to 127. We cannot just divide the whole transformed image by some number to get that range because we would lose too much information. Too many of the small coefficients would be set to zero. What we need is some sort of a non-linear scale that would bring the extremely high values down while doing relatively little to the small values. In other words, we need to multiply the image by something that looks like this:

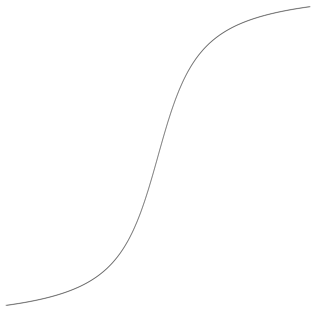


Figure 2: Nonlinear scaling function.

An arctan function or a log function would give us exactly what we want. It has a linear scale close to the center and would cap off the really large values. We decided to use the log (base e) function.

This is our basic algorithm for encoding our images:

(1) Shift the original image so that it is centered around the mean.

(2) DCT the shift image.

(3) Scale the DCT to the range -128 to 127.

(4) Write out the DCT to a binary file. The number of bits we write each value to is determined by our mask.

To decode, we read in our binary file and our mask. The number in the mask corresponds to the number of bits each block was coded in.

By shifting the original image, we also shift the DCT. We do this so that the DCT's magnitude of the lowest and highest values would be at least in the range of the same magnitude, thus allowing us maximum utilization of our -128 to 127 range. The scaling of the values allows us to merely round the DCT instead of doing a complicated quantization. By scaling the DCT, we guarantee that we have 2^8 values.

Our mask is created by looking at some sample DCT's. It is purely ad hoc. Furthermore, we found that the "niceness" of the images corresponded more to how the mask is made rather than to PSNR or to MSE. We were able to code a way to adaptively figure out what the mask needed to be to get lossless compression but that involved sending too much overhead in our binary file.

3.2 DWT

The Haar Wavelet Transform has given us an image with well localized energy. We want to quantize the upper left quadrant with a high number of bits, such as eight, whereas the lower right quadrant can be quantized with one or no bits. For different compression factors, various quantization levels were assigned to the upper right and lower left quadrants.

Depending on the number of bits assigned to quantize a block, a set of sub- intervals was set up between the maximum and minimum value in each block. The higher the number of quantization bits, the larger the number of intervals.

$$I = 2^b, \tag{1}$$

where I is the number of intervals and b is the number of quantization bits for that block. Each interval is assigned a code. Smaller number of intervals (small b) will have smaller codes. Each value in a block is assigned a code based on the interval it falls in. Then the code is mapped to a bit stream. A header containing the maximum and minimum value of each block is also added to the bit stream.

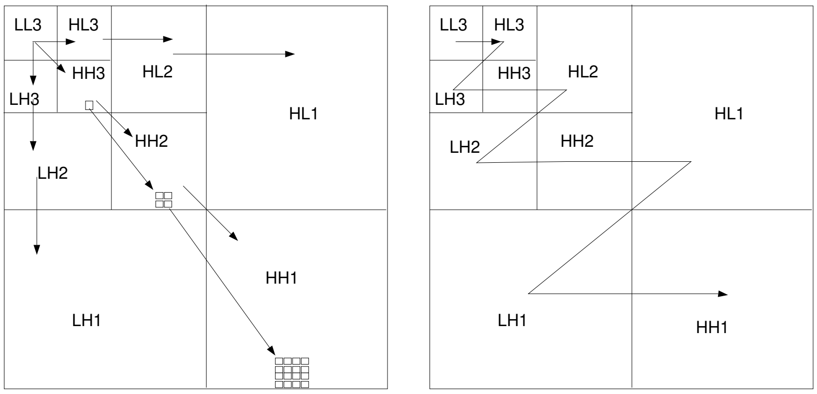
For decoding, we convert the bit stream back to integers. The intervals are recreated using the maximum and minimal values for each block that we saved in our header. Each integer is mapped to its corresponding interval based on the code and assigned the center value of the interval (uniform quantizer). Then the entire image is taken through the inverse Haar Wavelet Transform and checked for it Mean Square Error.

4 Embedded zerotree wavelet (EZW) algorithm

The embedded zerotree wavelet algorithm (EZW) is a simple, yet remarkable effective, image compression algorithm, having the property that the bits in the bit stream are generated in order of importance, yielding a fully embedded code. Using an embedded coding algorithm, an encoder can terminate the encoding at any point thereby allowing a target rate or target distortion metric to be met exactly. Also, given a bit stream, the decoder can cease decoding at any point in the bit stream and still produce exactly the same image that would have been encoded at the bit rate corresponding to the truncated stream. In addition to producing a fully embedded bit stream, EZW consistently produces compression results that are competitive with virtually all known compression algorithms. (See Figures 7 and 6)

4.1 Zerotree data structure

A wavelet coefficient x is said to be insignificant with respect to a given threshold T if $|x| < T$. The zerotree is based on the hypothesis that if a wavelet coefficient at a coarse scale is insignificant with respect to a threshold, then all wavelet coefficients of the same orientation in the same spatial location at the finer scale are likely to be insignificant with respect to the same threshold. More specifically, in a hierarchical subband system, with the exception of the highest frequency subbands, ever coefficient at a given scale can be related to a set of coefficients at the next finer scale of similar orientation. The coefficient at the coarse scale is called the *parent,* and all coefficients corresponding to the same spatial location at the next finer scale of similar orientation are called *children.* Similar, we can define the concepts *descendants* and *ancestors.*The data structure of the zerotree can be visualized in Figure 3. Given a threshold T to determine whether or not a coefficient is significant, a coefficient x is said to be an element of a *zerotree* for the threshold T if itself and all of its descendents are insignificant with respect to the threshold T. Therefore, given a threshold, any wavelet coefficient could be represented in one of the four data types: zerotree root (ZRT), isolated zero (IZ) (it is insignificant but its descendant is not), positive significant (POS) and negative significant (NEG).



(a) Zerotree data structure. (b) Coefficients scaning order

Figure 3: Coefficients are coded in a zerotree structure and scanned in a left-to-right order.

4.2 Dominant pass

Shapiro's algorithm creates rooted trees using a pixel of the LL subband for the root of each tree and a specific order of similarly positioned pixels from the other subbands for children. There are two types of passes performed: a dominant pass and a subordinate pass. The dominant pass finds pixel values above a certain threshold, and the subordinate pass quantizes all significant pixel values found in this and all previous dominant passes previous.

A dominant pass checks all trees for significant pixel values with respect to a certain threshold. The initial threshold is chosen to be one-half of the maximum magnitude of all pixel values. Subsequent dominant pass thresholds are always one-half the previous pass threshold. When an insignificant pixel value is found, and a check of all it's children reveals that they too are insignificant, then it is possible to encode that pixel and all it's children with one symbol, a zerotree root, in place of a symbol for that pixel and a symbol for each of that pixel's children, thus achieving compression. Pixel values found to be significant in the dominant pass are encoded with the symbol positive, for a value greater than zero, or negative, for a value less than zero, then those pixel values are added to a subordinate list for quantization, and the pixel value in the subband is then set to zero for the next dominant pass. Pixel values found to be insignificant in the dominant pass but with significant children are coded as isolated zeros. So, the dominant passes map pixel values to a four symbol alphabet which can then be further encoded by using an adaptive arithmetic coder.

4.3 Subordinate pass

After each dominant pass, a subordinate pass is then performed on the subordinate list which contains all pixel values previously found to be significant. The subordinate pass performs pixel value quantization which achieves compression by telling the decoder with a symbol roughly what the pixel value is instead of exactly what the pixel value is. Since the initial threshold is one-half the maximum magnitude of all pixel values for the first dominant pass, then in the first subordinate pass only two ranges are specified in which a significant pixel value could lie: the upper half of the range between the maximum pixel value and the initial threshold, or the lower half of the same range. A pixel value in the upper half of the range gets coded with the symbol upper (for upper part of the range), while a pixel value in the lower half gets coded with the symbol lower. A pixel value found to be in a particular range is quantized, from the decoders viewpoint, to the midpoint of that range. Upon subsequent subordinate passes the threshold has been cut in half and so there are twice as many ranges as the last subordinate pass plus two new ranges corresponding to the new lower threshold. By reading the subordinate symbol corresponding to a significant pixel and knowing the threshold, the decoder is able to determine the range in which the pixel lies and reconstructs the pixel value to the midpoint of that range. Thus from the decoders viewpoint the rough estimate of a significant pixel's value is getting more refined and accurate as more subordinate passes are made. So, the subordinate passes quantize pixel values to a two symbol alphabet which then get encoded by using an adaptive arithmetic coder as described by Witten, Neal, and Cleary, thus achieving compression.

4.4 Decoding

What is needed for decoding an image compressed by Shapiro's algorithm is the initial threshold, the original image size, the subband decomposition scale and, of course, the encoded bit stream. The decoder then decompresses the arithmetically encoded files into symbol files, creates all the proper size subbands needed since it knows the subband decomposition scale and the original image size, and proceeds to undo the Shapiro compression since it knows the initial threshold and the subband scanning order.

4.5 An example

We use a two scale 4 wavelet image as a simple to show the algorithm. The original image is shown in (a) of Figure 4.

18	3	2	2
6	5	1	-2
8	13	-6	4
-7	1	3	-2

(a) Initial image

0	3	2	2
6	5	1	-2
8	13	-6	4
-7	1	3	-2

(b) After DP 1

0	3	2	2
6	5	1	-2
0	0	-6	4
-7	1	3	-2

(c) After DP 2

0	3	2	2
0	0	1	-2
0	0	0	0
0	1	3	-2

(d) After DP 3

Figure 4: Example: ezw.

The procedure of encoding together decoding for the image is listed as follows.

Dominant pass 1. The threshold $T = 16$, and the quantization value is $1.5T = 24$. The output symbols are POS(18), ZTR(3), ZTR(6) and ZTR(5). The reconstruction values is 24, corresponding to the only positive significant coefficient 18. The subordinate list contains just one number 18. The coefficient maintained in the subordinate list is replaced by zero in the image, and the image is shown in (b) Figure 4.

Subordinate pass 1. One symbol is generated corresponding to the only coefficient (18) in the subordinate list, the symbol is 1, because 18 is larger than the threshold 16.

Dominant pass 2. The threshold is halved, i.e., $T = 8$, and the quantization value is $1.5T = 12$. The output symbols are ZTR(3), IZ(6),ZTR(5), POS(8), POS(13), ZTR(-7) and ZTR(1). The reconstruction values is 24, 12, 12, corresponding to the numbers 18, 8, 13 in the subordinate list, and then the coefficients maintained in the subordinate list are replaced by zero in the image, and the image is shown in (c) Figure 4.

Subordinate pass 2. Three symbols are generated corresponding to the coefficients 18, 8, 13 in the subordinate list. The output symbols are 0,1 and 1.

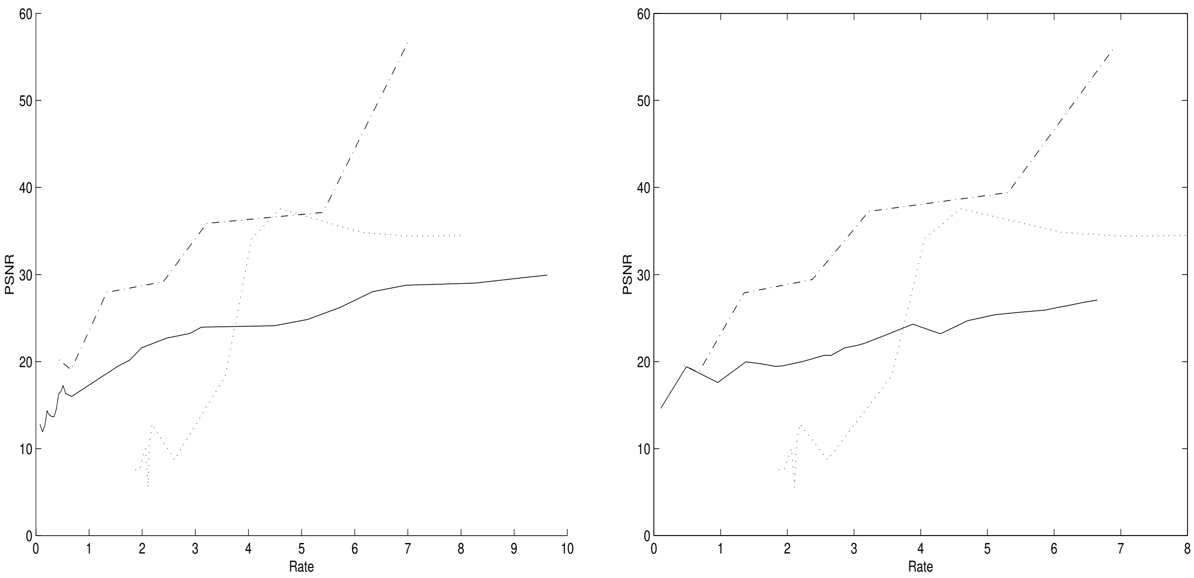
Dominant pass 3. The threshold is halved again, i.e., $T = 4$, and the quantization value is $1.5T = 6$. The output symbols are ZTR(3), POS(6),NEG(-5), NEG(-7), ZTR(1), NEG(-6), POS(4),ZTR(3) and and ZTR(-2). The reconstruction values is 24, 12, 12,6, 6, -6, -6, 6, corresponding to the numbers 18, 8, 13, 6, -5,-7,-6, 4 in the subordinate list, and then the coefficients maintained in the subordinate list are replaced by zero in the image, and the image is shown in (c) Figure 4.

Subordinate pass 3. Eight symbols are generated corresponding to the coefficient 18, 8, 13, 6, -5,-7,-6, 4 in the subordinate list. The output symbols are -1, 1, 1, 1, 1, 1, 1 and 1.

5 Experimental Results

To test our three algorithms we try to compress the standard images *Lenna* and *Clown.* (See Figures 7 and 6). For the DCT based compression algorithm we determined values for the PSNR of the compressed images for various compression rates in order to measure the performance of our coders. In order to do this, we had to generate a bit allocation mask for each desired compression rate. For the case of the wavelet transformation we developed a function (mask.m) to create masks at the compression rates of: 1, 75, .675, .5, .25, .125, and .0625. We used a slightly more ad hoc method for developing masks in the case of the DCT. Also, for the Haar DWT, we stopped transforming the input image at a minimum block size of 16. We then measured the MSE of the compressed images, and calculated the PSNR from that. We based The bits-per-pixel for each image on the file size of the gzipped compressed image. The results of this experiment performed on both the lenna image and the clown image can be seen in Figure 5.

For EZW algorithm the quantization is implicitly implemented after each subordinate pass, so the quantization table is actually related to the size of the bit stream for reconstruction. To do the experiment we first generate a bit stream which can be used for perfect reconstruction of the image. Then based on our desired compression ratio we can determine how many bits we need for reconstruction. Then we truncate it to the right size, and the image is reconstructed based on the truncated bits. Since the EZW is a progressive coding algorithm, the bit stream of any size will construct an approximation of the original image.

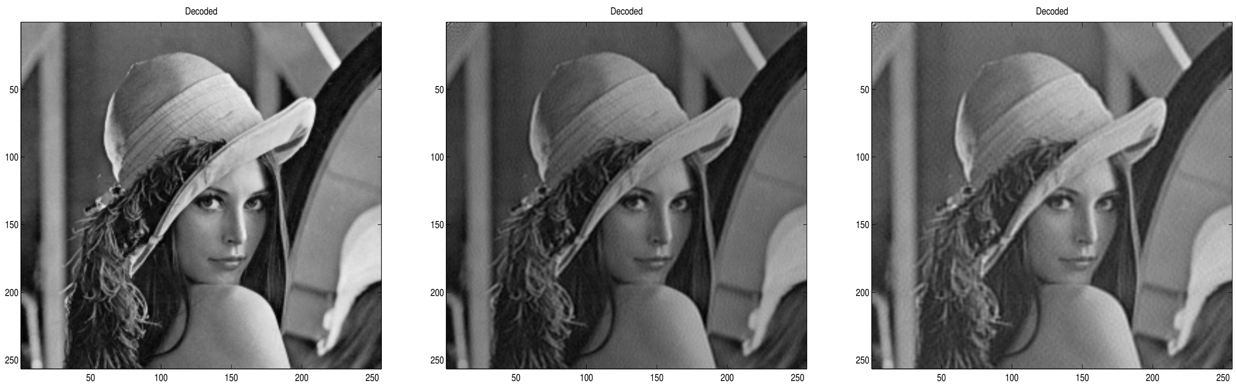


(a) Lenna (b) Clown

Figure 5: PSNR vs bite rate for *Lenna* and *Clown.* The solid curve is obtained using EZW algorithm, the dotted curve is obtained using DCT transform and the dashdot curve is obtained using wavelet transform together with specifically designed bit map.

6 Conclusions

From the curves we can see that in general with higher average rate for each pixel, the PSNR of the reconstructed image gets larger. This makes sense since when we represent the images with more bits, the images are better reconstructed in terms of PSNR and the visual performance as well. (See Figure 7). In our experiments this fact is proved by our three different algorithms. However, we still can see some jumps in the curves. The main reason for this is that using more bits does not necessarily mean a better reconstruction, since reconstruction is highly related to *how* we quantize our transformed data to get our bit stream. If we design a wrong quantization table, such as assigning more bits for small coefficients and less bits for large coefficients, then the reconstruction will be very much dependent the length of the bits stream. Although we believe that most of the time this design of the quantization tables is reasonable, the quantization mask does not necessarily match the block information for every block to be encoded. For EZW algorithm, we have a similar explanation. For example, if we uniformly quantize 15 in the range of 0 and 32 with one quantization level, then the reconstruction value is 16, which is a good approximation. If we quantize it in the same range with two level quantization, a better reconstruction value should be obtained, however, what we get is 8, which is an even worse approximation. This is the reason why in the EZW algorithm with more bits, meaning more level quantization, we don't necessarily get better reconstruction.



(a) bpp=3.5224, psnr = 34.5146

(b) bpp = 0.6276, psnr = 25.5008

(c) bpp = 0.4002, psnr = 16.1247

Figure 6: Reconstructed images and the performance.

Note that best transform method over almost all rates was the Haar Wavelet transform. This was extremely surprising given the ad hoc method used to determine the quantization size for each block. However, we believe that this is due to the storage of the minimum and maximum values of each block for decompression. In this way we were able to keep our interval size as small as possible for the cost of a bit of overhead. Thus, the uniform reconstruction interpolator was able to better approximate the original transformed values. This is especially important when you consider the smallest block in the upper left quadrant. We stated above that accuracy in this block is extremely valuable due to the high amount of information stored there. Moreover, unlike all the other blocks, the elements in this block can never be negative. So to assign a uniform maximum and minimum

(a) bpp=0.4706,psnr = 16.5532

(b) bpp = 4.4824,psnr = 24.1155

(c) bpp = 6.3365,psnr = 28.0311

Figure 7: Reconstructed images and the performance.

value to all blocks at this quantization level would end up placing negative values in the upper left block. This would give tremendous error, as was seen during experimentation. So the tighter bounds we get with stored block minima and maxima give much lower MSE for a small cost to the Rate. Notice, however, that as the compression gets very large, the overhead will add a lot to the MSE.

As for the DCT, MSE increased rapidly as the BPP decreased. However, we managed to keep the edges of the image at all times. What degraded was not a blurring of the edges, but an overall contrast loss. As the BPP decreased, the image continuously lost the dark colors it originally had. So at very high compression, we could still tell what the image is but it seemed to be covered by a layer of fog. Something else we noticed was that a high MSE does not necessarily correspond to a bad image. We actually saw this with all three compression methods. The quality of the image was related more to the mask placement than BPP. This suggests that adaptively creating the mask as we encode the image would give us better quality.

References

[1] M. Pennebaker and J. Mitchell, JPEG Still Image Data Compression Standard, Van Nostrand Reinhold, New York, 1993.

[2] J. Shapiro, Embedded image coding using zerotree of wavelet coefficients, IEEE Trans. Signal Proc., vol.41, no. 12, pp. 3445-3462, Dec. 1993.