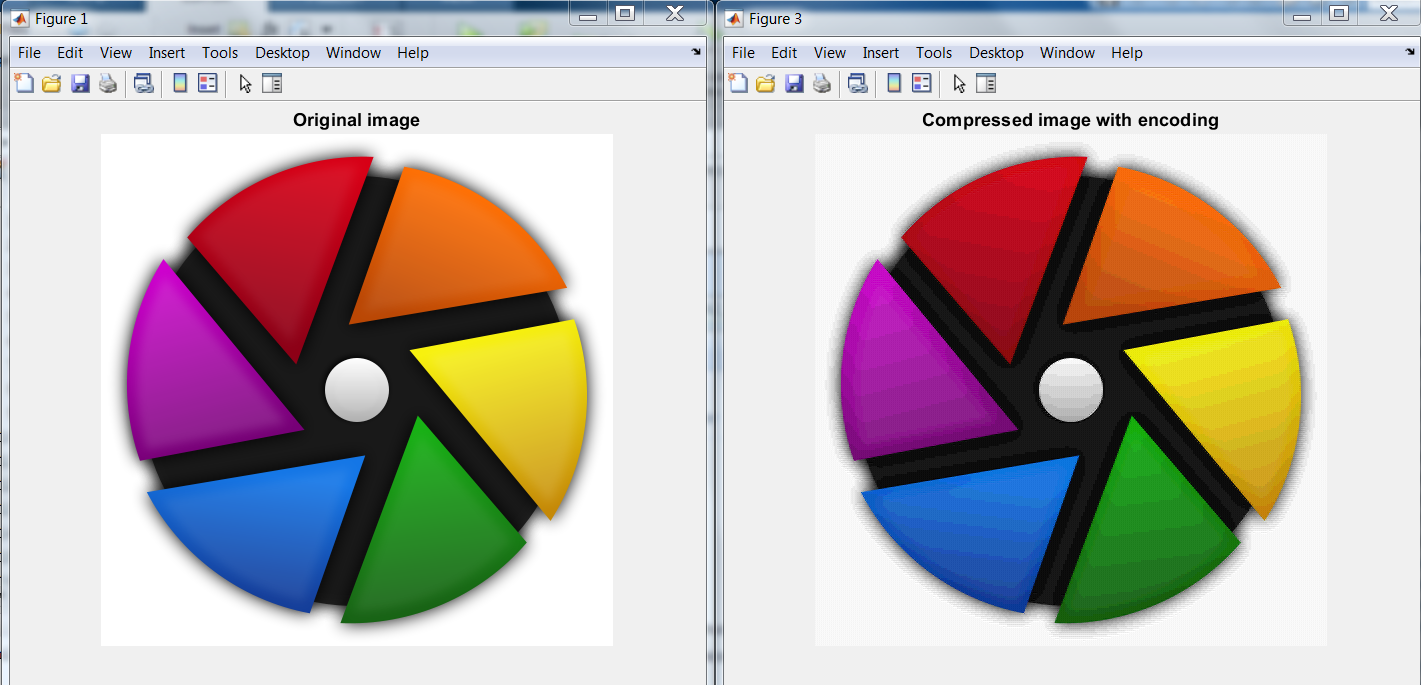
Bradley Clemente

Ryan O’Shea

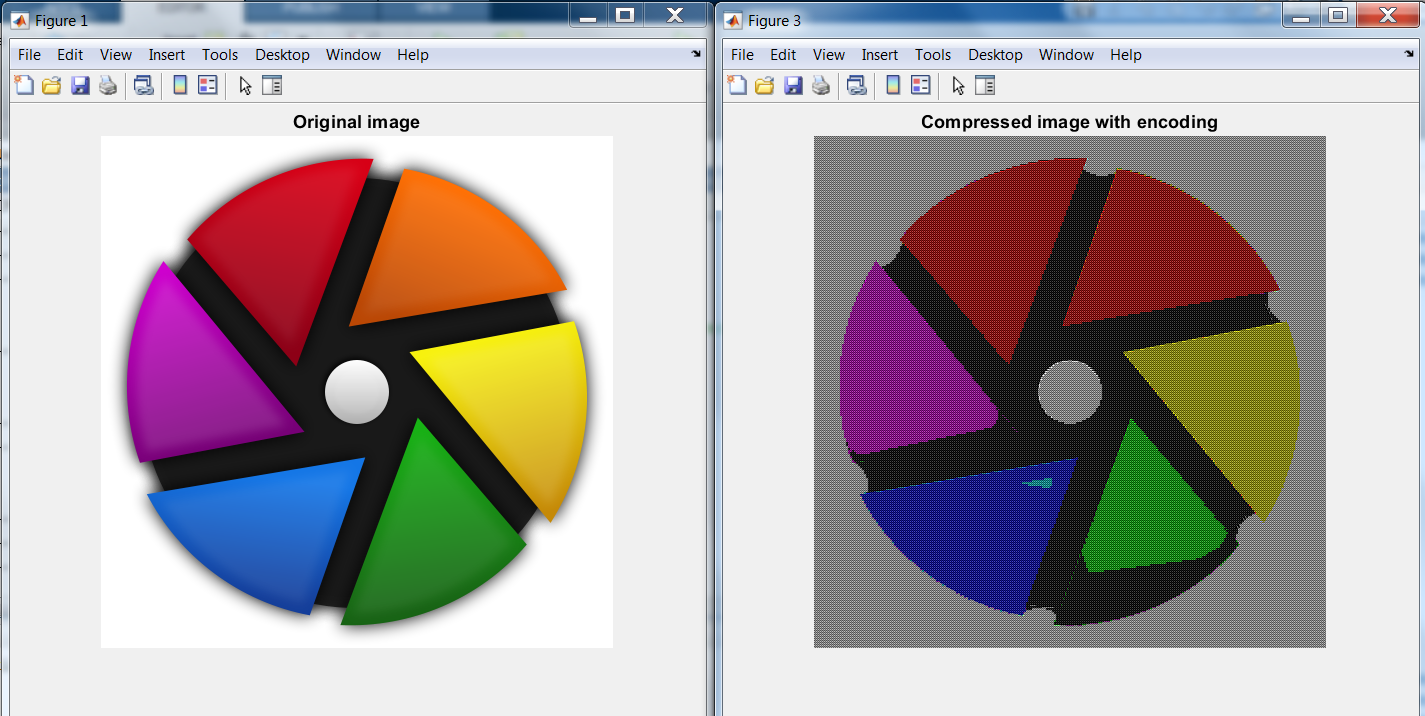
**Image Compression Using Wavelet Image Coding**

**What was accomplished:**

The group was able to create a MATLAB script that takes in a PNG file, prompts the user for desired quantization level, and then outputs a compressed version of the input image. The picture below shows the result of running the script on an image after getting a high number of quantization levels input from the user. The image was compressed from 104KB down to 75KB with only a minimal loss in quality. The loss in quality is seen mostly in the rough color transitions in the six different colored areas in the output image.

****

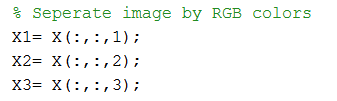
The picture below shows the results of running the script with a low number quantization levels input from the user. The output image has obviously lost a large amount of quality but the image was compressed from 104KB down to only 12KB. In terms of preserving quality this level of compression is impractical but the reduction in file size was incredibly substantial.



The compression script was found to be most effective when used on PNG files that did not have a background. The script also only worked on PNG files and would actually increase the size of the file when used on JPEG files.

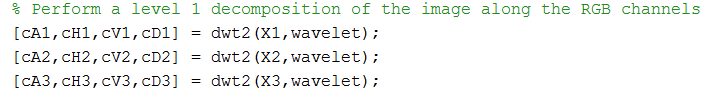
**Code explanation:**

The image compression script starts by first splitting the image to be compressed into its three different color channels as seen below. X is the input image.



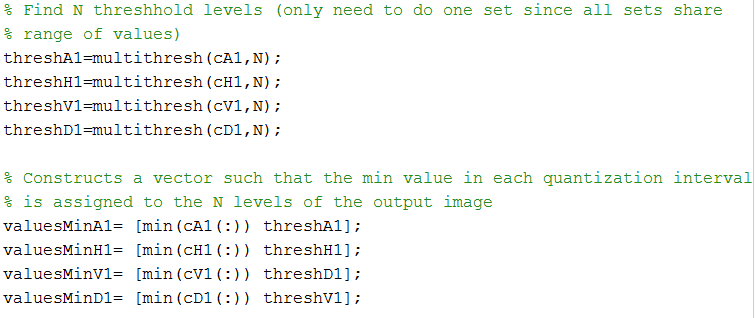
The image is ﬁltered with a low-pass and a high-pass ﬁlter, both horizontally and vertically. The four ﬁltered versions of the image are then subsampled. So, if the original image is of size 600 × 800 pixels, after one step of the decomposition we have 4 images of size 300 × 400. The ﬁlters are then applied recursively on the low-pass image. Following this, the three different color channels were run through the discrete wavelet transform function in order to separate each of them into their four components as seen below. The four produced channels are approximation (A), horizontal (H), vertical (V), and diagonal (D). The MATLAB function to do this subband decomposition is called dwt2 which performs a single-level discrete 2-D wavelet transform on the given image. As arguments, the function takes an image, and what type of wavelet to use. For our project we use the Daubechies family which is an orthogonal wavelet.

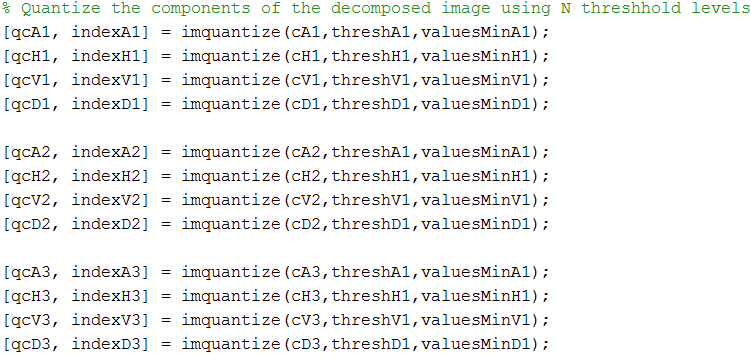




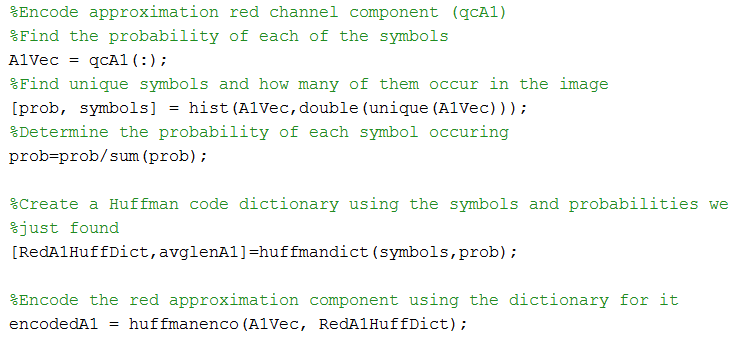
Next the subbands are quantized using imquantize which quantizes an image using specified quantization values contained in the N element vector levels. First the image is split into N+1 levels by obtaining N thresholds from the function multithresh. These levels are found by thresholding the four subbands using Otsu’s method. Then we construct the valuesMin vector such that the minimum value in each quantization interval is assigned to N levels of the output image. The threshold levels and their minimum values are given to the imquantize function for each color channel’s subbands to produce 12 quantized subbands.



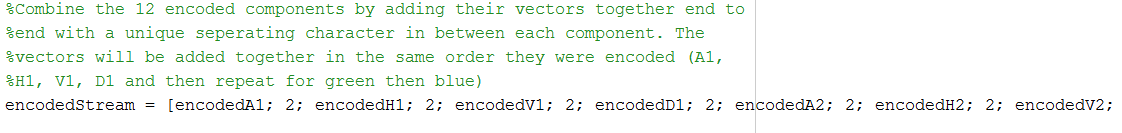




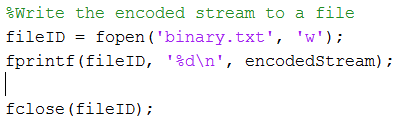
After the twelve components have been quantized they will be encoded using a Huffman encoder to minimize the total amount of data needed to represent them. In order to do this a component was first turned into a vector and then had its unique values and their probability of occurring calculated. The unique values and their probabilities were then used to create a Huffman dictionary for that specific component using the huffmandict() function. After the dictionary is created it is then used to encode the component using the huffmanenco() function. The series of operations used to encode one component can be seen below.



This process is repeated for all twelve components until all components have been encoded into their own unique variables. After all, twelve components have been encoded they are combined into one single vector so it can more easily be written to a file. In between each of the components a 2 is placed as an “End of Block” character to signify to the decoder that it has reached the end of encoded data for a component.



After all of the twelve different components were encoded and combined into one vector they were written to a text file named “binary.txt” using the frintf() function as seen below.

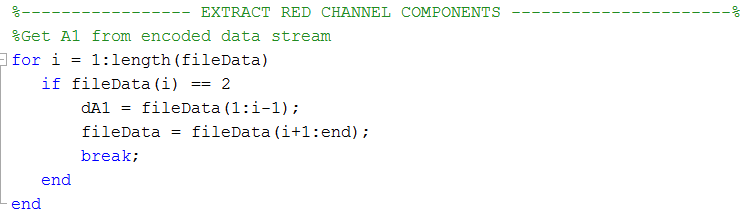


Using a text file with data stored as characters caused the encoded data to be quite a bit larger than the original image file which is a major area where the code could be improved. We tried to store the encoded stream as MATLAB logical type in a file but we were unable to find an associated function that could properly read the data back out of the file and into a MATLAB variable. There could have also been an issue with writing logical to the file itself from MATLAB but at the moment we are unsure where the actual issue was occurring.

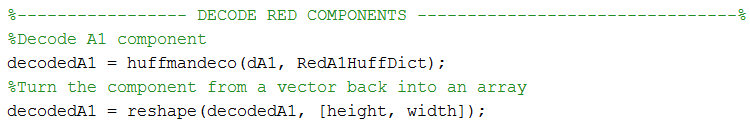
After storing the encoded bitstream in a file the job of the encoder was done and now the data needed to be decoded and reformed into its components so the full image compression process could be completed. In order to do this the data first needed to be imported in from the bitstream file as seen below.



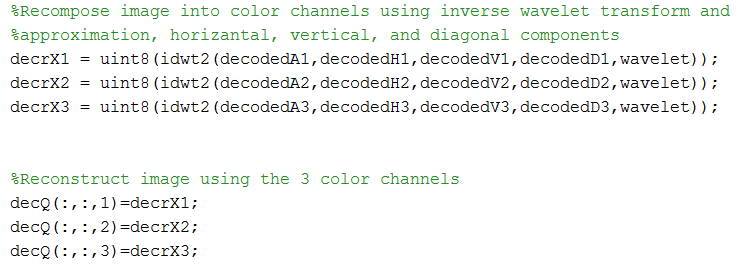
After importing the data it needed to be segmented back into the individual bitstreams for each of the twelve different decomposed components. To do this, fileData was looped through until the designated end of block character was reached which as previously mentioned was 2.



Once the end of block character was reached all data in the vector up to the that point would be stored in a new variable for that component. The fileData variable would then be updated to only contain the data after the end of block character that was just hit. The same “scan for character and then extract component segment” process was repeated until all twelve encoded components were extracted into their respective variables. The final encoded component written to the file, the diagonal component of the blue channel (dD3), does not have an end of block character after it so once the decoder has extracted the second to last component it will just assign the remained of the file to the dD3 variable. The twelve encoded components were written to the file in a specific order and are then extracted from the file in the exact same order in order to assure proper decoding. After all of the encoded components have been extracted from the file they are all decoded one by one using the same Huffman Dictionary that was used to encode them as seen below.

****

After decoding the component is still in vector form so to fix this it was reshaped to into the same dimensions that it started as. The height and width were calculated using the size() function on the components at the beginning of the encoder code. The dimensions for all of the components are the same so the same height and width can be used when decoding and reshaping all of the different components. The decoding and reshaping process is applied to all of the twelve different components which are stored in their respective variables to be used for recomposing the image. The twelve components are then used to reconstruct their respective color channels through the use of the inverse discrete wavelet transform function idwt2().



The 3 color channels are then added back into a full image which is then displayed in order to be compared to the original pre-compression input image.

To measure the distortion of images we use PSNR (Peak-to-peak Signal to Noise Ratio) that is in decibels. From the images we tested they fall into the 25-30dB range when N=20 for the thresholding levels. This result is significantly reduced as fewer threshold levels are used. The formula used by matlab is shown below where peakval is taken from the range of the image datatype (e.g. for uint8 image it is 255). MSE is the mean square error, i.e. MSE between the output image and the input image.





**Contributions:**

Bradley worked on the subband decomposition of the RGB image, as well as the thresholding, minimum values in those threshold levels. and quantization of the subbands using those levels. He also implemented the PSNR function to calculate the performance of the subband encoding script.

Ryan worked primarily on the Huffman coding, bitstream creation, file input and output, and recomposition of the compressed image. To accomplish this, he used a number of functions and features from the following toolboxes: Communications, Digital Signal Processing, Wavelet, Signal Processing, and Image Processing.