**Assignment 2**

**Coding/Algorithm Logic and Progression:**

The mentality behind the coding was to break each major step into its own function using Java and starting with the given sample code as a base. There is a separate function for doing the majority of the calls and rasterizing based on which button is selected from the starting panel. Once a scaling factor is selected, the program breaks down the input RGB image into 3 separate arrays for Y, U, and V which are then made into 2D arrays (height x width) from 1D. The Y array has the DCT function applied to it (which subtracts 128 from each value as well), followed by being quantized using the luminance quantization table and scaling factor. Based on this returned array it now has IDCT applied to it after being restored using the luminance quantization table and our final Y array is complete.

The process for our chrominance values U and V has chroma subsampling of 4:2:0 applied during this process. The original U and V arrays are split into 4x4 blocks that take the average U and V values. This results in an array that is half the height of the original image and half the width. Then DCT, chrominance quantization, and IDCT are applied the same as previously for the Y array. Once the U and V arrays are complete, it is necessary to expand them back to the original height and width values to match the input image. The values that were taken out are found by using interpolation (adding the two known points beside each unknown point and dividing by two).

Once the above steps are done, the final Y, U, and V arrays are translated back into 1D and then converted into an array compatible for RBG rasterizing (R1,G1,B1,R2,G2,B2,etc…).

**Theory:**

The main goal of this assignment is to use DCT/IDCT and quantization techniques to allow for JPEG image compression. These methods help reduce spatial redundancy and take advantage of human visual acuity.

When applying DCT, the higher the compression ratio, the choppier the image appears because of isolation of each 8x8 block in our image. The next step of quantization is the main source for loss in this process of JPEG compression. The entries of the luminance and chrominance quantization tables have larger values in the bottom right corner to introduce more loss at the higher spatial frequencies.

Chroma subsampling is used to encode images by using a lower resolution for chroma information (U and V) than for luma (Y) since the human visual system has a lower acuity for color differences than for luminance. By skipping every other pixel when applying quantization, we can reduce the image size while maintaining almost identical visible image quality.

Based on theory, applying a higher scaling factor will result in a blockier and choppy image and applying a lower scaling factor will result in a more fine and detailed image.

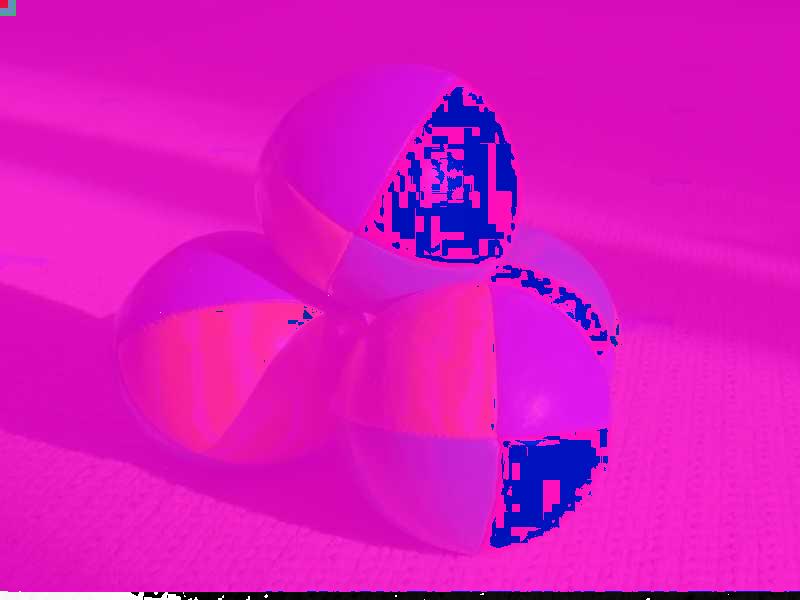
**Results/Remarks:**

The results from the Java program follow what should be occurring theoretically (1 outlier). When a larger scaling factor was used, the result was a less detailed image as well as a smaller file size and vice versa. This is because more data (pixel data) was lost when performing compression

Original File: balls.JPG (144 KB)



Scaling Factor 5 (26 KB):



Scaling Factor 0.20 (195 KB):



The table below shows the resulting file sizes based on scaling factor:

|  |  |
| --- | --- |
| Scaling Factor | Resulting File Size |
| 5 | 26 KB |
| 4 | 29 KB |
| 2 | 41 KB |
| 1 | 69 KB |
| 0.75 | 115 KB |
| 0.50 | 200 KB |
| 0.20 | 195 KB |

**Obstacles/Bugs/Notes/Completion:**

***Completed:***

* RGB to YUV conversion
* 1D to 2D array conversion
* 4:2:0 Chroma subsampling
* DCT implementation method
* Quantization implementation method using luminance and chrominance tables
* IDCT implementation method
* 2D to 1D array conversion
* YUV to RGB conversion (attempted but resulted in off colors)
* Buttons to display only Y, only U, and only V of original image
* Buttons to apply DCT/Quantization/Scaling/IDCT/RGB conversion to image
* Buttons to display only the Y values when using high scaling value and low scaling value. This was included to show that the scaling works correctly and the programs problem lies in conversion/rasterization back into RGB

The one major bug in the program is the conversion back into RGB. When the program was tested with the “Lena” sample 8x8 block it returned all the correct values, but there is a bug in the program afterwards when rasterizing in RGB which leads to the incorrect colors. Despite this bug it is apparent that a lower scaling factor produced much higher quality pixels since the color variation is extremely apparent, versus the higher scaling factor image which produced fewer color differences (despite being the wrong colors). This means that the blocks next to each other were less similar when a lower scaling factor was used, and gave more visible results to the human eye.

**Conclusion:**

Based on the results of the coded program, it is correct to state that a higher scaling factor will result in a worse image quality and smaller file size. The opposite is true for a low scaling factor, which produces higher quality results at the cost of a larger file size. In essence our program results match the theoretical results (despite off-coloring) taught in class!