CSI 394
Week 3: Wavelets!

1. From last week, we have code to read in a PPM and average side by side elements.

2. Extend our code from last week to also average elements that are on top of one another. Thus, if our initial image was 256x256, it will now be 128x128. Last week, it went from 256x256 to 128x256.

3. Output our new image to ppm to check it out. This should now look like a smaller copy or thumbnail of your original image. We can try out our three sampling methods to see which one is best. One might call this a low-pass filter. This is essentially what we are implementing. Of course, our filter is really simple. It averages neighboring pixels.

4. Make sure our code can read in our new 128x128 image and average it into a 64x64 image and output it. This might require some changes in code if we were not writing things generally in the past.

5. Now, in this new version, let's keep our averages as floating point values AND keep around the coefficients after the averages. We'll store them as a kind of separate image on the side. Some sample numbers as we go through the process are listed below.

6. Original Image:

```
255 255 255     255   0   0     0  255  0     0   0  255
255 255 255     255 255 255   255 255 255   255 255 255
255   0   0       0  255  0     0   0  255     0   0   0
 0    0   0       0   0   0     0   0   0      0   0   0
```

7. Averaged Width with offsets on the far right:

```
255 127.5 127.5 0  127.5  127.5  0 127.5 127.5  0 127.5 -127.5
255 255   255   255 255    255    0  0     0     0   0  0
127.5 127.5  0  0   0   127.5   127.5  -127.5 0  0   0 127.5
 0     0     0  0   0      0      0       0   0  0   0  0
```

8. Averaged Height (after Averaged Width):

```
255   191.25  191.25  127.5  191.25  191.25
                      ...   0     63.75   63.75   0 63.75 -63.75
63.75  63.75    0      0      0      63.75
                      ... 63.75  -63.75     0     0   0    63.75

0     -63.75 -63.75   -127.5  -63.75  -63.75
                      ...   0   63.75   63.75     0 63.75 -63.75
63.75  63.75    0          0     0     63.75
                      ... 63.75 -63.75      0     0   0    63.75
```

9. Provide a mechanism to output this as a special file. Let's use a magic number on the first line that is simply RWV1. We'll later introduce RWV0, RWV2 and more. The 1 is going to refer to the fact that the averaging step happened only once. The second line is width and height and a third line that is "maxval" just like PPM uses. Then we'll just output all the values as averaged and subtracted. Let's call this file .rwv and we'll call them "R WAVES" and "R WAVE N" in conversation.

10. Provide a mechanism that outputs this RWV format to a PPM in the following special way. For the normal image file, output the values as usual. For the subtraction area, simply output the absolute value of those numbers times two. (As an alternative, we could output the subtraction area as 128 plus their values.) Since this is not to be used for reconstruction, we will use this to get a sense of what this image might look like.

11. But, we would like to take this at least one step further and produce other RWV formats like RWV2 and RWV3. These will require us to average our rows and columns "N" times according to the number behind the RWV. The RWV0 should just be a reproduction of a PPM. Whereas, RWV1 is what is listed in the rest of this handout. RWV2, however, will take the numbers in #8 and average the new averages to get new averages and offsets. So it will not be required to operate on all 4 colors in each row and column, but rather it is only required to operate on the 2 new colors (6 values total) in the upper left of the RWV1 that was produced. In our 256x256 example, RWV1 has averages in the 128x128 region and differences throughout the rest. RWV2 then would average that 128x128 region to produce a 64x64 region and differences throughout the rest. RWV3 produces an averaged region that is 32x32 and RWV4 produces a 16x16 averaged region. RWV8 is as far as we can push this process for a 256x256 region.

12. Write a program to produce an RWVN from a ppm and vice versa. The N should be considered user input to the program when going from PPM to RWVN.

13. One last thing! The whole point of this process is allow for some compression of our images. Write some code that produces an RWVN file with more entries of 0 and yet produces an original PPM (now with some loss) that appears similar to the original. Here are some hints to get you started. First, pick a threshold that treats values near 0 as 0. Values that are in the x ¿ width/2 or y ¿ width/2 are providing subtraction and addition details that are less important than those that might appear elsewhere in an RWV8. Perhaps decide to "zero" out some of these details more aggressively in that particular region. A good idea might be to use a $1/2^j$ multiplier based on which averaging iteration the values occurred in. Discuss your choices and results in your github!

14. You should have a suite of python programs at this point. They should:

    (a) Read in an existing PPM and produce an RWVN which contains all the information required to reproduce the PPM without loss. The value of the N can be left to user input or command line parameters.

    (b) Read in an existing RWVN and produce the original PPM without loss.

(c) Read in an existing RWVN and produce a viewable version of the RWVN as a PPM.

(d) Read in an existing RWVN and output a new RWVN with more zeroes (which introduces some loss).