CS 576 – Assignment 1
Instructor: Parag Havaldar

Assigned on Mon Jan 24, 2022,
Solutions due on Mon Feb 14, 2022 - by 4:00 pm afternoon
Late policies - None

Theory Part (20 points)

Question 1 (*5 points*)
The following sequence of real numbers has been obtained sampling an audio signal: 1.8, 2.2, 2.2, 3.2, 3.3, 3.3, 2.5, 2.8, 2.8, 2.8, 1.5, 1.0, 1.2, 1.2, 1.8, 2.2, 2.2, 2.2, 1.9, 2.3, 1.2, 0.2, -1.2, -1.2, -1.7, -1.1, -2.2, -1.5, -1.5, -0.7, 0.1, 0.9 Quantize this sequence by dividing the interval [-4, 4] into 32 uniformly distributed levels (place the level 0 at -3.75, the level 1 at -3.5, and so on. This should simplify your calculations).

- Write down the quantized sequence. *(4 points)*
- How many bits do you need to transmit it? *(1 points)*

Question 2 (*5 points*)
Suppose a camera has 450 lines per frame, 520 pixels per line, and 25 Hz frame rate. The color sub sampling scheme is 4:2:0, and the pixel aspect ratio is 16:9. The camera uses interlaced scanning, and each sample of Y, Cr, Cb is quantized with 8 bits

- What is the bit-rate produced by the camera? *(2 points)*
- Suppose we want to store the video signal on a hard disk, and, in order to save space, the signal is re-quantized so that each channel (Y, Cr, Cb) uses 6 bits . What is the minimum size of the hard disk required to store 10 minutes of video *(3 points)*

Question 3 (*10 points*)
Suppose you are recording a rotating wheel with a video camera which records at 25 frames per second. The wheel has a white mark on it to gauge the speed of rotation.

- If the speed of rotation is 20 rotations per second, what is the observed speed (rotations/sec) of rotation? *(5 points)*
- What is the observed speed (rotations/sec) of rotation if the actual speed of rotation is 10 rotations per second? *(5 points)*

Programming Part (*130 points*)

This assignment will help you gain a practical understanding of *Resampling*, *Quantization* and *Filtering* to understand how these processes affect visual media types like images and video. The motivation here comes from the fact that high valued content is created/saved as a master copy with a high resolution in both samples and color. This master copy may then be appropriately scaled down for distribution depending on the platform. The conversion process needs to ensure that the final product is of the highest quality possible given the constraints. Eg For Digital Cinema, frames are mastered at 4K resolution (4096x2160 pixels) with each pixel at 12 bits per color channel (total 36 bits). For distribution via blue ray/DVD, the image frame may be down sampled to HD format (1920x1080 pixels) with each pixel at 8 bits per channel (total 24 bits). In this assignment we will not use such high resolution but will be starting with moderately high 512x512 images at 24-bit color depth and then further resample/requantize.

First, you need to know how to display images in the RGB format. We have provided sample code to read and display an image in Microsoft Visual C++ and java. This source has been provided as a reference for students who may not know how to read and display images. You are free to use this as a start or write your own in a language/IDE of your choice as long as we can appropriately evaluate it in the environment that USC provides. *However, you may **not** make use of any libraries, nor use scripting environments like python or matlab!*

The input to your program will take four parameters as explained below:

- The first parameter is the name of the image, which will be provided in an 8 bit per channel RGB format (Total 24 bits per pixel). You may assume that all images will be of the same size for this assignment of size 512 x 512. The .rgb file format stores the image in a particular order – first comes the red channel in scan line order, followed by the green and blue channels. Please refer to the reference code if needed.
- The second parameter will be a ***S***cale value. This will control by how much the image has to be scaled. It will be a floating-point number greater than 0.0 but less than or equal to 1.0. Effectively, this will result in down sampling your image.
- The third parameter will be a ***Q***uantization number giving the number of bits each channel needs to be quantized down to. The initial rgb file has 8 bits per channel So this Q value will be a value such that 1<= Q <= 8
- The fourth parameter will suggest a ***M***ode for quantization. Its value will be an integer. A -1 indicates uniform quantization (explained below) or a whole valued number (between 0 – 255) suggests a pivot for smarter logarithmic quantization (explained below) where the number represents the pivot.

To invoke your program, we will compile it and run it at the command line as

YourProgram.exe C:/myDir/myImage.rgb S Q M

where *S Q M* are the parameters as described above. Example invocations and their expected outputs are explained below which should give you a fair idea about what your input parameters do and how your program will be tested.

1. *YourProgram.exe myImage.rgb 1.0 8 -1*

Here the image will be scaled by 1.0, with the output quantized at 8 bits per channel and use uniform quantization. This effectively means that your output will be the same as your input. Note that if Q=8, then this is the same quantization as the input and there is no need to re quantize, so the last parameter has no meaning in this particular case

2. *YourProgram.exe myImage.rgb 0.5 8 -1*

Here the image will be scaled by 0.5, with out at 8 bits per channel. This is the same as the previous case except that the image is scaled down to half it size in each dimension.

3. *YourProgram.exe myImage.rgb 0.8 4 -1*

Here the image will be scaled by 0.8, each channel is being quantized to 4 bits (so a max of 16 values per channel) with uniform quantization. For display reason you will need to map the 16 values to an 8 bit scale (for each channel). For uniform quantization you will need to choose 16 appropriate values in the 0-255 range eg 8,24,40,56…etc. which correspond to the uniform ranges 0-15, 16-31, 32-47, 48-63 … etc.

4. *YourProgram.exe myImage.rgb 0.75 3 0*

Here the image will be scaled by 0.75, each channel is being quantized to 3 bits (so a max of 8 values per channel) with logarithmic quantization. The logarithmic quantization starts at 0. For display reason you will need to map the 8 value to an 8 bit logarithmic

scale with appropriate values in the 0-255 range eg 3, 9, 18, 27, 39, 55, 80, 176 which are the center of the logarithmic ranges T.

Details of implementation:
Every pixel has an RGB value and a *(x, y)* location. You will need to compute the new location of the pixel *(x_new, y_new)* and copy its *filtered* (or *anti-aliased*) RGB value to the new location. The filtered value depends on the filter kernel used and very many kernels are possible. A common practice is to use a uniform 3x3 averaging kernel shown below.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Average Filter Kernel

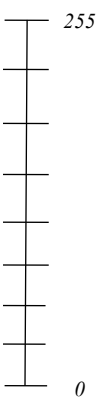
123	133	136
125	135	139
128	138	142

3x3 window at a location

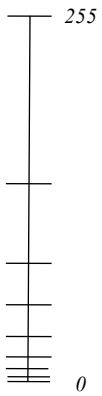
The unfiltered lookup value of the center pixel is 135.
The *average filtered* lookup value of the center pixel is computed using the kernel as
1/9*123 + 1/9*133 + 1/9*136 + 1/9*125 + 1/9*135+ 1/9*139 + 1/9*128+ 1/9*138+1/9*142 = 133.222
You will need to round this value to the closest representation = 133.
Changing the kernel value changes the filter output. When averaging a pixel on the image border or corner, appropriately ignore the undefined pixels in your average computation.

Uniform quantization (*m* = *1*) should be straight forward. Here you take the entire range of 0 – 255 and divide it into uniformly spaced intervals depending on the output number of bits of quantization. For example, if the output Q value is 3, you will want to divide this range into 8 intervals. You will then need to assign each range with a representative sample value which is typically the center value of the range. All the values in a given range then map to this representative value.

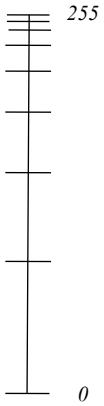
Logarithmic quantization (M=x where x is 0<=M<=255)
The motivation to use such non uniform quantization is to appropriately exploit the distribution of your input values. Here we desire to quantize the important pixels with less error and the less important ones with more error. For instance, if your image was taken when there was relatively less light and all the values were small (say less than 170), then there is no need to quantize any values above that. You would rather bias your quantization in the direction of less light. In this manner darker pixels get quantized with lesser error and lighter pixels with more error. Here are some distribution of intervals as value of *m* varies and the quantization Q = 3. As explained earlier, you may expect m to be a value between 0 and 255, inclusive.



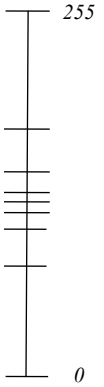
3 bits - Uniform Quantization



3 bits - Logarithmic Quantization centered at 0



3 bits - Logarithmic Quantization centered at 255



3 bits - Logarithmic Quantization centered at 128

What should you submit?

- Your source code, and your project file or makefile, if any, using the submit program. ***Please do not submit any binaries or images***. We will compile your program and execute our tests accordingly.
- If you need to include a readme.txt file with any special instructions on compilation, that is fine too.