CS 576 – Assignment 1
Instructor: Parag Havaldar

Assigned on Mon Jan 26, 2025,
Solutions due on Mon Feb 16, 2025 - by 5:00 pm afternoon
Late policies - None

Theory Part

Question 1

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?
- Suppose we want to store the video signal on a hard disk, and, 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?

Question 2

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.
- How many bits do you need to transmit it?

Question 3

Temporal aliasing can be observed when you attempt to record a rotating wheel with a video camera. In this problem, you will analyze such effects. Assume there is a car moving at 36 km/hr and you record the car using a film, which traditionally record at 24 frames per second. The tires have a diameter of 0.4244 meters. Each tire has a white mark to gauge the speed of rotation. Assume that the tire rotates without skidding.

- If you are watching this projected movie in a theatre, what do you perceive the rate of tire rotation to be in rotations/sec?
- If you use your camcorder to record the movie in the theater and your camcorder is recording at one third film rate (ie 8 fps), at what rate (rotations/sec) does the tire rotate in your video recording
- If you use an NTSC camera with 30 fps, what is the maximum speed that the car can go at so that you see no aliasing in the recording.

Programming Part (*200 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 Vscode 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 C, C++. java. *For assignments we do* ***not*** *allow usage 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. ***Please ensure that in your code you strictly follow the 512x512 resolution, test images will conform to this size***.
- The second parameter will be a ***S***cale value. This will control by how much the image must 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 used in quantization. The initial rgb file has 24 bits per pixel, which will be quantized down to a lower number. The value of Q will be 1<= Q <= 24, Q will be a multiple of 3 for the three channels (unless you are attempting the extra credit). Each channel will be quantized to Q/3 bits.
- The fourth parameter will suggest a ***M***ode for quantization. Its value will be an integer. A *M*=-1 indicates uniform interval quantization, A value 0<*M*<255 indicates a pivot value for a smarter logarithmic interval quantization. And a *M*=256 indicates a mode for optimal interval quantization. The value of *M*=256 indicates the most optimal interval implementation. All three methods are explained below.
- An optional fifth parameter ***E***=1 will be used if you indicated in your submission that you have done the extra credit.

How will you be evaluated:
Your submitted code will be compiled, and tests will be run for different scenarios. The evaluation will be comprised of:

1. Resampling tests
2. Quantization tests
3. Analysis Questions
4. Extra Credit (if attempted)

How will you program be invoked:
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 24 -1*

Here the image will be scaled by 1.0, with the output quantified to 8 bits per channel and use uniform quantization. With *Q*=24, this indicates that output needs to have 8 bits per channel, which is the same as input channel bits. This effectively means that your output will be the same as your input.

2. *YourProgram.exe myImage.rgb 0.5 24 -1*

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

IN THEATERS SEPTEMBER 18
CLOUDY
with a chance of
MEAT BALLS
IN 3D

IN THEATERS SEPTEMBER 18
CLOUDY
with a chance of
MEAT BALLS
IN 3D

3. *YourProgram.exe myImage.rgb 0.8 12 -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 reasons you will need to define a mapping for 16 values (for each channel) in the 0-255 range using uniform quantization. For example - 8,24,40,56…etc. are plausible values that correspond to the centers of a uniform range 0-15, 16-31, 32-47, 48-63 … etc.

4. *YourProgram.exe myImage.rgb 0.75 9 0*

Here the image will be scaled by 0.75, each channel is quantized to 3 bits (so a max of 8 values per channel) with logarithmic quantization. The logarithmic quantization starts at 0. For display reasons you will need to define a mapping for 8 values (for each channel) in the 0-255 range using logarithm quantization. For example, 4, 13, 25, 45, 74…etc. are plausible values that correspond to the centers of a logarithmic range 0-8, 8-17, 17-32, 32-57, 57-92 … etc.

Details of implementation:

Every pixel has an RGB value and an *(x, y)* location. When S in not equal to 1.0, you will need to compute the new location of the output 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.

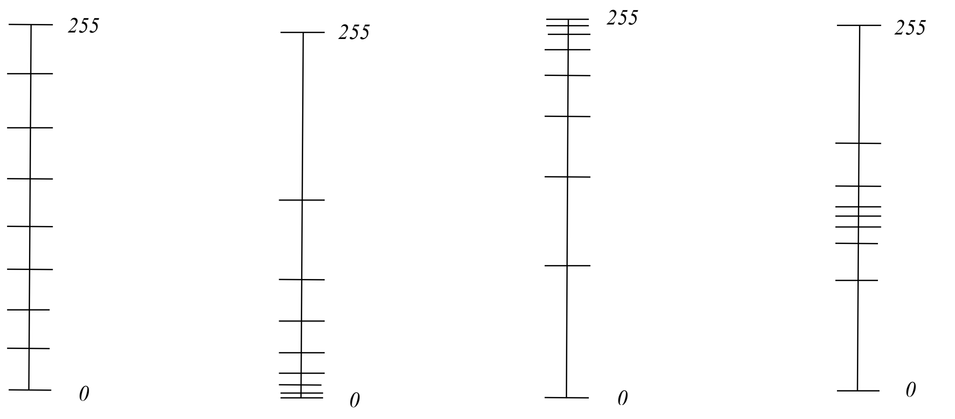
Changing the kernel value changes the filter output. When averaging a pixel on the image border or corner, appropriately ignore the undefined pixels in your average computation.

Uniform quantization (M = -1)

This mode 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 Q output is 9 (3 bits per channel), you will want to divide each channel's 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 each 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 quantization intervals for any values above that. You would rather bias your quantization in the direction of less light, where there are more pixels. In this manner darker pixels get quantized with less error and lighter pixels with more error. Here are some distributions of intervals as value of *m* varies and the quantization Q = 9. As explained earlier, you may expect *M* to be a value between 0 and 255, inclusive. Illustrated below is a figurative calculation of intervals for a channel to be quantized with 3 bits (when Q=9)



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

Optimal Interval Quantization (M=256)
The previous two methods had precomputed and decided interval values – whether uniform or logarithmic – once you know the number of bits to use. In this next mode, you are asked to take a more data driven approach and decide the most optimal interval sizes for the image at hand. So, all things being equal if a channel is quantized with 3 bits (when Q=9), you need to decide 8 non-overlapping sequential intervals, each interval of an appropriate size that can create the best possible output. You should be able to show that your outputs are visibly better (by displaying your image) and quantifiably better (in the next analysis part) when compared to the previous two modes.

Analysis Question:
In this analysis part, you are going to quantifiably show how well your different quantization modes work. The process of quantization is bound to lower the quality since it causes a loss. The loss can be measured in two standard ways as illustrated by the formulas below - the lower the error, the better your output is.

Mean Square Error = $\sum \sum (original[x,y] - output[x,y])^2$
Summed over all pixels, over all channels

Mean Absolute Error = $\sum \sum abs(original[x,y] - output[x,y])$
Summed over all pixels, over all channels

For an image, compute this error metric - leaving *S=1.0* (no scaling), for a variety of *Q* values for a given *M*. Your Q values should be varying as multiples of 3, 6, 9 …24 for each M when *M = -1, M=128 and M=256.* Plot a graph to show your error values, the X axis plots Q values and Y axis plots the error for your mode. You should have three curves for each mode.

Please show multiple sets of three curves for different images. Do you see any trend, explain!

Extra Credit:
For extra credit, we would like you to like to devise an even smarter method by relaxing the constraint that all channels need to have same bit representation which in our case has been a value which is a multiple of 3 (3,6, 9, …, 24). Modify (or implement) your algorithm so that we can give it any number between 1 and 24 inclusive and implement an optimal way to decide how to spread your bits between the channel eg if Q=9, you might use 2, 5, 2 bits for your R, G, B channels for example. For each channel you will still follow the *M* mode that is given as input.

- Modify your algorithm to find the most optimal spread of bits between channels. We will call by providing the last optional parameter E. An example invocation of this will look like

  YourProgram.exe myImage.rgb 1.0 11 256 1

  Here you are spreading 11 between optimally between the three channels using Optimal Interval Quantization.

- Given a mode *M*, compute your error metric for varying values of Q. Plot a graph that shows two curves for each M (M=-1, M=0, M=256) where one curve shows the error metric with equal bits per channel and the other curve shows the error metric with optimal distribution of bits between channels.
- Given that you are ***still not satisfied*** with the output quality and believe that you can improve by making better use of your bits, explain if there is any other way you can improve the quality of the output and get a better (lesser) error value? Explain your thoughts and reasoning.

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.
- Please include a readme.txt file with any special instructions on compilation if needed and with a note as to whether you have done your extra credit. If so, we will test it as shown above.
- For the analysis part (as well as the extra credit question if you choose to attempt), please submit a word or pdf document to illustrate your analysis.