

Image Processing

Spring 2024

Prof. George Wolberg

Homework 1

Due: Thursday, February 29

Objective: This assignment requires you to implement various point operations on digital images. See the sample solution for thresholding in the skeleton code (in CS470.skel/hw1/HW_threshold) to see what is expected of your programs in terms of style and comments. You should start by downloading the CS470.skel.zip file, which is available in the "Sample code" section of the class web page. The files in the CS470.skel

1) **HW_quantize** (ImagePtr I1, int levels, bool dither, ImagePtr I2)

Function *HW_quantize* reads the input image from I1 and uniformly quantizes it into *levels* quantization levels. The output is stored in I2. Quantization should be done by subdividing the range of 256 intensities into *levels* uniform intervals. Assign the midpoints of the uniform intervals as the output intensities. For instance, if *levels* = 4, then gray values 32, 96, 160, and 224 are used. In general, the intermediate gray values are increments of $256 / \text{levels}$ with a bias of $128 / \text{levels}$.

The *dither* flag denotes whether random noise is added/subtracted to each input pixel prior to quantization. This noise serves to reduce false-contour artifacts. The amplitude of the noise should be limited to the bias mentioned above. Use `rand()` to compute a random number, divide it by `RAND_MAX` to normalize it to the [0,1] range and scale it by bias to produce the random jitter. Then, alternatively add and subtract the computed jitter to successive pixels prior to quantization.

2) **HW_clip** (ImagePtr I1, int t1, int t2, ImagePtr I2)

Function *HW_clip* clips the input image in I1 to the range [t1,t2]. The output is saved in I2.

3) **HW_gamma** (ImagePtr I1, double gamma, ImagePtr I2)

Function *HW_gamma* performs gamma correction on the input image in I1 using the specified *gamma*.

4) **HW_contrast** (ImagePtr I1, double brightness, double contrast, ImagePtr I2)

Function *HW_contrast* applies contrast enhancement to I1. This function stretches the intensity difference from reference value (128) by multiplying the difference of each pixel from 128 by *contrast* and adding it back to 128. Shift result by adding *brightness* value. The output is saved in I2.

5) **HW_histoStretch** (ImagePtr I1, int t1, int t2, ImagePtr I2)

Function *HW_histoStretch* stretches the dynamic range of the input image in image I1 to fill the entire [0,255] range. The range that is stretched spans from t1 to t2. All intensity values below t1 or above t2 are pulled to 0 or 255, respectively. The output is stored in image I2. Note: you may want to look at the image histogram before selecting an appropriate t1 and t2.

6) **HW_histoMatch** (ImagePtr I1, ImagePtr targetHisto, bool approxAlg, ImagePtr I2)

Function *HW_histoMatch* performs a histogram matching operation to input image I1, saving the result in image I2. The histogram used is given in image targetHisto. That image has 256 numbers that denote the shape of the histogram curve. When all the numbers are the same, a flat histogram is specified and histogram equalization will be performed. You will have to make sure that the histogram entries are scaled properly so that their sum is equal to the total number of pixels in the input image. Modify the code supplied in the notes so that the histogram is matched *exactly*, with the possible exception of the very last histogram entry. If the boolean flag `approxAlg` is set to 1, then implement the simpler (non-exact) version of histogram matching, as shown in the textbook. If that flag is set to 0, then implement the exact version included in the notes, whereby the histogram of the output image will closely match the histogram specified in targetHisto.