Data Visualization HW1

(Cheng Zhong 16307110259)

1 (1) implement n-dimensional joint histogram and test the code on two-dimensional data; plot the results.

In this task, my program allows users to input the dimensions and size of each histogram, when the input dimension is 2, the generated histogram can be visualized. (Note: The program randomly generate the data in histogram.) An example output is shown below.

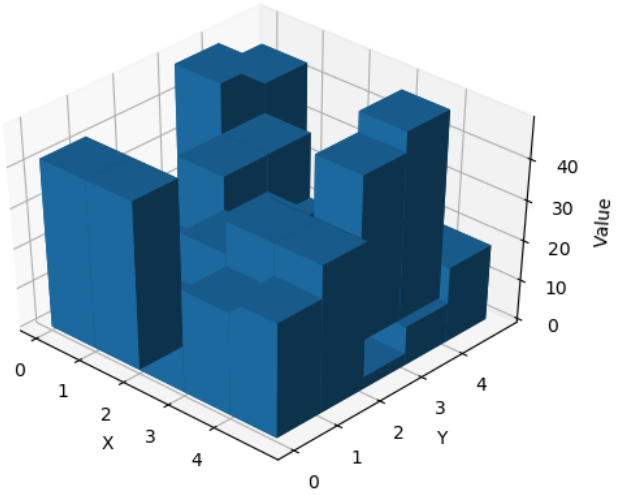


Figure 1: Example output of HW1_1

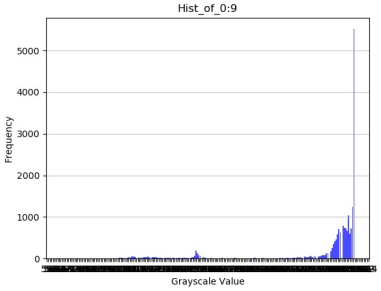
(2) implement computation of local histograms of an image using the efficient update of local histogram method introduced in local histogram processing.

The image we used for this task is as followed.

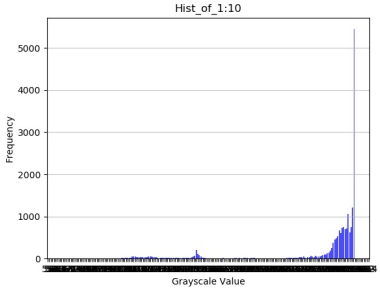
Here we first set the 10 columns on the left of the picture as the initial ROI field and plot the histogram, then every time we need to changes in a one-pixel translation of the neighborhood, we only need to update one column. Here I used the remainder operation to complete this step. For example, if we need to update the 11^{th} column, because (11 mod 10 = 1), so we only need to replace the first column by the 11^{th} column. For more details, please refer to the sample code provided.

The result of the first 5 iterations is followed. From the results, we can find that the gray value on the left of the picture is generally high, and there are certain differences between the different local histograms although they are generally the same, which is the same as the picture's presentation.

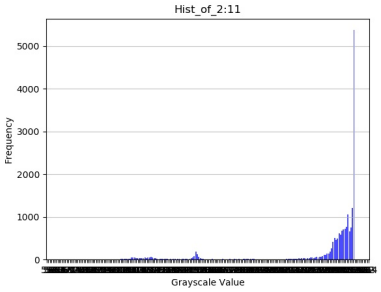
Figure 2: The picture used in HW1_2



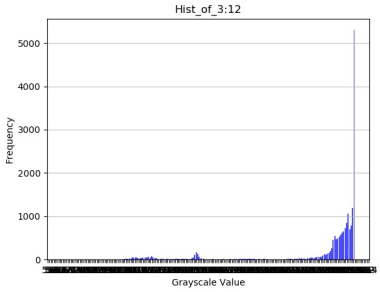
(a) Hist1



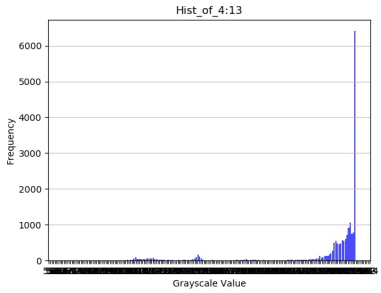
(b) Hist2



(c) Hist3



(d) Hist4



(e) Hist5

Figure 3: The result of the first 5 iterations

2. Implement a piece-wise linear transformation function (below figure) for image contrast stretching. The code should read in an image; for intensity of all pixels, use the function to compute new intensity values; and finally output/save the image with new intensity values.

The piece-wise linear transformation function we used in this task is:

$$f(x) = \begin{cases} 0.45x & 0 \leq x < 100 \\ x - 55 & 100 \leq 0 < 200 \\ 2x - 255 & 200 \leq 0 < 255 \end{cases} \tag{1}$$

The image before and after transformation is shown below. I divided the image into three channels (R, G, and B), and convert the value of each channel in any pixel. We can see that the values of pixel in the picture are moved to the middle, so that the colors of the picture are more obvious and gentle

(a) Before (b) After

Figure 4: The image before and after transformation

3 Implement the algorithm of local histogram equalization: (1) first implement histogram equalization algorithm, and then (2) implement the local histogram equalization using efficient computation of local histogram. Please test your code on images and show the results in your report.

In order to save time, we used a gray scale image with lower resolution to complete this experiment.

When implement the algorithm of local histogram, I first sort the grey value of all the pixel in ROI, and then use the formula $S_k = \sum_{j=0}^{k} \frac{n_j}{n}, k = 0, 1, 2, ...L - 1$ to calculate the new grey value.

Also, when I implement the local histogram equalization using efficient computation of local histogram, I use the same method in 1(2), which set the ROI first, and then using the remainder operation to update one column in a iteration. The result is as follows. After equalization, the contrast of the picture is higher and the colors are more vivid.

However, the color of the image changes greatly after local-equalization, especially when there are black and white colors in the ROI, which are very extreme in the gray value. For more details, please refer to the sample code provided.

(a) Origin Picture

(b) After equalization

(c) After local-equalizationn

Figure 5: The image before and after Equalization