







Course: Image Processing 31651

Assignment #22

Pixel to Pixel Operations (Part 2)

	ID (4 last digits)	Shorten Name	Photo of the student	
Student #1	1950	shienfeld		
Student #2	2210	pony		
Student #3	7939	akimov		

Assignment #22: Process set of Gray Images

In the file ImProcInPlainC.h defined numerical values of:
NUMBER_OF_ROWS and NUMBER_OF_COLUMNS

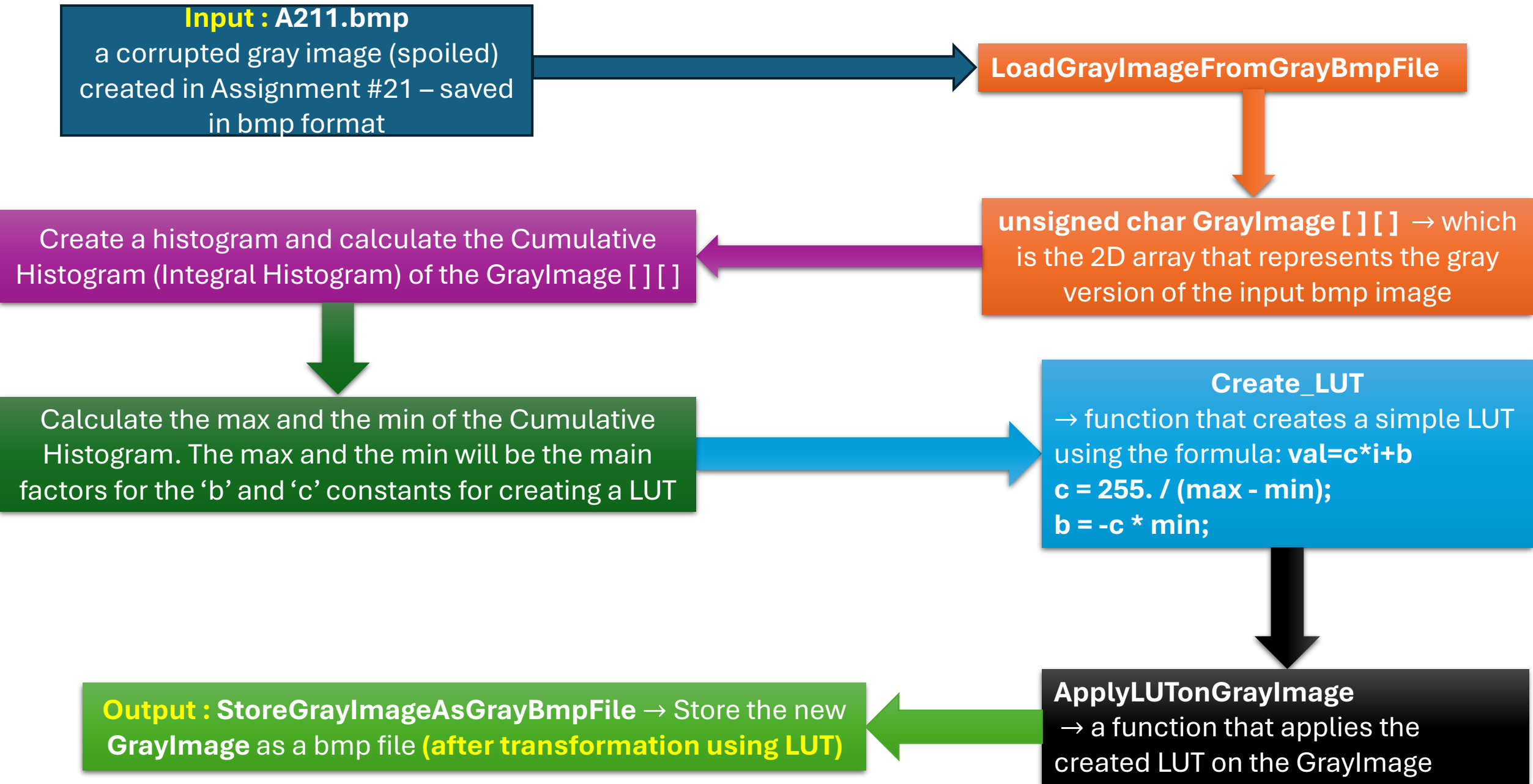
#	Image Description
0	Use earlier prepared image A211.bmp
1	Apply 20% - 80% algorithm to A211.bmp. Store processed Image as A221.bmp Apply 5% - 95% algorithm to A211.bmp. Store processed Image as A222.bmp The algorithm must use pointers to access the pixels
	For report use YOUR OWN STYLE. From YOUR Power Point customer must understand what was done, how and what are the results (Lecturer may later ask to add slides) Done 22.1 REMOVE ALL RED when the items are ready Important hint: Fill report pages while working. Do not complain "not enough time" in case you did not follow THIS rule.

Pay attention:

The main difference between that assignment (22) and the previews one (21) is the selection of the 'b' and 'c' constants. Now we need to create a histogram and to integrate it for calculating the 'b' and 'c' constants for the LUT. (In assignment 21 constants were selected using a simple formula given in the assignment)

22.0 - flow chart of the code – general look

3



22.1 – first important function – prepare a LUT

```
72 // Preparing Look-Up-Table
73 void create_percentage_lut(unsigned char* img, unsigned char* LUT, double bottom, double upper, int rows, int cols)
74 {
75     unsigned hist[PIXEL_VALUES] = { 0 };
76     unsigned IntOfHist[PIXEL_VALUES] = { 0 };
77     fill_hist_and_calc_int(img, hist, IntOfHist, rows, cols);
78
79     unsigned char min = 0, max = 0;
80     FindMinMaxFromIntOfHist(IntOfHist, min, max, bottom, upper);
81     double c = 255.0 / (max - min);
82     double b = -c * min;
83     Create_LUT(LUT, c, b);
84 }
```

The purpose of the `create_percentage_lut` function is to prepare a Look-Up Table (LUT) that will be used to adjust the pixel values of a grayscale image based on specified percentage thresholds.



The `create_percentage_lut` calls several functions that will be presented in the following slides when the ending point will be creating a LUT (look up table) of the image for further use

22.2 – second important function – create a histogram and its integral – part 1/2

```
44 void fill_hist_and_calc_int(unsigned char* img, unsigned* hist, unsigned* IntOfHist, int rows, int cols)
45 {
46     for (int i = 0; i < rows * cols; i++)
47     {
48         hist[*img + i] += 1;
49     }
50     IntOfHist[0] = hist[0];
51     for (unsigned i = 1; i < PIXEL_VALUES; i++)
52     {
53         *(IntOfHist + i) = *(IntOfHist + i - 1) + *(hist + i);
54     }
55 }
```

The `fill_hist_and_calc_int` function is responsible for:

1. **Calculating the Histogram:** Counting the number of occurrences of each pixel value in the image.
2. **Calculating the Cumulative Histogram (Integral Histogram):** Computing the cumulative sum of the histogram values.

```
for (int i = 0; i < rows * cols; i++) {
    hist[*img + i] += 1;
}
```

This loop iterates over all the pixels in the image (total rows * cols pixels).

`*(img + i)` accesses the pixel value at the i-th position using pointer arithmetic.

`hist[*img + i] += 1` increments the count for the pixel value in the histogram array. It counts how many times each pixel value (0-255) appears in the image.

hist is an array of size 256 (PIXEL_VALUES), where each index corresponds to a pixel value (0-255).

hist[i] represents the count of pixels in the image that have the value i

After the histogram calculation, hist contains the frequency distribution of pixel values in the image.

22.2 – second important function – create a histogram and its integral – part 2/2

```
IntOfHist[0] = hist[0];  
for (unsigned i = 1; i < PIXEL_VALUES; i++) {  
    IntOfHist[i] = IntOfHist[i - 1] + hist[i];  
}
```



IntOfHist[0] = hist[0]: Initializes the first value of the cumulative histogram with the first value of the histogram.



The loop iterates over all possible pixel values (0-255).



IntOfHist[i] = IntOfHist[i - 1] + hist[i] computes the cumulative sum for each pixel value, effectively creating the cumulative histogram.

IntOfHist is also an array of size 256.

IntOfHist[i] represents the cumulative count of pixels with values from 0 to i.

After the cumulative histogram calculation, IntOfHist contains the cumulative distribution of pixel values, which is useful for certain image processing techniques, such as histogram equalization.

22.3 – third important function – find max and min from histogram– **part 1/2**

7

```
57 void FindMinMaxFromIntOfHist(unsigned* IntOfHist, unsigned char& min, unsigned char& max, double bottom, double upper)
58 {
59     double bottomCount = bottom * (*(IntOfHist + PIXEL_MAXVAL));
60     double topCount = upper * (*(IntOfHist + PIXEL_MAXVAL));
61     min = 0;
62     max = 0;
63     for (unsigned i = 0; i < PIXEL_VALUES; i++)
64     {
65         if (*(IntOfHist + i) < bottomCount)
66             min = i;
67         if (*(IntOfHist + i) < topCount)
68             max = i;
69     }
70 }
```

The **FindMinMaxFromIntOfHist** function is designed to determine the minimum and maximum pixel values in an image based on specified percentage limits (bottom and upper). This is done on the basis of the cumulative histogram (IntOfHist).

```
59     double bottomCount = bottom * (*(IntOfHist + PIXEL_MAXVAL));
60     double topCount = upper * (*(IntOfHist + PIXEL_MAXVAL));
61     min = 0;
62     max = 0;
```



bottomcount is calculated by multiplying the lower percentage limit (bottom) by the total number of pixels (IntOfHist[PIXEL_VALUES - 1]), which is the last value in the cumulative histogram.

topcount is calculated similarly using the upper percentage limit (upper).

min and **max** are initialized to 0. They will store the minimum and maximum pixel values found within the specified limits.

22.3 – third important function – find max and min from histogram– part 2/2



```
63  for (unsigned i = 0; i < PIXEL_VALUES; i++)
64  {
65      if (*(IntOfHist + i) < bottomCount)
66          min = i;
67      if (*(IntOfHist + i) < topCount)
68          max = i;
69  }
70 }
```



This loop iterates through all possible pixel values (0 to 255):

if (IntOfHist[i] < bottomcount) → If the cumulative histogram value at i is less than bottomcount, update min to i.
This finds the pixel value corresponding to the lower percentage limit

if (IntOfHist[i] < topcount) → If the cumulative histogram value at i is less than topcount, update max to i.
This finds the pixel value corresponding to the upper percentage limit.

22.4 – fourth important function – create a LUT and apply it on the image

```
22 void Create_LUT(unsigned char* LUT, double c, double b)
23 {
24     for (unsigned i = 0; i <= PIXEL_MAXVAL; i++)
25     {
26         double val = c * i + b;
27         if (val > PIXEL_MAXVAL)
28             *(LUT + i) = PIXEL_MAXVAL;
29         else if (val < PIXEL_MINVAL)
30             *(LUT + i) = PIXEL_MINVAL;
31         else
32             *(LUT + i) = val;
33     }
34 }
```

Create_LUT is a very simple function for creating a LUT using the contrast (c) and brightness (b) constants.

```
36 void ApplyLUTOnGrayImage(unsigned char* img, unsigned char* LUT, int rows, int cols)
37 {
38     for (int i = 0; i < rows * cols; i++)
39     {
40         *(img + i) = *(LUT + *(img + i));
41     }
42 }
```

After defining the LUT – we need to apply the LUT of the original gray image (img [] [])

22.5 – fifth important function – the “main”

```
93 // Final stage, execute
94 ✓int main()
95 {
96     LoadGrayImageFromGrayBmpFile(GrayImage, "A211.bmp");
97     percentage_algorithm(&GrayImage[0][0], 0.2, 0.8, NUMBER_OF_ROWS, NUMBER_OF_COLUMNS);
98     StoreGrayImageAsGrayBmpFile(GrayImage, "A221.bmp");
99
100    LoadGrayImageFromGrayBmpFile(GrayImage, "A211.bmp");
101    percentage_algorithm(&GrayImage[0][0], 0.05, 0.95, NUMBER_OF_ROWS, NUMBER_OF_COLUMNS);
102    StoreGrayImageAsGrayBmpFile(GrayImage, "A222.bmp");
103
104    WaitForUserPressKey();
105    return 0;
106 }
```



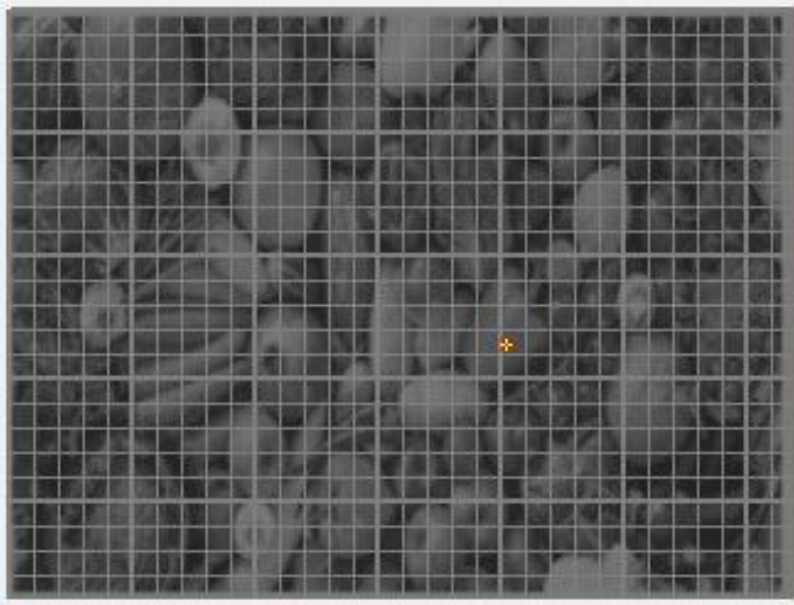
```
86 // In this function we insert 20%-80% & 5%-95% values:
87 ✓void percentage_algorithm(unsigned char* img, double bottom, double upper, int rows, int cols)
88 {
89     create_percentage_lut(img, LUT, bottom, upper, rows, cols);
90     ApplyLUTOnGrayImage(img, LUT, rows, cols);
91 }
```

The percentages will be assigned to the 'bottom' and 'upper' variables

22.6 – demonstration using profiler – part 1/4

11

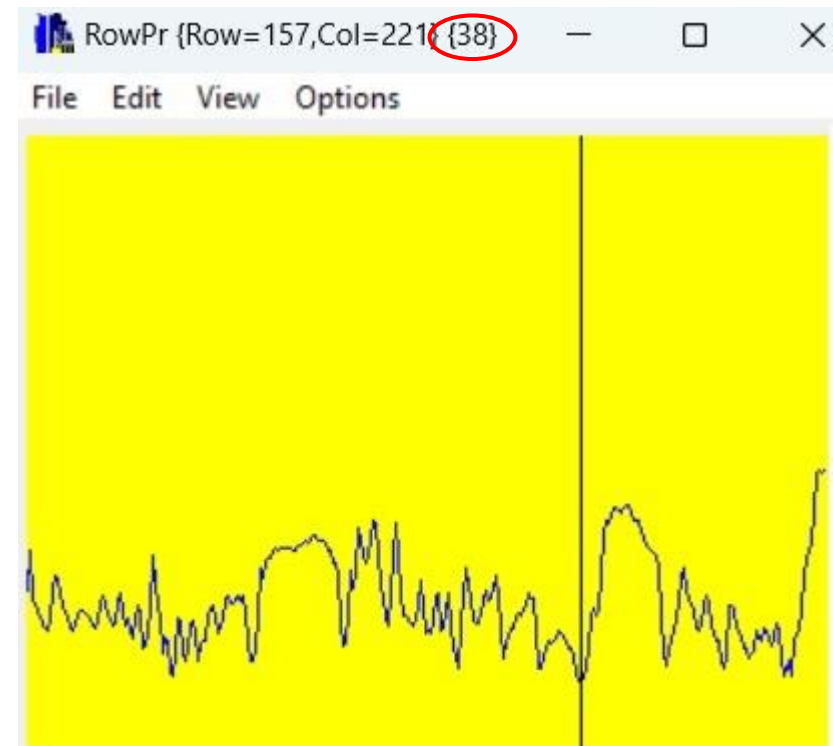
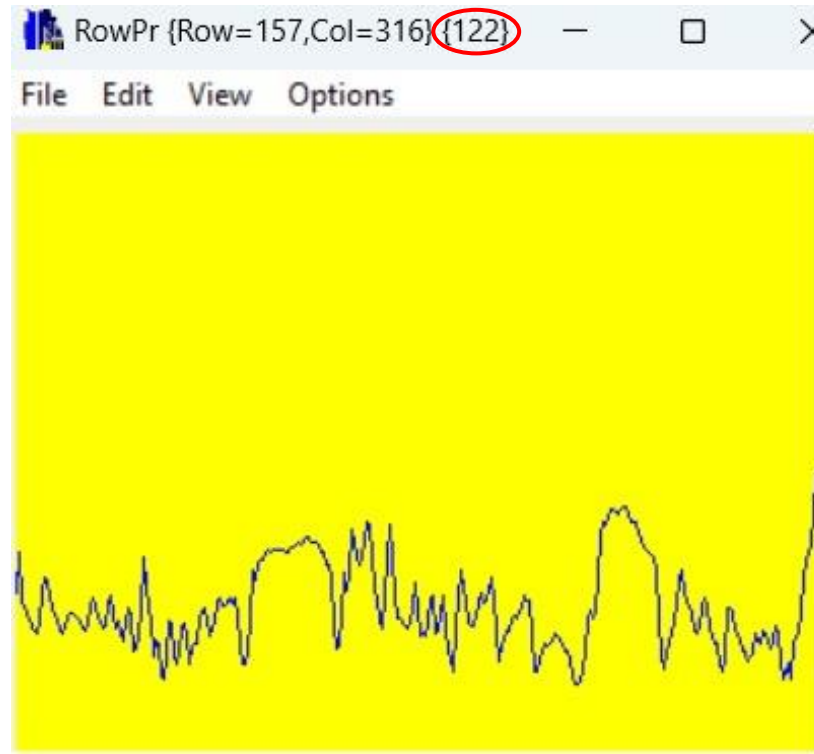
Original photo (spoiled): A211.bmp



Here we can see the original spoiled image we provided.

Important conclusion: we see that the image consists approximately 32% of the grayscale range (according to the profilers) because highest point is 122 and the lowest is 38.

The usage of the grayscale range is very low, and this is the reason for the poor quality of the image (we discussed it in Assignment 21) - the next algorithms that will be applied to improve the quality of the image – have to extend the range of the grayscale.



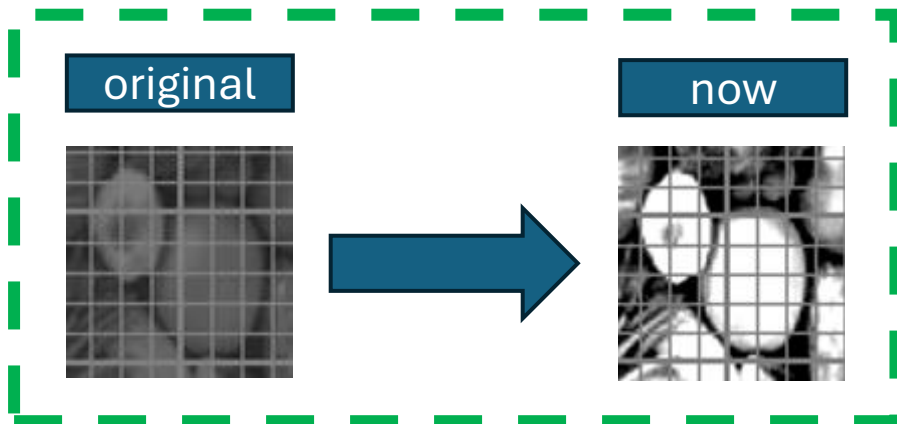
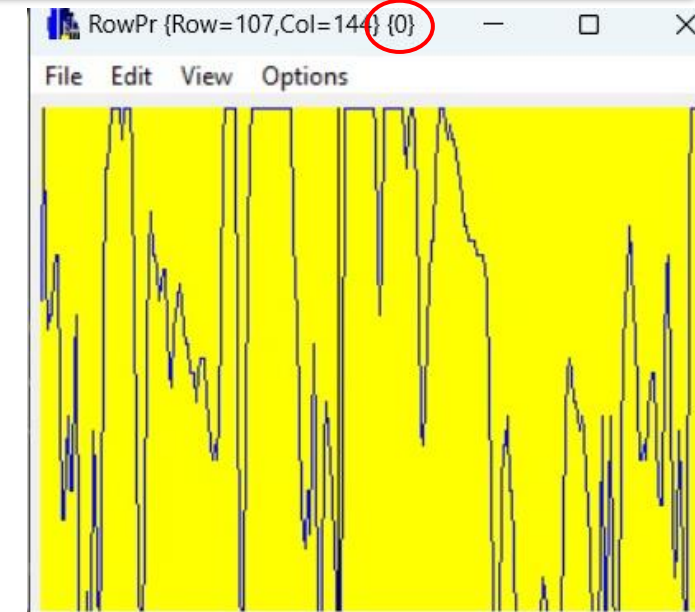
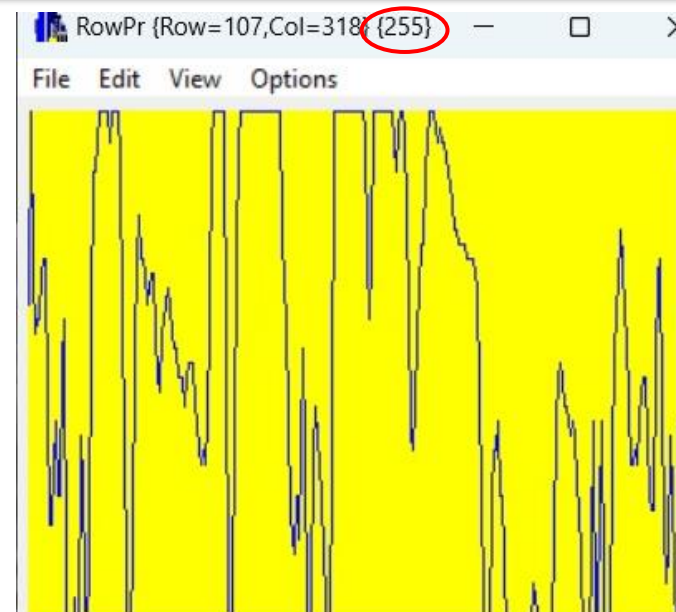
22.6 –demonstration using profiler – **part 2/4**

12

20%-80% Algorithm applied on A211.bmp image

One can see that the algorithm made the photo brighter, gray levels were changed significantly, and the objects (fruits & vegetables) are more distinguishable but unrealistic. The 20%-80% algorithm does not give us the preferred result.

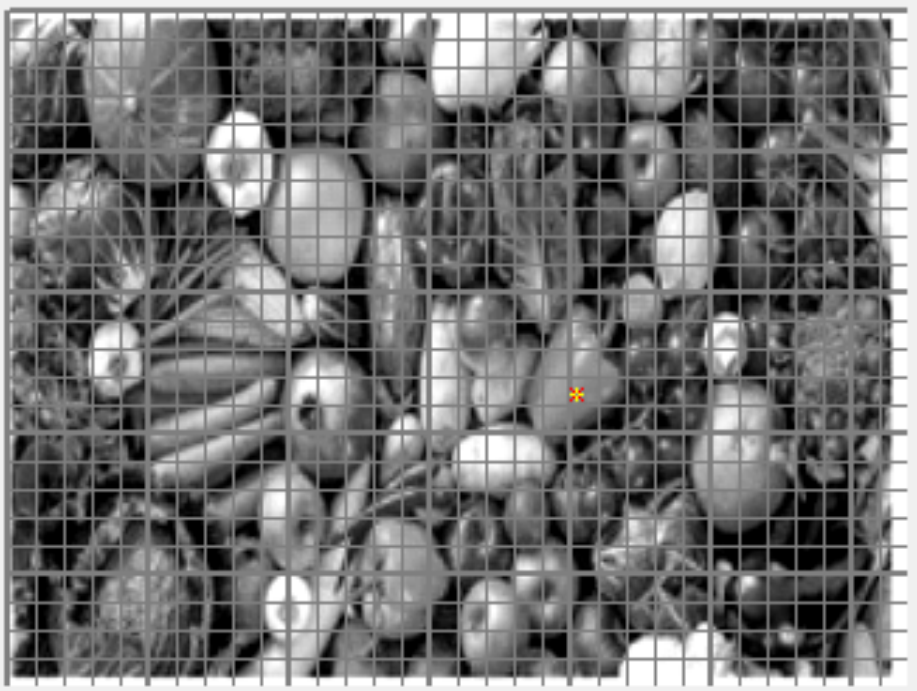
Important conclusion: we see that the image consists approximately 100% of the grayscale range (according to the profilers) because highest point is 255 and the lowest is 0. It means that the whole range of the grayscale is used – it explains the dramatic change in the image. From the other hand, we notice that the algorithm brightened the image too much – the 20%-80% is not a good implementation for improving the quality of a spoiled image!



22.6 – demonstration using profiler – **part 3/4**

13

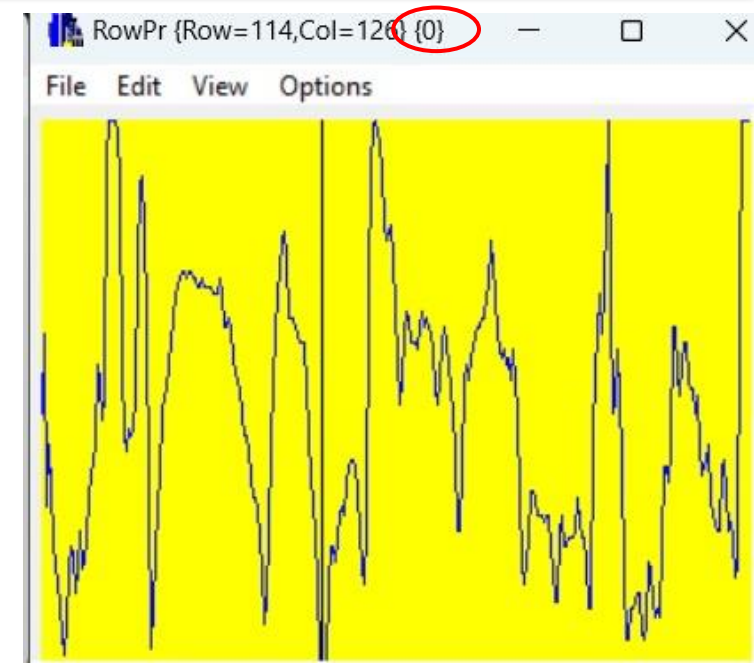
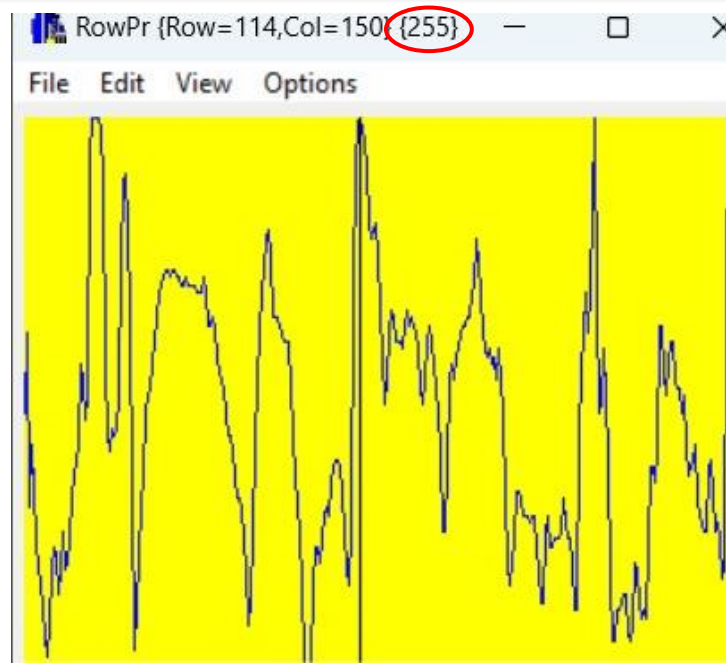
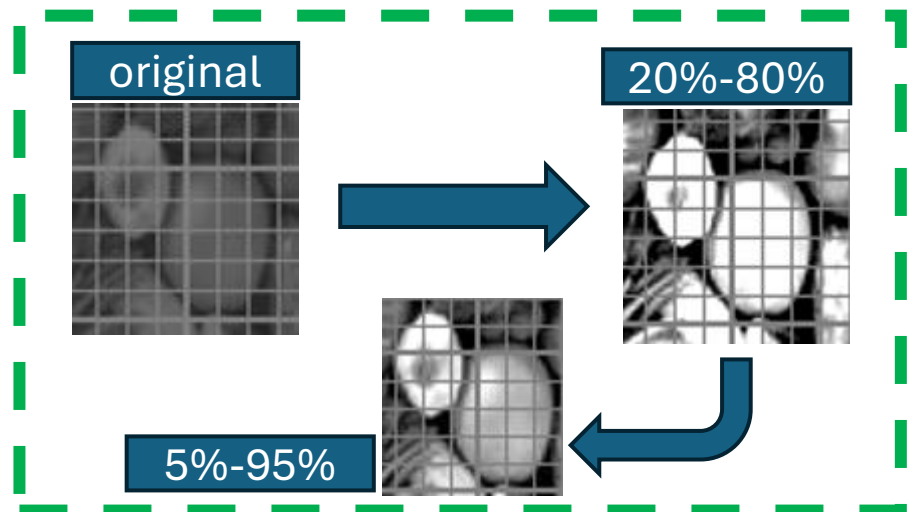
5%-95% Algorithm applied on A211.bmp image



One can see that the fruits and vegetables in our 5%-95% fixed image are way clearer and more convenient to watch (have better contrast). The 5%-95% algorithm gives us the preferred result.

Important conclusion: we see that the image consists approximately 100% of the grayscale range (according to the profilers) because highest point is 255 and the lowest is 0.

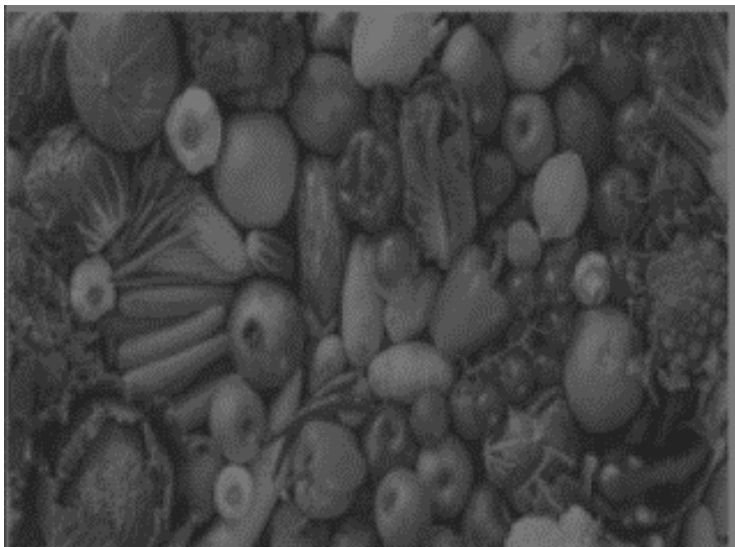
It means that the whole range of the grayscale is used – it explains the dramatic change in the image. In comparison to the 20%-80% algorithm – now the brightening is exactly in the appropriate amount and the image looks sharp and realistic. The extra brightening of the 20%-80% algorithm did not happen here.



22.7 –conclusions

1. The 5%-95% algorithm provides a significantly better contrast enhancement compared to the 20%-80% algorithm. The details in the image are more pronounced, making the objects and their textures more distinguishable.
2. From a perceptual standpoint, the 5%-95% algorithm provides a more balanced image. The objects are easily recognizable, and the overall visual quality is enhanced.
3. The usage range of the grayscale is not always a good indication for the quality and the visibility of the image – in assignment 21 it was , but now the whole range is used (100%) and yet the differences are dramatic.

Original photo (spoiled): A211.bmp



20%-80% Algorithm applied on A211.bmp image



5%-95% Algorithm applied on A211.bmp image



22.8 What did we learned?

- 1) We learned to improve an image using the principle of accumulative histograma.
- 2) We learned that it is better to take small portion (5% instead of 25% for example) in order to improve the image and it's objects visibility.
- 3) The difference between types of use in the algorithm
- 4) To use algorithm to fix images.