

Report Computer Vision TP2

Choules, Louis

`louis.choules@etu.univ-grenoble-alpes.fr`

Lkhagvajav, Dagvanorov

`dagvanorov.lkhagvajav@etu.univ-grenoble-alpes.fr`

October 2022

Contents

List of Code	3
List of Figures	5
1 Introduction	6
1.1 filtering_3x3	6
1.2 filtering_5x5	6
1.3 filtering	7
1.4 filter_median	7
1.5 histogram	8
1.6 Image Test Case	8
2 Program Description	10
2.1 Local Filter	10
2.1.1 Gaussian Filter	10
2.1.2 Median Filter	11
2.2 Histogram	11
2.2.1 Histogram Stretching	11
2.2.2 Histogram Equalization (Normalization)	12
3 Program	13
3.1 Code Explanation	13
3.1.1 get_byte_by_pos	13
3.1.2 createFilter	13
3.1.3 repeat_run	13
3.1.4 get_new_byte Gaussian Filter Version	14
3.1.5 get_new_byte Median Filter Version	14
3.1.6 clear_histogram	15
3.1.7 create_histogram	15

3.1.8	getMaxMin	15
3.1.9	histogram_stretching	16
3.1.10	make_Fi	16
3.1.11	histogram_equalization	16
3.2	Question	17
3.2.1	Exercise 1 : Filtering	17
3.2.2	Exercise 2 : Histograms	28
3.3	Result	40
3.3.1	Local Filters	40
3.3.2	Histogram Modifications	46

List of Code

1	Compile filtering3x3	6
2	Use filtering3x3	6
3	Compile filtering5x5	6
4	Use filtering5x5	6
5	Compile filtering	7
6	Use filtering	7
7	Compile filter_median	7
8	Use filter_median	7
9	Compile histogram	8
10	Use histogram	8
11	get_byte_by_pos	13
12	createFilter	13
13	repeat_run	13
14	get_new_byte Gaussian Filter Version	14
15	get_new_byte Median Filter Version	14
16	clear_histogram	15
17	create_histogram	15
18	getMaxMin	15
19	histogram_stretching	16
20	make_Fi	16
21	histogram_equalization	16
22	filtering_3x3	17
23	filtering_5x5	19
24	filter_median	22
25	filtering	24
26	histogram	28
27	histogram_stretching	31
28	histogram_equation	36
29	Make Figure boat_3_1.png	41
30	Make Figure boat_3_1.png	41
31	Make Figure boat_3_5.png	42
32	Make Figure boat_3_5.png	42
33	Make Figure boat_5_1.png	42
34	Make Figure boat_5_1.png	42
35	Make Figure boat_5_5.png	43
36	Make Figure boat_5_5.png	43
37	Make Figure boat_median_3_1.png	44
38	Make Figure boat_median_3_5.png	44
39	Make Figure boat_median_5_1.png	45
40	Make Figure boat_median_5_3.png	46
41	Make Figure boat_histogram_stretching_3_1.png	46
42	Make Figure boat_histogram_stretching_3_5.png	47
43	Make Figure boat_histogram_stretching_5_1.png	48
44	Make Figure boat_histogram_stretching_5_5.png	48

45	Make Figure boat_histogram_equalization_3_1.png	49
46	Make Figure boat_histogram_equalization_3_5.png	50
47	Make Figure boat_histogram_equalization_5_1.png	50
48	Make Figure boat_histogram_equalization_5_5.png	51

List of Figures

1	boat.png	8
2	boat_noise1.png	9
3	boat_noise2.png	9
4	Applying a 3x3 binomial filter 1 time. (boat_3_1.png)	41
5	Applying a 3x3 binomial filter 5 times. (boat_3_5.png)	41
6	Applying a 5x5 binomial filter 1 time. (boat_5_1.png)	42
7	Applying the 5x5 binomial filter 5 times. (boat_5_5.png)	43
8	Applying a 3x3 median filter 1 time. (boat_median_3_1.png)	44
9	Applying a 3x3 median filter 5 times. (boat_median_3_5.png)	44
10	Applying a 5x5 median filter 1 time. (boat_median_5_1.png)	45
11	Applying a 5x5 median filter 3 times. (boat_median_5_3.png)	45
12	Applying histogram stretching on the image after filtering with 3x3 binomial 1 time. (boat_histogram_stretching_3_1.png)	46
13	Applying histogram stretching on the image after filtering with 3x3 binomial 5 times. (boat_histogram_stretching_3_5.png)	47
14	Applying histogram stretching on the image after filtering with 5x5 binomial 1 time. (boat_histogram_stretching_5_1.png)	47
15	Applying histogram stretching on the image after filtering with 5x5 binomial 5 times. (boat_histogram_stretching_5_5.png)	48
16	Applying histogram equalization on the image after filtering with 3x3 binomial 1 time. (boat_histogram_equalization_3_1.png)	49
17	Applying histogram equalization on the image after filtering with 3x3 binomial 5 times. (boat_histogram_equalization_3_5.png)	49
18	Applying histogram equalization on the image after filtering with 5x5 binomial 1 time. (boat_histogram_equalization_5_1.png)	50
19	Applying histogram equalization on the image after filtering with 5x5 binomial 5 times. (boat_histogram_equalization_5_5.png)	51

1 Introduction

This is the report for the TP2 Computer Vision. For this project, we have several files as describe below.

1.1 filtering_3x3

This file is for creating the image with *Gaussian Filter* with filter 3x3.

How to compile

```
1 make filtering_3x3
```

List of Code 1: Compile filtering3x3

How to use

```
1 ./filtering_3x3 [filename]>[file_output] [times_of_repetition]
```

List of Code 2: Use filtering3x3

you need to pass some arguments in here.

- **filename** is the picture that you want to modified
- **file_output** is the name of output picture
- **times_of_repetition** is the total times of filter using Gaussian Filter (3x3)

1.2 filtering_5x5

This file is for creating the image with *Gaussian Filter* with filter 5x5.

How to compile

```
1 make filtering_5x5
```

List of Code 3: Compile filtering5x5

How to use

```
1 ./filtering_5x5 [filename]>[file_output] [times_of_repetition]
```

List of Code 4: Use filtering5x5

you need to pass some arguments in here.

- **filename** is the picture that you want to modified
- **file_output** is the name of output picture
- **times_of_repetition** is the total times of filter using Gaussian Filter (5x5)

1.3 filtering

This file is for creating the image with **Gaussian Filter** where you can specify the size of the filter manually.

How to compile

```
1 make filtering
```

List of Code 5: Compile filtering

How to use

```
1 ./filtering [filename]>[file_output] [size_of_the_blur] [  
    times_of_repetition]
```

List of Code 6: Use filtering

you need to pass some arguments in here.

- **filename** is the picture that you want to modified
- **file_output** is the name of output picture
- **size_of_the_blur** is the size of filter (size_of_the_blur x size_of_the_blur) expected in odd number and greater than 0
- **times_of_repetition** is the total times of filter

1.4 filter_median

This file is for creating the image with **Median Filter** where you can specify the size of the filter manually.

How to compile

```
1 make filter_median
```

List of Code 7: Compile filter_median

How to use

```
1 ./filter_median [filename]>[file_output] [size_of_the_blur] [  
    times_of_repetition]
```

List of Code 8: Use filter_median

you need to pass some arguments in here.

- **filename** is the picture that you want to modified
- **file_output** is the name of output picture
- **size_of_the_blur** is the size of filter (size_of_the_blur x size_of_the_blur) expected in odd number and greater than 0
- **times_of_repetition** is the total times of filter using median

1.5 histogram

This file is for creating the image with **Gaussian Filter** where you can specify the size of the filter manually and also add **Histogram Modification** at the result of the image.

How to compile

```
1 make histogram
```

List of Code 9: Compile histogram

How to use

```
1 ./histogram [filename]>[file_output] [size_of_the.blur] [  
times_of_repetition] [histogram_mode]
```

List of Code 10: Use histogram

you need to pass some arguments in here.

- **filename** is the picture that you want to modified
- **file_output** is the name of output picture
- **size_of_the.blur** is the size of filter (size_of_the.blur x size_of_the.blur) expected in odd number and greater than 0
- **times_of_repetition** is the total times of filter using Gaussian Filter
- **histogram_mode** is the mode of histogram transformation expected value "s" or "e"

1.6 Image Test Case



Figure 1: boat.png



Figure 2: boat_noise1.png



Figure 3: boat_noise2.png

2 Program Description

2.1 Local Filter

Local filtering is consist of replacing a pixel value by a weighted combination of values in local neighborhood of the pixel. The weight is based on the filter that we will use.

2.1.1 Gaussian Filter

for Gaussian Filter we have 3 files such as, *filtering_3x3*, *filtering_5x5*, and *filtering*. For this three files, we using the same concept of it. We define a binomial filter with n size from a Pascal Triangle.

$$\begin{array}{ll}
 n = 0 & 1 \\
 n = 1 & 1 \quad 1 \\
 n = 2 & 1 \quad 2 \quad 1 \\
 n = 3 & 1 \quad 3 \quad 3 \quad 1 \\
 n = 4 & 1 \quad 4 \quad 6 \quad 4 \quad 1 \\
 n = 5 & 1 \quad 5 \quad 10 \quad 10 \quad 5 \quad 1 \\
 n = 6 & 1 \quad 6 \quad 15 \quad 20 \quad 15 \quad 6 \quad 1 \\
 & \hline
 & 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6
 \end{array}$$

We will create a Pascal triangle like above, we need to get the filter by using the n at size-1; For example we will using 3×3 filter, then we will using the $n = 2$. We need to create a matrix from it by matrix multiplication. The divider is comes from the sum of all the values in the filter.

$$h = 1/16 \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

For example we have a pixel with the neighbors like below.

$$\begin{bmatrix} 32 & 12 & 4 \\ 21 & 41 & 23 \\ 41 & 32 & 13 \end{bmatrix}$$

then we apply the filter above with current matrix. we will get

$$\begin{aligned}
 h &= 1/16 \begin{bmatrix} 1 \times 32 & 2 \times 12 & 1 \times 4 \\ 2 \times 21 & 4 \times 41 & 2 \times 23 \\ 1 \times 41 & 2 \times 32 & 1 \times 13 \end{bmatrix} \\
 h &= \frac{1 \times 430}{16} \\
 h &= 26.875
 \end{aligned}$$

We have 3 option about how to change it into an integer, either we can use either `floor()`, `ceiling()`, or `round()`. But for our code, we only using the integer directly so it similar to `floor()`. So for the new pixel will be 26.

2.1.2 Median Filter

for Median Filter, we have 1 file called, filter_median. The concept of this filter is take all the value in it's pixel value and it's neighbor's pixel values. Then we sort all the values and for the new value is will be the one in the middle.

For example we have a pixel with the neighbors like below.

$$\begin{bmatrix} 32 & 12 & 4 \\ 21 & 41 & 23 \\ 41 & 32 & 13 \end{bmatrix}$$

Then we sort all the value so it become, [4, 12, 13, 21, 23, 32, 32, 41, 41]. Then we will take the value in the middle of the sorted value as the new value in that pixel. So for the new pixel in that pixel is 23.

2.2 Histogram

PGM formatted images are formatted as 2-dimensional arrays. Each element of the array is in the range between 0 and 255 in regard of the its intensity. Histogram of images is used to representing the distribution of the values between the range.

2.2.1 Histogram Stretching

In order to make image details more clear and perceivable, there should be gaps between the intensity values in the histogram. Histogram stretching is used to enhancing the contrast of an image. By considering the histogram of the image, the contrast is the difference between the maximum and the minimum pixel intensity. In other words, the histogram stretching expands the entire range of the pixel intensity value. It should be noted that the min and max pixel intensity of an image should not be the same as possible min and max intensities (0, 255).

Let P_i be a pixel intensity value.

Let a, b be the current minimum and maximum values of the image, respectively.

Let MIN, MAX be the possible minimum and maximum pixel intensity.

Thus, the modified value of the pixel intensity after stretching is P_{st} .

$$P_{st} = \frac{(P_i - a)(MAX - MIN)}{(b - a)} + MIN$$

Since MIN equals 0, we can retrieve the following formula.

$$P_{st} = \frac{(P_i - a) * MAX}{(b - a)}$$

2.2.2 Histogram Equalization (Normalization)

On the other side, the histogram equalization is used to normalize the histogram density into a uniform distribution. The expected result is for the cumulative histogram of the image becomes as flat as possible.

Let $hist(j)$ is the histogram value of j intensity.

Let P_i be a pixel intensity value.

Let Max be the maximum pixel intensity.

$$P_{st} = Max \frac{\sum_{j=0}^{P_i} hist(j)}{\sum_{j=0}^{Max} hist(j)}$$

3 Program

3.1 Code Explanation

3.1.1 get_byte_by_pos

```
1 int get_byte_by_pos(int col, int row){  
2     if(row < 0)  
3         row = 0;  
4     if(col < 0)  
5         col = 0;  
6     if(col >= cols)  
7         col = cols-1;  
8     if(row >= rows)  
9         row = rows-1;  
10  
11     return graymap[row * cols + col];  
12 }
```

List of Code 11: get_byte_by_pos

This line of code is for take the byte based on coordinate passed in the arguments. if the coordinate is out of the image, it will return the nearest pixel.

3.1.2 createFilter

```
1 void createFilter(int filter_size){  
2     int coef, i, j, temp_pascal[filter_size+2];  
3     divider = 0;  
4     for(i = 0; i<= filter_size; i++){  
5         if(i==0)  
6             coef = 1;  
7         else  
8             coef = coef * (filter_size-i)/i;  
9         temp_pascal[i] = coef;  
10    }  
11  
12    filter = malloc(sizeof(int) * filter_size * filter_size);  
13    for(i = 0; i < filter_size; i++){  
14        for(j = 0; j < filter_size; j++){  
15            int x = temp_pascal[i] * temp_pascal[j];  
16            filter[i*filter_size+j] = x;  
17            divider += x;  
18        }  
19    }  
20 }  
21 }
```

List of Code 12: createFilter

This line of code is for creating the filter dynamically, it will return the filter based on the size passed on the argument.

3.1.3 repeat_run

```
1 void repeat_run(int total_runs, int filter_size){  
2     gray temp_graymap[cols * rows];  
3     int k;
```

```

4   int i, j;
5   for(k = 0; k < total_runs; k++){
6     for(i=0; i < rows; i++){
7       for(j=0; j < cols ; j++) {
8         temp_graymap[i * cols + j] = get_new_byte(j,i, filter_size)
9       ;
10      }
11    }
12    for(i=0; i < rows; i++){
13      for(j=0; j < cols ; j++) {
14        graymap[i * cols + j] = temp_graymap[i * cols + j];
15      }
16    }
17 }

```

List of Code 13: repeat_run

This line of code is for do the filtering repeatedly.

3.1.4 get_new_byte Gaussian Filter Version

```

1 int get_new_byte(int col, int row, int filter_size){
2   int temp = 0;
3   int start_col = col - (filter_size/2);
4   int start_row = row - (filter_size/2);
5   for(int i = 0; i < filter_size; i++){
6     for(int j = 0; j < filter_size; j++){
7       temp += get_byte_by_pos(start_col + i, start_row + j) *
8         filter[i * filter_size + j];
9     }
10   }
11   return temp / divider;
12 }

```

List of Code 14: get_new_byte Gaussian Filter Version

This line of code is for get the new pixel based on Gaussian Filter that has been created before.

3.1.5 get_new_byte Median Filter Version

```

1 int get_new_byte(int col, int row, int size_filter){
2   int temp[size_filter*size_filter];
3   int i, j;
4   for(i = -(size_filter/2); i <= (size_filter/2); i++){
5     for(j= -(size_filter/2); j <= (size_filter/2); j++){
6       temp[i * size_filter + j] = get_byte_by_pos(col-j, row-i);
7     }
8   }
9
10  for(i = 1; i < size_filter*size_filter; i++){
11    for(j=0; j < i ; j++){
12      if(temp[j] > temp[j+1]){
13        int x = temp[j];
14        temp[j] = temp[j+1];
15        temp[j+1] = x;
16      }

```

```
17     }
18 }
```

List of Code 15: get_new_byte Median Filter Version

This line of code is for get the new pixel based on Median Filter, where it will take the byte in the specific coordinate then sort it then take the median of sorted area.

3.1.6 clear_histogram

```
1 void clear_histogram(int maxval){
2     for(int i = 0; i <= maxval; i++){
3         histogram[i] = 0;
4     }
5 }
```

List of Code 16: clear_histogram

This line of code is for create an empty histogram array.

3.1.7 create_histogram

```
1 void create_histogram(int col, int row, int maxval){
2     clear_histogram(maxval);
3     int i,j;
4
5     for(i=0; i < rows; i++){
6         for(j=0; j < cols ; j++){
7             histogram[graymap[i * cols + j]]++;
8         }
9     }
10 }
```

List of Code 17: create_histogram

This line of code is for create the histogram array.

3.1.8 getMaxMin

```
1 void getMaxMin(int col, int row){
2     max = __INT_MAX__ * -1;
3     min = __INT_MAX__;
4     int i,j;
5
6     for(i=0; i < rows; i++){
7         for(j=0; j < cols ; j++){
8             int x = graymap[i * cols + j];
9             if(max < x)
10                 max = x;
11             if(min > x)
12                 min = x;
13         }
14     }
15 }
```

List of Code 18: getMaxMin

This line of code is for define the maximum and the minimum of value in the histogram.

3.1.9 histogram_stretching

```

1 void histogram_stretching(int col, int row, int maxval){
2     create_histogram(col, row, maxval);
3     getMaxMin(col, row);
4     int i, j;
5
6     for(i=0; i < rows; i++){
7         for(j=0; j < cols ; j++){
8             graymap[i * cols + j] = maxval * (graymap[i * cols + j] - min
9                 ) / (max-min);
10            }
11        }

```

List of Code 19: histogram_stretching

This line of code is for image improvement using histogram linear transformation or histogram stretching.

3.1.10 make_Fi

```

1 void make_Fi(int maxval){
2     Fi = (int *) malloc(sizeof(int) * (maxval + 2));
3     int temp = 0;
4     for(int i = 0; i <= maxval; i++){
5         temp += histogram[i];
6         Fi[i] = temp;
7     }
8 }

```

List of Code 20: make_Fi

This line of code is for creating the F(i) values or $\sum_{j=0}^{P_i} hist(j)$.

3.1.11 histogram_equalization

```

1 void histogram_equalization(int col, int row, int maxval){
2     create_histogram(col, row, maxval);
3     make_Fi(maxval);
4
5     int total_pixel = row * col;
6     int i, j;
7
8     for(i=0; i < rows; i++){
9         for(j=0; j < cols ; j++){
10             graymap[i * cols + j] = maxval * Fi[graymap[i * cols + j]]/
11                 total_pixel;
12            }
13        }

```

List of Code 21: histogram_equalization

This line of code is for image improvement using histogram non linear transformation or histogram equalization.

3.2 Question

3.2.1 Exercise 1 : Filtering

- Write a c program that reads a PGM image and compute the image smoothed with the following binomial filter:

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include "Util.h"
4
5 int rows, cols;
6 gray *graymap;
7
8 int get_byte_by_pos(int col, int row){
9     if(row < 0)
10         row = 0;
11     if(col < 0)
12         col = 0;
13     if(col >= cols)
14         col = cols-1;
15     if(row >= rows)
16         row = rows-1;
17
18     return graymap[row * cols + col];
19 }
20
21 int get_new_byte(int col, int row){
22     int temp = 0;
23     temp += get_byte_by_pos(col-1, row-1) * 1;
24     temp += get_byte_by_pos(col-1, row) * 2;
25     temp += get_byte_by_pos(col-1, row+1) * 1;
26
27     temp += get_byte_by_pos(col, row-1) * 2;
28     temp += get_byte_by_pos(col, row) * 4;
29     temp += get_byte_by_pos(col, row+1) * 2;
30
31     temp += get_byte_by_pos(col+1, row-1) * 1;
32     temp += get_byte_by_pos(col+1, row) * 2;
33     temp += get_byte_by_pos(col+1, row+1) * 1;
34
35     return temp / 16;
36 }
37
38 void repeat_run(int total_runs){
39     gray temp_graymap[cols * rows];
40     int k;
41     int i, j;
42     for(k = 0; k < total_runs; k++){
43         for(i=0; i < rows; i++){
44             for(j=0; j < cols ; j++){
45                 temp_graymap[i * cols + j] = get_new_byte(j,i);
46             }
47         }
48         for(i=0; i < rows; i++){
49             for(j=0; j < cols ; j++){
50                 graymap[i * cols + j] = temp_graymap[i * cols + j];
51             }
52     }
53 }
```

```

52     }
53 }
54 }
55
56 int main(int argc, char* argv[]){
57     FILE* ifp;
58     int ich1, ich2, maxval=255, pgmraw, total_runs;
59     int i, j;
60
61     /*
62      Argument Handler
63     */
64     if ( argc != 3 ){
65         printf("\nUsage: %s file repetition \n\n", argv[0]);
66         exit(0);
67     }
68
69     ifp = fopen(argv[1], "r");
70     if (ifp == NULL) {
71         printf("error in opening file %s\n", argv[1]);
72         exit(1);
73     }
74
75     total_runs = atoi(argv[2]);
76     if(total_runs < 0){
77         printf("Filter need to run at least once\n");
78         exit(1);
79     }
80
81     ich1 = getc( ifp );
82     if ( ich1 == EOF )
83         pm_erreur( "EOF / read error / magic number" );
84     ich2 = getc( ifp );
85     if ( ich2 == EOF )
86         pm_erreur( "EOF /read error / magic number" );
87     if(ich2 != '2' && ich2 != '5')
88         pm_erreur(" wrong file type ");
89     else
90         if(ich2 == '2')
91             pgmraw = 0;
92         else
93             pgmraw = 1;
94
95     cols = pm_getint( ifp );
96     rows = pm_getint( ifp );
97     maxval = pm_getint( ifp );
98
99     graymap = (gray *) malloc(cols * rows * sizeof(gray));
100
101    for(i=0; i < rows; i++)
102        for(j=0; j < cols ; j++)
103            if(pgmraw)
104                graymap[i * cols + j] = pm_getrawbyte(ifp) ;
105            else
106                graymap[i * cols + j] = pm_getint(ifp);
107
108    /*

```

```

109     Processing
110     */
111     repeat_run(total_runs);
112
113     if(pgmraw)
114         printf("P2\n");
115     else
116         printf("P5\n");
117
118     printf("%d %d \n", cols, rows);
119     printf("%d\n",maxval);
120
121     for(i=0; i < rows; i++){
122         for(j=0; j < cols ; j++){
123             if(pgmraw){
124                 printf("%d ", graymap[i * cols + j]);
125             }
126             else{
127                 printf("%c", graymap[i * cols + j]);
128             }
129         }
130     }
131
132     fclose(ifp);
133     return 0;
134 }
```

List of Code 22: filtering_3x3

- Add the option to smooth n times as well as the option to smooth with a filter of dimension 5x5 (binomial coefficients can be obtained from the pascal triangle).

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include "Util.h"
4
5 int rows, cols;
6 gray *graymap;
7
8 int get_byte_by_pos(int col, int row){
9     if(row < 0)
10         row = 0;
11     if(col < 0)
12         col = 0;
13     if(col >= cols)
14         col = cols-1;
15     if(row >= rows)
16         row = rows-1;
17
18     return graymap[row * cols + col];
19 }
20
21 int get_new_byte(int col, int row){
22     int temp = 0;
23     temp += get_byte_by_pos(col-2,row-2) * 1;
24     temp += get_byte_by_pos(col-2,row-1) * 4;
```

```

25     temp += get_byte_by_pos(col-2, row) * 6;
26     temp += get_byte_by_pos(col-2, row+1) * 4;
27     temp += get_byte_by_pos(col-2, row+2) * 1;
28
29     temp += get_byte_by_pos(col-1, row-2) * 4;
30     temp += get_byte_by_pos(col-1, row-1) * 16;
31     temp += get_byte_by_pos(col-1, row) * 24;
32     temp += get_byte_by_pos(col-1, row+1) * 16;
33     temp += get_byte_by_pos(col-1, row+2) * 4;
34
35     temp += get_byte_by_pos(col, row-2) * 6;
36     temp += get_byte_by_pos(col, row-1) * 24;
37     temp += get_byte_by_pos(col, row) * 36;
38     temp += get_byte_by_pos(col, row+1) * 24;
39     temp += get_byte_by_pos(col, row+2) * 6;
40
41     temp += get_byte_by_pos(col+1, row-2) * 4;
42     temp += get_byte_by_pos(col+1, row-1) * 16;
43     temp += get_byte_by_pos(col+1, row) * 24;
44     temp += get_byte_by_pos(col+1, row+1) * 16;
45     temp += get_byte_by_pos(col+1, row+2) * 4;
46
47     temp += get_byte_by_pos(col+2, row-2) * 1;
48     temp += get_byte_by_pos(col+2, row-1) * 4;
49     temp += get_byte_by_pos(col+2, row) * 6;
50     temp += get_byte_by_pos(col+2, row+1) * 4;
51     temp += get_byte_by_pos(col+2, row+2) * 1;
52
53     return temp / 256;
54 }
55
56 void repeat_run(int total_runs){
57     gray temp_graymap[cols * rows];
58     int k;
59     int i, j;
60     for(k = 0; k < total_runs; k++){
61         for(i=0; i < rows; i++){
62             for(j=0; j < cols ; j++){
63                 temp_graymap[i * cols + j] = get_new_byte(j,i);
64             }
65         }
66         for(i=0; i < rows; i++){
67             for(j=0; j < cols ; j++){
68                 graymap[i * cols + j] = temp_graymap[i * cols + j];
69             }
70         }
71     }
72 }
73
74 int main(int argc, char* argv[]){
75     FILE* ifp;
76     int ich1, ich2, maxval=255, pgmraw, total_runs;
77     int i, j;
78
79     /*
80      Argument Handler
81     */

```

```

82     if ( argc != 3 ){
83         printf("\nUsage: %s file repetition\n\n", argv[0]);
84         exit(0);
85     }
86
87     ifp = fopen(argv[1], "r");
88     if (ifp == NULL) {
89         printf("error in opening file %s\n", argv[1]);
90         exit(1);
91     }
92
93     total_runs = atoi(argv[2]);
94     if(total_runs < 0){
95         printf("Filter need to run at least once\n");
96         exit(1);
97     }
98
99     ich1 = getc(ifp);
100    if ( ich1 == EOF )
101        pm_erreur( "EOF / read error / magic number" );
102    ich2 = getc(ifp);
103    if ( ich2 == EOF )
104        pm_erreur( "EOF /read error / magic number" );
105    if(ich2 != '2' && ich2 != '5')
106        pm_erreur(" wrong file type ");
107    else
108        if(ich2 == '2')
109            pgmraw = 0;
110        else
111            pgmraw = 1;
112
113    cols = pm_getint(ifp);
114    rows = pm_getint(ifp);
115    maxval = pm_getint(ifp);
116
117    graymap = (gray *) malloc(cols * rows * sizeof(gray));
118
119    for(i=0; i < rows; i++)
120        for(j=0; j < cols ; j++)
121            if(pgmraw)
122                graymap[i * cols + j] = pm_getrawbyte(ifp) ;
123            else
124                graymap[i * cols + j] = pm_getint(ifp);
125
126    /*
127     Processing
128    */
129    repeat_run(total_runs);
130
131    if(pgmraw)
132        printf("P2\n");
133    else
134        printf("P5\n");
135
136    printf("%d %d \n", cols, rows);
137    printf("%d\n",maxval);
138

```

```

139     for(i=0; i < rows; i++){
140         for(j=0; j < cols ; j++){
141             if(pgmraw){
142                 printf("%d ", graymap[i * cols + j]);
143             }
144             else{
145                 printf("%c", graymap[i * cols + j]);
146             }
147         }
148     }
149
150     fclose(ifp);
151     return 0;
152 }
```

List of Code 23: filtering_5x5

- Implement a median filter and compare it with the binomial filters.

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include "Util.h"
4
5 int rows, cols;
6 gray *graymap;
7
8 int get_byte_by_pos(int col, int row){
9     if(row < 0)
10         row = 0;
11     if(col < 0)
12         col = 0;
13     if(col >= cols)
14         col = cols-1;
15     if(row >= rows)
16         row = rows-1;
17
18     return graymap[row * cols + col];
19 }
20
21 int get_new_byte(int col, int row, int size_filter){
22     int temp[size_filter*size_filter];
23     int i, j;
24     for(i = -(size_filter/2); i <= (size_filter/2); i++){
25         for(j= -(size_filter/2); j <= (size_filter/2); j++){
26             temp[i * size_filter + j] = get_byte_by_pos(col-j, row-i);
27         }
28     }
29
30     for(i = 1; i < size_filter*size_filter; i++){
31         for(j=0; j < i ; j++){
32             if(temp[j] > temp[j+1]){
33                 int x = temp[j];
34                 temp[j] = temp[j+1];
35                 temp[j+1] = x;
36             }
37         }
38     }
}
```

```

39     return temp[size_filter*size_filter/2];
40 }
41
42 void repeat_run(int total_runs, int size_filter){
43     gray temp_graymap[cols * rows];
44     int k;
45     int i, j;
46     for(k = 0; k < total_runs; k++){
47         for(i=0; i < rows; i++){
48             for(j=0; j < cols ; j++){
49                 temp_graymap[i * cols + j] = get_new_byte(j,i,
50 size_filter);
51             }
52         }
53         for(i=0; i < rows; i++){
54             for(j=0; j < cols ; j++){
55                 graymap[i * cols + j] = temp_graymap[i * cols + j];
56             }
57         }
58     }
59 }
60
61 int main(int argc, char* argv[]){
62     FILE* ifp;
63     int ich1, ich2, maxval=255, pgmraw, total_runs, size_filter;
64     int i, j;
65
66     /*
67      Argument Handler
68     */
69     if ( argc != 4 ){
70         printf("\nUsage: %s file \n\n", argv[0]);
71         exit(0);
72     }
73
74     ifp = fopen(argv[1], "r");
75     if (ifp == NULL) {
76         printf("error in opening file %s\n", argv[1]);
77         exit(1);
78     }
79
80     size_filter = atoi(argv[2]);
81     if(size_filter % 2 == 0 || size_filter <= 0){
82         printf("error in size of filter %s\n", argv[2]);
83         printf("expecting %s is greater than 0 and an odd number\n",
84               argv[3]);
85         exit(1);
86     }
87
88     total_runs = atoi(argv[3]);
89     if(total_runs < 0){
90         printf("Filter need to run at least once\n");
91         exit(1);
92     }
93     ich1 = getc( ifp );

```

```

94     if ( ich1 == EOF )
95         pm_erreur( "EOF / read error / magic number" );
96     ich2 = getc( ifp );
97     if ( ich2 == EOF )
98         pm_erreur( "EOF /read error / magic number" );
99     if(ich2 != '2' && ich2 != '5')
100        pm_erreur(" wrong file type ");
101    else
102        if(ich2 == '2')
103            pgmraw = 0;
104        else
105            pgmraw = 1;
106
107    cols = pm_getint( ifp );
108    rows = pm_getint( ifp );
109    maxval = pm_getint( ifp );
110
111    graymap = (gray *) malloc(cols * rows * sizeof(gray));
112
113    for(i=0; i < rows; i++)
114        for(j=0; j < cols ; j++)
115            if(pgmraw)
116                graymap[i * cols + j] = pm_getrawbyte(ifp) ;
117            else
118                graymap[i * cols + j] = pm_getint(ifp);
119
120    /*
121     Processing
122    */
123    repeat_run(total_runs, size_filter);
124
125    if(pgmraw)
126        printf("P2\n");
127    else
128        printf("P5\n");
129
130    printf("%d %d \n", cols, rows);
131    printf("%d\n",maxval);
132
133    for(i=0; i < rows; i++){
134        for(j=0; j < cols ; j++){
135            if(pgmraw){
136                printf("%d ", graymap[i * cols + j]);
137            }
138            else{
139                printf("%c", graymap[i * cols + j]);
140            }
141        }
142    }
143
144    fclose(ifp);
145    return 0;
146 }
```

List of Code 24: filter_median

Bonus modified filtering universal

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include "Util.h"
4
5 int rows, cols;
6 gray *graymap;
7 int *filter;
8 int divider;
9
10 int get_byte_by_pos(int col, int row){
11     if(row < 0)
12         row = 0;
13     if(col < 0)
14         col = 0;
15     if(col >= cols)
16         col = cols-1;
17     if(row >= rows)
18         row = rows-1;
19
20     return graymap[row * cols + col];
21 }
22
23 /*
24 Create Pascal triangle
25 Courtesy: https://www.faceprep.in/c/program-to-print-pascals-triangle/
26 */
27 void createFilter(int filter_size){
28     int coef, i, j, temp_pascal[filter_size+2];
29     divider = 0;
30     for(i = 0; i<= filter_size; i++){
31         if(i==0)
32             coef = 1;
33         else
34             coef = coef * (filter_size-i)/i;
35
36         temp_pascal[i] = coef;
37     }
38
39     filter = malloc(sizeof(int) * filter_size * filter_size);
40     for(i = 0; i < filter_size; i++){
41         for(j = 0; j < filter_size; j++){
42             int x = temp_pascal[i] * temp_pascal[j];
43             filter[i*filter_size+j] = x;
44             divider += x;
45         }
46     }
47 }
48
49 int get_new_byte(int col, int row, int filter_size){
50     int temp = 0;
51     int start_col = col - (filter_size/2);
52     int start_row = row - (filter_size/2);
53     for(int i = 0; i < filter_size; i++){
54         for(int j = 0; j < filter_size; j++){
55             temp += get_byte_by_pos(start_col + i, start_row + j) *

```

```

57     filter[i *filter_size + j];
58 }
59
60     return temp / divider;
61 }
62
63 void repeat_run(int total_runs, int filter_size){
64     gray temp_graymap[cols * rows];
65     int k;
66     int i, j;
67     for(k = 0; k < total_runs; k++){
68         for(i=0; i < rows; i++){
69             for(j=0; j < cols ; j++){
70                 temp_graymap[i * cols + j] = get_new_byte(j,i,
71                     filter_size);
72             }
73         }
74         for(i=0; i < rows; i++){
75             for(j=0; j < cols ; j++){
76                 graymap[i * cols + j] = temp_graymap[i * cols + j];
77             }
78         }
79     }
80
81 int main(int argc, char* argv[]){
82     FILE* ifp;
83     int ich1, ich2, maxval=255, pgmraw, filter_size, total_runs;
84     int i, j;
85
86     /*
87      Argument Handler
88     */
89     if ( argc != 4 ){
90         printf("\nUsage: %s file filter_size repetition \n\n",
91             argv[0]);
92         exit(0);
93     }
94     ifp = fopen(argv[1],"r");
95     if (ifp == NULL) {
96         printf("error in opening file %s\n", argv[1]);
97         exit(1);
98     }
99
100    filter_size = atoi(argv[2]);
101    if(filter_size < 0 || filter_size %2 != 1){
102        printf("error in size of filter %s\n", argv[2]);
103        printf("expecting %s is greater than 0 and an odd number\n",
104            argv[3]);
105        exit(1);
106    }
107    total_runs = atoi(argv[3]);
108    if(total_runs < 0){
109        printf("Filter need to run at least once\n");

```

```

110     exit(1);
111 }
112
113 ich1 = getc( ifp );
114 if ( ich1 == EOF )
115     pm_erreur( "EOF / read error / magic number" );
116 ich2 = getc( ifp );
117 if ( ich2 == EOF )
118     pm_erreur( "EOF /read error / magic number" );
119 if(ich2 != '2' && ich2 != '5')
120     pm_erreur(" wrong file type ");
121 else
122     if(ich2 == '2')
123         pgmraw = 0;
124     else
125         pgmraw = 1;
126
127 cols = pm_getint( ifp );
128 rows = pm_getint( ifp );
129 maxval = pm_getint( ifp );
130
131 graymap = (gray *) malloc(cols * rows * sizeof(gray));
132
133 for(i=0; i < rows; i++)
134     for(j=0; j < cols ; j++)
135         if(pgmraw)
136             graymap[i * cols + j] = pm_getrawbyte(ifp) ;
137         else
138             graymap[i * cols + j] = pm_getint(ifp);
139
140 /*
141     Processing
142 */
143 createFilter(filter_size);
144 repeat_run(total_runs , filter_size);
145
146 if(pgmraw)
147     printf("P2\n");
148 else
149     printf("P5\n");
150
151 printf("%d %d \n", cols, rows);
152 printf("%d\n",maxval);
153
154 for(i=0; i < rows; i++){
155     for(j=0; j < cols ; j++){
156         if(pgmraw){
157             printf("%d ", graymap[i * cols + j]);
158         }
159         else{
160             printf("%c", graymap[i * cols + j]);
161         }
162     }
163 }
164
165 fclose(ifp);
166 return 0;

```

```
167 }
```

List of Code 25: filtering

3.2.2 Exercise 2 : Histograms

- Add to the previous program a function that computes the intensity histogram. This histogram will be represented by an array of dimension the maximal number of grayscale intensities (i.e. 255).

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include "Util.h"
5
6 int rows, cols;
7 gray *graymap;
8 int *filter;
9 int divider;
10 int *histogram;
11 int *Fi;
12 int max = __INT_MAX__ * -1;
13 int min = __INT_MAX__;
14
15 int get_byte_by_pos(int col, int row){
16     if(row < 0)
17         row = 0;
18     if(col < 0)
19         col = 0;
20     if(col >= cols)
21         col = cols-1;
22     if(row >= rows)
23         row = rows-1;
24
25     return graymap[row * cols + col];
26 }
27
28 void clear_histogram(int maxval){
29     for(int i = 0; i <= maxval; i++){
30         histogram[i] = 0;
31     }
32 }
33
34 void create_histogram(int col, int row, int maxval){
35     clear_histogram(maxval);
36     int i,j;
37
38     for(i=0; i < rows; i++){
39         for(j=0; j < cols ; j++){
40             histogram[graymap[i * cols + j]]++;
41         }
42     }
43 }
44 /*
45 Create Pascal triangle
46 Courtesy: https://www.faceprep.in/c/program-to-print-pascals-triangle/

```

```

47 */
48 void createFilter(int filter_size){
49     int coef, i, j, temp_pascal[filter_size+2];
50     divider = 0;
51     for(i = 0; i <= filter_size; i++){
52         if(i==0)
53             coef = 1;
54         else
55             coef = coef * (filter_size-i)/i;
56
57         temp_pascal[i] = coef;
58     }
59
60     filter = malloc(sizeof(int) * filter_size * filter_size);
61     for(i = 0; i < filter_size; i++){
62         for(j = 0; j < filter_size; j++){
63             int x = temp_pascal[i] * temp_pascal[j];
64             filter[i*filter_size+j] = x;
65             divider += x;
66         }
67     }
68 }
69
70
71 int get_new_byte(int col, int row, int filter_size){
72     int temp = 0;
73     int start_col = col - (filter_size/2);
74     int start_row = row - (filter_size/2);
75     for(int i = 0; i < filter_size; i++){
76         for(int j = 0; j < filter_size; j++){
77             temp += get_byte_by_pos(start_col + i, start_row + j) *
78                 filter[i * filter_size + j];
79         }
80     }
81
82     return temp / divider;
83 }
84
85 void repeat_run(int total_runs, int filter_size){
86     gray temp_graymap[cols * rows];
87     int k;
88     int i, j;
89     for(k = 0; k < total_runs; k++){
90         for(i=0; i < rows; i++){
91             for(j=0; j < cols ; j++){
92                 temp_graymap[i * cols + j] = get_new_byte(j,i,
93                     filter_size);
94             }
95         }
96         for(i=0; i < rows; i++){
97             for(j=0; j < cols ; j++){
98                 graymap[i * cols + j] = temp_graymap[i * cols + j];
99             }
100    }
101 }
```

```

102 int main(int argc, char* argv[]){
103     FILE* ifp;
104     int ich1, ich2, maxval=255, pgmraw, filter_size, total_runs;
105     int i, j;
106     char *mode;
107
108     /*
109      Argument Handler
110     */
111     if ( argc != 5 ){
112         printf("\nUsage: %s file filter_size repetition mode\n\n",
113               argv[0]);
114         exit(0);
115     }
116
117     ifp = fopen(argv[1], "r");
118     if (ifp == NULL) {
119         printf("error in opening file %s\n", argv[1]);
120         exit(1);
121     }
122
123     filter_size = atoi(argv[2]);
124     if(filter_size < 0 || filter_size %2 != 1){
125         printf("error in size of filter %s\n", argv[2]);
126         printf("expecting %s is greater than 0 and an odd number\n",
127               argv[3]);
128         exit(1);
129     }
130
131     total_runs = atoi(argv[3]);
132     if(total_runs < 0){
133         printf("Filter need to run at least once\n");
134         exit(1);
135     }
136
137     mode = argv[4];
138     if(strcmp(mode, "e") != 0 && strcmp(mode, "s") != 0){
139         printf("Filter mode must be s[Stretching] or e[Equalization]\n");
140         exit(1);
141     }
142
143     ich1 = getc(ifp);
144     if ( ich1 == EOF )
145         pm_erreur( "EOF / read error / magic number" );
146     ich2 = getc(ifp);
147     if ( ich2 == EOF )
148         pm_erreur( "EOF /read error / magic number" );
149     if(ich2 != '2' && ich2 != '5')
150         pm_erreur(" wrong file type ");
151     else
152         if(ich2 == '2')
153             pgmraw = 0;
154         else
155             pgmraw = 1;
156
157     cols = pm_getint(ifp);

```

```

156 rows = pm_getint(ifp);
157 maxval = pm_getint(ifp);
158
159 graymap = (gray *) malloc(cols * rows * sizeof(gray));
160 histogram = (int *) malloc((maxval+1) * sizeof(int));
161
162 for(i=0; i < rows; i++)
163     for(j=0; j < cols ; j++)
164         if(pgmrw)
165             graymap[i * cols + j] = pm_getrawbyte(ifp) ;
166         else
167             graymap[i * cols + j] = pm_getint(ifp);
168
169 /*
170     Processing
171 */
172 createFilter(filter_size);
173 repeat_run(total_runs, filter_size);
174
175 if(pgmrw)
176     printf("P2\n");
177 else
178     printf("P5\n");
179
180 printf("%d %d \n", cols, rows);
181 printf("%d\n",maxval);
182
183 for(i=0; i < rows; i++){
184     for(j=0; j < cols ; j++){
185         if(pgmrw){
186             printf("%d ", graymap[i * cols + j]);
187         }
188         else{
189             printf("%c ", graymap[i * cols + j]);
190         }
191     }
192 }
193
194 fclose(ifp);
195 return 0;

```

List of Code 26: histogram

- Modify the program so that it computes the image transformed with a histogram stretching.

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include "Util.h"
5
6 int rows, cols;
7 gray *graymap;
8 int *filter;
9 int divider;
10 int *histogram;
11 int *Fi;

```

```

12 int max = __INT_MAX__ * -1;
13 int min = __INT_MAX__;
14
15 int get_byte_by_pos(int col, int row){
16     if(row < 0)
17         row = 0;
18     if(col < 0)
19         col = 0;
20     if(col >= cols)
21         col = cols-1;
22     if(row >= rows)
23         row = rows-1;
24
25     return graymap[row * cols + col];
26 }
27
28 void clear_histogram(int maxval){
29     for(int i = 0; i <= maxval; i++){
30         histogram[i] = 0;
31     }
32 }
33
34 void create_histogram(int col, int row, int maxval){
35     clear_histogram(maxval);
36     int i,j;
37
38     for(i=0; i < rows; i++){
39         for(j=0; j < cols ; j++){
40             histogram[graymap[i * cols + j]]++;
41         }
42     }
43 }
44
45 void getMaxMin(int col, int row){
46     max = __INT_MAX__ * -1;
47     min = __INT_MAX__;
48     int i,j;
49
50     for(i=0; i < rows; i++){
51         for(j=0; j < cols ; j++){
52             int x = graymap[i * cols + j];
53             if(max < x)
54                 max = x;
55             if(min > x)
56                 min = x;
57         }
58     }
59 }
60
61 void histogram_stretching(int col, int row, int maxval){
62     create_histogram(col, row, maxval);
63     getMaxMin(col, row);
64     int i, j;
65
66     for(i=0; i < rows; i++){
67         for(j=0; j < cols ; j++){
68             graymap[i * cols + j] = maxval * (graymap[i * cols + j]

```

```

        - min) / (max-min);
69     }
70 }
71 /*
72 Create Pascal triangle
73 Courtesy: https://www.faceprep.in/c/program-to-print-pascals-triangle/
74 */
75
76 void createFilter(int filter_size){
77     int coef, i, j, temp_pascal[filter_size+2];
78     divider = 0;
79     for(i = 0; i <= filter_size; i++){
80         if(i==0)
81             coef = 1;
82         else
83             coef = coef * (filter_size-i)/i;
84
85         temp_pascal[i] = coef;
86     }
87
88     filter = malloc(sizeof(int) * filter_size * filter_size);
89     for(i = 0; i < filter_size; i++){
90         for(j = 0; j < filter_size; j++){
91             int x = temp_pascal[i] * temp_pascal[j];
92             filter[i*filter_size+j] = x;
93             divider += x;
94         }
95     }
96 }
97
98
99 int get_new_byte(int col, int row, int filter_size){
100    int temp = 0;
101    int start_col = col - (filter_size/2);
102    int start_row = row - (filter_size/2);
103    for(int i = 0; i < filter_size; i++){
104        for(int j = 0; j < filter_size; j++){
105            temp += get_byte_by_pos(start_col + i, start_row + j) *
106                  filter[i * filter_size + j];
107        }
108    }
109
110    return temp / divider;
111 }
112
113 void repeat_run(int total_runs, int filter_size){
114     gray temp_graymap[cols * rows];
115     int k;
116     int i, j;
117     for(k = 0; k < total_runs; k++){
118         for(i=0; i < rows; i++){
119             for(j=0; j < cols ; j++){
120                 temp_graymap[i * cols + j] = get_new_byte(j,i,
121                     filter_size);
122             }
123         }
124     }

```

```

122     for(i=0; i < rows; i++){
123         for(j=0; j < cols ; j++){
124             graymap[i * cols + j] = temp_graymap[i * cols + j];
125         }
126     }
127 }
128 }
129
130 int main(int argc, char* argv[]){
131     FILE* ifp;
132     int ich1, ich2, maxval=255, pgmraw, filter_size, total_runs;
133     int i, j;
134     char *mode;
135
136     /*
137      Argument Handler
138     */
139     if ( argc != 5 ){
140         printf("\nUsage: %s file filter_size repetition mode\n\n",
141               argv[0]);
142         exit(0);
143     }
144     ifp = fopen(argv[1], "r");
145     if (ifp == NULL) {
146         printf("error in opening file %s\n", argv[1]);
147         exit(1);
148     }
149     filter_size = atoi(argv[2]);
150     if(filter_size < 0 || filter_size %2 != 1){
151         printf("error in size of filter %s\n", argv[2]);
152         printf("expecting %s is greater than 0 and an odd number\n",
153               argv[3]);
154         exit(1);
155     }
156     total_runs = atoi(argv[3]);
157     if(total_runs < 0){
158         printf("Filter need to run at least once\n");
159         exit(1);
160     }
161
162     mode = argv[4];
163     if(strcmp(mode, "e") != 0 && strcmp(mode, "s") != 0){
164         printf("Filter mode must be s[Stretching] or e[
165             Equalization]\n");
166         exit(1);
167     }
168
169     ich1 = getc( ifp );
170     if ( ich1 == EOF )
171         pm_erreur( "EOF / read error / magic number" );
172     ich2 = getc( ifp );
173     if ( ich2 == EOF )
174         pm_erreur( "EOF /read error / magic number" );
175     if(ich2 != '2' && ich2 != '5')

```

```

176     pm_erreur(" wrong file type ");
177 else
178     if(ich2 == '2')
179         pgmraw = 0;
180     else
181         pgmraw = 1;
182
183 cols = pm_getint(ifp);
184 rows = pm_getint(ifp);
185 maxval = pm_getint(ifp);
186
187 graymap = (gray *) malloc(cols * rows * sizeof(gray));
188 histogram = (int *) malloc((maxval+1) * sizeof(int));
189
190 for(i=0; i < rows; i++)
191     for(j=0; j < cols ; j++)
192         if(pgmraw)
193             graymap[i * cols + j] = pm_getrawbyte(ifp) ;
194         else
195             graymap[i * cols + j] = pm_getint(ifp);
196
197 /*
198     Processing
199 */
200 createFilter(filter_size);
201 repeat_run(total_runs , filter_size);
202
203 if(strcmp(mode, "s") == 0){
204     histogram_stretching(cols, rows, maxval);
205 }
206
207 if(pgmraw)
208     printf("P2\n");
209 else
210     printf("P5\n");
211
212 printf("%d %d \n", cols, rows);
213 printf("%d\n",maxval);
214
215 for(i=0; i < rows; i++){
216     for(j=0; j < cols ; j++){
217         if(pgmraw){
218             printf("%d ", graymap[i * cols + j]);
219         }
220         else{
221             printf("%c", graymap[i * cols + j]);
222         }
223     }
224 }
225
226 fclose(ifp);
227 return 0;

```

List of Code 27: histogram_stretching

- Modify the program so that it computes the image transformed with a histogram equalization.

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include "Util.h"
5
6 int rows, cols;
7 gray *graymap;
8 int *filter;
9 int divider;
10 int *histogram;
11 int *Fi;
12 int max = __INT_MAX__ * -1;
13 int min = __INT_MAX__;
14
15 int get_byte_by_pos(int col, int row){
16     if(row < 0)
17         row = 0;
18     if(col < 0)
19         col = 0;
20     if(col >= cols)
21         col = cols-1;
22     if(row >= rows)
23         row = rows-1;
24
25     return graymap[row * cols + col];
26 }
27
28 void clear_histogram(int maxval){
29     for(int i = 0; i <= maxval; i++){
30         histogram[i] = 0;
31     }
32 }
33
34 void create_histogram(int col, int row, int maxval){
35     clear_histogram(maxval);
36     int i,j;
37
38     for(i=0; i < rows; i++){
39         for(j=0; j < cols ; j++){
40             histogram[graymap[i * cols + j]]++;
41         }
42     }
43 }
44
45 void getMaxMin(int col, int row){
46     max = __INT_MAX__ * -1;
47     min = __INT_MAX__;
48     int i,j;
49
50     for(i=0; i < rows; i++){
51         for(j=0; j < cols ; j++){
52             int x = graymap[i * cols + j];
53             if(max < x)
54                 max = x;
55             if(min > x)
56                 min = x;
57         }

```

```

58     }
59 }
60
61 void histogram_stretching(int col, int row, int maxval){
62     create_histogram(col, row, maxval);
63     getMaxMin(col, row);
64     int i, j;
65
66     for(i=0; i < rows; i++){
67         for(j=0; j < cols ; j++){
68             graymap[i * cols + j] = maxval * (graymap[i * cols + j]
69             - min) / (max-min);
70         }
71     }
72
73 void make_Fi(int maxval){
74     Fi = (int *) malloc(sizeof(int) * (maxval + 2));
75     int temp = 0;
76     for(int i = 0; i <= maxval; i++){
77         temp += histogram[i];
78         Fi[i] = temp;
79     }
80 }
81
82 void histogram_equalization(int col, int row, int maxval){
83     create_histogram(col, row, maxval);
84     make_Fi(maxval);
85
86     int total_pixel = row * col;
87     int i, j;
88
89     for(i=0; i < rows; i++){
90         for(j=0; j < cols ; j++){
91             graymap[i * cols + j] = maxval * Fi[graymap[i * cols + j
92             ]]/total_pixel;
93         }
94     }
95
96 /*
97 Create Pascal triangle
98 Courtesy: https://www.faceprep.in/c/program-to-print-pascals-triangle/
99 */
100 void createFilter(int filter_size){
101     int coef, i, j, temp_pascal[filter_size+2];
102     divider = 0;
103     for(i = 0; i<= filter_size; i++){
104         if(i==0)
105             coef = 1;
106         else
107             coef = coef * (filter_size-i)/i;
108
109         temp_pascal[i] = coef;
110     }
111 }
```

```

112 filter = malloc(sizeof(int) * filter_size * filter_size);
113 for(i = 0; i < filter_size; i++){
114     for(j = 0; j < filter_size; j++){
115         int x = temp_pascal[i] * temp_pascal[j];
116         filter[i*filter_size+j] = x;
117         divider += x;
118     }
119 }
120 }
121
122 int get_new_byte(int col, int row, int filter_size){
123     int temp = 0;
124     int start_col = col - (filter_size/2);
125     int start_row = row - (filter_size/2);
126     for(int i = 0; i < filter_size; i++){
127         for(int j = 0; j < filter_size; j++){
128             temp += get_byte_by_pos(start_col + i, start_row + j) *
129                 filter[i * filter_size + j];
130         }
131     }
132
133     return temp / divider;
134 }
135
136 void repeat_run(int total_runs, int filter_size){
137     gray temp_graymap[cols * rows];
138     int k;
139     int i, j;
140     for(k = 0; k < total_runs; k++){
141         for(i=0; i < rows; i++){
142             for(j=0; j < cols ; j++){
143                 temp_graymap[i * cols + j] = get_new_byte(j,i,
144                     filter_size);
145             }
146         }
147         for(i=0; i < rows; i++){
148             for(j=0; j < cols ; j++){
149                 graymap[i * cols + j] = temp_graymap[i * cols + j];
150             }
151         }
152     }
153
154 int main(int argc, char* argv[]){
155     FILE* ifp;
156     int ich1, ich2, maxval=255, pgmraw, filter_size, total_runs;
157     int i, j;
158     char *mode;
159
160     /*
161      Argument Handler
162     */
163     if ( argc != 5 ){
164         printf("\nUsage: %s file filter_size repetition mode\n\n",
165             argv[0]);
166         exit(0);

```

```

166 }
167
168 ifp = fopen(argv[1], "r");
169 if (ifp == NULL) {
170     printf("error in opening file %s\n", argv[1]);
171     exit(1);
172 }
173
174 filter_size = atoi(argv[2]);
175 if(filter_size < 0 || filter_size %2 != 1){
176     printf("error in size of filter %s\n", argv[2]);
177     printf("expecting %s is greater than 0 and an odd number\n",
178           argv[3]);
179     exit(1);
180 }
181
182 total_runs = atoi(argv[3]);
183 if(total_runs < 0){
184     printf("Filter need to run at least once\n");
185     exit(1);
186 }
187
188 mode = argv[4];
189 if(strcmp(mode, "e") != 0 && strcmp(mode, "s") != 0){
190     printf("Filter mode must be s[Stretching] or e[Equalization]\n");
191     exit(1);
192 }
193
194 ich1 = getc(ifp);
195 if (ich1 == EOF)
196     pm_erreur( "EOF / read error / magic number" );
197 ich2 = getc(ifp);
198 if (ich2 == EOF)
199     pm_erreur( "EOF /read error / magic number" );
200 if(ich2 != '2' && ich2 != '5')
201     pm_erreur(" wrong file type ");
202 else
203     if(ich2 == '2')
204         pgmraw = 0;
205     else
206         pgmraw = 1;
207
208 cols = pm_getint(ifp);
209 rows = pm_getint(ifp);
210 maxval = pm_getint(ifp);
211
212 graymap = (gray *) malloc(cols * rows * sizeof(gray));
213 histogram = (int *) malloc((maxval+1) * sizeof(int));
214
215 for(i=0; i < rows; i++)
216     for(j=0; j < cols ; j++)
217         if(pgmraw)
218             graymap[i * cols + j] = pm_getrawbyte(ifp) ;
219         else
220             graymap[i * cols + j] = pm_getint(ifp);

```

```

221  /*
222   Processing
223 */
224 createFilter(filter_size);
225 repeat_run(total_runs, filter_size);
226
227 if(strcmp(mode, "s") == 0){
228     histogram_stretching(cols, rows, maxval);
229 }
230 else if(strcmp(mode, "e") == 0){
231     histogram_equalization(cols, rows, maxval);
232 }
233
234 if(pgmraw)
235     printf("P2\n");
236 else
237     printf("P5\n");
238
239 printf("%d %d \n", cols, rows);
240 printf("%d\n", maxval);
241
242 for(i=0; i < rows; i++){
243     for(j=0; j < cols ; j++){
244         if(pgmraw){
245             printf("%d ", graymap[i * cols + j]);
246         }
247         else{
248             printf("%c", graymap[i * cols + j]);
249         }
250     }
251 }
252
253 fclose(ifp);
254 return 0;
255 }
```

List of Code 28: histogram_equation

3.3 Result

3.3.1 Local Filters

In this section, we find the result of applying local filters on images. First, the 3x3 binomial filter is applied on the default image.

Binomial Filters



Figure 4: Applying a 3x3 binomial filter 1 time. (boat_3_1.png)

You can create the image using following code.

```
1 ./filtering_3x3 boat.pgm>boat_3_1.png 1
```

List of Code 29: Make Figure boat_3_1.png

or

```
1 ./filtering boat.pgm>boat_3_1.png 3 1
```

List of Code 30: Make Figure boat_3_1.png



Figure 5: Applying a 3x3 binomial filter 5 times. (boat_3_5.png)

You can create the image using following code.

```
1 ./filtering_3x3 boat.pgm>boat_3_5.png 5
```

List of Code 31: Make Figure boat_3_5.png

or

```
1 ./filtering boat.pgm>boat_3_5.png 3 5
```

List of Code 32: Make Figure boat_3_5.png

From the images above, we can observe that the boat image gets blurred over a 3x3 binomial filter.



Figure 6: Applying a 5x5 binomial filter 1 time. (boat_5_1.png)

You can create the image using following code.

```
1 ./filtering_5x5 boat.pgm>boat_5_1.png 1
```

List of Code 33: Make Figure boat_5_1.png

or

```
1 ./filtering boat.pgm>boat_5_1.png 5 1
```

List of Code 34: Make Figure boat_5_1.png



Figure 7: Applying the 5x5 binomial filter 5 times. (boat_5_5.png)

You can create the image using following code.

```
1 ./filtering_5x5 boat.pgm>boat_5_5.png 5
```

List of Code 35: Make Figure boat_5_5.png

or

```
1 ./filtering boat.pgm>boat_5_5.png 5 5
```

List of Code 36: Make Figure boat_5_5.png

After applying the 5x5 binomial filter, it can be noticed that blurring is stronger than the 3x3 binomial. We can see from the result images that the text ("ZENTIM") on the boat tend to be become unreadable after applying binomial filter. And, the corner details vanish after the filter.

Median Filter



Figure 8: Applying a 3x3 median filter 1 time. (boat_median_3_1.png)

You can create the image using following code.

```
1 ./filter_median boat.pgm>boat_median_3_1.png 3 1
```

List of Code 37: Make Figure boat_median_3_1.png



Figure 9: Applying a 3x3 median filter 5 times. (boat_median_3_5.png)

You can create the image using following code.

```
1 ./filter_median boat.pgm>boat_median_3_5.png 3 5
```

List of Code 38: Make Figure boat_median_3_5.png



Figure 10: Applying a 5x5 median filter 1 times. (boat_median_5_1.png)

You can create the image using following code.

```
1 ./filter_median boat.pgm>boat_median_5_1.png 5 1
```

List of Code 39: Make Figure boat_median_5_1.png



Figure 11: Applying a 5x5 median filter 3 times. (boat_median_5_3.png)

You can create the image using following code.

```
1 ./filter_median boat.pgm>boat_median_5_3.png 5 3
```

List of Code 40: Make Figure boat_median_5_3.png

From the result above, we can say that binomial and median filters are useful when removing noise from images. But, it should be noted that applying too much filter can lead to blur. In other words, filters may lose detailed information of the image, since the filters are replacing the value with neighbouring values.

3.3.2 Histogram Modifications

As it has been described on the task description, we have applied histogram modifications on images after binomial and median filter.

Histogram Stretching



Figure 12: Applying histogram stretching on the image after filtering with 3x3 binomial 1 time. (boat_histogram_stretching_3_1.png)

You can create the image using following code.

```
1 ./histogram boat.pgm>boat_histogram_stretching_3_1.png 3 1 s
```

List of Code 41: Make Figure boat_histogram_stretching_3_1.png



Figure 13: Applying histogram stretching on the image after filtering with 3x3 binomial 5 times. (boat_histogram_stretching_3_5.png)

You can create the image using following code.

```
1 ./histogram boat.pgm>boat_histogram_stretching_3_5.png 3 5 s
```

List of Code 42: Make Figure boat_histogram_stretching_3_5.png



Figure 14: Applying histogram stretching on the image after filtering with 5x5 binomial 1 time. (boat_histogram_stretching_5_1.png)

You can create the image using following code.

```
1 ./histogram boat.pgm>boat_histogram_stretching_5_1.png 5 1 s
```

List of Code 43: Make Figure boat_histogram_stretching_5_1.png



Figure 15: Applying histogram stretching on the image after filtering with 5x5 binomial 5 times. (boat_histogram_stretching_5_5.png)

You can create the image using following code.

```
1 ./histogram boat.pgm>boat_histogram_stretching_5_5.png 5 5 s
```

List of Code 44: Make Figure boat_histogram_stretching_5_5.png

First, we applied 3x3 and 5x5 binomial filters on the boat image. Then, applied histogram stretching on the filtered images. And, there is a slight difference between the input filtered images and the images after stretching.

Histogram Equalization



Figure 16: Applying histogram equalization on the image after filtering with 3x3 binomial 1 time. (boat_histogram_equalization_3_1.png)

You can create the image using following code.

```
1 ./histogram boat.pgm>boat_histogram_equalization_3_1.png 3 1 e
```

List of Code 45: Make Figure boat_histogram_equalization_3_1.png



Figure 17: Applying histogram equalization on the image after filtering with 3x3 binomial 5 times. (boat_histogram_equalization_3_5.png)

You can create the image using following code.

```
1 ./histogram boat.pgm>boat_histogram_equalization_3_5.png 3 5 e
```

List of Code 46: Make Figure boat_histogram_equalization_3_5.png



Figure 18: Applying histogram equalization on the image after filtering with 5x5 binomial 1 time. (boat_histogram_equalization_5_1.png)

You can create the image using following code.

```
1 ./histogram boat.pgm>boat_histogram_equalization_5_1.png 5 1 e
```

List of Code 47: Make Figure boat_histogram_equalization_5_1.png



Figure 19: Applying histogram equalization on the image after filtering with 5x5 binomial 5 times. (boat_histogram_equalization_5_5.png)

You can create the image using following code.

```
1 ./histogram boat.pgm>boat_histogram_equalization_5_5.png 5 5 e
```

List of Code 48: Make Figure boat_histogram_equalization_5_5.png

In addition to the histogram stretching, we have also applied histogram equalization on the images after binomial filters. We can see that after equalization, the images became darker, and details on the image (such as items on the boat) became hard to be distinguished.