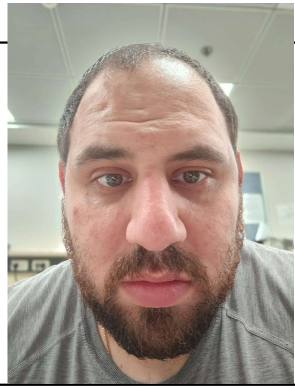
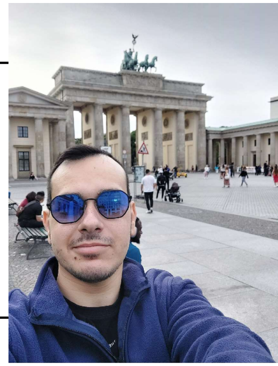
Course: Image Processing 31651

Micro Project 1 (MP1) : Convert Color Image to Grayscale Image (N=1)

Presentation date: 1/8/2024 on laboratory session (V=1)

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

Course: Image Processing 31651

Micro Project 1 (MP1) : Convert Color Image to Grayscale Image (N=1)

N=1 task:

N=1. Convert Color Image to Grayscale Image

1. Read and understand idea of 4 options to convert color image to grayscale image
   http://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/

2. By using BMP Library (plain C) implement above 4 algorithms

3. Prepare examples demonstrating that every algorithm can be better than other in some case.

4. Discuss results.

Course: Image Processing 31651
Micro Project 1 (MP1) : Convert Color Image to Grayscale Image (N=1)
Presentation contains:

#	Description
1	Description of the problem to be solved
2	Description of the algorithm including description of: Input Image(s), algorithm' parameter(s), output image(s)/value(s)
3	Detailed Block chart (Flow Chart/Block Diagram) of the algorithm(s)
4	Extracts of the important parts of the code with proper comments
5	Examples of the code operation by using images selected by students including a discussion when algorithm' worked and when not
6	List of sources used (Bibliography)

P1.1 Description of the problem to be solved – part 1

1

Grayscale images reduce the amount of data that needs to be processed, as they contain only intensity information compared to three channels (red, green, and blue) in color images.
This reduction in data simplifies algorithms and decreases computational load. [6]

Consequently - With fewer data points to process, operations on grayscale images are generally faster, which is crucial for real-time applications such as video processing.

2

Grayscale images require less storage space and memory compared to color images, making them more efficient for storage and processing. [6]

3

Many image processing techniques, such as edge detection, histogram equalization, and thresholding, are simpler to implement and perform more efficiently on grayscale images.

4

Combining photos taken in different weather conditions into one panoramic picture can make them look as if they were taken at the same time and under the same weather conditions.

P1.1 Description of the problem to be solved – part 2

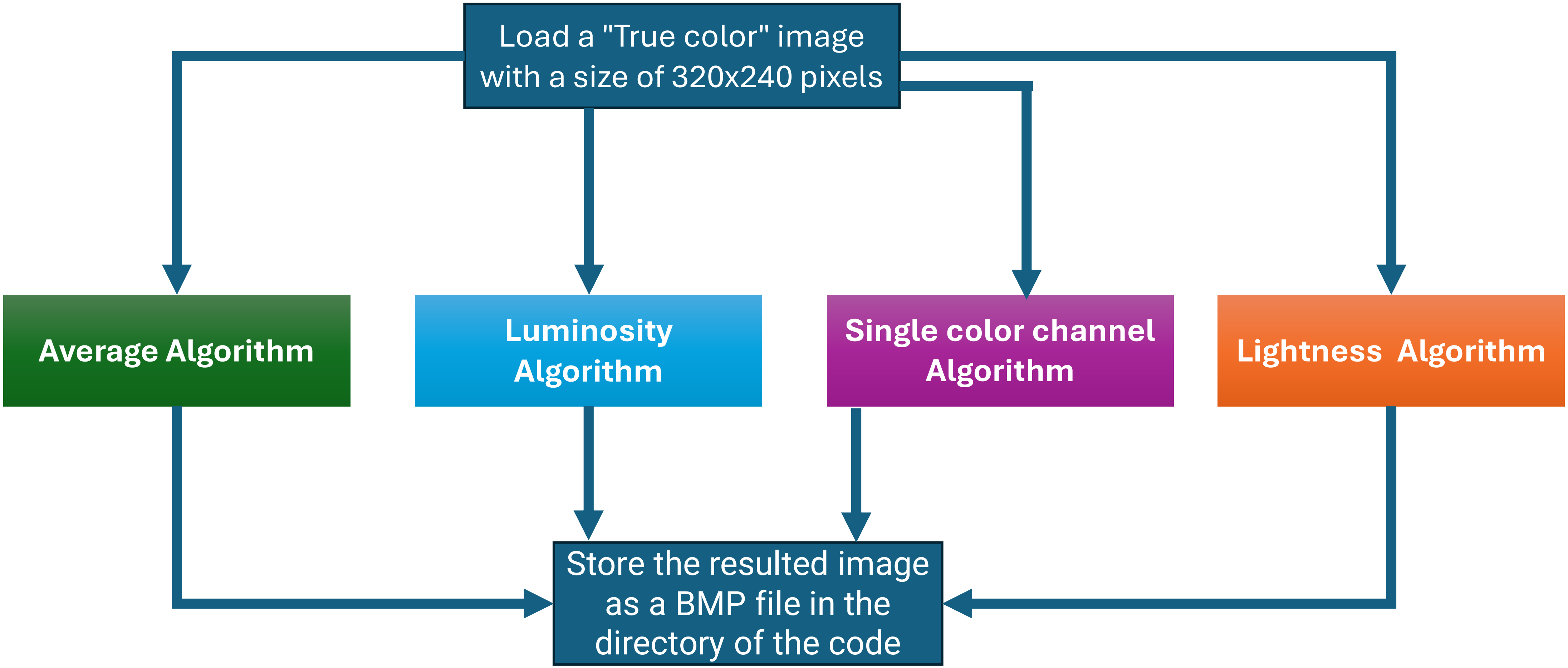
Some reasons from academic papers explaining the significance of using grayscale images instead of RGB images

"Many medical imaging techniques, such as X-rays and MRIs, produce grayscale images. Machine learning algorithms can be used to analyze these images to identify abnormalities and diseases. By focusing on the grayscale values of the pixels, machine learning algorithms can detect subtle changes that would be difficult to see with the naked eye." **(Gray Scaling with the Algorithms) [1]**

Color information is useless for distinguishing significant edges and features in numerous applications. In image processing, a gray image discards much-unrequired data in a color image. The primary drawback of colour-to-grey conversion is eliminating the visually significant image pixels. **(Color to Grayscale Image Conversion Based on Singular Value Decomposition) [4]**

"Conversion of color image to grayscale image is one of the image processing applications used in different fields effectively. In publication organizations' printing, a color image is expensive compared to a grayscale image. Thus, color images have converted to grayscale image to reduce the printing cost for low priced edition books. Similarly, color deficient viewer requires good quality of grayscale image to perceive the information, as the normal people perceive the color picture. Likewise, various image processing applications require conversion of color image to grayscale image for different purpose." **(Color Image to Grayscale Image Conversion) [5]**

P1.2 Description of the algorithms – part 1



P1.2 Description of the algorithms – part 2

Average Algorithm

Input Image(s) : The algorithm takes one "True color" image with a size of 320x240 pixels. (predefined in the attached header file)

Algorithm Parameters :
one parameter: 0.33 (division by 3)

Algorithm explanation:
simply averages the values: $\frac{(R+G+B)}{3}$.

Benefits of the algorithm:
This formula generates a reasonably nice grayscale equivalent, and its simplicity makes it easy to implement and optimize.
However, this formula is not without shortcomings - while fast and simple, **it does a poor job of representing shades of gray relative to the way humans perceive luminosity** (brightness).

Output Image(s):
This algorithm produces one grayscale image with a size of 320x240 pixels.

drawbacks of the algorithm:
Because humans do not perceive all colors equally, the “average method” of grayscale conversion is inaccurate.

P1.2 Description of the algorithms – part 3

Lightness Algorithm (or desaturation Algorithm)

Input Image(s) : The algorithm takes one "True color" image with a size of 320x240 pixels. (predefined in the attached header file)

Algorithm Parameters :
one parameter: ½ (division by 2)

a maximum decomposition provides a brighter grayscale image, while a minimum decomposition provides a darker one. This method of grayscale reduction is typically used for artistic effect. [7]

Benefits of the algorithm:
Lightness results in a flatter, softer grayscale image. If you compare this desaturated sample to the human-eye-corrected sample, you should notice a difference in the contrast of the image. So that method tends to reduce contrast.

Algorithm explanation:
A pixel can be desaturated by finding the midpoint between the maximum of (R, G, B) and the minimum of (R, G, B) From all three values (Red, Green, and Blue) of every pixel - averages the most prominent and least prominent colors (max & min):

$$Gray = \frac{\max(R,G,B) + \min(R,G,B)}{2}$$

Output Image(s):
This algorithm produces one grayscale image with a size of 320x240 pixels.

Another look of the algorithm:
“To calculate Grayscale, we can use the Lightness component of the HSL (Hue, Saturation, Lightness) color space.” [3]

P1.2 Description of the algorithms – part 4

Luminosity Algorithm

Algorithm Parameters :
First parameter (R) : 0.21, second parameter (G) : 0.72, third parameter (B) : 0.07

Input Image(s) : The algorithm takes one "True color" image with a size of 320x240 pixels. (predefined in the attached header file)

Benefits of the algorithm 1:
This algorithm plays off the fact that cone density in the human eye is not uniform across colors. Humans perceive green more strongly than red, and red more strongly than blue. This makes sense from an evolutionary biology standpoint - much of the natural world appears in shades of green, so humans have evolved greater sensitivity to green light. [7]

Benefits of the algorithm 2:
Instead of treating red, green, and blue light equally, a good grayscale conversion will weight each color based on how the human eye perceives it.

Output Image(s):
This algorithm produces one grayscale image with a size of 320x240 pixels.

Algorithm explanation:
A more sophisticated version of the average method. It also averages the values, but it forms a weighted average to account for human perception. We're more sensitive to green than other colors, so green is weighted most heavily.

P1.2 Description of the algorithms – part 5

Single color channel Algorithm

Input Image(s) : The algorithm takes one "True color" image with a size of 320x240 pixels. (predefined in the attached header file)

Algorithm Parameters : none

The source to the fourth algorithm is: **https://tannerhelland.com/2011/10/01/grayscale-image-algorithm-vb6.html**

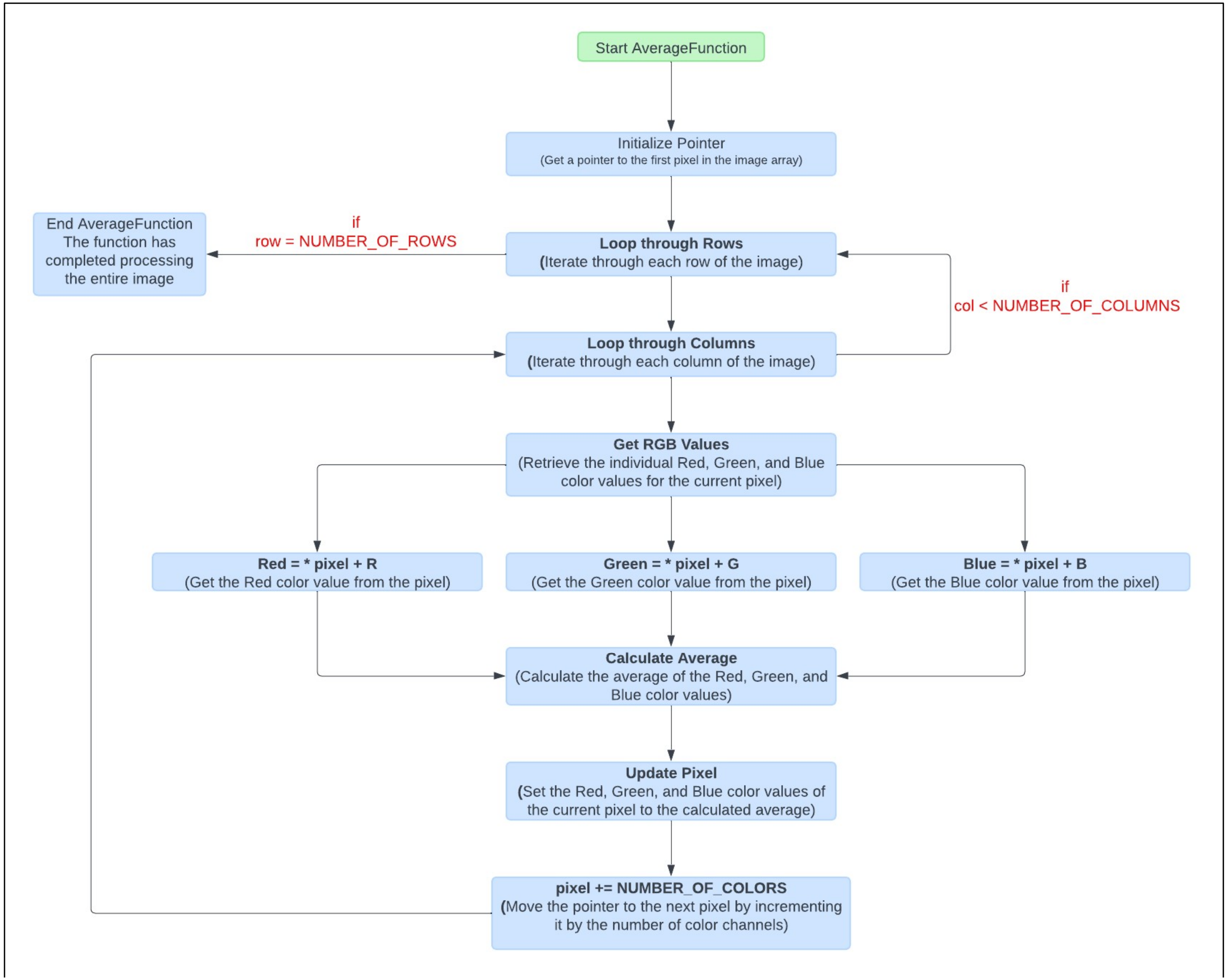
Benefits of the algorithm: The fastest computational method for grayscale reduction - using data from a single-color channel. Unlike all the methods mentioned so far, this method requires no calculations.

"Believe it or not, this weak algorithm is the one most digital cameras use for taking "grayscale" photos. CCDs in digital cameras are comprised of a grid of red, green, and blue sensors, and rather than perform the necessary math to convert RGB values to gray ones, they simply grab a single channel (green, for the reasons mentioned in Method #2 - human eye correction) and call that the grayscale one. For this reason, most photographers recommend against using your camera's built-in grayscale option. Instead, shoot everything in color and then perform the grayscale conversion later, using whatever method leads to the best result." [7]

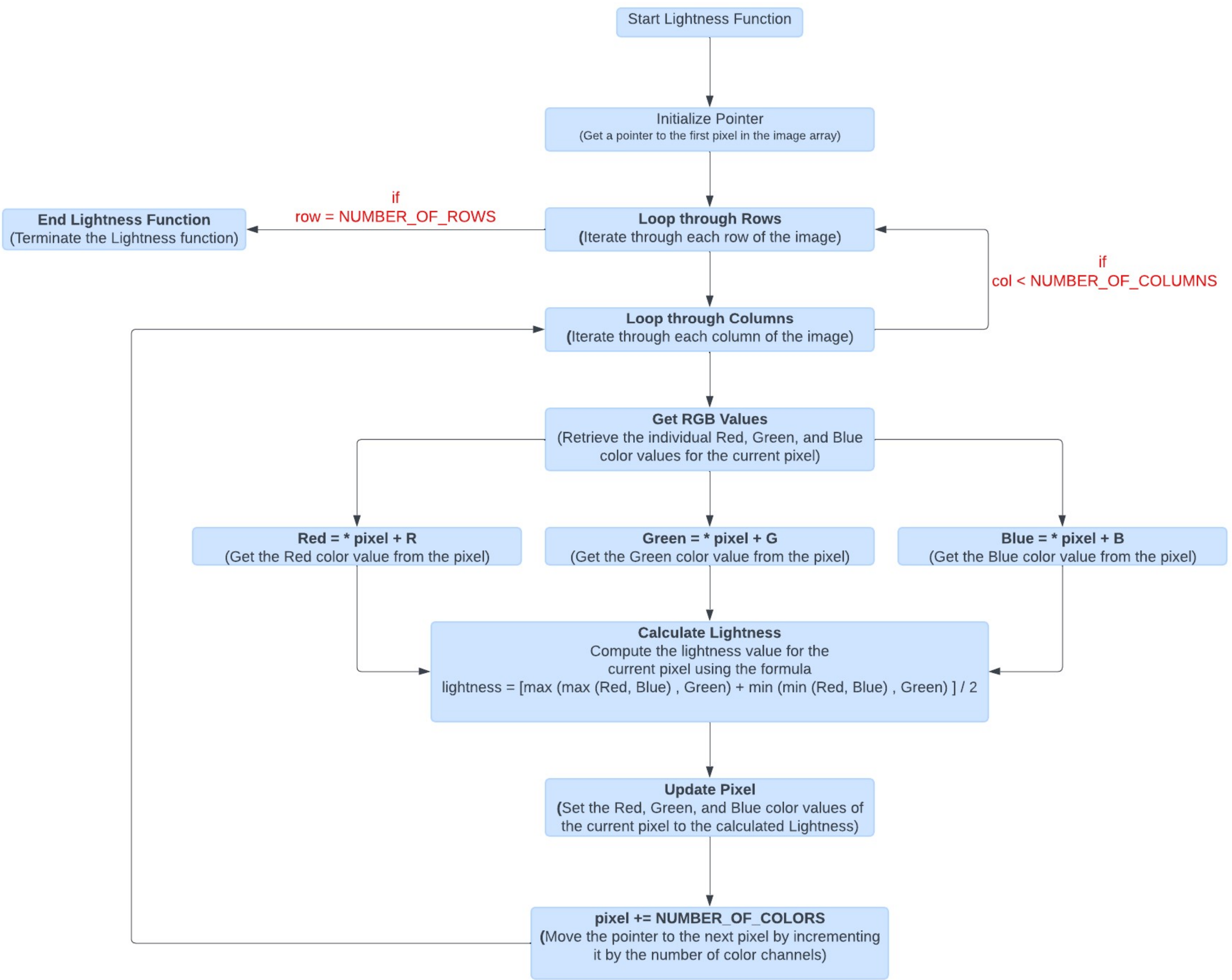
Output Image(s): This algorithm produces one grayscale image with a size of 320x240 pixels.

Algorithm explanation: All it does is pick a single channel and make that the grayscale value

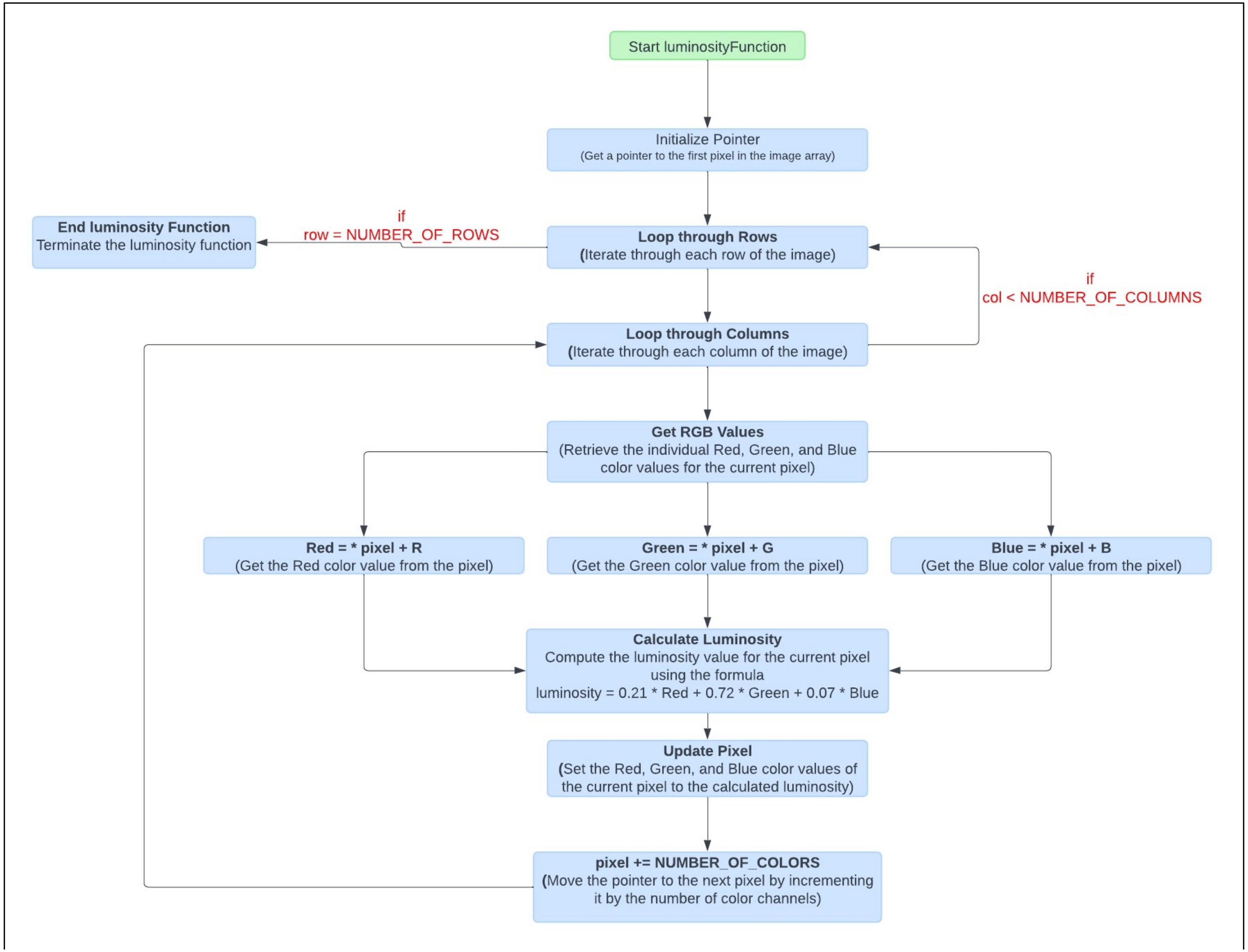
P1.2 Detailed block chart of the algorithm – part 1



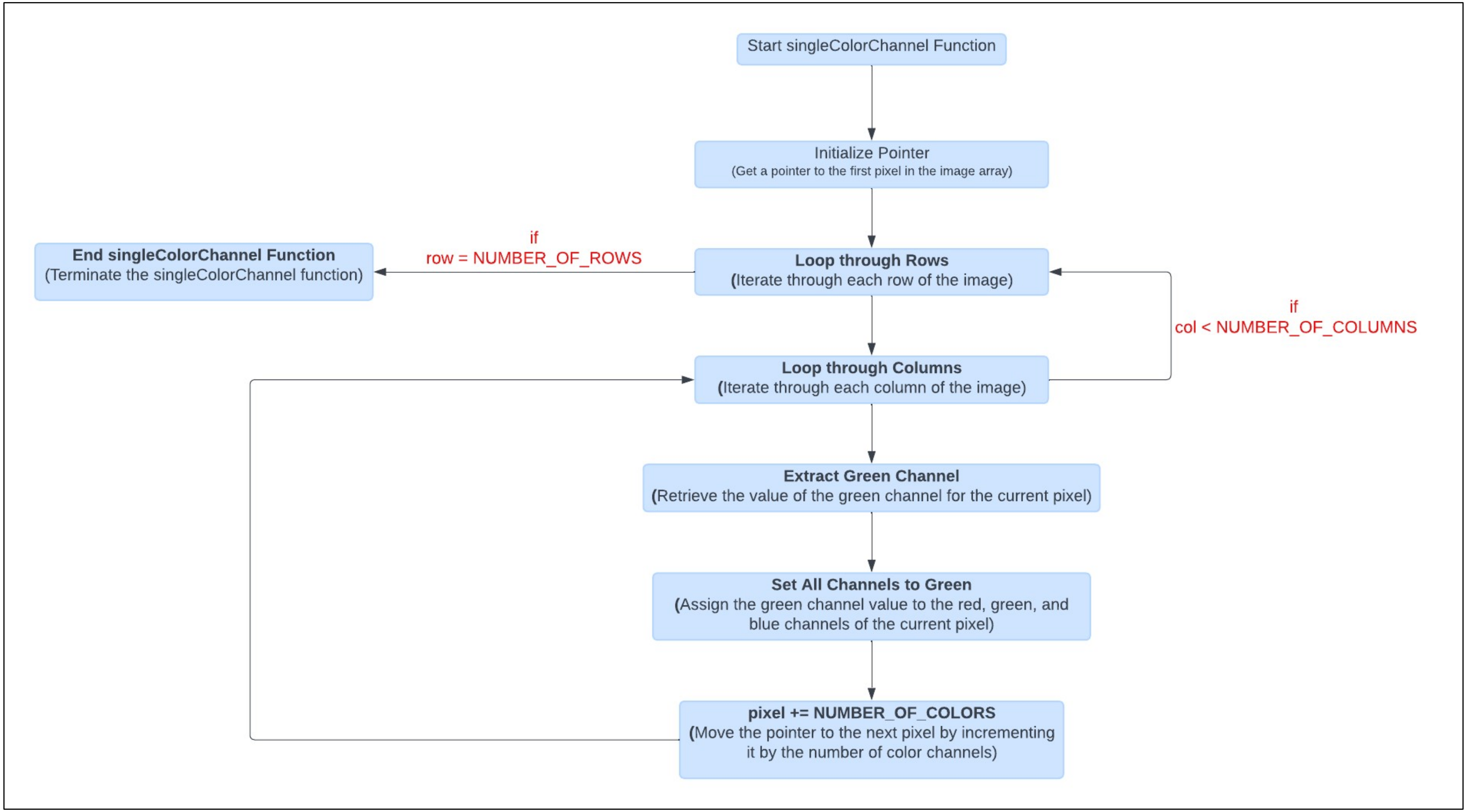
P1.2 Detailed block chart of the algorithm – part 2



P1.2 Detailed block chart of the algorithm – part 3



P1.2 Detailed block chart of the algorithm – part 4



P1.4 Extracts of the important parts of the code with proper comments – part 1

```c
void average(unsigned char image[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS][NUMBER_OF_COLORS])
{
    unsigned char* pixel = &image[0][0][0];
    for (int row = 0; row < NUMBER_OF_ROWS; row++)
    {
        for (int col = 0; col < NUMBER_OF_COLUMNS; col++)
        {
            int Red = *(pixel + R);
            int Green = *(pixel + G);
            int Blue = *(pixel + B);
            int avg = (Red + Green + Blue) / 3;
            *(pixel + R) = *(pixel + G) = *(pixel + B) = avg;
            pixel += NUMBER_OF_COLORS;
        }
    }
}
```

sets a pointer **pixel** to point to the beginning of the image data.

The **Red**, **Green**, and **Blue** are variables that store the color values of the current pixel – they are accessed using pointer arithmetic: Here, R, G, and B are defined as constants representing the offsets for red, green, and blue channels within a pixel.

The average of the three color values is calculated

The calculated average is assigned to all three color channels of the pixel

The pointer is incremented to point to the next pixel.

The average method converts a color image to grayscale by computing the average of the Red, Green, and Blue components for each pixel. This method is straightforward and considers all three-color channels equally.

P1.4 Extracts of the important parts of the code with proper comments – part 2

```c
void luminosity(unsigned char image[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS][NUMBER_OF_COLORS])
{
    unsigned char* pixel = &image[0][0][0];
    for (int row = 0; row < NUMBER_OF_ROWS; row++)
    {
        for (int col = 0; col < NUMBER_OF_COLUMNS; col++)
        {
            int Red = *(pixel + R);
            int Green = *(pixel + G);
            int Blue = *(pixel + B);
            int lum = int(0.21 * Red + 0.72 * Green + 0.07 * Blue);
            *(pixel + R) = *(pixel + G) = *(pixel + B) = lum;
            pixel += NUMBER_OF_COLORS;
        }
    }
}
```

sets a pointer **pixel** to point to the beginning of the image data.

The **Red**, **Green**, and **Blue** are variables that store the color values of the current pixel – they are accessed using pointer arithmetic: Here, R, G, and B are defined as constants representing the offsets for red, green, and blue channels within a pixel.

The luminosity is calculated using weighted averages of the RGB values.

The calculated average is assigned to all three color channels of the pixel

The pointer is incremented to point to the next pixel.

The luminosity method converts a color image to grayscale by taking a weighted sum of the Red, Green, and Blue components. This method accounts for human perception, giving more importance to the Green component as the human eye is more sensitive to green light.

P1.4 Extracts of the important parts of the code with proper comments – part 3

```
void lightness(unsigned char image[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS][NUMBER_OF_COLORS])
{
    unsigned char* pixel = &image[0][0][0];
    for (int row = 0; row < NUMBER_OF_ROWS; row++)
    {
        for (int col = 0; col < NUMBER_OF_COLUMNS; col++)
        {
            int Red = *(pixel + R);
            int Green = *(pixel + G);
            int Blue = *(pixel + B);
            int lightness = (max(max(Red, Blue), Green) + min(min(Red, Blue), Green)) / 2;
            *(pixel + R) = *(pixel + G) = *(pixel + B) = lightness;
            pixel += NUMBER_OF_COLORS;
        }
    }
}
```

sets a pointer **pixel** to point to the beginning of the image data.

The **Red**, **Green**, and **Blue** are variables that store the color values of the current pixel – they are accessed using pointer arithmetic:
Here, R, G, and B are defined as constants representing the offsets for red, green, and blue channels within a pixel.

The pointer is incremented to point to the next pixel.

Instead of calculating the average, the **lightness** is calculated using the maximum and minimum color values.

The lightness method converts a color image to grayscale by taking the average of the maximum and minimum RGB values of each pixel. This method considers the extremes of the color range, making it sensitive to the lightest and darkest parts of the image.

P1.4 Extracts of the important parts of the code with proper comments – part 4

```
void singleColorChannel(unsigned char image[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS][NUMBER_OF_COLORS])
{
    unsigned char* pixel = &image[0][0][0];
    for (int row = 0; row < NUMBER_OF_ROWS; row++)
    {
        for (int col = 0; col < NUMBER_OF_COLUMNS; col++)
        {
            int green = *(pixel + G);
            *(pixel + R) = *(pixel + G) = *(pixel + B) = green;
            pixel += NUMBER_OF_COLORS;
        }
    }
}
```

sets a pointer **pixel** to point to the beginning of the image data.

The **green** variable store the color values of the current pixel – it is accessed using pointer arithmetic

The green value of the current pixel is directly assigned to all color channels.

The pointer is incremented to point to the next pixel.

The Single-Color Channel method simplifies the conversion of a color image to grayscale by using the intensity value from one of the color channels (red, green, or blue) for all three channels of the grayscale image. This method is straightforward and computationally efficient.

The green channel is often chosen because the human eye is more sensitive to green light, providing a balanced representation of the image's brightness.

This method retains the detail and brightness of the original image by focusing on the green channel only.

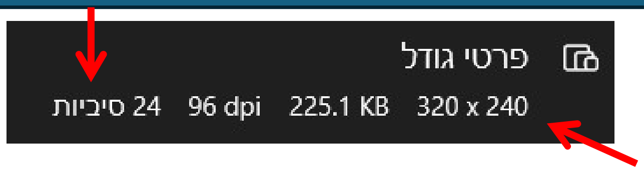
P1.4 Extracts of the important parts of the code with proper comments – part 5

```c
void main()
{
    static unsigned char Picture[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS][NUMBER_OF_COLORS];

    //First picture
    LoadBgrImageFromTrueColorBmpFile(Picture, "1.bmp");
    average(Picture);
    StoreBgrImageAsGrayBmpFile(Picture, "1_averge.bmp");

    LoadBgrImageFromTrueColorBmpFile(Picture, "1.bmp");
    lightness(Picture);
    StoreBgrImageAsGrayBmpFile(Picture, "1_lightness.bmp");

    LoadBgrImageFromTrueColorBmpFile(Picture, "1.bmp");
    luminosity(Picture);
    StoreBgrImageAsGrayBmpFile(Picture, "1_luminosity.bmp");

    LoadBgrImageFromTrueColorBmpFile(Picture, "1.bmp");
    singleColorChannel(Picture);
    StoreBgrImageAsGrayBmpFile(Picture, "1_singleColorChannel.bmp");

    //Second picture
    LoadBgrImageFromTrueColorBmpFile(Picture, "2.bmp");
    average(Picture);
    StoreBgrImageAsGrayBmpFile(Picture, "2_averge.bmp");

    LoadBgrImageFromTrueColorBmpFile(Picture, "2.bmp");
    lightness(Picture);
    StoreBgrImageAsGrayBmpFile(Picture, "2_lightness.bmp");

    LoadBgrImageFromTrueColorBmpFile(Picture, "2.bmp");
    luminosity(Picture);
    StoreBgrImageAsGrayBmpFile(Picture, "2_luminosity.bmp");

    LoadBgrImageFromTrueColorBmpFile(Picture, "2.bmp");
    singleColorChannel(Picture);
    StoreBgrImageAsGrayBmpFile(Picture, "2_singleColorChannel.bmp");

}
```

The main contains the static allocation of the picture [ ] [ ] [ ] and the calls to the algorithm's functions.

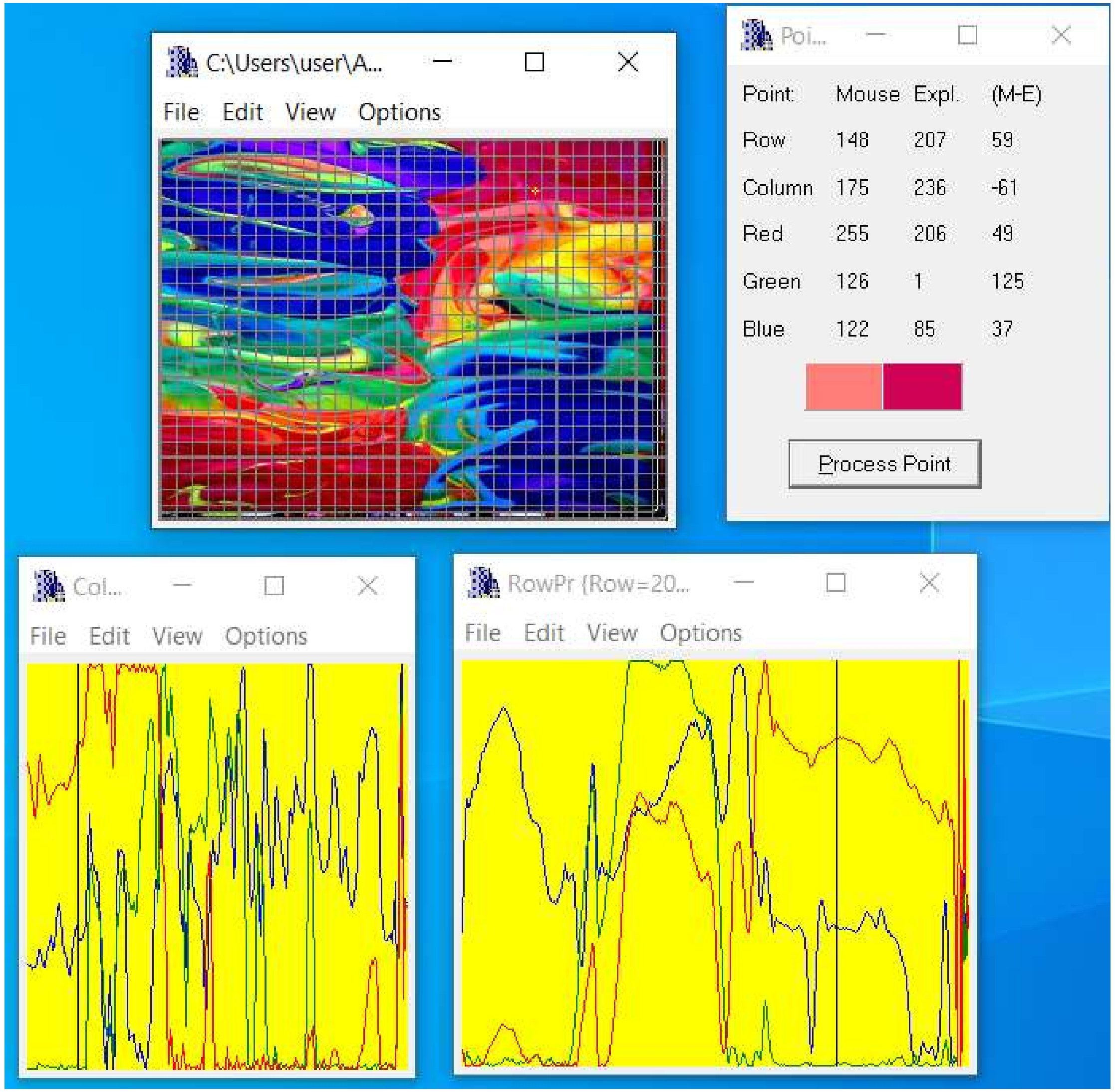
Of Couse we use:
LoadBgrImageFromTrueColorBmpFile
And StoreBgrImageAsGrayBmpFile
for excessing the relevant color image located in the directory and finally saving the created gray image

Another notation is that when loading the color image we made sure (otherwise the loading fail) that the image stands on the requirement (size, format , bpp)



P1.5 Examples of the code operation by using images selected by students – part 1

The original photo

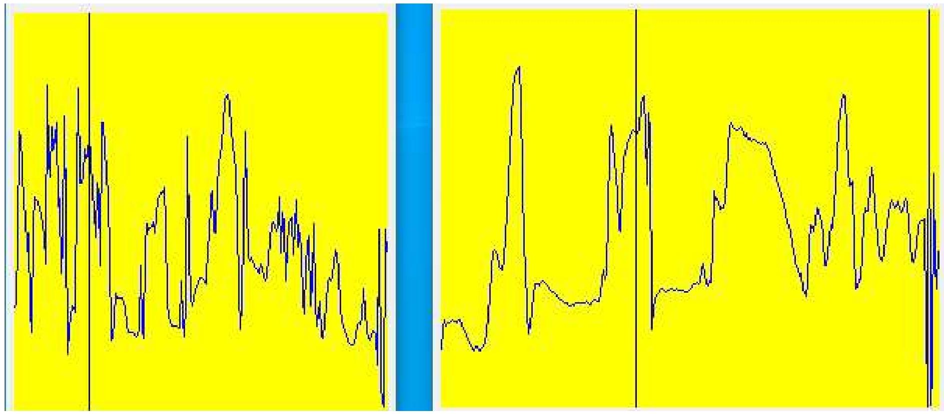


P1.5 Examples of the code operation by using images selected by students – part 2

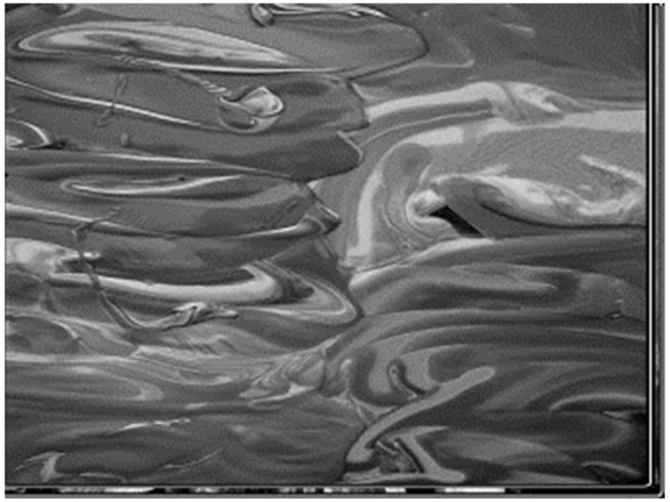
We can see that the lightness and the average algorithms provides us almost the same result. The gray level for the specific point is 63 in both case and the profiles (row & columns) are very much alike.
The main difference is that the average turns strong colors into darker level of grayscale than the lightness ⇒ average > lightness (for this example only)

Using “average algorithm”

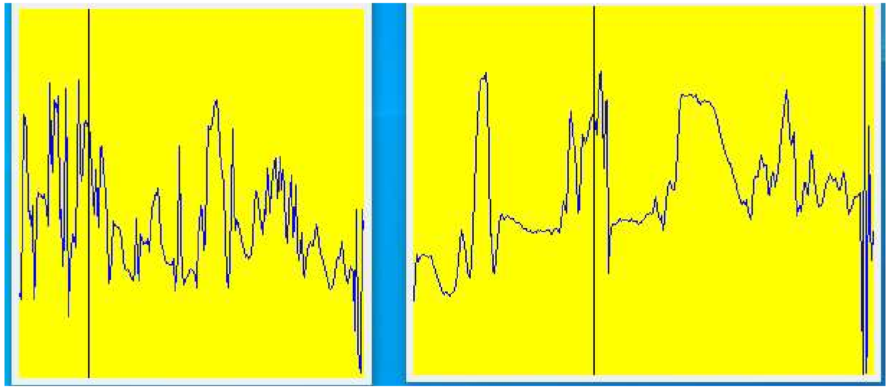
Point:	Mouse	Expl.	(M-E)
Row	205	191	-14
Column	293	125	168
Red	63	177	-114
Green	63	177	-114
Blue	63	177	-114



Using “lightness algorithm”



Point:	Mouse	Expl.	(M-E)
Row	205	191	-14
Column	293	125	168
Red	63	177	-114
Green	63	177	-114
Blue	63	177	-114

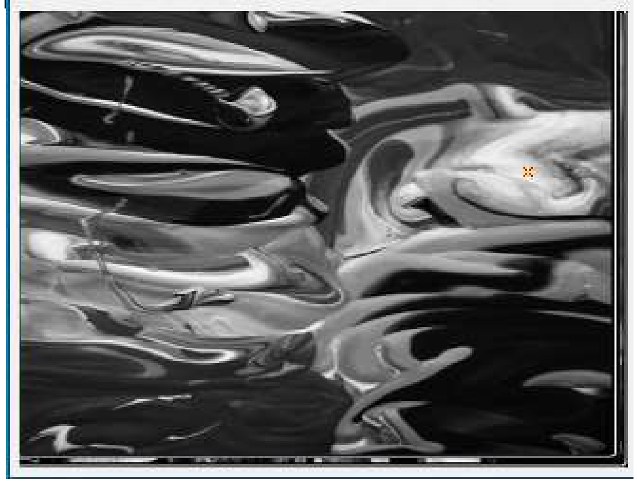


P1.5 Examples of the code operation by using images selected by students – part 3

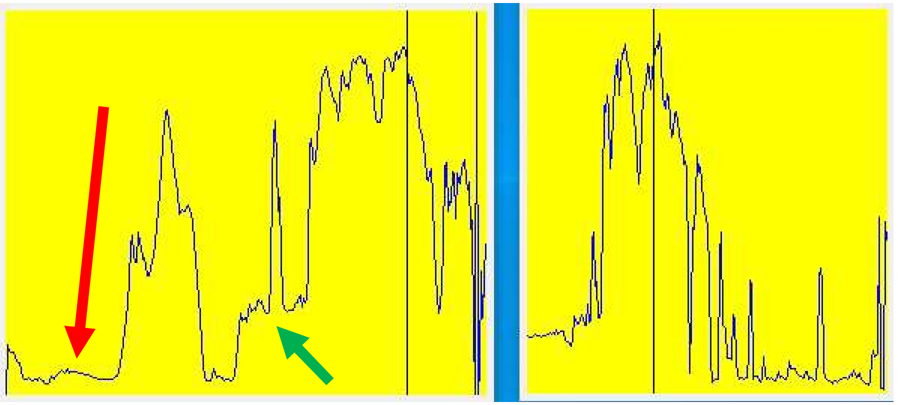
We can see that the Single-color channel and the luminosity algorithms provides us similar results. By looking at the profiles we can say that the main difference between the two is a certain level of offset in the luminosity profiles in comparison to the Single-color channel. The main visual difference between the two is that the green areas are stronger (whiter) in the Single-color channel.

⇒ luminosity > Single color channel (for this example only)

Using "luminosity algorithm"

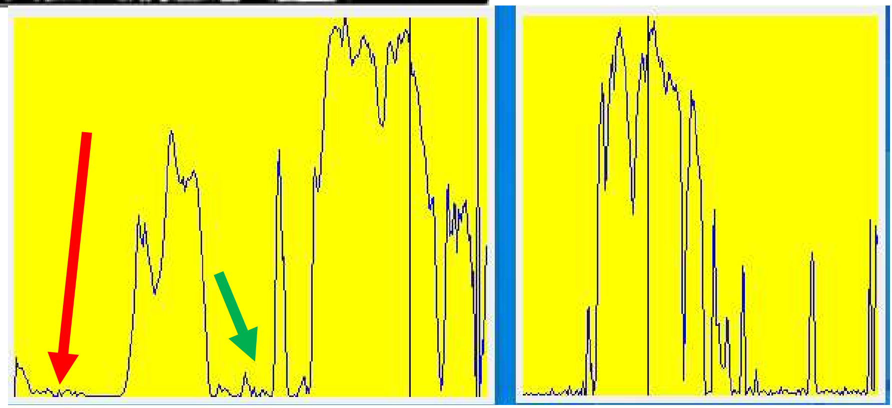


Point:	Mouse	Expl.	(M-E)
Row	155	155	0
Column	267	267	0
Red	221	221	0
Green	221	221	0
Blue	221	221	0

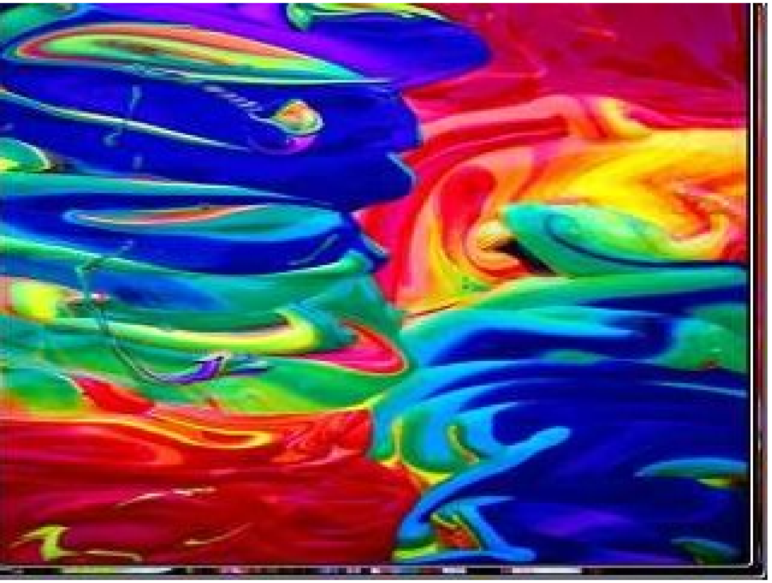


Using "Single color channel Algorithm"

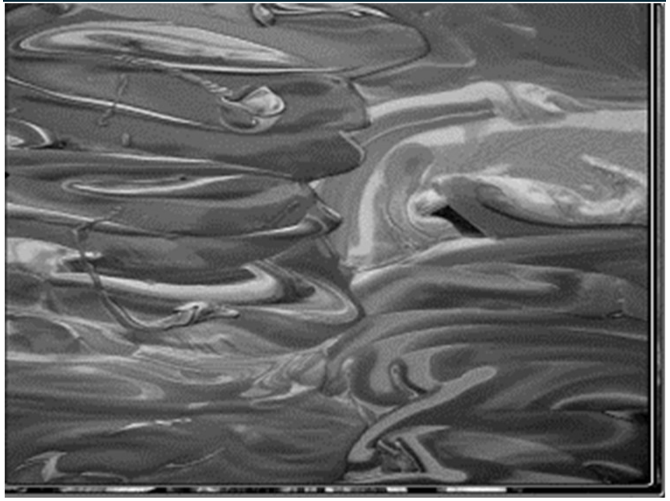
Point:	Mouse	Expl.	(M-E)
Row	155	155	0
Column	267	267	0
Red	234	234	0
Green	234	234	0
Blue	234	234	0



P1.5 Examples of the code operation by using images selected by students – part 4



(2) Using "lightness algorithm"



(1) Using "average algorithm"

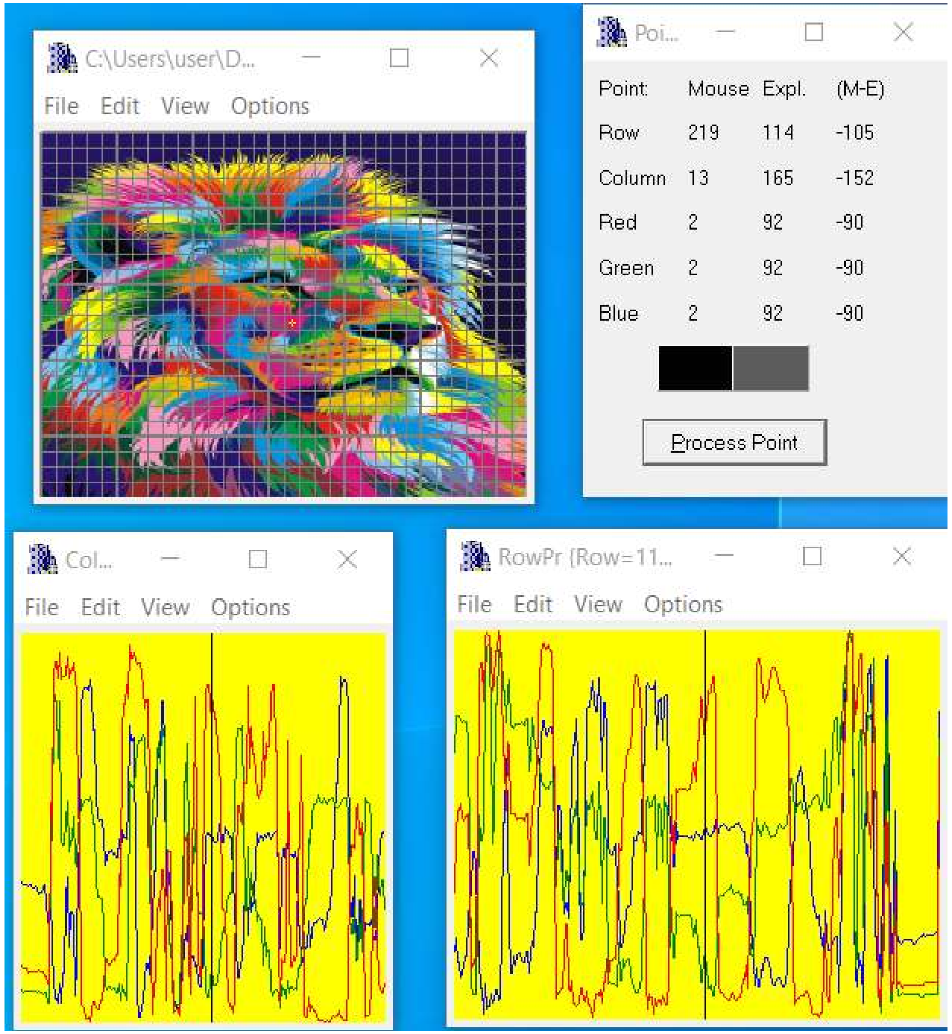
In this comparison the Average algorithm maintained the transition between the colors.
The Luminosity algorithm "exaggerate" the strong colors and turned them black. The lightness algorithm kept all the colors in a limited range and did not highlight any color changes.
conclusion : Average wins.
Total score will be:
Average>lightness > luminosity > SCH

(3) Using "single color channel Algorithm"

(4) Using "luminosity algorithm"

P1.5 Examples of the code operation by using images selected by students – part 5

The original photo

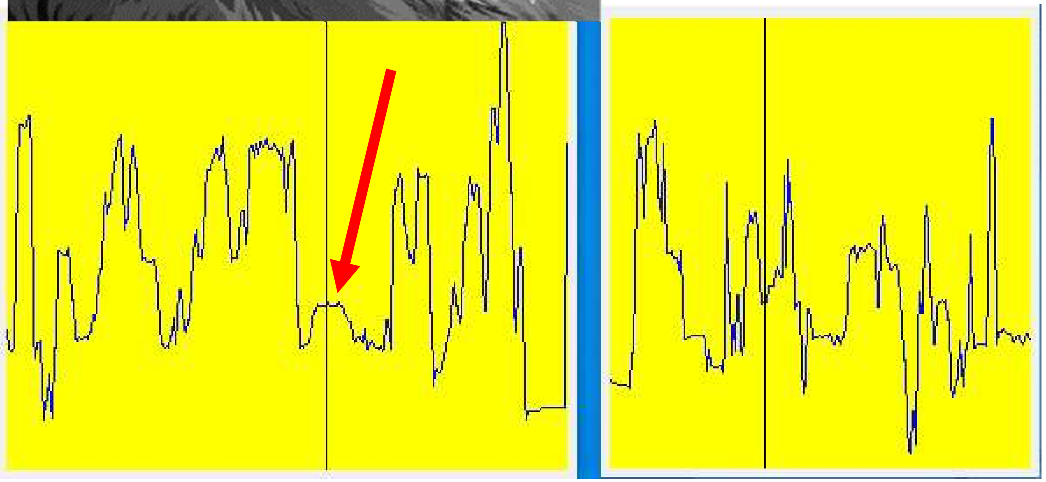


P1.5 Examples of the code operation by using images selected by students – part 6

We can see that the lightness and the average algorithms provides us almost the same result. The main visual difference between the two is the red areas. The average algorithm gives better results for these areas, and it can be seen for the specific point we picked. The profiles are almost identical but the lightness has larger peaks in the red areas (122 vs 94) ⇒ average > lightness (for this example only)

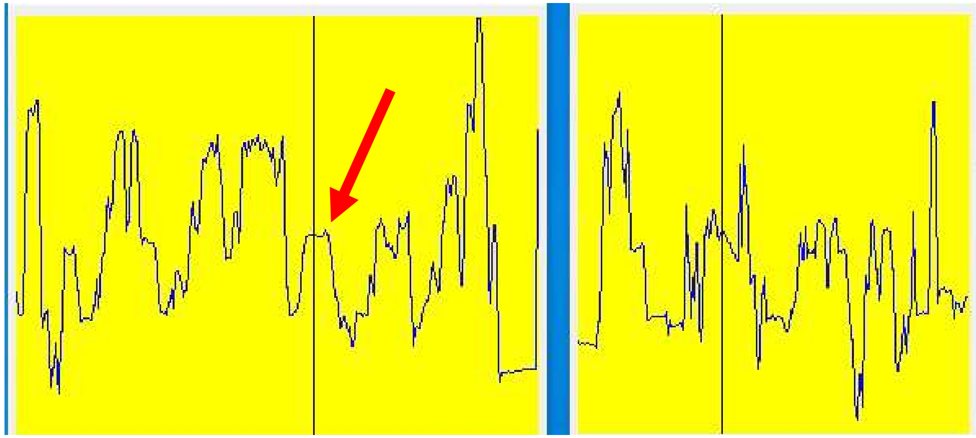
Using “average algorithm”

Point:	Mouse	Expl.	(M-E)
Row	151	151	0
Column	182	182	0
Red	94	94	0
Green	94	94	0
Blue	94	94	0



Using “lightness algorithm”

Point:	Mouse	Expl.	(M-E)
Row	151	151	0
Column	182	182	0
Red	122	122	0
Green	122	122	0
Blue	122	122	0

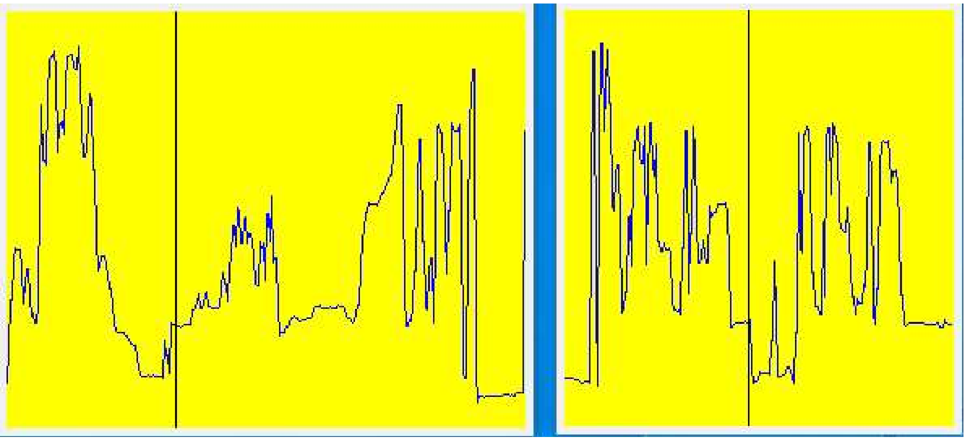


P1.5 Examples of the code operation by using images selected by students – part 7

We can see that the Single-color channel and the luminosity algorithms provides us different visual results. The main visual difference is the red areas that become much darker in the Single-color channel in a way that damages its quality. The luminosity's results are more realistic. The profiles of the Single-color channel are more tense and deliver more details of the color image but it hearts its realism. ⇒ luminosity > Single color channel (for this example only)

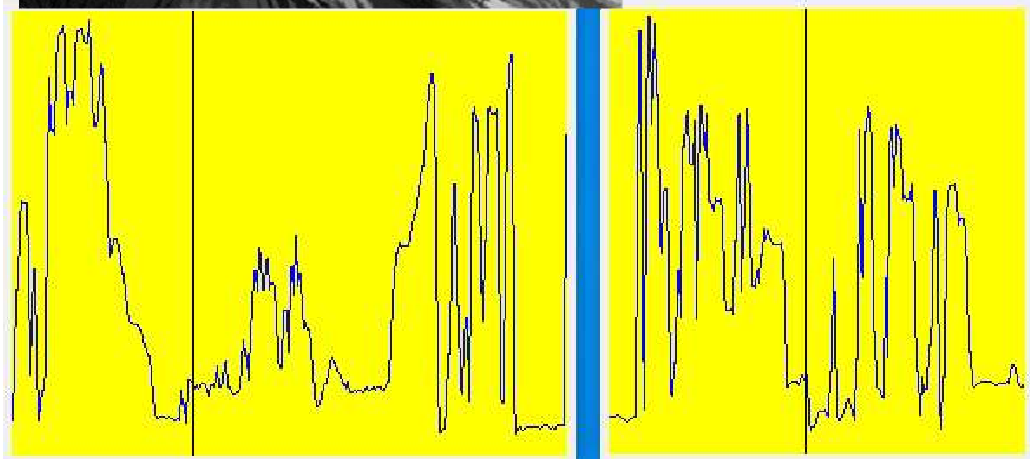
Using "luminosity algorithm"

Point:	Mouse	Expl.	(M-E)
Row	126	126	0
Column	104	104	0
Red	63	63	0
Green	63	63	0
Blue	63	63	0



Using "Single color channel Algorithm"

Point:	Mouse	Expl.	(M-E)
Row	126	126	0
Column	104	104	0
Red	42	42	0
Green	42	42	0
Blue	42	42	0



P1.5 Examples of the code operation by using images selected by students – part 8

It can be seen that the algorithms of Average and Lightness produce quite similar results. The average is better between them. (more details in red areas). However Luminosity blackens the background and gets more dark spots on the inside of the lion, making more realistic facial features. The single-color channel blacken the red and pink areas even more makes it unrealistic.

conclusion : luminosity wins.

Total score will be:

luminosity>average > lightness > SCH

(2) Using "lightness algorithm"

(1) Using "average algorithm"

(3) Using "single color channel Algorithm"

(4) Using "luminosity algorithm"

P1.5 Examples of the code operation by using images selected by students – part 9

Conclusions:

- The lightness algorithm works by reduce the contrast, so it will work better in images where the contrast between the colors is stronger.

- The average algorithm is the most simple one. You just have to take the average of three colors. But sometimes the images can be more black image then grayscale image(or vice versa).
  Its arise due to the fact, that we take average of the three colors. Since the three different colors have three different wavelength and have their own contribution in the formation of image. If we want to improve this conversion so we have to take average according to their contribution, not done it averagely using average method.

- Sometimes couple of methods produce very similar results.

- The single color channel Algorithm turns the green areas into white areas and strong colors (like red and pink) into almost black color.

- The two most realistic results came from the **luminosity** and the **average** algorithms

P1.6 List of sources used (Bibliography) – part 1

https://medium.com/@mjbharmal2002/gray-scaling-with-the-algorithms-b83f87975885 ↔ [1] Gray Scaling with the Algorithms / Mustafa Bharmal - Nov 3, 2023

https://do-marlay-ka-moonh.medium.com/converting-color-images-to-grayscale-ab0120ea2c1e ↔ [2] Converting Color Images to Grayscale - Sep 25, 2017 – In medium

https://fiveko.com/convert-rgb-to-grayscale-algorithms/ ↔ [3] = Algorithms for RGB to Grayscale conversion

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10132453 ↔ [4] Color to Grayscale Image Conversion Based on Singular Value Decomposition (SVD)

https://ieeexplore.ieee.org/document/5445596 ↔ [5] Color Image to Grayscale Image Conversion

https://www.jeremymorgan.com/tutorials/opencv/how-to-grayscale-image/?source=post_page-----b83f-87975885 ↔ [6] Grayscaling Images Made Simple with OpenCV

https://tannerhelland.com/-image-grayscale/2011/10/01 html.6vb-algorithm ↔ [7] Seven grayscale conversion algorithms (with pseudocode and VB6 source code) -Tanner Helland -Oct 1, 2011

P1.6 List of sources used (Bibliography) – part 2 (additional we used without citations)

https://mmuratarat.github.io/rgb_to_grayscale_formulas/2020-05-13 ⟷ RGB to Grayscale Conversion-Mustafa Murat ARAT-May 13, 2020

https://tannerhelland.com/-algorithm-image-grayscale/2011/10/01 html.6vb ⟷ Seven grayscale conversion algorithms (with pseudocode and VB6 source code) -Tanner Helland -Oct 1, 2011

https://e2eml.school/convert_rgb_to_grayscale ⟷ How to Convert an RGB Image to Grayscale

https://www.baeldung.com/cs/convert-rgb-to-grayscale ⟷ How to Convert an RGB Image to a Grayscale / by Panagiotis Antoniadis & Michal Aibin on March 18, 2024