1. Project’s title and authors

Title of the project is Grayscale Conversion and it was realized by a team of 3 members:

1. Konrad Walas
2. Marcin Filipek
3. Adrian Furman

2. Project description

Reasons to use grayscale conversion are various, for example artistics or in order to extract some details from photos in special medicine visualisations. In most cases basic functions built in numerous simple graphics editors are not enough to obtain satisfying results. Main idea of the project is to allow user to decide how much informations from red, green, blue channels should be included in brightness of the final photo. Moreover program allows user to decide if he/she want to keep any color and it’s hue.

3. Initial assumptions and requirements

Basic:

1. Program displays user interface with menu and area for live preview of the transformed image.
2. Program allows user to load images which can be then manipulated.
3. Program allows to save converted image
4. Using sliders on sidebar menu user can choose proportions between colours channel mixture, from -200% to 200% each.
5. Besides grayscale conversion user can apply bichrome.
6. Program generates live preview from image thumbnail. Longer side of thumbnail is at most 500px long.

Extended:

7. User is able to save/load all configurations to/from a file.
8. Program allows to mix between proportions of original image and converted one.
9. It is possible to keep specific hue on the image during all conversions. Strength of this effect is controlled by tolerance parameter.

4. Project analysis

4.1 Input specification

User can load images from filesystem, manipulate various parameters (channels, hue, tolerance etc.) by dragging sliders, click buttons to apply specific conversion and choose color from window dialog. Program supports all popular image formats provided by wxWidgets.

Additionally it is possible to load all parameters from configuration file.

4.2 Output specification

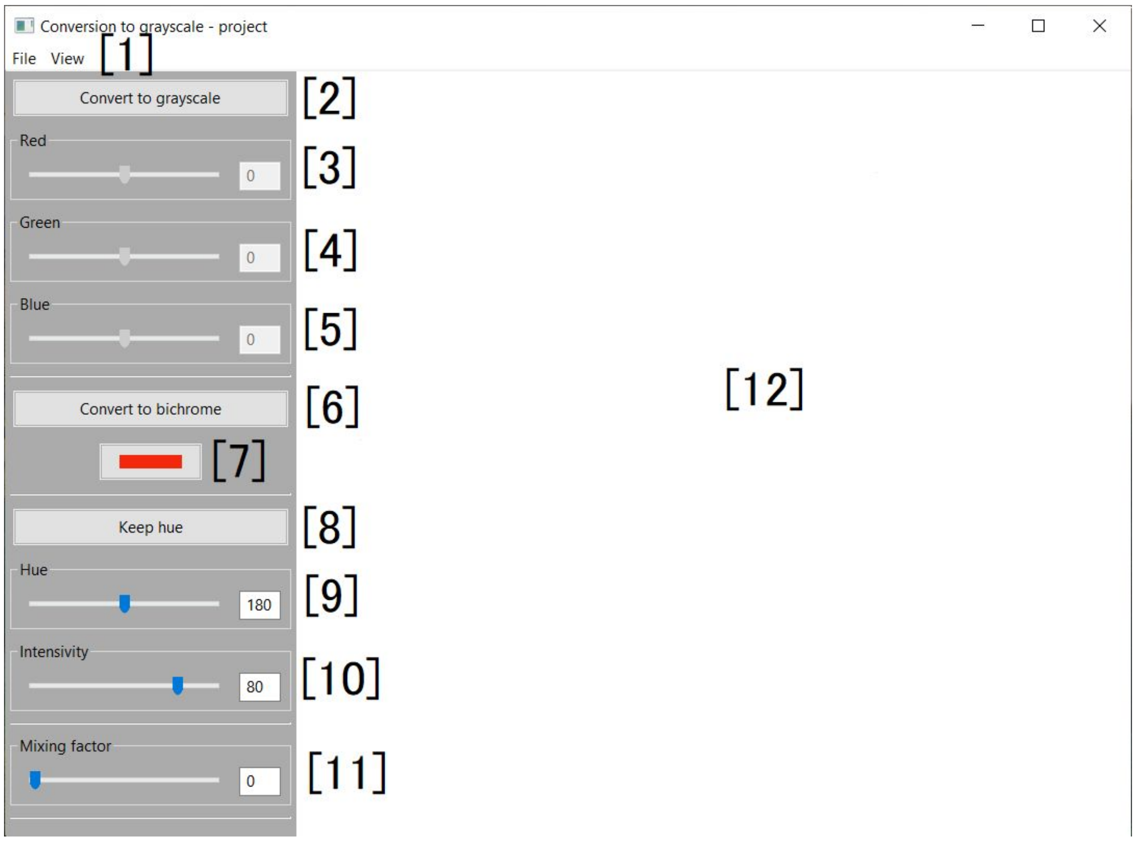
On panel there is live preview of the modification on the image.

User is able to save results of the work in all popular image formats.
What's more, current set of parameters can be saved to a config file.

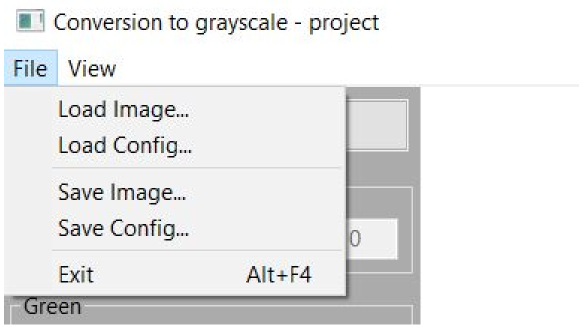
4.3 Data structures

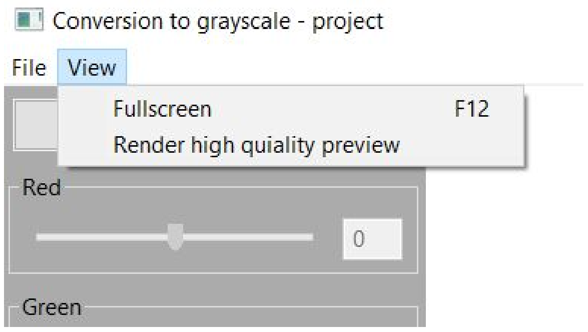
Since aim of the program is image manipulation and conversions, main data structure that we operate on is image data stored in unsigned char array that is returned by GetData() method from wxImage class.

4.4 UI specification



[1] - Menu bar containing two lists that allows user to perform actions connected with both - image and config file loading/saving as well as some view settings - fullscreen mode and high quality preview:




Conversion to grayscale - project
File
View
Fullscreen
F12
Render high quiality preview
Red
0
Green

[2] - Toggle button performing grayscale conversion.
[3], [4], [5] - Sliders for setting the contribution of corresponding value of color channel.
[6] - Button performing conversion to bichrome on image.
[7] - Button opening dialog to choose color for bichrome.
[8] - Toggle button for keeping chosen hue as original during conversions.
[9] - Slider for setting value of kept hue.
[10] - Slider for determining tolerance of kept hue. The higher value it has, the more neighbouring values of hue will be kept.
[11] - Slider for determining the proportion between original image and its transformed version.
[12] - Panel on which preview will be displayed.

All sliders have a text field to their right that displays current value and also can be used to set value manually.

4.5 Modules extraction and definition

Since we decided to apply MVC design pattern to our program, natural division on Model, Controller and View appeared.

In our case, Controller class (**ControllerFrame**) inherits from Frame class wich is implementation of wxFrame class. Frame class was generated using wxFormBuilder program. Frame class contains all GUI elements and controls but implementation of methods for event handling are located in Controller class. This enables controller to react on specific actions and manipulate the View.

Model class contains all conversion parameters given by the user and holds data of loaded image with it’s copies that are necessary for different actions. Model provides setters and getters for all parameters together with few methods that perform specific actions on image data.

View **(ImageView)** class represents area where conversion results are displayed. It contains method for showing live preview of image located in model.

ImageConversion is static class for performing actual image conversion. It contains methods for grayscale conversion, bichrome conversion along with helper methods and keeping chosen hue handling.

4.6 Programing and tools

All team members used Visual Studio 2019 code editor. Library wxWidgets 3.0.4 was used for UI/GUI implementation. To simplify process of GUI creation we decided to make use of wxFormBuilder. Code documentation was generated by Doxygen. For task tracking Trello task management app was used - it allowed us to have a look at current stage of each task at any moment in time. Synchronization and cohesion of code was was acquired by GIT tool.

5. Work split and time analysis

Since there was 3 project members and MVC pattern has 3 segments it was natural to split work in this way.

Konrad Walas - Team leader, Controller class implementation, putting together all segments
Marcin Filipek - GUI design, Image conversion class implementation
Adrian Furman - Model class and documentation

6. Algorithms

6.1 Greyscale conversion

Each pixel is separated to 3 channels: R, G and B as numbers in [0, 1]. Then each channel is multiplied by factor within range [-2, 2]. The factor is taken from sliders and scaled to that range. We take the mean of those channels - our grey colour. It is necessary do perform thresholding: if the value is above 1, we set it to 1. The same if the value is negative, we set it to 0. Our grey value is assigned to each R,G,B channel, after scaling to 8-bit ineger value.

6.2 Bichrome conversion

First the given colour is converted from RGB to HSL values. Let’s call them Hx, Sx and Lx. Hx is in range [0, 359] but this is not important, as it will be shown later. Sx and Lx are in range [0, 1]. We scale Lx to [-1, 1].
After that we convert every pixel using the same conversion. Now we can modify the received HSL values. The essential part of bichrome is:

$H = Hx$.

We simply replace the hue of the pixel with the Hx. But the image is not so visually appealing. To obtain preetier colours we also manipulate S and L:

$S = Sx + 1.1S$
$L = Lx + 1.1L$

The factors were found empirically. If S and L are outside range [0, 1] we apply thresholding (like in greyscale conversion). In order to print the changed pixel, we convert it back from HSL to RGB.

6.3 Mixing original and conveted images

This effect is simply achieved by making a weighted mean of each pixel. The weight are taken from slider. The value of slider is the percant weight of original image and 100 - value is the percent weight of coverted image, so they sum up to 100%.

6.4 Saving one hue

This is probaly he most complicated part of the program. If the button is down the algorithm is performed inside previous convertions. he method is genarally on the weighted mean, like mixing images. For every pixel the weight is calculated from its distance to designated hue in HSL using the following formula:

$$\begin{cases} \frac{|H-H_x|}{t}, & [H-t,\ H+t] \subseteq [0,\ 359] \\ \left|\frac{-|H-H_x-180|+180}{t}\right|, & H_x - t < 0 \\ \left|\frac{-|H-H_x+180|+180}{t}\right|, & H_x + t > 359 \end{cases}$$

where Hx is the hue set on slider, H is current hue of the pixel and t is tolerance scaled to [0, 360]. If the function value is bigger than 1, it is set to 1. Also if the weight is negative,it is set to 0.
There is also a special case when tolerance i 0. We cannot divite by 0 so in that case the function simply checks if H = Hx and sets binary wages: 0 and 1. After that colour each pixel is calculated using weighted mean:

C = C*w + C`*(1-w),

where C is the colour of converted pixel and C` is the colour of original pixel. This algorithm is in fact the implementation of the transparency mask.

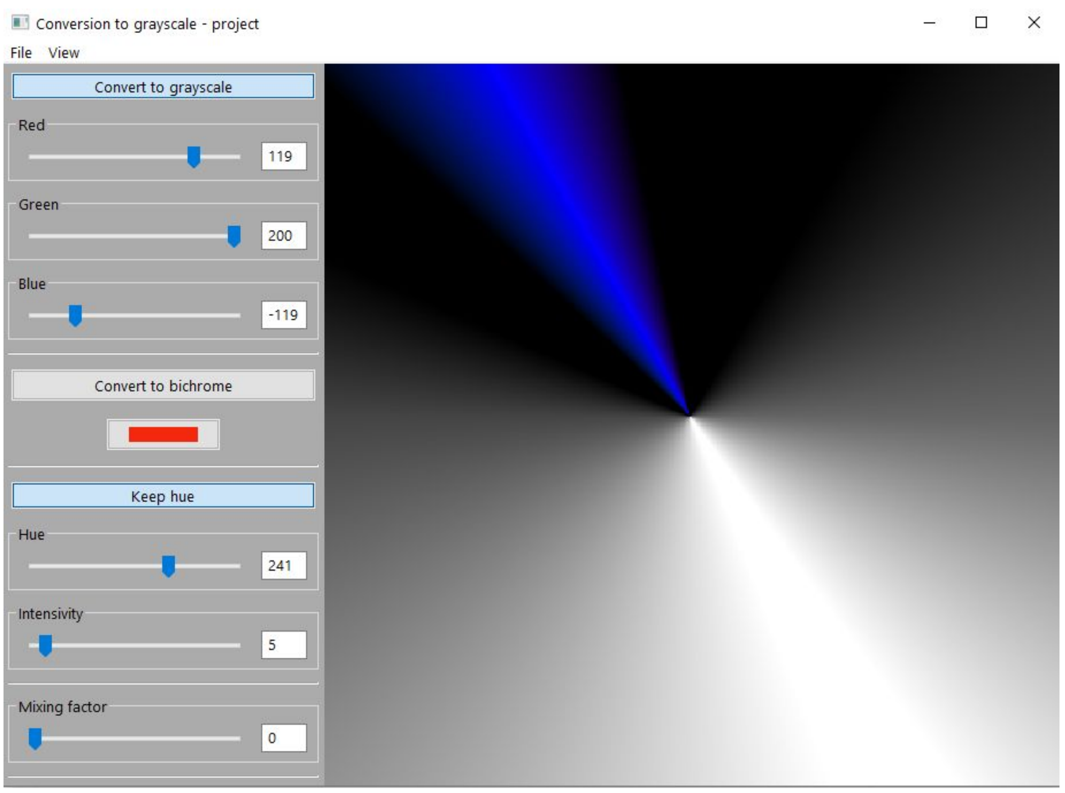
7. Coding

Whole code and structure is documented in Doxygen.
This documentation is also hosted on
https://kerdamon-university-tasks.github.io/PGK_Project_Greyscale/

8. Testing

Every module functionality was tested shortly after it was finished to ensure ourselves that everything worked fine so we could move forward without worrying about already implemented features.

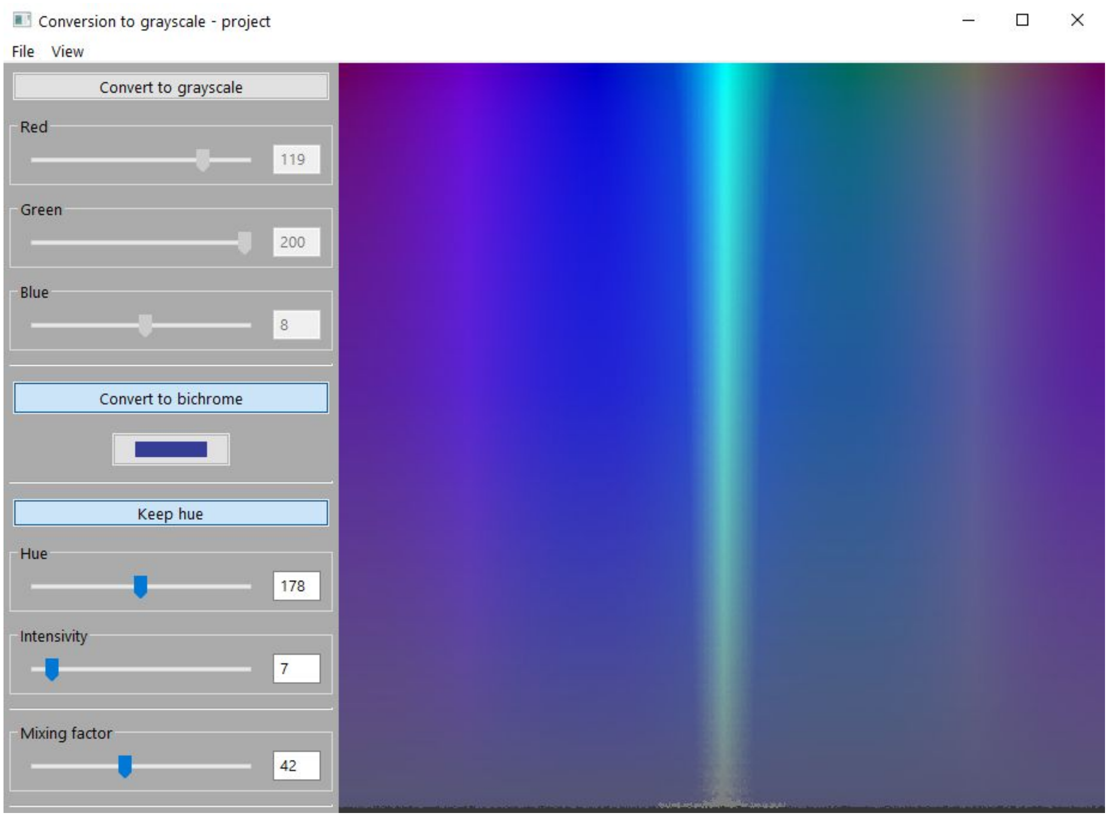
Conversions functionality was tested with special rainbow images:



Red 119
Green 200
Blue -119

Hue 241
Intensivity 5
Mixing factor 0

Test 1: Test of conversion to greyscale and kue keeping. The test was conducted on radial HSL hue palette with fully saturaed colours.

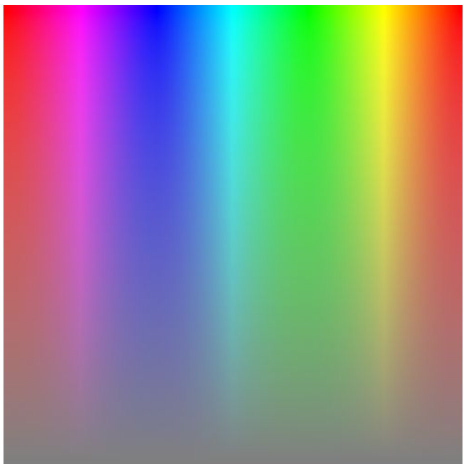


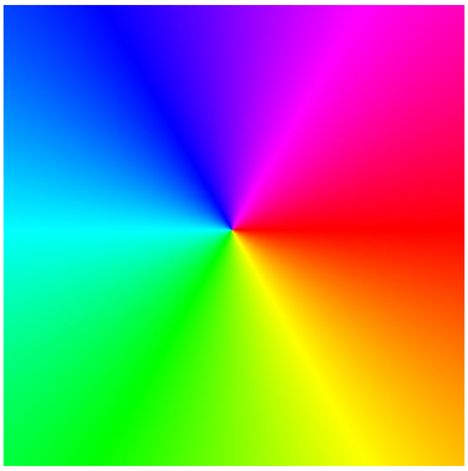
Red 119
Green 200
Blue 8

Hue 178
Intensivity 7
Mixing factor 42

Test 2: Test of bichrome conversion with keeping hue and mixing factor. The image was a rainbow gradient with full HSL satration on top and no saturation at bottom.

The original images are shown below:





9. Deployment, conclusions, report

Program is working correctly, all goals was reached. However there is room for future improvements. Tha main feature might be better conversion for low saturated colours. The current artifacts are caused by HSL conversion algorithm where for low saturation values are nearly 0 and we lose precision while operating on them.

Splitting panel could be added so that on one side there is original image, and on other side - converted image. The bar separating both images would be movable.

Another feature could be layer system. In current version, the program can mix original and converted image with keeping selected hue, but can’t mix more conversions like keeping two hues or combine two bichomes. Layers might save every transformation separately so they could be mixed. Every layer would have its own opacity value.