Assignment I: Illustrated Report

Garrenlus de Souza
Informatics Institute
Federal University Of Rio Grande do Sul - UFRGS
Porto Alegre, Brazil
gsouza@inf.ufrgs.br

I. Introduction

More than three decades past the start of the desktop publishing revolution, the application of image editing techniques is now quite widespread with software products aimed to meet the demands of the quite wide spectra composed with the demands of modern society. This work fits somewhat in the education/learning range as commercial factors such as portability, performance and user experience received the least amount of work needed to achieve the specifications given by the course instructor.

The development process of each feature is described revolving around key aspects covering rationale, implementation and obtained results. Some understanding about the employed resources (*e.g.:* tools, programming language, etc.) is expected from the reader, although references were generously added along the text in order to better guide the adventurous.

Godspeed.

II. Setup

The requested functionalities were implemented in C++ 11[1] under a Linux Operating System[2] environment. The text editor of choice is Visual Studio Code[3]. The GCC[4] compiler suite was employed alongside CMake[5] to compile and link the source code to the libraries needed. OpenCV[6] C++ was used as a means to provide multiple image formats functionalities and some of the means necessary to handle image data in a clean way. Qt5 on the other hand was used in the development process of the Graphical User Interface[7], providing widgets, buttons, windows and the input callback functionality needed to show the image processing implemented features in a more user friendly way.

A. The Image Class

In order to better capture the relationship between the requested functionality and the underlying processed data, some Object Orientation[8] concepts were explored. The storage container utilized to handle image data was joined by the relevant data and desired behavior into what ended up being the Image class, whose source code can be found below.

```
class Image {
private:
    std::string _file_path;
    cv::Mat _matrix;

public:
    Image(std::string filePath);
    Image(const Image &other);
    Image &operator=(const Image &other);
    Image &mirrorH();
    Image &mirrorV();
    Image &toGrayscale();
    Image &quantize(int noTones);
    const cv::Mat &underlyingContainer();
    bool saveToDisk(std::string filePath, int
        quality);
    std::string info();
    std::string getFilePath();
};
```

III. Features

A. Reading and Writing

The input and output functionality was built into the *Image* class through a constructor and member function respectively. The image file is read into a *cv::Mat* object that is stored as a class member in construction time, as can be seen in the code fragment shown below.

```
Image::Image(std::string filePath):_file_path(
    filePath){
    _matrix = cv::imread(filePath);
}
```

A functionality analogous to the ubiquitous *"save to disk"* was achieved through the *Image::saveToDisk* member function shown right below. While performing some some inconsistencies were observed between the original file size and the one written to the storage after processing. The first tests did not involved any parameter passed to *cv::imwrite*. So the default parameters were used by *OpenCV* that applied a 95 coefficient to the JPEG[9] algorithm, and no optimization features of the standard (so the initial code lacked the parameter effects of line 2 in the fragment below).

```
1  bool Image::saveToDisk(std::string filePath,
       int quality) {
2      std::vector<int> compression_params = {cv
           ::IMWRITE_JPEG_QUALITY, quality, cv::
           IMWRITE_JPEG_OPTIMIZE, 1};
3      return cv::imwrite(filePath, _matrix);
4  }
```

JPEG [6] is quite known for the amazing compression rates it can achieve so after some tests using the presented code instead, it could be concluded that when going below some picture-specific threshold, the transformed image file size could be considerably reduced.

B. Mirroring

The main ideia behind the implementation of mirroring revolves around using pointers to regions in memory related to the relevant opposite sections in the image and run some sort of swapping between them, achieving at the end the desired result.

1) Vertical: The vertical mirroring transformation can be achieved by swapping image element rows which are opposite to each other in the y-axis. *cv::Mat* objects store image element channels grouped linearly with respect to how rows are laid out in memory. Assuming A and B to be vertically-opposite image rows, one could swap their data using a large enough buffer and some clever pointer arithmetic. The proposed approach is shown in the code fragment depicted below. Notice that we loop over at most half the total amount of image rows (line 1) and effectively avoid neutral operations (*i.e.:* swapping a row with itself).

```
1  for (size_t i = 0; i < size.height / 2; i++){
2      row = _matrix.ptr(i);
3      int a_row = i;
4      int b_row = size.height - 1 - i;
5      if (a_row != b_row){
6          b_buffer = _matrix.ptr(b_row);
7          memcpy(buffer, row, row_blength*sizeof
               (uchar));
8          memcpy(row, b_buffer, row_blength*
               sizeof(uchar));
9          memcpy(b_buffer, buffer, row_blength*
               sizeof(uchar));
10     }
11 }
```

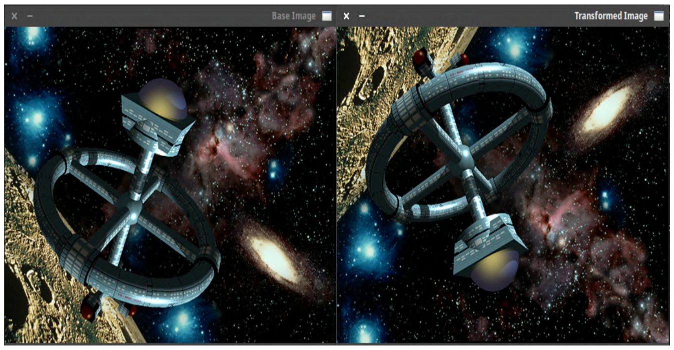


Fig. 1. In the left we can see the original picture and in the right the resulting vertically mirrored transformation.

2) Horizontal: The proposed approach for achieving a similar effect in the horizontal direction is bound by the image elements memory organization. That way the amount of data that can be swapped at once (with respect to *memcpy* calls) is steeply reduced to the size of a picture element (*i.e.:* in this case the total amount of bytes used to encode the RGB channels). A similar strategy is proposed to address the possibility of a swap being applied wastefully (line 5).

```
1  for (size_t i = 0; i < size.height; i++){
2      row = _matrix.ptr(i);
3      for (size_t j = 0; j < size.width/2; j++){
4          int b_column = size.width - 1 - j;
5          if (j != b_column){
6              a_buffer = row + j * pixel_size;
7              b_buffer = row + b_column *
                   pixel_size;
8              memcpy(buffer, a_buffer,
                   pixel_size * sizeof(uchar));
9              memcpy(a_buffer, b_buffer,
                   pixel_size * sizeof(uchar));
10             memcpy(b_buffer, buffer,
                   pixel_size * sizeof(uchar));
11         }
12 }
```

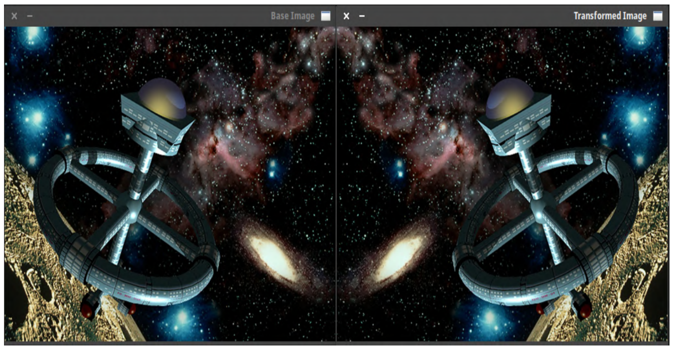


Fig. 2. In the left we can see the original picture and in the right the resulting horizontally mirrored transformation.

C. Grayscale

The grayscale transformation proposed is quite simple and consists primarily as a linear combination that takes the Red(R), Green(G) and Blue(B) channels as variables. Where L corresponds to the resulting *Luminance*, and R,G and B, to the values corresponding to each channel previously mentioned.

$$L = 0.299 * R + 0.587 * G + 0.114 * B \tag{1}$$

This operation is performed for each picture element and the obtained value is then written to every channel as shown in lines 7, 8 and 9 in the code section presented below.

```
1  for (size_t i = 0; i < size.height; i++){
2      row = _matrix.ptr(i);
3      for (size_t j = 0; j < size.width; j++){
4          offset = j*pixel_size;
5          L = 0.299*row[offset+2] + 0.587*row[
               offset+1] + 0.114*row[offset];
6
7          row[offset] = (uchar) L;
8          row[offset + 1] = (uchar) L;
9          row[offset + 2] = (uchar) L;
10     }
11 }
```

Notice that the channels are laid out in memory using the reverse order when compared with the commonly used RGB acronym.

D. Quantization (from grayscale)

The proposed approach assumes that the base image was transformed with a grayscale translation identical to the one presented in the previous section. It consists in a mapping from the original tone values to a (strictly smaller) user parameterized amount of target new tones. This range is

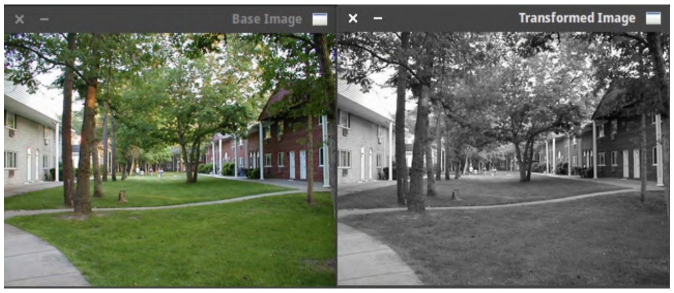


Fig. 3. In the left we can see the original picture and in the right the resulting grayscale transformation.

defined around reference values centered inside sections called *bins*, which in turn are defined as follows:

$$[t_{min} - 0.5 + \alpha * binSize, t_{min} - 0.5 + (\alpha + 1) * binSize) \quad (2)$$

Where t_{min} is the lowest tone value found in the picture and t_{max} is the greatest. In order to calculate the quantized value for a given pixel element (remember that every channel has the same value) one would need to solve for α as it defines the *bin* in which the original value must be mapped. Then one needs to round the center of the bin to the nearest integer that still falls inside the defined range. The code below implements the aforementioned algorithm for the proposed quantization. Notice the pointer arithmetic, the same applied to the mirroring features, and the variables *alpha*, *bin_center* and *quantized_value* (lines 6, 7 and 8 respectively).

```
1   for (size_t i = 0; i < size.height; i++) {
2       image_row = _matrix.ptr(i);
3       for (size_t j = 0; j < size.width; j++) {
4           offset = j * pixelSize;
5
6           alpha = floor((image_row[offset] - (
                tmin - 0.5)) / bin_size);
7           bin_center = alpha * bin_size + tmin -
                .5 + bin_size / 2;
8           quantized_value = uchar(round(
                bin_center));
9
10          image_row[offset] = quantized_value;
11          image_row[offset + 1] =
                quantized_value;
12          image_row[offset + 2] =
                quantized_value;
13      }
14  }
```

The user can define how many bins are going to be used and the end result is an image with grayscale tones remapped evenly across the original interval.

E. User Interface

1) Graphical: The development process of a Graphic User Interface (GUI) was remarkably difficult. Not by the usability of the employed libraries in terms of actually integrating them with the underlying back-end code. Building errors, paths to headers and version inconsistencies led to major delays in the production, causing the entire project to fall behind in the schedule. The library of choice was Qt5[10] but by the time it was integrated into the development process (after trying twice, with no success, before and after the development of the back-end image processing features) there was not much time available to design the window management logic needed to present the proposed functionalities. Nevertheless a valuable lesson was learned when it comes to unexpected sources of complexity that only show themselves once the observer is done through a reasonably amount of the path towards the realization of an idea.

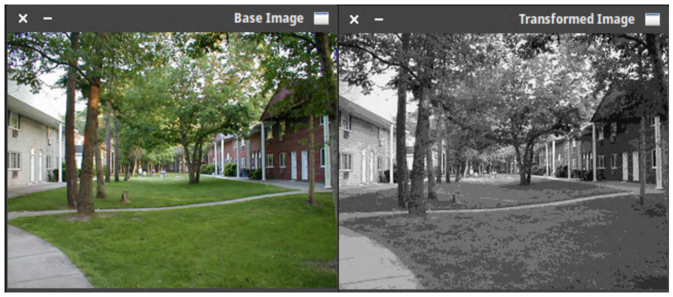
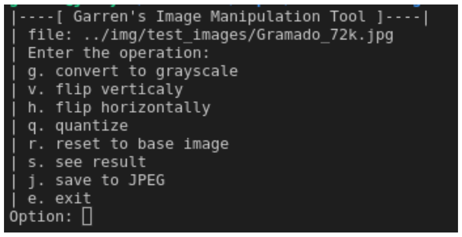


Fig. 4. In the left we can see the original picture and in the right the grayscale transformation with tone values mapped from 256 to a set of 8 tones. Notice the loss of details in smoother color transition areas such as the grass and the darker spots under the roof.

2) Command Line: In order not to fail on presenting the features that were already built the alternative chosen was to implement a Command Line User Interface. It was done entirely in C++ 11 as the rest of the code, with no libraries and a simple menu that centralizes all the necessary functionality as shown in Fig. 5.



```
|----[ Garren's Image Manipulation Tool ]----|
| file: ../img/test_images/Gramado_72k.jpg
| Enter the operation:
| g. convert to grayscale
| v. flip verticaly
| h. flip horizontally
| q. quantize
| r. reset to base image
| s. see result
| j. save to JPEG
| e. exit
Option: 
```

Fig. 5. The application menu waiting for the user to input the desired option through the keyboard.

IV. Conclusion

The project source code was made available at Github[11] from the start and by the time of this writing there are at two main development branches[12], focused in each one of the mentioned User Interfaces.

References

[1] "C++ 11," https://en.wikipedia.org/wiki/C%2B%2B11, Accessed: 2022-02-14.
[2] "Linux," https://en.wikipedia.org/wiki/Linux, Accessed: 2022-02-14.
[3] "Visual studio code," https://code.visualstudio.com/, Accessed: 2022-02-14.
[4] "Gcc," https://gcc.gnu.org/, Accessed: 2022-02-14.
[5] "Cmake," https://cmake.org/, Accessed: 2022-02-14.

[6] “OpenCV, howpublished = https://opencv.org/, note = Accessed: 2022-02-14,” .
[7] “Graphics user interface,” https://en.wikipedia.org/wiki/Graphical_user_interface, Accessed: 2022-02-14.
[8] “Object oriented programming,” https://en.wikipedia.org/wiki/Object-oriented_programming, Accessed: 2022-02-14.
[9] “Jpeg,” https://en.wikipedia.org/wiki/JPEG, Accessed: 2022-02-14.
[10] “Qt5,” https://www.qt.io/, Accessed: 2022-02-14.
[11] “Github,” https://github.com/GarrenSouza/INF01046-Assignment_I, Accessed: 2022-02-14.
[12] “Git,” https://git-scm.com/, Accessed: 2022-02-14.