Image Processing C++ Tool

Adrien Ruault, Guillaume Mollard

1 Introduction

The idea of this small library is to use Fourier transforms to modify either greyscale or colored JPEG images by applying different kind of filters. To do so the program rests on an external library called CImg [1] whose purpose is to extract the pixel intensities of an image. The library only

2 Requirements for running the program

To compile the program the user needs a gcc compiler (version 4.7.2 or above) and the CMake software (version 2.6 or above) to link the files. Moreover the ImageMagick® software is needed to save and display images. Note that the CImg library that is used to extract pixel intensities is already included in the program directory (see CImg.h).

3 Typical execution and usage

The program can produce a user-friendly executable from the file *user.cpp*. This executable is made so that the user is able to only interact with the console in order to use all the available features. Otherwise the typical way of proceeding is to first open an image by instantiating a class template *Image<P>* with P being either a *PixelBW* object for greyscale pixels or a *PixelRGB* for colored pixels. Then several options are available:

- Creating an intensity histogram by instantiating a *HistogramBW* or a *HistogramRGB* object depending on the image type.
- Applying a filter on an image. This requires first to compute the *DFT* by instantiating a *FourierTransform<Image <P >>* object. Then a *Filter* object must be instantiated according to the desired characteristics (Low, Band or High-pass) and finally call the *Apply* method that returns the filtered image.

4 List of features

Pixel (Class) Pixel is a class with two derived classes: PixelBW and PixelRGB. It is the building block of our *Image<P>* and contains the intensity of each channel.

Image<P>(Class Template) *Image<P>* is the class based on Pixel that reads images to extract their intensity. It is mainly based on an array of Pixels. This is the only part of the code that uses the CImg library: in its constructor to extract intensities, to save images and to display images.

HistogramBW and HistogramRGB (Class) These classes derive from *Image<P>*. They are used to create histograms of the number of pixels associated to a particular intensity versus the intensity ranging from 0 to 255. For colored image it will produce three histograms, one for each channel, that is to say: RGB. The histograms are built from an image with their copy constructor.

FourierTransform<Image<P>>(Class Template) This class is used to compute the *DFT* of an image and produce maps for modulus, argument, real part and imaginary part. It is also possible to instantiate it with an array of complex numbers which do not require *DFT* to be calculated.

Filter (Class) The *Filter* is used to create different kinds of filters using its constructor. Indeed it is possible to instantiate Low-Pass, High-Pass and Band-Pass filters with desired values of cut-off and sharpness to work in the frequency space. The idea is then to use the *Apply* method on a *FourierTransform* object so that it returns a modified *FourierTransform* with the frequencies of interest. We then have to go back in the space domain to obtain our filtered image. Note that the class can alternatively filter in the real part by convolving a mask, if the user provides a plain image to the *Apply* method. To do so it takes the cut-off matrix that is usually used to filter in the frequency space and apply inverse FFT on it to produce a matrix which represents the mask we use in the spatial domain. After computing the mask, the later is convolved on the real Image. All this process seems to filter the image in a similar way as the Fourier filter.

5 List of tests

Histograms The histograms are tested by first displaying the histograms of greyscale and colored images. Then the intensities of each pixel channel is set to a given value so that the histograms only show a peak corresponding to that intensity.

Fast-Fourier-Transform The forward *FFT* and the inverse *FFT* correctness is verified by computing the *DFT* of greyscale and colored images. These transformed images are then computed back to the space domain and displayed to the screen to show that they are unchanged by comparison with the initial images.

Filtering The three available filters are applied to both greyscale and colored images. The tests display plots of modulus and arguments in the frequency domain before and after filtering. They then display the filtered images to show enhanced contours for High-Pass and blur for Low-Pass.

6 Running the tests

Histograms Histograms are tested with *test_histogram.cpp*. Simply compile it and run the executable to get the results.

Fast-Fourier-Transform To check the *FFT* and inverse *FFT* behaviour for both greyscale and colored images, just compile and run the following files: *test_fft.cpp* and *test_fft_color.cpp*.

Filtering To test the filtering behavior for each filter type (that is three) and each image type (that is two), just compile and run the following files, the three first ones being for greyscale images: *test_low_pass.cpp*, *test_high_pass.cpp*, *test_band_pass.cpp*, *test_low_pass_colored.cpp*, *test_high_pass_colored.cpp*, *test_band_pass_colored.cpp*.

7 Perspectives

The filtering is of better quality when carried out through a convolution in the real part but it takes more computational time. Indeed, we observe some boundary effects on the filtered image when the filtering is processed in the frequency domain. The result is better in space domain, in particular thanks to the mirror boundaries we can add with the Image class. However, since we limit the size of the influence of our convolution to 31 not to have even longer time of computation, we may lose some filtering at very low frequencies, but the overall result is most of the time similar.

It is important to well define the parameters when constructing a filter, in particular the sharpness, since a too low one is susceptible to produce Gibb's effects on the filtered image. Also, since we use the FFT algorithm for the comptation of our Fourier Transform, our images must have a dimension of 2^m x 2^n.

8 Notes

The ComplexNumber class of our project is inspired from [2] and the *FFT* algorithm is inspired from [3]

References

[1] David Tschumperlé. The CImg Library.

[2] Joe Pitt-Francis and Jonathan Whieteley. *Guide to Scientific Computing in C++*. Springer London Dordrecht Heidelberg New York.

[3] John Tukey. The Fast Fourier Transform, 2012.