INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

PFE

End-of-Study project

presented by

Yu Liu

Engineer student of INSA Rennes

Department EII

Scholar year 2019-2020

Improvement and optimization of software <ImageINSA>

Location of the End-of-Study project

INSA Rennes (telecommuting)

Tutor of the End-of-Study project

Luce Morin and Julien Heulot

Pedagogical correspondent INSA

Luce Morin

PFE presented on 20/10/2020

Contents

Acknowledgement

First, I would like to thank my tutors, Mrs. Luce Morin and Mr. Julien Heulot. They selflessly spend a huge amount of time and energy to have meeting with me every week and put forward valuable suggestions and advices. The progress of my project is closely related to their guidance and availability, I want to express my sincerest thanks to them.

I would like to thank Mrs. Muriel Pressigout, responsible in charge of internships for fifth grade students. Under her advice and approval, I could obtain the opportunity to complete the project in such a difficult period severely affected by the epidemic.

Finally, I would like to thank the entire EII department for educating me, through solid theoretical study and substantial practical experience acquired from school, I was able to carry out this project.

Introduction

As an important learning approach, image processing has always been one of the focus of EII department. To show students more intuitively the methods and operations encountered in theoretical study, EII has developed itself an image processing software called ImageINSA.

Concepted towards a convenient and efficient tool, the software carries the entire content of the practical courses. To meet the educational requirements, it has been continuous improved in recent years.

However, there are still some problems and shortcomings found by users during practical courses. In such case, this project has been proposed to fix some already known bugs which may lead to crash, to settle several lack of functionalities and to correct some inappropriate contents of the user interface, moreover, an automatic test set for the software is waiting to be completed, in order to strengthen the robustness of the program and facilitate subsequent development.

This report will not only describe the work which has been completed during the project but will also analyze the structure of the software and explain how functional components are designed, how they are connected through the services and operations modules. And from the perspective of the whole project, how classes defined in different location communicate to each other through the Qt’s signal and slot mechanism.

This report starts from the history of the software, then explains the reasons for choosing Qt as development platform, next, it analyzes the structure of the software as well as the role of each component, the importance of CMake in project construction will also be introduced. The explication of the improvements to the software is given through a few examples, and the construction process of the test framework will be introduced at the end.

History of ImageINSA

2003: IMAGE EII

Developed internally by EII department in 2003 as an image processing software, it was initially called Image EII. This first version is strongly based on the Microsoft Visual C++ 6 environment and its MFC 6.0 library. It was therefore impossible to deploy it on systems other than Microsoft Windows. In addition, this technology was established in 1998, its utility was insufficient comparing to the Visual C++ .NET framework released in 2002. The software became soon very difficult to maintain according to its original design. Therefore, the needs to bring this project to more recent technologies become essential.

2009: IMAGE EII

The first attempt to implement the software to C ++ & Qt technologies dates to 2009. A group of EII students took this charge during their 5th year's project. The mission was complicated, because the graphical interface is based on MFC technology and the entire source code of Image EII is based on VC6, which unfortunately had no support for the ANSI C99 standard.

This project gave birth to an unfinished software, equipped with a part of the functionalities defined in original software but most of the processing algorithms had not been implemented.

2011: IMAGE EII

The second attempt started in 2011, within an end-of-study internship of a 5EII student. He took the response for completing the porting of Image EII which started two years ago.

The architecture of project was relatively improved, many missing contents had been added and most of the errors were corrected, it became much more robust. The result given was a functional software, but some functions and algorithms were taken away comparing with the original software.

2012-2013: detiq-t

A 4th year INFO project called detiq-t was realized by a group of 6 students. The main achievement of this project includes two complete libraries, ImageIn and GenericInterface. The first provides services related to image modeling and processing, the second offers graphical interface related tools. Their structures are presented in detail in the following section. The software was coded in C++ and the graphical interface was constructed under Qt.

2012-2013: EIIMAGE

The eiimage project was proposed as a summer internship aiming to reconstruct IMAGE EII based on the detiq-t project. Many functions had been added to GenericInterface, a dynamic library named core was created and the add plugin function was re-implemented. The architecture ensured no dependency between the extension module and the final application.

EIIMAGE has been used for practical work in EII department since early 2013. Unfortunately, the source file was lost, only the executable file left for this version.

2014-2015: ImageINSA

This project was carried out by a research engineer at INSA. The lost source of EIIMAGE was restored according to the executable version using in practical courses. Several bugs were fixed, and the compilation was since then managed by CMake.

The software changed its name to ImageINSA and the project repository was created under GitHub repository to enable the access for users outside of INSA.

2016-2017: ImageINSA

A summer internship was realized by a 4EII student, the content of this internship included error corrections and interface improvements, several new features had been added to the software.

2018-2019: ImageINSA

A 3EII and a 3INFO student worked together in summer internship to maintain the software. One worked in the algorithm filed, optimized the existing operations and proposed new features. The other concentrated on the user interface improvements, the test class of interface was created by him and many remaining bugs were solved.

Why Qt

In this section we discuss the advantages of Qt and the reasons it was chosen as the development platform for this project.

As a cross-platform application development framework extendedly developed on C++, Qt possesses a large number of available tools for graphical interface development. Its advantages can be summarized in the following six points :

1. Cross-platform

Qt can not only run on Windows, but also on various platforms such as Linux and Mac OS, which means that programs written with Qt are able to migrate to multiple platforms with affordable modifications.

Writing software for multiple platforms is trivial and errors may occur at any time. This is especially true for maintaining build files, through the qmake tool, Qt is able to face this challenge well. Qmake can generate accurate and error-free compilation files for the target platform. (Although qmake is good enough, CMake is still chosen to build the project, the reason will be revealed later)

2. Open Source

Qt provides a dual software licensing model, under this model it can be used to develop proprietary software under commercial licenses as well as the open source software under general public licenses.

3. Rich Libraries

In addition to providing interface libraries, it also provides audio libraries, web engines, 3D libraries, database SDKs and so on, which allows us to easily make cross-platform programs. Although most of the libraries are not used in this project, it still provides a variety of attractive possibilities.

4. Expansibility

Qt itself can be called as an extension of C++ since its classes are written in C++, which means it inherits many advantages of C++, such as fast and object-oriented. It is easy to extend and allows real component programming.

In addition, it uses special code generation extensions such as moc (meta object compiler) and uic (user interface complier). Before compiled by the standard gcc compiler, the C++ source file must be analyzed with moc. If the macro Q_OBJECT was found, a new source file that contains the implementation of corresponding macro methods will be generated, the new file will have a “moc_” in front of the original name and will also enter the compilation system before being linked into the binary code.

5. Robust architecture

In a Qt project, the interface source code is separate from the program source code, which means the program logic is isolate from the creation and setting of interface control objects. Qt's strategy is to generate the corresponding source code as if there is no interface to implement.

Qt runtime does not need to rely on virtual machine like Java, without simulation layers or large-capacity runtime environments, it can directly write low-level graphics functions at a satisfactory speed.

6. Qt Signals & Slots mechanism

This feature is listed separately as an advantage because its practicality and efficiency.

For the convenience of event processing, Qt provides flexible signal and slot mechanism, which can reduce coupling and improve code reusability. When an event occurs, for example, a button has been clicked, it will send out a signal. This kind of transmission is purposeless like broadcasting. If an object is interested in this signal, it will call the connect function, which means that it binds the signal to be handled with a slot function which gives a reaction after receiving the signal. In other words, when the signal is sent, the connected slot function will automatically be called back.

A slot is actually a member function of class, and it can take any type of parameters just like a normal C++ function. It can even be a virtual function and be overloaded. The only difference is that a slot can be connected to a signal and it is called whenever a signal bound to the slot is emitted.

There are some rules that I summarized to be aware of when deploying this mechanism:

(1) A signal can be connected to multiple slots but the calling order is uncertain, additionally multiple signals can be connected to one slot. Both the sender and the receiver need to be subclasses of QObject.

(2) The signal is a function declaration without function code, it can be connected to another signal.

(3) Slot functions are member functions of a class which means they will be affected by public, private, and protected assignment.

(4) Qt automatically cancels all slots connected to the object when it is deleted.

(5) Lambda expression can be used as a slot function.

Structure of ImageINSA

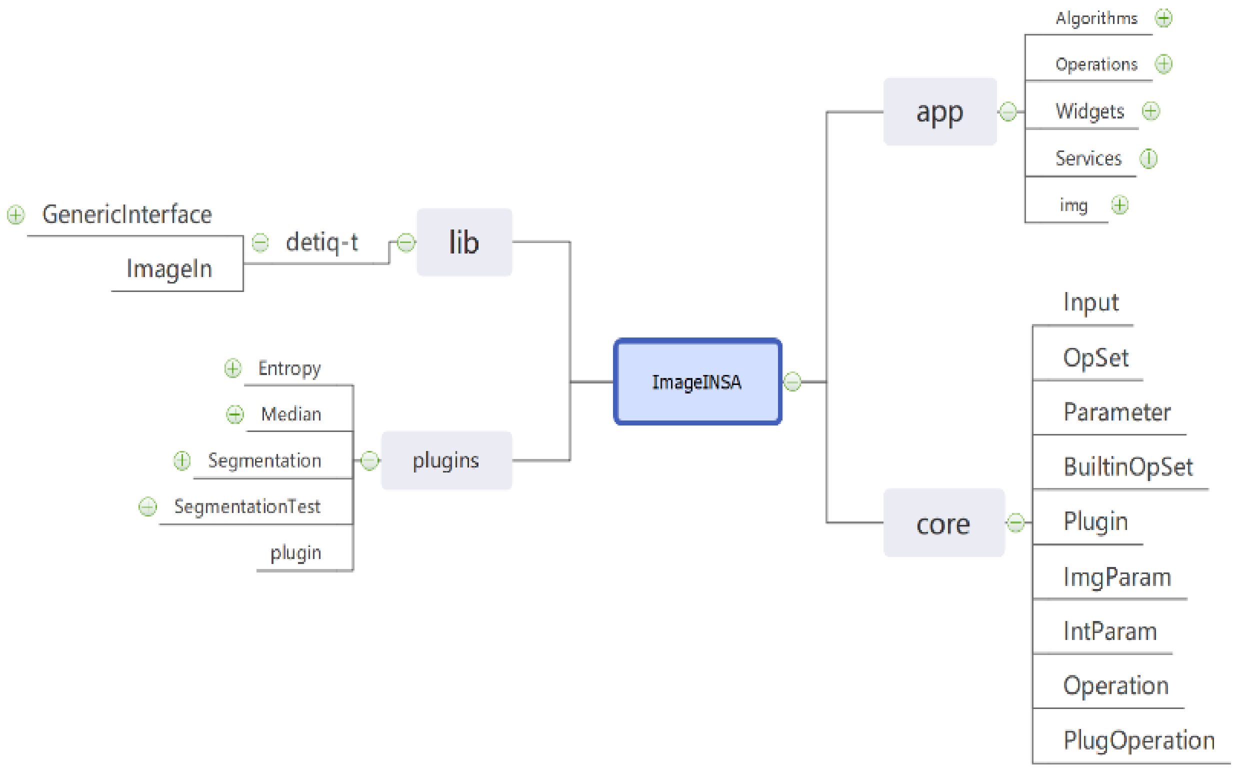


Figure 1 : The structure of ImageINSA

ImageIn

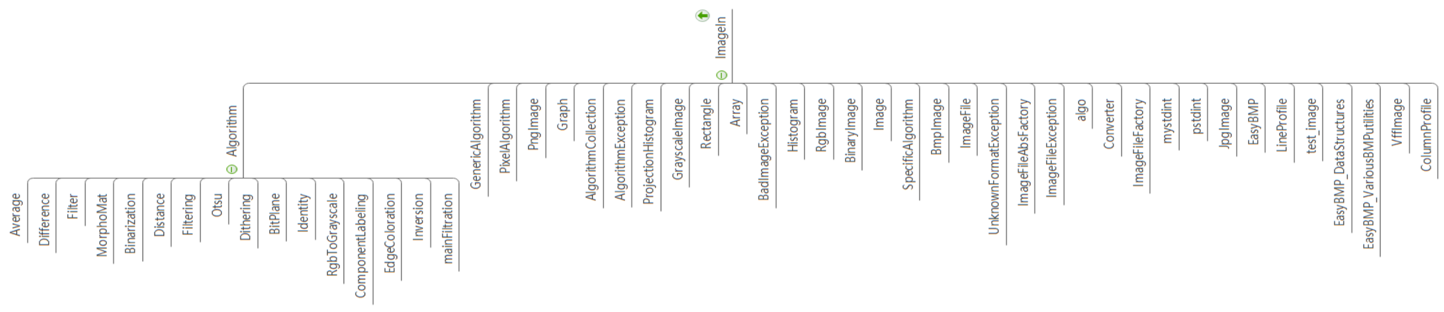


Figure 2 : Contents of ImageIn library

ImageIn library provides a set of basic operations for manipulating raster images and applying algorithms to them. The basic concept of image class includes a three-dimensional matrix: width, height and number of channels, all raster images can therefore be represented by this class (GrayscaleImage and RgbImage inherits the Image class, but HSVImage inherits directly the Qcolor provided by Qt).

ImageIn allows to apply algorithms to images. The algorithms are defined in the form of functor objects, they may take parameters in the constructor function and can be applied via the function operator(), a pointer to the newly allocated result image will then return. If the image passed in parameters differ from the correct type, an exception will be thrown. To ensure the operation take images in a correct format, images can be converted to the desired type via Converter class.

ImageIn requires external libraries (libjpeg and libpng) to support image processing within JPEG and PNG format (BMP image can work independently without external library) and it overrides some methods in these libraries to redefine the error manager structure and its handler. When working with these libraries, program exits directly when a fatal error occurs, but it is better to raise an exception instead.

The ImageIn library is also very genericity oriented, the concept includes an extensive use of templates and inheritance. Most of the classes in the library have one or more template parameters, so data structures, images classes and algorithms can therefore be reused for other types beyond the original design. It also provides many typedefs to ignore the template parameters and use classes directly without suffix.

To illustrate its extensivity, here are two examples of creating new image format and new algorithm:

ImageFile class is an interface to open image files in different formats. To add a new image format, the class inherits ImageFile including the specification of the new format needs to be created, a new class inherits ImageFileFactory is also necessary to enable the instantiation of the newly created ImageFile. And finally, the setFactory function will enable the use of this new factory. With this kind of settings, the new image format becomes acceptable to the software.

As the main and most useful algorithm interface, Algorithm_t template is developed to enable the implementation of an algorithm by redefining only the method corresponding to the algorithm. The various overloads of the application operator are already equipped, only the return type needs to be changed in order to adapt to the type designated by template parameter. If the new algorithm needs other parameters besides the input images, the constructor should be overloaded to remember these parameters.

GenericInterface

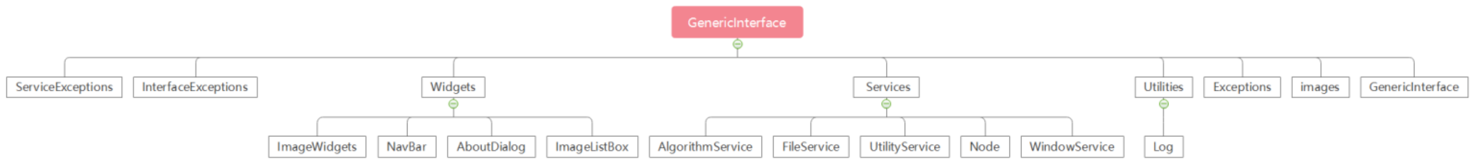


Figure 3 : Contents of GenericInterface library

Genericinterface serves as the basis for all types of operations that need to be implemented. Its main role is to design a framework of interface that unify the way to implement operations and restrict them towards a common behavior.

The structure of the genericinterface inherits QMainWindow as the main entrance of the program, all functional modules of the program are initially launched through the tool bar that it provides.

A pure virtual class Service is defined to connect services to its owner (genericinterface). The two most used services are windowService and fileService. For example, windowService manages all window-related activities, through this service, it is possible to identify different windows according to their ids, all feasible operations can be done with this unique id, such as changing window title or updating the display image. Apart from the QMainWindow, the genericinterface inherits also QMdiArea, which draws the windows that it manages and arranges them in a cascading or tile pattern.

The navigation bar, on the left, is the central element that governs the policy of automatically minimize or maximize windows. When an image is opened, it will be added to the navigation bar. When a new window popped up after a certain operation, it is linked automatically to its original window. After a click on a thumbnail in the navigation bar, all the linked windows are brought to the foreground, the others are minimized, it is even possible to make a multiple selection by keeping the control key pressed.

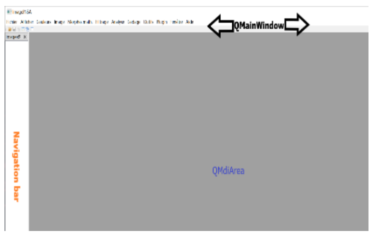


Figure 4 : The arrangement of the mentioned components

Core

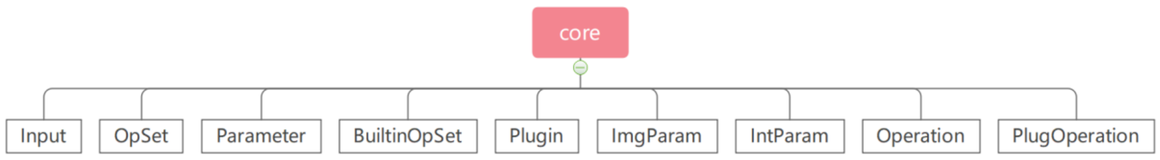


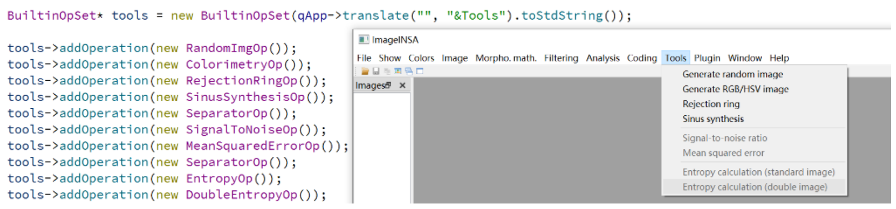
Figure 5 : Contents of Core library

As described by its name, the core part of the program is defined here, it is called GenericOperation. Each GenericOperation run an image-processing algorithm, it may take images and other parameters as input and return some images, text, or other widgets as output, this can be customized according to different needs.

Operator is a pure virtual function defined in GenericOperation, used as the entry point of every operation, all applications containing the implementation of an operation need to override this function.

An OpSet is a set of GenericOperation objects, represent the instances of different implementations of the GenericOperation.

The upper class of an OpSet is called BuiltinOpSet, it is an implementation of OpSet and it is used inside the main application to organize the built-in operations.



```
BuiltinOpSet* tools = new BuiltinOpSet(qApp->translate("", "&Tools").toStdString());

tools->addOperation(new RandomImgOp());
tools->addOperation(new ColorimetryOp());
tools->addOperation(new RejectionRingOp());
tools->addOperation(new SinusSynthesisOp());
tools->addOperation(new SeparatorOp());
tools->addOperation(new SignalToNoiseOp());
tools->addOperation(new MeanSquaredErrorOp());
tools->addOperation(new SeparatorOp());
tools->addOperation(new EntropyOp());
tools->addOperation(new DoubleEntropyOp());
```

Figure 6 : How operations are added to the user interface

In summary, the whole set of Core defines the drop-down menu of the software. As a standard entry point for all kinds of available operations, it contains all the components related to the organization of its menu.

App

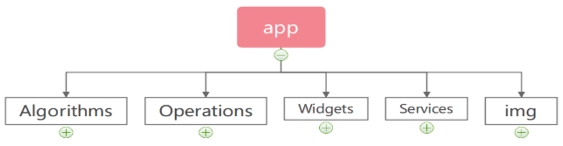


Figure 7 : Contents of App library

App is an extension of the core part, it contains all the implementation details of algorithms and operations, comply with the framework which has been defined in Core to present the available content to the user. The most used structure to implement an operation is writing in three separated parts.

To illustrate how an operation is implemented, we take DCT operation as an example:

The DCTDialog.ui describes the graphical interface, which shows the available options to user and receives the requirements entered by the user.

The DCTOp.cpp is then used to define the behavior of each element showed in the graphical interface, the operation to perform depends on the options obtained in the ui class, the useful input information will be extracted and saved as variables. It is also responsible for managing the output.

The correspond arithmetic part is registered in Algorithm folder, DCT.cpp will be called by Op class to do the real calculation, the result is then returned to Op and finally, the encapsulated result will be displayed to the user.

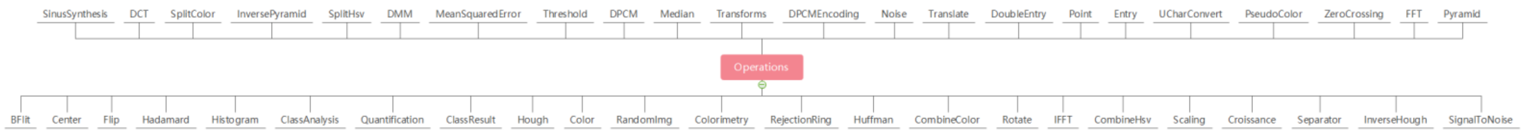


Figure 8 : List of operations defined in App library

CMake

At the beginning of this project, a lot of time was spent to compile the source file of the program. I got many failures because multiple external libraries that the program depends on needed to be compiled at first, since my working environment is on windows, several versions of Qwt library fail to compile, even if it succeeds, the path cannot be automatically identified. I still need to manually add its path to the environment variable.

After the prompt message kept warning me that the relevant FindQwt.cmake cannot be found, I began to study the structure of the project carefully, look for the connector that binds different parts together. Until I recognized CMake as a powerful tool and learned its usage, I could successfully compile the project.

CMake is a cross-platform build tool that describes the compilation process of all platforms with simple commands. As we learned how to write makefile from school, makefile describes the compilation rules of the entire project. The series of rules it defines specify the compile order of files.

CMake will generate corresponding Makefiles suit for the current IDE. One strict reason that we do not use directly makefile is that our project is strongly based on Qt. Since it has an extended part of C++, it employs a special meta-object compiler and a user interface complier. The program code including Qt sources needs to be preprocessed before compiling, according to this feature, ordinary makefile are not applicable.

The qmake tool is manufactured by Qt Company to generate special makefile files for Qt. Similarly, qmake is also a cross-platform tool and could be used in non-Qt projects and independent of IDE. But CMake also supports Qt applications and the use of CMake is wider than qmake, so project chooses CMake instead of qmake. There might be another historical reason, CMake was released in 2000 and qmake in 2003. When the project started its migration from MFC 6.0 to Qt, the technology of CMake was more reliable.

The use of CMake greatly simplifies the workload, especially for large-scale project. With the help of CMake, it's possible to determine automatically the location of libraries and the library dependencies handling, executable files generation and static libraries creation can be programmed with one line of code, which greatly simplifies the project building process.

Modifications implemented in the project

Histogram

As an image processing application, the most used tool in ImageINSA is no doubt the histogram. It allows visualization by channels the number of occurrences of each pixel value appears in the image.

Close to this concept, horizontal (respectively vertical) projection histogram takes a numeric value entered by user as a threshold, it counts the occurrences of pixel value greater than the threshold on each row (respectively column) of the image.

However, this module requires new features because the current version has three imperfections. At first, the current histogram does not support zooming operation. Secondly, the existing histogram of double image is meaningless since the counting interval is fixed to 1, it would be better to choose the bin size independently. The last problem is that the cumulative histogram of double Image is not yet implemented.

Before trying to solve the problem, let us first understand the structure and components of the histogram, the graphic below shows the classes related to the histogram, from calculation of occurrences to the display window of histogram.

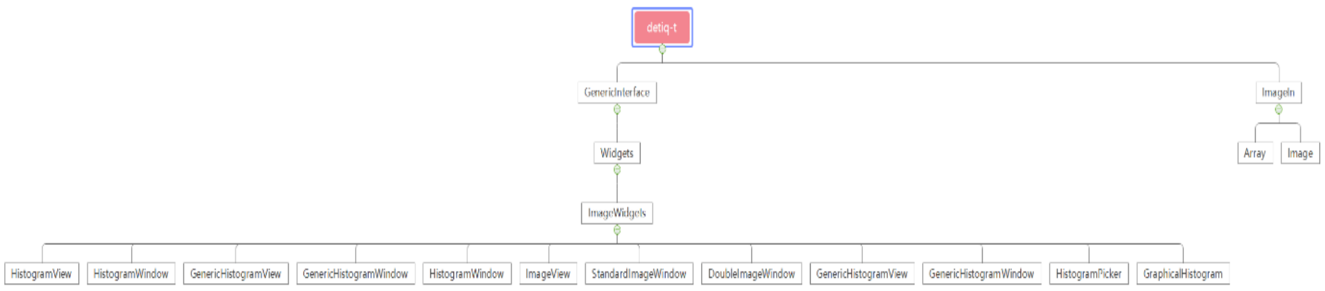


Figure 9 : Classes related to histogram

Structure of histogram

The class Array presents a fixed-size array, it is the basic component of a histogram because the number of occurrences of each value stores in an Array. The class Histogram inherits Array, as an extension of Array it enables the notion of channel.

To display graphically the calculated histogram, the project employs Qwt library to control the drawing part. Class GraphicalHistogram inherits the class QwtPlotHistogram to adapt to the situation.

Class GenericHistogramView inherits QWidget, proceeds a vector (for different channels) of GraphicalHistogram as protected member. It may create and display a graphical histogram from a selected zone of the target image.

GenericHistogramWindow inherits QWidget, it has GenericHistogramView as a private member. Its role is to add supplementary content in addition to the simple histogram display, such as the status bar which shows the L/R clicked value and hovered value.

Class HistogramWindow inherits GenericHistogramWindow, it stays at the top level of this structure with a role to switch the view to display (histogram or cumulative histogram) and change the window title based on the type of histogram.

Another class called HistogramPicker has also been created to handle mouse clicks on histograms, it inherits from QwtPlotPicker.

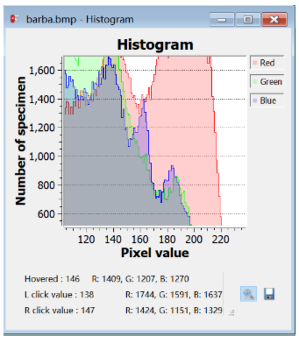
Class DoubleImageWindow inherits ImageWindow, it stores a double Image and its displayable version in uchar format. ImageWindow inherits QWidget, it is a basic and wildly used class describing image output.

The process of converting a double image to its displayable version will be discussed later. Respectively, class StandardImageWindow stores a standard image, they could both create and display an ImageView and update the status bar.

Add zoom widget to the histogram window

In order to integrate zoom operations to the histogram window, the easiest way is to employ a widget called QwtPlotZoomer in Qwt library. The working mode of zooming in and out can be customized.

In the initial concept which consistent with other software usages, left select un area to zoom in, right click to zoom out one step, ctrl & right click together returns the histogram to its original size.



```
Hovered : 146    R: 1409, G: 1207, B: 1270
L click value : 138       R: 1744, G: 1591, B: 1637
R click value : 147       R: 1424, G: 1151, B: 1329
```

Figure 10 : A histogram window

However, there were some conflicts in the definition of mouse events, originally left or right click shows the values related to the current mouse position to the status bar, which coincides with our zooming control. To avoid repeated definitions of right-click event, the zoom control needs be enabled by a QToolButton before applying the zoom function.

Limit the input range of bin size

The current implementation of double histogram is meaningless because every pixel has unlimited value possibilities, we cannot categorize them into statistics unless they are artificially divided into intervals.

So, the first task is to create a QLineEdit widget to receive the customized bin size value, it only appears when the displayed image contains double values. In order to avoid invalid calculation results from the bin size entered by users, a widget called QDoubleValidator is used to help customize the range of our bin size, according to the maximum and minimum pixel values in a double image. The upper threshold of bin size is selected to not exceed the maximum pixel value of an image and lower threshold is fixed to the value which leads to 1000 intervals.

Unfortunately, this widget does not work as expected, even with the QDoubleValidator employed, we can actually type any value into the QLineEdit.

After analyzing the source code of QValidator, I find that values outside the allowed range will only return an Intermediate flag instead of an Invalid flag. It only plays a role as a reminder and will not restrict illegal input. In order to improve its performance, I created its inherited class and override validate method to artificially limit its range. Considering the various situations of inputting a value, empty field and incompletely entered value are enabled, decimal position checking is enabled too, at the end, the input range is guaranteed by returning Invalid flag for entered values out of the limit.

However, the new created validator may achieve the goal at 99% but is theoretically still not perfect. For example, if the minimum value allowed is 0.755, and the bin size wanted by user is 0.78, before typing entirely 0.78, the user must enter 0.7 at first and the mechanism will not prevent user from doing this because it doesn't know if the next number is 5 or 7. It means although the minimum is set to 0.755, we can literally pass 0.7 to the program. Fortunately, such error will only appear when limiting the minimum value and has no significant effect on the final result.

Recalculate a histogram

After the custom bin size value is emitted as a signal, the correspond slot function needs to start a process to recalculate the histogram, and this requires read the information stocked in the original image. Therefore, the most suitable location of the slot is the DoubleImageWindow itself, where we could launch the recalculation directly when receiving a request.

Cumulative histogram of a double image is not implemented yet, the calculation method is somehow different from the traditional one, it needs to establish the ordinary histogram first and then compute the proportion based on the whole image. Since the same window is shared by these two histograms, they can be modified together. The first step is to turn on its operation entrance defined in class UtilityService, and then we can return to the calculation of the standard histogram.

The relationship between the classes has been introduced before, such modification of algorithm needs to start from the lowest level. Class Array is updated at first to prepare for limiting the range of bin size, two functions getvMin (respectively Max) are added to return the maximum or minimum values of the target image, respectively from the original getMin and getMin which return the extreme number of occurrences after counting.

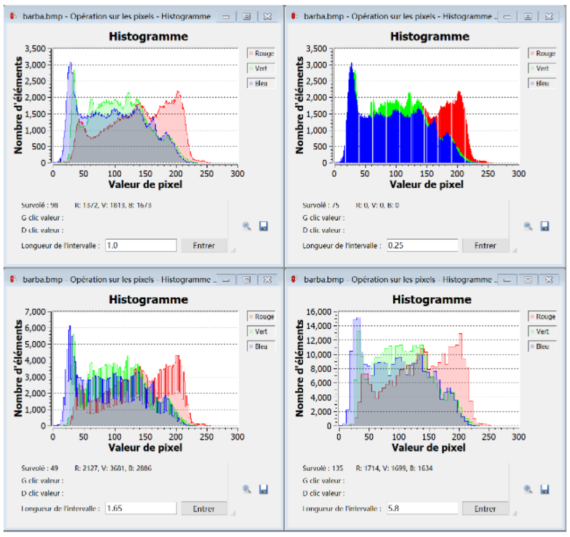
Class Histogram is the lowest level of the histogram calculation process, its constructor has been modified to support the new bin size element. A new computing method computeDoubleHistogram has also been established to determine the serial number of intervals that each pixel fits to.

```
/*Traversing the image again to find the corresponding position of each pixel value in the array*/
for(unsigned int i=rect.y; i<maxh; i++) {
    for(unsigned int j=rect.x; j<maxw; j++) {
        double pixel = img.getPixel(j, i, channel);
        int pos = (int)floor((pixel - _vmin)/binSize);
        if (pos >= size){
            pos = size-1;
        }
        _array[pos]++;
    }
}
```

Figure 11 : A part of computeDoubleHistogram method

In the same way, the obtained result needs to be divided by the image size and accumulated together to get the cumulative histogram result. After calculating the data, we still have to plot it and show graphically the result to users. For this purpose, modifications need to be done in the GraphicalHistogram class. Since it inherits the QwtPlotHistogram class, we need to process the data in its own way.

First, we need to create a vector of QwtIntervalSample. Each QwtInterval corresponds to a bin, the entire histogram is composed of bins, which is gain by dividing the maximum and minimum values into the customized bin size. After filling QwtIntervals with the calculated data, we can use the setData method to draw graphic histogram. Additionally, in all related classes, we need to add support for the new bin size variable, after doing continuous modifications layer by layer, the complete function is finally realized.



Survolé : 98 R: 1372, V: 1813, B: 1673
G clic valeur :
D clic valeur :
Longueur de l'intervalle : 1.0

Survolé : 75 R: 0, V: 0, B: 0
G clic valeur :
D clic valeur :
Longueur de l'intervalle : 0.25

Survolé : 49 R: 2127, V: 3681, B: 2886
G clic valeur :
D clic valeur :
Longueur de l'intervalle : 1.65

Survolé : 135 R: 1714, V: 1699, B: 1634
G clic valeur :
D clic valeur :
Longueur de l'intervalle : 5.8

Figure 12 : New histogram working with different bin sizes

File open error

The image file cannot be opened when its path contains character in other languages outside the ASCII table. The path itself was found to be displayed correctly in the error message prompt, it doesn't seem that the characters are parsed incorrectly during the cast (The file name was read as a QString and converted to std::string when opening by std::fstream).

By default, English characters are represented by 1 byte, while the characters of other countries (such as Chinese characters and French characters with phonetic symbols) cannot be represented by a single byte. Multiple bytes are used to represent these characters. The error was at first considered as a dislocation in decoding that produce garbled. In the test phrase, utf-8 encoding was kept during the whole transfer process, but still did not work. The root cause of this problem was found to be the Windows code page. Windows kernel has already adopted Unicode encoding, so that all languages in the world can be supported and displayed correctly. However, since many existing programs are encoded in a specific language, such as GBK, it is therefore impossible to stop the support of the existing encoding and force everything to Unicode. That's why Windows uses code pages to adapt to various countries and regions, the current code page is by default 936 (GBK) so the image file is encoded by this way, while the pc in the lab room uses 850 (Latin-1). To adjust automatically the program ignoring the difference of computers, Local8Bit are used to adapt the encoding method to the host's code page.

This change is only necessary for windows users, because on Linux file names have no concept of character set. They are just a bunch of bytes which then get interpreted as strings based on the local 8-bit encoding.

Empty image error

Reported by issue #90, the program occasionally crashes when selecting an area in the image window. The problem could not be reproduced at the beginning, but after many attempts it was successfully discovered.

In normal cases, selecting an area will create an invisible rectangular object, the rectangular is defined by the point of departure and the corresponding diagonal according to the distance moved by the mouse. Smoothly dragging the mouse out of the image window leads to copy the selected area into a new image, as defined in GenericOperation.

The problem occurs when we double click an image with left button, the area selecting operation might be triggered by mistake and since the mouse has not moved, the size of rectangular is set to 0x0. Then if the user drags around the mouse and releases it somewhere outside the image window, unfortunately, a void image will be copied to the navigation bar and lead the crash.

To solve this, a verifying step is added to mouseDoubleClickEvent. Before dropping a copy of image into the navigation bar, it checks the size of the target rectangle, nothing further will happen with an empty rectangle.

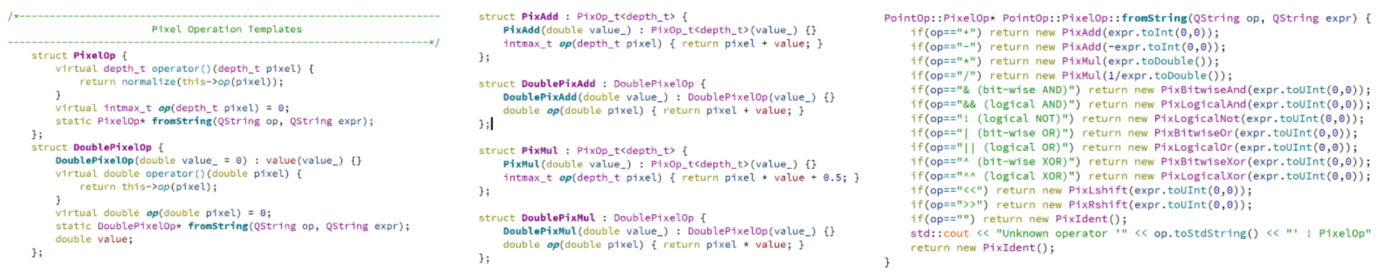
Empty image leads to crash in a general way, not only happens in the case mentioned below but also after the generation of empty images. It is necessary to block the generation of empty images provided by this program itself, there are three operations under the Tools menu which lead to image generation: Generate random image, Generate RGB image and Sinus synthesis.

A verification step is added to test the input value of width and height, neither of them can be 0 and the proposed value by default was changed from 0 to 512.

Logical operation error

A few logical operations between images do not work properly, the first image is returned instead of the desired result.

After verified the program output in debug mode, "Unknown operator'&'!" was threw as an error message. Class PointOp is found as the one which contains all the algorithms of pixel operations. It inherits GenericOperation class, the basic class for all kinds of operation.
The following example shows how a standard pixel operation is defined as template and processed according to the selected option in the drop-down bar.



```
/*-----------------------------------------------------------------------
                        Pixel Operation Templates
-----------------------------------------------------------------------*/
    struct PixelOp {
        virtual depth_t operator()(depth_t pixel) {
            return normalize(this->op(pixel));
        }
        virtual intmax_t op(depth_t pixel) = 0;
        static PixelOp* fromString(QString op, QString expr);
    };
    struct DoublePixelOp {
        DoublePixelOp(double value_ = 0) : value(value_) {}
        virtual double operator()(double pixel) {
            return this->op(pixel);
        }
        virtual double op(double pixel) = 0;
        static DoublePixelOp* fromString(QString op, QString expr);
        double value;
    };
```

```
struct PixAdd : PixOp_t<depth_t> {
    PixAdd(double value_) : PixOp_t<depth_t>(value_) {}
    intmax_t op(depth_t pixel) { return pixel + value; }
};

struct DoublePixAdd : DoublePixelOp {
    DoublePixAdd(double value_) : DoublePixelOp(value_) {}
    double op(double pixel) { return pixel + value; }
};

struct PixMul : PixOp_t<depth_t> {
    PixMul(double value_) : PixOp_t<depth_t>(value_) {}
    intmax_t op(depth_t pixel) { return pixel * value + 0.5; }
};

struct DoublePixMul : DoublePixelOp {
    DoublePixMul(double value_) : DoublePixelOp(value_) {}
    double op(double pixel) { return pixel * value; }
};
```

```
PointOp::PixelOp* PointOp::PixelOp::fromString(QString op, QString expr) {
    if(op=="+") return new PixAdd(expr.toInt(0,0));
    if(op=="-") return new PixAdd(-expr.toInt(0,0));
    if(op=="*") return new PixMul(expr.toDouble());
    if(op=="/") return new PixMul(1/expr.toDouble());
    if(op=="& (bit-wise AND)") return new PixBitwiseAnd(expr.toUInt(0,0));
    if(op=="&& (logical AND)") return new PixLogicalAnd(expr.toUInt(0,0));
    if(op=="! (logical NOT)") return new PixLogicalNot(expr.toUInt(0,0));
    if(op=="| (bit-wise OR)") return new PixBitwiseOr(expr.toUInt(0,0));
    if(op=="|| (logical OR)") return new PixLogicalOr(expr.toUInt(0,0));
    if(op=="^ (bit-wise XOR)") return new PixBitwiseXor(expr.toUInt(0,0));
    if(op=="^^ (logical XOR)") return new PixLogicalXor(expr.toUInt(0,0));
    if(op=="<<") return new PixLshift(expr.toUInt(0,0));
    if(op==">>") return new PixRshift(expr.toUInt(0,0));
    if(op=="") return new PixIdent();
    std::cout << "Unknown operator '" << op.toStdString() << "' ! PixelOp"
    return new PixIdent();
}
```

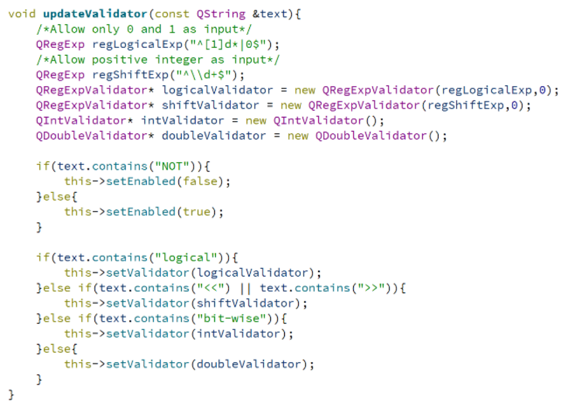
Figure 13 : An example shows how pixel operations work

By analyzing the current implementation, a logical error was found when processing the selected options in the dialog box. The result image can be output in a uchar or double format according to user's choice and this output format option is used as an important basis for the judgement of branch switching. In the existing logic, if the second operand is an image, the program will enter the wrong branch and consider the operand image as a double image. But currently there is no '&' logic implemented for double images, when the program cannot find a suitable processing function, it will by default send back the original picture to user.
After fixing this error, the user interface is still not clear enough. If only symbols are used, it may cause confusion between bit operations and logic operations. A better way is to cancel the unified drop-down menu bar and make the content update automatically according to the selected options and the type of the second operand.

First, it is necessary to clarify the conditions under which the optional items appear. Arithmetic operators will always be available regardless of the input and output types.

Bitwise operators will be deactivated if one of the 2 operands is double, this type of operation takes 2 standard images or 1 standard image and a positive integer as operands, the output will be always in uchar format (unless the double option is selected, pixel values will be converted to double by adding .00 at the end). Logical operators take all kind of operands and it return the result processed by standard C logic.

In order to achieve this, a signal will emit when the item selected in the drop-down menu changes. The bound slot will then check the existing options to select the corresponding validator. A new class inherits QLineEdit is created, it verifies whether the string contains keyword such as bitwise, logical or shifting symbols to enable the corresponding validator for the common input widget. When a Not operation is detected, the input field will be closed, because the inversion of images takes the complement value of 255, it does not require a second operand.



```
void updateValidator(const QString &text){
    /*Allow only 0 and 1 as input*/
    QRegExp regLogicalExp("^[1]d*|0$");
    /*Allow positive integer as input*/
    QRegExp regShiftExp("^\\d+$");
    QRegExpValidator* logicalValidator = new QRegExpValidator(regLogicalExp,0);
    QRegExpValidator* shiftValidator = new QRegExpValidator(regShiftExp,0);
    QIntValidator* intValidator = new QIntValidator();
    QDoubleValidator* doubleValidator = new QDoubleValidator();

    if(text.contains("NOT")){
        this->setEnabled(false);
    }else{
        this->setEnabled(true);
    }

    if(text.contains("logical")){
        this->setValidator(logicalValidator);
    }else if(text.contains("<<") || text.contains(">>")){
        this->setValidator(shiftValidator);
    }else if(text.contains("bit-wise")){
        this->setValidator(intValidator);
    }else{
        this->setValidator(doubleValidator);
    }
}
```

Figure 14 : Change strategy based on the selected content

The newly added logical operations are directly defined on the default logic of C, they do not seem to be very meaningful on image manipulation.

Image save error

When trying to save the image of the current window, the selected suffix does not take effect, it remains always the same as the original file regardless of the option selected.

This suffix adding process is then reimplemented, now it determines the position of the last “.” in the original image if there exist more than one. It will intercept the pure file name through a series of string operations and then add the desired suffix to the back.

The proposed image name was also changed to retain the operations that have been performed before, the desired suffix will be added to the end of string.

Log scale error

Log scale is a tool to enlarge the display values on a logarithmic scale to facilitate the observation when the original values are close to zero and no important information can be observed. It seems not to work on the imaginary part of the calculated FFT operation.

$$value = \frac{255 * log(mag * logConstant * \_logConstantScale + 1)}{log(255 * logConstant * \_logConstantScale + 1)}$$

logConstant = $e^{-log2(mean)}/8$ _logConstantScale = $8^{0.5*logScale-3}$ logScale = input value of slider mag = original pixel value

Figure 15 : Log scale formula

After analyzing and refining the existing code, the log scale function is found to perform the above formula. By testing several cases which the function is not working, the problem was found to be related to the mean value. The scaling function will not work if the mean value of an image is too close to zero. In such situation, no matter how we manipulate the slider, a multiplier close to zero can absolutely take off the effect of the it.

To solve this problem, mean of absolute values is used instead of the native mean value of an image. Since its value is not near zero, the function of slider back to normal as expected.

Optimize the display of double images

As mentioned earlier, the program can process images in any format with the help of its template definition which is expansible enough. But the current mechanism does not support any display of image with decimal or negative values.

In order to retain the complete information of the original image, a processed copy in uchar format that can be displayed normally will be generated by a function call makeDisplayable to meet this requirement and the real result image is hiding in the DoubleImageWindow, it never shows itself directly to users.

In general, operations that may output a double image will always propose some available options such as truncation, offset and scaling for users, these options are then used as parameters of makeDisplayable. This treatment is done quietly after the main operation and users will only get a displayable image in uchar format.

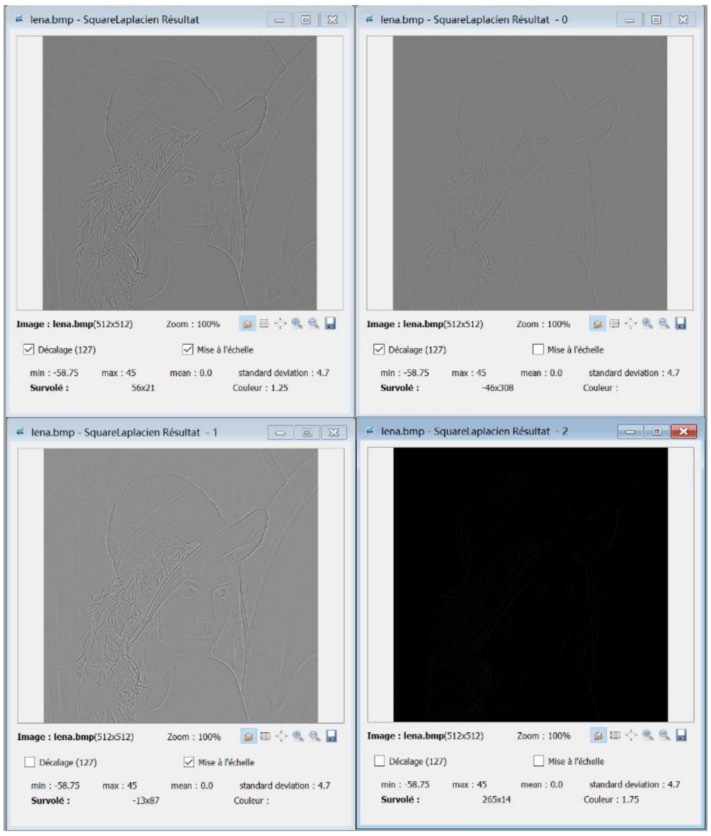
In order to make users have the right to regret, to be able to see images that are closer to the original results, to get intuitively an impression of the differences between truncation, offset and scaling, two checkboxes are added to help restore the unprocessed (truncation and double to uchar conversion will still be applied if necessary) result image, which leaves the choices to users.

That means after the calculation of any operations, if the result image contains any negative values, there will be two options available and already on checked state in the image window, representing the offset & scaling which if performed by default.

The user can then uncheck these two options to recalculate the display image based on the original image, no new windows will pop up like the previous version, all changes are done in the target window. There are actually three different situations, the first one is standalone without unnecessary processing. The second case adds an offset of 127 to all pixels. Third, linearly map the value beyond the limit to the target interval and then add an offset of 127, so that the minimum value becomes zero, 0 becomes 127, and the maximum value becomes 255. At the same time, the formula used in the transformation will display in the information window.

After introducing these changes, we still need to modify the makeDisplayable function since it involves other functions such as log scale. If we ignore this step, they will be executed in a parallel way, and each change will start from the original image, in this case the position of the slider will become meaningless.

So, I integrate all the related functions into makeDisplayable and reorganize the order of execution. Now the checkbox will be executed after the log scale function and based on its result, so there is no more disharmony.



lena.bmp - SquareLaplacien Résultat

Image : lena.bmp(512x512) Zoom : 100%

[✓ Décalage (127)] [✓ Mise à l'échelle]

min : -58.75 max : 45 mean : 0.0 standard deviation : 4.7

Survolé : 56x21 Couleur : 1.25

lena.bmp - SquareLaplacien Résultat - 0

Image : lena.bmp(512x512) Zoom : 100%

[✓ Décalage (127)] Mise à l'échelle

min : -58.75 max : 45 mean : 0.0 standard deviation : 4.7

Survolé : -46x308 Couleur :

lena.bmp - SquareLaplacien Résultat - 1

Image : lena.bmp(512x512) Zoom : 100%

Décalage (127) [✓ Mise à l'échelle]

min : -58.75 max : 45 mean : 0.0 standard deviation : 4.7

Survolé : -13x87 Couleur :

lena.bmp - SquareLaplacien Résultat - 2

Image : lena.bmp(512x512) Zoom : 100%

Décalage (127) Mise à l'échelle

min : -58.75 max : 45 mean : 0.0 standard deviation : 4.7

Survolé : 265x14 Couleur : 1.75

Figure 16 : Comparison of different treatments of an undisplayable double image

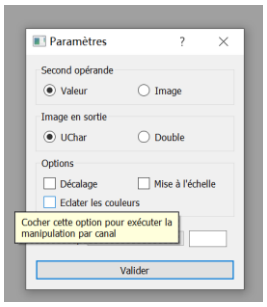
UI issues

A large number of issues registered on GitHub is related to the user interface. In some cases, the output information generated by operation does not match the actual function, a lot of notions in the current langue version are inconsistent with other versions (only English and French are available), and there is no helping prompt to help users understand the available options and features of an complex operation.

Each situation has its own approach to deal with, in the first case, we just need to modify the output information to make it coherent with the reality. To solve the translation problems, we need to use an internal tool called Qt's linguist, which is very efficient and practical and greatly simplifies the workload.

It only requests to add the tr() conversion to the user-visible strings. Once it is enabled in CMakeList, it may generate automatically the correspond ts files (a readable translation file using XML format) according to the changes detected. After manually modified each translation text in the target language, it can generate a qm file (a binary file) which can be loaded by the program to switch language.

To make users understand the meaning of each option, there is a very practical function called setWhatsThis that connect various components of an user interface to a help message, which is activated by the question mark in the upper right corner of the window. Some other types of tips are also added in different ways to help users understand the current operation.



Paramètres

Second opérande: [✓ Valeur] Image

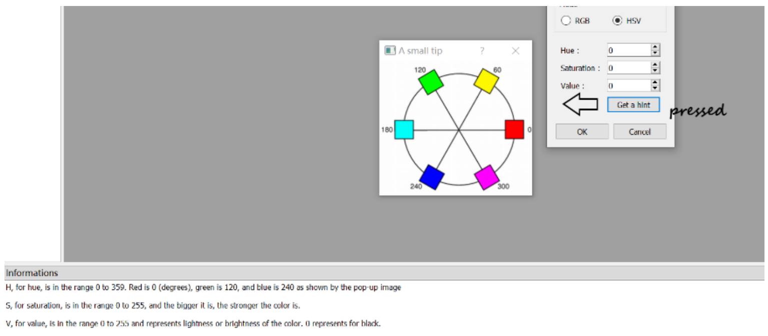
Image en sortie: [✓ UChar] Double

Options: ☐ Décalage ☐ Mise à l'échelle ☐ Eclater les couleurs

Cocher cette option pour exécuter la manipulation par canal

Valider

Figure 17 : Tips managed by setWhatsThis function



RGB [✓ HSV]

Hue : 0
Saturation : 0
Value : 0

Get a hint
OK
Cancel

Informations

H, for hue, is in the range 0 to 359. Red is 0 (degrees), green is 120, and blue is 240 as shown by the pop-up image

S, for saturation, is in the range 0 to 255, and the bigger it is, the stronger the color is.

V, for value, is in the range 0 to 255 and represents lightness or brightness of the color. 0 represents for black.

Figure 18 : A tip given by a pop-up window

Deploy Qt application on Linux

The existing configurations can successfully deploy the executable file for the windows system with the help of windeployqt, an official deployment tool and the executable can be directly used on other windows machines.

But in Linux machines, it cannot be constructed in the same way, the executable file generated by Qt on Linux needs to be accompanied by dynamic link libraries and is not able to run on other machines without Qt environment or with Qt but the dependent libraries' versions are different.

The root cause of this problem is that the executable file is dynamically compiled, and when executed it needs to call the commands in the corresponding dynamic link library. Although the size of the executable file is well reduced, it is an unsound application without portability.

To solve this problem, all the associated link libraries need to be attached to the executable. This requires employing a shell script called ldd to display the dependencies of the executable, and then we need to find (I used apt-file search to do this job) the locations of libraries that the program depends on and copy them to the release folder. There may also be some problems in this process, sometimes the ldd results point to some symbolic links ending with .so.5 instead of the correct shared object, but the ones need to be copied are always the source files not the link files. When trying to launch the program on other pc, I received an error message: "The Qt platform plugin "xcb" could not be loaded... This application could not be started because the Qt platform plugin could not be initialized." To solve this problem, two libraries libxcb-xinerama0 and build-essential are required on the target Linux system, and there will be no problems after installation.

In addition, there are two other methods that can also be used to deploy the program on Linux, a deployment tool called linuxdeployqt that automates the procedures is recommended by Qt. Unfortunately, due to the lack of support for the newest Ubuntu system, this tool failed to run on my pc.

The third method is to statically compile the program under Linux, this approach requires a statically compiled and installed Qt platform. After downloading the source files of Qt, the -static option needs to be enabled when compiling. In this case, environment variables need to be set manually to enable the static version of Qt.

Unfortunately, because I did not uninstall the ordinary version, there were many conflicts that I could not even open the Qt Creator, I had to give up this idea.

Unit Testing

Unit testing is used to test the correctness of a module, a function or a class. The goal is to isolate each part of the program and show that the individual parts are correct.

In previous versions of detiq-t project, there were class tests for the ImageIn library, a macro-level test for the GenericInterface and several basic applications were also covered. Unfortunately, due to excessive changes, their executions were all failed.

There is another test written for the generic user interface. Starting from the main window and add progressively available modules to test its functionalities. The processing results need to be compared with MATLAB in the verification step. Unfortunately, this test was not able to pass the compilation stage either.

Under this situation, a test framework that is easy to modify when the version changes are strongly needed. Since it is supposed to serve as a basic module for subsequent development, it needs to be robust enough to and its availability must be ensured.

The project contains many classes, and these classes depend on each other. These relationships may be intricate and cannot be separated. Making individual test outside the project is not feasible, if a class is tested separately, due to dependencies, the test code requires a large number of related mock objects to achieve a successful compilation.

So, the test code needs to be integrated into product code to reduce the cost and difficulty of testing. If tests and the entire project are compiled together, almost all resources of the product code become available for the tests and so solved the dependency problem.

There are two ways to achieve this structure, one is to write the code directly in the original class to get more convenience and efficiency, it shares a main entrance with existing program, and the test content is easy to update when the program code changes. I chose this path when first tried and two external libraries were used to simplify the workload.

The first one called Catch2, it is a header only C++ test framework which requires no external dependencies. Tests can be tagged and divided into sections, each one could run in isolation way and the output is done through the modular reporter object, detailed reports are given in xml format. I found it very user-friendly and easy to get started. As a supplement, I chose FakeIt as another external library, it is a header only library for C++ mocking framework. Although it is not able to mock classes with multiple inheritance, it remains a very advantageous tool and the dynamic casting is well supported. The algorithm part was written smoothly at the beginning, but I encountered a serious problem when entering the service part. Most of the service classes inherit QObject because they need to make a response to signals, however, the two external libraries do not support Qt's features well, and irreconcilable contradictions continue to emerge. In the end, considering that the introduction of external libraries is not conducive to subsequent development, I decided to abandon this approach and write the test class separately.

The test code is now completely independent of the program code and is completely controlled by CMake. In this case the Qt Test framework is used, which has a huge advantage in the regard of testing Qt based libraries.

The reason why Qt Test was not used at the beginning related to its poor support for fixtures and mocks. But on the other hand, it does a good job in GUI event simulation, with the help of QSignalSpy which may connect to any signal of any object and record its emission information, it becomes very useful in most of the scenarios. After using QSignalSpy to capture the signal, we may directly call the slot for testing.

Qt Test also provides a series of simulation interfaces to facilitate interface testing, by simulating mouse and keyboard activities, it becomes possible to verify directly the availability of contents of the practical courses if we use QTestEventList to encapsulate a series of mouse and keyboard operations which meet the requirements of the practical works.

During the implementation process, many problems were encountered, and the header file problem was the most influential one, a lot of time was spent to troubleshoot and solve this. Without changing the program code and adding only test classes, the test entrance is managed by CMake which remains independent from the main program entrance. When compiling the test, the multiple inclusion relationship of the header files it contains will become chaotic due to the including style of header files, some headers use absolute references while the others using relative path.

When the entry is the main program, such a problem will not occur because the header files and source files are packaged in the order of the modules under the rules of CMake, so any search for references can point to the correct location. But the test classes do not have this preferential treatment, what they give were actually the unclear errors. Once the cause of the error is known, the solution will be found quickly, just need to compile the source code into libraries, and then integrate them to the test classes.

Conclusion

In conclusion, I have to say that what I learned from this project far exceeded my expectations. Its meaning is not only reflected in the programming level but has profoundly affected me at multiple aspects and made me greatly improved.

Thinking from the perspective of a software developer, instead of being oriented toward completing a certain task like what used to do in practical classes. I start to think which features are actually beneficial to users and whether certain options are misleading, considering the overall product.

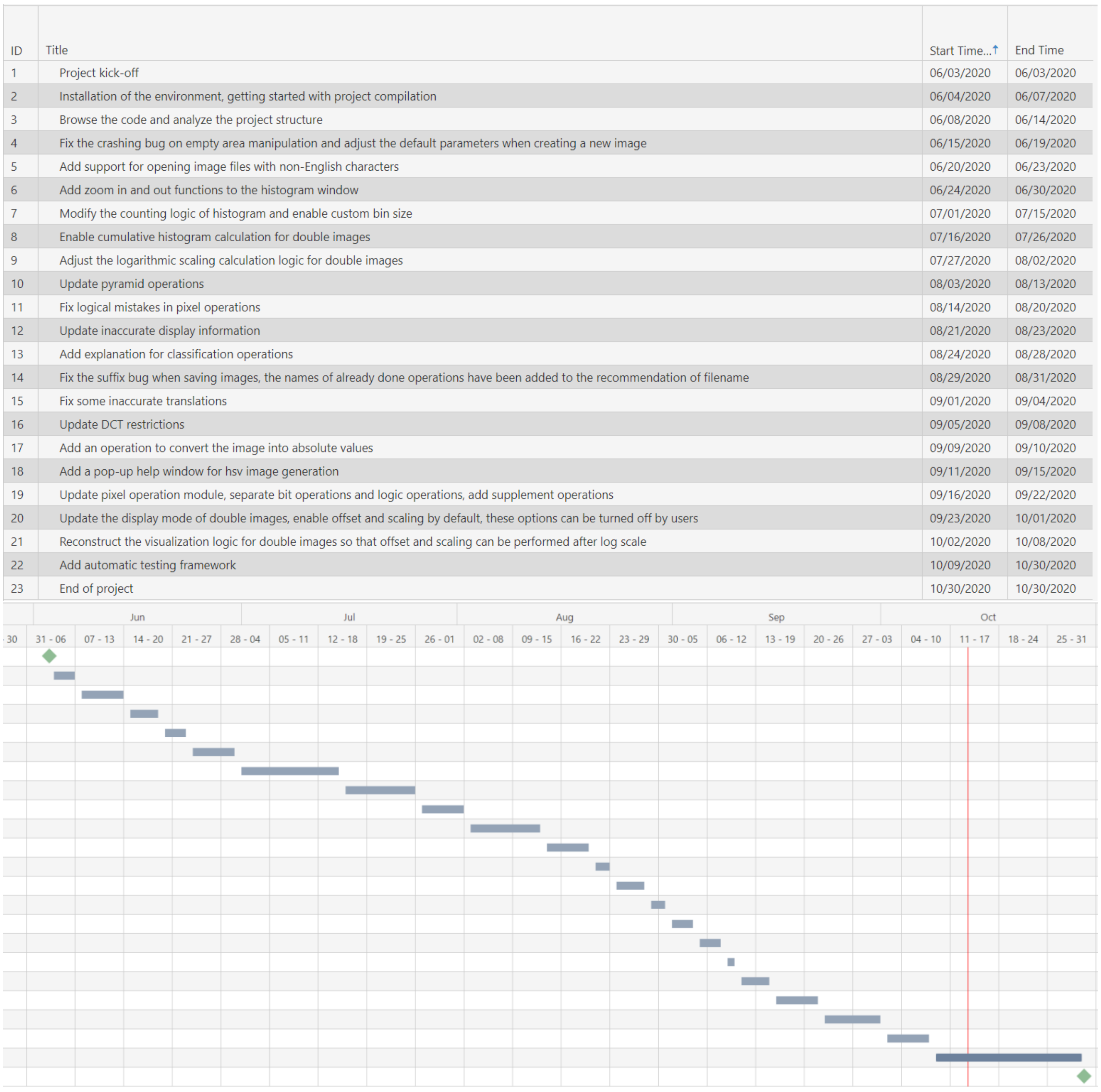
From a debugging perspective, I realized that it is better to jump to the source of the problematic component for in-depth analysis, because in a large project many components and functions are interrelated, errors may occur in irrelevant places. How to view the relationship between components from a global perspective and how to manage their relationship has brought me a lot of thinking.

From a point of view of EII student, the software is developed for the purpose of education, its functions are exactly what we have encountered at school. Through the inspection of algorithms, I have reviewed the previous courses and deepened my understanding of image processing methods.

Considering the programming skills, I become a proficient user of Qt through this project and be able to apply its various features to complete the tasks, especially the communication between signals and slots. I have also gained a lot in C++ field, as a software which has been optimized for many times, it has many gorgeous and outstanding code writing style that benefited me a lot. Not to mention the CMake that I learnt for this project, the practical tool which might be very helpful for the future.

Back to ImageINSA itself, I am honored to be able to make some contributions to it, solve bugs, improve some of the previously defective functions and add new components to the software. I am very pleased that it has become better with my efforts.

Annex A -- Timeline



ID	Title	Start Time...↑	End Time
1	Project kick-off	06/03/2020	06/03/2020
2	Installation of the environment, getting started with project compilation	06/04/2020	06/07/2020
3	Browse the code and analyze the project structure	06/08/2020	06/14/2020
4	Fix the crashing bug on empty area manipulation and adjust the default parameters when creating a new image	06/15/2020	06/19/2020
5	Add support for opening image files with non-English characters	06/20/2020	06/23/2020
6	Add zoom in and out functions to the histogram window	06/24/2020	06/30/2020
7	Modify the counting logic of histogram and enable custom bin size	07/01/2020	07/15/2020
8	Enable cumulative histogram calculation for double images	07/16/2020	07/26/2020
9	Adjust the logarithmic scaling calculation logic for double images	07/27/2020	08/02/2020
10	Update pyramid operations	08/03/2020	08/13/2020
11	Fix logical mistakes in pixel operations	08/14/2020	08/20/2020
12	Update inaccurate display information	08/21/2020	08/23/2020
13	Add explanation for classification operations	08/24/2020	08/28/2020
14	Fix the suffix bug when saving images, the names of already done operations have been added to the recommendation of filename	08/29/2020	08/31/2020
15	Fix some inaccurate translations	09/01/2020	09/04/2020
16	Update DCT restrictions	09/05/2020	09/08/2020
17	Add an operation to convert the image into absolute values	09/09/2020	09/10/2020
18	Add a pop-up help window for hsv image generation	09/11/2020	09/15/2020
19	Update pixel operation module, separate bit operations and logic operations, add supplement operations	09/16/2020	09/22/2020
20	Update the display mode of double images, enable offset and scaling by default, these options can be turned off by users	09/23/2020	10/01/2020
21	Reconstruct the visualization logic for double images so that offset and scaling can be performed after log scale	10/02/2020	10/08/2020
22	Add automatic testing framework	10/09/2020	10/30/2020
23	End of project	10/30/2020	10/30/2020

Figure 17 : Timeline of the project

Annex B -- Suggestions for building the project

----- The 5 essential libraries (RandomLib, Qwt. Zlib, Jpeg and Png) needs to be installed before compilation of the project-----

For the installation of RandomLib, it is better to use cmake rather than make, otherwise it must be manually added into the PATH:

```
cd RandomLib-1.10
mkdir BUILD
cd BUILD
cmake ..
make
sudo make install
```

For the installation of Qwt:

```
cd qwt-6.1.5
qmake qwt.pro
make   /* In this step, if the OPENGL library is missing (fatal error: GL / gl.h: No such
file   or directory), the installation of build-essential libgl1-mesa-dev will solve this problem */
sudo make install
```

Important: After completing the above steps, you still need to manually copy some files from Qwt directory to your Qt directory (required under both Linux and Windows systems):

1. Copy qwt-6.1.5/designer/plugins/designer/libqwt_designer_plugin.so (or .dll) to Qt/5.15.1/gcc_64 /plugins/designer

2. Copy all files in qwt-6.1.5/lib to Qt/5.15.1/gcc_64/lib

3. Create a new folder in Qt/5.15.1/gcc_64 /include and name it qwt, copy all the header files under qwt-6.1.5/src to the new created qwt folder.

For the last three, if under Linux system, the easiest way is to sudo apt install zlib1g, libjpeg-dev and libpng-dev. There is no shortcut for Windows users, you need to install the ordinary version of the three libraries.

Annex C -- Manual of Test

The test framework has been repaired and integrated into the main program through CMake, it will compile together with the application and generate an executable file called detiq-t_Test in your BUILD directory.

In order to run the test, you have to manually copy the res folder (where this file is located) to the same location of the executable. (Note: better to enable the "Run in terminal" option (found at Projects--Build & Run--Run Settings--Run) if you decide to run the test through Qt Creator.

The test is mainly written for the ImageIn library, to test several low-level actions provided by this library, such as reading/writing operations towards different image formats, copy & crop operations, format conversion and serval algorithm tests including the MorphoMats.

If you want to add other tests for further development, please refer to the Test.h and Tester.h files, they define a basic test framework to be inherited, including the common functions of a test such as constructor, init, test cleanup and info (to point out where the error occurred).

You can add the content and method you want to test in your own test file and start the test by creating a tester file that can execute multiple tests (Testers need to be added to the main function for the purpose of unification).

Annex D -- List of Figures

Annex E -- Glossary

Qt : a free and open-source widget toolkit for creating graphical user interfaces as well as cross-platform applications that run on various software and hardware platforms such as Linux, Windows, macOS, Android or embedded systems.

CMake : a cross-platform free and open-source software tool for managing the build process of software using a compiler-independent method. CMake is not a build system but rather a build-system generator.

Code page : a character encoding and as such it is a specific association of a set of printable characters and control characters with unique numbers.

HSV (hue, saturation, value) : an alternative representation of the RGB color model, designed to align more closely with the way human vision perceives color-making attributes.

Histogram : an approximate representation of the distribution of numerical data.

Unit testing : a software testing method by which individual units of source code are tested to determine whether they are fit for use.

Annex F -- Work to be done in the future

Morphomath :

Improve the display o the structure element, visually highlight the center (position: pixel index = size / 2 with entire division) and allow users to move it interactively.

Since the image of the view is defined as a grayscale image, to present the center by a red square, I need to refactor this infrastructure or rewrite several methods of QGraphicsScene to achieve a three-layer display. I tried to improve this but failed because of the time limit.

Double image saving :

At present, this function has not been implemented, just as the display of a double image is actually its processed version by function makeDisplayable, the image saved is actually a record of its Uchar copy.

Quantification :

It is not yet able to quantify double images.

(Note: If you make changes to the quantification module, test also the feasibility in the DPCM module and vice versa)

Interactive line/column profile :

In the Line / Column Profile display, allow the user to dynamically select the line to observe.

Operation of double images :

Ideally, it is better to make all existing operations possible for double images.

Tests :

I fixed some old tests that failed to run and corrected some algorithmic errors, they are the companion products of the detiq-t project. These tests are integrated into the main project through CMake, they are basically towards the ImageIn library, to complete other types of tests, please check the README file in the Test folder.

Abstract

Keywords : *Image processing, C ++, Qt, Software development, Unit testing*

This end-of-study project was carried out in EII department of INSA Rennes (Done at home actually, according to government's epidemic prevention measures) from 3rd June 2020 to 30th October 2020. The motif of this project aims to the maintenance and improvement of ImageINSA, an image processing software developed at EII department in INSA for educational purposes. During this project, many bugs were fixed, some functional components are optimized and new features were added including an automated testing framework.

Résumé

Mots clés : *Traitement de l'image, C++, Qt, Développement logiciel, Test unitaire*

Ce projet de fin d'étude a été réalisé dans le département EII de l'INSA de Rennes (Réalisé chez soi en raison des mesures de prévention des épidémies du gouvernement) du 3 juin 2020 au 30 octobre 2020. Le but du projet vise le maintien et l'amélioration de l'ImageINSA, un logiciel de traitement d'image développé au département EII de l'INSA de Rennes pour des fins pédagogiques. Au cours de ce projet, des nombreux bugs ont été corrigés, certains composants fonctionnels sont optimisés et nouvelles fonctionnalités ont été ajoutées ainsi qu'un cadre de test automatisé.