A Morphing Application - C++ Project

Report

by Flavien DAVID

Pr. Yohan FOUGEROLLE

0.1 General overview

The purpose of this project has been to deal with Object-Oriented Programming (OOP) by using Qt. After having the green ligth on the project, I have to study how work the morphing process, and then try to implement the code. We will be evaluate on the project management, the code and the last one will be during the presentation.

The management of the project has been tough between matching information and implement patch of code until I make a mind map or moreover a euristic schematic and try to have the best overwatch on the project.

If the project is well understood it's easier to code when we already know the language and how to perfectly code, otherwise the easy way is to change the design of the code and try again until it works.

0.2 The Software - Qt

The software Qt allows us to program under the C++ language but not only. It is also a cross-platform software development for embedded, desktop or mobile devices. It's a multi-lingual platform available on Windows, MacOSX, and Ubuntu.

0.3 We must not

Implement tricky and unnecessarily complex algorithms. If nothing works because there is a tricky bug in a non linear min-square minimization, this will be of your responsibility because no one ever asked you to tackle such problems (and you will see them next year anyway). Forget advanced mathematics in this project because this is simply not the objective. If you have a doubt, just ask for clarification. Be a console application only. Graphical user interface is mandatory. Be developed using other IDEs than Qt and be a mix of various languages (avoid QML). You can tweak Qt CSS. Use Matlab engine, mex, or anything related to other programming languages: stick to C++

Contents

0.4 The code

0.4.1 Header File

- mainwindow.

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>

#include <iostream>
#include <QFileDialog>
#include <QLabel>
#include <QPainter>
#include <QPixmap>
#include <QSlider>
#include <algorithm>
#include <numeric>
#include <QDebug>

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    int meanA_x = 0;
    int meanA_y = 0;
    int meanB_x = 0;
    int meanB_y = 0;

private slots:
    void on_loadImageButton_clicked();
    void on_loadImageButton2_clicked();
    void on_saveImageButton_clicked();
    void on_horizontalSlider_valueChanged(int value);
    void Slot_Mouse_Pressed();
    void Slot_Mouse_Current_Position();
    void Slot_Mouse_Current_Position2();

private:
    Ui::MainWindow *ui;
```

```
};

#endif // MAINWINDOW_H
```

- my_tool.h

```
#ifndef MY_TOOL_H
#define MY_TOOL_H

#include <QLabel>
#include <QMouseEvent>
#include <QEvent>
#include <QDebug>
#include <QPaintEvent>
#include <QPainter>
#include <QPen>
#include <QPoint>
#include <QVector>

class my_tool : public QLabel
{
    Q_OBJECT
public:
    explicit my_tool(QWidget *parent = 0);
    ~my_tool();

    void mouseMoveEvent(QMouseEvent *e);
    void mousePressEvent(QMouseEvent *e);
    //void paintEvent(QPaintEvent *e);

    int x,y;
    QVector <int> Coord_X;
    QVector <int> Coord_Y;

signals:
    void Sig_Mouse_Pressed();
    void Sig_pos();
    void Sig_Mouse_Left();
};

#endif // MY_TOOL_H
```

- morph.h

```
/*
#include "morph.h"
```

```
#include <QImage>

//Constructor
Morph::Morph()
{

}

//Destructor
Morph::~Morph()
{

}

Morph::matchPixel()
{

}

Morph::barycentre()
{

}

Morph::opacity()
*/
```

0.4.2 Source File

- main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

It's always better to have a program which does nothing, or not so much.

- morph.cpp

```
/*
#include "morph.h"
#include <QImage>

//Constructor
Morph::Morph()
{

}

//Destructor
Morph::~Morph()
{

}

Morph::matchPixel()
{

}

Morph::barycentre()
{

}

Morph::opacity()
*/
```

I choose to show the way I saw the design at the first place with a class morph where this one can interact with the other.

At the end I just use two classes mainwindow, and my_tool and interact with these two in different manner like Global variable, inheritance, and Signal and Slot.

- my_tool.cpp

```
#include "my_tool.h"

using namespace std;

my_tool::my_tool(QWidget *parent) :
    QLabel(parent)
{

}

my_tool::~my_tool()
{
```

```
}

void my_tool::mouseMoveEvent(QMouseEvent *e)
{
    this->x = e->x();
    this->y = e->y();
    emit Sig_pos();
}

void my_tool::mousePressEvent(QMouseEvent *e)
{
    Coord_X.append(x);
    Coord_Y.append(y);
    qDebug() << Coord_X;
    qDebug() << Coord_Y;
    emit Sig_Mouse_Pressed();
    emit Sig_pos();
}

//void my_tool::paintEvent(QPaintEvent *e)
//{
//    for(int i = 1; i <= Coord_X.size(); i ++)
//    {
//        QLabel::paintEvent(e);
//        QPainter painter(this);
//        QPen drawpen(Qt::red);
//        drawpen.setWidth(7);
//        QPoint point;
//        point.setX(Coord_X.value(i));
//        point.setY(Coord_Y.value(i));
//        painter.setPen(drawpen);
//        painter.drawPoint(point);
//        emit Sig_Mouse_Pressed();
//    }
//}
```

- mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "my_tool.h"

using namespace std;

//Global flag
int flagCurrentPix = 0;
```

```
//Global vectors
QVector<int> PixA_x;
QVector<int> PixA_y;
QVector<int> PixB_x;
QVector<int> PixB_y;

//Global picture
QString Image_A = ":/img/img/jolie.jpg";
QString Image_B= ":/img/img/pitt.jpg";

//Constructor
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    //Default picture
    QPixmap pix1(":/img/img/jolie.jpg");
    ui->label_pic1->setPixmap(pix1);
    QPixmap pix2(":/img/img/pitt.jpg");
    ui->label_pic2->setPixmap(pix2);

    //Connections established Signal-Slot
    //From :\my_tool\Sig_Pos to :\mainwindow\Slot_Mouse_Current_Position
    connect(ui->label_pic1, SIGNAL(Sig_pos()),
            this,SLOT(Slot_Mouse_Current_Position()));
    connect(ui->label_pic2, SIGNAL(Sig_pos()),
            this,SLOT(Slot_Mouse_Current_Position2()));
    //From :\my_tools\. to :\mainwindow\.
    connect(ui->label_pic1, SIGNAL(Sig_Mouse_Pressed()),
            this,SLOT(Slot_Mouse_Pressed()));
    connect(ui->label_pic2, SIGNAL(Sig_Mouse_Pressed()),
            this,SLOT(Slot_Mouse_Pressed()));

    //Instruction in Status bar
    ui->statusBar->showMessage("Do your matching point at each time, e.g a point or

}

//Destructor
MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_loadImageButton_clicked()
```

```
{
    //Code to open the picture through a browser
    QFileDialog dialog (this);
    dialog.setNameFilter(tr("Images (*.png *.jpg *TIFF)"));
    dialog.setViewMode(QFileDialog::Detail);
    QString fileName = QFileDialog::getOpenFileName(this,
            tr("Open images"), "/Users", tr("Images File (*.png *.bmp *.jpg *TIFF)"

    //Select picture
    if (!fileName.isEmpty())
    {
        QImage image (fileName);
        ui->label_pic1->setPixmap(QPixmap::fromImage(image));
        ui->label_pic1->setScaledContents(false);
        ui->label_pic1->adjustSize();
    }
   //Extra feature to play with label
   //ui->label_pic1->setSizePolicy( QSizePolicy::Ignored, QSizePolicy::Ignored );
   //ui->label_pic1->hasScaledContents();
   //ui->label_pic1->keepScaledRatio
   //QImage::scaled(const QSize &size, Qt::AspectRatioMode aspectRatioMode = Qt::K
}

void MainWindow::on_loadImageButton2_clicked()
{
    //Code to open the picture through a browser
    QFileDialog dialog (this);
    dialog.setNameFilter(tr("Images (*.png *.jpg *TIFF)"));
    dialog.setViewMode(QFileDialog::Detail);
    QString fileName = QFileDialog::getOpenFileName(this,
            tr("Open images"), "/Users", tr("Images File (*.png *.bmp *.jpg *TIFF)"

    //Select picture
    if (!fileName.isEmpty())
    {
        QImage image (fileName);
        ui->label_pic2->setPixmap(QPixmap::fromImage(image));
        ui->label_pic2->setScaledContents(true);
        ui->label_pic2->adjustSize();
    }
    //Extra feature to play with label
    //ui->label_pic2->hasScaledContents();
    //ui->label_pic2->setSizePolicy( QSizePolicy::Ignored, QSizePolicy::Ignored );
}

void MainWindow::on_saveImageButton_clicked()
{
    QFileDialog dialog (this);
```

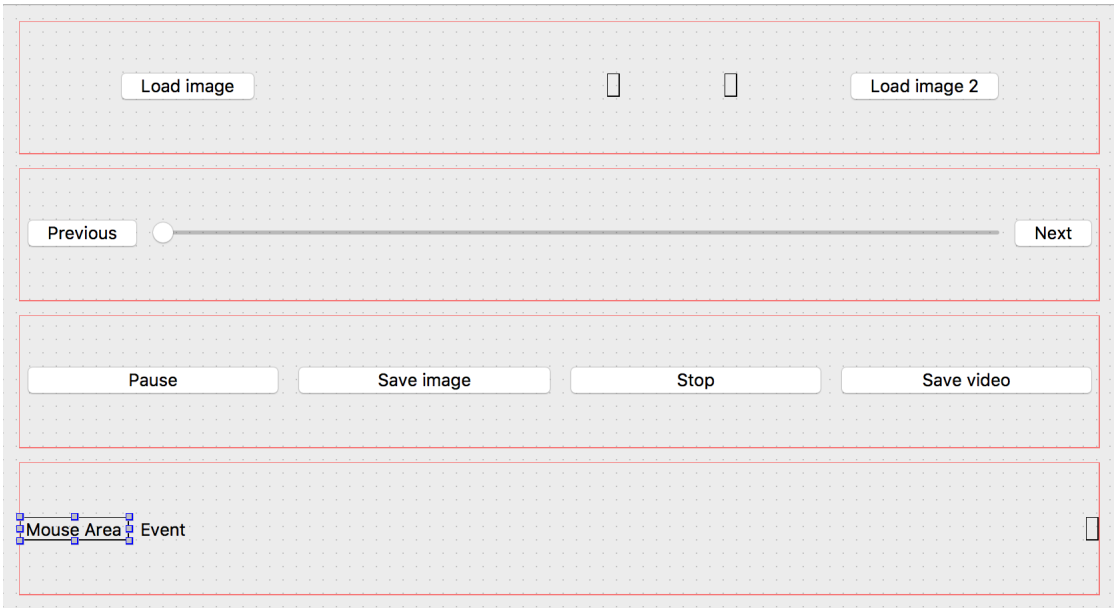
```
    QString fileName = QFileDialog::getSaveFileName(this,
                                                    tr("Save images"), "/Users/Flav
}

void MainWindow::Slot_Mouse_Pressed()
{

    if(flagCurrentPix == 1 && PixA_x.size() <= PixB_x.size())
    {
        ui->label_event->setText("Mousse Pressed 1!");
        PixA_x.append(ui->label_pic1->x);
        qDebug() << PixA_x;
        PixA_y.append(ui->label_pic1->y);
        qDebug() << PixA_y;
        const auto meanA_x = std::accumulate(PixA_x.begin(), PixA_x.end(), .0) / Pi
        qDebug() << "MeanA_x:" << meanA_x;
        const auto meanA_y = std::accumulate(PixA_y.begin(), PixA_y.end(), .0) / Pi
        qDebug() << "MeanA_y:" << meanA_y;

////attempt to draw a point at each click
//        int i =1;

//        QPainter painter(this);
//        QPen drawpen(Qt::red);
//        drawpen.setWidth(7);
//        QPoint point;
//        point.setX(PixA_x.value(i));
//        point.setY(PixA_y.value(i));
//        painter.setPen(drawpen);
//        painter.drawPoint(point);
    }
    else if(flagCurrentPix == 2 && PixB_x.size() <= PixA_x.size())
    {
        ui->label_event->setText("Mousse Pressed 2!");
        PixB_x.append(ui->label_pic2->x);
        qDebug() << PixB_x;
        PixB_y.append(ui->label_pic2->y);
        qDebug() << PixB_y;
        auto meanB_x = std::accumulate(PixB_x.begin(), PixB_x.end(), .0) / PixA_x.s
        qDebug() << "MeanB_x:" << meanB_x;
        auto meanB_y = std::accumulate(PixB_y.begin(), PixB_y.end(), .0) / PixB_y.s
        qDebug() << "MeanB_y:" << meanB_y;

////attempt to draw a point at each click
//        int i = 1;
//        paintEvent(pix2);
//        QPainter painter(this);
//        QPen drawpen(Qt::blue);
```

```
//          drawpen.setWidth(7);
//          QPoint point;
//          point.setX(PixB_x.value(i));
//          point.setY(PixB_y.value(i));
//          painter.setPen(drawpen);
//          painter.drawPoint(point);
    }

    flagCurrentPix=0;
}

void MainWindow::Slot_Mouse_Current_Position()
{
    flagCurrentPix = 1;
    ui->label->setText(QString("X = %1, Y = %2").arg(ui->label_pic1->x).arg(ui->lab
}

void MainWindow::Slot_Mouse_Current_Position2()
{
    flagCurrentPix = 2;
    ui->label->setText(QString("X = %1, Y = %2").arg(ui->label_pic2->x).arg(ui->lab
}
```

In this following part, I attend to make the morphing function by using the horizontalSlider, and as mention in the guideline I try dto not embedded too higher math for such application. Then I tried to find a morphing solution by using the average of the x_point, y_point, in one image and comparing it to the second image. The method could be extend and use the barycenter of the area of interest. It's also possible to consider other method such like triangulation but this method require more skills.

```
void MainWindow::on_horizontalSlider_valueChanged(int value)
{
    QImage bot(Image_A);
    QImage top(Image_B);

    //format conversion for alpha channel
    bot = bot.convertToFormat(QImage::Format_ARGB32);
    top = top.convertToFormat(QImage::Format_ARGB32);

    int w = bot.width()-((bot.width()-top.width())*value/99);
    int h = bot.height()-((bot.height()-top.height())*value/99);

    int shiftX = (meanB_x-meanA_x)/ui->label_pic1->width();
    int shiftY = (meanB_y-meanA_y)/ui->label_pic1->height();

    int shiftA_x = shiftX*top.width()*value/99;
    int shiftA_y = shiftY*top.height()*value/99;

    int shiftB_x = shiftX*bot.width()*(1-value/99);
```

```
    int shiftB_y = shiftY*bot.height()*(1-value/99);

    QPixmap combined(w, h);
    QPainter p(&combined);
    float opacity = value/99.0;
    p.setOpacity(1-opacity);
    p.drawImage(QPoint(shiftA_x, shiftA_y), bot);
    p.setOpacity(opacity);
    p.drawImage(QPoint(shiftB_x,shiftB_y), top);

    int wa = ui->label_picMixed->width();
    int ha = ui->label_picMixed->height();
    ui->label_picMixed->setPixmap(combined.scaled(wa,ha,Qt::KeepAspectRatio));

    p.end();
}
```



The User Interface is supposed to be user-friendly but actually it's one of the big step to understand how everything work out

0.5 Conclusion

I'am not really sure of what I learn during this module because I can't figure out to find a way to make things work properly or easily. Instead, in my opinion I do make a huge step by knowing how the things are related in Qt and how it's dense. Most of the time I struggle on the debug step where normally I shouldn't, except for typing error, if I know how to program and how the class are embedded in Qt.