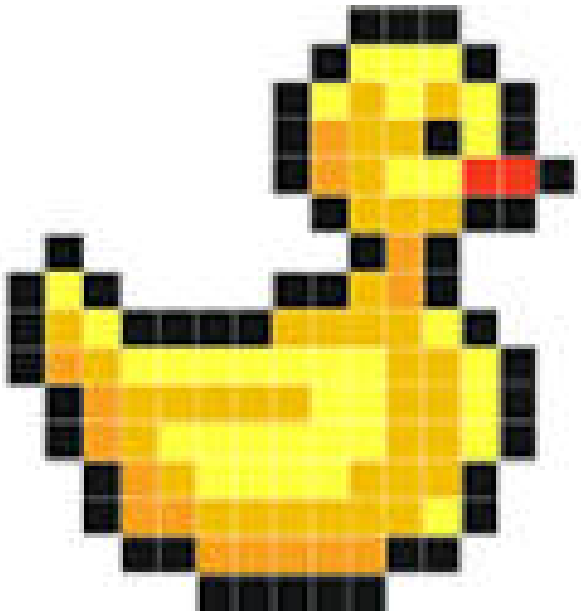
UE 2-1 Computer Science

Pixel Art Rendering Project



Submitted by **Selma Boudissa**

Contents

Acknowledgment

I will like to take time to thank each person who helped me to complete this project. I firstly give thanks to Yohan Fougerolle my supervisor for having patiente for me and explain things into details. Also I give special thanks to my classmates espacially Khoi Pham Nhat, Meldrick Ferdinand Reimmer and Antoine Merlet, who provides me lot of help during the entire project. I really appreciate this pixel art project which helped me to improve myself in term of C++ programmation under Qt.

Dear beloved machine ...

Introduction

In my final project for my course module Computer Science. I am to implement a C++ using QT a program which will load and display an image from any part of my hard-drive and get the average of that picture to display it as my pixelized image. Upon getting my image, my program should also be able to load Multiple images and replace my big and blurred images with the best match image from the multiple images in which I get.

In this report I will give explanation about each part of my project and the library I used. The Pixel Art Rendering contains three principles goals:

- **Goal 1:Load, Display and Save an image**
- **Goal 2:Pixelized an image**
- **Goal 3:Art transformation of the pixelized image**

In achieving these goals I took specific steps.Firstly creating an interface which contains buttons and labels to accomplish all the goals.

Chapter 1

Library

1.1 QFileDialog

The QFileDialog class provides a dialog that allow users to select files or directories. This class enables a user to traverse the file system in order to select one or many files or a directory.

1.2 QLabel

The QLabel widget provides a text or display an image. Is used for displaying text or an image. The visual appearance of the label can be configured in various ways.

1.3 QPixmap

The QPixmap class is an off-screen image representation that can be used as a paint device. A QPixmap can esasily be displayed on the screen using QLabel.

1.4 QImage

The QImage class provides a hardware-independent image representation that allows direct access to the pixel data, and can be used as a paint device.

1.5 QDir

The QDir provides acces to a directory structures and their contents. Use to manipulate path names, access information regarding paths and files...

Chapter 2

Interface

2.1 Definition of a Graphique User Interface

Allows developers to build user interfaces in a declarative way. Qt creator is one of the software who provide a GUI which combines Qt Widget with Qt Quick.

2.2 The GUI of my project

In the figure above the GUI of my pixel art project.

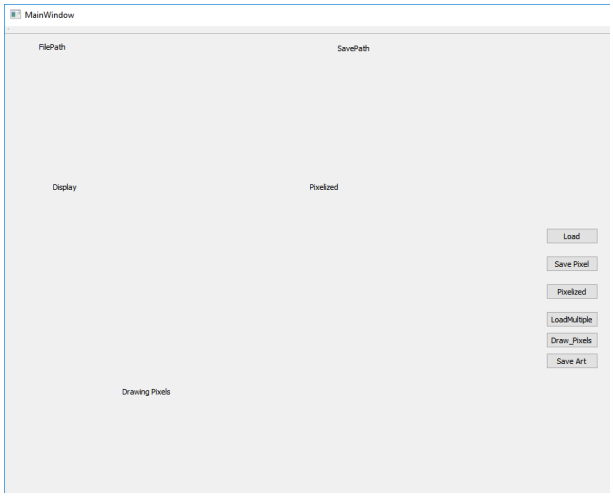


Figure 2.1: User Interface

Explanation: As you can see in the figure for my GUI I used 6 push- button.

- Load button to load the image
- Pixelized button to pixelized the loaded image
- Save Pixel button to save the pixelized image
- Load Multiple to load severals images at the same time
- Draw pixels button for the art trasnformation of the pixelized image
- Save Art Transfo button which the art transformation of the pixelized image.

Chapter 3

Load, Display and Save an image

3.1 Load and Display

For this part I have to show how to load an image located anywhere in my hard drive display it and be able to save an image. The things is that I have to write a programme that allow me to select the image I want in any format, at anyplace on my hard drive.

```
void MainWindow::on_Load_clicked()
{
    QString filePath = QFileDialog :: getOpenFileName(
                this,
                "Open a file",
                QString(),
                "Images (*.png *.jpig *.jpg)");//to open images in any formats.

    // I work with my teammate Meldrick Reimmer

    pimg = new QImage(filePath);

    ui->FilePathLabel->setText(filePath);

}
```

load_display.cpp

As you can see in the code above I use the QFileDialog which allow me to select the image that I want to display using a label on my graphical user interface (GUI). This label I named it "displayLabel". I also show in the FilePathLabel the location of the file I display. The pimg is a pointer to my image.

```
// get the label size
int w, h;
w=ui->displayLabel->width();
h=ui->displayLabel->height();

// with scale preserved
ui->displayLabel->setPixmap(QPixmap::fromImage(pimg->scaled(w,h, Qt ::
    KeepAspectRatio)));
```

image_scale.cpp

This part of the code is used to keep the scale of the image choose in order to display the all image and not a part. No matter the size of the label in the GUI.

3.2 Save button

```
void MainWindow::on_Save_clicked()
{
    QString savePath = QFileDialog :: getSaveFileName(
                this,
                "Save a file",
                QString(),
                "Images (*.gif *.jpig *.jpg)");

    ui->SavePathLabel->setText(savePath);

    // pixmap = new QPixmap (savePath);

    finalimage->save(savePath);




}
```

save_button.cpp

For the save button the principle is the same as the load button the only thing is that for the QFileDialog we use getSaveFileName instead of getopenFileName.

Chapter 4

Pixelization of the image

After the first part which was tload display and save, I wa s supposed to trasnform the loaded image into a second one such that the pixels color of the second image is computed accord to various method. I choose the method of the average to accomplish this task.

```
void MainWindow::on_Pixelized_clicked()
{

    pixelizedimage = new QImage (pimg->scaled(valueimagesize,valueimagesize,Qt::
    KeepAspectRatio));
    // ui->pixelizedLabel->setText(pixelizedimage);

    // get the label size
    int w, h;
    w=ui->displayLabel->width();
    h=ui->displayLabel->height();

    // with scale preserved
    ui->pixelizedLabel->setPixmap(QPixmap::fromImage(pixelizedimage->scaled(w,h, Qt
    ::KeepAspectRatio)));

}
```

pixelized_button.cpp

Here after the result of the pixelized code.

Figure 4.1: Pixelization with a size of the image = 250

Explanation: As you can see on the figure 4-1 the pixels are not directly showing we need to zoom as shown on figure 4-2 on the picture to analyze the pixels. This is due to the resolution choose for the image by default declare in the header file under two variables named valueimagesize which was fixed at 250.

Figure 4.2: Zoom on the figure 4-1

Now let's test the result when the valueimagesize is declare for a value of 50 intead of 250.

Figure 4.3: Pixelization with a size of the image = 50

Explanation: As the figure 4-3 show the result is much better using a low value for the size of the image.

Chapter 5

Drawing the pixels

The goal of this part is to transform the pixelized image into an art one where all the pixels will I been replace by a small image.

5.1 Load multiple images

```
void MainWindow::on_LoadMultiple_clicked()
{

    // StackOverflow, User : Apin, Date : Mar 15 2016
    // explanation : Here we will load all the image that we want to store into te
    database
    QString adressPath = QFileDialog::getExistingDirectory(this, tr("Select DB
    folder"));
    QDir dir(adressPath);
    QStringList filter;
    filter << QLatin1String("*.png");
    filter << QLatin1String("*.jpeg");
    filter << QLatin1String("*.jpg");
    dir.setNameFilters(filter);
    QFileInfoList filelistinfo = dir.entryInfoList();
    QStringList fileList;
    // END OF SOURCE

    // Go through every file

    foreach (const QFileInfo& fileinfo, filelistinfo) {
        QString imageFile = fileinfo.absoluteFilePath();
        //imageFile is the image path, just put your load image code here

        // My classemate Antoine Merlet provide me help for this part

        QImage img(imageFile); // creation of a temporary image available only in
    this part of the code.

        img = img.scaled( valuepixelsize, valuepixelsize, Qt::KeepAspectRatio); //
    creation of the pixelized image
        QImage *newimage = new QImage(img); // creation of a pointer of the new
    image
        resolutiondatabase.push_back(newimage); // store into the database

    }

}
```

loadmutiple.cpp

Explanation: First we have to be able to load multiple images, you then use QDir to allow us to access to a directory. in the first part of the code we will just analyze the folder choose and keep inside this folder only the images for our database. We also have to create a foreach loop which permit to go through all the files loaded. To load the image we have to create a temporary image used only in this part of the program named img. After we transform that tempory image into a pixelized one to also create a new image using dynamical cariable (new).

And finally we will store that images into the database.

5.2 Calculation of the mean

```
QColor* MainWindow::mean(QImage *img){ // pointer to my image which return a pointer
     to QColor
    // I use my classemate Antoine Merlet code for this part.

    // get image size
    int h = img->height();
    int w = img->width();

    // initilalization of the RGB
    int sumR = 0;
    int sumG = 0;
    int sumB = 0;

    // Going through the image
    for ( int i = 0; i < w; i++){
        for ( int j = 0; j < h; j++){

            // get the pixel value
            QColor pixelvalue( img->pixel( i, j));


            sumR += pixelvalue.red();
            sumG += pixelvalue.green();
            sumB += pixelvalue.blue();
        }
    }
    // compute the mean.
    sumR /= h * w;
    sumG /= h * w;
    sumB /= h * w;

    // creating the resulting color as a QRgb
    return new QColor( sumR, sumG, sumB);
}
```

mean.cpp

Explanation: We have to calculate the mean of the color in the image for the pixels. We will need the function mean later in our code.

5.3 Optimization of the pixels

```
QImage* MainWindow::optimizationpixels(const QColor& colorofpixel)
{

    // Explanation: In this part we have to find the best image corresponding to
    each pixels. Using the function mean above.

    // Counters for RGB and Total
    int differenceR, differenceG, differenceB, differenceT;

    // goal is the value to achieve for the best optimization
    int goal = 255 * 3;
    // we choose 255 cause the value of a pixel is between 0 and 255 and we
    multiplie by 3 cause 3 channels for the RGB

    // The image which obtain the best result in term of matching with the pixels
    QImage* matchingimage;


    // loop on both vectors at the same time
    for (std::vector<QImage*>::iterator it = resolutiondatabase.begin(); it !=
    resolutiondatabase.end() ; it++) {


        QImage* actualImage = *it;
        QColor* actualMean = mean(actualImage);

        // Compute the absolute mean difference
        differenceR = abs(colorofpixel.red() -  actualMean->red()); // absolute
    difference value of the given pixels and the mean of the pixels
        differenceG = abs(colorofpixel.green() - actualMean->green());
        differenceB = abs(colorofpixel.blue() -  actualMean->blue());

        // Total mean absolute difference
        differenceT = differenceR + differenceG + differenceB;

        // Check if new matching image result
        if (differenceT <= goal) {
            matchingimage = actualImage;

            // Everytime upload the best goal
            goal = differenceT;
        }
    }
    return matchingimage;
}
```

optimization.cpp

Explanation: We look to optimize every pixels in order to get them the best image matching with the corresponding pixels.

5.4 Art transformation

```
void MainWindow::on_Draw_Pixels_clicked()
{


    // get pixelized image size
    int h = pixelizedimage->height();
    int w = pixelizedimage->width();


    // initialization of the matching image result for a given pixel
    QImage* matchingimageResult ;

    // Prepare the result image with good size
   finalimage = new QImage( w * valuepixelsize, h * valuepixelsize, QImage::
    Format_RGB32);

    // for every pixel in the pixelizedimage
    for ( int i = 0; i < w; i++){
        for ( int j = 0; j < h; j++){

            // Get the actual pixel
            QColor actualpixel(pixelizedimage->pixel( i,j));

            // Get the best result in the DataBase
            matchingimageResult = optimizationpixels(actualpixel);


            // For every pixel in the ebst match low resolution
            for (int k = 0; k< matchingimageResult->width();k++){
                for (int l = 0; l < matchingimageResult->height(); l++){

                    // actual pixel
                    QColor actualMatch =matchingimageResult->pixel(k,l);

                    // copy on the output finalimage the actual pixel above.
                   finalimage->setPixel(i * valuepixelsize + k, j * valuepixelsize +
     l, actualMatch.rgb());
                }
            }
        }
    }

    ui->label->setPixmap(QPixmap::fromImage(*finalimage));// display the final image
     is the label and dereference the pointer final image
}
```

drawing_button.cpp

Explancation: This is the last part of the project code we have to recover all the matching images with their corresponding pixels and display it into a final image.

5.5 Result

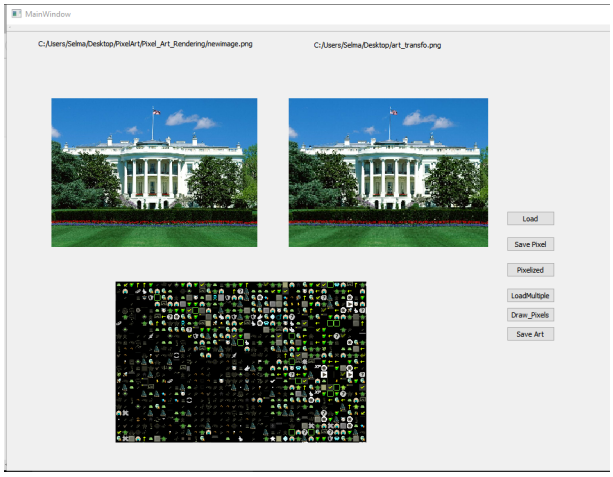


Figure 5.1: Final Interface

Explanation: The final output is not satisfying regarding on the result the interface. But the art transformation is well done to observe that you have to open the save art transform picture you have a much better result with details of the multiple images loaded from the database choose. Here after the result of the image saved.

Figure 5.2: Art transformation

Chapter 6

Conclusion

I was able, with the help provided by all the people that I thank, to achieve the goals of this all project. I am now able to load, display and save an image. And by using the average method I can transform the loaded image into a pixelized one.
This project help me a lot in term of project manadgement. I used trello which is a good way to organize the work and assign a task for each goal.
I realize my C++ level and saw where my difficulties was. I know with more work and research I will improve myself in the C++ programming.

References

1. **Definition of the QFileDialog**
   http://doc.qt.io/qt-5/qfiledialog.html
2. **Definition of the QImage**
   http://doc.qt.io/qt-5/qimage.html
3. **Definition of the QPixmap**
   http://doc.qt.io/qt-4.8/qpixmap.html
4. **Definition of the QLabel**
   http://doc.qt.io/qt-4.8/qlabel.html#details
5. **Definition of the QDir**
   http://doc.qt.io/qt-4.8/qdir.html
6. **My Trello board**
   https://trello.com/b/R1xYcKze/project-topics
7. **My Github account**
   https://github.com/SelmaBoubou/PixelArt
8. **Sources stack overflow**
   https://stackoverflow.com/questions/36005814/load-images-from-folder-with-qt