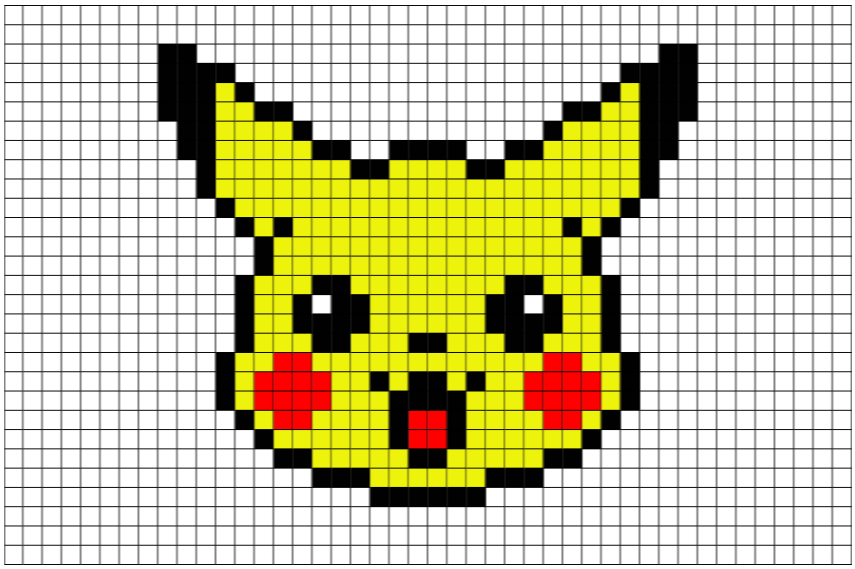
UE 2-1 Computer Science

Pixel Art Rendering Project



Submitted by **Meldrick REIMMER**

Contents

Acknowledgment

I would like to convey my gratefulness to all those that helped me during this project. I would not have been able to achieve all these had it not been you. Thank you for the time spent with me and showing how kind you are.

Special appreciation goes to Mr.Yohan Fougerolle (Supervisor),for helping me in times of need with his explanations during class sections and always giving exemplary notes for us to work on.

I also will like to say a big Thank you to my classmate Khoi Pham for helping me also in approaching the specific goals in which i set myself for this project. And not forgetting Selma Boudissa who sat by me during lab works and started this project with.

Thank you to my other classmates also for always encouraging each other and being there for one another.

Chapter 1

Introduction

Having a semester in a course module Computer Science, which involved acquiring knowledge in programming languages especially in C++ using Qt, which is also basically a cross-platform application used for developing an application software.

For my final project, I am to propose and implement a C++ using Qt, an application which is related to color manipulations. I am to create a program where by, an image could be loaded and displayed from any part of my hard-drive and my program should be able to get a second image from the loaded image such that the pixel color of the second image is computed using an average method and the results is displayed as a Pixelized Image.

After getting my Pixelized image, I am to get a third image from it where its "big and blurred" pixels will now be represented by a set of images in which I choose from my hard-drive and display the results as my Art Transform Image. In this report, you will see explained details for each step I took to get my project done.

Chapter 2

Objectives

My goals for this project has been set and for me to achieve each goal I undertook these various steps to get great results and be in track during the implementation of the project. For the first goal of the project, I am to allow my program to load and display an image from any part of my hard-drive and get a second image from the loaded image by finding the average of the pixel color of the loaded image.
For this part of the goal I undertook these various steps:

- Create an interface for the user to interact
- Load and Display an image.
- Find the average color value of each pixel block of my loaded image and display the results which will be my results for the "Pixelized Image"
- Save the "Pixelized Image"

For the second goal of the project, I am to allow my program to now load sample of images of my choice from my hard-drive and replace the "big and blurred" pixels of my "Pixelized Image" with the sample images and display it. For this part, I undertook these various steps:

- Load multiple sample images
- Access the sample images and calculate the mean of each.
- Compare the mean of the each pixel of the second image (Pixelized Image) to these sample images.
- Replace the pixel block by the best image found from the sample images and paint it on the position of the pixel block
- Display the new image which will be the results for the "Art Transform Image"

Chapter 3

User Interface

The user interface is the main window to the user which allows the user to interact with the program designed. A good user interface provides a "user-friendly" experience, allowing the user to interact with the software in a natural and intuitive way. I tried my possible best to design a good interface for my users to interact with.

With my interface, I used two QLabels, five QPushButtons and an horizntal line. One Qlabel for displaying the images and the other Qlabel for showing the path of the first loaded image. With the buttons, it consists of a load and display, pixelized, Loadx, ArtTransform and save buttons. The line just divides the button area from the Qlabel area.

Figure 3.1: User Interface

Chapter 4

Load and Display

My load and display button is suppose to perform the task of allowing the user to be able to go to any part of the hard-drive and select an image and display it once clicked on.

I first started by creating a QFileDialog for my user to be able to get the opportunity to select an image and I created a Qstring and named it "myimagelink", which will be used to store the absolute path of the selected image. After, I then assigned one of my Qlabel which I named "FilePathLabel" to show the path of the selected image. After I used my path which is my "myimagelink" to create "QPixmap", which I declared in my header file and gave it the value "pixmap", which will be used to store the image data, then I convert my QPixmap to QImage which a gave a value in the header file as "pimg".

Having this I then set the width and height of the selected image so that, the required size will be displayed. With this we then assign the other Qlabel which I named "DrawingLabel" to display the image selected by the user which is being stored as pixmap in a scaled using its appropriate size as it is.

Find the code below:

```
//Loading and Displaying function
void MainWindow::on_pushButton_clicked()
{
    QString myimagelink = QFileDialog::getOpenFileName(
                this,
                "Open a file",
                "",
                "Images (*.png *.gif *.jpeg *.jpg)");

    //Shows the path of my loaded image.
    ui->FilePathLabel->setText(myimagelink);

    // create a pixmap using image url above
    pixmap = new QPixmap(myimagelink);

    // convert QPixmap to QImage
    pimg = pixmap->toImage();

    //Setting the picture to display the required size of the image
    int w, h;
    w = ui->DrawingLabel->width();
    h = ui->DrawingLabel->height();

    //Scale the picture and load it on the drawinglabel
    ui->DrawingLabel->setPixmap(pixmap->scaled(w,h, Qt::KeepAspectRatio));

}
```

load_display.cpp

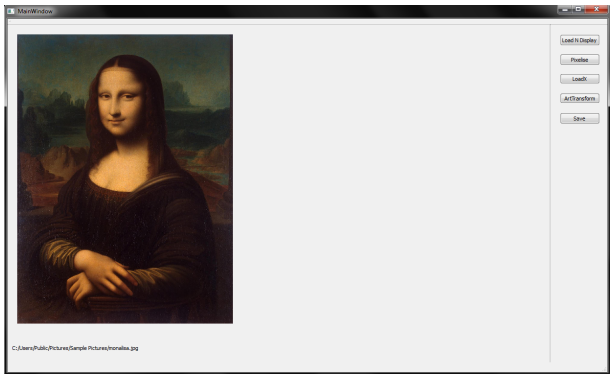


Figure 4.1: Load and Display

In the figure you will see i was able to load and display my selected image from any part my hard-drive and showing the required exact size of the image.

Chapter 5

Pixelization

The main concept this part is to find the average of the loaded picture and extract its color to replace it with a new color. To do this I will first calculate the average of the loaded picture. By doing this,I created two for loops. One loop to divide the image into smaller regions by shifting the window vertically and horizontally.

```
//Pixelizing the loaded Image.
//Having assitance from Khoi I was able to find the average of the picture and
    extract its color to replace it with new color.
void MainWindow::on_pushButton_3_clicked()
{
    //Loop to divide the image into smaller region.
    for(int i = 0; i < pimg.width(); i += size)             // this loop will shift
    the window vertically
        for(int j = 0; j < pimg.height(); j += size){       // this loop will shift
    the window horizontally

            int r=0,g=0,b=0,a=0;
```

pixelization.cpp

The second loop to go through every pixel of each window. Then I initialized variables to store the accumulated values of the pixels "r,g,b,a". After I divide each variable by the number of the pixel, to get the mean of each pixel and assign the new value to the variable

```
//A loop to go through every pixels of the each window
for(int k = 0; k < size; ++k)
    for(int l = 0; l < size; ++l){
        // stopping criterion
        if (i+k < pimg.width() && j+l <pimg.height()) {
            // Converting the QRgb to QColor
            QColor color(pimg.pixel(i+k,j+l));
            // extracting color channels using QColor built-in functions
            r += color.red();
            g += color.green();
            b += color.blue();
            a += color.alpha();
        }
    }
// calculate the mean color value of every channels
r /= size*size; g /= size*size; b /= size*size; a /= size*size;
```

pixelization2.cpp

After the mean value for each variable we then combine it into QRgb, since the "pimg.setPixel()" function will only accept QRgb function. So now we pass the mean in the function. Then after we display the results by assigning it to the Qlabel named "DrawingLabel" to show the results which will be our Pixelized results.

```
// combine 4 channels into QRgb data type
QRgb meanColor = qRgba(r,g,b,a);
```

```
// replacing the pixel of the window by the new color extracted
for(int k = 0; k < size; ++k)
    for(int l = 0; l < size; ++l)

        // same stopping criterion as above
        if (i+k < pimg.width() && j+l <pimg.height())
            pimg.setPixel(i+k, j+l, meanColor);

}


//Displayin the pixelised image.
*pixmap = QPixmap::fromImage(pimg);

//Setting the picture to display the required size of the image
int w,h;
w=ui->DrawingLabel->width();
h=ui->DrawingLabel->height();

//scale the picture and load it on the drawinglabel
ui->DrawingLabel->setPixmap(pixmap->scaled(w,h, Qt::KeepAspectRatio));
}
```

pixelization3.cpp

Below you will find my results after executing the above codes:

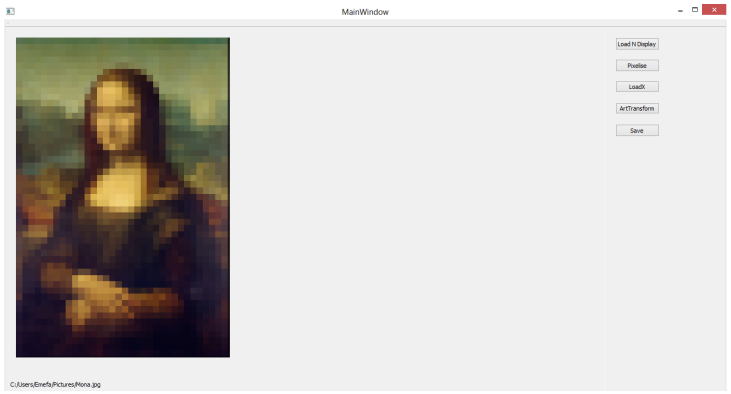


Figure 5.1: Pixelized Image

You will see from the figure that my loaded image now has become in a form of rough painted image this is due to the fact that its in the average form.

Chapter 6

Saving An Image

After getting the results we then, let the user to be able to save the results on any part of the hard-drive. Just has the QFileDialog was done in the load and display we will do the same with this part and in this case, I created a QString which I named "saveurl", to store the path in which the user chooses to store the pixelized image and after we save the image at the path the user choose by just assigning it to the QImage by "pimg.save(saveurl);"

Below you will see the code :

```
//Saving an image
void MainWindow::on_pushButton_2_clicked()
{
    // get the url of the file and folder when user click save in QFileDialog and
    use it to save the image
    QString saveurl = QFileDialog::getSaveFileName(this, tr(""),"",tr("Images (*.png
     *.jpeg *.jpg)"));
    pimg.save(saveurl);
}
```

save.cpp

You will find the results below :

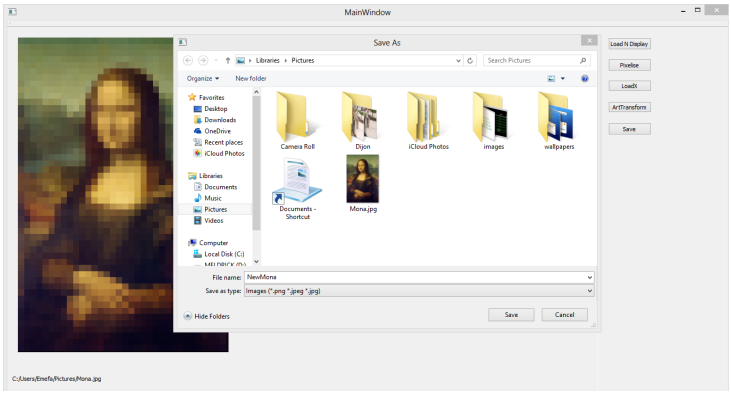


Figure 6.1: Saving The Pixelized Image

As you can see in the figure below I was able to save my pixelized image at any part of my hard-drive but I choose to save it in the same folder in which I had my loaded image and i named it as "NewMona"

Chapter 7

Loading Multiples Images

Just as I did in the first part for loading of an image and displaying. In this case, I will create a QStringlist for me to have a list of my path, of my selected images. I named my Qstringlist "myimagelinks" and a QFiledialog to allow the user to be able to select the multiple images at any part of the hard-drive. I then created a for loop to go through my QStringList "myimagelinks" which contains the path of my selected images to be able to create an image at every path and then stores the sample images created in a Qvector "sample.images".

After I need to get the mean colors of every sample image. Just as I did in the Pixelization section where I calculated the mean color of each window in this situation I will rather calculate the mean of every image and summing it and just store it in a Qvector "samplecolors".

```
//load multiple sample images funciton
void MainWindow::on_pushButton_4_clicked()
{
    QStringList myimagelinks = QFileDialog::getOpenFileNames(
                this,
                "Open a file",
                "",
                "Images (*.png *.gif *.jpeg *.jpg)");


    //store sample images in sampleimages attribute
    for(int i=0; i<myimagelinks.length(); i++){
        QImage tempimage(myimagelinks[i]);

        sampleimages.append(tempimage);   // store the created image in the vector
    to use in next function
    }

    //loop through sample image list and calculate mean colors
    //it will be the same like I did with pixel blocks on previous function
    QVector<QImage>::iterator it;
    for(it = sampleimages.begin(); it != sampleimages.end(); ++it){

        int r=0,g=0,b=0,a=0,sumcolor;

        for(int k = 0; k < it->width(); ++k)
            for(int l = 0; l < it->height(); ++l){
                // Converting the QRgb to QColor
                QColor color(it->pixel(k,l));
                // extracting color channels using QColor built-in functions
                r += color.red();
                g += color.green();
                b += color.blue();
                a += color.alpha();

            }
        // calculate the mean color value of every channels
```

```
            r /= it->width()*it->height();
            g /= it->width()*it->height();
            b /= it->width()*it->height();
            a /= it->width()*it->height();

            //sum the color to store in samplecolors list for value comparision in next
        function
            sumcolor = r + g + b + a;
            samplecolors.append(sumcolor);
        }

}
```

loadingx.cpp

You will find the results below :

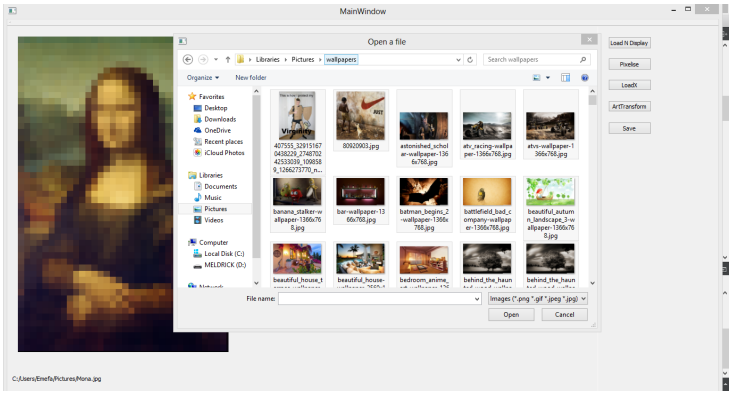


Figure 7.1: Loading Multiple Images

In the figure below you will see I was able to load multiple images from my hard-drive. I just copied all my images from my folder wallpapers which was around 150 images.

Chapter 8

Art Transformation

The first part of my art transform will just be the same as I did for the pixelization part here I reuse the same function as I created to do since I must loop through this process again to get the mean color of every pixel blocks and so I can do color comparison with sample colors of all the sample images.

```
//Art transform function
void MainWindow::on_pushButton_5_clicked()
{
    //here I reuse the same function as I created to do pixelizing
    //since I have to loop through this process again to get the mean color
    //of every pixel blocks and so I can do color comparision with sample colors

    //Loop to divide the image into smaller region.
    for(int i = 0; i < pimg.width(); i += size)
        for(int j = 0; j < pimg.height(); j += size){

            int r=0,g=0,b=0,a=0,sumcolor;

            //A loop to go through every pixels of the each window
            for(int k = 0; k < size; ++k)
                for(int l = 0; l < size; ++l){
                    //stopping criterion
                    if (i+k < pimg.width() && j+l <pimg.height()) {
                        // Converting the QRgb to QColor
                        QColor color(pimg.pixel(i+k,j+l));
                        // extracting color channels using QColor built-in functions
                        r += color.red();
                        g += color.green();
                        b += color.blue();
                        a += color.alpha();
                    }
                }
            //calculate the mean color value of every channels
            r /= size*size; g /= size*size; b /= size*size; a /= size*size;


            //sum color to do color comparision below
            sumcolor = r+g+b+a;


        }

}
```

art_transform.cpp

After getting the sum color of the pixelized image,I then must compare to my Qvector "samplecolors" and see the one which best matches to replace it within my images in my Qvector. With the comparison, I first created variable to store the best match color index which will be use to get the best match image which I called "bestcol-

orindex" and "initoffset" after I used an iterator to loop through each color to in finding the best match color in my QVector "samplecolors",

I then created a condition to update new values to the variables in which I created early on. Simply I am saying if the sample value is less than the value given in the variable "initoffset" then update that value in the variable. Then I get the position of the best color which will be my value of the variable "bestcolorindex".

```
//variables to store temporary best values
//bestColorIndex will be used to get the best matched image
int bestcolorindex, initoffset = 10000;

//loop through sample color to find best match color
QVector<int>::iterator it;
for(it = samplecolors.begin(); it != samplecolors.end(); ++it){
    //color value comparision to choose best match color
    if(abs(sumcolor-*it) < initoffset){
        initoffset = abs(sumcolor-*it);     //i used abs to neglect the minus number
     cases
        bestcolorindex = it-samplecolors.begin();   // get the position of the best
    color
    }
}
```

art_transform2.cpp

To display I then create a Qlabel to store the matched picture on the same location of the pixel block. I created it with the attributes of the coordinators of the pixel root where the label will be created, which is (i, j, size, size) where i,j is the coordinator of the pixel root and the size being the width and height of that location and I just display it.

```
//create label to store this matched picture, on the same
//location of the pixel block

QLabel *pixelimage = new QLabel(this);

pixelimage->setGeometry(QRect(i,j,size,size)); // The label to be in the right place
    .
pixelimage->setPixmap(QPixmap::fromImage(sampleimages[bestcolorindex]).scaled(QSize(
    size,size)));
pixelimage->show();

   }

}
```

art_transform3.cpp

Below you will find the results for my Art Transform :

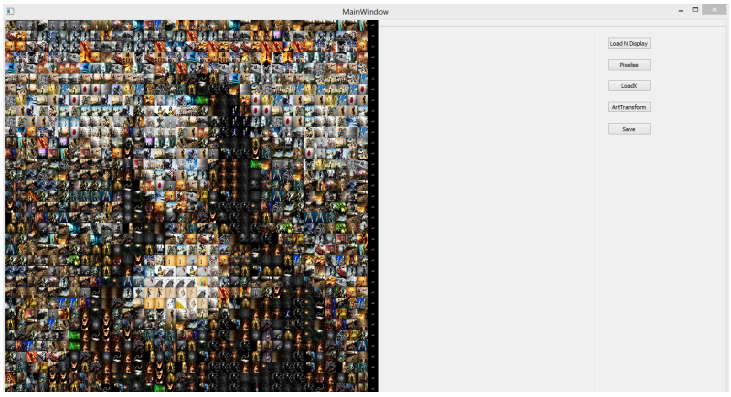


Figure 8.1: Loading Multiple Images

As you can see in the figure below you realized that with the pixelized image it appeared blurred due to the fact that we found the average of the loaded image but with this image we rather replaced the blurred pixels with best matching image from the multiple images loaded.

For you to get satisfactory results you then would have to load more images. The less images you load the less details of the initial output you will get but the more images you load, the more details you get since there would be many images at disposal for the program to process and select the best fit image to replace the pixels of the pixelized image.

Chapter 9

Library

Using Qt to create this program made me play around with few of its library. I will give you brief explanation to the library I used and why I used it in each case.

9.1 QString

This provide a unicode character string. I used this in creating ?myimagelink? ,?myimagelinks?, ?saveurl? all these stores the path in which my user selects when selecting and saving the image in every case.

9.2 QStringList

This provides a list of strings, I used this in creating ?myimagelinks? when loading multiples images in order for us to have the list of the path where the images was selected.

9.3 QPixmap

This is class is an off-screen image representation that can be used as a paint device. I used this to create ?pixmap? where I store my image data.

9.4 QFileDialog

This class provides a dialog that allows users to select files or directories. I used this anytime i needed and input from the user through my interface orthe user need to perform an action.For example when loading and saving an image.

9.5 QRect

This class defines a rectangle in the plane using integer precision.

9.6 Qsize

This class defines the size of a two dimensional object using integer point precision.

9.7 QColor

This class provides colors based on RGB, HSV, or CMYK values.

9.8 QVector

This Class is s template class that provides a dynamic array. I used this to store the mean of my multiple images and also the color i extracted from it.

9.9 Qt::KeepAspectRation

This defines what happens to the aspect ratio when scaling an rectangle.i used this in my loading and displaying function and also in my pixelization function to let the image to be shown be in its accurate ratio.

Chapter 10

Conclusion and Perspective

My goals have been achieved and I have had satisfactory results. I have been able to implement a program which loads and display an image from any part of my hard-drive. I have been able to find the average of the loaded image and displayed the result which was my Pixelized Image and I was able to load multiple images and replaced the pixelized image ?big and blurred? pixels with the best fit of the images I loaded.

I believe more work can be done with this project with future studies. This is not my end here I will go on to add more goals to this project.

References

1. **QT Help**
   https://www.qt.io/
2. **Stack Overflow**
   https://stackoverflow.com/
3. **Mr. Yohan Fougerolle**