UNIVERSITÉ DE BOURGOGNE

Pixel Art Rendering

PROJECT REPORT-Computer Science

Professor: Yohan Fougerolle

Author: Fahad Khalid

Table of Contents

AIMS 3
Pixel Art rendering 3
Preface 4
View of GUI 5
Load and Display the Image 5
CODE: 7
Pixelizing 8
CODE 9
DATABASE 10
CODE 11
Least Difference 11
CODE 12
ARTIT 13
CODE 13
Results 14
References 15

AIMS

Implement a Qt/C++ application related to color in the general sense.

- The ability to load and display an image located anywhere on your hard drive, and to save any processed image.
- The provided software should allow to transform the loaded image into a second one such that the pixels' color of the second image is computed according to various methods (average, median, most represented color, etc.) so that the image is pixelised. For instance, the image below shows an example of the expected result (using pixel filters in Gimp).

Pixel Art rendering

In this context, transform the pixelized image into a third one in which its "big and blurred" pixels are now represented with images from a set of images of your choice. You can choose as many images as you want, the only restriction being that these images should not be included as resources of the project, so that your application can automatically load any "database" of provided images from your drive without recompiling the entire programme.

Preface

The report presents the detailed study, proposed Architecture, implementation, conclusion, problems and future work of the project **"**Pixel Art rendering**"**. This development has gone through a series of sequential steps to its final form, which is explained by the chapters included in this report.

To remain illustrative, Creative, restricted and focused on developing a rich GUI Application, I developed a GUI desktop Application which I named as "Pixel Art". The applications is able to load any picture located anywhere on the hardrive. Once the user has loaded the picture a display button will be activated, when it is clicked on it will display the selected image. After the image has been displayed another button will be activated which is called 'pixelize", the pixelize button when pressed will convert the displayed picture into a pixeized version of it. After this step is where the real ability of my application comes in, there is another button located on the GUI called ART IT. The button when pressed will insert pictures coming from a data base located on the hardrive and not as resources in the application. This way each pixel will be replaced by a picture and the new picture will look like a montage of many pictures.

View of GUI

The flow of the report will be based on the Graphical User Interface.

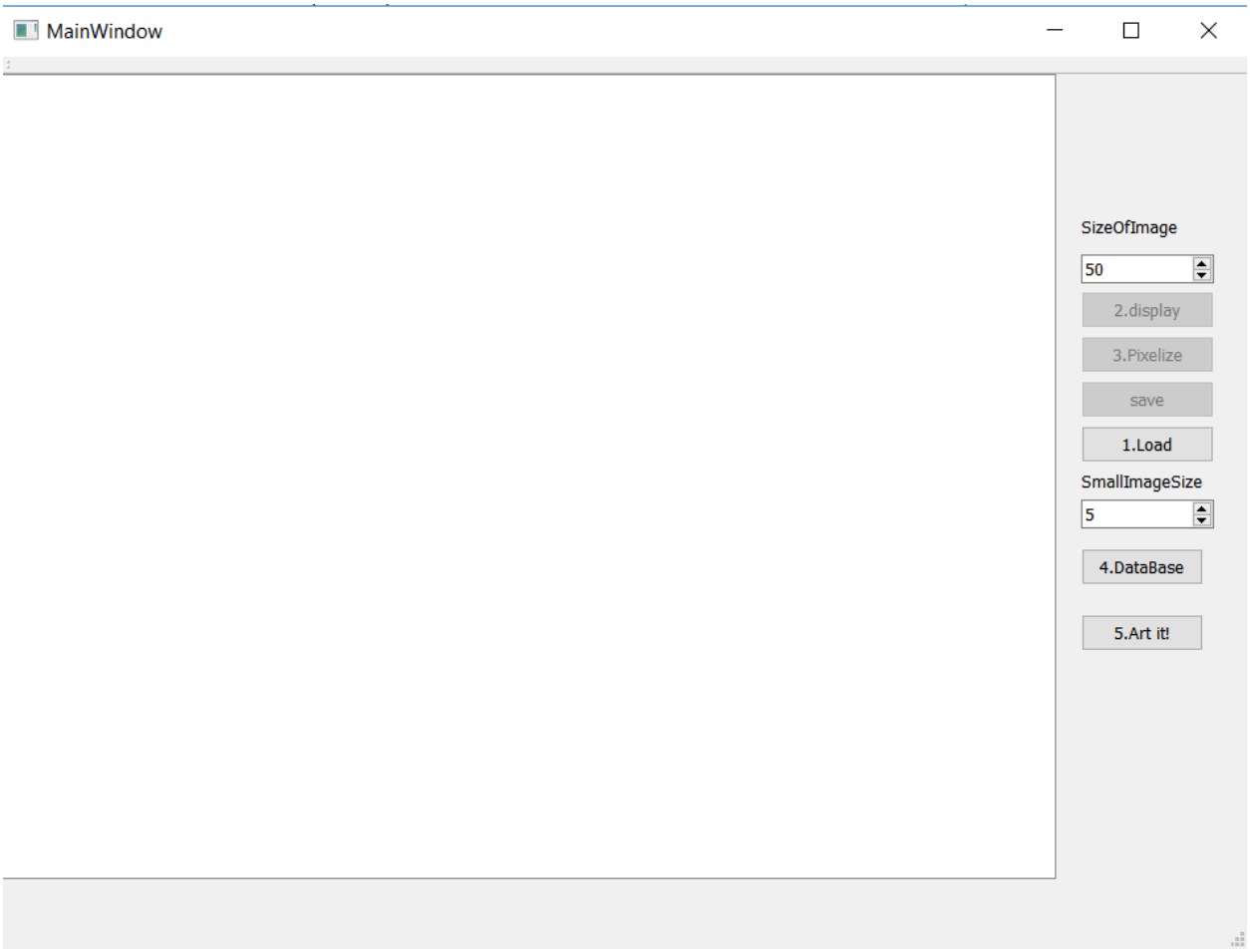

Figure 1: Graphial User Interface

Load and Display the Image

When the application is run, the first button to be used is the “Load” button. The purpose of the button is to open a dialogue and allow the user to browse and select the desired picture located anywhere on the hardrive.

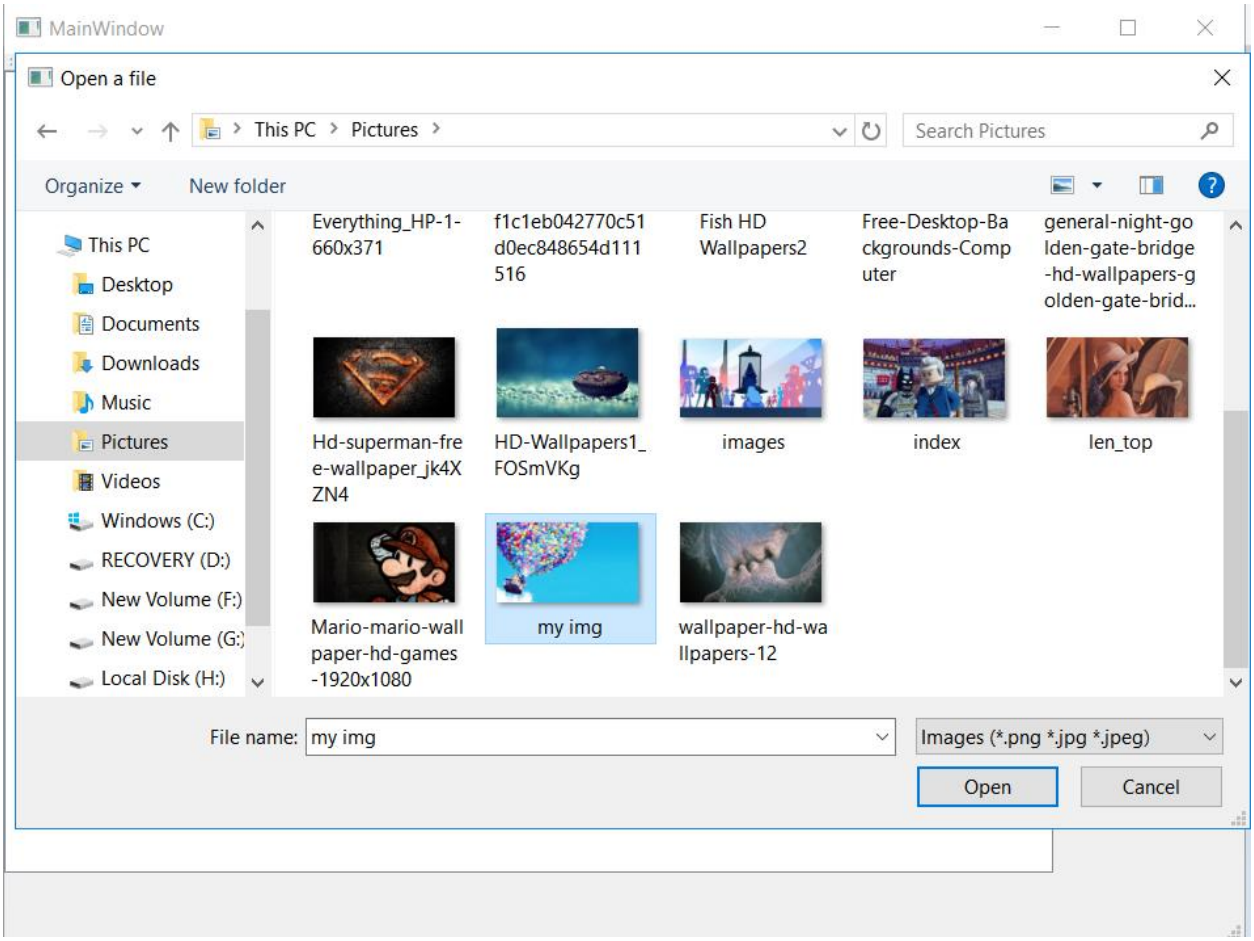

Figure 2: loading the image

Once the desired picture is selected. The display button is enabled. If the user chooses to select the display button the selected image will be displayed.

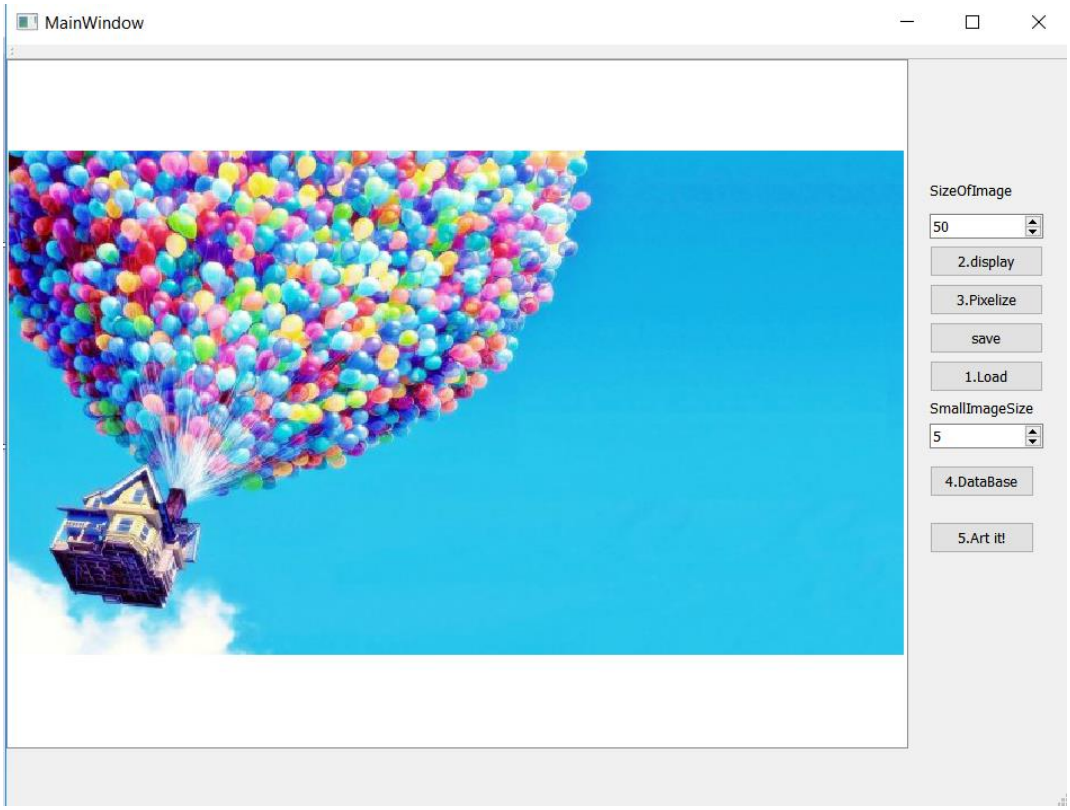

Figure 3: Displaying the image

CODE:

```
void MainWindow::on_pushButton_clicked()// load button
{


    QString imgpath=getImagePath(); // here we provide the path of the
    if (!imgpath.isNull()){
        originalImage->load(imgpath); // varible img set is equal to the QImage
        ui->pushButton_3->setEnabled(true); // enabeling our display button
    }
    //ui->graphicsView->fitInView(scn->itemsBoundingRect(),Qt::KeepAspectRatio);
    //image.save(saveImage(), "PNG");
}


QString MainWindow::getImagePath(){
    return QFileDialog::getOpenFileName(this,
                                        "Open a file",
                                        QString(),
                                        "Images (*.png *.jpg *.jpeg)");
}
```

```
void MainWindow::DisplayImage(QImage *img){ // display button
    scene->clear();
    scene->addPixmap(QPixmap::fromImage(*img)); // this
    ui->graphicsView->setScene(scene);
    ui->graphicsView->fitInView(img->rect(),Qt::KeepAspectRatio);// resize image to fit graphics view
    ui->pushButton_2->setEnabled(true);//enabeling save button
    ui->btnpixelize->setEnabled(true);// enable pixelize button
    displayedImage = img;// to keep trak of the latest displayed image.
}

void MainWindow::on_pushButton_3_clicked()// display button
{
    DisplayImage(originalImage);//calling display function
}

void MainWindow::on_pushButton_2_clicked()// save button
{
    QString savename = QFileDialog::getSaveFileName(this,// open dialogue to save image
                                                    tr("Save the image"),
                                                    "",
                                                    tr("Images (*.png)"));
    displayedImage->save(savename, "png");//save any displayed img
}
```

Pixelizing

The next step is to be able to pixelize the image. Basically when talked about pixelizing the image it implies that the resolution of the image is being reduced.

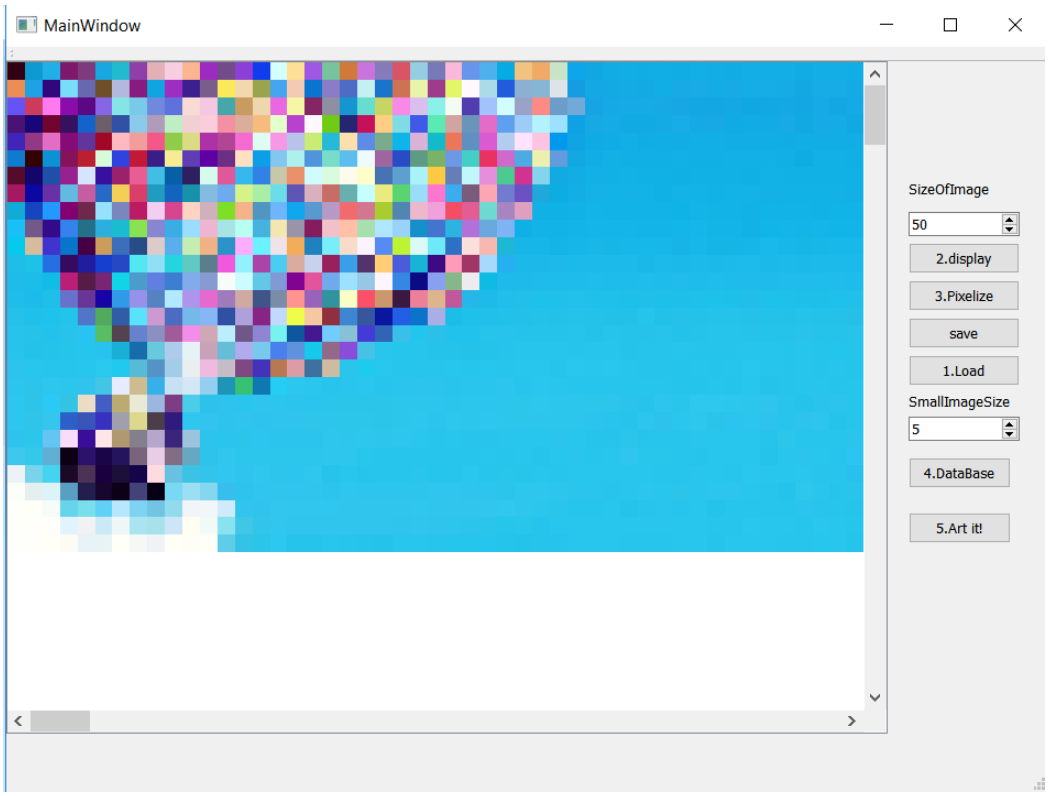

Figure 4: pixelized image

CODE

```
void MainWindow::on_btnpixelize_clicked()// pixelize
{
    //lowering he resolution
    LowResImage = new QImage(originalImage->scaled(SizeOfImage,SizeOfImage,Qt::KeepAspectRatio));
    DisplayImage(LowResImage);// display
}

void MainWindow::on_spinBox_2_valueChanged(int arg1)// to set pixel size.
{
    SizeOfImage = arg1;// to set pixel size.
}
```

"SizeOfImage" is the variable given to the spin box which takes the value from the user as a parameter to pixelize the image. For example if the size given by the user is 50. This enables the function to run a 50 by 50 filter on the image. It runs through the image calculating the mean of the pixels.

```
QColor* MainWindow::Average(QImage *img){
    // Computes the mean value of an Image{
    // determine image size
    int height = img->height();
    int width = img->width();

    // init counters in RGB
    int AllR = 0;
    int AllG = 0;
    int AllB = 0;
    // Goign through the image
    for ( int a = 0; a < width; a++){
        for ( int b = 0; b < height; b++){

            // get the current pixel value
            QColor presentpixel( img->pixel( a, b));

            // add the corresponding channels
                        AllR += presentpixel.red();
                        AllG += presentpixel.green();
                        AllB += presentpixel.blue();
                    }
                }
                // '/' to get the mean.
                AllR /= height * width;
                AllG /= height * width;
                AllB /= height * width;

                // creating the resulting color as a qRGB
                return new QColor( AllR, AllG, AllB);

}
```

The function computes the mean of the image, this is done by first determining the size(width and height) of the image. The function goes through the image giving mean RGB values these values are used to reconstruct a pexilized version of the image.

DATABASE

Now the database containing pictures to be replaced onto our pixelized version of the picture needs to be loaded. To achieve this there is a button on the GUI named “Database”. When clicked it opens a dialogue allowing the user to browse and select the desired folder to serve the purpose of data base for our application. The function will filter out all the unwanted files and only consider .png .jpeg and .jpg files. The function will move forward by making a list of all the recognized files and link their paths. Over here the second spinbox named “SmallImageSize” will come into play as it will have a value given by the user. This value will lower the resolution of all the data base images. In the end the function will create a single vector of all he images.

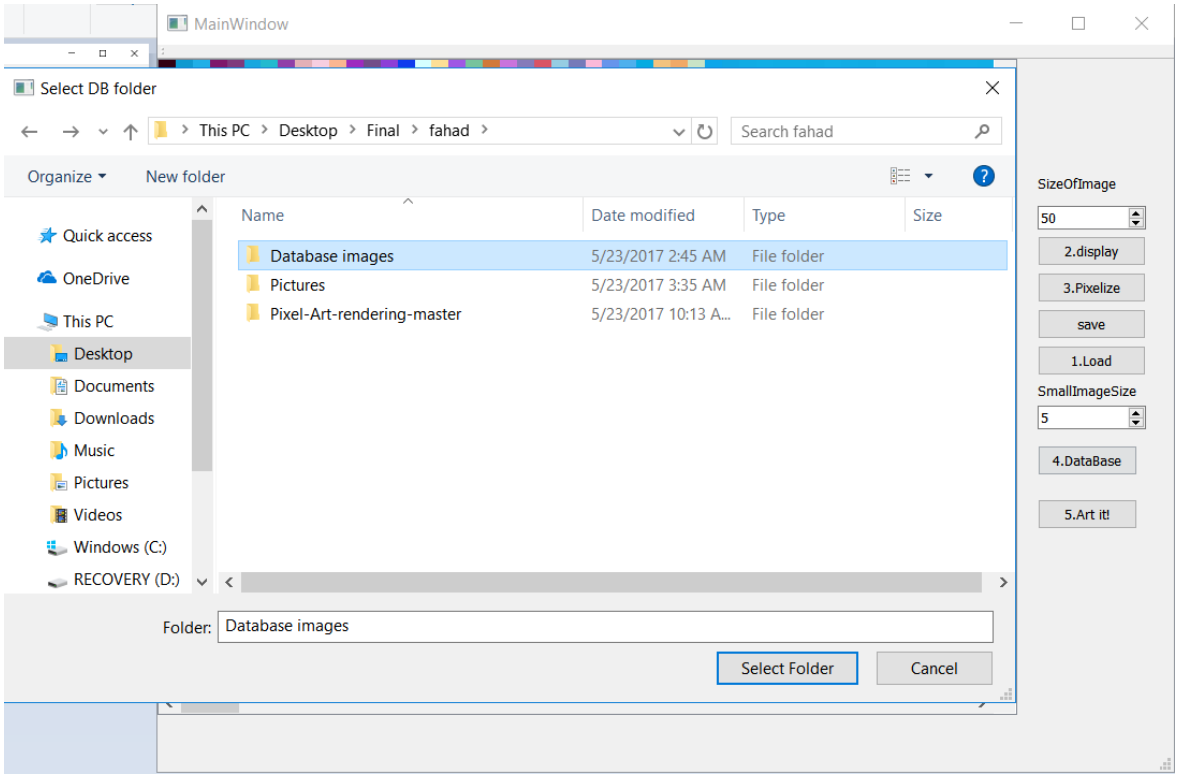

Figure 5: Selecting data base

CODE

```
void MainWindow::on_pushButton_4_clicked()
{
    // source :http://stackoverflow.com/questions/36005814/load-images-from-folder-with-qt
    // StackOverflow, User : Apin, Date : Mar 15 2016
    QString datapath = QFileDialog::getExistingDirectory(this, tr("Select DB folder"));
    QDir dir(datapath);
    QStringList filter;
    filter << QLatin1String("*.png");//filter to select .png image
    filter << QLatin1String("*.jpeg");//filter to select .jpeg image
    filter << QLatin1String("*.jpg");//filter to select .pjg image
    dir.setNameFilters(filter);
    QFileInfoList filelistinfo = dir.entryInfoList();//list of the filtered images


    // END OF SOURCE



    foreach (const QFileInfo& fileinfo, filelistinfo) {

        // Get file name
        QString itemroot = fileinfo.absoluteFilePath();//filepath

        QImage tmp(itemroot);

        //  pixelized version
        QImage *img = new QImage(tmp.scaled(SmallImageSize,SmallImageSize,Qt::KeepAspectRatio));

        // add data to vector
        lowImage.push_back(img);
    }
}
```

Least Difference

This function maybe one of the most important function for the working of our program. This function takes in the value of the pixels from the original pixelized image in the form of QColor. The range is set at a maximum of 765 (255*3) which is the range between the pixel color and the average value. The comparison between the pexilized image and average values calculated earlier is done. The image found to be the one with the least difference is selected and stored, updating the Goal.

CODE

```
QImage* MainWindow::LeastDifference(const QColor& pixcolor){

    // Try to find the best match in the DataBase given a color
   // Base on the mean absolute difference otpimisation fucntion
 // Counters for RGB and Total
       int divergentR, divergentG, divergentB, divergentT;

       // Base score, 3 channels, max 255/channel
       int Goal = 255 * 3;

       // latest result
       QImage* latestbest;

       // Parse the vector
       for (std::vector<QImage*>::iterator iter = lowImage.begin(); iter != lowImage.end(); iter++) {

           // Get current item
           QImage* readyImg = *iter;//current img
           QColor *readyMean = Average(readyImg);

           // Compute the absolute mean difference
           divergentR = abs(pixcolor.red() -  readyMean->red());
           divergentG = abs(pixcolor.green() - readyMean->green());
           divergentB = abs(pixcolor.blue() -  readyMean->blue());

           // Total mean absolute difference
           divergentT = divergentR + divergentG + divergentB;

           // Check if new latest result
           if (divergentT < Goal) {
               latestbest = readyImg;

               // Update latest goal
               Goal = divergentT;
           }
       }
       return latestbest;
}
```

ARTIT

The final button in the GUI, this button is connected to a common name function “ARTIT”. This function works by replacing each pixel of the pixelized version of our original image by the pixelixed data base images depending on the least difference in value.

A new image is created containing the data base image pixels with the least difference, they are printed at the corresponding position and this way the function finishes its job by having recreated a new image using the data base images only.

CODE

```
void MainWindow:: Artit(){

    {
        // Replace evry pixel in baseImg->lowImage by an image in DB which is the
        // Retreive the image
        // get Width and Height
        int h = LowResImage->height();
        int w = LowResImage->width();

        // best result for a given pixel initialization
        QImage* bestResult ;

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

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

                // Get current pixel
                QColor currentpixel(LowResImage->pixel( i,j));

                // Get the best result in the DataBase
                bestResult = LeastDifference(currentpixel);

                // Get pixelized version

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

                        // Get current pixel
                        QColor currentMatch =bestResult->pixel(k,l);

                        // Place on res at the good position
                        Result->setPixel(i * SmallImageSize + k, j * SmallImageSize + l, currentMatch.rgb());
                    }
                }
```

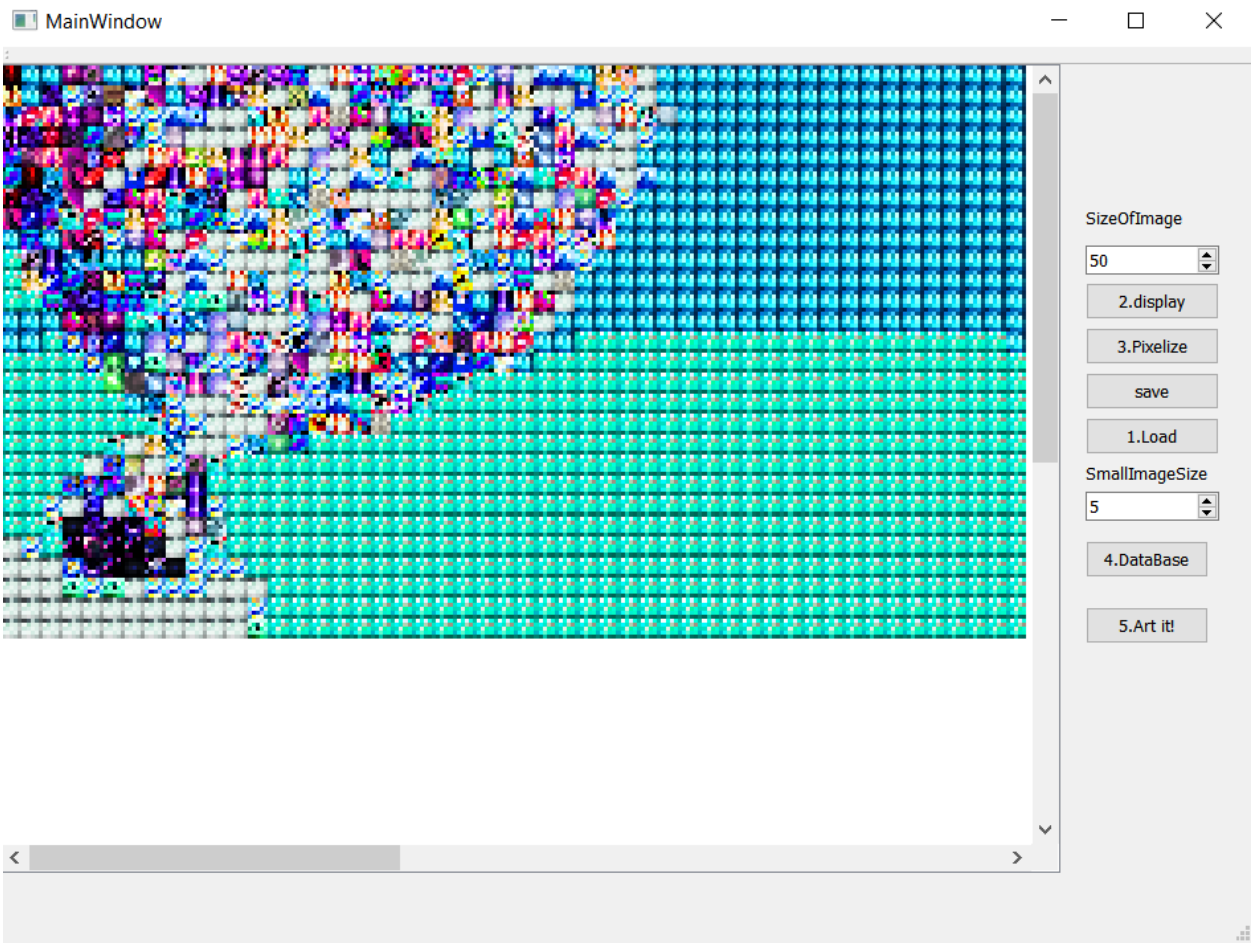

Figure 6: Arted image

Results

The application is able to put together an image corresponding to the original image by using data base pixelized images. The code and report will be available on my github.

References

http://stackoverflow.com/questions/36005814/load-images-from-folder-with-qt

StackOverflow, User : Apin, Date : Mar 15 2016