

Informa2 S.A.S

Parcial 2 - Implementación

Carolina Jimenez Restrepo c.c
1020470694

Departamento de Ingeniería Electrónica y
Telecomunicaciones
Universidad de Antioquia
Medellín
Septiembre de 2021

Índice

1. Sección introductoria	2
1.1. Clases implementadas	2
1.1.1. Clase escalar	2
1.1.2. Clase rgb	2
1.2. Estructura del circuito montado	3
1.3. Código implementado en Qt	3
2. Inclusión de imágenes	7

1. Sección introductoria

1.1. Clases implementadas

Las clases implementadas para el desarrollo del código que permitirá realizar los procesos de escalar la imagen, obtener los colores que componen la imagen y copiar estos en un archivo son las siguientes:

1.1.1. Clase escalar

Esta clase contiene funciones que permiten redimensionar la imagen, hacer el sobremuestreo o submuestreo según lo requiera la imagen, este proceso se realiza con base en el algoritmo de interpolación vecino más cercano, el cual permite recorrer los puntos de la nueva imagen y dividirlos por el factor de escala o zoom que necesita la imagen sea disminuir o aumentar, luego por la izquierda encontrar los puntos más cercanos la posición actual del pixel, por ultimo se le asignan los valores en cada posición a la nueva imagen

Explicacion del algoritmo interpolacion vecino mas cercano:

Interpolación vecino más cercano es un algoritmo que se usa para el procesamiento de imágenes en el proceso de escalado, esta interpolación consiste en tomar cada píxel original en la imagen de origen y copiarlo en su posición correspondiente en la imagen de destino. Al hacer la copia de pixel se generan espacios entre pixeles en la imagen de destino, estos espacios se llenan asignando a los pixeles vacíos el color del pixel de origen a la izquierda de la ubicación actual.

1.1.2. Clase rgb

Esta clase implementa una matriz tridimensional la cual está compuesta por 3 filas que representan cada color rgb y el número de leds que tenemos, en este caso 12 y 12 datos. Recorre pixel a pixel la imagen, empieza en la fila 0 que es el color rojo y recorre columna a columna pasando por cada pixel y guarda los datos de la intensidad del color rojo, el mismo proceso es para el color verde pero ya en la fila 1 y para el azul, pero en la fila 2. Se implementa una función que permite cambiar el valor del color blanco cuando este se encuentra en las tres filas y columnas por un valor muy cercano al valor del color blanco.

La escritura de la matriz: Para escribir la información obtenida en el archivo txt se usaron varios ciclos los cuales van recorriendo y escribiendo los valores por fila columna y dato, lo primero que se escribe es un llave que abre la estructura, cada fila va separada por llaves, cada dato separado por comas y cada fila va entre llaves, al finalizar se cierra la estructura con llave y coma.

1.2. Estructura del circuito montado

El montaje del circuito se compone de 12 tiras de NeoPixels en sentido horizontal, las cuales forman una matriz 12x12.

Una Protoboard que permite la conexión de los cables.

Una fuente de alimentación de 5v.

Un Arduino que permite el funcionamiento en la consola de tinkercad.

Las conexiones se hicieron de la siguiente manera:

Conexiones de la matriz: De la entrada a la salida, de potencia a potencia y de tierra a tierra de cada tira de Neopixels.

Conexiones matriz al Arduino: Se usa una protoboard como medio de conexión entre la fuente, el Arduino y la matriz, la primer conexión es de la fuente a la protoboarda en sus respectivos lugares de potencia y tierra la fuente se pone en 5v ya que debe tener el mismo voltaje del Arduino , la segunda es del Arduino a la protoboard la cual va del voltaje(5v) a la parte que representa la potencia en la protoboard y la ultima es de la entrada de la primer tira de Neopixels al pin 2 del Arduino, la potencia y la tierra van a los lugares representados en la protoboard que también se conectan con la potencia y tierra de la fuente

1.3. Código implementado en Qt

A continuacion se presenta el código desarrollado para el escalamiento y obtencion de colores de la imagen, esta dividido por clases.

main

```
#include <iostream>
#include <QImage>
#include <math.h>
#include "escalar.h"
#include "rgb.h"

using namespace std;

int main()
{
    Escalar img;
    rgb matr;
    QImage *imagen;
    QImage *imagen2;
    string imgg;
    cout<<"Ingrese la direccion de ubicacion de la imagen a proyectar:";
    getline(cin , imgg);
    imagen = new QImage(imgg.c_str());
    if(imagen->load(imgg.c_str())){
        cout << "Se cargo exitosamente la imagen" << endl;
        *imagen = imagen->convertToFormat(QImage::Format_RGB888); // formato de
```

```

        imagen2 =img.redimensionar(imagen);
        imagen2->save("Escalo.jpg");// para ver como se guardo
        matr.Obcolores(imagen2);
        matr.GuardarInfo();
    }
    else{
        cout << "No se pudo cargar la imagen.\nVerifique la ruta ingresada." <<
    }
    return 0;
}

escalar.h

#ifndef ESCALAR_H
#define ESCALAR_H
#include <iostream>
#include <QImage>
#include <math.h>

class Escalar
{
public:
    Escalar();
    QImage *redimensionar(QImage *imagen);
    float pixel = 12;

    //QImage *redimensionar(QImage *imagen);

    QRgb mezclar(QRgb r1, QRgb r2){
        return qRgb((qRed(r1)+qRed(r2))/2, (qGreen(r1)+qGreen(r2))/2, (qBlue(r1)+qBlue(r2))/2);
    };
};
#endif // ESCALAR_H

escalar.cpp

#include "escalar.h"
#include <iostream>
#include <QImage>
#include <math.h>
#include <fstream>
Escalar::Escalar()
{

}

QImage *Escalar::redimensionar(QImage *imagen)
{

```

```

//vecino mas cercano
float Zw = pixel/imagen->width(); // anchoOriginal*factorEscala=anchoEscala
float Zh = pixel/imagen->height();
int w = round(imagen->width()*Zw); // ancho original * factor de escala (zoom)
int h = round(imagen->height()*Zh);
int Nptx = 1, Npty = 1;
float x = 1.0, y = 1.0;

QImage *redimagen = new QImage(w,h,QImage::Format_RGB888);

for(int i = 0; i < h; i++){
    for(int j = 0; j < w; j++){
        x = j/Zw;
        y = i/Zh;
        Nptx = x;
        Npty = y;
        if((x-Nptx > 0.5) && (Nptx+1 < w)) Nptx+=1; //comprueba si el punto
        if((y-Npty > 0.5) && (Npty+1 < h)) Npty+=1;
        redimagen->setPixel(j,i,imagen->pixel(Nptx,Npty)); // asigno a la nu
    }
}
return redimagen;
}

rgb.h

#ifndef RGB_H
#define RGB_H
#include <QImage>
#include <math.h>
#include <fstream>

class rgb
{
private:
int matriz[3][12][12];
int colorR;
int colorG;
int colorB;

public:
rgb();
int leds=12;
void Obcolores(QImage *imagen);
void GuardarInfo();
};

```

```

#endif // RGB.H

rgb.cpp

#include "rgb.h"
#include <iostream>
#include <QImage>
#include <math.h>
#include <fstream>
using namespace std;
rgb::rgb()
{

}

void rgb::Obcolores(QImage *imagen) // funcion para obtener los valores de cada
{

    for(int fila = 0; fila < imagen->height(); fila++){ // recorre matriz
        for(int columna = 0; columna < imagen->width(); columna++){
            colorR= imagen->pixelColor(columna, fila).red();
            colorG= imagen->pixelColor(columna, fila).green();
            colorB= imagen->pixelColor(columna, fila).blue();
            if((colorR==255) && (colorG ==255)&&(colorB==255)){ // Para evitar c
                colorR=254;
                colorG=254;
                colorB=253;
            }

            matriz[0][fila][columna] = colorR;
            matriz[1][fila][columna] = colorG;
            matriz[2][fila][columna] = colorB;
        }

    }

    // for (int j = 0; j < 3; j++){ // imprime matriz en consola
    //     for(int i = 0; i < imagen->height(); i++){
    //         for(int m = 0; m < imagen->width(); m++){
    //             cout<<matriz[j][i][m]<<ends;
    //         }
    //         cout<<endl;
    //     }
    // }

}

```

```

void rgb::GuardarInfo() // funcion para guardar en txt informacion
{
    ofstream archivo;
    archivo.open("Matriz_Colores.txt", ios::out); // Abre el archivo o lo crea s

    //Para recorrer la matriz
    archivo << "{" << endl; //abre llaves
    for(int fila = 0; fila < 3; fila++){ //fila , columna, datos
        archivo << "{" << endl; // cada fila entre llaves
        for(int col = 0; col < leds; col++){ // se cierran con llaves
            archivo << "{";
            for(int dato = 0; dato < leds; dato++){
                archivo << matriz[fila][col][dato]; //separados con ,
                if(dato < leds-1) archivo << ",";
            }
            if(col < leds-1) archivo << "},"; //Longitud para saber si es la ultima
            else archivo << "}";
        }
        archivo << endl;
        if(fila < 2) archivo << "}," << endl;
        else archivo << "}" << endl;
    }

    archivo << "}"; //cierra llaves
    //Se cierra el archivo luego de trabajar con el
    archivo.close();
}

```

2. Inclusión de imágenes

En la Figura (1), se presenta el montaje del circuito.

Referencias

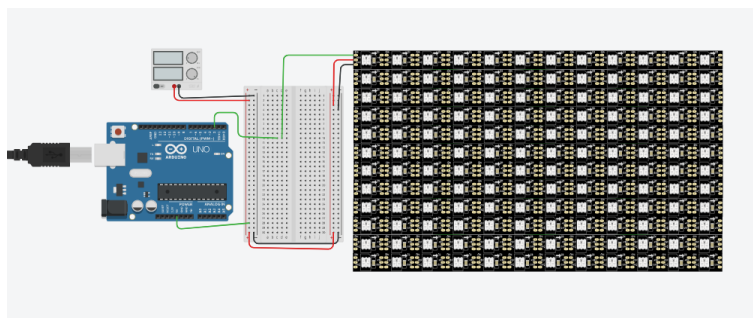


Figura 1: Circuito