Informa2 S.A.S

Parcial 2 - Implementación

Carolina Jiménez Restrepo
c.c 1020470694

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

Índice

1. Sección introductoria

El siguiente informe presenta como se aplicaron las soluciones encontradas y descritas en el informe de análisis y diseño, algunas soluciones planteadas tuvieron unas modificaciones ya que al aplicar las ideas planteadas surgían algunos imprevistos, pero todo se llevó a cabo según las tareas establecidas. También se describe el circuito implementado y el código usado para cumplir con el desafío propuesto.

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 a 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. Al

igual que con el color blanco, para el color negro se implementa una función que permite cambiar el valor del color negro cuando este se encuentra en las tres filas y columnas por un valor muy cercano al valor del color negro.

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.

1.1.3. Esquema de clases

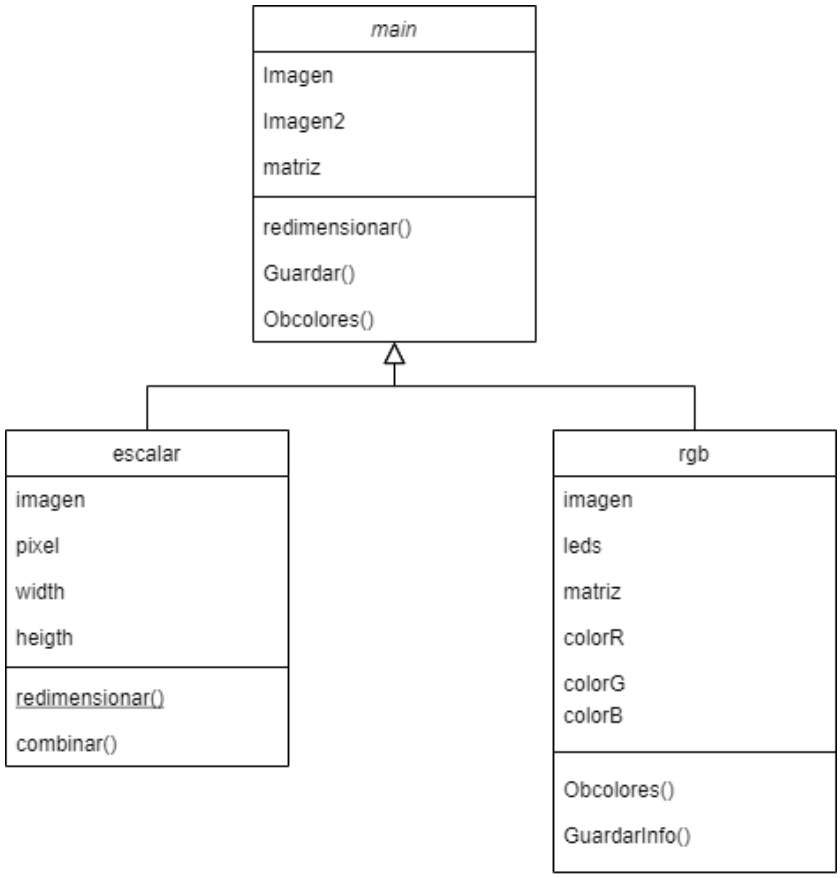

Figura 1: Clases

1.2. Codigo implementado en Qt

A continuacion se presenta el codigo 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;

    QRgb combinar(QRgb r1, QRgb r2){
        return qRgb((qRed(r1)+qRed(r2))/2, (qGreen(r1)+qGreen(r2))/2, (qBlue(r1)-
    }
};
#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=anchoEscalad
    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==0) && (colorG ==0)&&(colorB==0)){ // Para evitar conflic
                colorR=1;
                colorG=1;
                colorB=1;
            }
            if((colorR==255) && (colorG ==255)&&(colorB==255)){ // Para evitar c
                colorR=254;
                colorG=254;
                colorB=254;
            }

            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 ulti
            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();
}
```

1.3. 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 fuete 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 pin de tierra a la parte que representa la tierra en la protoboard y asi crear un acople de tierras, 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.

1.4. Problemas presentados

Al implementar la solución planteada en el informe anterior se encontraron problemas con el escalamiento de la imagen ya que el código que se implemento para este solo permita ampliar imágenes, por lo cual se debió recurrir a buscar una nueva solución, la cual consistía en realizar un algoritmo de interpolación que permite ampliar una imagen al tamaño deseado.

Al momento de extraer la información de los colores de los pixeles y llevar la matriz a tinkercad, cuando se ejecuta el programa si la matriz tiene color blanco en sus tres columnas esto presenta un comportamiento inestable en tinkercad, por lo cual se debió crear una condición al momento de extraer los colores, para que cambiara los valores del blanco en las tres columnas por un valor muy

cercano que represente el color blanco. Color blanco representado por 255 cambiado a 254.

Cuando la imagen a proyectar contiene color negro este se ve apagado en la matriz de leds ya que el color negro no tiene luz y por tal motivo no se nota que emite un color, además la representacion rgb de negro (0 0 0) representa los leds apagados por esto se hizo una condición parecida a la del color blanco para que se vieran los leds encendidos en un color muy cernado al color negro (gris) y así poder notar que los leds encienden cuando una imagen contenga el color negro. Negro en rgb representado por 0 cambiado a 1.

Durante la implementación, al momento de montar el circuito fueron encontradas dos formas de conectar las tiras de Neopixels entre ellas, esto no fue como tal un problema, pero si causo algo de duda ya que no se sabia si alguna de las dos era más eficiente que la otra, para dar solución a la duda se consultó con el Profesor y las dos formas de conectar son correctas e igual de eficientes.

En la Figura (2), (3), se presenta la diferencia en la matriz cuando se realiza el cambio de valores que representan el color negro.

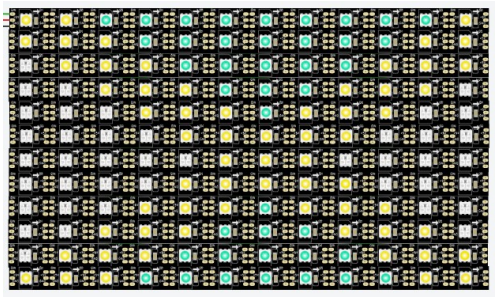

Figura 2: Matriz con leds negros

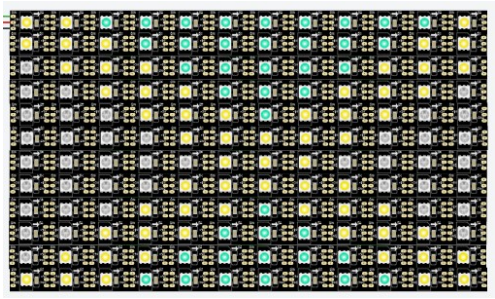

Figura 3: Matriz con cambio de valores

2. Inclusión de imágenes

En la Figura (4), se presenta el montaje del circuito.

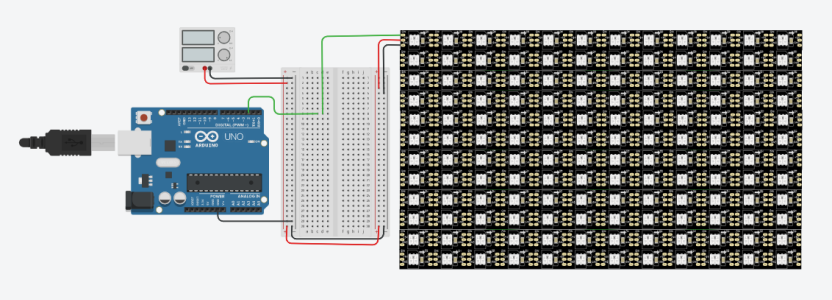

Figura 4: Circuito

[1].

Referencias

[1] NetinBag. ¿qué es el escalado de imagen? [Online]. Available: https://www.netinbag.com/es/internet/what-is-image-scaling.html