Implementacion Planteada

Informa2.

JESUS ANTONIO IBARRA AGUDELO
JUAN ANDRES URBIÑEZ GOMEZ

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

Índice

1. Contenido **2**
1.1. Las clases implementadas 2
1.2. Esquema donde describa la estructura final de las clases implementadas. 2
1.3. Módulos de código implementado donde se evidencie la interacción entre las diferentes clases. 4
1.4. Estructura del circuito montado. 9
1.5. Problemas presentados durante la implementación. 9

2. Manual **10**

1. Contenido

Segun los requisitos establecidos por la actividad, se describira la implementacion de la solucion planteda para este desafio.

1.1. Las clases implementadas

- Se creó la clase ImagenRead, en esta clase se añadieron dos métodos, el submuestreo que nos permite modificar la imagen original a una versión más pequeña y el sobremuestreo, que amplía la imagen original.
- Se incluyó la clase QImage de QT, de esta clase se utilizaron los métodos pixelColor para obtener el color del pixel en las coordenadas (X, Y). Además, se utilizó los métodos heigth y width para obtener la altura y el ancho respectivamente de nuestra imagen; como también el método load para cargar la imagen desde nuestro archivo y red, green, blue para obtener los valores de color de cada uno de los pixeles.
- Se utilizó la clase fstream, ya que nos permite guardar los valores de los espacios de color de los pixeles en un archivo txt después del submuestreo o sobremuestreo, se guarda de tal forma que parezca una matriz.
- En tinkercad, se llamó la clase Adafruit-NeoPixel, la cual nos permitió ingresar la cantidad total de leds a encender, como también los valores de color de los pixeles, con estos elementos podemos encender la tira NeoPixel con los colores deseados.

1.2. Esquema donde describa la estructura final de las clases implementadas.

1. Se creó la clase ImagenRead, en esta se añadieron los métodos submuestreo y sobremuestreo.
    - El metodo submuestreo, con ayuda de la lectura de imagen, se verificará que la altura y el ancho de la imagen original sea divisible por 16, ya que nuestra matriz es de 16x16.

      La altura y el ancho de la imagen se divide entre 16 y se resta 1, esta va a ser el valor inicial de la indexación de la imagen en X y en Y. Ha este valor después de cada ciclo se le sumara el valor del ancho o el largo entre 16, lo que hará que se tomen los valore RGB de 16 pixeles por fila 16 veces, por lo que conseguiremos hacer la matriz de 16x16.
    - En el sobremuestreo se tuvieron en cuenta dos nuevos ciclos, los cuales dividían entre 16 la altura y el ancho para permitirnos saber cuántas veces se debían repetir los pixeles para que tuvieran la misma dimensión de nuestra matriz de 16x16. En caso que no sean divisibles,

teníamos que añadir la cantidad de pixeles suficientes para que se pudiera repetir los ciclos de tal forma de que se adaptara a nuestra matriz de 16x16.

Cuando había residuo entre la división de 16 por la altura o el ancho, se guardaba en una variable, esto seria los pixeles que tenemos que añadir. Cuando se hacia el ciclo para repetir los pixeles de nuestra imagen, con un condicional añadíamos los pixeles restantes en el transcurso del ciclo, mientras que nuestra variable donde se añadía los pixeles que faltaban, iba disminuyendo hasta cero, eso nos quiere decir que ya no hay más pixeles que tenemos que añadir y cuando ya no se añade más, quiere decir que todos los pixeles necesarios para completar nuestra matriz de 16x16 ya se han añadido.

- Se creó un método llamado modificador, este método lo que hará es submuestrear y sobremuestrear la imagen al mismo tiempo. Se creó para modificar las imágenes que tiene una dimensión muy desigual, por ejemplo, una imagen 4x100. Con imágenes de dimensiones tan disparejas no se puede usar el submuestreo o sobremuestreo, ya que estos métodos conservan la relación del aspecto de nuestra imagen, por tanto, se hizo este nuevo método para cuando se necesite modificar la imagen de tal forma de que no conserva la relación del aspecto.

  Para aplicar este método, primero se sabe que el submuestreo se hace cuando la altura y el ancho es mayor a 16, y el sobremuestreo es cuando la altura y el ancho son menores a 16. Ahora, para este método, se mira si la altura es mayor a 16, si esto es verdadero sabemos que tenemos que submuestrear en Y y sobremuestrear en X, ya que si la altura es mayor a 16, podemos inferir que el ancho es menor a 16.

2. Además, tanto el submuestreo como el sobremuesreo guardaran los pixeles en un archivo txt utilizando los métodos de la clase fstream para abrir el archivo en modo de escritura, permitiendo guardar la información de los espacios de color de los pixeles de tal forma que parecerá una matriz de 16x16 con tres elementos cada uno, estos elementos serán los colores rojo, verde y azul. Tampoco se nos puede olvidar que estos dos métodos utilizan las funciones de la clase QImage para leer la imagen original y obtener el valor de los espacios de color de los pixeles deseados.

3. En tinkercad, utilizaremos dos macros, las cuales serán el PIN y la cantidad total de leds que están conectados desde la protoboard a las tiras NeoPixel, con un arreglo tridimensional guardaremos la matriz copiada del archivo de txt. En el setup, utilizando un ciclo recorremos todo el arreglo y los almacenamos en unas variables, con la clase Adafruit-NeoPixel, ingresaremos la macro de total de leds y los valores obtenidos del arreglo, que serán los espacios de color que los pixeles, con el método show de la clase

Adafruit-NeoPixel y estos valores ingresados en el método setPixelColor, podremos encender los leds de las tiras NeoPixel de nuestro circuito.

1.3. Módulos de código implementado donde se evidencie la interacción entre las diferentes clases.

- Primero que todo, hemos creado nuestra clase ImagenRead como muestra la siguiente imagen:

```
#ifndef IMAGENREAD_H
#define IMAGENREAD_H

#include <iostream>
#include <QImage>
#include "fstream"

using namespace std;

class ImagenRead
{
public:
    ImagenRead();
    void submuestreo(string fillname);
    void sobremuestreo(string fillname);
    void modificador(string fillname);


private:
    int rojo,verde,azul;
    int c=0,c2=0;
    string nombreArchivo= "../Parcial_2/Matriz.txt";
    ofstream archivo;
};

#endif // IMAGENREAD_H

```

En esta clase hemos añadido los métodos de submuestreo, sobremuestro y modificador, incluimos dos clases llamadas QImage y fstream, como atributos definimos los colores de los pixeles, dos contadores, un string que almacena la dirección de nuestro archivo txt llamada “Matriz.txt” y la variable archivo siendo un objeto de la clase ofstream.

- En el método de submuestreo:

```
void ImagenRead::submuestreo(string fillname)
{
    QImage im(fillname.c_str());

    archivo.open(nombreArchivo.c_str(), fstream::out);
    archivo<<"{"<<endl;
```

Lo primero que haremos es abrir el archivo txt en modo de escritura, en este archivo será donde guardemos nuestra imagen en forma matriz después de haberle hecho el submuestreo. Se podrá abrir el archivo con ayuda de la clase fstream.

```
archivo<<"{"<<endl;

for(int indey=im.height()/16-1;indey<im.height();indey+=im.height()/16)
{
    archivo<<"{"<<endl;
    for(int index=im.width()/16-1;index<im.width();index+=im.width()/16)
    {
        rojo=im.pixelColor(index,indey).red();
        verde=im.pixelColor(index,indey).green();
        azul=im.pixelColor(index,indey).blue();
        if(rojo==255 && verde==255 && azul==255){azul=254;}else if(rojo==0 && verde==0 && azul==0){azul=1;}
        archivo <<"{"<<rojo<<","<<verde<<","<<azul<<"}";
        c=c+1;
        if(c%16!=0)
        {
            archivo<<",";
        }
    }
    c2++;
    archivo<<endl<<"}"<<endl;
    if(c2%16!=0)
    {
        archivo<<",";
    }
}

archivo<<"};"<<endl;
archivo.close();
}
```

- Después de abrirse el archivo, con un ciclo recorreremos la altura de nuestra imagen original dividiendo entre 16 y restando 1, esta va a ser el valor inicial de la indexación de la imagen en Y, con otro ciclo, haremos el mismo procedimiento en el ancho de la imagen, así obtendremos la indexación de la imagen en X. Ha este valor después de cada ciclo se le sumara el valor del ancho y el largo entre 16 mientras que la indexación sea menor al largo y el ancho de la imagen original, lo que hará que se tomen los valore RGB de 16 pixeles por fila 16 veces, por lo que conseguiremos hacer la matriz de 16x16. Cada vez que haga el ciclo iremos añadiendo los valores de rojo, verde y azul en al archivo txt, estos valores de los pixeles los sacaremos con el método pixelColor al proporcionarle coordenadas, y con red, Green

y blue sacaremos el color, estos métodos son proporcionados por la clase QImagen. finalizamos cerrando el archivo y obtenemos nuestra matriz de 16x16.

- En el caso del sobremuestreo, comenzaremos de la misma forma que el submuestreo, abriendo un archivo txt en modo de escritura con el método proporcionado por la clase fstream.

  Después se crearán dos variables llamadas PixelesFaltantesAltura y PixelesFaltantesAncho, estas dos variables almacenaran el módulo de 16 entre la altura y el ancho de nuestra imagen original, este módulo nos servirá para saber cuántos pixeles hay que añadir a la altura y ancho para que la imagen resultante nos quede de dimensiones 16x16, concordando con la matriz que queremos.

```
PixelesFaltantesAltura=16%im.height();
archivo.open(nombreArchivo.c_str(), fstream::out);
archivo<<"{"<<endl;


int PixelesFaltantesAncho=16%im.width();

for(int indey=0;indey<im.height();indey+=1)
{
    for(int clockY=0;clockY<16/im.height();clockY+=1)
    {
        archivo<<"{"<<endl;

        if(PixelesFaltantesAltura!=0)
        {
            for(int index=0;index<im.width();index+=1)
            {
                for(int clockX=0;clockX<16/im.width();clockX+=1)
                {
                    rojo=im.pixelColor(index,indey).red();
                    verde=im.pixelColor(index,indey).green();
                    azul=im.pixelColor(index,indey).blue();

                    if(rojo==255 && verde==255 && azul==255){azul=254;}else if(rojo==0 && verde==0 && azul==0){azul=1;}

                    archivo <<"{"<<rojo<<","<<verde<<","<<azul<<"}";
                    c++;

                    if(c%16!=0)
                    {
                        archivo<<",";
                    }
                }
```

Se utilizarán dos ciclos para recorrer la altura y el ancho, además, se crearán dos nuevos ciclos (con variables clockY para la altura y clockX para el ancho), los cuales dividían entre 16 la altura y el ancho para permitirnos saber cuántas veces se debían repetir los pixeles para que tuvieran la misma dimensión de nuestra matriz de 16x16. En caso que no sean divisibles, tenemos que añadir la cantidad de pixeles suficientes que están almacenados en nuestras variables PixelesFaltantesAltura y PixelesFaltantesAncho, en cada ciclo iremos añadiendo los valores de rojo, verde y azul en al archivo txt, estos valores de los pixeles los sacaremos con el método pixelColor al proporcionarle coordenadas, y con red, Green y blue sacaremos el color, estos métodos son proporcionados por la clase QImagen.

```
//agregado
if (PixelesFaltantesAncho!=0)
{
    rojo=im.pixelColor(index,indey).red();
    verde=im.pixelColor(index,indey).green();
    azul=im.pixelColor(index,indey).blue();

    if(rojo==255 && verde==255 && azul==255){azul=254;}else if(rojo==0 && verde==0 && azul==0){azul=1;}

    archivo <<"{"<<rojo<<","<<verde<<","<<azul<<"},";
    c++;
    PixelesFaltantesAncho--;
}
```

Con este condicional, podremos añadir la cantidad faltante de pixeles en el ancho de la imagen, mientras que PixelesFaltantesAncho sea diferente de cero. Este mismo método se aplica para añadir los pixeles faltantes en la altura, solo que en este caso, el condicional está ubicado en la parte superior antes de entrar en la parte del ciclo que recorre y añade los pixeles en el ancho. Utilizamos los mismos métodos proporcionados por QImagen mencionados anteriormente.

- En nuestro método modificador código, se empezará de la misma forma que en submuestreo o sobremuestreo, abriendo un archivo txt en modo de escritura con el método proporcionado por la clase fstream.

```
if (im.height()>16)
{
    for(int indey=im.height()/16-1;indey<im.height();indey+=im.height()/16)
    {
        archivo<<"{"<<endl;
        for(int index=0;index<im.width();index+=1)
        {
            for(int clockX=0;clockX<16/im.width();clockX+=1)
            {
                rojo=im.pixelColor(index,indey).red();
                verde=im.pixelColor(index,indey).green();
                azul=im.pixelColor(index,indey).blue();
                if(rojo==255 && verde==255 && azul==255){azul=254;}else if(rojo==0 && verde==0 && azul==0){azul=1;}
                archivo <<"{"<<rojo<<","<<verde<<","<<azul<<"}";
                c++;
                if(c%16!=0)
                {
                    archivo<<",";
                }
            }

            //agregado
            if (PixelesFaltantesAncho!=0)
            {
                rojo=im.pixelColor(index,indey).red();
                verde=im.pixelColor(index,indey).green();
                azul=im.pixelColor(index,indey).blue();
                if(rojo==255 && verde==255 && azul==255){azul=254;}else if(rojo==0 && verde==0 && azul==0){azul=1;}
                archivo <<"{"<<rojo<<","<<verde<<","<<azul<<"},";
                c++;
                PixelesFaltantesAncho--;
            }
        }

        PixelesFaltantesAncho=16%im.width();

        c2++;
        archivo<<endl<<"}"<<endl;
        if(c2%16!=0)
```

Miremos que esta vez hay un condicional al comienzo, este nos dice que mientras la altura sea mayor a 16, utilizaremos el método de sobremues-

treo o submuestreo anteriormente mencionado incluso con el condicional que nos permite agregar pixeles que nos hace falta en el ancho, obtenemos los pixeles con pixelColor, red, Green y blue proporcionado por la clase QImagen.

En caso contrario:

```
for(int indey=0;indey<im.height();indey+=1)
{
    for(int clockY=0;clockY<16/im.height();clockY+=1)
    {
        archivo<<"{"<<endl;

        if(A!=0)
        {
            for(int index=im.width()/16-1;index<im.width();index+=im.width()/16)
            {
                rojo=im.pixelColor(index,indey).red();
                verde=im.pixelColor(index,indey).green();
                azul=im.pixelColor(index,indey).blue();
                if(rojo==255 && verde==255 && azul==255){azul=254;}else if(rojo==0 && verde==0 && azul==0){azul=1;}
                archivo <<"{"<<rojo<<","<<verde<<","<<azul<<"}";
                c=c+1;
                if(c%16!=0)
                {
                    archivo<<",";
                }

            }
            PixelesFaltantes=16%im.width();    ⚠ Value stored to 'PixelesFaltantes' is never read [clang-analyzer-d


            archivo<<endl<<"},"<<endl;
            c2++;
            archivo <<"{"<< endl;
            A--;
        }
        for(int index=im.width()/16-1;index<im.width();index+=im.width()/16)
        {
            rojo=im.pixelColor(index,indey).red();
            verde=im.pixelColor(index,indey).green();
            azul=im.pixelColor(index,indey).blue();
            if(rojo==255 && verde==255 && azul==255){azul=254;}else if(rojo==0 && verde==0 && azul==0){azul=1;}
            archivo <<"{"<<rojo<<","<<verde<<","<<azul<<"}";
            c=c+1;
            if(c%16!=0)
```

Aplicamos los mismos métodos de sobremuestreo o submuestreo para modificar la altura de la imagen. Se utilizan los mismos métodos de QImagen para obtener los valores del espacio de color de los pixeles de la imagen.

- Cabe mencionar que los ciclos y algunos condicionales aplicados en el submuestreo, sobremuestro y modificador, son casi los mismo explicados anteriormente por separado.

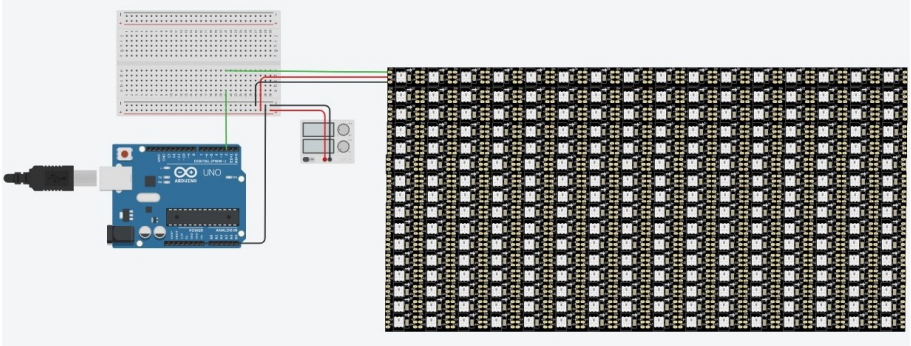
- Al principio de todo el proyecto, se dijo que se iban hacer métodos para crear la matriz, leer la imagen y guardar el archivo, pero ahora en la implementación, fue suficiente con crear los métodos de submuestreo, sobremuestro y modificador, ya que en cada uno de los métodos se recorre la imagen mientras que se guarda en el archivo en forma de matriz al añadir paréntesis. Se implementaron estos métodos de tal forma que hizo varios trabajos a la vez, los cuales originalmente se podrían hacerse en varios métodos, pero se prefirió hacerse así por motivos de funcionalidad y porque tratábamos de hacer el proyecto rápido y simple, ya que estábamos contra el tiempo.

1.4. Estructura del circuito montado.

En tinkercad se creó el circuito, se colocó una fuente conectada a la protoboard, la protoboard y la fuente se conectaron a las 16 tiras NeoPixel de 16 leds cada una, de tal forma que la salida de una tira se conecta a la entrada de la siguiente tira.

Ya para el código, se definió dos macros, una será el pin que está conectada la tira NeoPixel a la protoboard y la otra será la cantidad total de leds. También se creó un arreglo 16x16x3, donde el 16 será las filas y columnas mientras que el 3 serán los espacios de color de los pixeles de nuestra imagen (rojo, verde, azul), no se nos puede olvidar inicializar un contador desde cero.

Con un ciclo se iterará entre las filas y columnas una por una, además de un condicional que nos permitirá aumentar nuestro contador mientras sea menor a la cantidad total de leds, este contador es importante ya que, con el arreglo, permitirá que los leds se enciendan, es decir, el contador representara el número de leds donde nos encontramos y el arreglo, que contiene una matriz con los valores de los pixeles, tendrá el valor del color que nos mostrara cuando los leds se enciendan.



1.5. Problemas presentados durante la implementación.

Unos de los primeros problemas que nos encontramos fue como realizar el submuestreo y el sobremuestreo.

- Al investigar sobre el submuestreo nos dimos cuenta de que se trata de comprimir la imagen para reducir el tamaño, se toman unos pixeles y dependiendo de un factor elegido, se desecha los otros. Con esto se decidió que íbamos a iterar la imagen de 16 en 16, ya que nuestra matriz es de 16x16, pero hubo unos casos en el que no se podía aplicar por las diferentes dimensiones de la imagen, incluso llegamos a utilizar diferentes condicio-

nales para cada caso, pero al final se solucionó de la manera descripta en el ítem de esquema de estructura final de las clases implementadas.

- En el caso del sobremuestreo, fue más difícil. Había varios métodos para sobremuestrear la imagen como la replicación de pixeles, interpolación del vecino cercano, interpolación lineal y la interpolación bilineal, estas fueron las que más encontramos. Nos decidimos por la replicación de pixeles que consiste en copiar un dato en la posición (x,y) de la matriz de la imagen original, pero también miramos la del interpolación del vecino cercano que trata de encontrar el punto más cercano de la imagen original y luego insertar un valor de pixel en ese punto de la imagen. Pero al final utilizamos la replicación de pixeles ya que se mencionaba que era más fácil de usar, y aun así, tuvimos muchos problemas al implementarla de tal forma de que la imagen quedara de dimensiones de 16x16, hubo muchas pruebas y errores, incluso se utilizó hardcoding, todo para que se solucionara nuestros problemas con un condicional más en nuestro código, el cual pasamos por alto después de habernos quedado afectados por muchos pruebas y errores. Se logró implementar de tal manera mencionada en el ítem de esquema de estructura final de las clases implementadas.

2. Manual

Para usar nuestro proyecto de QT se debe tener en cuenta:

1. La imagen que con la deseas trabajar debes guardarla en la carpeta de Mapas, que es donde están todas las imágenes.

2. Nuestro programa recibe archivos JPG y PNG ya que estas fueron las con las que probamos, con los otros tipos de formato se desconoce, pero por la clase QImage se deben de aceptar y el proceso de modificación de dimensiones de la imagen debe de aplicarse para los formatos restantes. Pero podemos asegurar de que con el archivo JPEG no funcionara, debido a que ya se probó y nos resultó que no se guardaba en el archivo txt.

3. Una vez que el usuario pone a correr el programa de QT, en la terminal aparecerá un mensaje que dice “Escriba el nombre de la imagen”, ahí es donde escribirás el nombre de la imagen que desea modificar junto con su formato, por ejemplo Colombia.png, japon.jpg, etc.

4. Una vez hecho el proceso de submuestreo, sobremuestreo o modificador, los datos de los espacios de color de cada pixel se guardan en el archivo txt llamada “Matriz”, este archivo se copiará.

5. Una vez en tinkercad, se pegará los datos del archivo “Matriz” en el arreglo que hemos implementado para almacenar estos datos, este arreglo se encuentra fuera del setup y el loop.

6. Una vez que se han almacenado los datos que representan los espacios de color de cada pixel en nuestro arreglo, lo único que falta es iniciar la simulación para que al final los leds RGB de la tira NeoPixel se enciendan y nos muestre la imagen que fue modificada, en nuestro caso, las banderas.