Práctica 5: Piecewise Transformations

Visión por Computadora

Elaborado por:
José Baltazar Ramírez Rodríguez

Dra María Susana Ávila García

01 de marzo del 2019

I. DESCRIPCIÓN DEL PROBLEMA

La transformación por partes de una imagen, resulta ser un método complementario a los vistos en la práctica anterior, “Transformación Espacial” para mejorar la calidad de la imagen. La principal desventaja de la transformación por partes es que sus especificaciones requieren considerablemente más entradas por parte del usuario. (Rafael C. González, 2007). En este tipo de transformación se trabajará con Contrast Stretching, Intensity-Level Slicing y Bit-Plane Slicing.
Contrast Stretching será utilizado para como su nombre indica para aplicar un constraste a la imagen proporcionada para estos algoritmos. El Contrast Streching es un proceso que expande el rango de niveles de intensidad en una imagen. (Rafael C. González, 2007).
Intensity-Level Slicing puede ser implementado en dos maneras diferentes. De acuerdo a lo requerido se implementará una aproximación para desplegar en un valor (blanco) todos los valores que se encuentren en el rango especificado. El resto será negro.

II. ALGORITMO UTILIZADO PARA RESOLVER EL PROBLEMA

El algoritmo que se utilizó para resolver estos ejercicios consiste en cargar la imagen a una matriz. Esta será nuestra imagen de entrada.

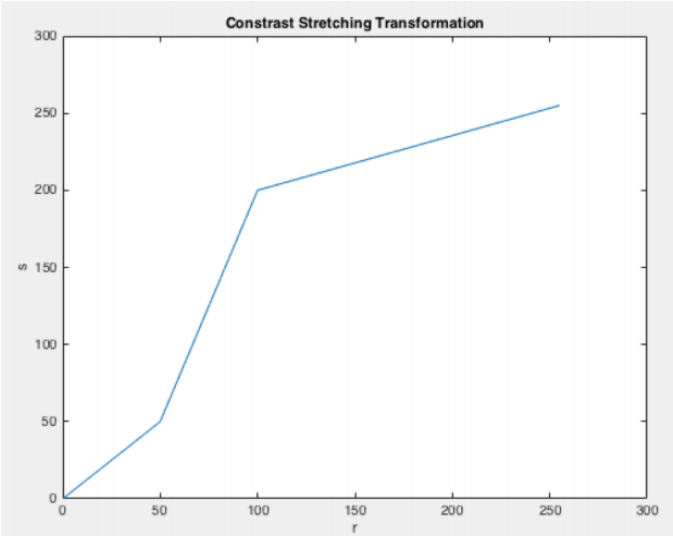
```
//leer la imagen
Mat imagen = imread("C:/Users/HOLA/Desktop/Balta/Visual Studio/HolaMundo/fogata.png");

if (imagen.empty()) // Verificar que se haya cargado la imagen
{
    cout << "No se pudo abrir o encontrar la imagen." << endl;
    system("pause");
    return -1;
}
```

Se comienza por obtener su número de filas y columnas de esta imagen de entrada para construir las nuevas matrices de salida, a las que se aplicará la transformación por partes. Estos pasos se muestran en la siguiente figura

```
int filas = imagen.rows;
int columnas = imagen.cols;
Mat matrizGrises = Mat::zeros(filas, columnas, CV_64FC1);
cvtColor(imagen, matrizGrises, COLOR_BGR2GRAY); //Para convertir la imagen a escala de grises
Mat matrizGrises1 = Mat::zeros(filas, columnas, CV_64FC1);
cvtColor(imagen, matrizGrises1, COLOR_BGR2GRAY);    //Para convertir la imagen a escala de grises
Mat matrizGrises2 = Mat::zeros(filas, columnas, CV_64FC1);
cvtColor(imagen, matrizGrises2, COLOR_BGR2GRAY);    //Para convertir la imagen a escala de grises
```

Se decide convertir estas matrices a escala de grises para trabajar con un solo canal y trabajar de una manera más clara y sencilla.

El **contrast stretching** como se menciona a la introducción del documento, consiste en un proceso que expande el rango de niveles de intensidad en una imagen que se rige con las siguientes fórmulas en las que se calcula la pendiente de cada recta de la gráfica que ilustra la imagen:



Para 0 <= r <= r1

$$s = \frac{S_1}{R_1} * r$$

Para r1 < r y r < =r2

$$s = \left(\frac{S_2 - S_1}{R_2 - R_1}\right) * (r - R_1) + S_1$$

Para r2 < r y r <=255

$$s = \frac{255 - S_2}{255 - R_2} * (r - R_2) + S_2$$

Donde los valores de S_1, S_2, R_1 y S_2 son valores asignados con un valor específico. Para este caso, se asigna a R_1, R_2, S_1 y S_2, 50, 100, 50 y 200 respectivamente como se muestra en la siguiente imagen:

```
/*Valores para realizar el contrast stretching*/
double r1 = 50;
double r2 = 100;
double s1 = 50;
double s2 = 200;

double minimo = 0;
double maximo = 255;
```

Posteriormente se recorre la matriz de acuerdo con sus filas y columnas, y se define un Scalar para obtener la intensidad de cada píxel. Se extrae la intensidad de cada píxel y se asigna a r. Lo siguiente es comprobar una de las tres condiciones para definir qué fórmula aplicar para obtener la pendiente de acuerdo con la intensidad de r.

```
//Para Contrast Stretching
for (int x = 0; x < matrizGrises.rows; x++) {
    for (int y = 0; y < matrizGrises.cols; y++) {
        Scalar intensity = matrizGrises.at<uchar>(x, y);
        double r = intensity.val[0];
        if (0 <= r && r <= r1) {
            s = (s1 / r1)*r;
            matrizGrises.at<uchar>(x, y) = s;
        }
        else if (r1 < r && r <= r2) {
            s = ((s2 - s1) / (r2 - r1)) * (r - r1) + s1;
            matrizGrises.at<uchar>(x, y) = s;
        }
        else if (r2 < r && r <=255){
            s = ((255 - s2) / (255 - r2))*(r - r2) + s2;
            matrizGrises.at<uchar>(x, y) = s;
        }

    }
}
```

El resultado que se obtiene es el siguiente:

La siguiente función fue **intensity-level slicing**, puede ser implementada en dos maneras diferentes. De acuerdo con lo requerido se implementará una aproximación para desplegar en un valor (blanco) todos los valores que se encuentren en el rango especificado. El resto será negro. El rango que se definió para este ejercicio fue de A = 15 y B = 50. Lo que esté fuera de este rango, será negro. Lo que esté dentro será blanco.

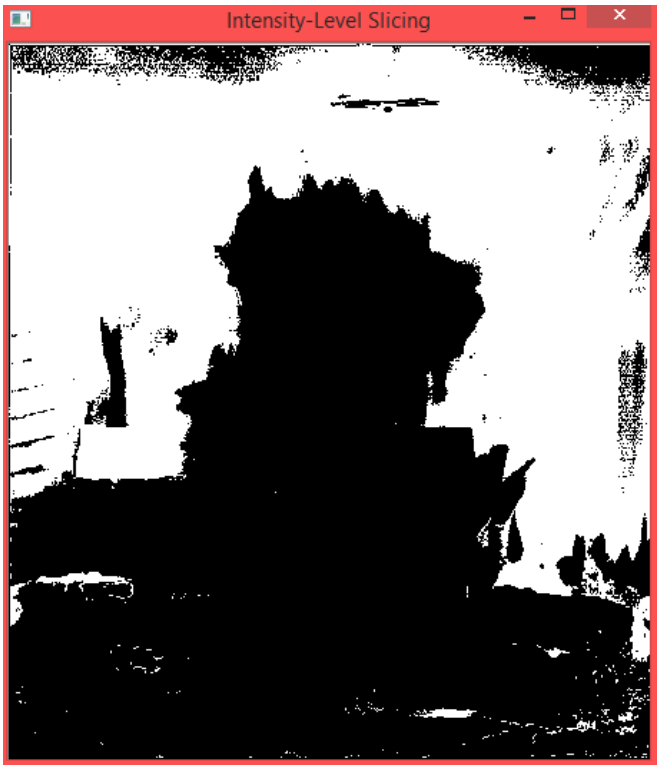
Se define entonces el rango

```
/*----------------------
double A = 15;
double B = 50;
```

Después igual que en los demás procesos, se recorre la matriz de la imagen en escala de grises de acuerdo con sus dimensiones para obtener la intensidad r de cada píxel. Y se verifica que esté nuestra intensidad en el rango de valores definido. Se asigna el valor correspondiente a la nueva matriz de salida.

```
for (int x = 0; x < matrizGrises1.rows; x++) {
    for (int y = 0; y < matrizGrises1.cols; y++) {
        Scalar intensity = matrizGrises1.at<uchar>(x, y);
        double r = intensity.val[0];
        if (A < r && r < B) {   //¿Se encuentra la intensidad del pixel dentro del rango de A y B?
            matrizGrises1.at<uchar>(x, y) = 255;
        }
        else {
            matrizGrises1.at<uchar>(x, y) = 0;
        }
    }
}
```

La salida que se obtuvo fue la siguiente:



Se sigue trabajando para lograr el algoritmo de bit-plane slicing.