Universidad de Oviedo

Escuela de Ingeniería Informática
Universidad de Oviedo

Arquitectura de Computadores

Trabajo en grupo

PL1-A

Jorge Gota Ortín UO301023

Javier Ortín Rodenas UO299855

Andrés Fernández-Junquera Fernández UO302086

Alejandro Jarillo Pineda UO302394

Índice

1. Algoritmo a aplicar a la imagen

1.1. Algoritmo matemático

En nuestro caso, el algoritmo a implementar es el número 3. Tiene como entrada una única imagen y genera como salida una versión de la misma en blanco y negro. Sean (R, G, B) las componentes de un píxel de la entrada, las componentes (R', G', B') del píxel equivalente en la imagen de salida vienen dadas por:

$$R' = G' = B' = 255 - (0.3 \cdot R + 0.59 \cdot G + 0.11 \cdot B)$$

Se pasa a escala de grises con la siguiente media ponderada:

$$L = 0.3 \cdot R + 0.59 \cdot G + 0.11 \cdot B$$

A continuación, se invierte al hacer $R' = G' = B' = 255 - L$, de ahí el nombre de "inversión en blanco y negro".

Matemáticamente, los valores de la imagen destino deberían estar comprendidos siempre entre 0 y 255. Debido a la sencillez de este algoritmo, no será sencillo saturar, pues el resultado de las operaciones tendrá un valor adecuado siempre.

1.2. Funcionamiento general e implementación monohilo

Comencemos analizando la función `main` y el funcionamiento general del programa:

```
int main() {
    // 1 -- Load source image
    CImg<data_t> srcImage;
    if(loadSourceImage(srcImage)) {
        return IMG_ERROR;
    }

    // 2 -- Allocate memory and prepare filter structures
    filter_args_t filter_args;
    data_t *pDstImage = NULL;
    unsigned int width = 0, height = 0, nComp = 0;

    if(prepareFilter(srcImage, filter_args, pDstImage, width, height, nComp)) {
        return MEMORY_ERROR;
    }

    // 3 -- Measure execution time over REPS iterations
    double elapsedTimeS = 0.0;
    if(measureExecutionTime(filter_args, elapsedTimeS)) {
        free(pDstImage); // cleanup
        return TIMING_ERROR;
    }
    printf("\nElapsed time: %f s\n", elapsedTimeS);

    // 4 -- Display and save output image
    showAndSaveResult(pDstImage, width, height, nComp);

    // 5 -- Free memory
    free(pDstImage);

    return 0;
}
```

El funcionamiento general del programa es el mismo para las tres implementaciones:

En primer lugar, cargamos la imagen teniendo en cuenta la posibilidad de que esta pueda no existir y manejando el error en tal caso. A continuación, iniciamos las variables necesarias, deteniendo la ejecución si no fuese posible asignar memoria adecuadamente. Una vez hecho esto, medimos el tiempo que tarda en ejecutarse el algoritmo 100 veces (para que tarde el tiempo mínimo deseado), manejando los errores de cronometrado en caso de haberlos. Finalmente, se muestra el tiempo transcurrido así como la imagen destino.

Veamos ahora cómo se estructura el algoritmo del filtro:

```
// Filter function: converts RGB to grayscale and applies the inversion
void filter (filter_args_t args) {
    for (uint i = 0; i < args.pixelCount; i++) {
        // Compute grayscale using weighted sum of RGB components
        data_t value = MAX_VAL - (R_COEF * args.pRsrc[i] + G_COEF* args.pGsrc[i] + B_COEF * args.pBsrc[i]);
        // Write the same grayscale value to each output channel
        args.pRdst[i] = value;
        args.pGdst[i] = value;
        args.pBdst[i] = value;
    }

}
```

El algoritmo recorre los píxeles de la imagen, aplicando para cada uno de ellos el algoritmo matemático visto.