DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico II

Procesamiento de imágenes (SIMD)

Organización del Computador II

Primer Cuatrimestre de 2019

Integrante	LU	Correo electrónico
Ivo Pajor	460/19	ivo_pajor@hotmail.com
Luciana Gorosito	577/18	lugorosito0@gmail.com
Laureano Muñiz	498/19	lau2000m@hotmail.com

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

http://www.fcen.uba.ar

Resumen

En el presente trabajo se describe la problemática de ...

Índice

1. Introducción

El objetivo de este Trabajo Práctico es analizar y comprender el modelo de procesamiento SIMD *(Single Instruction, Multiple Data)* y su relación con la microarquitectura del procesador, mediante la implementación en lenguaje ensamblador de cuatro filtros gráficos: «*Imagen Fantasma*», «*Color Bordes*» y «*Reforzar Brillo*».

Más detalladamente, el filtro «*Imagen Fantasma*» combina una imagen original con su versión en escala de grises y del doble de tamaño, generando así un efecto de imagen fantasma sobre la imagen destino. El filtro «*Color Bordes*» detecta los bordes de una imagen y el filtro «*Reforzar Brillo*» modifica el brillo de una imagen, aumentándolo en el caso de que supere al valor del parámetro *umbralSup* y disminuyéndolo en el caso de que esté por debajo del valor del parámetro *umbralInf*.

Las implementaciones de estos filtros se realizaron utilizando el set de instrucciones **SSE** y técnicas de programación vectorial, que permitieron procesar en paralelo de 2 a 4 píxeles, dependiendo del filtro. Posterior a la implementación se realizó un análisis de rendimiento en comparación a las implementaciones en lenguaje C, provistas por la cátedra. Además, se diseñaron dos experimentos motivados en entender las posibles causas de la variación del rendimiento y la limitación de la performance de los algoritmos en el procesamiento de imágenes, que serán detallados en las secciones siguientes.

2. Desarrrollo

2.1. Implementaciones en ASM

En esta sección se incluyen las descripciones de las funciones implementadas en lenguaje ensamblador.

2.1.1. Imagen Fantasma

En primer lugar, se definieron en la sección *section .data* los offset necesarios para obtener los parámetros que fueron pasados por la pila (offset_x y offset_y) y las máscaras a utilizar en la operatoria con los pixeles, estas son:

- *transparencia*: máscara utilizada para borrar la componente de transparencia del pixel, la misma será luego restaurada al terminar el ciclo usando la máscara *sumar*.
- *green*: máscara utilizada para extraer la componente g del pixel, ya que es la única componente que duplica su valor al calcularse el brillo.
- *dividir*: máscara utilizada para completar el cálculo del brillo y la conversión a escala de grises en un mismo paso.
- *multiplicar*: máscara utilizada para multiplicar por 0.9 en la conversión a escala de grises.

  Previo al inicio del ciclo, se calculó el offset en bytes correspondiente al pixel de la posición src[jj][ii], del cual se obtiene el brillo. La idea principal de este algoritmo es recorrer a la imagen en submatrices de 2x2, levantando de memoria en los registros *xmm1, xmm2, xmm3 y xmm4* a los pixeles que les corresponde el mismo valor de brillo, para así poder procesarlos en paralelo.

  Luego, en cada iteración del ciclo se realizaron los siguientes pasos:

  1. Cálculo de brillo.
     Se levantó el pixel src[jj][ii] en el registro *xmm0*, extendiendo cada componente a dword, para poder relizar cálculos sin perder precisión. Además, como el cálculo del brillo solo necesita de las componentes RGB del pixel, antes de proceder se seteó en cero a la componente A, aplicando la máscara de *transparencia* antes mencionada. Seguido a esto, se extrajo en el registro *xmm13* la componente G del pixel, aplicando la máscara *green*. Se sumaron los registros *xmm0* y *xmm13*, y se realizaron dos sumas horizontales consecutivas del registro *xmm0*, obteniendose en este registro el resultado de la operación:

$$b * 4 = src[jj][ii].r + 2 * src[jj][ii].g + src[jj][ii].b \tag{1}$$

Para completar con el cálculo del brillo y con una parte de la conversión a escala de grises, se convirtió el dato a punto flotante de 32 bits y se lo dividió por 8, usando la máscara *dividir*.

2.

2.1.2. Color Bordes

2.1.3. Reforzar Brillo

2.2. Comparación entre implementaciones en ASM y C

2.3. Diseño experimental

```
struct Pepe {

    ...

};
```

3. Resultados

4. Conclusión