DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Nº 2

Segundo cuatrimestre de 2015 — Organización del Computador II

Grupo: Smelly Cat

Integrante	LU	Correo electrónico
Frizzo, Franco	013/14	francofrizzo@gmail.com
Martínez, Manuela	160/14	martinez.manuela.22@gmail.com
Rabinowicz, Lucía	105/14	lu.rabinowicz@gmail.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

Índice

1. Introducción

En el presente trabajo, aplicamos el modelo de programación vectorial SIMD (*Single Instruction, Multiple Data*) para la implementación de filtros para el procesamiento de imágenes. Más precisamente, llevamos a cabo la implementación de los siguientes dos filtros:

- *Diferencia*, que recibe como entrada dos imágenes y devuelve como resultado otra imagen que indica dónde difieren las dos primeras.
- *Blur gaussiano*, que suaviza la imagen reemplazando cada píxel por un promedio de los píxeles circundantes, ponderado según una función gaussiana.

La elaboración del trabajo se dividió en dos etapas. En primer lugar, implementamos ambos filtros tanto en lenguaje C como en lenguaje ensamblador para la arquitectura x86 de Intel. En este último caso, se utilizaron las instrucciones SSE de dicha arquitectura, que aprovechan el ya mencionado modelo SIMD para procesar datos en forma paralela.

Una vez realizadas estas implementaciones, las sometimos a un proceso de comparación para extraer conclusiones acerca de su rendimiento. Con este fin, experimentamos con variaciones tanto en los datos de entrada como en detalles implementativos de los mismos algoritmos. De esta manera pudimos recopilar datos sobre el comportamiento de cada implementación, y contrastar estos resultados con diversas hipótesis previamente elaboradas.

2. Desarrollo

2.1. Consideraciones generales

Las imagenes que utilizaremos como entrada y salida de los algoritmos a implementar serán matrices de píxeles. Cada uno de estos píxeles estará representado por cuatro enteros sin signo de 8 bits de profundidad (es decir, en el rango $[0, 256)$), que representarán, respectivamente, los valores de los colores azul (b), verde (g) y rojo (r), y la transparencia (a).

Usaremos la notación $I_{x,y}$ para referirnos al píxel ubicado en la fila x y la columna y de la imagen I, y la notación $I^k_{x,y}$ para hacer referencia al valor de la componente k de este píxel, donde $k \in \{\mathsf{b}, \mathsf{g}, \mathsf{r}, \mathsf{a}\}$.

2.2. Diferencia de imágenes

Este filtro recibe dos imágenes como entrada y devuelve como salida una tercera imagen que muestra, en cada píxel, la diferencia entre los píxeles correspondientes de las imágenes de entrada, ignorando la componente a. Más especificamente, si I_1 e I_2 son las imágenes de entrada y O es la imagen de salida, entonces:

$$O^k_{x,y} = \begin{cases} \max\limits_{k \in \{\mathsf{b},\mathsf{g},\mathsf{r}\}} \left(\left| {I_1}^k_{x,y} - {I_2}^k_{x,y} \right| \right) & \text{si } k \in \{\mathsf{b}, \mathsf{g}, \mathsf{r}\} \\ 255 & \text{si } k = \mathsf{a} \end{cases}$$

2.2.1. Implementación en lenguaje C

2.2.2. Implementación en lenguaje ensamblador

Dado que los registros `XMM` son de 16 bytes, se los puede utilizar para procesar 4 píxeles de las imágenes en paralelo, reduciendo la cantidad de iteraciones del algoritmo y, particularmente, de accesos a memoria necesarios para completar el algoritmo.

Nuestra implementación del filtro consiste principalmente de un ciclo que itera sobre la imagen. Al comienzo de cada ejecución, se copian 4 píxeles de I_1 al registro `XMM0`, y los correspondientes 4 píxeles de I_2 a `XMM1`.

`XMM0:`	A_4^{b}	A_4^{g}	A_4^{r}	A_4^{a}	A_3^{b}	$\cdots$	A_1^{a}
`XMM1:`	B_4^{b}	B_4^{g}	B_4^{r}	B_4^{a}	B_3^{b}	$\cdots$	B_1^{a}

El paso siguiente es, para cada una de las componentes de estos píxeles, calcular el valor absoluto de la diferencia entre ambas imágenes. Para realizar esto, realizamos las dos restas, es decir, `XMM0` $-$ `XMM1` y `XMM1` $-$ `XMM0`. En las posiciones donde el valor contenido en `XMM0` sea mayor que el de `XMM1`, será válido el resultado de la primera operación, mientras que en las demás posiciones deberemos quedarnos con el segundo resultado.

Para seleccionar con cuál de los dos resultados quedarnos, utilizamos una máscara que obtenemos comparando los valores de `XMM0` y `XMM1`. Aquí nos encontramos con un problema, ya que debemos comparar enteros sin signo, y SSE no brinda instrucciones para hacer esto. Es por eso que recurrimos a desempaquetar los números y considerarlos como enteros con signo de dos bytes, que sí podemos comparar.

En primer lugar, usando la instrucción `PUNPCKLBW`, desempaquetamos las partes bajas de `XMM0` y `XMM1` en los registros `XMM2` y `XMM3`.

`XMM2:`	0	A_2^{b}	0	A_2^{g}	0	A_2^{r}	$\cdots$	0	A_1^{a}
`XMM3:`	0	B_2^{b}	0	B_2^{g}	0	B_2^{r}	$\cdots$	0	B_1^{a}

Luego los comparamos mediante la instrucción `PCMPGTW`, almacenando la máscara resultante en `XMM2`. A continuación hacemos lo mismo con las partes altas, usando la instrucción `PUNPCKHBW` para desempaquetar y guardando el resultado en `XMM3`. Por último, la instrucción `PACKSSWB XMM2, XMM3` nos permite obtener en `XMM2` la máscara que buscábamos, ya que transforma los valores de 2 a 1 byte usando saturación.

2.3. Blur gaussiano

Este filtro recibe una imagen como entrada y devuelve como salida el resultado de aplicarle una convolución[1] con una función gaussiana, que dependerá de un parámetro σ que podrá ser modificado. Dada la naturaleza del problema, trabajaremos con una convolución discreta en dos dimensiones, y como nuestro poder de cómputo es limitado, procesaremos solo una vecindad acotada de cada píxel, cuyo radio quedará determinado por un parámetro configurable r. En

[1] Dadas dos funciones f y g, una *convolución* $f * g$ es una operación que las transforma en una tercera función: $(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)\,d\tau$ en el caso continuo, $(f * g)_n = \sum_{k=-\infty}^{+\infty} f_k g_{n-k}$ $(n, k \in \mathbb{Z})$ en el caso discreto.

definitiva, el resultado del filtro será

$$O^k_{x,y} = \begin{cases} \sum_{i=-r}^{r} \sum_{j=-r}^{r} O^k_{x+i,y+j} K_{r-i,r-j} & \text{si } k \in \{\mathsf{b}, \mathsf{g}, \mathsf{r}\} \\ 255 & \text{si } k = \mathsf{a} \end{cases}$$

donde K es la matriz o *kernel* de la convolución, con

$$K_{i,j} = \frac{1}{2\pi\sigma^2} e^{-\frac{(r-i)^2+(r-j)^2}{2\sigma^2}} \qquad \text{para todo } 0 \leq i, j \leq 2r$$

2.3.1. Implementación en lenguaje C

2.3.2. Implementación en lenguaje ensamblador