



**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

<b>1. Introducción</b>	<b>3</b>
<b>2. Desarrollo</b>	<b>3</b>
2.1. Implementaciones en ASM . . . . .	3
2.1.1. Imagen Fantasma . . . . .	3
2.1.2. Color Bordes . . . . .	4
2.1.3. Reforzar Brillo . . . . .	6
2.2. Comparación entre implementaciones en ASM y C . . . . .	6
2.3. Diseño experimental . . . . .	6
<b>3. Resultados</b>	<b>6</b>
<b>4. Conclusión</b>	<b>6</b>

# 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. Desarrollo

### 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 píxeles, estas son:

- *transparencia*: máscara utilizada para borrar la componente de transparencia del píxel, la misma fue luego restaurada al terminar el ciclo usando la máscara *sumar*.
- *green*: máscara utilizada para extraer la componente G del píxel, ya que es la única componente que duplica su valor al calcularse el brillo.
- *multiplicar*: máscara utilizada para multiplicar por 29, ya que  $\frac{29}{32} \cong 0.9$ , en la conversión a escala de grises.
- *sumar*: máscara utilizada para arreglar las transparencias de los píxeles al finalizar los cálculos.

Previo al inicio del ciclo, se calculó el offset en bytes correspondiente al píxel de la posición (jj,ii) de la imagen Fuente, 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* y *xmm2* los píxeles que les corresponde el mismo valor de brillo, para así poder procesarlos en paralelo.

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

#### 1. Cálculo de brillo

Se levanta el píxel *src[jj][ii]* en el registro *xmm0*, extendiendo cada componente a dword, para poder realizar cálculos sin perder precisión. Además, como el cálculo del brillo solo necesita de las componentes RGB del píxel, antes de proceder se setea en cero a la componente A, aplicando la máscara de *transparencia* antes mencionada. Seguido a esto, se extrae en el registro *xmm13* la componente G del píxel, aplicando la máscara *green*. Se sumaron los registros *xmm0* y *xmm13*, y se realizaron dos sumas horizontales consecutivas del registro *xmm0*, obteniéndose en las cuatro *dwords* de este registro el resultado de la operación:

$$4B = src[jj][ii].r + 2 * src[jj][ii].g + src[jj][ii].b \quad (1)$$

Para finalizar el cálculo de brillo se transforman las *dwords* a *words* saturando de manera *unsigned*. Por último, resta dividir por 4 pero como en la conversión a escala de grises se utiliza el valor de  $\frac{B}{2}$  se dividen por 8 las *words* del registro *xmm0* utilizando operaciones de shifteo hacia derecha.

## 2. Conversión a escala de grises

Se levantan los píxeles de la submatriz de 2x2 en los registros *xmm1* y *xmm2*, extendiendo sin signo de *byte* a *word*. Se multiplican los valores de las componentes por 29 utilizando la máscara *multiplicar*. Luego, se shiftean las componentes hacia derecha 5 bits, lo cual equivale a dividir por 32 tomando piso. Por último, se suma *xmm0* a ambos registros de forma saturada para luego empaquetarlos de la misma manera de *words* a *bytes*.

Antes de cargar los resultados en la imagen Destino, se arreglan las transparencias de los píxeles con la máscara *sumar*.

### 2.1.2. Color Bordes

Se definieron en la sección *section .data* las siguientes máscaras:

- *blanco*: máscara utilizada para colorear de blanco los bordes de la imagen destino.
- *transparencias1*: máscara utilizada para restaurar la transparencia de un píxel.
- *transparencias2*: máscara utilizada para filtrar la componente de transparencia de un píxel.

y, antes de comenzar el ciclo, fueron guardadas en los registros *xmm15*, *xmm14* y *xmm13*, respectivamente.

El algoritmo consta de dos ciclos cortos ("*.bordesPrimeraFila*" y "*.bordesUltimaFila*") que arman los bordes blancos horizontales de la imagen destino, un "*preciclo*", que actualiza las variables antes de ingresar a los otros dos ciclos ("*.cicloFilas*" y "*.cicloColumnas*") que recorren la imagen Fuente realizando la operatoria de detección de bordes y armando los bordes verticales blancos de la imagen destino. A continuación se detallan las operaciones que se realizan en cada parte.

#### ■ *.bordesPrimeraFila*

En cada iteración del ciclo se escriben 16 bytes de la imagen destino con el valor de 255( correspondiente al blanco), usando la máscara previamente guardada en el registro *xmm15*. Como se escriben cuatro píxeles de manera simultánea en cada iteración, el registro contador *edx* fue inicializado con el valor de  $\# \frac{columnas}{4}$  y, para compensar el factor multiplicativo que le falta a la escala, el registro índice *rdi* fue incrementado en 2 unidades cada iteración.

#### ■ *.preciclo*

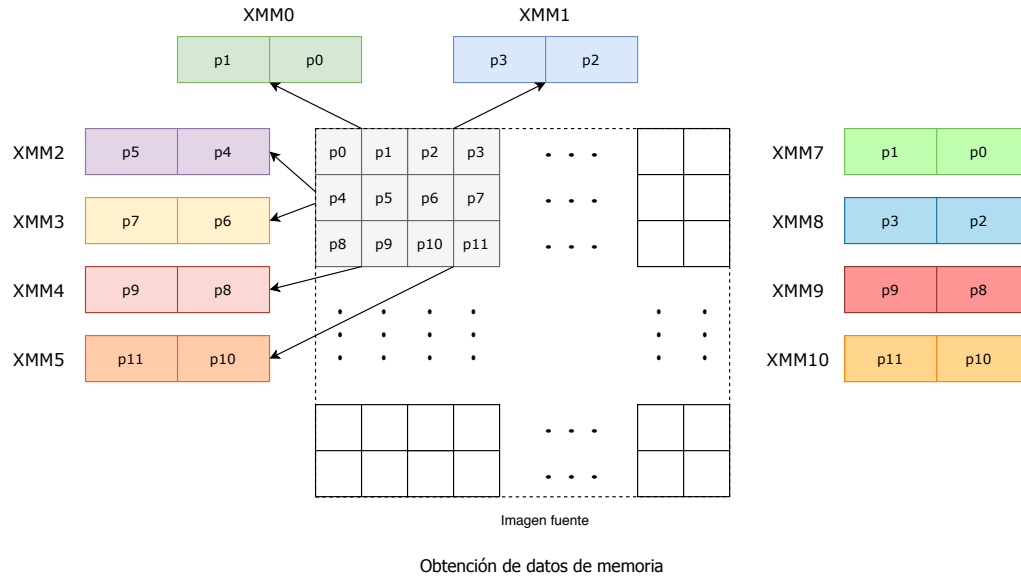
Se guarda en *rdi* la dirección efectiva a la posición (1,1) de la imagen fuente y se guarda en *rsi* la dirección efectiva a la posición (1,0) de la imagen destino. Además, se decrementa el valor del registro contador de filas *ecx*, porque la primera fila ya fue pintada de blanco en el primer ciclo de bordes.

#### ■ *.cicloFila* y *.cicloColumnas*

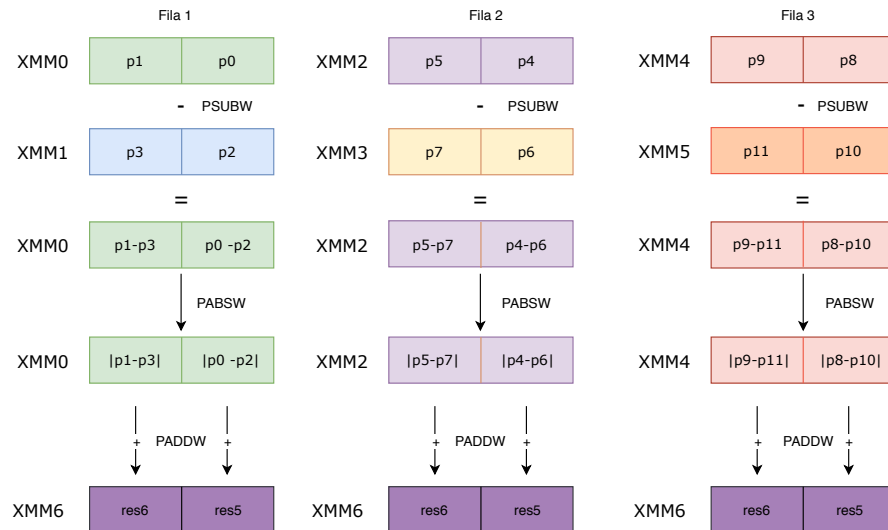
El ciclo externo (*.cicloFilas*) recorre a las imágenes fuente y destino por filas, empezando desde la primera. La guarda del ciclo compara a *edx* (registro contador de filas) con 1, por el margen de un píxel que debe dejarse y que es pintando de blanco en la imagen destino durante el último ciclo de bordes. Cada vez que se ingresa al ciclo de las filas, se pinta de blanco el primer píxel de la imagen destino y se avanza un píxel en la misma. De esta manera se va generando el margen vertical izquierdo esa imagen. Además, antes de ingresar al ciclo interno, se resetea el contador de columnas *eax*. El valor del contador es  $\# \frac{columnas-2}{2}$ , ya que los píxeles de la primera y última columna de la imagen fuente no se procesan y el resto de la operatoria se hace procesando de a 2 píxeles.

En cada iteración del ciclo interno (*.cicloColumnas*) se levantan de memoria los seis píxeles necesarios para procesar los píxeles de la posición [*rdi*] y [*rdi* + 4]. Se levantan dos píxeles por registro,

extendiendo las componentes de cada píxel de byte a word, quedando así 2 píxeles por registro xmm. Se limpia el registro xmm6, que es usado como registro acumulador y se reservan en otros registros los píxeles necesarios para calcular las diferencias verticales.

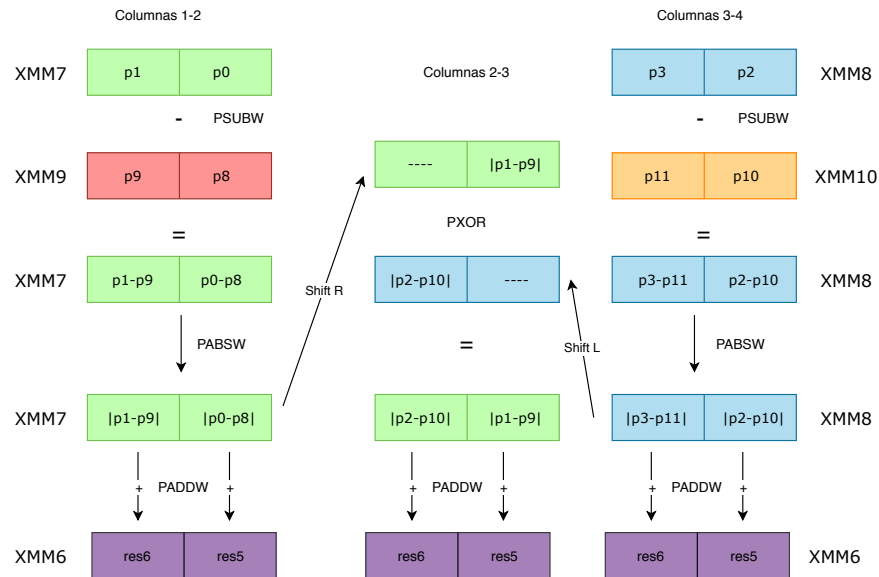


Para calcular las diferencias horizontales, se restan los registros correspondientes a una misma fila y luego se toma valor absoluto. El resultado de cada diferencia calculada se suma al registro xmm6, acumulándose en la parte alta la suma de diferencias horizontales para el píxel de la posición  $[rdi + 4]$  (píxel 6 en el ejemplo) y en la parte baja la suma de diferencias horizontales para el píxel de la posición  $[rdi]$  (píxel 5 en el ejemplo).



Para calcular las diferencias verticales, se restan los registros correspondientes al mismo par de columnas, se toma valor absoluto y se suman a xmm6. En este caso, las diferencias verticales de las columnas 2 y 3 deben ser sumadas tanto a la parte alta como a la parte baja del registro xmm6, por lo que es necesario reordenar los resultados para sumar los valores que faltan.

El siguiente esquema muestra gráficamente la operatoria realizada para obtener las sumas de las diferencias verticales:



Luego de acumular en xmm6 las sumas de las diferencias horizontales y verticales, se empaqueta el registro para convertir devuelta a byte y se arreglan las transparencias usando las máscaras destinadas a esto. Hecho esto, se mueven los resultados de los dos píxeles procesados a la imagen destino y se actualizan los índices sumando 8 bytes a rdi y a rsi.

Antes de avanzar de fila, se pinta de blanco el último pixel de la matriz destino, de esta manera se arma el margen vertical derecho de la imagen destino.

#### ■ .bordesUltimaFila

Este ciclo realiza la misma operatoria que .bordesPrimeraFila, pero usa como registro contador a **r10d** y como registro índice a **edx**, que terminó con el valor 0 luego del primer ciclo bordes.

#### 2.1.3. Reforzar Brillo

### 2.2. Comparación entre implementaciones en ASM y C

#### 2.3. Diseño experimental

## 3. Resultados

## 4. Conclusión