

# Trabajo Práctico 2

Organización del Computador II

Primer cuatrimestre 2015

## 1. Introducción

En este trabajo práctico se busca una primera aproximación al procesamiento con instrucciones que operan con mutiples datos (SIMD). El objetivo es conocer y comprender los principios de la programación con dichas instrucciones. La familia de procesadores x86-64 de Intel posee una serie de extensiones para operaciones SIMD. En un comienzo, éstas se denominaron MMX (MultiMedia eXtension), luego SSE (Streaming SIMD Extensions) y por último AVX (Advanced Vector Extensions).

En este trabajo práctico deberán desarrollar ciertas funciones utilizando el conjunto de instrucciones de SSE. Las funciones corresponden a una serie de filtros para el procesamiento de imágenes en formato BMP. Dichas imágenes serán cargadas mediante una biblioteca de funciones provista por la catedra.

El trabajo práctico consiste en desarrollar implementaciones distintas de cada filtro y luego evaluar su rendimiento. La evaluación debe ser un análisis exhaustivo de las propiedades del código ejecutado y de las circunstancias que justifican por qué una implementación funciona mejor (o peor) que otra.

## 2. Filtros

Una imagen BMP es una matriz de pixeles. Cada pixel está determinado por cuatro bytes: A = Transparencia, R = Rojo, G = Verde, B = Blue (o sea: el tamaño de cada pixel es 4 bytes).

Para la descripción de los filtros se considerará una imagen de tamaño  $m \times n$  donde  $j$  es un iterador para filas,  $i$  un iterador para columnas y  $k$  es un iterador para componenetes de color. La notación utilizada será de la forma:  $\mathbf{m}[j][i][k]$ , donde  $\mathbf{m}$  es la imagen en cuestión.

El trabajo consistirá en realizar dos implementaciones distintas de cada uno de los filtros presentados a continuación.

### 2.1. Blur

*Blur* es un filtro que suaviza una imagen. Consiste en asignarle a cada pixel el promedio (media aritmética) con sus pixeles vecinos. Es decir:

$$\mathbf{m}[j][i][k] = (\mathbf{m}[j-1][i-1][k] + \mathbf{m}[j-1][i][k] + \mathbf{m}[j-1][i+1][k] + \mathbf{m}[j][i-1][k] + \mathbf{m}[j][i][k] + \mathbf{m}[j][i+1][k] + \mathbf{m}[j+1][i-1][k] + \mathbf{m}[j+1][i][k] + \mathbf{m}[j+1][i+1][k]) / 9;$$

En el caso de los bordes la expresión anterior no está definida; en esos casos se deja el mismo valor en el pixel.



Figura 1: Filtro Blur sobre la imagen Lena

### 2.1.1. Implementación 1

Procesar en cada iteración el valor de un sólo pixel. Es decir, cargar en registros toda la matriz correspondiente a un pixel (el pixel y sus vecinos) como muestra la figura 2a y luego promediar los valores según su canal (rojo, verde, azul).

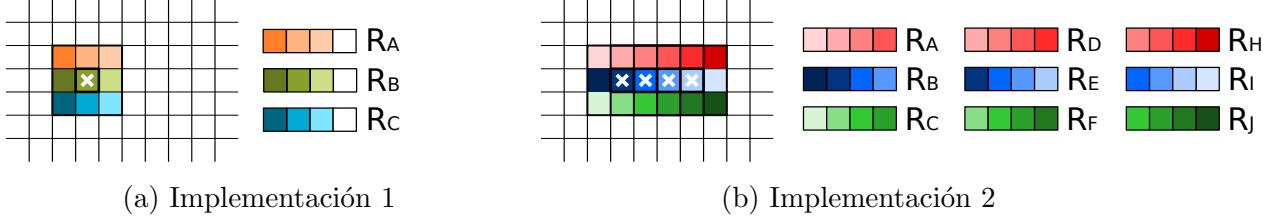


Figura 2: Implementaciones del Filtro Blur

### 2.1.2. Implementación 2

Procesar de a cuatro pixeles. Cargar en registros las matrices correspondientes a cuatro pixeles simultáneamente según indica la figura 2b. Operar con todas las matrices al mismo tiempo y calcular los resultados de 4 pixeles en simultaneo.

## 2.2. Merge

*Merge* es un procedimiento por el cual se funden o mezclan dos imágenes. La operación toma dos imágenes de entrada y consiste en realizar un promedio ponderado entre los pixeles de las dos imágenes. La ponderación está dada por el parámetro `value` (un número en punto flotante, tal que  $0 \leq \text{value} \leq 1$ ).

La formula que se aplica entre cada par de pixeles correspondientes de las imágenes `m1` y `m2` es,

```
m1[j][i][k] = value * m1[j][i][k] + (1-value) * m2[j][i][k];
```

Donde k itera sobre las componentes RGB, dejando intacta la componente A de transparencia.

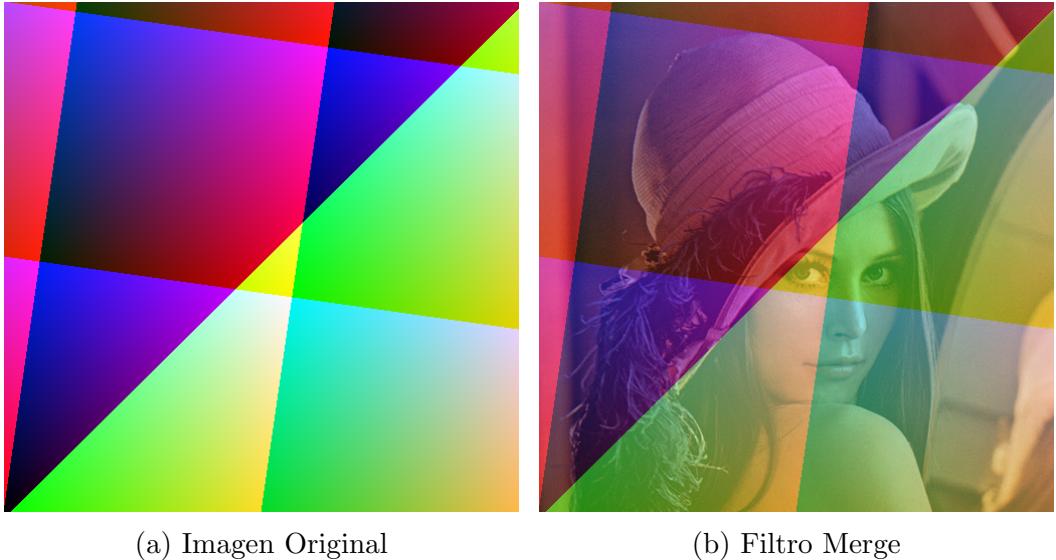


Figura 3: Merge entre la imagen Lena y Colores. Value 0,5

### 2.2.1. Implementación 1

Implementar el filtro realizando operaciones en **punto flotante** procesando la mayor cantidad de pixeles posibles por iteración.

### 2.2.2. Implementación 2

Implementar el filtro realizando operaciones en **enteros** procesando la mayor cantidad de pixeles posibles por iteración.

## 2.3. HSL

Este filtro consta de tres etapas, *RGBtoHSL*, *Suma* y *HSLtoRGB*. La primera consiste en un cambio del espacio de color de los pixeles, desde RGB a HSL. Luego se procesan los pixeles sumándole valores a cada una de las componentes (los valores son tomados por parámetro). Por último, se realiza el pasaje del espacio HSL a RGB (operación inversa a la primera).

El espacio RGB está dado por un cubo donde cada componente corresponde a la intensidad de uno de los colores primarios de la luz (rojo, verde, azul). En la figura 4b se muestra una representación gráfica del espacio de color.

El espacio HSL está dado por tres componentes: Hue (color o matiz), Saturation (Saturación) y Lightness (Luminosidad). Este espacio se representa gráficamente como un bicono (fig. 5). Notemos que un punto en un bicono está determinado por tres valores: la altura de la circunferencia a la que pertenece, el radio de dicha circunferencia y el ángulo dentro de la circunferencia. El color corresponde al ángulo (varía de 0 a 360 grados), la saturación al radio y la luminosidad a la altura.

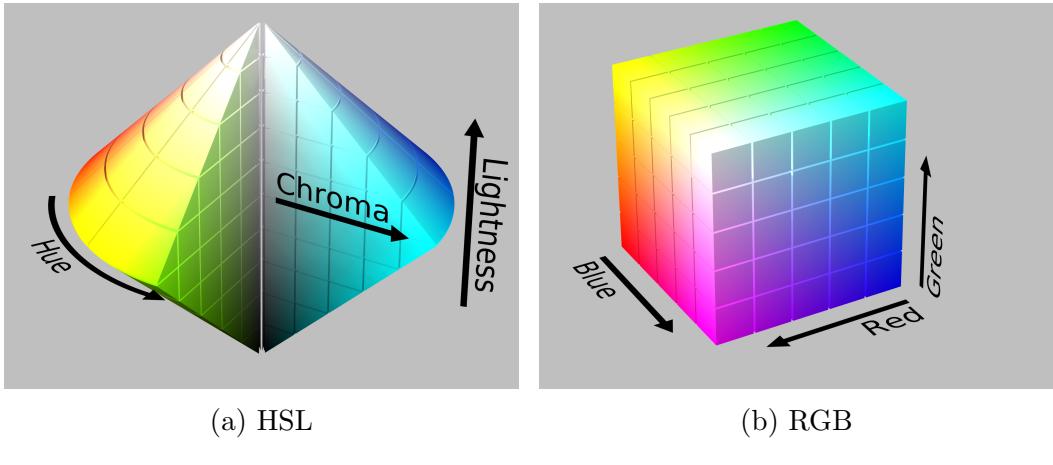


Figura 4: Espacios de color

Transformar del espacio RGB al HSL no es lineal. Las fórmulas para convertir pixeles entre RGB y HSL son las siguientes (notar que los pixeles transformados a HSL son números en punto flotante):

#### ■ Convertir de RGB a HSL

- Entrada: r, g y b (tres enteros de 8 bits)
- Salida: h, s y l (tres números en punto flotante de 32bits)

a) Cálculo de máximos y mínimos

```
cmax = max(b,g,r)
cmin = min(b,g,r)
d = cmax - cmin
```

b) Cálculo de H

```
if ( cmax == cmin ) h = 0
else if ( cmax == r ) h = 60 * ( (g-b)/d + 6 )
else if ( cmax == g ) h = 60 * ( (b-r)/d + 2 )
else if ( cmax == b ) h = 60 * ( (r-g)/d + 4 )
if ( h >= 360 ) h = h - 360
```

c) Cálculo de L

```
l = ( cmax + cmin ) / 510
```

d) Cálculo de S

```
if ( cmax == cmin )
    s = 0
else
    s = d / ( 1 - fabs( 2*l - 1 ) ) / 255.0001f;
```

Nota: la ultima operación en el cálculo de s se debe realizar contra 255.0001f como indica el algoritmo. Esto evita la propagación de errores en el calculo impidiendo que s supere 1.

#### ■ Convertir de HSL a RGB

- Entrada:  $h$ ,  $s$  y  $l$  (tres números en punto flotante de 32bits)
- Salida:  $r$ ,  $g$  y  $b$  (tres enteros de 8 bits)

a) Cálculo de  $c$ ,  $x$  y  $m$

```
c = ( 1 - fabs( 2*l - 1 ) ) * s;
x = c * ( 1 - fabs( fmod( h/60, 2 ) - 1 ) )
m = l - c / 2
```

b) Cálculo de RGB

```
if( 0<=h and h<60 ) r=c b=x g=0
else if( 60<=h and h<120 ) r=x b=c g=0
else if( 120<=h and h<180 ) r=0 b=c g=x
else if( 180<=h and h<240 ) r=0 b=x g=c
else if( 240<=h and h<300 ) r=x b=0 g=c
else if( 300<=h and h<360 ) r=c b=0 g=x
```

c) Cálculo de escala

```
r = (r+m) * 255
g = (g+m) * 255
b = (b+m) * 255
```

#### ■ Cálculo del filtro (*Suma*)

Consiste en sumarle a cada componente HSL los valores pasados por parámetro (con saturación o reinicio según corresponda).

El filtro debe tomar cada pixel y convertirlo a HSL, modificarlo y por ultimo convertirlo nuevamente a RGB.

El código que está a continuación corresponde a la operatoria para un pixel sobre las componentes HSL, donde HH, SS y LL son los parámetros que hay que sumarle a las componentes  $h$ ,  $s$  y  $l$ , respectivamente.

```
if( h+HH >= 360 ) h = h + HH - 360
else if( h+HH < 0 ) h = h + HH + 360
else
           h = h + HH

if( s+SS >= 1 ) s = 1
else if( s+SS < 0 ) s = 0
else
           s = s + SS

if( l+LL >= 1 ) l = 1
else if( l+LL < 0 ) l = 0
else
           l = l + LL
```

#### 2.3.1. Implementación 1

Reemplazar la etapa de *Suma* por una implementación en ASM. Desde la misma se llamarán a las funciones en C para convertir entre RGB y HSL.



(a) Imagen Original

(b) Filtro HSL

Figura 5: Filtro HSL para la imagen Paisaje. Parámetros  $H=-30$ ,  $S=1$ ,  $L=0,1$

### 2.3.2. Implementación 2

Realizar todas las etapas del filtro en ASM.

## 3. Formato BMP

El formato BMP es uno de los formatos de imágenes más simples: tiene un encabezado y un mapa de bits que representa la información de los píxeles. En este trabajo práctico se utilizará una biblioteca provista por la cátedra para operar con archivos en ese formato. Si bien esta biblioteca no permite operar con archivos con paleta, es posible leer dos tipos de formatos, tanto con o sin transparencia. Ambos formatos corresponden a los tipos de encabezado: BITMAPINFOHEADER (40 bytes) y BITMAPV5HEADER (124 bytes).

El código fuente de la biblioteca está disponible como parte del material, deben seguirlo y entenderlo. Las funciones que deben implementar reciben como entrada un puntero a la imagen. Este puntero corresponde al mapa de bits almacenado en el archivo. El mismo está almacenado de forma particular: **las líneas de la imagen se encuentran almacenadas de forma invertida**. Es decir, en la primera fila de la matriz se encuentra la última línea de la imagen, en la segunda fila se encuentra la anteúltima y así sucesivamente.

## 4. Ejercicios

Los ejercicios están divididos en dos partes, por un lado la implementación de los filtros y por otro el análisis de los mismos (que implica también implementar código).

### Implementación

La cátedra provee los filtros implementados en C. Deberán implementar en ASM para cada filtro las dos versiones explicadas anteriormente.

### Análisis

Consiste en estudiar los tres filtros propuestos con sus tres implementaciones distintas.

- a) Comparación entre `ASM_blur1`, `ASM_blur2` y `C_blur`

b) Comparación entre `ASM_merge1`, `ASM_merge2` y `C_merge`

c) Comparación entre `ASM_HSL1`, `ASM_HSL2` y `C_HSL`

Las siguientes preguntas deben ser usadas como guía. La evaluación del trabajo práctico no solo consiste en responder las preguntas, sino en desarrollar y responder nuevas preguntas sugeridas por ustedes mismos buscando entender y razonar sobre el modelo de programación SIMD y la microarquitectura del procesador.

- ¿Cuál implementación tiene mejor rendimiento?
- ¿En qué casos? ¿Depende del tamaño de la imagen? ¿Depende de la imagen en sí?
- ¿Cómo mejoraría el rendimiento de las implementaciones propuestas?
- ¿Es una comparación justa?, ¿La cantidad de operaciones en cada implementación es la misma?, ¿y los accesos a memoria?
- ¿La limitación de rendimiento está en el acceso a memoria?, ¿o en la memoria cache?, ¿esta se podría acceder mejor?
- ¿Hay problemas de rendimiento por los saltos condicionales? ¿Es posible evitarlos?
- ¿El patrón de acceso a la memoria es desalineado? ¿Hay forma de mejorarlo? ¿Es posible medir cuánto se pierde?
- ¿Hay diferencias en operar con enteros o punto flotante? ¿La imagen final tiene diferencias significativas?
- ¿El overhead de llamados a funciones es significativo? ¿Se puede medir?

#### 4.1. Consideraciones

Tener en cuenta las siguientes características generales,

- El ancho de las imágenes es siempre mayor a 16 píxeles y múltiplo de 4 pixeles.
- No se debe perder precisión en los cálculos (excepto las conversiones).
- Las funciones implementadas en ASM deberán operar con la mayor cantidad posible de bytes.
- El procesamiento de los pixeles se deberá hacer **exclusivamente** con instrucciones SSE.
- El trabajo práctico se debe poder ejecutar en las máquinas de los laboratorios.

## 4.2. Desarrollo

Para facilitar el desarrollo se cuenta con todo lo necesario para poder compilar y probar las funciones a medida que las implementan. Los archivos entregados están organizados de la siguiente forma:

- **src**: Contiene los fuentes del programa principal *tp2*
- **src/bmp**: Contiene los fuentes de la biblioteca de BMP
- **src/filters**: Contiene las implementación de todos los filtros
- **src/tools**: Contiene los fuentes del programa *diff*
- **bin**: Contiene los ejecutables, *tp2* y *diff*
- **img**: Contiene imágenes de prueba
- **test**: Contiene scripts para realizar tests sobre los filtros y uso de la memoria

### 4.2.1. Compilar

Ejecutar `make` desde la carpeta **src**.

### 4.2.2. Uso

Ejecutando `./bin/tp2 --help` obtenemos:

Uso: `./tp2 <c/asm1/asm2> <filtro> <parametros...>`

Opcion C o ASM

    c : ejecuta el codigo C  
    asm1 : ejecuta el codigo ASM version 1  
    asm2 : ejecuta el codigo ASM version 2

Filtro:

    <c/asm1/asm2> blur <src> <dst>  
    <c/asm1/asm2> merge <src1> <src2> <dst> <value>  
    <c/asm1/asm2> hsl <src> <dst> <h> <s> <l>

El programa toma dos parámetros inicialmente y luego una serie de parámetros adicionales dependiendo del filtro seleccionado.

El primer parámetro es la implementación a utilizar, puede ser **c**, **asm1** o **asm2**. El segundo parámetro es el nombre del filtro, puede ser **blur**, **merge** o **hsl**. Por ultimo los parámetros adicionales dependiendo de cada filtro y respetando el siguiente:

- **src**: Ruta del archivo de entrada
- **dst**: Ruta del archivo de salida

En Merge,

- **value**: valor entre 0 y 1 que indica el porcentaje de mezcla

En HSL,

- **h**: valor entre 0 y 360 que indica la cantidad de grados de rotación del color
- **s**: valor entre -1 y 1 que indica cuánto sumar a la componente de saturación
- **l**: valor entre -1 y 1 que indica cuánto sumar a la componente de luz

#### 4.2.3. Ejemplo de uso

- `./tp2 asm1 blur ../img/lena.bmp ../img/lena_blur_asm1.bmp`

Aplica el filtro **blur** a la imagen `../img/lena.bmp` utilizando la implementación en `asm1` del filtro y almacena el resultado en `../img/lena_blur_asm1.bmp`

- `./tp2 c hsl ../img/lena.bmp ../img/lena_hsl_c.bmp 31.0 1.0 0.2`

Aplica el filtro **hsl** a la imagen `../img/lena.bmp` utilizando la implementación en `c` del filtro con parámetros  $h=31.0$ ,  $s=1.0$ ,  $l=0.2$  y almacena el resultado en `../img/lena_hsl_c.bmp`.

#### 4.2.4. Mediciones de rendimiento

La forma de medir el rendimiento de nuestras implementaciones se realizará por medio de la toma de tiempos de ejecución. Como los tiempos de ejecución son muy pequeños, se utilizará uno de los contadores de performance que posee el procesador.

La instrucción de assembler `rdtsc` permite obtener el valor del Time Stamp Counter (TSC) del procesador. Este registro se incrementa en uno con cada ciclo del procesador. Obteniendo la diferencia entre los contadores antes y después de la llamada a la función, podemos obtener la cantidad de ciclos de esa ejecución. Esta cantidad de ciclos no es siempre igual entre invocaciones de la función, ya que este registro es global del procesador y se ve afectado por una serie de factores.

Existen principalmente dos problemáticas a solucionar:

- a) La ejecución puede ser interrumpida por el *scheduler* para realizar un cambio de contexto, esto implicará contar muchos más ciclos (*outliers*) que si nuestra función se ejecutara sin interrupciones.
- b) Los procesadores modernos varian su frecuencia de reloj, por lo que la forma de medir ciclos cambiará dependiendo del estado del procesador.

Para medir tiempos deberán idear e implementar una metodología que les permita evitar estos dos problemas. En el archivo `rdtsc.h` encontrarán las funciones necesarias para implementarla.

#### 4.2.5. Herramientas

En el código provisto, podrán encontrar una herramienta que les permitirá comparar dos imágenes. El código de la misma se encuentra en `src/tools`, y se compila junto con el resto del trabajo práctico. El ejecutable, una vez compilado, se almacenará en `bin/diff`. La aplicación se utiliza desde linea de comandos de la forma:

`./bin/diff <opciones> <archivo_1> <archivo_2> <epsilon>`.

Esto compara los dos archivos según las componentes de cada pixel, siendo epsilon la diferencia máxima permitida entre pixeles correspondientes de las dos imágenes. Tiene dos opciones: listar las diferencias o generar imágenes blanco y negro por cada componente, donde blanco es marca que hay diferencia y negro que no.

Las opciones soportadas por el programa son:

-i, --image	Genera Imágenes de diferencias por cada componente
-v, --verbose	Lista las diferencias de cada componente y su posición en la imagen
-a, --value	Genera las imágenes mostrando el valor de la diferencia
-s, --summary	Muestra un resumen de diferencias

#### 4.2.6. Tests

Para verificar el correcto funcionamiento de los filtros sin tener que hacer pruebas manualmente. Se provee un conjunto de scripts que se encuentran en la carpeta `solucion/tests`. Para utilizarlos se deben realizar los siguientes pasos:

1- Generar las imágenes para probar.

Ejecutar el script `./1_generar_imagenes.sh`

2- Correr los casos con el TP implementado por la cátedra y generar los resultados de cada filtro.

Ejecutar el script `./2_generar_resultados_catedra.sh`

3- Correr los casos con el TP implementado por ustedes. Genera resultados y chequea **diferencias** con los resultados de la cátedra.

Ejecutar el script `./3_correr_tests_diff.sh`

4- Correr los casos con el TP implementado por ustedes. Genera resultados y chequea **uso de memoria**.

Ejecutar el script `./4_correr_tests_mem.sh`

Una vez ejecutados los dos primeros pasos, es posible ejecutar los siguientes sin tener que ejecutar los primeros nuevamente.

### 4.3. Informe

El informe debe incluir las siguientes secciones:

- a) Carátula: La carátula del informe con el **número/nombre del grupo**, los **nombres y apellidos** de cada uno de los integrantes junto con **número de libreta** y **email**.
- b) Introducción: Describe lo realizado en el trabajo práctico. (y si quedó algo sin realizar)
- c) Desarrollo: Describe **en profundidad** cada una de las funciones que implementaron. Para la descripción de cada función deberán decir cómo opera una iteración del ciclo de la función. Es decir, cómo mueven los datos de la imagen a los registros, cómo los reordenan para procesarlos, las operaciones que se aplican a los datos, etc. Para esto pueden utilizar pseudocódigo, diagramas (mostrando gráficamente el contenido de los registros **XMM**) o cualquier otro recurso que les resulte útil para describir la adaptación del algoritmo al procesamiento vectorial. No se deberá incluir el código assembler de las funciones (aunque se pueden incluir extractos en donde haga falta).

- d) Resultados: **Deberán analizar y comparar** las implementaciones de las funciones en su versión **C** y **assembler** y mostrar los resultados obtenidos a través de tablas y gráficos. Para esto deberán plantear experimentos que les permitan medir el rendimiento y comparar entre las implementaciones. Deberán además explicar detalladamente los resultados obtenidos y analizarlos. En el caso de encontrar anomalías o comportamientos no esperados deberán construir nuevos experimentos para entender qué es lo que sucede.
- e) Conclusión: Reflexión final sobre los alcances del trabajo práctico, la programación vectorial a bajo nivel, problemáticas encontradas, y todo lo que consideren pertinente.

El informe no puede exceder las **20** páginas, sin contar la carátula.

**Importante:** El informe se evalúa de manera independiente del código. Puede reprobarse el informe y en tal caso deberá ser reentregado para aprobar el trabajo práctico.

## 5. Entrega

Se deberá entregar un archivo comprimido que contendrá la carpeta **src** junto con el informe.

La fecha de entrega de este trabajo es **5/5** y deberá ser entregado a través de la página web. El sistema sólo aceptará entregas de trabajos hasta las **17:00hs** del día de entrega.

Ante cualquier problema con la entrega, comunicarse por mail a la lista de docentes.