DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico II

subtítulo del trabajo

Organización del Computador II

Primer Cuatrimestre de 2019

Integrante	LU	Correo electrónico
Nombre	XXX/XX	`mail`
Nombre	XXX/XX	`mail`

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

Figura 1: Descripcion de la figura

1. Objetivos generales

El objetivo de este Trabajo Práctico es ...

2. Contexto

Título del párrafo Bla bla bla bla. Esto se muestra en la figura **??**.

```
struct Pepe {

    ...

};
```

3. Enunciado y solución

4. Sharpen

El filtro sharpen tiene el propósito de realzar las diferencias entre los pixeles, acentuando todo cambio de color en la imagen. Es por esto que para calcular cada pixel de la imagen destino usaremos el pixel correspondiente en la imagen original y los pixeles que lo rodean. Diremos entonces que el pixel correspondiente es el pixel central y que los ocho pixeles que lo rodean son el resto. Para cada color el pixel destino cumplira la siguiente propiedad: $pixelDestino = pixelCentral * 9 - pixelesPerifericos$. Este cálculo no se puede realizar para calcular el valor de los píxeles de los bordes de la imágen destino. Es por esto que los dejaremos pintados de negro. En todos los píxeles, queremos que la cuarta componente, el alfa o coeficiente de transparencia, siempre tenga un valor de 255, o sea el máximo.

Como cada pixel está compuesto por cuatro componentes de un byte cada una, el píxel en si pesa 32 bytes y por lo tanto, podemos almacenar hasta cuatro píxeles en un registro de 128 bits.

En nuestra implementación del filtro sharpen nos propusimos obtener el resultado para dos píxeles de la imagen destino al mismo tiempo. Dadas las coordenadas (i, j) e (i, j+1) de un par de píxeles destino que queremos calcular, para tres filas diferentes levantamos cuatro píxeles contiguos obteniendo así doce píxeles de la imagen original con los siguientes índices:

Nótese que los 9 píxeles izquierdos son los píxeles necesarios para calcular el píxel destino izquierdo y los 9 píxeles derechos son los píxeles necesarios para calcular el pixel destino derecho.

Cuadro 1: Píxeles a procesar de la imágen fuente en la iteración i, j

Registro	Píxel 1	Píxel 2	Píxel 3	Píxel 4
xmm0	(i - 1, j - 1)	(i - 1, j)	(i - 1, j + 1)	(i - 1, j + 2)
xmm1	(i, j - 1)	(i, j)	(i, j + 1)	(i, j + 2)
xmm2	(i + 1, j - 1)	(i + 1, j + 2)	(i + 1, j + 2)	(i + 1, j + 2)

Habiendo cargado los valores de los píxeles en registros de 128 bits, nuestro objetivo es empezar a hacer sumas y restas entre los registros para lograr el resultado deseado. Quisiéramos poder obtener los valores finales para cada componente. Pero como en los resultados intermedios podrían crecer mucho, tendremos a bien aumentar la precisión con la cual representamos los números enteros. Subiremos de 8 bits a 16 bits, pasando de representar cada componente con un byte a hacerlo con un word. Para esto usamos la operación PUNPCKLBW (packed unpack low byte to word) y la operación PUNPCKHBW (packed unpack high byte to word). Entonces nuestra matriz de pixeles queda repartida en seis registros de la siguiente manera:

Cuadro 2: Píxeles desempaquetados

	Columna 1	Columna 2	Columna 3	Columna 4
	Low	High	Low	High
Fila i-1	xmm0		xmm3	
Fila i	xmm1		xmm5	
Fila i+1	xmm2		xmm6	

Cada registro contiene dos píxeles almacenados con precisión de words en cada componente. Al haber representado nuestros datos de esta manera, nuestra implementación difiere de la implementación de la cátedra, que castea cada byte a float. Dado que no existe representación exacta para los enteros al usar floats, y que usando words tenemos asegurado que no haya overflow, consideramos que nuestra implementación es adecuada y nos evita problemas numéricos.

Para calcular el valor del píxel destino izquierdo, es necesario obtener el resultado de la cuenta $pixelDestino = pixelCentral * 9 - pixelesPerifericos$. Comenzaremos realizando las operaciones necesarias para poder obtener el valor de la suma de los píxeles periféricos, luego obtendremos el valor del píxel central multiplicado por nueve y finalmente realizaremos la resta entre ambos valores, obteniendo así el valor deseado.

A la hora de realizar sumas y restas de píxeles aprovecharemos que sus componentes siempre siguen el mismo orden y nuestra estrategia estará basada en utilizar este “alineamiento” de los valores, eligiendo con qué registros operar en base a qué píxeles contengan estos registros.

Para calcular la suma de los píxeles periféricos, comenzaremos observando que los ocho píxeles que nos interesan están distribuidos en 6 registros diferentes.

Los registros $xmm0$ y $xmm2$ contienen píxeles que nos interesan tanto en el high como en el low. Sumamos ambos almacenando el resultado en el registro $xmm7$. Digamos que $xmm7$ contiene ahora píxeles llamados [A B]. Copiamos el contenido de $xmm7$ a $xmm8$, realizamos un right shift de 8 bytes en $xmm8$, de manera que ahora contiene [0 A] y almacenamos la suma de $xmm7$ y $xmm8$ en $xmm7$ el cual pasa a contener [A A+B]. Nótese que A+B es un píxel equivalente al total de la suma de todos los píxeles originalmente contenidos por $xmm0$ y $xmm2$.

A continuación, nuestro objetivo es obtener el valor de la suma de los cuatro píxeles restantes. En este paso aprovecharemos que todos estos píxeles están en el low de un registro. Limpiamos $xmm8$ y le sumamos consecutivamente los registros $xmm3$, $xmm5$, $xmm6$ y $xmm1$. Siendo los primeros tres los registros cuyo low almacena los píxeles periféricos del borde derecho y el low de $xmm1$ siendo el píxel del borde izquierdo y la fila central.

Almacenamos en $xmm7$ la suma de $xmm7$ y $xmm8$ con lo cual el low de $xmm7$ es equivalente a la suma de los píxeles periféricos. Hemos logrado así obtener uno de los operandos para nuestro cálculo principal.

A continuación cargamos en $xmm10$ una máscara, definida en words, que contiene nueves en el high, exceptuando por el número más significante, que está seteado en cero. Utilizando $pmullw xmm1, xmm10$ obtenemos en el high de $xmm1$ el valor del píxel central multiplicado por nueve, que es el segundo operando necesario para nuestro cálculo principal.

Realizamos la resta correspondiente entre ambos operandos con psubusw y almacenamos nuestro resultado, el valor del pixel destino izquierdo, en el low del registro $xmm1$.

Para calcular el valor del píxel central derecho se realizan exactamente las mismas operaciones, pero intercambiando low por high en cada caso y almacenando el valor del píxel destino derecho en el high del registro $xmm5$.

Para escribir en memoria ambos píxeles, primero necesitamos que ambos estén en un mismo registro con precisión de byte en cada componente. Logramos ambas cosas con un solo comando: $packusb\, xmm1, xmm5$. Como este comando nos deja $xmm1$ = [0 pi pd 0], realizamos un shift derecho de 4 bytes para mandar ambos píxeles resultado al low de $xmm1$. Con esto estamos listos para escribir los píxeles resultado en memoria

5. Conclusiones y trabajo futuro