DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Final

Procesamiento Digital de Imágenes Utilizando Instrucciones SSE

3 de Marzo de 2011 — Organización del Computador II

Integrante	LU	Correo electrónico
Heredia Favieri, Nadia Mariel	589/08	heredianadia@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

1.1. Objetivo

El objetivo de este trabajo práctico es proveer diversas funciones para el procesamientos de imágenes, como por ejemplo filtros y transformaciones. Estas funciones se implementan en **Assembly** utilizando instrucciones SIMD de la arquitectura IA-32 de Intel correspondientes a los subconjuntos de instrucciones denominados MMX, SSE1, SSE2 y SSE3. Consecuentemente, es requisito contar con un procesador de la familia Intel o similar que tenga soporte para estas instrucciones.

El set de instrucciones SSE utiliza registros XMM de 128 bits que permiten por ejemplo multiplicar 4 números empaquetados de 32 bits de punto flotante con otros 4 números de igual tipo en una sola instrucción.

1.2. Instrucciones SIMD

La tecnología SIMD, (Single Instrucion Multiple Data), consiste en realizar operaciones a un vector de varios elementos de forma atómica. La ventaja de SIMD radica en que las operaciones realizadas a los elementos dentro del vector, se realizan en paralelo mediante hardware e instrucciones especializados. De esta manera, se logra paralelizar a la hora de procesar información. Se puede llevar a cabo el cómputo de varios puntos de una imagen en un mismo paso, obteniendo una ventaja frente al procesamiento utilizando solamente registros de propósito general.

Las instrucciones SIMD son muy utiles para la manipulación de imágenes ya que generalmente es necesario realizar una misma operación sobre muchos datos, permitiendo optimizar estas operaciones repetitivas.

1.3. Procesamiento Digital de Imágenes

El **Procesamiento Digital de Imágenes** es el conjunto de técnicas que se aplican a las imágenes digitales con el objetivo de mejorar la calidad, o de modificar ciertos aspectos en particular que sean de interés.

2. Descripción Funcional

2.1. Funciones

Las siguientes funciones de **Detección de bordes** reciben dos imágenes, una fuente y una destino, ambas en escala de grises. Utilizando distintas técnicas, buscan los bordes de la imagen fuente. El resultado se guarda en la imagen destino.

- **Roberts** Utiliza el operador de Roberts para buscar bordes en la imagen fuente.
- **Prewitt** Utiliza el operador de Prewitt para buscar bordes en la imagen fuente.
- **Sobel** Utiliza el operador de Sobel para buscar bordes en la imagen fuente.
- **Scharr** Utiliza el operador de Scharr para buscar bordes en la imagen fuente.
- **Laplace** Utiliza el operador de Laplace para buscar bordes en la imagen fuente.

Las siguientes funciones de **Transformaciones morfológicas** reciben dos imágenes, una fuente y una destino, ambas en escala de grises. El resultado se guarda en la imagen destino.

- **Dilation (Dilatación)** Se aplica un operador de máximo local.
- **Erotion (Erosión)** Es el opuesto a la operación de dilatación, se aplica un operador de mínimo local.
- **Open (Apertura)** Es una combinación de las operaciones de dilatación y erosión. Primero se erosiona la imagen y luego se dilata.
- **Close (Cierre)** Como open, es una combinación de las operaciones de dilatación y erosión, pero al revés. Primero se dilata la imagen y luego se erosiona.
- **Gradient (Gradiente Morfológico)** Es la diferencia entre la dilatación y la erosión.
- **Top Hat** Es la diferencia entre la imagen original y la apertura de la imagen.

- **Black Hat** Es la diferencia entre el cierre de la imagen y la imagen original.

Las siguientes funciones aplican **filtros** sobre una imagen fuente y devuelven el resultado en una imagen destino.

- **Negative** Devuelve el negativo de una imagen.
- **Blur** Suaviza la imagen.

Las siguientes funciones realizan **Operaciones** utilizando dos imágenes fuente, devolviendo como destino una combinación de estas dos.

- **Suma de imágenes** Suma los valores de las dos imágenes.
- **Promedio de imágenes** Realiza el promedio de los valores de las dos imágenes.
- **Alpha Blending** Se mezclan las imágenes, teniendo en cuenta el valor de α.

2.2. Ejemplos

(a) Original Lenna

(b) Roberts

(c) Prewitt

(d) Sobel

(e) Scharr

(f) Laplace

(g) Dilatación, 10 iteraciones

(h) Erosión, 10 iteraciones

(i) Apertura

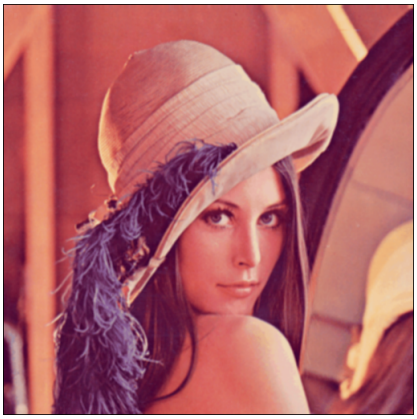
(j) Cierre

(k) Gradiente Morfológico

(l) TopHat

(m) BlackHat

(n) Negative



(ñ) Blur

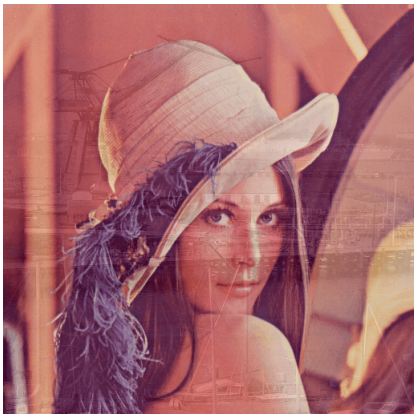
(o) Original Kiel

(p) Suma Lenna con Kiel

(q) Promedio Lenna con Kiel

(r) Alpha Blending Lenna con Kiel con $\alpha = 0{,}25$

(s) Alpha Blending Lenna con Kiel con $\alpha = 0{,}66$



(t) Alpha Blending Lenna con Kiel con $\alpha = 0{,}80$

3. Explicación y Fundamentos Teóricos

3.1. Convoluciones

Una convolución es una operación matemática simple que es fundamental para muchas operaciones de procesamiento de imágenes. Las convoluciones proveen una forma de operar con dos matrices de números, generalmente de distinto tamaño, para producir una tercera matríz. Pueden usarse para implementar operadores en los que los valores de los píxeles de salida son una combinación lineal de ciertos valores de los píxeles de entrada.

En el contexto de procesamiento de imágenes, una de las matrices es normalmente una imagen fuente y la segunda matríz es más chica y se conoce como **kernel**.

Para realizar una convolución es necesario ir desplazando el kernel sobre la imagen, colocando el centro de éste sobre la posición a procesar. En general si el kernel es de n x n y n es impar el centro del kernel se define como la casilla central, pero en otros casos puede variar. Se empieza generalmente en la esquina superior izquierda, y se mueve el kernel por todas las posiciones en las que quepa entero dentro de los límites de la imagen. El valor de cada posición de la imagen de salida es calculado multiplicando el valor de cada una de las casillas del kernel por los valores de la imagen fuente correspondientes, y luego sumando todos estos números.

Matemáticamente una convolución puede definirse como:

$$dst(x, y) = \sum_{k=1}^{m} \sum_{l=1}^{n} src(x + k - 1, y + l - 1) * K(k, l)$$

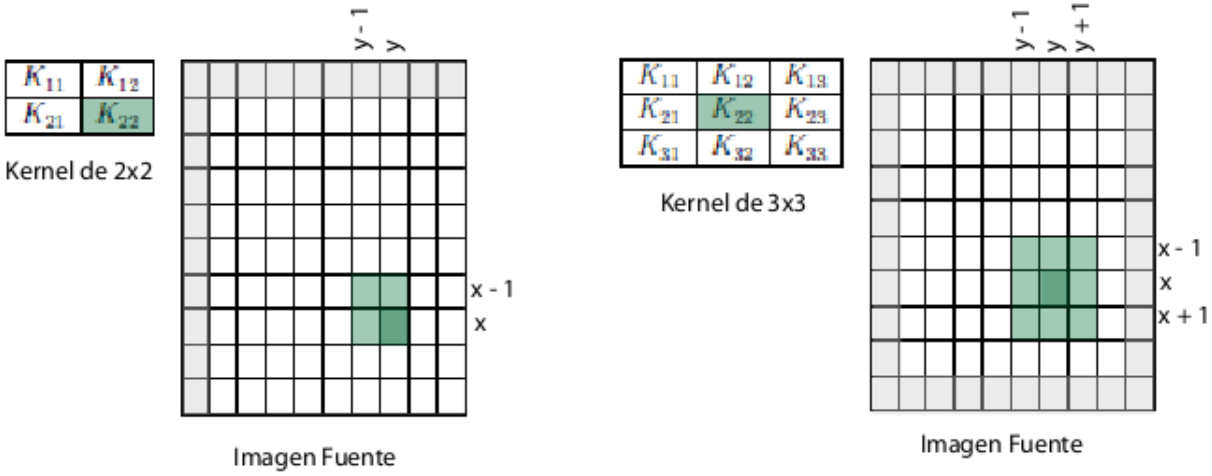
para cada posición x, y que se procese. Donde K es el kernel, src es la imagen fuente, y dst es la imagen destino.

Las convoluciones pueden ser usadas para implementar muchos operadores diferentes, como por ejemplo el de suavizado (blur) y el operador de Sobel para detección de bordes.

Los siguientes gráficos muestran como funcionan las convoluciones con kernels de 2x2, de 3x3 y de 5x5. Hay ciertos valores de la imagen fuente que no se procesan, ya que al intentar realizar una convolución con el kernel quedarían

fuera de rango algunos valores necesarios para realizar la convolución correspondiente. Estos valores están marcados en gris, y la cantidad de elementos que no se procesan depende del tamaño del kernel en cuestión.

Para cada posición x, y de la fuente, hay que realizar entonces una convolución con el kernel correspondiente, posicionando el kernel con el centro (marcado en verde oscuro) sobre la posición x, y, y realizando la operación adecuada. En verde claro están marcadas sobre la imagen fuente todas las casillas necesarias para procesar la casilla (x, y).



Ejemplo gráfico de convoluciones con kernels de distintos tamaños

Una forma concreta de calcular los valores de cada posición de la imagen destino en función de los valores de la imagen fuente y el kernel es la siguiente:

$$\begin{aligned} \mathrm{dst}(x,\, y) = \mathrm{src}(x-1,\, y-1) * K_{11} + \mathrm{src}(x-1,\, y) * K_{12} \\ + \mathrm{src}(x,\, y-1) * K_{21} + \mathrm{src}(x,\, y1) * K_{22} \end{aligned}$$

$$\begin{aligned} \mathrm{dst}(x,\, y) = \mathrm{src}(x-1,\, y-1) * K_{11} + \mathrm{src}(x-1,\, y) * K_{12} + \mathrm{src}(x-1,\, y+1) * K_{13} \\ + \mathrm{src}(x,\, y-1) * K_{21} + \mathrm{src}(x,\, y1) * K_{22} + \mathrm{src}(x,\, y+1) * K_{23} \\ + \mathrm{src}(x+1,\, y-1) * K_{31} + \mathrm{src}(x+1,\, y) * K_{32} + \mathrm{src}(x+1,\, y+1) * K_{33} \end{aligned}$$

Límites de las Convoluciones

Un problema que surge al trabajar con convoluciones es cómo tratar los valores en las fronteras de la imagen. ¿Qué ocurre cuando se quiere realizar una convolución con un elemento que se encuentra en el borde de la imagen?

Existen diversas formas de tratar este problema, una es trabajar con una imagen auxiliar más grande, que permita procesar todos los elementos de la imagen original. La imagen más grande puede tener bordes replicados o bien pueden tener un valor constante (en general 0). Otra forma es ignorar los valores de los bordes y sólamente procesar aquellos elementos pertenecientes a la imagen sin bordes.

El tamaño de los bordes depende del tamaño del kernel usado, si éste es de n x n (con n impar), entonces los bordes serán de (n-1)/2 de ancho en todos lados (arriba, abajo, derecha e izquierda). En el caso particular de n=2, los bordes son de una fila y una columna, arriba y a la izquierda respectivamente.

3.2. Detección de Bordes

La detección de bordes en una imagen filtra la información innecesaria, preservando las características fundamentales de la imagen. Puede definirse como un borde a los píxeles donde la intensidad de la imagen cambia de forma abrupta. Si se considera una función de intensidad de la imagen, entonces lo que se busca son saltos en dicha función. La idea básica detrás de cualquier detector de bordes es el cálculo de un operador local de derivación.

El valor de la primera derivada sirve para detectar la presencia de bordes, mientras que el signo de la segunda derivada indica si el pixel pertenece a la zona clara o a la zona oscura de cada borde. Además, la segunda derivada presenta siempre un cero en el punto medio de la transición.

Lo dicho hasta ahora se refiere solamente a perfiles unidimensionales, sin embargo la extensión a dos dimensiones es inmediata. Se define simplemente el perfil en la dirección perpendicular a la dirección del borde y la interpretación anterior seguirá siendo válida. La primera derivada en cualquier punto de la imagen estará dada por la magnitud del gradiente.

El gradiente de una imagen I en la posición (x, y) está dado por el vector:

$$\nabla I = \begin{bmatrix} I_x \\ \\ I_y \end{bmatrix} = \begin{bmatrix} \frac{\partial I}{\partial x} \\ \\ \frac{\partial I}{\partial y} \end{bmatrix}$$

El vector gradiente siempre apunta en la dirección de la máxima variación de la imagen I en el punto (x, y). Para los métodos de gradiente interesa conocer la magnitud de este vector gradiente de la imagen, denotado por $||\nabla I||$. $||\nabla I|| = \sqrt{{I_x}^2 + {I_y}^2}$

Esta cantidad representa la variación de la imagen I por unidad de distancia en la dirección del vector ∇I. El gradiente de la imagen se suele aproximar, haciendo que sea más facil de calcular computacionalmente.
$||\nabla I|| \approx ||I_x|| + ||I_y||$

Para el cálculo del gradiente de la imagen es necesario obtener las derivadas parciales, éstas se pueden implementar digitalmente de varias formas. Una forma es mediante máscaras (kernels). Si el kernel utilizado tiene tamaño mayor, los errores producidos por efectos del ruido son reducidos mediante promedios locales, pero requieren más procesamiento. El requisito básico de un operador de gradiente es que la suma de los coeficientes del kernel sea nula, para que la derivada de una zona uniforme de la imagen sea cero. Se realiza una convolución con el kernel y la imagen.

Los operaciones de gradiente se realizan en dos etapas. Primero se buscan bordes horizontales utilizando el kernel correspondiente a la derivada parcial en x, y luego se buscan los bordes verticales usando el kernel correspondiente a la derivada parcial en y.

Operadores de gradiente para X y para Y:

1	0
0	-1

0	1
-1	0

Roberts

-1	0	1
-1	0	1
-1	0	1

-1	-1	-1
0	0	0
1	1	1

Prewitt

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

Sobel

-3	0	3
-10	0	10
-3	0	3

-3	-10	-3
0	0	0
3	10	3

Scharr

La detección de bordes utilizando el operador de Laplace, aproxima las derivadas segundas, no es un método de gradiente como los anteriores. Utiliza un solo kernel de 3x3 para la segunda derivada tanto en la dirección de x como en la de y. Una desventaja de este método es que es muy sensible al ruido. Utilizando el kernel de Laplace también se realiza una convolución con la imagen pero en este caso todo se realiza en una sola etapa.

0	1	0
1	-4	1
0	1	0

Laplace

Desarrollando las convoluciones en cada caso, se obtiene que:

$$\textbf{Roberts (x)}(x,\, y) = \mathrm{src}(x-1,\, y-1) - \mathrm{src}(x,\, y)$$

$$\begin{aligned}\textbf{Prewitt (x)}(x,\, y) = {} & \mathrm{src}(x-1,\, y+1) - \mathrm{src}(x-1,\, y-1) + \mathrm{src}(x,\, y+1) \\ & - \mathrm{src}(x,\, y-1) + \mathrm{src}(x+1,\, y+1) - \mathrm{src}(x+1,\, y-1)\end{aligned}$$

$$\begin{aligned}\textbf{Sobel (x)}(x,\, y) = {} & \mathrm{src}(x-1,\, y+1) - \mathrm{src}(x-1,\, y-1) + 2 * (\mathrm{src}(x,\, y+1) \\ & - \mathrm{src}(x,\, y-1)) + \mathrm{src}(x+1,\, y+1) - \mathrm{src}(x+1,\, y-1)\end{aligned}$$

$$\begin{aligned}\textbf{Scharr (x)}(x,\, y) = {} & 3 * (\mathrm{src}(x-1,\, y+1) - \mathrm{src}(x-1,\, y-1)) \\ & + 10 * (\mathrm{src}(x,\, y+1) - \mathrm{src}(x,\, y-1)) \\ & + 3 * (\mathrm{src}(x+1,\, y+1) - \mathrm{src}(x+1,\, y-1))\end{aligned}$$

Para la versión en la dirección de y, simplemente hay que invertir los roles de x e y en el lado derecho de la igualdad.

$$\textbf{Laplace}(x,\, y) = \mathrm{src}(x-1,\, y) + \mathrm{src}(x,\, y-1)) - 4 * (\mathrm{src}(x,\, y)) + \mathrm{src}(x,\, y+1)) + \mathrm{src}(x+1,\, y)$$

3.3. Transformaciones Morfológicas

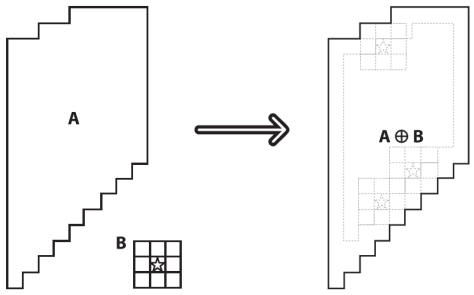
Las transformaciones morfológicas básicas se llaman dilatación y erosión, y surgen en una gran variedad de ocasiones como al remover ruido, al aislar elementos individuales, y al separar elementos dispares en una imagen.

La dilatación es la convolución de una imagen con un kernel morfológico, que en principio puede ser de cualquier forma y tamaño. En este caso, se trabaja con un kernel cuadrado de 3x3. En la dilatación el kernel morfológico tiene el efecto de un operador de máximo local. Para cada elemento, el valor de éste es el máximo entre todos los valores dentro del kernel posicionado con el centro en el elemento. Un kernel morfológico, a diferencia de los utilizados en detección de bordes, no tiene valores, sólo importa su forma y tamaño. Para la erosión se utiliza también un kernel morfológico cuadrado de 3x3 pero con efecto de un operador de mínimo local.

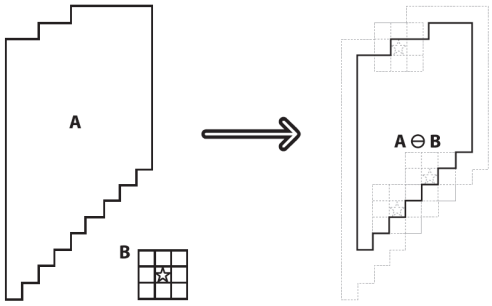
La dilatación hace que las regiones con más brillo en la imagen crezcan y que se suavicen las concavidades. La erosión hace que se suavicen las protuberancias. En general, si la dilatación expande cierta región, la erosión la disminuye.

La dilatación de la imagen A bajo el kernel morfológico B se denota A $\oplus$ B. La erosión de la imagen A bajo el kernel morfológico B se denota A $\ominus$ B.

Las siguientes imágenes muestran los resultados de las operaciones de dilatación y erosión sobre la imagen A. El centro del kernel morfológico B está marcado con una estrella.



(a) Dilatación: Tomar el máximo bajo el kernel morfológico B.



(b) Erosión: Tomar el mínimo bajo el kernel morfológico B.

$$\textbf{Dilatación}(x,\, y) = \text{máx}_{(x',y') \in \text{kernel}_{x,y}} \text{src}(x',\, y')$$

$$\textbf{Erosión}(x,\, y) = \text{mín}_{(x',y') \in \text{kernel}_{x,y}} \text{src}(x',\, y')$$

Al saber que en este caso, se trabaja con un kernel cuadrado de 3x3, se pueden reescribir de forma más concreta las fórmulas matemática que definen la dilatación y erosión.

$$\textbf{Dilatación}(x, y) = \text{máx}\left\{ \begin{array}{c} src(x-1, y-1),\ src(x-1, y),\ src(x-1, y+1) \\ src(x, y-1),\ src(x, y),\ src(x, y+1) \\ src(x+1, y-1),\ src(x+1, y),\ src(x+1, y+1) \end{array} \right\}$$

$$\textbf{Erosión}(x, y) = \text{mín}\left\{ \begin{array}{c} src(x-1, y-1),\ src(x-1, y),\ src(x-1, y+1) \\ src(x, y-1),\ src(x, y),\ src(x, y+1) \\ src(x+1, y-1),\ src(x+1, y),\ src(x+1, y+1) \end{array} \right\}$$

El resto de las operaciones morfológicas están basadas en dilatación y erosión.

Para el cierre y la apertura se utilizan combinaciones de estas operaciones. El efecto más prominente del cierre es eliminar los outliers que tienen valores más bajos que los de sus vecinos, y el de la apertura es eliminar los outliers cuyo valor es más alto que el de sus vecinos.

$$\textbf{Apertura}(\text{src}) = \text{Dilatación}(\text{Erosión}(\text{src}))$$

$$\textbf{Cierre}(\text{src}) = \text{Erosión}(\text{Dilación}(\text{src}))$$

El gradiente morfológico aporta información sobre cuán rápidamente el brillo de la imagen cambia, de ahí el nombre. Puede usarse para detectar bordes ya que realiza restas entre regiones extendidas y contraídas de la imagen.

$$\textbf{Gradiente}(\text{src}) = \text{Dilatación}(\text{src}) - \text{Erosión}(\text{src})$$

La operación de TopHat hace que se revelen áreas cuyas posiciones son más luminosas que sus vecinas.

$$\textbf{TopHat}(\text{src}) = \text{src} - \text{Apertura}(\text{src})$$

La operación de BlackHat hace que se revelen áreas cuyas posiciones son más oscuras que sus vecinas.

$$\textbf{BlackHat}(\text{src}) = \text{Cierre}(\text{src}) - \text{src}$$

3.4. Píxeles

La estructura de un píxel en formato rgba es la siguiente:

```
struct {
unsigned char r; //red
unsigned char g; //green
unsigned chart b; //blue
unsigned char a; //alpha
} pixel;
```

y en grayscale es simplemente un unsigned char que indica el valor en la escala de grises correspondiente.

3.5. Filtros

Para obtener el negativo de una imagen, se considera el valor máximo que puede tener un unsigned char, que es 255. Para cada píxel, su valor será el máximo menos el valor anterior.

$$\begin{aligned}\textbf{Negative}(x,\, y) &= \text{valorMáximo} - \text{src}(x,\, y) = 255 - \text{src}(x,\, y) \\ &= 255(1 - \tfrac{src(x,y)}{255})\end{aligned}$$

Para suavizar una imagen, se realiza un promedio del valor de cada elemento con el de sus vecinos. Se convoluciona un kernel de 3x3 cuyos elementos son todos 1/9 con la imagen. Los valores del kernel representan el promedio, ya que se consideran 9 elementos en total para calcular un valor.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Blur

$$\begin{aligned}\textbf{Blur}(x,\, y) &= \text{src}(x-1,\, y-1) * 1/9 + \text{src}(x-1,\, y) * 1/9 \\ &+ \text{src}(x-1,\, y+1) * 1/9 + \text{src}(x,\, y-1) * 1/9 + \text{src}(x,\, y) * 1/9 \\ &+ \text{src}(x,\, y+1) * 1/9 + \text{src}(x+1,\, y-1) * 1/9 \\ &+ \text{src}(x+1,\, y) * 1/9 + \text{src}(x+1,\, y+1) * 1/9\end{aligned}$$

3.6. Operaciones

Estas funciones consisten en simplemente recorrer elemento a elemento los valores de las dos imágenes fuente, multiplicarlos por algún coeficiente si hiciera falta y sumarlos. En la suma simple, no se multiplican los elementos por ningún coeficiente.

$$\textbf{Suma}(x,\, y) = \text{src}_1(x,\, y) + \text{src}_2(x,\, y)$$

En el promedio los elementos se multiplican por 1/2.

$$\textbf{Promedio}(x,\, y) = \frac{\text{src}_1(x,\, y) + \text{src}_2(x,\, y)}{2} = \frac{1}{2}\,\text{src}_1(x,\, y) + \frac{1}{2}\,\text{src}_1(x,\, y)$$

En Alpha Blending el elemento de la primera imagen se multiplica por α y el de la segunda imagen por $1 - \alpha$.

$$\textbf{Alpha Blending}(x,\, y) = \alpha \times \text{src}_1(x,\, y) + (1 - \alpha) \times \text{src}_2(x,\, y)$$

4. Alcance

Consideraciones Generales:

El ancho y el alto de las imágenes quedan fijos al elegir una imagen. Siempre se considera que la imagen destino tiene las mismas dimensiones que la imagen fuente, o en el caso de haber dos imágenes, las mismas dimensiones que la primera imagen fuente.

4.1. Detección de Bordes

Parámetros

- src: imagen fuente en grayscale
- dst: imagen destino en grayscale
- ancho: ancho de la imagen src en bytes
- alto: alto de la imagen src en bytes

Opciones

No tienen, los kernels que utilizan son de tamaño fijo, el de Sobel es de 2x2, y el de todas las demás funciones es de 3x3. Además en las funciones de detección de bordes por el método del gradiente, siempre se realizan ambas pasadas para la búsqueda de bordes, tanto en x como en y.

Declaración de las funciones

void `asmSobel` (const char* src, char* dst, int ancho, int alto)
void `asmPrewitt` (const char* src, char* dst, int ancho, int alto)
void `asmRoberts` (const char* src, char* dst, int ancho, int alto)
void `asmScharr` (const char* src, char* dst, int ancho, int alto)
void `asmLaplace` (const char* src, char* dst, int ancho, int alto)

4.2. Transformaciones Morfológicas

Parámetros

- src: imagen fuente en grayscale
- dst: imagen destino en grayscale
- ancho: ancho de la imagen src en bytes
- alto: alto de la imagen src en bytes
- (erosión y dilatación) iteraciones: cantidad de veces que se realiza la operación de erosión o dilatación
- (open, close, gradient, topHat y BlackHat) aux: imagen auxilar usada para cálculos intermedios

Opciones

El parámetro iteraciones sirve para elegir cuántas veces se desea que se erosione o dilate una imagen, ya que efecto será más notorio para valores más grandes. Las funciones que llaman a erosión y/o dilatación lo hacen con un parámetro iteraciones fijo de 10, este valor hace que los efectos sean apreciables pero que no de forma exagerada. El kernel morfológico tiene tamaño y forma fija, es un cuadrado de 3x3.

Declaración de las funciones

void `asmErosion` (const char* src, char* dst, int ancho, int alto,
int iteraciones)
void `asmDilation` (const char* src, char* dst, int ancho, int alto,
int iteraciones)
void `asmOpen` (const char* src, char* dst, int ancho, int alto, char* aux)
void `asmClose` (const char* src, char* dst, int ancho, int alto, char* aux)
void `asmGradient` (const char* src, char* dst, int ancho, int alto,
char* aux)
void `asmTopHat` (const char* src, char* dst, int ancho, int alto, char* aux)
void `asmBlackHat` (const char* src, char* dst, int ancho, int alto,
char* aux)

4.3. Filtros

Parámetros

- src: imagen fuente en rgba
- dst: imagen destino en rgba
- ancho: ancho de la imagen src en bytes
- alto: alto de la imagen src en bytes

Opciones

No tienen. El tamaño del kernel que se utiliza para suavizar en la función de blur es fijo, de 3x3. Es decir que el valor de un píxel depende de él mismo y de sus vecinos inmediatos.

Declaración de las funciones

void `asmNegative` (const char* src, char* dst, int ancho, int alto)
void `asmBlur`(const char* src, char* dst, int ancho, int alto)

4.4. Operaciones

Parámetros

- src1: primera imagen fuente en rgba
- src2: segunda imagen fuente en rgba
- dst: imagen destino en rgba
- ancho: ancho de la imagen src en bytes
- alto: alto de la imagen src en bytes
- alpha: porcentaje que aporta la primera imagen al image blending con la segunda imagen.

Opciones

El parámetro alpha sirve para regular la forma en que dos imágenes se mezclan, cuanto más chico sea, más se verá la segunda imagen y cuanto más grande sea, más se verá la primera imagen. Si alpha = 0.5 el efecto producido por alpha blending es el mismo que en el promedio. Se obtiene una imagen que combina las dos imágenes fuente, utilizando $100 * \alpha\,\%$ de src1 y $100 * (1 - \alpha)\,\%$ de src2.

Declaración de las funciones

void `asmAdd` (const char* src1, const char* src2, char* dst,
int ancho, int alto)
void `asmPromedio` (const char* src1, const char* src2, char* dst,
int ancho, int alto)
void `asmAlphaBlending` (const char* src1, const char* src2, char* dst,
int ancho, int alto, float alpha)

5. Sobre la Implementación

5.1. Entorno de desarrollo

La interacción con el usuario y el manejo de archivos fueron realizados en lenguaje C, utilizando la librería OpenCV para facilitar el manejo de imágenes. Es decir se utiliza OpenCV para la carga, el guardado, y la conversión de imágenes a grayscale o a rgba.

Las funciones que realizan el procesamiento de las imágenes, están implementadas en Assembly, utilizando SSE. El proyecto tiene una modalidad interactiva, el usuario ingresa los parámetros y luego se realiza todo el procesamiento.

5.2. Decisiones

5.2.1. Modularidad

Se apunto en todo momento a la modularización y claridad del código, por lo tanto para la mayoría de las funciones realizadas en código Assembly existen macros que facilitan el trabajo. Las macros más utilizadas se encuentran en el archivo macros.asm. Estas macros permiten crear un stack frame y luego destruirlo, ambas reciben como parámetro la cantidad de bytes que se desea que tenga el stack frame.

Las funciones que poseen macros que realizan las operaciones correspondientes son aquellas que realizan convoluciones, es decir todas las de detección de bordes, dilatación, erosión y blur; y negative. Por ejemplo para la función blur existe un archivo macros_blur que contiene todas las operaciones necesarias para que en cada paso del algoritmo se puedan procesar una cierta cantidad de bytes es paralelo, en este caso 4.

5.2.2. Tipos de datos y Precisión numérica

Como ya se vió en la explicación, los valores de los píxeles están formados por unsigned chars, tanto si la imagen está en formato RGBA como si está en grayscale. Este formato permite realizar una cantidad acotada de operaciones, que además debe tener en cuenta la saturación. Al trabajar con imágenes es necesario usar saturación para no obtener valores sin sentido.

Un problema de la saturación, especialmente cuando se realiza a nivel de byte, es la pérdida de precisión, ya que si un byte tiene el valor máximo, se le pueden seguir sumando valores sin ver ningún cambio y este comportamiento no es necesariamente el buscado.

Por lo tanto en muchos casos fue necesario aumentar la precisión, ya sea a nivel de word para trabajar con saturación a nivel de word; o a double word para luego convertir a float. Trabajar con números de punto flotante es no sólo más cómodo sino necesario para operaciones como la multiplicación por números no enteros.

Al trabajar al nivel de byte con saturación, hubo que tener en cuenta el orden en que se realizaban las operaciones, ya que podía cambiar significativamente el resultado. Si un byte está ya saturado al máximo y primero se le suma un valor y luego se le resta otro, solamente se notará el valor que se resta; mientras que si en cambio primero se realiza la resta y luego la suma, las dos operaciones tendrán un efecto visible. Saber cuándo alterar el orden de las operaciones no es una tarea sencilla y requiere de mucha experimentación, obteniendo distintos resultados.

La elección del tipo de datos que mejor se adecue a la operación a realizar no es simple, muchas veces requiere de un proceso de prueba y error antes de poder tomar una decisión, que lleve a que la implementación quede realizada de la mejor manera posible.

A continuación se muestra la forma en que se puede cambiar de tipo de datos. En xmm0 se leen los elementos, xmm6 es un registro auxiliar con ceros que se utiliza para extender la precisión

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}

xmm0: 16 bytes

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

xmm6: ceros

`punpcklbw xmm0, xmm6`

0..0 x_0	0..0x_1	0..0x_2	0..0x_3	0..0x_4	0..0x_5	0..0x_6	0..0x_7

xmm0: 8 words

0	0	0	0	0	0	0	0

xmm6: ceros

`punpcklwd xmm0, xmm6`

0..0 x_0	0..0x_1	0..0x_2	0..0x_3

xmm0: 4 double words

0	0	0	0

xmm6: ceros

`cvtdq2ps xmm0, xmm0`

0..0 x_0	0..0x_1	0..0x_2	0..0x_3

xmm0: 4 single precision floating points

5.3. Cuestiones de implementación

Las fórmulas provistas en la sección teórica proveen una forma bastante directa de trabajar con los bytes de las imágenes. Resta encontrar una forma de relacionar esto con la implementación en código Assembly.

En general, la relación establecida entre los registros y las variables es la siguiente:

- esi: src (En el caso del gradiente morfológico, se usa como puntero a aux y si hay dos imágenes fuente a src1)
- edi: dst
- edx: width
- eax: contador de filas
- ecx: contador de columnas
- ebx: contador de iteraciones en las funciones de dilatación y erosión, o puntero a src2 en las funciones que trabajan con dos imágenes fuente.

En las siguientes tablas puede verse la forma de direccionar con esi, suponiendo que apunta al elemento src(x,y).

esi - ancho - t	esi - ancho	esi - ancho + t
esi - t	esi	esi + t
esi + ancho - t	esi + ancho	esi + ancho + t

Direccionamiento con esi

src(x - 1, y - 1)	src(x - 1, y)	src(x - 1, y + 1)
src(x, y - 1)	src(x, y)	src(x, y + 1)
src(x + 1, y - 1)	src(x + 1, y)	src(x + 1, y + 1)

Direccionamiento con (x,y)

donde **ancho** es el ancho de la imagen en bytes, y **t** es el tamaño de un píxel en bytes (4 para rgba y 1 para grayscale).

Utilizando esta relación, se puede acceder de forma simple a los elementos necesarios para calcular el valor de un píxel destino.

5.4. Descripción de los algoritmos

Para todas las funciones que realizan convoluciones, es decir todas las de detección de bordes, dilatación, erosión y blur; y negative, el esquema general del algoritmo es el siguiente:

Se recorre la matríz, primero a lo ancho y luego a lo alto, salteando los bordes de ser necesario (los píxeles que no se procesan), utilizando un ciclo. En cada paso del ciclo se llama a una macro (o dos si se está en el caso de detección de bordes con operadores de gradiente que primero realizan una operación en x y luego en y), que realiza las operaciones correspondientes a la función en cuestión. Luego se guarda el valor obtenido en destino, y se continúa con el ciclo.

Los punteros pueden avanzarse de a 16, 8, o 4 dependiendo de a cuántos elementos en paralelo se estén procesando. Los contadores de columna se avazan la misma magnitud que los punteros o un cuarto según si se está trabajando en formato rgba o grayscale, ya que un píxel en formato rgba está formado por 4 bytes. La diferencia reside en que el direccionamiento es a byte y por lo tanto los punteros tienen que avanzarse según la cantidad de bytes procesados, y los contadores de columnas tienen en cuenta la cantidad de píxeles.

La siguiente tabla muestra como se trabaja para procesar los elementos en cada función:

Referencias

- **Precisión**: Se refiere a la precisión numérica con la que se trabaja, que depende del tipo de datos. Puede ser a nivel de byte usando saturación sin signo, a nivel de word o float.
- **Bytes**: Se refiere a la cantidad de bytes procesados en cada ciclo. Determina de a cuántas unidades se avanzan los punteros.
- **Píxeles**: Se refiere a la cantidad de píxeles procesados en cada ciclo. Determina de a cuántas unidades se avanzan los contadores de columna.

- **Cols-pre**: Se refiere a la cantidad de columnas que se saltean antes del ciclo.
- **Filas-pre**: Se refiere a la cantidad de fila que se saltean antes del ciclo.
- **Cols**: Se refiere a la cantidad de columnas que se procesan.
- **Filas**: Se refiere a la cantidad de fila que se procesan.

Cols-pre, filas-pre, cols y filas reflejan el hecho de saltear los bordes.

Función	Precisión	Bytes	Píxeles	Cols-pre	Filas-pre	Cols	Filas
Roberts	byte	16	16	1	1	ancho - 1	alto - 1
Prewitt	byte	16	16	1	1	ancho - 2	alto - 2
Sobel	byte	16	16	1	1	ancho - 2	alto - 2
Scharr	word	8	8	1	1	ancho - 2	alto - 2
Laplace	word	8	8	1	1	ancho - 2	alto - 2
Erosión	byte	16	16	1	1	ancho - 2	alto - 2
Dilatación	byte	16	16	1	1	ancho - 2	alto - 2
Apertura	byte	16	16	0	0	ancho	alto
Cierre	byte	16	16	0	0	ancho	alto
Gradiente	byte	16	16	0	0	ancho	alto
TopHat	byte	16	16	0	0	ancho	alto
BlackHat	byte	16	16	0	0	ancho	alto
Negative	float	4	1	0	0	ancho	alto
Blur	float	4	1	1	1	ancho - 2	alto - 2
Suma	byte	16	4	0	0	ancho	alto
Promedio	float	4	1	0	0	ancho	alto
Alpha Blending	float	4	1	0	0	ancho	alto

A modo de ejemplo se presenta el pseudocódigo de Negative.

Algorithm 1: Negative(unsigned char* src, unsigned char* dst, int width, int height)

```
{ Punteros a las matrices. }
esi = src
edi = dst
{ Contadores del ciclo. }
eax = 0
ecx = 0
while eax < height do
    while ecx < width do
        { Macro de la función. Deja el resultado en xmm0.}
        negative
        { Guardar el resultado.}
        [edi] = xmm0
        { Avanzar los punteros.}
        esi = esi + 4
        edi = edi + 4
        { Avanzar el contador de columnas.}
        ecx = ecx + 1
    end
    { Resetear el contador de columnas.}
    ecx = 0
    { Avanzar el contador de filas.}
    eax = eax + 1
end
```

En esta función, es necesario trabajar con números de punto flotante, para no perder precisión al procesar, aunque la información dentro del píxel se almacene con números enteros. Al operar solamente con números enteros se estaría perdiendo información.

Algorithm 2: negative (macro)

{ *Limpiar xmm6 para extender la precisión.* }
pxor xmm6, xmm6

{ *Leer los valores.* }
xmm1 = [esi]

{ *Extender la precisión y pasar a float.* }
punpcklbw xmm1, xmm6
punpcklwd xmm1, xmm6
cvtdq2ps xmm1, xmm1

{ *Cargar el valor 255 definido como una constante en xmm4.* }
movups xmm4, [numeros]

{ *Dividir por 255.* }
divps xmm1, xmm4

{ *Cargar unos en xmm0.* }
movups xmm0, [unos]

{ *Restar.* }
subps xmm0, xmm1

{ *Multiplicar por 255.* }
mulps xmm0, xmm4
{ *xmm0:* $255(1-\frac{x_0}{255})f \mid 255(1-\frac{x_1}{255})f \mid 255(1-\frac{x_2}{255})f \mid 255(1-\frac{x_3}{255})f$}

{ *Volver a pasar a enteros y empaquetar para pasar a byte.* }
cvtps2dq xmm0, xmm0
packssdw xmm0, xmm0
packuswb xmm0, xmm0

Como se dijo al principio, todas las funciones de detección de bordes, erosión, diltación y blur siguen el mismo esquema que negative, pero teniendo en cuenta los valores que aparecen en la tabla anterior. Es decir que se pueden saltear filas y columnas antes de empezar con el ciclo, puede cambiar el valor hasta el cual iteran eax, y ecx, y también varía la cantidad en la que se incrementan los punteros y los contadores.

Por último, lo más importante que cambia, es el código que se encuentra dentro de las macros de cada función. En este código se utiliza el direccionamiento como se vió antes, usando la relación establecida entre esi y src(x, y).

Transformaciones Morfológicas

- Apertura: Se realiza una llamada a erosión y luego una a dilatación usando a aux como imagen intermedia.
- Cierre: Como en la apertura, se realiza una llamada a dilatación y luego una a erosión usando a aux como imagen intermedia.
- Gradiente: Se usa aux para guardar el resultado de la llamada a dilatación y dst para la erosión y luego se realiza la resta entre ambas.
- TopHat: Se deja el resultado de la apertura en dst, y aux se usa para llamar a open. Luego se realiza la resta entre la original (src) y open (dst).
- BlackHat: Se deja el resultado del cierre en dst, y aux se usa para llamar a close. Luego se realiza la resta entre close (dst) y la original (src).

Operaciones

Se recorren a la vez las matrices de las dos imágenes fuente y se leen los valores.

- Suma: Simplemente se suman los elementos.
- Promedio: Se extiende la precisión a float, se multiplican los valores por 1/2 y luego se suman. Finalmente se vuelve a pasar a byte.
- Alpha Blending: Se extiende la precisión a float, se multiplica el primer valor por α, el segundo por 1 - α y se suman. Finalmente se vuelve a pasar a byte.

6. Testing

Para testear el funcionamiento correcto del programa, se realizó un script que genera para cada función varios archivos de entrada para el programa con los parámetros, luego se ejecuta el programa con estos archivos como entrada. Para las funciones que reciben una sola imagen, se realizaron archivos para cada una de estas imágenes, y en el caso de recibir dos se tuvieron en cuenta algunos pares de imágenes compatibles. El parámetro iteraciones se fijó en 10, y alpha se fijó en 0.30. Lo que se obtiene es una visualización de los resultados obtenidos, presionando cualquier tecla, se cierra la ventana con la imagen procesada y se pasa a la siguiente. De esta forma puede verse rápidamente el funcionamiento del programa.

La forma de utilizar el tester es mediante el comando `./tester.sh` dentro de la carpeta src.

7. Rendimiento

Para medir el rendimiento se realizaron programas, uno para medir el tiempo de las funciones realizadas en Assembly y otro para las funciones en C. Estos programas ejecutan 100 veces cada función con parámetros fijos y miden el tiempo. Para medir el tiempo se utilizó la instrucción rdtsc, que lee el timestamp counter y lo devuelve en el registro eax. Los resultados de los tiempos se guardan en archivos de salida. Luego de haber obtenido las 100 mediciones, se realizó un promedio para tener un valor que representara el tiempo medio de cada función.

Los resultados obtenidos se encuentran en la carpeta **performance**. Para conseguir los promedios existe un script, `proms.sh`, al ejecutarlo se obtienen los promedios de los tiempos de las funciones en los archivos promedios_c.txt y promedios_asm.txt.

En la siguiente tabla se ve reflejada la diferencia de rendimiento que existe entre la implementación en C y la implementación en SSE. La última columna representa el porcentaje de tiempo que tarda la función implementada en SSE frente a la misma función implementada en C.

Función	Tiempo SSE (clocks)	Tiempo C (clocks)	Porcentaje SSE (%)
Roberts	380442	20837501	18.25
Prewitt	464725	68242424	0.68
Sobel	473977	76716413	0.61
Scharr	6472154	59601683	10.85
Laplace	3418034	62320036	5.48
Erosión	12691694	468619731	2.708
Dilatación	13111225	393670146	3.33
Apertura	25403296	828161948	3.067
Cierre	25542215	792055972	3.22
Gradiente Morfológico	125432123	868258560	14.44
TopHat	25504898	833696570	3.059
BlackHat	25479094	798134988	3.19
Negative	3422013	38230640	8.95
Blur	9382944	84980138	11.041
Suma de Imágenes	503266	28010959	1.79
Promedio de Imágenes	2291218	45590845	5.025
Alpha Blending	2315470	47013546	4.92

En todos los casos puede apreciarse que la implementación en SSE es mucho más rápida que la realizada en C.

Las funciones que no ofrecen tanto beneficio son aquellas en las que hay que extender la precisión para trabajar, ya sea a word o a float. Para la función de Roberts, al ser un kernel tan chico, de 2x2, los beneficios que ofrece trabajar con SSE no son tan grandes como cuando se usan kernels de 3x3.

8. Modo de Uso

El proyecto utliza la librería de OpenCV, por lo tanto es necesario tenerla instalada para poder compilar. Para instalarla en las distribuciones de Linux basta con `sudo apt-get install libcv-dev`.

Para más información, puede leerse el manual de referencia en `http://opencv.willowgarage.com/documentation/index.html`.

El proyecto contiene un Makefile que permite compilar distintos ejecutables. Por default, el comando `make` produce dos ejecutables: tp y main_c. Ambos permiten procesar las imágenes de la carpeta img-in ofreciendo un menú de opciones. El ejecutable tp utiliza las funciones implementadas en Assembly usando SSE, y main_c utiliza las mismas funciones pero implementadas en C.

Al ejecutar el programa aparece un menú de opciones, luego de elegir la función deseada, hay que elegir la imagen a procesar. Algunas funciones requieren más parámetros, que también se ingresarán por consola. En el caso de utilizar dos imágenes, en ancho y el alto de la segunda imagen deben ser mayores o iguales a los de la primera. Las operaciones de dilatación y erosión tienen el parámetro de cantidad de iteraciones, que debe ser mayor a 0 y menor o igual a 50. Alpha blending requiere un valor de alpha que debe ser float entre 0 y 1 inclusive.

Al finalizar el procesamiento de la imagen, se mostrará la imagen procesada en una ventana. Las imágenes procesadas se guardan en la carpeta img-out.

También es posible compilar los ejecutables utilizados para la medición de tiempos, mediante el comando `make tiempos`, que produce los ejecutables tiempos_asm y tiempos_c.

9. Recursos y Bibliografía

Las imágenes utilizadas para explicar las transformaciones morfológicas fueron tomadas del libro 'Learning OpenCV: Computer Vision with the OpenCV Library'.

- **Libro: 'Learning OpenCV: Computer Vision with the OpenCV Library'**
  http://books.google.com/books?id=seAgiOfu2EIC
  http://www.amazon.com/Learning-OpenCV-Computer-Vision-Library/dp/0596516134
- **Image Processing Learning Resources**
  http://homepages.inf.ed.ac.uk/rbf/HIPR2/hipr_top.htm
- **Image Processing For Fire Arts**
  http://www.cescg.org/CESCG97/boros/
- **Edge Detection Tutorial**
  http://www.pages.drexel.edu/~weg22/edge.html
- **Set of classic test images**
  http://www.hlevkin.com/TestImages/classic.htm
- **Set of additional test images**
  http://www.hlevkin.com/TestImages/additional.htm
- **Discrete Laplace Operators.**
  http://nptel.iitm.ac.in/courses/Webcourse-contents/IIT-KANPUR/Digi_Img_Pro/chapter_8/8_26.html