Organización del Computador II

Segundo Cuatrimestre de 2009

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Práctico 2

Procesamiento de imágenes para la detección de bordes en lenguaje ensamblador

Grupo XOR

Integrante	LU	Correo electrónico
Daniel Grosso	694/08	dgrosso@gmail.com
Nicolás Varaschin	187/08	nicovaras22@gmail.com
Mariano De Sousa	389/08	marian_sabianaa@hotmail.com

Índice

1. Desarrollo

El desarrollo de este trabajo se basó en modificar la implementación anterior con el fin de alcanzar los objetivos. Para lograr una mejor performance usando instrucciones del set SSE se propusieron diversos algoritmos posibles, siendo el último de los que describimos a continuación el elegido para la versión final. En todo momento se asumió un ancho de imagen múltiplo de 16 píxeles y cada algoritmo fue pensado independientemente de la matríz a usar.

El primer algoritmo pensado agarraba 16 píxeles, los 16 píxeles por encima y los 16 píxeles por debajo de esos de la siguiente manera:

```
xmm0  x  x  x  x  x  x  x  x  x  x  x  x  x  x  x  x
xmm1  x  p  p  p  p  p  p  p  p  p  p  p  p  p  p  x
xmm2  x  x  x  x  x  x  x  x  x  x  x  x  x  x  x  x
```

Sólo los píxeles indicados con una `p` se procesaban, usando como información adicional los que están marcados con una `x`.

Luego la matríz correspondiente se cargaba varias veces en otros registros `xmm`, de tal forma que pudieran ser procesados los 14 píxeles buscados en un solo ciclo del bucle principal. Luego, como la imagen es múltiplo de 16 píxeles, se avanzaba esa cantidad para seguir procesando. Esto creó un problema insalvable en el algoritmo: el último píxel de un ciclo y el primero del siguiente ciclo no se procesaban. Se pensó en salvar este problema manipulando esos dos píxeles con registros generales, pero la complejidad del algoritmo aumentaría y se optó por pensar un método más eficaz.

Un segundo algoritmo se basaba en cargar 16 píxeles a procesar, siendo 8 pertenecientes a una fila y los 8 restantes a la fila siguiente, cargando a su vez toda la información necesaria para procesar los dos conjuntos de 8 píxeles. La ventaja de este método está en la simplificación de cálculos, ya que al tratarse de *bytes* sin signo que tienen que ser multiplicados por algún posible valor signado en la matríz del operador, se tiene que extender el *byte* a *word*. Entonces entrando 8 *words* en cada registro `xmm`, tiene sentido procesar de a 8 píxeles. El problema de este método era la ineficiencia en accesos a memoria, y que al tener que procesar de a dos filas al mismo tiempo, el alto de la imagen tenia que ser múltiplo de dos, o manejar el caso impar de otra forma, y esto no era conveniente.

El tercer algoritmo volvió a la idea del primero, procesar de a 14 píxeles. La diferencia está en que procesaba 14 píxeles del principio de la fila y a la vez, 14 del final de la fila y en vez de avanzar de a 16 píxeles, avanzaba de a 14 tanto desde el comienzo hacia el final de la fila como del final hacia el comienzo y cuando llegaba a la mitad de la fila, pasaba a la siguiente. La principal ventaja es que nos librábamos del problema de saber cuando terminó una fila, ya que, recorriendo de ambos lados a la vez, a lo sumo se procesarían dos veces algunos píxeles del medio. El problema fue que este algoritmo necesitaba una mayor cantidad de registros e implicaba un difícil manejo de punteros.

El algoritmo final es una modificación del anterior: se procesan 14 píxeles y

se avanza de a 14 píxeles de izquierda a derecha solamente, y cuando el ancho restante de la imagen es menor que 14 píxeles, se procesan 14 de derecha a izquierda. Otra vez, a lo sumo se procesarán dos veces algunos píxeles del final, pero se justifica, ya que cuesta mas ciclos revisar el ancho y procesar esos píxeles restantes de alguna otra forma.

En la siguiente sección se detalla el funcionamiento del algoritmo junto con diferentes problemas que surgieron al programarlo.

2. Discusión

El código de este trabajo utiliza las funciones en lenguaje `C` del trabajo anterior y nuevas funciones en assembler. El trabajo en assembler está separado en un archivo por filtro a aplicar y un archivo para las *macros* en común, utilizadas por los filtros.

A continuación se detallara lo escrito y pensado para los diferentes algoritmos, que fueron ajustados para mejor rendimiento según la matríz utilizada. Antes de empezar se definen nombres específicos (*macros*) para cada registro `xmm`, para facilitar la lectura del código. Los nombres definidos son: `srcl`, `srch` que contendrán las líneas leídas de la imagen original; `acul`, `acuh` que acumularán los resultados parciales del cálculo; y cuatro registros temporales (en el caso del Frei-Chen el cuarto registro temporal se renombra a `sqrt2` que tendrá lo necesario para calcular la raíz).

Los algoritmos siguen una línea general, empiezan y terminan con lo definido en la convención C, se definen nombres de variables para los parámetros y se usan los registros generales de la siguiente manera:

- `eax`: Almacena las variables `xOrder` e `yOrder` que indican sobre qué derivada realizar el procesamiento.
- `ebx`: Contador para las filas.
- `ecx`: Contador para las columnas.
- `edx`: Ancho de la imagen.
- `esi`: Puntero a la imagen fuente.
- `edi`: Puntero a la imagen destino.

Luego para cada fila de la imagen se realiza el procesamiento sobre la derivada de `x` o de `y` según sea necesario procesando de a 14 píxeles, y antes de terminar la fila se revisa si se pueden calcular exactamente 14 píxeles, sino se retroceden los punteros para poder lograrlo y se pasa a la siguiente fila.

Los algoritmos de *Sobel*, *Prewitt* y *Frei-Chen* están estructurados de la misma manera. Cada uno tiene definidas sus propias *macros* denominadas `procesarLineaX`, `procesarLineaY`, `calcularX` y `calcularY`. De esta manera, el código de los tres filtros es bastante similar y para comprenderlos basta con entender la estructura básica de uno de ellos y luego los detalles particulares.

2.1. Sobel

2.1.1. Procesamiento en `x`

En Sobel en `x` la idea general es llamar a la *macro* `procesarLineaX`, luego avanzar el puntero fuente a la siguiente línea para llamar otra vez a la *macro* y de nuevo avanzar una línea y llamar a la *macro*. La *macro* se encargará de aplicar respectivamente la primer, segunda y tercer fila de la matríz de Sobel en `x`, en cada llamado, acumulando el resultado adecuadamente. Luego del proceso, se devuelve el puntero fuente a su posición original para poder continuar el procesamiento en `y`.

La *macro* `procesarLineaX` comienza cargando en los registros `srcl` y `srch` 16 píxeles de la imagen original, luego desempaqueta a ocho *words* adecuadamente con las instrucciones `punpcklbw` (*Unpack Low Bytes to Words* y `punpckhbw` (*Unpack High Bytes to Words*. Luego se trabajan los ocho píxeles menos significativos, que son los que contiene el registro `srcl`, usando la *macro* `calcularX` y un *define* `SAVE_RESULT` que actuará como condicional dentro de la *macro*.

Esta *macro* recibe como parámetro ocho píxeles a procesar, los guarda en el registro `tmp1`, que luego es *shifteado* 4 *bytes* a la izquierda, eliminando los dos primeros píxeles. A continuación se resta con la línea de píxeles originales (los recibidos como parámetro de la *macro*) logrando seis píxeles correctamente procesados y dos datos no significativos en los 4 bytes inferiores. Estos se eliminarán *shifteando* 4 *bytes* a la derecha, y luego 2 *bytes* a la izquierda para que el registro `tmp1` termine teniendo seis píxeles procesados y ceros en los 2 *bytes* del principio y del final. El condicional `SAVE_RESULT` se utiliza solo cuando es procesado el registro srcl, y guarda en tmp2 un resultado que luego será utilizado para calcular los píxeles centrales de los 14 originales.

Después de ejecutar la *macro* `calcularX`, se usa otro condicional (`X_LINE_2`) para duplicar el resultado obtenido sólo en el caso de estar procesando la segunda línea de la matríz Sobel en `x`. La idea es que, en la matríz Sobel en `x`, la primer y tercer líneas son iguales, mientras que la línea central tiene los mismos valores multiplicados por 2.

Los valores intermedios se acumulan en `acul` y `acuh`, y al final de la *macro* se procesan los dos píxeles centrales de los 14 píxeles originales que debían ser procesados. Esto se logra *shifteando* y acumulando adecuadamente sobre los registros `tmp1` y `tmp2`, luego el resultado se suma a los ya obtenidos en `acul` y `acuh`.

2.1.2. Procesamiento en y

La idea detrás del procesamiento de Sobel en `y` es básicamente la misma al procesamiento en `x`, sólo que la línea central no se procesa, ya que la matríz contiene sólo ceros. En cuanto a la implementación de la *macro* `procesarLineaY` se acumula el resultado de `calcularY` restando si se trata del proceso de la primer línea, o sumando si se trata de la tercera. Esto es porque la matríz es igual en la primer y tercer línea, salvo por el hecho de que la primer línea es negativa. En `calcularY` lo significativo es que se usa un registro para tener la línea a procesar multiplicada por dos, para lograr el patrón que necesita Sobel en `y`, y se usa el `tmp3` dentro del condicional `SAVE_RESULT` para tener una copia de `tmp1` que se usará en el procesamiento de los píxeles centrales.

Por último se empaquetan los datos y se guardan en la imagen resultante, repitiendo el algoritmo si es necesario, para los últimos píxeles de la fila.

3. Manual de usuario

3.1. Ayuda rápida

Uso:
`./bordes [opciones] [archivo]`

Opciones:
`-r#` Aplica el operador #
`-g` Modo gráfico

Operadores posibles:
1: Operador de Roberts
2: Operador de Prewitt
3: Operador de Sobel derivando por X
4: Operador de Sobel derivando por Y
5: Operador de Sobel derivando por X e Y

Si no se especifica un archivo de entrada, se usará 'lena.bmp'

3.2. Descripción

El programa se puede invocar en modo gráfico (`-g`) o directo (`-r#`) y opcionalmente una imagen. En modo directo, se leerá una imagen pasada como parámetro o la imagen por defecto (`lena.bmp`), se le aplicará el filtro seleccionado y se guardará con el nombre original mas un postfijo que indica que filtro fue aplicado y con la extensión original.

En modo gráfico, se puede abrir una imagen y aplicar los filtros en tiempo real, con las teclas del `1` al `5`, restaurar la imagen en escala de grises con la tecla `0`, y guardar el resultado actual con la tecla `s`. Se puede salir de este modo con la tecla `ESCAPE`.

El orden de los parámetros no importa, puede pasarse primero la dirección de la imagen seguida por el modo a usar o viceversa.

Tipos de imágenes soportados:

- `Windows bitmaps - BMP, DIB`
- `JPEG files - JPEG, JPG, JPE`
- `Portable Network Graphics - PNG`
- `Portable image format - PBM, PGM, PPM`
- `Sun rasters - SR, RAS`
- `TIFF files - TIFF, TIF`

(extraído de la documentación de la librería *OpenCv*)

3.3. Instrucciones de compilación

Dirigirse a la carpeta del código (`src/`) y ejecutar el comando `make`.