TP1: Detección de bordes

J. Enríquez - LU 36/08 - juanenriquez@gmail.com
N. Gleichgerrcht - 160/08 - nicog89@gmail.com
J. Luini 143/08 - marianosoy@gmail.com

Resumen

En el presente trabajo hacemos una investigación comparando las representaciones de punto flotante existentes con una implementación emulada por software que se propone ser más precisa que las disponibles con la desventaja de menor performance en relación a al implementada por hardware.
Para ello exploraremos la frontera del conjunto de Mandelbrot y veremos que sucede al acercarnos con las distintas opciones de manipulación de datos en punto flotante.

Keywords

Detección Bordes - ASM - Sobel

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://exactas.uba.ar

Índice

1. Introducción **2**

2. Desarrollo **3**
2.1. Programa 3
2.2. Algoritmos 3
2.3. Implementación 5

3. Discusión **6**

4. Conclusiones **7**

5. Apéndice A: Manual de usuario **8**

1. Introducción

La detección de bordes en una imagen consiste en hallar las zonas de la misma en donde el color cambia .ªbruptamente". Esta herramienta es utilizada tanto para compresión de archivos como para lograr efectos sobre las imágenes.

Los detectores de bordes calculan cómo cambia la intensidad de una imagen entorno a cada uno de sus pixeles. Esto se logra tomando la imagen en escala de grises y aplicando en cada punto un operador de derivación, que es una matriz de números cuyo producto interno con el entorno del punto refleja la tasa y dirección de la variación de intensidad. Esta técnica se llama convolución.

Existen muchas matrices de convolución, con diferentes dimensiones y coeficientes. En general detectan sólo saltos en una única dirección (horizontal, vertical u oblicua). Las de mayores dimensiones permiten un análisis más suave y menos preciso en el que se reducen los efectos del ruido".

En este trabajo hicimos un programa que permite aplicar distintos operadores de derivación a imágenes y visualizar los resultados en forma de una nueva imagen en escala de grises. Las funciones de procesamiento fueron hechas en lenguaje ensamblador, utilizando Medimos la performance de nuestras funciones, en cuanto a cantidad de clocks insumidos por el procesador, comparándolas entre sí y con una implementación de detección de bordes de la biblioteca de procesamiento de imágenes OpenCV (función cvSobel).

En este informe describiremos brevemente el programa realizado, discutiremos las cuestiones surgidas durante su desarrollo y expondremos los resultados y conclusiones extraídos.

2. Desarrollo

2.1. Programa

El programa que hicimos permite aplicar ciertos operadores de derivación a una imagen especificada por línea de comandos. Los operadores implementados son Roberts, Prewitt y Sobel. El primero tiene matrices de 2x2 y detecta bordes en diagonal. Los otros dos tienen matrices de 3x3 y detectan bordes verticales u horizontales.

Para cada operador solicitado el programa aplica la matriz correspondiente en X, luego en Y, y finalmente suma los resultados. Para el caso de Sobel, también permite aplicar convolución en una única dirección (X o Y por separado).

La imagen de entrada puede tener cualquier formato que la función cvLoadImage de OpenCV soporte; todos los formatos comunes están incluidos. Por cada operador utilizado se guarda una nueva imagen en escala de grises (con un sufijo en el nombre que indica el operador y la misma extensión y tamaño que la imagen de entrada) que grafica los bordes detectados.

Por último, el programa muestra por salida estádar la cantidad aproximada de clocks de procesador utilizados para la ejecución del algoritmo.

Las implementaciones están hechas en lenguaje ensamblador, aunque el programa también permite usar la implementación del operador Sobel de la biblioteca OpenCV (función cvSobel), así como también grabar la versión en escala de grises de la imagen de entrada. Por último, permite también aplicar una primera implementación del operador de Sobel que hicimos en C. Esta implementación fue realizada para ayudarnos a desarrollar el algoritmo.

Para cada punto de la imagen fuente el programa aplica la matriz correspondiente en X, satura el valor a 0 y 255, luego aplica la matriz en Y, satura también su resultado, y finalmente suma ambos coeficientes y satura la suma. Ese valor es grabado en la imagen generada.

Además de la función cvSobel usamos OpenCV para cargar y guardar imágenes en archivos y transformar imágenes a escala de grises.

2.2. Algoritmos

Si bien se trata de procedimientos bastante sencillos e intuitivos, hubo varios aspectos en los que surgieron dudas, problemas o diversidad de posibilidades.

En un píxel en donde la intensidad de la imagen crece hacia la derecha, diremos que se trata de un borde "hacia la derecha". Si en ese punto la imagen en cambio oscurece hacia la derecha, diremos que es un borde "hacia la izquierda".

Notar que, en el caso de Prewitt o Sobel, la matriz de convolución en X (que detecta bordes verticales) arroja un resultado positivo en los bordes hacia la derecha y negativo en los bordes hacia la izquierda. Asimismo la matriz de convolución en Y (que detecta bordes horizontales) produce números positivos en bordes hacia abajo y negativos en bordes hacia arriba. Con las matrices de

Roberts pasa lo mismo, salvo que los bordes son positivos hacia el noroeste para la matriz X o hacia el noreste para la Y, y negativos hacia el sudeste o sudoeste respectivamente.

Hay muchas formas de tratar los valores númerico de la derivación en X y en Y de cada píxel para graficarlos. Con la implementación que hicimos en C de Sobel probamos varias de ellas. Llamemos dx y dy a los valores obtenidos aplicando derivación en X y en Y respectivamente.

Una opción es graficar un punto cuya intensidad refleje el módulo del vector (dx, dy). De esta manera se obtiene una imagen oscura con los puntos borde más claros. La imagen muestra los bordes en todas las direcciones, aunque no permite distinguir en qué sentido van. Es decir, un borde hacia la derecha se grafica igual que uno hacia la izquierda. Para el cálculo del módulo puede usarse la norma 2 o bien la norma 1, bastante más rápida de computar.

Otra transformación típica es saturar los valores a un mínimo y a un máximo, lo que permite visualizar los bordes a grandes rasgos sin importar todos los matices de variación. Por ejemplo se los puede saturar a 0 y a 255. Hay que notar que haciendo esto se desprecian todos los bordes negativos, por lo tanto sólo se grafican los bordes que van en un sentido. Si en cambio se suma 128 al valor antes de saturarlo, se puede generar una imagen en la que predomina el gris y donde los bordes positivos se muestran como zonas claras y los bordes negativos como zonas oscuras.

Las variantes que implementamos nosotros son las siguientes: 1) norma 1 de (dx, dy) saturada a 255 2) dx + dy + 128 saturado a 0 y a 255 3) saturación de dx y dy a 0 y 255, suma, y saturación de la suma a 255

La idea inicial que teníamos, sugerida en la consigna del trabajo, era implementar la variante 1, que grafica bordes en cualquier dirección. Finalmente, sin embargo, decidimos usar la tercera variante, ya que sus resultados coinciden con los arrojados por la implementación de Sobel de OpenCV, contra la cual deseábamos comparar performance. Esto quiere decir que las imágenes generadas por nuestro programa son oscuras con zonas claras donde la imagen original tiene aumentos de intensidad hacia abajo o hacia la derecha (hacia el noroeste o hacia el noreste en el caso de Roberts). Los bordes que van en sentido contrario no se reflejan de ninguna manera así que es imposible saber dónde están o cuán abruptos son.

La variante 2 produce gráficos diferentes e interesantes de observar. Se pueden ver bordes en ambos sentidos, graficados de manera distinta. Los bordes positivos se muestran en blanco y los negativos en negro. Sin embargo no muestra bordes en todas las direcciones. De hecho, el efecto de sumar la derivada en X con la de Y es que los bordes se grafican en una única dirección (una diagonal). En los bordes perpendiculares a esta diagonal dx se anula con dy y por lo tanto estos bordes no se visualizan.

En el momento que optamos sumar las derivadas en X y en Y porque esa parecía ser la técnica usada por OpenCV, surgió una idea: en lugar de aplicar la matriz de X, luego la de Y y sumar los resultados, sería más eficiente (y más sencillo de implementar) hacer una sola pasada usando la matriz suma de las dos matrices. Este razonamiento es, además de intuitivo, matemáticamente

correcto.

Sin embargo las imágenes generadas haciendo ese "truco"no eran iguales a las de OpenCV. Algunos bordes, pese a ser "positivos", quedaban ocultos. Lo que sucedía era que, usando la matriz suma, por ejemplo una derivada positiva en X se puede anular con una derivada negativa en Y. Esto hace que se vean casi exclusivamente los bordes que apuntan en dirección sudeste.

Finalmente se concluyó que la técnica correcta (para generar dibujos como los de OpenCV) era saturar cada derivada parcial a 0 y a 255 y recién después sumarlas. Como la primera saturación elimina negativos, es imposible que las derivadas se cancelen, y se visualizan todos bordes que apuntan hacia abajo y/o hacia la derecha.

2.3. Implementación

Cuando hicimos la primera implementación (la del operador de Sobel hecha en C) nos topamos con un primer problema: la imagen resultante parecía mostrar los bordes, sin embargo estaba llena de puntos blancos, como si el algoritmo saturara ante el borde más suave. Finalmente resultó que el problema se solucionaba haciendo casts explícitos de los char obtenidos de las imágenes de OpenCV a unsigned char.

Dado que la correcta interpretación de las imágenes de OpenCV se realiza tratando los elementos del array imageData como unsigned char, todavía nos preguntamos por qué este campo del struct está declarado como char* (y no como unsigned char*).

Una vez que tuvimos la primera versión correcta hecha en lenguaje ensamblador (que aplicaba derivación en X mediante el operador de Sobel) nos tuvimos que embarcar en la tarea de mejorar el código. Si bien funcionaba correctamente abusaba de los accesos a memoria ya que para cada píxel estudiado salvaba y recuperaba información usando la pila (instrucciones push y pop). Esto provino del .ªpuro"por terminar la implementación, sin embargo muy pronto logramos que el algoritmo utilizara sólo los registros (salvo en unos pocos casos).

La versión final del trabajo todavía permite ejecutar el programa con esta implementación provisoria de Sobel. Como es posible verificar, la implementación que usa la pila es menos eficiente (toma aproximadamente el doble de tiempo).

3. Discusión

4. Conclusiones

5. Apéndice A: Manual de usuario

El programa realizado permite aplicar diferentes implementaciones de detección de bordes a imágenes. Tanto el nombre de las imagen fuente como los operadores se indican por línea de comandos de la siguiente manera (suponiendo que el ejecutable está en .^exe/bordes"):

```
$ exe/bordes {fuente} {destino} {operadores}
```

Donde:

* fuente: es la imagen de origen; puede tener cualquier formato soportado por la función cvLoadImage de OpenCV; los formatos más comunes están soportados;

* destino: es el nombre de las imágenes de salida (generadas por el programa) SIN INCLUIR LA EXTENSIÓN; los archivos generados tendrán ese nombre más un sufijo que indica el operador utilizado, más la extensión; la extensión, el formato y el tamaño son obtenidos de la imagen de entrada;

* operadores: una o más claves separadas por espacio; cada clave representa una implementación particular de detección de bordes; las claves posibles y sus significados son los que siguen:

CLAVE	OPERADOR	DIRECCION	IMPLEMENTACIÓN	SUFIJO
r1	Roberts	XY	Ensamblador	_asm_roberts
r2	Prewitt	XY	""	_asm_prewitt
r3	Sobel	X	""	_asm_sobelX
r4	Sobel	Y	""	_asm_sobelY
r5	Sobel	XY	""	_asm_sobelXY
cv3	Sobel	X	OpenCV	_cv_sobelX
cv4	Sobel	Y	""	_cv_sobelY
cv5	Sobel	XY	""	_cv_sobelXY
c3	Sobel	X	C	_c_sobelX
c4	Sobel	Y	""	_c_sobelY
c5	Sobel	XY	""	_c_sobelXY
push	Roberts	X	Ensamblador usando pila	_asm_roberts(push)
byn	Escala de grises			_byn

Por ejemplo, si tenemos el ejecutable en `"exe/bordes"` y una imagen en `"pics/lena.bmp"`, llamando al programa como

```
$ exe/bordes pics/lena.bmp pics/lena byn r1 r2 r3 r4 r5 cv3 cv4 cv5 c3 c4 c5 push
```

se aplicarán todas las implementaciones disponibles y se generarán los archivos `"pics/lena_asm_roberts.bmp", "pics/lena_asm_prewitt.bmp"`, etc... Además el programa mostrará por salida estándar la cantidad aproximada de clocks de procesador insumida por cada implementación.