¡!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"¿ ¡html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"¿
¡!– ViewVC :: http://www.viewvc.org/ –¿ ¡head¿ ¡title¿[unison] Log of /trunk/doc/moreverb.sty¡/title¿ ¡meta name="generator" content="ViewVC 1.1.6" /¿
¡link rel="shortcut icon" href="/viewvc-docroot/images/favicon.ico" /¿ ¡link rel="stylesheet" href="/viewvc-docroot/styles.css" type="text/css" /¿
¡/head¿ ¡body¿ ¡div class="vc$_n$*avheader*" >< *table* >< *tr* >< *td* >< *strong* >< *ahref* = "/*viewvc*" >< *spanclass* = "*pathdiv*" > / < /*span* >< /*a* >< *ahref* = "/*viewvc/unison/*" > [*unison*] < /*a* >< *spanclass* = "*pathdiv*" > / < /*span* >< *ahref* = "/*viewvc/unison/trunk/*" > *trunk* < /*a* >< *spanclass* = "*pathdiv*" > / < /*span* >< *ahref* = "/*viewvc/unison/trunk/doc/*" > *doc* < /*a* >< *spanclass* = "*pathdiv*" > / < /*span* > *moreverb.sty* < /*strong* >< /*td* >< *tdstyle* = "*text* − *align* : *right*;" >< /*td* >< /*tr* >< /*table* >< /*div* >< *divstyle* = "*float* : *right*; *padding* : 5*px*;" >< *ahref* = "*http* : //*www.viewvc.org/*"*title* = "*ViewVCHome*" >< *imgsrc* = "/*viewvc*−*docroot/images/viewvc*−*logo.png*"*alt* = "*ViewVClogotype*"*width* = "240"*height* = "70"/ >< /*a* >< /*div* >< *h*1 > *Logof/trunk/doc/moreverb.sty* < /*h*1 >
¡p style="margin:0;"¿
¡a href="/viewvc/unison/trunk/doc/"¿¡img src="/viewvc-docroot/images/back$_s$*mall.png*"*class* = "*vc*$_i$*con*"*alt* = "*ParentDirectory*"/ > *ParentDirectory* < /*a* >
— ¡a href="/viewvc/unison/trunk/doc/moreverb.sty?view=log"¿¡img src="/viewvc-docroot/images/log.png" class="vc$_i$*con*"*alt* = "*RevisionLog*"/ > *RevisionLog* < /*a* >
¡/p¿
¡hr /¿ ¡table class="auto"¿
¡tr¿ ¡td¿Links to HEAD:¡/td¿ ¡td¿ (¡a href="/viewvc/unison/trunk/doc/moreverb.sty?view=markup"¿view¡/a¿)
(¡a href="/viewvc/unison/trunk/doc/moreverb.sty?view=annotate"¿annotate¡/a¿)
¡/td¿ ¡/tr¿
¡tr¿ ¡td¿Sticky Revision:¡/td¿ ¡td¿¡form method="get" action="/viewvc/unison" style="display: inline"¿ ¡div style="display: inline"¿ ¡input type="hidden" name="orig$_p$*athtype*"*value* = "*FILE*"/ >< *inputtype* = "*hidden*"*name* = "*orig*$_v$*iew*"*value* = "*log*"/ >< *inputtype* = "*hidden*"*name* = "*orig*$_p$*ath*"*value* = "*trunk/doc/moreverb.sty*"/ >< *inputtype* = "*hidden*"*name* = "*view*"*value* = "*redirect*$_p$*athrev*"/ >
¡input type="text" name="pathrev" value="" size="6"/¿
¡input type="submit" value="Set" /¿ ¡/div¿ ¡/form¿
¡/td¿ ¡/tr¿ ¡/table¿
¡div¿ ¡hr /¿
¡a name="rev1"¿¡/a¿
Revision ¡a href="/viewvc/unison?view=revisionamp;revision=1"¿¡strong¿1¡/strong¿¡/a¿
-
(¡a href="/viewvc/unison/trunk/doc/moreverb.sty?revision=1amp;view=markup"¿view¡/a¿)
(¡a href="/viewvc/unison/trunk/doc/moreverb.sty?annotate=1"¿annotate¡/a¿)
- ¡a href="/viewvc/unison/trunk/doc/moreverb.sty?r1=1amp;view=log"¿[select for diffs]¡/a¿
¡br /¿
Added
¡em¿Mon Feb 7 03:09:03 2005 UTC¡/em¿ (8 years, 3 months ago) by ¡em¿tjim¡/em¿
¡br /¿File length: 5777 byte(s)
¡pre class="vc$_l$*og*" > *Initialimport* < /*pre* >< /*div* >
¡hr /¿ ¡p¿¡a name="diff"¿¡/a¿ This form allows you to request diffs between any two revisions of this file. For each of the two "sides" of the diff,
enter a numeric revision.
¡/p¿ ¡form method="get" action="/viewvc/unison/trunk/doc/moreverb.sty" id="diff$_s$*elect*" >< *tablecellpadding* = "2"*cellspacing* = "0"*class* = "*auto*" >< *tr* >< *td* > *nbsp*; < /*td* >< *td* >< *inputtype* = "*hidden*"*name* = "*view*"*value* = "*diff*"/ > *Diffsbetween*
¡input type="text" size="12" name="r1" value="1" /¿
and
¡input type="text" size="12" name="r2" value="1" /¿
¡/td¿ ¡/tr¿ ¡tr¿ ¡td¿nbsp;¡/td¿ ¡td¿ Type of Diff should be a ¡select name="diff$_f$*ormat*"*onchange* =

Integrante	LU	Correo electrónico
Facundo Bálsamo	xxx/xx	balsa@mo.com.ar
Pablo González	xxx/xx	pablo@gonzalez.com.ar
Lasso, Nicolás	763/10	lasso.nico@gmail.com

[Breve introduccion de no mas de 200 palabras]

palabraClave1 **palabraClave2** **palabraClave3** **palabraClave4**

1 Introducción Teórica

El siguiente trabajo práctico tiene como finalidad el uso de la programación SIMD orientado al procesamiento de imágenes. Para ello se aplican filtros para procesar dichas imágenes mediante el uso tanto de lenguaje ensamblador como lenguaje C.

La finalidad de la implementación en dos lenguajes es medir la velocidad de ejecucion y cantidad de cilos que son necesarios al aplicar los filtros para cada uno y así compararlos.

2 Desarrollo

En esta sección se explica detalladamente cada uno de los filtros útilizados para procesar las imágenes:

2.1 Recortar

El filtro Recortar únicamente requiere copiar una cantidad tam de píxeles de las esquinas de la imagen fuente para generar la imagen destino con las esquinas invertidas.

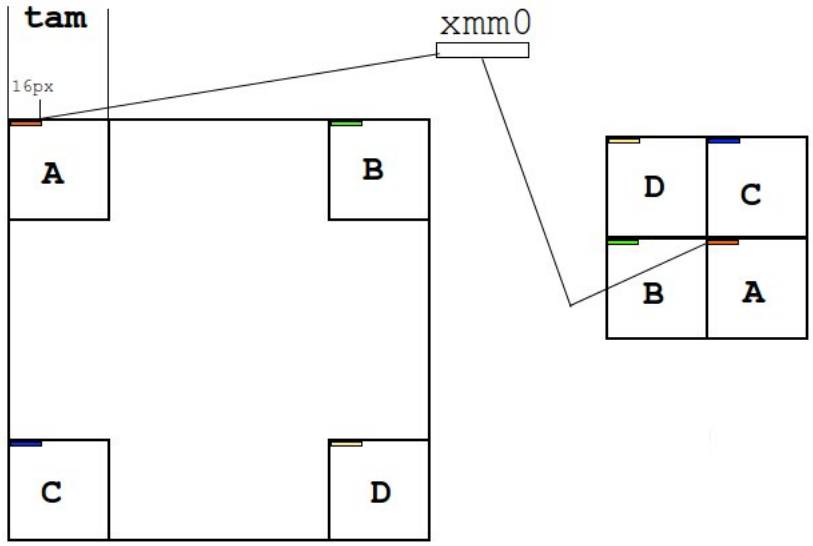
2.1.1 Implementación en C:

Con dos ciclos anidados se recorre el área de tam x tam en cada esquina de la imagen fuente y se escribe en la esquina opuesta de la imagen destino.

2.1.2 Implementación en asm

También se realiza con dos ciclos anidados, pero trayendo de a 16 píxeles por vez.

En cada iteracin se traen 16 píxeles de la parte A a xmm0 con la operación MOVUPS y luego se copian a la parte D, lo mismo con 16 píxeles de B en C, de C en B y de D en A. Esto puede verse en el siguiente gráfico.



2.1.3 Resultados

Dado que en este filtro no se modifican los píxeles, la única diferencia debería ser en los accesos a memoria. Los ciclos en ambas implementaciones hacen 8 accesos a memoria, 4 lecturas y 4 escrituras, pero la implementación en ASM accede de a 16 bytes vs 1 de la implementación en C.

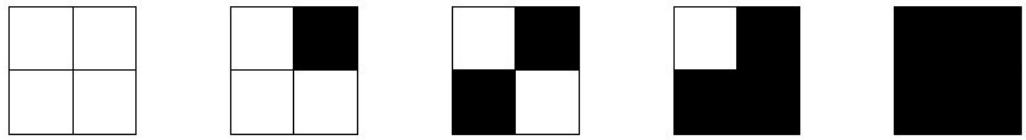
Por lo tanto la mplementacin en C debería ser aproximadamente 16 veces más rápida.

Medición	Implementación en C	Implementación en asm
Ciclos en Total	422596036	24781328
Ciclos por llamada	422596.031	24781.328

En promedio, cada ejecución de la implementación en C tarda 422 596 ciclos y cada ejecución de ASM 24 781, es decir, 17 veces más rápido, corroborando nuestra hipótesis.

2.2 Halftone

El filtro Halftone parte a la imagen original en bloques de 2x2 píxeles, se obtiene t como la sumatoria del valor de cada pixel y genera un bloque del mismo tamaño en la imagen destino que cumpla:



820 <= t 615 <= t < 820 410 <= t < 615 205 <= t < 410 t < 205

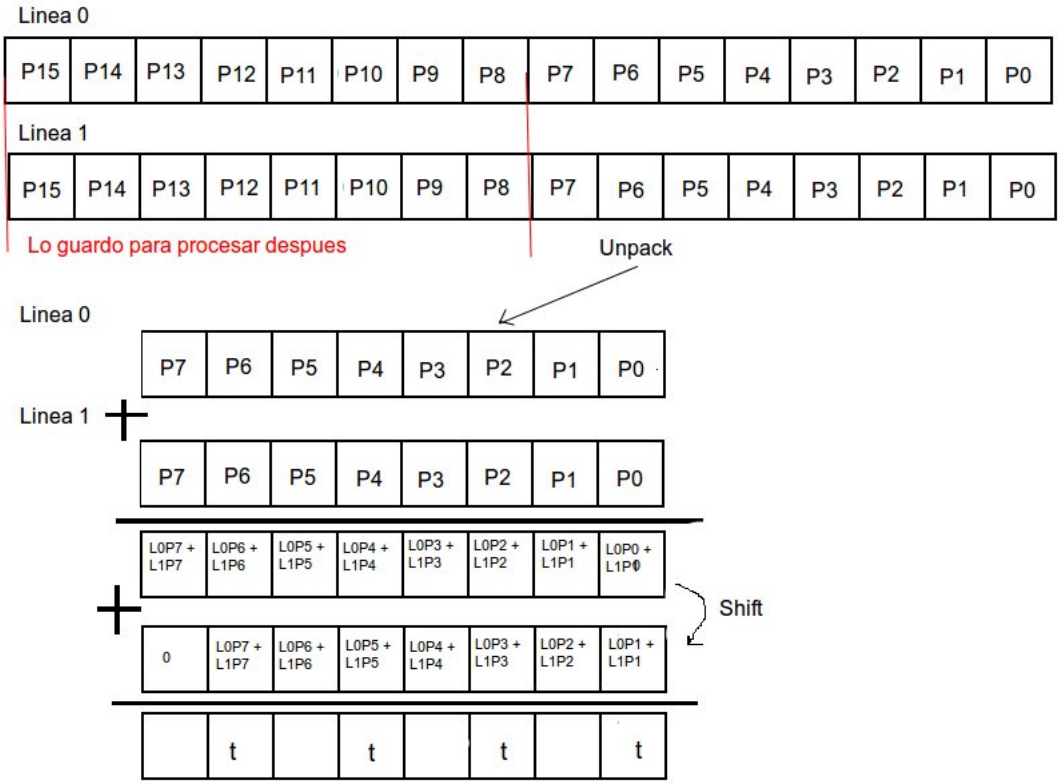
Esto implica:
El pixel de arriba a la izquierda es blanco si t >= 205.
El pixel de arriba a la derecha es blanco si t >= 820.
El pixel de abajo a la izquierda es blanco si t >= 615.
El pixel de abajo a la derecha es blanco si t >= 410.
Si el ancho y/o alto de la imagen es impar, se descarta la última fila y/o columna, según corresponda.

2.2.1 Implementación en C

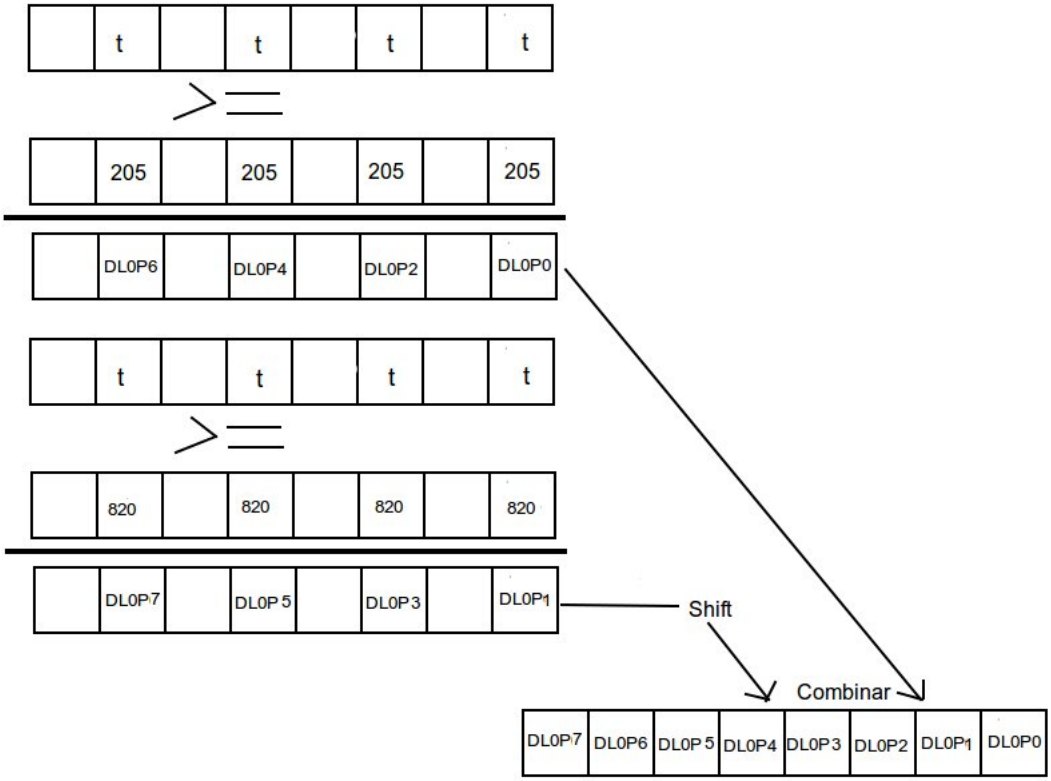
Con dos ciclos anidados se recorre de 2 en 2 la imagen original.
Se calcula t con la suma de de los valores de los 4 píxeles y luego se setean los píxeles en blanco o negro según corresponda por el valor de t.

2.2.2 Implementación en asm

También se realizan dos ciclos alineados, trayendo cada vez de a 32 píxeles.
Se traen 16 píxeles de la primera línea a xmm0 y 16 de la segunda línea a xmm1.
Como para calcular t se necesita un espacio de más de 1 byte (t puede ser mayor a 255), se desempaquetan y se trabaja primero con los 8 bytes menos significativos, guardando el resto.
Se suman los 2 registros verticalmente, luego se shiftea un registro para acomodar y volver a sumar. Esto da como resultado el valor t para cada uno de los 4 bloques.



Se compara por mayor o igual en un registro distinto para cada uno de los valores límite.
Luego, ya que el valor que va a cada pixel es el mismo que el resultado de la comparación (todos 1s o todos 0s), simplemente se ordenan los resultados en los primeros 8 bytes de cada registro.



Este proceso se repite con los 8 bytes más significativos que guardamos al principio. Estos resultados se shiftean 8 bytes a la izquierda y combinan (con un por) con los primeros resultados, dándonos 2 registros con 16 bytes procesados cada uno, que se copian a la memoria de la imagen destino.

En el caso que el ancho no sea múltiplo de 16, antes de entrar al ciclo se calcula el resto (width

2.2.3 Resultados

En la implementación en C, para procesar un bloque de 4 píxeles se requieren 8 accesos a memoria (4 lecturas y 4 escrituras), en cambio en la implementación en ASM, se procesa un bloque de 32 píxeles con tan sólo 4 accesos a memoria.

Con esos datos y, teniendo en cuenta que el acceso a memoria es mucho más lenta que cualquier otra operación que se aplique en el filtro, la implementación en ASM debería ser 16 veces mś rápida que la implementación en C.

Medición	Implementación en C	Implementación en asm
Ciclos en Total	3229822124	230612631
Ciclos por llamada	3229822.250	230612.625

En promedio, cada ejecución de la implementación en C tarda 3 229 822 ciclos y cada ejecución de ASM tarda 230 612, es decir, 14 veces ms rápido, corroborando nuestra hipótesis.

2.3 Rotar

Este filtro consiste en rotar la imagen 45 en sentido antihorario. Lo que provoca que algunos pixels se pierdan mientras que otros se anulen (es decir, se vuelvan 0), otros intercambien lugares y por último el centro rote sobre si mismo.

Un caso que resulta párticular con este filtro y es que su pérformance casi no varía respecto de

assembler o C. Esto sucede porque hay que tratar cada pixel (o byte si nos referimos al código) de manera párticular, haciendo que no sea posible paralelizar tanto como sucede con otros filtros.

2.3.1 Algoritmo

1. Crear las variables/Registros para almacenar los datos y las constantes a utilizar.
2. Loop:
   (a) Crear los respectivos u y v que se calculan dependiendo de la posicion de mi pixel de entrada.
   u = $c_x + \frac{\sqrt{2}}{2}(x - c_x) - \frac{\sqrt{2}}{2}(y - c_y)$
   $v = c_y + \frac{\sqrt{2}}{2}(x - c_x) - \frac{\sqrt{2}}{2}(y - c_y)$
   $c_x = [I_{src\_width}/2]$
   $c_y = [I_{src\_height}/2]$

   (b) Analizo si u y v cumplen los requisitos de para realizar el intercambio de pixels.
   $0 <= u < I_{src\_width}$ y $0 <= v < I_{src\_height}$

   (c) Si lo cumple entonces el nuevo valor de $I_{dst}[x][y] = I_{src}[u][v]$.

   (d) De lo contrario, $I_{dst}[x][y] = 0$.[1]

3. Finalmente devuelve a matriz rotada.

2.3.2 Resultado

Como resultado de este filtro remarcamos la poca diferencia entre C y assembler con respecto a lo que sucede en los demás filtros. Como datos, sabemos que:

Medición	Implementación en C
Ciclos Totales	1645260830
Ciclos Por Llamada	16452608.000

[1] ver codigo de la implementación para un mejor entendimiento.

3 Resultado

4 Conclusiones

5 Apéndices

6 Referencias

Indice