Trabajo Práctico Obligatorio
Universidad Nacional del Comahue
Facultad de Informática
Arquitectura y Organización de Computadoras

Braian Ledantes

Legajo: FAI-1686

1. Introducción

En este trabajo se deben implementar tres filtros sobre una imagen. Las imágenes están representadas como un arreglo de pixeles, donde cada pixel a su vez está formado por tres enteros sin signo de 8 bits cuyos valores representan la intensidad de las componentes roja, verde y azul del pixel, con el valor cero indicando que no se emite luz de ese color, y un valor de 255 indicando que se emite la cantidad máxima de ese color
Ejemplo: la siguiente figura

es una figura de 4x2 (cuatro columnas y dos filas), y en la memoria de la computadora estará representado por el siguiente arreglo de pixeles (aquí cada valor está expresado en base 10, pero claro está, en la memoria de la computadora se almacenan en base 2):

0	255	0	255	0	0	255	255	255	0	0	255	0	0	0	255	255	255	0	0	255	120	120	120

Las funciones que aplican los filtros se deben implementar en el lenguaje ensamblador de la arquitectura MIPS. El primer parámetro de todas las funciones es la dirección de memoria donde se encuentra el arreglo de pixeles de la imagen (es la dirección del primer elemento del arreglo). Todas las imágenes tienen una resolución de 640x480 pixeles.

Se realizaron los siguientes filtros:

1. Filtrar rojo: modificar la imagen de manera que sólo quede visible el componente rojo. Para realizar esto debe colocar en cero los componentes verde y azul de cada pixel de la imagen. filtro_rojo(unsigned byte *pixeles);

2. Scanlines: para simular el efecto a veces visible en un monitor de rayos catódicos, en cada línea impar reducir la intensidad de cada pixel a la mitad. filtro_scanlines(unsigned byte *pixeles);

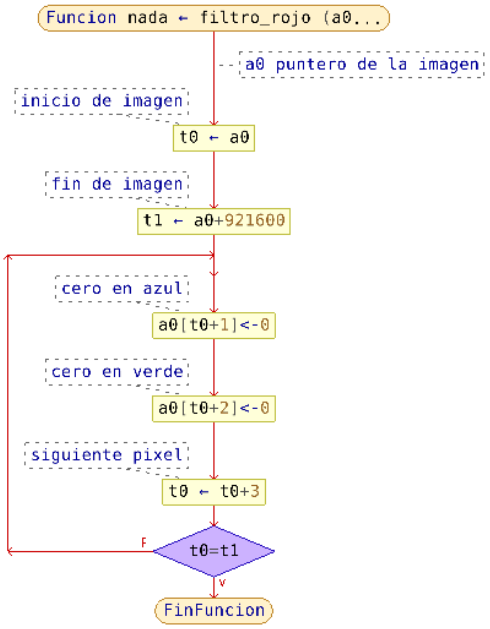
3. Restablecer imagen: sobrescribir todos los pixeles de la imagen por un color pasado por parámetro. El color se pasa en representación porcentual como tres enteros de 8 bits cuyos valores están entre cero y cien, representando el valor porcentual de luminosidad de las componentes roja, verde y azul . Debe mapear el rango del parámetro al rango de representación de los enteros de 8 bits sin signo. Por ejemplo, si los parámetros son 0, 100, y 20, los componentes de todos los pixeles deben quedar como rojo=0, verde=255 y azul=51. filtro_reset(unsigned byte *pixeles, byte rojo, byte verde, byte azul);

4. Escala de grises: todos los componentes de cada pixel deben tomar como valor el promedio de las tres componentes originales. filtro_gris(unsigned byte *pixeles);

2. Desarrollo

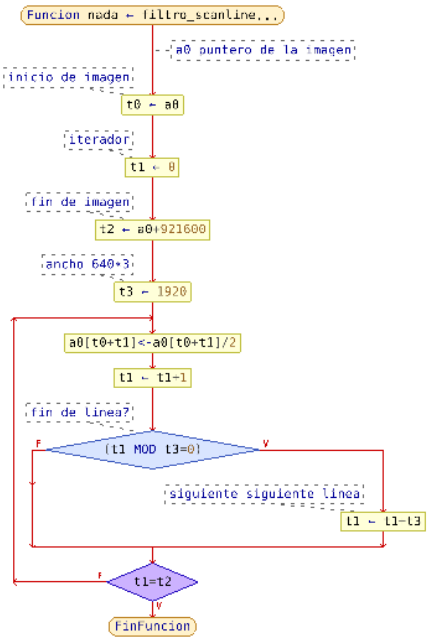
El algoritmo del filtro rojo se utiliza básicamente tres registros, t0 donde guarda la posición de memoria donde inicia la imagen, el registro t1 donde termina, (ancho * alto * 3) + t0 y t3 que guarda el valor cero como byte; se itera sobre t0 cada para ir al siguiente pixel y se modifica la posición de los colores verde y azul hasta que t0 llegue a t1.
El uso de la pila en este algoritmo no es de ninguna utilidad.

```
15  filtro_rojo:
16      addiu   $sp, $sp, -24
17      sw      $ra, 20($sp)
18      sw      $fp, 16($sp)
19
20      move    $t0, $a0            #pos donde inicia
21      li      $s0, 921600
22      add     $t1, $t0, $s0       #pos donde termina
23      li      $t2, 0x00           #cero
24
25  loopRojo:
26      sb      $t2, 1($t0)         #guardo cero
27      sb      $t2, 2($t0)         #guardo cero
28      addi    $t0, $t0, 3         #al sig pixel
29      bne     $t0, $t1, loopRojo  #mientras no llego al final
30
31      lw      $ra, 20($sp)
32      lw      $fp, 16($sp)
33      addiu   $sp, $sp, 24
34      jr      $ra
```



Para el filtro scanlines guardo en el registro t0 la posición de memoria donde inicia la imagen, en t1 guardo el valor cero para funcionar como iterador, t2 la cantidad de bytes que ocupa la imagen y en t3 el ancho de la imagen por los 3 colores.
Al entrar en el bucle, identificado por la etiqueta ‘loop_scanlines’, se guarda con la instrucción sb el byte que representa un componente del pixel dividido por dos, se aumenta el iterador (t1) en uno y si llego al final de esa linea de la imagen (t1 % t3 == 0), salta a la etiqueta ‘salto’ donde al iterador se le aumenta el ancho de la imagen (t3) de la misma para que no modifique la siguiente linea y así sucesivamente hasta que el iterador llegue a la posición de memoria donde termina la imagen.

```
36  filtro_scanlines:
37      move    $t0, $a0            #pos donde inicia img
38      li      $t1, 0x00           #iterador
39      li      $t2, 921600         #fin iterador
40      li      $t3, 1920           #ancho 640x3
41
42  loop_scanlines:
43      add     $t4, $t1, $t0       #tomo la pos del color
44      lbu     $t5, 0($t4)         #tomo el color
45      div     $t5, $t5,2          #divido el color
46      sb      $t5, 0($t4)         #guardo en memoria color/2
47      add     $t1, $t1, 1         #iterador++
48
49      rem     $t4, $t1, $t3       #i mod ancho
50      bne     $t4, $zero, next
51  salto:
52      add     $t1,$t1,$t3         #iterador += ancho
53  next:
54      bne     $t1, $t2, loop_scanlines
55      jr      $ra
```



El filtro reset en el registro t0 se guarda la posición de memoria donde inicia la imagen, en t1 donde termina, en s0 se guarda el valor 255 que representa la intensidad máxima del componente y en s1 el valor 100. Para los colores, como por parámetro (los registros a1, a2 y a3) el valor del componente es un porcentaje, por regla de 3 se calcula el valor del componente en intensidad de color. En t2 se guarda el resultado de la operación (a1*s0/s1), lo mismo para el verde t3=(a2*s0/s1) y para el azul t4=(a3*s0/s1). Luego en un bucle marcado por la etiqueta 'loop_reset' se modifican todos los pixeles con el valor correspondiente en t2, t3, t4 a cada componente, a t0 se le aumenta en 3 hasta que sea igual a t1 (posición de memoria final).
La pila en este método tampoco tiene utilidad.

```
60  filtro_reset:
61      move    $t0, $a0            #pos donde inicia
62      li      $s0, 921600
63      add     $t1, $t0, $s0       #pos donde termina
64      move    $t2, $a1            #rojo
65      move    $t3, $a2            #verde
66      move    $t4, $a3            #azul
67      li      $s0, 255            #255
68      li      $s1, 100            #100
69
70      mul     $t2, $t2, $s0       #rojo*255/100
71      div     $t2, $t2, $s1
72      mul     $t3, $t3, $s0       #verde*255/100
73      div     $t3, $t3, $s1
74      mul     $t4, $t4, $s0       #azul*255/100
75      div     $t4, $t4, $s1
76
77      loop_reset:
78      sb      $t2, 0($t0)         #guardo el rojo
79      sb      $t3, 1($t0)         #guardo el verde
80      sb      $t4, 2($t0)         #guardo el azul
81      addi    $t0, $t0, 3         #al sig pixel
82      bne     $t0, $t1, loop_reset #mientras no llego al final
83      jr      $ra
```

El filtro gris guarda en el registro t0 la posición donde inicia la imagen en memoria y t1 donde termina, se itera sobre t0 hasta que t0 sea igual a t1. Cuando llega a la etiqueta 'loop_gris' se guardan en t2, t3 y t4 los componentes de el pixel sobre el que está posicionado el iterador (t0) para calcular el promedio y guardarlo en el registro t2 y luego guardar ese resultado en todos los componentes de ese pixel, y se le aumenta a t0 3 para que itere sobre el siguiente pixel. Logrando así que la imagen quede en blanco y negro.

```
85  filtro_gris:
86      move    $t0, $a0            #pos donde inicia
87      li      $s0, 921600
88      add     $t1, $t0, $s0       #pos donde termina
89      li      $s0, 3              #3
90
91  loop_gris:
92      lbu     $t2, 0($t0)         #rojo
93      lbu     $t3, 1($t0)         #verde
94      lbu     $t4, 2($t0)         #azul
95
96      add     $t2, $t2, $t3       #(rojo+verde+azul)/3
97      add     $t2, $t2, $t4
98      div     $t2, $t2, $s0
99
100     sb      $t2, 0($t0)         #guardo el promedio
101     sb      $t2, 1($t0)
102     sb      $t2, 2($t0)
103
104     addi    $t0, $t0, 3         #al sig pixel
105     bne     $t0, $t1, loop_gris #mientras no llego al final
106     jr      $ra
```

El filtro barras guarda en el registro t0 el iterador, en t1 el ancho de la imagen en bytes, y al ancho de la barra enviada por parámetro se multiplica por tres, que son los componentes del pixel. En el loop lo primero que hace la función es verificar si el modulo del iterador con el parámetro es cero, de ser falso al iterador le suma el ancho para que no pinte la barra en esos pixeles, luego le suma a la posición de la imagen en memoria el iterador y lo guarda en el registro t5, se setean los colores y al iterador se le suma 3 para pasar al siguiente pixel. Todo esto se repite en el loop mientras el iterador sea distinto del tamaño de la imagen.

```
108  filtro_barras:
109      li      $t0, 0x00           #iterador
110      li      $t1, 921600         #tamanyo en bytes de la imagen
111      mul     $t2, $a1, 3         #ancho de barra * 3 componentes
112      li      $t3, 255            #intensidad del rojo y azul
113      li      $t4, 0x00           #intensidad del verde
114
115  loop_barras:
116      rem     $t5, $t0, $t2       #iterador mod ancho == 0
117      bne     $t5, $zero, cont
118      add     $t0, $t0, $t2       #sumar al iterador el ancho
119
120  cont:
121      add     $t5, $a0, $t0       #pos del pixel
122      sb      $t3, 0($t5)         #seteo el rojo
123      sb      $t4, 1($t5)         #seteo el verde
124      sb      $t3, 2($t5)         #seteo el azul
125      add     $t0, $t0, 3         #iterador + 3
126      blt     $t0, $t1, loop_barras #mientras no llego al final
127      jr      $ra
```

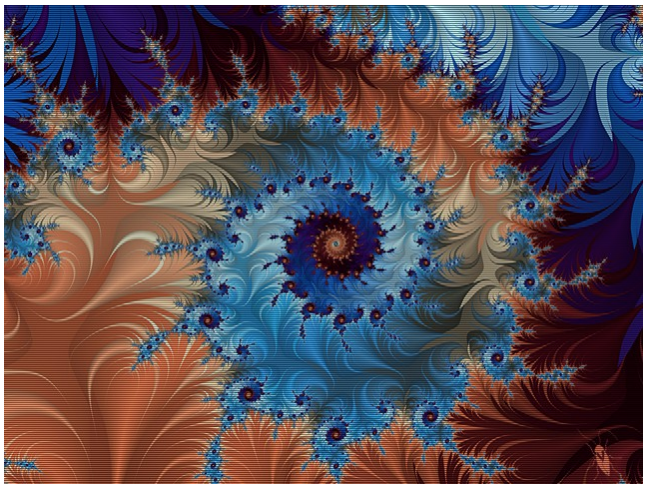
3. Conclusión

Los filtros mostrados anteriormente se le aplica a la siguiente imagen:

El resultado aplicando los filtros son los siguientes:

filtro rojo:

filtro_scanlines:



filtro_reset:

filtro_gris:

filtro_barras: