**Trabajo Práctico N°2**

*Procesamiento de Imágenes:*

*Etapa 1: Interpolación*

**Materia:**

* Organización del computador II (OC2)

**Docente:**

* Carlos Holtman (choltman@ungs.edu.ar)

**Alumnos:**

* Gonzalo Pesado (gonzapesa@gmail.com)
  + LEG: 38990987 / 2014
* Lucas Vargas ([lucasjv92@gmail.com](mailto:lucasjv92@gmail.com))
  + LEG: 37369589 / 2014

**Año 2017**

**Desarrollo del proyecto**

Como primer etapa del Trabajo Práctico, nos focalizamos en la creación de un programa capaz de interpolar linealmente 2 imágenes indicando qué proporción de valor de cada canal de cada par de pixeles de las dos imágenes para luego guardarlo en un archivo nuevo.

El software está desarrollado en 2 lenguajes de programación: Assembler de Intel IA32 (Lenguaje de código máquina) y C. El entorno de desarrollo y ejecución se realizó bajo sistema operativo Windows de 32 bits.

A su vez, el software debe ejecutarse colocando en orden los siguientes argumentos: el nombre de archivo de la primer imagen, el nombre de archivo de la segunda imagen, el alto de la imagen, el ancho, el valor de proporción (Que debe estar entre 0 y 1), y el nombre del archivo donde se guardará la nueva imagen.

El programa a su vez también cuenta con un .bat el cual compila el código fuente (“Interpolar.c”) y sus respectivos headers, junto con el OBJ que crea NASM del archivo que posee todas las funciones de Assembler “funcionesASM.asm”). Este .bat mostrará al usuario las rutas de NASM y GCC que utilizará para compilar, en caso de que no sean las rutas correctas, se deberá cambiar la ruta directamente del .bat a la ruta correcta.

Por último, el .bat convierte 2 imágenes de prueba con extensión .png a .rgb usando el programa **GraphicsMagick** para que luego el programa compilado las lea, y luego se visualice la imagen final con GraphicsMagick.

Con respecto al corazón de las funciones de Assembler de Intel IA32, el programa se basa en 2 pasos los cuales son:

1) **cargarPatrones**: En él se cargaran todas las máscaras a los registros para que las operaciones entre vectores sean mucho más rápidas, se asume que llegan a entrar vectores de 4 colores RGB, por lo que estas máscaras están acordes a los mismos. Dentro de esta también se realiza el carga p y cargar1menosP para tener dichos valores en un vector de 4 posiciones, para ya directamente realizar los cálculos en el mismo registro.

2) **ciclar**:

2.A) **llenarVectorIMG1/IMG2:** Carga las 4 primeras posiciones de los punteros de ambas IMGs a un registro XMM (con respecto al puntero ecx utilizado como contador.

2.B) **convertirINT\_PF**: Realiza la conversión a PF de los vectores de ambas imágenes para poder operar con mayor precisión y realizar los cálculos de la interpolación que fueron explicados en las consignas del TP.

2.C) **operarConPF**: Mediante operaciones entre los registros cargados previamente, se calcula P x V1 + (1-p) x V2 para determinar el vector que deberia estar en la IMG resultado.

2.D) **recuperarResultado**: Realiza la conversión del vector en PF al vector de formato entero.

2.E) **copiarResultado**: Escribe en el puntero de la imagen resultado el vector calculado previamente.

2.F) **operarConLosRestantes**: Fija el puntero a los próximos 4 valores de los vectores de IMG 1 e IMG 2, si todavía no se llego al tamaño total pasado por parámetro, seguirá realizando el ciclo con las próximas 4 posiciones de los vectores de IMG1 e IMG2.

Como herramientas para desarrollar el mismo se usaron los siguientes:

* Editores: Sublime, Notepad++, Notepad, Wordpad.
* IDE: SASM para testear código de assembler, y Codeblocks para probar código de C.
* Compiladores: NASM para compilar los códigos .asm, y GCC para compilar el código de C y llamar a los objetos de código creados anteriormente por NASM para vincular las funciones de Assembler llamadas desde C.
* GraphicsMagick para exportar las imágenes a un archivo .rgb (vector de bytes) y visualizar las imágenes con ese formato.

**Código Fuente: C**

**Interpolar.c**

1. #include <stdio.h>
2. #include <stdlib.h>
3. #include "Funciones.h"
5. **void** interpolar(**int** longitud, unsigned **char**\* img1, unsigned **char**\* img2,unsigned **char**\* resultado,**float** p);
7. **int** main(**int** argc, **char** \* argv[])
8. {
9. **if**(argc != 7)
10. {
11. printf("Los argumentos del programa deben ser: \n");
12. printf("imagen1.rgb imagen2.rgb filas columnas p(entre 0 y 1) resultado.rgb");
13. **return** 0;
14. }
15. ///argv[0] Ejecutable (Interpolar)
16. ///argv[1] Imagen1 rgb
17. ///argv[2] Imagen2 rgb
18. ///argv[3] filas
19. ///argv[4] columnas
20. ///argv[5] Proporcion de interpolacion (FLoat entre 0 y 1)
21. ///argv[6] ImagenResultado rgb
22. **char** \*nombreImagen1 = argv[1];
23. **char** \*nombreImagen2 = argv[2];
24. **int** filas = atoi(argv[3]);
25. **int** columnas = atoi(argv[4]);
26. **float** p = atof(argv[5]);
27. **char** \*nombreImagenResultado = argv[6];
29. unsigned **char** \*img1;
30. unsigned **char** \*img2;
31. unsigned **char** \*resultado;
33. **int** cantidadPixelPorPunto = 3;
34. **int** longitud = filas \* columnas \* cantidadPixelPorPunto;
36. img1 = malloc (longitud);
37. img2 = malloc (longitud);
38. resultado = malloc (longitud);

41. leer\_rgb(nombreImagen1,img1,filas,columnas);
43. leer\_rgb(nombreImagen2,img2,filas,columnas);
45. interpolar(longitud,img1,img2,resultado,p);
47. escribir\_rgb(nombreImagenResultado,resultado,filas,columnas);
49. free(img1);
50. free(img2);
51. free(resultado);
53. **return** 0;
54. }

**Funciones.h**

1. #ifndef FUNCIONES\_H\_INCLUDED
2. #define FUNCIONES\_H\_INCLUDED
4. **void** leer\_rgb(**char** \*archivo, unsigned **char** \*buffer, **int** filas, **int** columnas)
5. {
6. **FILE** \*fp;
7. fp = fopen(archivo, "rb");
8. fread (buffer, filas \* columnas \* 3, 1, fp);
9. fclose (fp);
10. }
12. **void** escribir\_rgb(**char** \*archivo, unsigned **char** \*buffer, **int** filas, **int** columnas)
13. {
14. **FILE** \*fpNuevo;
15. fpNuevo = fopen(archivo,"wb");
16. fwrite(buffer,filas\*columnas\*3,1,fpNuevo);
17. fclose(fpNuevo);
18. }
20. **void** separar\_rgb(unsigned **char** \*rgb, **int** cantidad, unsigned **char** \*r, unsigned **char** \*g, unsigned **char** \*b)
21. {
22. **int** i;
23. **int** cont = 0;
24. **for**(i = 0; i < cantidad; i+=3)
25. {
26. r[cont] = rgb[i];
27. g[cont] = rgb[i+1];
28. b[cont] = rgb[i+2];
29. cont++;
30. }
31. }
33. **void** combinar\_rgb(unsigned **char** \*r, unsigned **char** \*g, unsigned **char** \*b, **int** cantidad, unsigned **char** \*rgb)
34. {
35. **int** i;
36. **int** cont = 0;
37. **for**(i = 0; i < cantidad; i++)
38. {
39. rgb[cont] = r[i];
40. cont++;
41. rgb[cont] = g[i];
42. cont++;
43. rgb[cont] = b[i];
44. cont++;
45. }
46. }
48. **void** mostrarRGB(unsigned **char** \*vector, **int** cantidad)
49. {
50. printf("Mostrando vector\n");
51. **int** i;
52. **for**(i = 0; i < cantidad; i++)
53. {
54. printf("%d",vector[i]);
55. }
56. printf("\n");
57. }
58. #endif // FUNCIONES\_H\_INCLUDED

**Código Fuente: Assembler**

global \_interpolar

section .data

align 16 ;PARA ALINEAR LAS MASCARAS

PATRON\_INT\_PF db 3,255,255,255,2,255,255,255,1,255,255,255,0,255,255,255

PATRON\_PF\_INT db 12,8,4,0,255,255,255,255,255,255,255,255,255,255,255,255

PATRON\_CARGA\_P db 0,1,2,3,0,1,2,3,0,1,2,3,0,1,2,3

unoXMM dd 1.0,1.0,1.0,1.0

section .text

;1RA ETAPA

\_interpolar:

**push** ebp

**mov** ebp,esp

;EBP +0 PUSH EBP RECIEN

;EBP +4 DIR RETORNO

;EBP +8 FLOAT P

;EBP +12 PUNTERO CHAR RESULTADO

;EBP +16 PUNTERO CHAR IMG2

;EBP +20 PUNTERO CHAR IMG1

;EBP +24 INT TAMA�O

;EBP +4 DIR RETORNO

;EBP +8 INT TAMA�O

;EBP +12 PUNTERO CHAR IMG 1

;EBP +16 PUNTERO CHAR IMG 2

;EBP +20 PUNTERO CHAR RESULTADO

;EBP +24 FLOAT P

**mov** edx,[EBP + 8] ;tama�o

**mov** ecx,0 ;contador

cargarPatrones:

movdqu xmm5,[PATRON\_INT\_PF] ;MASCARA DE VECTOR - PF

movdqu xmm6,[PATRON\_PF\_INT] ;MASCARA DE PF - VECTOR

cargarP:

movdqu xmm4,[PATRON\_CARGA\_P];MASCARA CARGA P

**movd** xmm2,[EBP + 24] ;PASO P A REGISTRO

pshufb xmm2,xmm4 ;ACOMODO P

cargar1menosP:

movdqu xmm3,[unoXMM] ;CARGO 1

**subps** xmm3,xmm2 ;(1-P)

ciclar:

llenarVectorIMG1:

**mov** eax,[EBP + 12] ;muevo puntero de IMG1

**movd** xmm0,**dword** [eax + ecx] ;copio vector IMG1

pshufb xmm0, xmm5 ;ACOMODO VECTOR IMG 1

llenarVectorIMG2:

**mov** eax,[EBP + 16] ;MUEVO PUNTERO DE IMG2

**movd** xmm1,**dword** [eax + ecx] ;COPIO VECTOR IMG2

pshufb xmm1,xmm5 ;ACOMODO VECTOR IMG 2

convertirINT\_PF:

;CONVIERTO VECTOR DE IMG 1 EN PF

cvtdq2ps xmm0,xmm0;CONVIERTO VECTOR - PF

;CONVIERTO VECTOR DE IMG 2 EN PF

cvtdq2ps xmm1,xmm1;CONVIERTO VECT - PF

operarConPF:

;XMM0 = VECTOR IMG 1

;XMM1 = VECTOR IMG 2

;XMM2 = 1

;XMM3 = (1-P)

**mulps** xmm0, xmm2 ;P x v1

**mulps** xmm1, xmm3;(1-P)x V2

**addps** xmm0,xmm1 ;P x V1 + (1-p) x V2

;AHORA EN XMM0 ESTA EL RESULTADO

recuperarResultado:

;TRASFORMAR PF A VECTOR

cvtps2dq xmm0,xmm0 ;CONVIERTE EL RESULTADO A VECTOR

;ahora hay que volver a armarlos

;ya que quedaron en la parte baja

;de cada numero flotante

pshufb xmm0, xmm6

copiarResultado:

**mov** eax,[EBP + 20] ;PUNTERO DE IMG RESULTADO

**movd** [eax + ecx],xmm0 ;COPIO EL RESULTADO AL VECTOR

;EN LA POSICION INDICADA

operarConLosRestantes:

**add** ecx,4 ;ACCEDO A LOS PROXIMOS 4 VECTORES

**cmp** ecx, edx

**jl** ciclar

**mov** esp,ebp ;RETORNO EL STACK

**pop** ebp ;RECUPERO EBP

**ret** ;FIN DE LA FUNCION

**Imágenes de Prueba**

**Imagen 1 Imagen 2**

** **

**Resultados de interpolación con diferentes valores de P:**

**P = 0.3**

****

**P=0.5**

****

**P=0.7**

****