Diseño e Implementación de una aplicación para Nitidez en Imágenes

María Gabriela Ávila Quesada email: magaby.tec@gmail.com
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Abstract*—In this document, the implementation of an application for image sharpness was analyzed, where the development was at a high level (Python) and low level (mips), in addition, the images that result from applying the kernel are shown, and the time of duration of different image sizes. On the other hand, it explains the way in which the original image was separated, how it is traversed, and the addresses in memory that were used in assembler. In addition, the main conclusions obtained from the implementation of the project are explained.**

***Palabras clave*—Píxeles, sharpened, oversharpened, imagen, kernel.**

I. Introducción

En el presente documento se hablará sobre el código implementado para el procesamiento de imágenes aplicando los kernel de sharpened y over sharpened.

Para F. Giménez-Palomares [1] el kernel de sharpened y oversharpened sería como observa en la matriz I.

TABLE I
KERNEL SHARPENED.

sharpened			Over sharpened		
0	-1	0	-2	-1	0
-1	5	-1	-1	1	1
0	-1	0	0	1	2

Los algoritmos desarrollados en alto nivel fueron escritos en el lenguaje de Python, en el IDE del mismo, para el código de bajo nivel se desarrolla en mips con el simulador de Mars, además, se realizó un makefile .bat, el cual, ejecuta los tres códigos en orden y en consola se muestra mensajes informativos.

II. Algoritmo desarrollado

Para el procesamiento de imágenes se desarrolló tres códigos, dos de alto nivel, los cuales, son utilizados para transformar la imagen en archivos binarios, estos son utilizados en el procesamiento de la misma (algoritmo en bajo nivel) y la transformación de dos archivos correspondientes a los kernels aplicados a la imagen, además, en lenguaje ensamblador se realiza el procesamiento de la imagen, es decir, se aplica la convolución.

A. Algoritmos de alto nivel

*1) Pasar la imagen a un archivo binario (**I_to_B**):* En el emulador utilizado para del ISA seleccionado tiene una limitación, la cual es, que no se puede abrir archivos de no mas de 4 MB debido que la memoria esta limitada, por lo que se divide la imagen original en seis bloques o rectángulos con la misma altura y ancho.

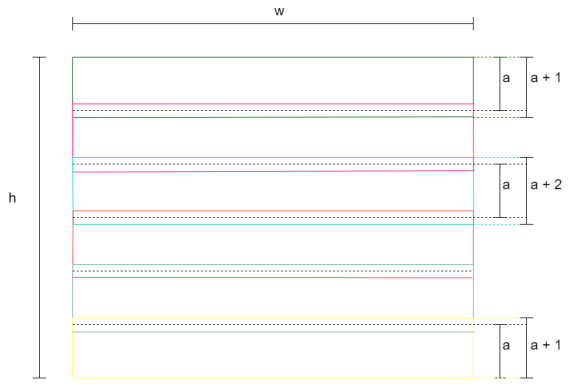

Fig. 1. Matriz divida en 6 rectángulos.

El la figura 1 se puede observar como es la división de los seis rectángulos, se tiene el alto h y el ancho w de la imagen original, de esta forma si se divide el alto h en seis se obtendría el nuevo alto correspondiente a la cantidad de fila que los rectángulos tendrían, todos los rectángulos tiene el mismo ancho w que la imagen original pero el nuevo alto sería a.

Además, como se ve en la imagen 1 el primer rectángulo se agarra la primera fila del segundo rectángulo, esto se hace para poder hacer la convolución de la fila a del primer archivo, en los archivos del 2 al 5 se tiene que agarrar dos filas de mas, una en parte superior y otra en la parte inferior, los motivos es poder hacer la convolución del la fila 1 y la fila a de los respectivos archivos.

Por otro lado, se tiene que el alto de una imagen es de 267, si se divide el alto en 6, el alto a de los rectángulos seria 44, pero si se multiplica 44 * 6 se tiene que el alto es 264, por lo que nos queda 3 filas por fuera, esas filas se agregan al archivo seis junto con la ultima fila del archivo 5 para poder hacer la convolución de la final 1 del archivo 6.

En relación más a los algoritmos desarrollados en esa parte del proyecto se tiene cinco partes o funciones a la hora de ser ejecutado, las cuales son:

1) Se abre una ventana utilizando la librería de tkinter para que el usuario navegue y seleccione la imagen que desea procesar. Las extensiones que permite el programa son: .png, .jpg, .bmp, .jpeg.

2) Cuando el usuario seleccionó la imagen deseada se obtiene el path o dirección donde la misma se encuentra.
3) Por medio de la libreria cv2 y el path obtenido anteriormente, se obtiene la matriz de píxeles en escala de grises (del 0 al 255).
4) Por medio de la función **pixel**, que recibe como parámetro la matriz de píxeles se recorre la matriz y se guarda los píxeles en binario obtenidos por la función **binario** en el archivo que corresponden mencionados anteriormente.
   Además, esta función abre y cierra los seis archivos de escritura, llamados imagen#, donde # es un número de 1 a 6.
5) La función **binario** convierte el píxel, que es un decimal a binario y se le agrega ceros en caso de que no se tenga ocho bits.

*2) Pasar de archivos binarios a imagen (**B_to_I**):* Cuando en ensamblador se realiza el procesamiento de la imagen se genera dos archivos binarios llamados **shaperned** y **over_shaperned**, estos archivos se utilizan para obtener las imágenes resultantes de aplicarles los filtros shaperning y overshaperning.

Los pasos que sigue esta parte del proyecto son:

1) Por medio de librería tkinter se muestra una ventana donde se solicita en ancho y el alto de la imagen procesada.
   Como consideración para ingresar los datos se tiene un botón de *Aceptar* que se tiene que presionar para que se acedan los datos, por medio de la tecla *Enter* del teclado no se realiza ninguna acción.
2) Cuando se presiona el botón *Aceptar* se obtiene los datos de los labels donde se ingreso el ancho y el alto, con esos datos se obtiene el tamaño de la matriz de la imagen, estos se realiza por medio de la función **Aceptar**.
3) Posteriormente, se abre los archivos de lectura **shaperned**, **over_shaperned**, se recorren y se obtienen de 8 en 8 los datos, estos 8 datos se pasan a entero y se almacenan en una matriz dependiendo del archivo que se esta leyendo, si se esta leyendo del archivo **shaperned** se almacena en la matriz llamada **matrizShaperned**, en caso contrario se almacena en la matriz **matriz_OverShaperned**.
   Una vez leído todos los datos de los archivos estos se cierran.
4) Se decidió guardar las imágenes, por los que las matrices anteriores se pasan a una matriz tipo array de la librería de numpy y con la librería de cv2 se transforma los datos en imagen y se guardan.
5) Por último, para visualizar las imágenes en pantalla, con la librería matplotlib se ”gráfica” las matrices obtenidas anteriormente.

*B. Algoritmos de bajo nivel **(ProcesadorImagen)***

En ensamblador existen las etiquetas, las cuales, son porciones de código que se ejecutan secuencialmente, por lo que en esta parte del documento se va a explicar la funcionalidad de cada etiqueta.

En mips se tiene dos secciones .data y .text. En el .data se almacena datos tipo variable para un lenguaje de alto nivel, en el caso del programa se tiene las variables de:

- **ancho:** se tiene el mensaje que ingrese el ancho de la imagen.
- **alto:** se tiene el mensaje donde se solicita el alto de la imagen.
- **mensaje:** sirve para indicar al usuario que se esta procesando la imagen.
- **open_file#:** nombre y dirección de los archivos binarios, donde # es un número de 1 al 6.
- **shapernedFile:** nombre y dirección donde se va a almacenar el resultado de la convolución de la imagen con el kernel de shaperning.
- **over_shapernedFile:** nombre y dirección donde se va a almacenar el resultado de la convolución de la imagen con el kernel de overshaperning.
- **buffer:** es un número que representa el espacio de memoria a almacenar.

Como se mencionó anteriormente en mips existe dos secciones una de ellas es el .text donde se encuentra en código y las etiquetas utilizadas, y esas etiquetas son:

- **main:** En esta etiqueta se solicita por medio de una ventana de dialogo el ancho, y el alto de la imagen y se almacena en memoria, además, se obtiene el alto de los rectángulos, se inicia el contador del ancho y altura en 1, se inicia el contador datos de los archivos en 0, y se inicia la bandera en 1, todo esto se almacena en memoria. Por otro lado, se separa en el heap 8 bits y la dirección donde esta el inicio del heap se almacena en memoria.
  Se llaman a las etiquetas de **Shaperned_kernel**, **OverShaperned_kernel**, **openRead_File1**, **closeFile**, **openShaperned_file** y **openOverShaperned_file**, mas delante se explica que hace cada etiqueta.
- **imageProcessor:** Esta etiqueta verifica si el contador del alto es igual al alto de la imagen + 1, si es así salta a **end**, en caso contrario, se verifica si el contador del ancho es igual a 1, igual al ancho de la imagen, o diferente a las anteriores, salta a **widthOne**, **widthWidth** o **differentWidth** respectivamente se cumpla con las comparaciones anteriores.
- **widthOne:** Se compara el contador de la altura con la altura de la imagen, si el contador es igual a 1 salta a **topLeftCorner**, si es igual a la altura salta a **leftCornerDown**, si no salta a **left**.
- **widthWidth:** Se compara el contador de la altura con la altura de la imagen, si el contador es igual a 1 salta a **topRightCorner**, si es igual a la altura salta a **rightCornerDown**, si no salta a **right**.
- **differentWidth:** Se compara el contador de la altura con la altura de la imagen, si el contador es igual a 1 salta a **top**, si es igual a la altura salta a **down**, si no salta a **inMiddle**.
- Para obtener los valores de la submatriz II se utiliza la etiqueta **getBinary**
  - Para **topLeftCorner** se obtiene 5, 6, 8, 9, los demás son 0.

  - Para **leftCornerDown** 2, 3, 5, 6 los demás son 0.
  - Para **left** 2, 3, 5, 6, 8, 9 los demás son 0.
  - Para **topRightCorner** 4, 5, 7, 8, los demás son 0.
  - Para **rightCornerDown** 1, 2, 4, 5, los demás son 0.
  - Para **right** 1, 2, 4, 5, 7, 8, los demás son 0.
  - Para **top** 4, 5, 6, 7, 8, 9, los demás son 0.
  - Para **down** 1, 2, 3, 4, 5, 6, los demás son 0.
  - Para **inMiddle** 1, 2, 3, 4, 5, 6, 7, 8, 9.

TABLE II
SUBMATRIZ DE LA IMAGEN.

1	2	3
4	5	6
7	8	9

Mas delante se explica con la figura II como se obtiene los valores.

- Cuando se esta en la etiqueta de **right** se verifica si se tiene que abir un nuevo archivo con la etiqueta **widthRectangule**, esta etiqueta verifica si el contador del alto es igual a:
  - la altura del rectángulo, se abre el archivo 2.
  - la altura del rectángulo * 2, se abre el archivo 3.
  - la altura del rectángulo * 3, se abre el archivo 4.
  - la altura del rectángulo * 4, se abre el archivo 5.
  - la altura del rectángulo * 5, se abre el archivo 6.
- **convolution:** Esta etiqueta revisa la bandera, si la bandera es 1 salta a **convolutionShaperned**, si es 2 salta a **convolution_overShaperned**.
- **convolutionShaperned:** Esta etiqueta realiza 9 multiplicaciones, la forma en la que se hace es que la columna 1, fila 1 de la submatriz II se multiplica con la columna 1, fila 1 de la matriz I del kernel Shaperned, todos los resultados de las multiplicaciones se suma y ese resultado se almacena en memoria.
  Una vez almacenados en memoria se salta a **verifyNumber** y **begin_allocateMemory**
- **convolution_overShaperned:** Esta etiqueta hace los mismo que la etiqueta anterior solo que con el kernel que se utilizar es el de overshaperned.
- **addWidth:** Aumenta el contador del archivo en 8 y el contador del ancho en 1.
- **addHigh:** Aumenta el contador del archivo en 8, el contador del alto en 1 y el contador del ancho lo pone en 0.
- **verifyNumber:** Si el número obtenido por la convolución es menor que 0 salta a **saveZero**, si el número es mayor que 255 salta a **saveTop**, si no almacena el número en memoria.
- **saveZero:** Almacena un 0 en memoria.
- **saveTop:** Almacena un 1 en memoria.
- **getBinary:** Obtener un dato del archivo, si ese dato es el primero se salta a **beginBinary**, si el dato es 48 en ascii salta a **zero**, si es 49 salta a **one**.
- **beginBinary:** Si el dato es 48 en ascii salta a **beginZero**, si es 49 salta a **beginOne**.
- **beginZero:** Se agrega un 0 a la cadena.
- **zero:** Se agrega un 0 a la cadena.
- **beginOne:** Se agrega un 1 a la cadena.
- **one:** Se agrega un 0 a la cadena.
- **Shaperned_kernel:** Se almacena el kernel de shaperned en memoria.
- **OverShaperned_kernel:** Se almacena el kernel de overshaperned en memoria.
- La etiqueta, **begin_allocateMemory**, **allocateMemory**, **saveFile**, **checkBit**, **checkBegin**, **allocateOne**, **allocateOne_begin**, **allocateZero**, **allocateZero_begin:** lo que hace es utilizar la dirección del heap separada anteriormente, se posiciona en la primera posición y vamos agarrando bit por bit del resultado de la convolución, si es 0 se guarda un 48 en ascii y si es 1 se guarda un 49 en ascii en el archivo.
- **writeShaperned_file** y **writeOver_shapernedFile:** estas etiquetas abren el archivo correspondientes a shaperned y overshaperned para ser escritos. Cuando la etiqueta **writeShaperned_file** salta cambia la bandera y salta a **convolución** para que se hace haga la convolución del overshaperned, y cuando **writeOver_shapernedFile** se ejecuta la etiqueta **imageProcessor**.
- **openRead_File#:** el # representa un número de 1 al 6. Estas etiquetas lo que hace es abrir los archivos binarios que contienen la imagen recortada, y se abre conforme pesan, para el primer archivo el archivo debe de pesa el ancho de la imagen * (el alto del rectángulo + 1) * 8, para los archivos del 2 al 5 el archivo pesa el ancho de la imagen * (el alto del rectángulo + 2) * 8 y por ultimo el sexto archivo pesa el ancho de la imagen * (el alto del rectángulo + 1 + (alto del rectángulo * 6 - alto de la imagen)) * 8.
- **closeFile**, **closeFile_shaperned**, **closeFile_Overshaperned:** estas etiquetan cierran los archivos que se están manipulando para lectura y escritura.
- **end:** cierra el programa.

En la tabla III se ve las direcciones de memoria y el contenido que en ella se almarecena.

TABLE III
TABLA CON LAS DIRECCIONES DE MEMORIA 0X100000XX.

Dirección	Contenido	Dirección	Contenido	Dirección	Contenido
00	ancho	28	5 (s)	64	2 (sm)
04	alto	2c	f.d (s)	68	3 (sm)
08	c. ancho	38	d. heap	6c	4 (sm)
0c	c. alto	40	0 (Os)	70	5 (sm)
10	c. archivo	44	-1 (Os)	74	6 (sm)
14	n. guardado	48	-2 (Os)	78	7 (sm)
18	bandera	4c	1 (Os)	7c	8 (sm)
1c	alto rec.	50	2 (Os)	80	9 (sm)
20	0 (s)	54	f.d (Os)		
24	-1 (s)	60	1 (sm)		

Donde c. es contador, n. es número, rec. es rectángulo, s es sharpened, d.f es file descriptor, d. es dirección, Os es over sharpened, sm es submatriz.

La forma que se recorre la matriz de la imagen 2 es:

- Contador de la altura es 1.

  - El contador de ancho es 1, se obtiene de la matriz II los valores 5, 6, 8, 9, los demás son ceros.
  - El contador del ancho es el ancho de la imagen de la matriz II se obtiene los valores 4, 5, 7, 8, los demás son ceros.
  - El contador del ancho no es 1 o el ancho de la imagen de la matriz II se obtiene los valores 4, 5, 6, 7, 8, 9 los demás son ceros.
- Contador de la altura es la altura de la imagen.
  - El contador de ancho es 1, se obtiene de la matriz II los valores 2, 3, 5, 6, los demás son ceros.
  - El contador del ancho es el ancho de la imagen de la matriz II se obtiene los valores 1, 2, 4, 5, los demás son ceros.
  - El contador del ancho no es 1 o el ancho de la imagen de la matriz II se obtiene los valores 1, 2, 3, 4, 5, 6 los demás son ceros.
- Contador de la altura no es 1 ni al altura de la imagen.
  - El contador de ancho es 1, se obtiene de la matriz II los valores 2, 3, 5, 6, 8, 9 los demás son ceros.
  - El contador del ancho es el ancho de la imagen de la matriz II se obtiene los valores 1, 2, 4, 5, 6, 7 los demás son ceros.
  - El contador del ancho no es 1 o el ancho de la imagen de la matriz II se obtiene los valores todos los valores.

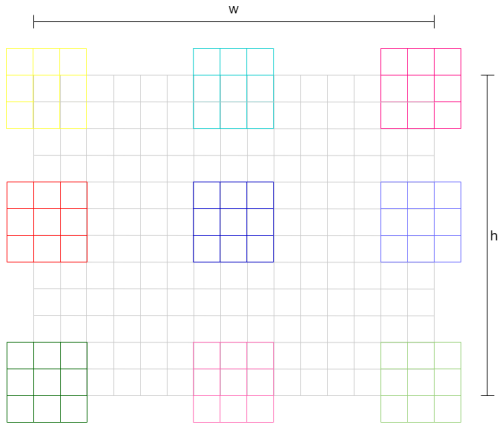

Fig. 2. Posicionamiento de la submatriz II en la matriz de la imagen.

III. Resultados

Cuando se ejecuta el makefile con la figura 3 se obtiene como resultado la figuras 4, como se observa la imagen de la izquierda se ven los borden un poco resaltados con respecto a la imagen original 3, y en la imagen de la derecha se nota que los borden son aun más notorio en especial en el perico que seria la parte de la imagen que tiene mas textura.

En la imagen oversharpened de la figura 4 se aplica un filtro que realiza un ”repujado” por lo que los bordes de los objetos se ven aun mas resaltados en comparación a la imagen original y a la imagen de la izquierda.

La tabla IV se tiene la duración según sea la resolución, se puede notar que entra mas grande la imagen mas tiempo

Fig. 3. Imagen original.

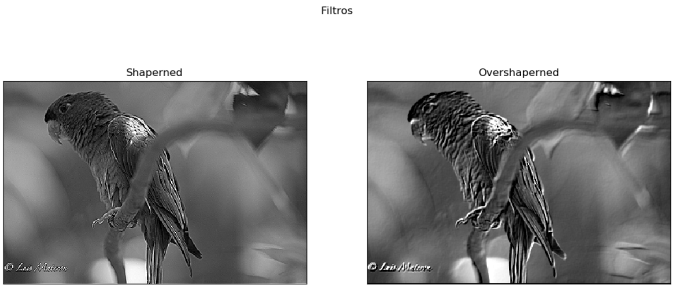

Fig. 4. Imágenes resultantes de haber aplicado los kernel de sharpened y oversharpened.

se dura. Notas: se corrió en el power shell de windows con el comando Measure-Command java jar Mars4_5.jar ProcesadorImagen.asm.

TABLE IV
TABLA CON LAS DIRECCIONES DE MEMORIA.

Resolución	Duración (min)
185 x 275	0,34
400 x 267	1,6
900 x 600	5,18
1920 x 1080	20,24

IV. Conclusiones

En la implantación permite generar el resultado de aplicar filtros de convolución kernel de sharpened y over sharpened. Además, dependiendo del filtro se puede observar de una manera mas notoria los bordes resaltados. Se compara el tiempo que dura el procesamiento diferentes tamaños de la imagen, por lo que como resultado que entre mas grande la imagen mas es el tiempo. Por ultimo, a pesar que un simulador tiene limitaciones siempre se puede llegar a una solución obtima con el fin que se puede ejercutar cualquier tamaño de la imagen.

References

[1] Giménez-Palomares, F., Monsoriu, J., Alemany-Martínez, E. (2016). Aplicación de la convolución de matrices al filtrado de imágenes. Riunet.upv.es. Retrieved 28 May 2020, from https://riunet.upv.es/bitstream/handle/10251/69639/4524-15767-1-PB.pdf?sequence=1.