



# TP 2: Filtros de imágenes

Nicolas Mastropasqua

Programación sobre redes

August 14, 2020

# 1 Introducción

Este trabajo va a estar orientado en el dominio de un campo de la computación conocido como **Computer Vision**, que se centra en poder hacer que una computadora pueda "entender" y extraer información de imágenes y videos, entre otras tantas cosas.

En particular, nos enfocaremos en un pequeña parte de este campo, que tiene que ver con el **procesamiento de imágenes digitales**.

## 1.1 Imagenes

Podemos pensarlas como funciones  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$  siendo  $f(x, y)$  la intensidad de la luz en la posición  $(x, y)$ . En particular, vamos a restringir su dominio a un rectángulo y su codominio a un rango de intensidades. Es decir,  $f : [a, b] \times [c, d] \rightarrow [0, 255]$ .

Por supuesto que, formalmente esta definición tiene sentido, pero en una computadora no podemos almacenar los infinitos valores del dominio.

### 1.1.1 Imagenes digitales

Vamos a decir que una imagen digital<sup>1</sup> surge a partir de muestrear el dominio de  $f$  con cierta granularidad.

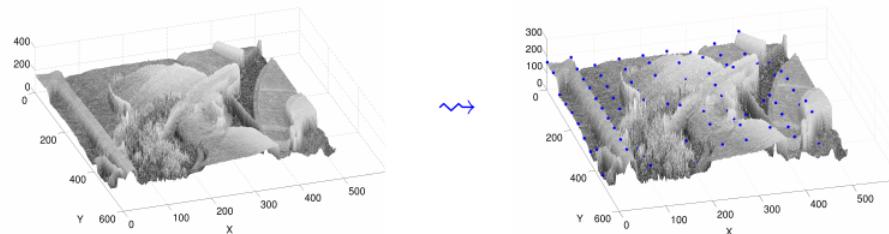


Figure 1: Una imagen se puede interpretar como una función de dos variables.

Luego, hecho esto, podemos pensar una imagen digital, en escala de grises, como una matriz  $I \in \mathbb{R}^{m \times n}$  donde  $I(x, y)$  es la intensidad del píxel en la fila  $x \in \{1, \dots, n\}$ , columna  $y \in \{1, \dots, m\}$ . Nos vamos a restringir a intensidades en un intervalo determinado. Es decir  $I(x, y) \in [0, 255]$ , por lo que un píxel podrá tomar 256 posibles intensidades distintas de grises.

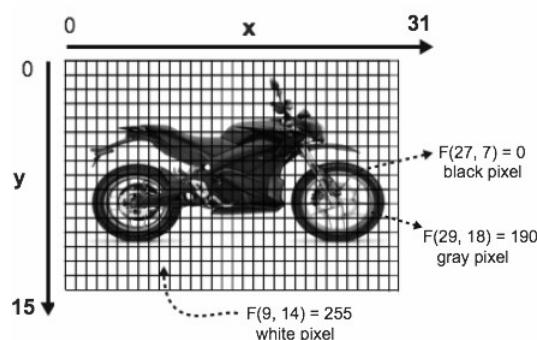


Figure 2: Una imagen se puede interpretar como una función de dos variables. Una vez discretizada, se la puede representar como una matriz de  $\mathbb{R}^{m \times n}$

<sup>1</sup><https://www.youtube.com/watch?v=060Hf1WNCOE>

<sup>2</sup><https://www.youtube.com/watch?v=LWxu4rkZBLw>

## 1.2 Imágenes digitales a color

Figure 3: Descomposición en canales de una imagen RGB



Para representar una imagen digital a color utilizaremos el modelo RGB. En el mismo, se consideran tres canales que representan la intensidad, en una escala de 256 valores, de cada color primario: rojo, verde y azul. RGB es un modelo aditivo, por lo tanto, el color final de un píxel estará determinado por la mezcla que se obtiene de sumar la intensidad de cada canal. Finalmente, sería conveniente tener, para cada canal, una matriz  $I$  como la anteriormente mencionada. Alternativamente, como se propone en este trabajo, podemos considerar que  $I(x, y)$  es la intensidad de un pixel color que es una tripla  $\langle r, g, b \rangle$  con  $r, g, b \in [0, 255]$ .

## 1.3 Filtros de imágenes

Este tipo de transformaciones operan sobre el rango de cada píxel (es decir, el valor de la intensidad). Otra alternativa es **Image warping** que considera un remapeo de la posición de los píxeles de una imagen.

Si bien hoy en día, con el surgimiento de las redes sociales, el concepto de filtros de imágenes está bastante instaurado (Instagram, Snapchat, etc), es importante mencionar que el proceso de filtrado de una imagen juega un rol importante en distintas etapas de los *pipelines* de algoritmos con arquitecturas muy complejas.

En nuestro caso particular, vamos a apuntar a desarrollar filtros sencillos de más "bajo nivel" (comparado con los que se ven en 4). Sin embargo, varios de esos filtros resultan de la composición de otros más elementales.

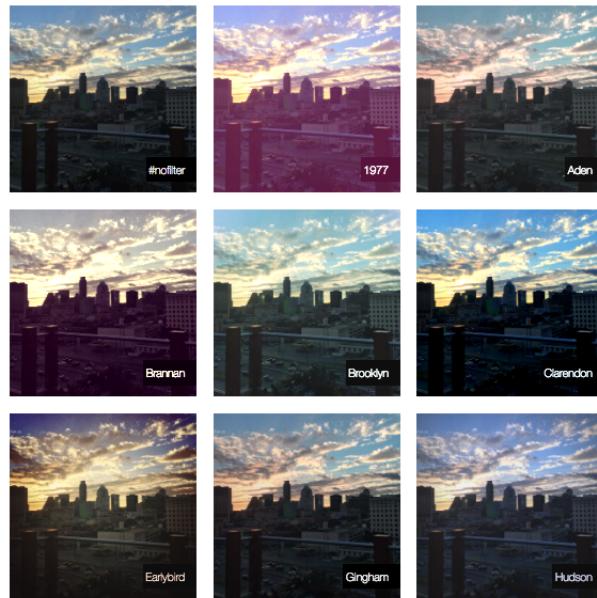


Figure 4: Una selección de filtros de más "alto nivel"

## 1.4 Filtros pixel to pixel

Vamos a concentrarnos en algunos filtros que se encargan de modificar el valor de cada píxel de una imagen, aplicando una transformación en donde solo interviene ese píxel (y ninguno de su *vecindad*). A continuación se detalla la idea de cada uno de ellos

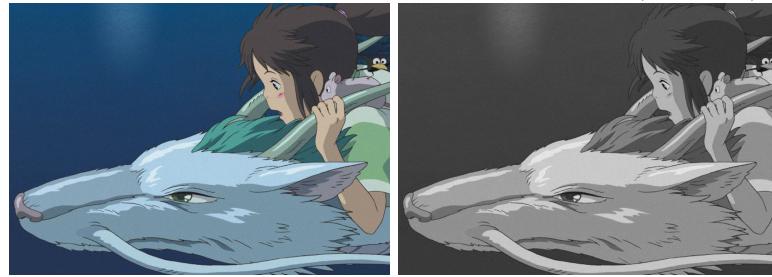
### 1.4.1 Black and White:

Consiste en mapear el píxel color de cada imagen a uno en escala de grises. La intensidad del gris al que se debe mapear cada canal RGB de cada píxel color de la imagen, estará determinada por el promedio de los mismos.

Es decir, si para cada píxel  $(x, y)$  se tiene  $I(x, y) = \langle r, g, b \rangle$ , luego de la transformación se obtendría

$$I(x, y) = \langle g, g, g \rangle \text{ con } g = (r + g + b)/3, g \leq 255$$

Figure 5: Black and White sobre la imagen original (izquierda)



### 1.4.2 Shades:

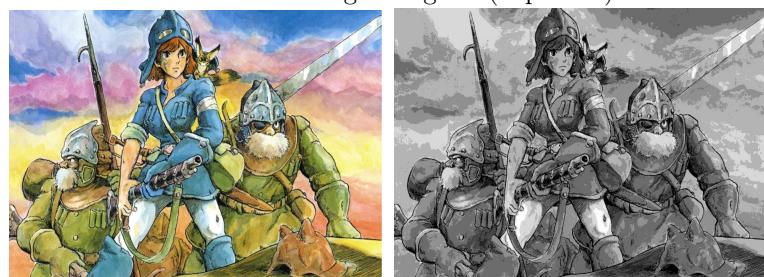
Consiste en convertir la imagen a una escala de  $n$  distintos grises. El algoritmo es similar al anteriormente mencionado, pero ahora varias intensidades de color se mapean a una misma tonalidad de gris.

Más específicamente si para cada píxel  $(x, y)$  se tiene  $I(x, y) = \langle r, g, b \rangle$ , luego de la transformación se obtendría (usando división entera)

$$I(x, y) = \langle g, g, g \rangle \text{ con } g = (\frac{g'}{\text{rango}}) \text{ rango},$$

$$g' = \frac{r + g + b}{3} \text{ y rango} = \frac{255}{n - 1}$$

Figure 6: Transformación de la imagen original (izquierda) a escala de 8 grises



### 1.4.3 Brightness:

Consiste en aumentar o disminuir el brillo de una imagen a partir de un porcentaje específico. Es decir, si para cada píxel  $(x, y)$  se tiene  $I(x, y) = < r, g, b >$ , luego de la transformación se obtendría

$$I(x, y) = < r + 255 * b, g + 255 * b, b + 255 * b > \text{ con } b \in [-1, 1]$$

En este caso  $b$  representa el porcentaje de brillo a aplicar. Notar que es posible que el valor de cada canal resulte superior a 255 o inferior a 0 luego de la suma. En ese caso, es necesario **truncar** el valor resultante para que no supere el rango establecido de intensidades([0,255])

Figure 7: Reducción del brillo 40 % sobre la imagen original (izquierda)



### 1.4.4 Contrast:

Para modificar el contraste de la imagen podemos considerar la siguiente transformación aplicada a cada pixel de la imagen.

Si para cada pixel  $(x, y)$  se tiene  $I(x, y) = < r, g, b >$ , luego de la transformación se obtendría

$$I(x, y) = < f(r - 128) + 128, f(g - 128) + 128, f(b - 128) + 128 > \text{ con}$$

$$f = \frac{259(c + 255)}{255(259 - c)} \text{ y } c \in [-225, 255]$$

En este caso,  $c$  especifica el factor de intensidad del contrast (que no está normalizado como el anterior). Tener en cuenta que luego de las operaciones el valor de cada píxel podría resultar superior a 255 o menor a 0.

Figure 8: Aumento del contraste sobre la imagen original (izquierda)



### 1.4.5 Merge:

Consiste en mezclar, superponer, dos imágenes de igual tamaño a partir de un porcentaje que determinará la intensidad de mezclado de cada una.

Si para cada píxel  $(x, y)$  se tiene  $I_1(x, y) = < r_1, g_1, b_1 >$  y  $I_2(x, y) = < r_2, g_2, b_2 >$  luego de la transformación se obtendría

$$I_1(x, y) = < r_1p_1 + r_2p_2, g_1p_1 + g_2p_2, b_1p_1 + b_2p_2 >$$

En este caso  $p_1$  el porcentaje de mezcla de la primer imagen y  $p_2 = 1 - p_1$

Figure 9: Merge de las primera (30%) y segunda (70%)



## 1.5 Convolution filters

Estos filtros <sup>3</sup> consisten en aplicar una convolución entre un kernel, para nosotros una matriz de  $\mathbb{R}^{3 \times 3}$  y una imagen como se muestra en la figura 10. Es interesante notar que no se podrá aplicar directamente el filtro sobre todos los píxeles de la imagen. Para esos casos, existen varias alternativas, pero una bastante sencilla es simplemente ignorarlos. A su vez, notemos que esto hará que la imagen resultante sea más chica. A continuación se detallan algunos filtros de convolución posibles.

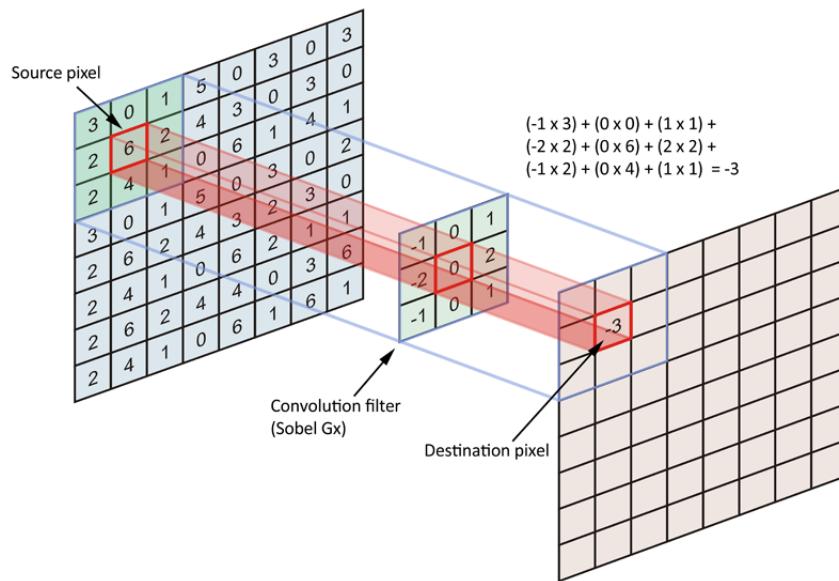


Figure 10: Filtro de convolución

<sup>3</sup><https://www.youtube.com/watch?v=XuD4C8vJzEQ>

### 1.5.1 Box Blur:

Box Blur <sup>4</sup> consiste en aplicar un desenfoque general a la imagen. Para ello, cada canal de cada píxel de la imagen adopta la intensidad del promedio de los 9 píxeles de su vecindad. Para esto, vamos a considerar el

siguiente kernel:  $\frac{1}{9} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

Figure 11: Suave desenfoque (box blur) sobre la imagen original (izquierda)



### 1.5.2 Edge detection con Sobel filter

Con Sobel Filter <sup>5</sup> es posible detectar bordes en imágenes. Eligiendo cuidadosamente el kernel, podemos generar una imagen en escala de grises que acentúe los bordes, es decir zonas de gran cambio de intensidad.

En particular, utilizaremos el siguiente Sobel kernel  $\begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$  para detectar bordes verticales, y el transpuesto para los bordes horizontales.

Sea  $G_x$  el resultado de aplicar el filtro vertical y  $G_y$  el resultado de aplicar el filtro horizontal sobre el pixel  $(x, y)$ , podemos combinar ambos resultados, obteniendo

$$I(x, y) = \sqrt{G_x^2 + G_y^2}$$

Previamente, se sugiere aplicarle un filtro Black White a la imagen y un blur. Al igual que en casos anteriores, es posible que sea necesario truncar el valor final del pixel.

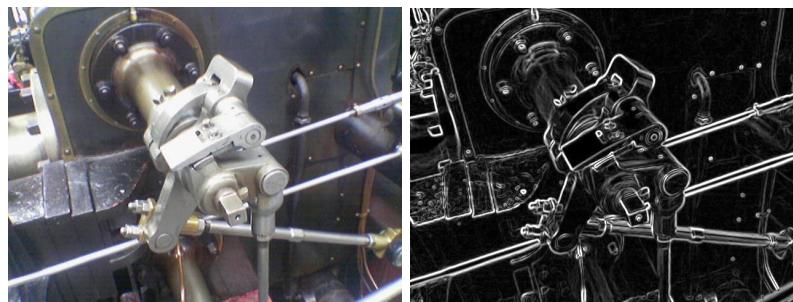


Figure 12: Detección de bordes con sobel filter sobre la imagen original(izquierda)

<sup>4</sup>[https://www.youtube.com/watch?v=C\\_zFhWdM4ic](https://www.youtube.com/watch?v=C_zFhWdM4ic)

<sup>5</sup><https://www.youtube.com/watch?v=uihBwtPIBxM>

### 1.5.3 Sharpen

Eligiendo cuidadosamente el kernel, podemos lograr generar una imagen con el denominado efecto. En particular, utilizaremos el siguiente kernel

$$\begin{pmatrix} 0 & -1 & -1 \\ -1 & 5 & -2 \\ 0 & -1 & -1 \end{pmatrix}$$


Figure 13: Aplicación de sharpen sobre imagen original(izquierda)

## 1.6 Otros filtros

### 1.6.1 Crop:

Consiste en generar una nueva imagen que resulta de la eliminación de las primeras  $k$  filas y  $t$  columnas de la original.

### 1.6.2 Frame:

Consiste en agregar un marco, de alguna tonalidad de grises y tamaño específico, al contorno de la imagen.

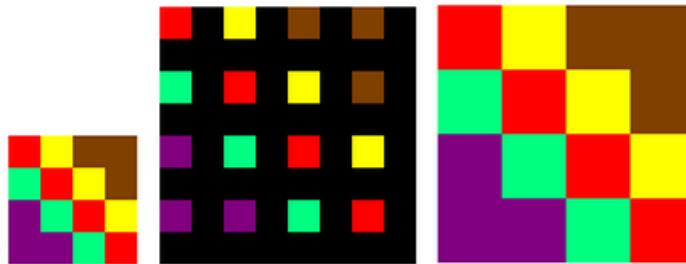
Figure 14: Frame de 30 píxeles negros sobre la imagen original (izquierda)



### 1.6.3 Digital Zoom:

Digital Zoom <sup>6</sup> consiste en transformar una imagen  $I_1$  en otra  $I_2$  de mayor tamaño. Para esto, es necesario determinar como se obtendrán los nuevos píxeles de  $I_2$  y cuanto más grande será. En nuestro caso, utilizaremos la técnica del vecino más cercano, que se ilustra a continuación, para determinar las nuevas intensidades y los zooms del tipo  $nX$ , que multiplican por  $n$  la cantidad de filas y columnas de  $I_1$  para obtener  $I_2$

Figure 15: Zoom 2x sobre imagen 4x4 aplicando vecino más cercano para obtener imagen 8x8



## 1.7 Procesamiento de imágenes con threads

Una aplicación suficientemente interesante de *multi-threading* es el procesamiento de imágenes. De hecho, se han mostrado distintas implementaciones que logran mejorar la performance de dichos algoritmos [1].

Parte de este trabajo consistirá en explorar distintas implementaciones multi-thread para alguno de los filtros mencionados. Es decir, encontrar alguna forma de subdividir el trabajo realizado por un algoritmo de filtrado en distintos threads. Para esto, habrá que tener en cuenta que estamos trabajando con un esquema de memoria compartida y por lo tanto, es el escenario ideal para que aparezcan **race conditions**.

Además de lo anterior, nos va a interesar desarrollar un algoritmo que pueda filtrar un lote de imágenes. En este sentido, buscaremos mejorar la performance también con la incorporación de threads.

## 2 Herramientas

Para trabajar con imágenes, se utilizará, exclusivamente, el formato *Portable Pixel Map* (.ppm)<sup>7</sup> y el lenguaje C++. En este sentido, en el archivo **ppm.cpp** se provee una clase para tal fin y en **example.cpp** un ejemplo de utilización de la misma. Se sugiere el uso de esta clase, aunque es posible modificarla de acuerdo a sus necesidades con la justificación adecuada en tal caso.

Si bien la mayoría de las imágenes de ejemplo de este documento se entregan a modo ilustrativo en formato ppm, es probable que para la experimentación necesiten otras. Desde algunos programa de edición de imágenes, como **GIMP**<sup>8</sup>, es posible exportar de JPEG, u otros formatos, a ppm.

Tener en cuenta que será necesario **C++ threads**, por lo tanto se sugiere utilizar alguna distribución de Linux para el desarrollo del trabajo (o Cygwin en su defecto).

Para la experimentación, se sugiere diseñar scripts en C++ para el análisis de tiempos entre otras cosas. A su vez, también será posible integrarlo esta parte con alguna notebook de Python, por ejemplo.

<sup>6</sup>[https://www.youtube.com/watch?v=AqscP7rc8\\_M](https://www.youtube.com/watch?v=AqscP7rc8_M)

<sup>7</sup><http://netpbm.sourceforge.net/doc/ppm.html>

<sup>8</sup><http://www.gimp.org.es/>

### 3 Enunciado

Para organizar este trabajo, se sugiere dividirlo en tres etapas que se detallan a continuación.

#### 3.1 Implementación en C++

- En filters.cpp se deberán implementar un subconjunto, a determinar luego por el docente, de las distintas clases de algoritmos de filtros de imágenes (3 pixel-to-pixel, 1 de convolución, 2 de otros).
- También en **filters.cpp** se deberá implementar un subconjunto, a determinar luego por el docente, de los algoritmos en su versión multi-thread (1 de pixel-to-pixel y 1 de convolución).
- Se deberá entregar un archivo **tp.cpp** que permita generar un ejecutable para correr los distintos algoritmos implementados. El formato esperado es el siguiente:

```
./TP <FILTRO> <N_THREADS> <P1> <P2> <IMG1> <IMG2>
```

- FILTRO es el nombre del filtro a utilizar, por ejemplo "contrast".
- N\_THREADS indica la cantidad de threads a utilizar por el filtro. Notar que 1 indica la implementación single-thread.
- P1 y p2 son parámetros que puede recibir el filtro. No todos necesitan los dos parámetros, pero el parser se simplifica si los ponen de forma obligatoria.
- IMG1 y IMG2 son el nombre de las imágenes a procesar. Por ejemplo, "imgs/totoro.ppm". Notar que la segunda es requerida solamente para merge o zoom. Al igual que antes, pueden ponerlo como obligatorio para simplificar el parser

- Se deberá entregar un archivo **loader.cpp** que permita generar un ejecutable para correr el algoritmo que carga el lote de imágenes desde un directorio y le aplica un filtro determinado. El formato esperado es el siguiente:

```
./LOADER <FILTRO> <N_THREADS> <ROOT_DIR>
```

- FILTRO es el nombre del filtro a utilizar, por ejemplo "contrast".
- N\_THREADS indica la cantidad de threads a utilizar por el filtro. Notar que 1 indica la implementación single-thread.
- ROOT\_DIR es el directorio donde se encuentra el lote de imágenes.

En todos los casos es importante garantizar que el código entregado **compile sin problemas**.

### 3.2 Experimentación

Parte del trabajo consiste en experimentar distintos aspectos de los algoritmos *multi-thread* para luego elaborar un informe, que debería incluir gráficos claros donde se muestren los resultados obtenidos, así también como una discusión sobre los mismos. Como guía para esto, se proponen las siguientes preguntas disparadoras:

- Considerando un filtro particular, comparar la performance de ejecución en su versión single y multi-thread (para alguna cantidad determinada de threads)
  - ¿Qué impacto tiene considerar imágenes "grandes" e imágenes "chicas"?
  - ¿Cómo cambia el escenario si considero una mayor cantidad de threads?
  - ¿Cuán determinante es la configuración de hardware donde corro los experimentos\*
- ¿Hay diferencias de performance para los distintos tipos de filtros multi-thread?
- En base a todo lo anterior, ¿siempre es conveniente paralelizar? ¿De qué factores de la entrada depende esto?
- Considerar un análisis similar, pero para el caso del loader single-thread y multi-thread

Tener en cuenta que es parte de la evaluación la prolividad, claridad y rigor del informe. También es necesario que se entreguen **todos los scripts** que utilizaron para el desarrollo del mismo.

## 4 Entrega

La entrega del tp, tanto código como informe, deberá ser subida al *Classroom* en la tarea correspondiente. El formato de entrega será un archivo zip con el apellido de los integrantes del grupo, que **no puede exceder las dos personas**. La fecha límite propuesta para la entrega final es el **4 de Septiembre** hasta las 23:59:59. Es obligatorio entregar un checkpoint del trabajo el **27 de Agosto**. De no hacerlo, se perderá la oportunidad de la última entrega.

Es posible que seleccione algunos grupos para hacer una defensa del trabajo (se van a aclarar los detalles más adelante).

Los siguientes ejes serán evaluados:

- **Implementación:** Correctitud. Legibilidad del código y claridad. Uso de recursos vistos en clase.
- **Experimentación:** Desarrollo de experimentos. Completitud del enunciado. Resultados. Claridad de redacción y prolividad del informe. Rigor de las ideas desarrolladas y conclusiones.

## References

- [1] Alda Kika, Silvana Greca. *multi-threading Image Processing in Single-core and Multi-core CPU using Java*. International Journal of Advanced Computer Science and Applications 4, January 2013.