Trabajo práctico: Multithreading

Programación sobre redes

Nicolas Mastropasqua

Programación sobre redes

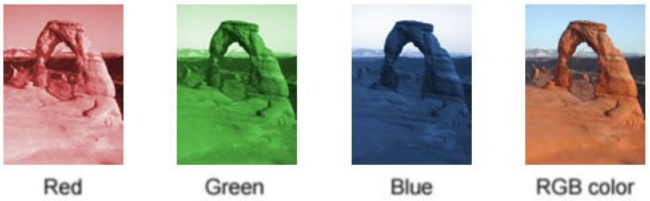
July 10, 2019

1 Procesamiento de imágenes

1.1 Imágenes digitales

Una aplicación suficientemente interesante de *multithreading* es el procesamiento de imágenes. De hecho, se han mostrado distintas implementaciones que logran mejorar la performance de dichos algoritmos [1]. Si bien el espectro es amplio, el trabajo se centrará en explorar y desarrollar distintos filtros de imágenes sencillos. El objetivo, más allá de implementar los algoritmos haciendo *multithreading*, es luego analizar distintos escenarios donde resulte ventajoso y poder sacar conclusiones al respecto.
Podemos pensar una imagen digital, en escala de grises, como una matriz $I \in R^{mxn}$ donde $I(x, y)$ es la intensidad del píxel en la fila $x \in \{1, ..., n\}$, columna $y \in \{1, ..., m\}$. 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.
Para representar una imagen 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, serían conveniente tener, para cada canal, una matriz I como la anteriormente mencionada.

Figure 1: Descomposición en canales de una imagen RGB



Alternativamente, como se propone en este trabajo, podemos considerar que $I(x, y)$ es la intensidad de un pixel color que es una tripla $< r, g, b >$ con $r, g, b \in [0, 255]$.

1.2 Filtros de imágenes

En este trabajo se desarrollaran distintos algoritmos para manipular imágenes. Esencialmente, cada uno de ellos tomará al menos una imagen I_1 y modificará cada uno de sus píxeles color, obteniendo una I_2. A continuación se detalla la idea detrás de cada uno de ellos:

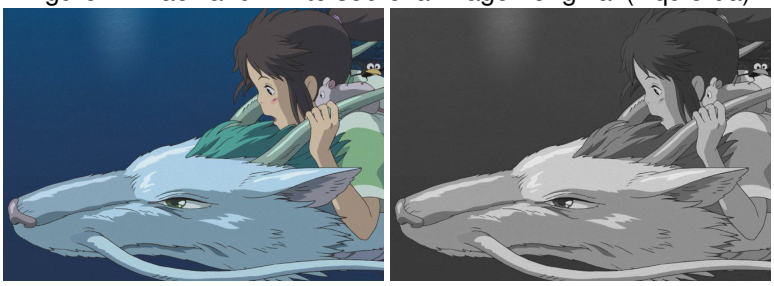
1.2.1 Crop:

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

1.2.2 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 pixel (x, y) se tiene $I(x, y) =< r, g, b >$, luego de la transformación se obtendría $I'(x, y) =< gris, gris, gris >$ siendo $gris = (r + g + b)/3$.

Figure 2: Black and White sobre la imagen original (izquierda)



1.2.3 Frame:

Consiste en agregar un marco, de algún color y tamaño específico, al contorno de la imagen.

Figure 3: Frame de 30 píxeles negros sobre la imagen original (izquierda)

1.2.4 Box Blur:

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 las intensidades de los ocho vecinos más cercanos. La vecindad de un píxel son todos aquellos que se encuentran a distancia exactamente uno de él (considerando todas las direcciones). Es decir, la vecindad del píxel (x, y) está compuesta por :
$\{(x, y+1), (x, y-1), (x-1, y-1), (x-1, y), (x-1, y+1), (x+1, y-1), (x+1, y), (x+1, y+1)\}$
Tener en cuenta que existen casos en los que la vecindad anterior no está bien definida, por lo tanto será necesario tomar alguna decisión al respecto.

Figure 4: Suave desenfoque (box blur) sobre la imagen original (izquierda)

1.2.5 Brightness:

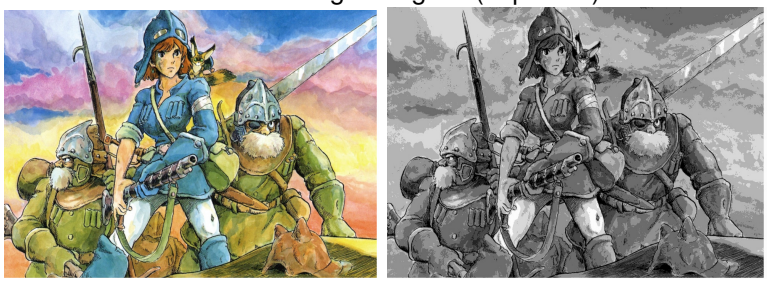
Consiste en aumentar o disminuir el brillo de una imagen a partir de un porcentaje específico.
Es decir, si para cada píxel (x, y) se tiene $I(x, y) =< r, g, b >$, luego de la transformación se obtendría $I'(x, y) =< r + 255 * b, g + 255 * b, b + 255 * b >$ donde $b \in [-1, 1]$ representa el porcentaje de brillo. 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 5: Reducción del brillo 40 $\%$ sobre la imagen original (izquierda)

1.2.6 Gray Scale:

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 mapearan a una misma tonalidad de gris.
Más específicamente si para cada pixel (x, y) se tiene $I(x, y) =< r, g, b >$, luego de la transformación se obtendría $I'(x, y) =< gris, gris, gris >$ siendo $gris = (gris'/rango)rango$ y $gris' = (r + g + b)/3$. Tener en cuenta que $/$ denota división entera.

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



1.2.7 Contrast:

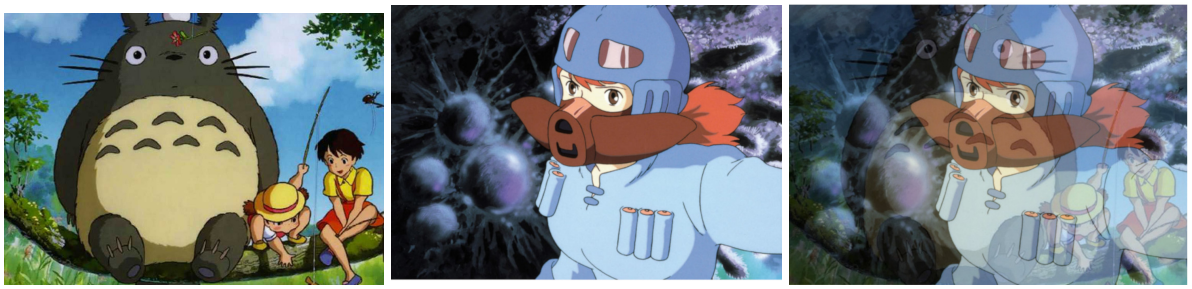
Para modificar el contraste de la imagen podemos considerar la siguiente transformación aplicada a cada pixel de la imagen.
Si para cada pixel (x, y) se tiene $I(x, y) =< r, g, b >$, luego de la transformación se obtendría $I'(x, y) =< f(r - 128) + 128, f(g - 128) + 128, f(b - 128) + 128 >$ siendo $f = (259(c + 255))/(255(259 - c))$ y c un valor entre $[-225, 255]$ especificando el factor de intensidad del contraste. Tener en cuenta que en I' el valor en cada píxel podría resultar superior a 255 o menor a 0, por lo que habría que considerar truncar.

Figure 7: Aumento del contraste sobre la imagen original (izquierda)

1.2.8 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 una nueva imagen I' tal que $I'(x, y) =< r_1p_1 + r_2p_2, g_1p_1 + g_2p_2, b_1p_1 + b_2p_2 >$ siendo $p1$ el porcentaje de mezcla de la primer imagen y $p2 = 1 - p1$

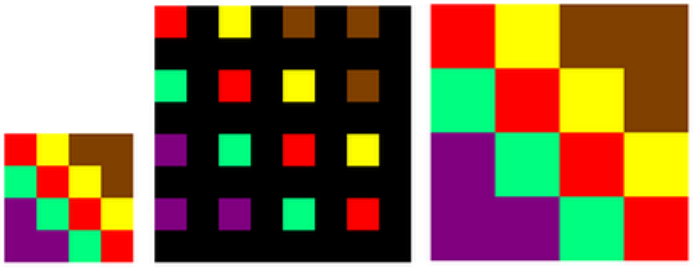
Figure 8: Merge de las primera (30%) y segunda (70%)



1.2.9 Digital Zoom:

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



2 Herramientas

Para trabajar con imágenes, se utilizará, exclusivamente, el formato *Portable Pixel Map* (.ppm)[1] 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 puedan necesitar otras. Desde algunos programa de edición de imágenes, como **GIMP** [2], es posible exportar de JPEG, u otros formatos, a ppm.
Tener en cuenta que será necesario utilizar la API **pthreads**, por lo tanto sería conveniente utilizar alguna distribución de Linux para el desarrollo del trabajo. Alternativamente, aunque menos recomendado, se puede considerar Cygwin en Windows.
Para la experimentación, se sugiere diseñar scripts en C++ para el análisis de tiempos entre otras cosas. Recordar que, normalmente, tomar una sola medición de la corrida de un algoritmo no alcanza, ya que, por diversos motivos, puede estar sujeta a ruido. Es conveniente, entonces, realizar más de una medición del mismo y luego reportar alguna medida final.

3 Enunciado

3.1 Implementación en C++

- Se deberán implementar los algoritmos de filtros de imágenes, tanto su versión *single thread* como *multi thread*, en el archivo **filters.cpp**. Se sugiere fuertemente realizar los mismos por etapas, comenzando por la versión single thread.
- 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> <TIPO> <P1> <P2> <IMG1> <IMG2>

  - FILTRO es el nombre del filtro a utilizar, por ejemplo "contrast".
  - TIPO indica si se utiliza la versión *single-thread* o *multi-thread*
  - P1 y P2 son parámetros que puede recibir el filtro. Por ejemplo, el porcentaje de brightness.
  - 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.

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

[1] http://netpbm.sourceforge.net/doc/ppm.html
[2] http://www.gimp.org.es/

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 dónde se muestren los resultados obtenidos, así también como una discusión sobre los mismos. Como guía para esto, se proponen las siguientes preguntas disparadoras:

- Dada la configuración de hardware donde se están corriendo los experimentos, ¿cuántos threads, a priori, resultarían convenientes crear para un filtro determinado? En base a lo observado, ¿se cumplió lo esperado?
- Dado un filtro ¿Cómo son los tiempos en comparación con el algoritmos *single-thread*?¿Siempre es conveniente paralelizar?¿Qué papel juega el tamaño de las imágenes y la complejidad del algoritmo?
- Según la ley de Amdahl, ¿cuánto sería el *speedup* para un filtro determinado en la configuración de hardware que se está utilizando? ¿Qué ocurre en la práctica? Si se obtienen tiempos peores, ¿cómo se podría explicar?
- Para las preguntas anteriores, ¿qué ocurre si cambiamos el filtro analizado? ¿Qué similitudes se encuentran y qué conclusiones, más generales, se pueden obtener?

Tener en cuenta que es parte de la evaluación la prolijidad, 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 limite propuesta para la primer entrega es el **24 de Julio** a las 23:59:59. Es obligatorio entregar el estado del trabajo, sea cual sea, hasta ese momento. De no hacerlo, se perderá la oportunidad de obtener una segunda entrega, cuya fecha límite es el **11 de Agosto** a las 23:59:59. Cualquier trabajo entregado fuera del plazo para la primer entrega **no será tenido en cuenta**. 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 prolijidad del informe. Rigor de las ideas desarrolladas y conclusiones.
- **Checkpoint**: El **15 de Julio** cada grupo deberá mostrar sus avances sobre el trabajo y ser capaz de responder alguna pregunta sobre lo realizado.
- **Defensa oral**: El **8 de Agosto** cada integrante deberá realizar una defensa oral del trabajo. Durante la misma se discutirán, con cierta generalidad, distintos aspectos del trabajo. A su vez, cada integrante deberá contestar algunas preguntas particulares sobre detalles de la implementación o experimentación realizada. Por lo tanto es importante que ambos estén al tanto de la integridad del trabajo.

References

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