

Tarea 3 : Introducción a OpenCL

IIC3373

Lluch, Pablo	Fuentes, Tomás
<code>pablo.lluch@gmail.com</code>	<code>tafuentesc@gmail.com</code>

15 de octubre de 2013

1. Funcionalidades Implementadas

El programa elaborado cumple hasta la funcionalidad número 7 especificada, vale decir: Recibe un video a color de más de 30 segundos y lo devuelve ecualizado a al menos 25 fps y con el mismo audio original. En la carpeta input, se encuentra el video de prueba utilizado por los alumnos, llamado short.mp4. El video tiene 40 segundos de duración, está en color y tiene audio stereo.

2. Simplificaciones

Para el correcto funcionamiento del programa, se hicieron algunas simplificaciones, no por falta de capacidad pero más bien porque se consideró que no era necesario tener en cuenta todos los formatos posibles de imágenes y videos para tener una buena demostración del funcionamiento.

1. La imagen de prueba está en formato RGB, donde sus elementos están descritos por un float entre 0 y 255
2. El video de prueba tiene audio stereo

3. Algoritmo

El algoritmo consiste en los siguientes pasos:

3.1. Procesamiento del video

Primeramente, se extraen los frames y el sonido del video original de forma separada. Al tener el audio en un archivo a parte, se permite ecualizar tranquilamente el video sin modificar el audio. La extracción de los frames se hace utilizando el programa ffmpeg, invocado por Java externamente, y se asume que se encuentra en el computador del usuario. Mediante diversas líneas de comando, se pueden ejecutar las funcionalidades deseadas. Ellas son:

1. Extracción del audio - `/usr/bin/ffmpeg -y -i input/short.mp4 -vn -ac 2 -f mp3 input/short.mp3`

2. Extracción de frames - `/usr/bin/ffmpeg -y -i input/short.mp4 -r 25.0 -ss 0 -t 40 -q:v 1 input/image-%3d.jpeg`
3. Unión de frames (después de la ecualización) - `/usr/bin/ffmpeg -start_number 1 -y -i input/image-%3d.jpeg -r 25.0 -vcodec mjpeg -q:v 1 input/equalized.mp4`

En general:

- `-i` = input
- `-r` = frame rate
- `-y` = reemplazar
- `-ss` = tiempo de comienzo
- `-t` = duración
- `-vcodec`, `-acodec` = codecs utilizados
- `-f` = formato de audio
- `-q:v`, `q:a` = calidad del video y audio

3.2. Ecualización

Para la ecualización del video, se utiliza el clásico algoritmo descrito en [1]. Clásicamente, con imágenes de tonos grises, este algoritmo funciona obteniendo, primero, la frecuencia de aparición de cada nivel de gris en una imagen. Una imagen con buen contraste se obtiene haciendo que el histograma acumulativo de tonos en una imagen sea aproximadamente lineal.

El problema resulta cuando se quiere tratar con imágenes en colores, en donde ecualizar por canal usando RGB, por ejemplo, tiene malos resultados pues la información del contraste viene entremezclada entre los canales. Para solucionar este problema, se decidió utilizar uno de los approach mencionados en [2], que consiste en primero transformar la imagen al espacio de colores esférico mencionado en [2]. Esto tiene la gracia de que el canal `r` resume bastante bien la información del contraste, pues reúne información de los 3 canales RGB originales. Luego, se puede ecualizar sobre este canal para luego transformar de vuelta a RGB y rearmar el video.

Como nota a parte, debido a problemas de las librerías usadas para trabajar con ciertos rangos de valores en imágenes, al ecualizar imágenes en coordenadas esféricas se escala por un factor para no tener una imagen con rangos de valores exacerbadamente altos. Esto es un detalle interno pero no afecta el algoritmo de ecualización.

3.3. Paralelismo

Por supuesto, la gracia de esto es que casi todos los procedimientos son altamente paralelos. Considerando el algoritmo descrito, en la transformación de espacio de colores, el valor final de un pixel es independiente de aquellos que lo rodean. Por el otro lado, para el cálculo de histogramas, aunque sí hay mayor dependencia, se puede calcular el histograma por partes. En este caso, se decidió calcular primero el histograma por columnas, dado que generalmente las imágenes cuentan con más columnas que filas. Al final se pueden sumar estos histogramas para obtener el histograma final. Para el cálculo de histogramas y transformación de colores, se utilizaron diversas funciones de kernel con OpenCL. El uso de este mismo se hizo a través de un wrapper de OpenCL para Java, JOCL [3], que facilitó la implementación de los diversos algoritmos y maneja buffers, kernels y otros elementos de OpenCL utilizando un approacho orientado a objetos.

3.4. Formación del nuevo video

Para cumplir con esta función, se reintegraron los frames procesados por OpenCL con la metodología descrita anteriormente, utilizando ffmpeg. Una posibilidad habría sido paralelizar también esta parte, aunque la complejidad agregada era mayor pues habría requerido meterse con encodings de video, por lo que se rechazó esta opción. Otro motivo por el cual se decidió no paralelizar este proceso es que usando ffmpeg externamente, el mayor problema resulta ser el input y output con el disco duro, por lo que las ganancias de paralelizar este proceso (incluso via CPU, por ejemplo) habrían sido bastante menores.

Para la integración del audio, simplemente se unen los tracks de audio y video ecualizado a través de los métodos en la clase Video.java, que utilizan, nuevamente, ffmpeg.

Referencias

- [1] Robert Krutsch and David Tenorio, *Histogram Equalization*. Freescale Semiconductor
- [2] Fan-Chieh Cheng, Shanq-Jang Ruan and Chang-Hong Lin *Color Contrast Enhancement using automatic weighting mean-separated histogram equalization with spherical color model* International Journal of Innovative Computing, Information and Control Volume 7, Number 9, September 2011
- [3] <http://jogamp.org/jocl/www/> *Java Bindings for the OpenCL API*