Tarea 2 - Vicente Dragicevic

Utilicé python (3) y pyCUDA. Hice el último punto del enunciado (video a color, 30 o más segundos, 25 fps, audio original), pero para obtener buenos resultados, primero transformo cada frame al espacio HSV y aplico la ecualización en el canal "V". Al final de la ecualización lo transformo de vuelta a RGB.

El algoritmo que utilizo es básicamente lo que aparece en wikipedia sobre histogram equalization: computar histograma, computar *cumulative distribution function* y luego transformar los valores originales a los nuevos valores ecualizados.

Por completitud, implementé una versión serial y una paralela. Si el ambiente en donde se está ejecutando el programa no cuenta con soporte para CUDA, se ejecuta por default la versión serial. Si se quiere forzar la versión serial se debe incluir el flag `--serial` al ejecutar el programa. Para más información de las opciones disponibles, `./main.py --help`.

Utilicé moviePy para leer y modificar el video. Esto se hace iterando por cada frame del video, aplicando la transformación y luego escribiendo los nuevos valores.

Lo que paralelicé fue lo siguiente:

- La transformación de RGB a HSV y HSV a RGB: utilizo funciones que transforman un pixel de un espacio a otro, lo que paralelicé fue aplicar esta función a cada pixel de la imagen (puede realizarse cada transformación por separado).
- El cálculo de los histogramas: nada muy complejo, recorro los valores del canal *V* y en un arreglo voy guardando la frecuencia. Incremento la frecuencia de forma atómica porque cualquier thread podría modificar cierto valor del arreglo al mismo tiempo.
- El cálculo de la función de transformación: recorro el arreglo de la función de transformación y lo seteo según la *cumulative distribution function* calculada en el paso anterior.

El principal *bottleneck* era la transformación entre espacios de colores. Cuando paralelicé esto subió de procesar ~2 frames por segundo a procesar ~20 (en titan).

Estructura del código

En *main.py* se realiza el parseo de las opciones y argumentos entregados al programa, y luego se llama al ecualizador respectivo para el video indicado en los argumentos.

En *equalizer.py* y *serial-equalizer.py* se encuentra el código del ecualizador paralelo y el serial respectivamente.

En el directorio *kernels* se encuentran los kernels de CUDA que utilicé en el ecualizador paralelo. Además se encuentra *loader.py*, que utilizo para leer los archivos de todos los kernels, los que luego son compilados en *equalizer.py*.

Ejecución

Primero que todo, asegurarse de tener un ambiente con CUDA 6.5 funcionando. Fue difícil configurar pyCUDA por diversos errores extraños, pero lo principal es tener bien seteada la variable de entorno `$LD_LIBRARY_PATH`. Segundo, debe tenerse instalado moviePy. Por último, utilizo python 3.

Para ejecutar el programa, se le entrega primero el video a transformar y luego la ubicación donde se escribirá el video ecualizado.

- `./main.py --help`: información de uso y opciones
- `./main.py [options] source.mp4 target.mp4`: ejecución normal

Testié con el video que se encuentra en https://vimeo.com/108032632. Puede ser descargado en la misma página en distintas resoluciones. En particular probé con la versión HD (720p).

Dependencias

- pyCUDA
- moviePy
- scikit-image (sólo para las transformaciones de espacios de colores en la versión serial)
- numpy (requerido para usar pyCUDA, y para manejar arreglos)

Referencias

- https://github.com/jakebesworth/Simple-Color-Conversions para las transformaciones de colores en C.
- https://en.wikipedia.org/wiki/Histogram_equalization
- https://documen.tician.de/pycuda/tutorial.html
- http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html