Práctica 4

Tone Mapping

Julio Martín y Raquel Peces

ÍNDICE

ÍNDICE 2
DESCRIPCIÓN DEL CÓDIGO 3
Check 3
FindMinMax 3
ComputeHistogram 3
ExclusiveScan 3
Calculate_cdf 3
RESULTADOS OBTENIDOS 5
Memorial_law_large 5
Ennis_latlong 5
Waterfall_bg 6
MEJORAS INTRODUCIDAS 7
Tamaño de bloque 7
Memoria compartida 7
COMENTARIOS PERSONALES 8
Problemas encontrados 8
Crítica constructiva 8
Propuesta de mejora 8
Evaluar tiempo dedicado 8
ÍNDICE DE FIGURAS 9

DESCRIPCIÓN DEL CÓDIGO

La descripción de las principales funciones implementadas sería:

Check

Función auxiliar para gestionar la salida del checkCudaErrors, para poder mostrar los posibles errores por pantalla.

FindMinMax

Esta función realiza una reducción para encontrar el valor máximo y mínimo en distintos arays.

Por un lado tiene como entrada la variable *"const float *d_in_min"* en la cual buscará el valor mínimo y lo devolverá en la variable *"float *d_out_min"* y por otro lado tiene la variable de entrada *"const float *d_in_max"*en la que buscará el valor máximo devolviéndolo en la variable *"float *d_out_max"*.

ComputeHistogram

Método para extraer características notables y patrones en grandes volúmenes de datos. Genera de forma paralela un histograma de todos los valores de luminancia en los diferentes bins existentes, para cada elemento encontrado, se incrementa en uno el bin correspondiente a su tipo. Para ello hace uso de la expresión:

```
int bin = ((lumi[t] - min) / range)*bins;
```

ExclusiveScan

Este método realiza un exclusive scan, realizando primeramente una reducción, construyendo hacia abajo las sumas parciales en los nodos internos del árbol y posteriormente realizando un reverse, que en sentido contrario suma las sumas parciales, que todavía sean necesarias. En este caso lanzaremos el kernel en un grid con tamaño de bloque numBins/2.

Calculate_cdf

Es el método principal que se encarga de definir el tamaño de grid y de bloque y de realizar cada uno de los pasos necesarios para realizar el programa, llamando a los kernels anteriormente descritos:

1) Encontrar el valor máximo y mínimo de luminancia en min_logLum and max_logLum a partir del canal logLuminance

2) Obtener el rango a representar

3) Generar un histograma de todos los valores del canal logLuminance usando la formula bin = (Lum [i] - lumMin) / lumRange * numBins

4) Realizar un exclusive scan en el histograma para obtener la distribución acumulada (cdf) de los valores de luminancia. Se almacenará en el puntero c_cdf

Para calcular el tamaño de grid y de bloque, el tamaño de grid estará definido por el tamaño de la imagen y de bloque para aprovechar al máximo los recursos, mediante la expresión:

```
int gridSize = ceil((float)(numRows*numCols) / (float) BLOCK_SIZE);
```

El tamaño de bloque de 1024, ya que no lo especificamos como una matriz, al contrario que en la práctica anterior, viene definido por las especificaciones de nuestra tarjeta gráfica, una GeForce 840M, que puede lanzar hasta un máximo de 1024 threads por bloque y con un tamaño de Warp de 32, como podemos observar en la captura de pantalla que se muestra debajo de estas líneas en la *Figura 1*:

Figura 1: Especificaciones de GeForce 840M

Por último se encargará de liberar la memoria de las variables d_min, d_max y d_histogram.

RESULTADOS OBTENIDOS

Memorial_law_large

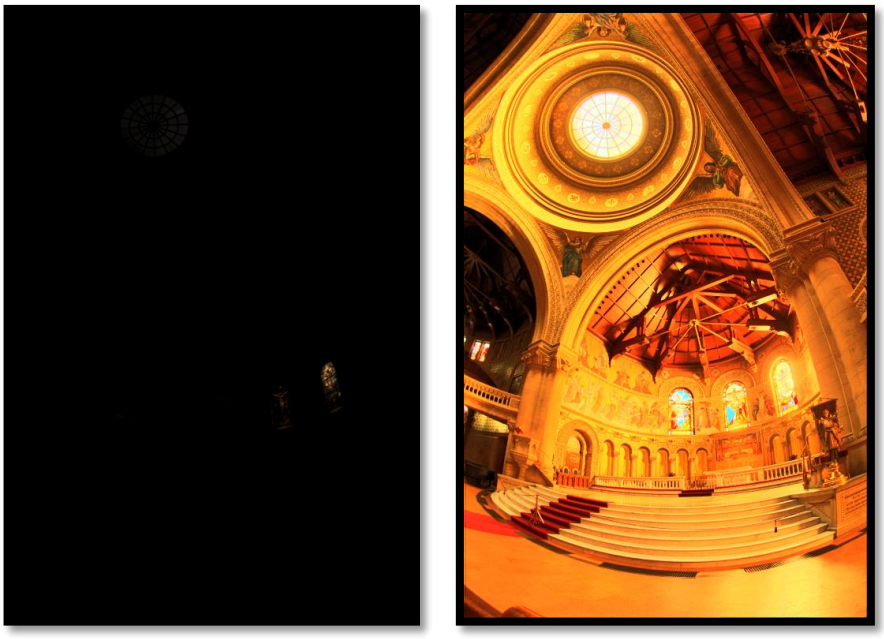

Figura 2: Comparativa memorial_law_large sin tone mapping a la izquierda y con tone mapping a la derecha

Ennis_latlong

Figura 3: Ennis_latlong SIN tone mapping

Figura 4: Ennis_latlong CON tone mapping

Waterfall_bg

Figura 5: Waterfall_bg SIN tone mapping

Figura 6: Watterfall_bg CON tone mapping

MEJORAS INTRODUCIDAS

Tamaño de bloque

Como se ha especificado en el primer apartado, la tarjeta gráfica sobre la que lanzamos el código es una nVidia GeForce 840M, que acepta 1024 threads por bloque como se muestra en la *"figura 22"*, por lo tanto para mejorar el rendimiento lo único que debemos hacer es especificar el tamaño de bloque a 1024.

Figura 7: Resultado de deviceQuery sobre nVidia GeForce 840M

Memoria compartida

Partimos de la idea de que en este tipo de ejercicios, serán los arrays de datos, las variables que necesitaremos compartir por cada thread. En el caso del kernel findMinMax las variables que deberemos de llevar a memoria compartida, serán *"ds_min"* y *"ds_max"*. En el caso del kernel exclusiveScan, está claro que la variable a compartir será tempArray:

```
/*              Share memory              */
    __shared__ float ds_min[BLOCK_SIZE];
    __shared__ float ds_max[BLOCK_SIZE];

__shared__ int tempArray[BLOCK_SIZE * 2];
```

Esta implementación evitará que tengamos que dirigirnos a memoria global cada vez que queramos consultar un dato, mejorando significativamente el tiempo de acceso a los datos, al tratarse de memoria compartida.

En el caso de estos kernels es principalmente importante acordarse de usar el método __syncthreads();, para asegurarse de que todos los threads han acabado su tarea, antes de pasar a la siguiente iteración de cada uno de los algoritmos, sino podríamos encontrarnos con valores erróneos.

COMENTARIOS PERSONALES

Problemas encontrados

La dificultad a la hora de descifrar el formato de un archivo .exr, no es explícitamente necesario, pero ayudaría saber de antemano como están definidos distinto tipo de extensiones de este tipo.

El uso de memoria compartida, también suele ser una dificultad añadida en este tipo de prácticas, el controlar la sincronización, aunque en este caso específico, nos fue más sencillo por el código facilitado en clase.

Crítica constructiva

En este caso, creemos que la resolución de la práctica de multiplicación de matrices, deberíamos haberla hecho directamente en el laboratorio, en lugar de verla primero de forma teórica y más tarde de nuevo en el laboratorio, lo que nos hubiera dado tiempo a implementar otro tipo de métodos como el eclusive scan en clase, lo que hubiese rebajado dificultad a la práctica o incluso nos hubiese permitido implementar algo más.

Propuesta de mejora

Creemos que el tema de la práctica está bien, las visualizaciones son más llamativas que en la primera práctica y realmente se nota una mejora en la imagen, el orientarla a temas gráficos es algo que le aporta calidad a la asignatura de cara al master que realizamos.

Quizás como propuesta de mejora, es la utilización de más tipos de extensiones de este tipo, dónde se facilita mayor información que en los formatos habituales.

Evaluar tiempo dedicado

El tiempo de la práctica es adecuado, porque además se nos facilitaba gran cantidad de código en los apuntes de la asignatura, lo que más nos ha retrasado ha sido el entender el formato de las extensiones .exr, pero el resto de la práctica se ha realizado en un tiempo que creemos adecuado para lo que se pedía, la mayor pega ha sido tener que realizarla durante el periodo de exámenes. Aunque en este caso no ha sido tan contraproducente, porque en parte nos ha ayudado a repasar patrones que debíamos estudiar.

ÍNDICE DE FIGURAS

Figura 1: Especificaciones de GeForce 840M 4
Figura 2: Comparativa memorial_law_large sin tone mapping a la izquierda y con tone mapping a la derecha 5
Figura 3: Ennis_latlong SIN tone mapping 5
Figura 4: Ennis_latlong CON tone mapping 5
Figura 5: Waterfall_bg SIN tone mapping 6
Figura 6: Watterfall_bg CON tone mapping 6
Figura 7: Resultado de deviceQuery sobre nVidia GeForce 840M 7