UNIVERSIDAD DE ALICANTE

CUDA WORKSHOP 2018

PRÁCTICAS

Sergio Orts Escolano (sorts@ua.es)

Albert García García (agarcia@dtic.ua.es)

José García Rodríguez (jgarcia@dtic.ua.es)

PRÁCTICA 5: CUDA OPENGL INTEROPERABILITY

En este laboratorio se propone la utilización de forma conjunta de dos tecnologías muy relacionadas, CUDA y OpenGL. OpenGL es la tecnología más utilizada para crear aplicaciones capaces de producir gráficos 2D y 3D. OpenGL nos presenta una interfaz capaz de dibujar escenas tridimensionales complejas a partir de la utilización de primitivas geométricas más simples. Para llevar a cabo esta visualización se hace un uso intensivo de la GPU. Como ya vimos, OpenGL no es un lenguaje que permita fácilmente al programador utilizar la capacidad de cómputo de la GPU para computar algoritmos tradicionalmente ejecutados en la GPU, pero no por ello deja de tener importancia y como veremos es posible utilizar esta tecnología de forma conjunta con CUDA para llevar a cabo el cómputo de operaciones complejas y su visualización en tiempo real. Un ejemplo de esto es la posibilidad de calcular la interacción que producen entre sí un conjunto de partículas y su posterior visualización en tiempo real gracias a OpenGL y su capacidad de modelar escenas tridimensionales complejas. OpenGL es capaz también de visualizar gráficos 2D.

Para llevar a cabo la interoperabilidad entre CUDA y OpenGL tendremos que sincronizar el acceso a la memoria de la GPU de forma que un kernel en CUDA compute los valores necesarios sobre esa zona de memoria y posteriormente OpenGL acceda para visualizarlos.

Para ver como se lleva esto a cabo vamos a renderizar una onda sinusoidal en el tiempo. Los valores de los distintos puntos en el tiempo se calculan en paralelo utilizando CUDA y posteriormente se visualizan en OpenGL.

Paso 1: Abre el proyecto de visual studio “simpleGL” y localiza en el código los distintos pasos mencionados en la sesión de teoría para utilizar de forma conjunta CUDA y OpenGL.

```
Pasos:
1. Crear un VBO (OpenGL)
2. Registrar el VBO en CUDA
3. Mapear (bloquear) el VBO para escribir desde CUDA
4. Ejecutar kernel CUDA para modificar la posición de los vértices
5. Desmapear (liberar) el VBO (CUDA)
6. Renderizar resultados OpenGL
```

Analiza el flujo de ejecución del programa y como las distintas tecnologías se coordinar para finalmente mostrar por pantalla el resultado.

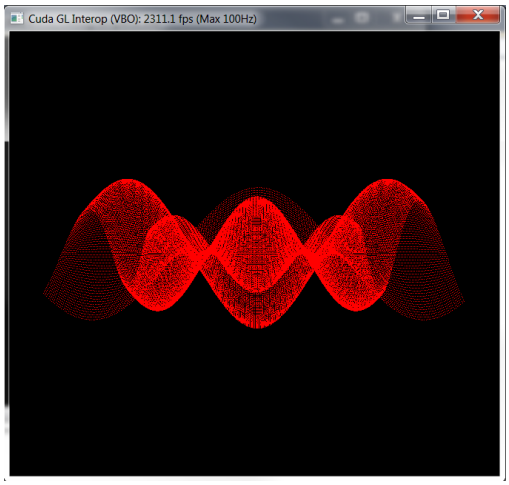

FIGURA 1. VISUALIZACIÓN ONDA SINUSOIDAL OPENGL + CUDA.

Paso 2: Una vez computados los datos podemos visualizarlos en OpenGL de distintas formas. Modifica la instrucción de pintado de puntos por una visualización utilizando líneas, donde cada fila de la malla se dibuje como una línea:

Reemplaza en el código la instrucción `glDrawArrays(GL_POINTS, 0, mesh_width * mesh_height);` dentro de la función `display()` por

```
for(int i=0 ; i < mesh_width*mesh_height; i+= mesh_width)
  glDrawArrays(GL_LINE_STRIP, i, mesh_width);
```

de esta forma cada línea de la malla se visualizará utilizando la primitiva línea. OpenGL nos ofrece muchas posibilidades para visualización 2D/3D.

Paso 3: Abre el proyecto *"cudaOpenGL_PBO"* y analiza el código. Como puedes observar, CUDA puede inter-operar también con el objeto Pixel Buffer de OpenGL permitiendo procesar los píxeles de una imagen y finalmente visualizándolos en OpenGL en tiempo real. Repite el paso 1 y localiza en el código los distintos pasos mencionados anteriormente para utilizar de forma conjunta CUDA y OpenGL.

¿Encuentras alguna diferencia entre la forma de inter-operar CUDA y OpenGL utilizando Pixel Buffer Objects y Vertex Buffer Objects?