Ejercicios Tema 4 – parte 3
(Shaders de cómputo)

Materia de consulta

Se recomienda repasar los siguientes recursos online antes de comenzar a resolver los ejercicios.

- Diapositivas del tema 4 de la asignatura, diap: 64-104.
- Video explicativo de las diapositivas 64-104 del tema 4. (https://aulavirtual.uv.es/mod/kalvidres/view.php?id=593042)
- Red book OpenGL Programming guide, 8th Edition, pag: 623-649. (https://www.cs.utexas.edu/users/fussell/courses/cs354/handouts/Addison.Wesley.OpenGL.Programming.Guide.8th.Edition.Mar.2013.ISBN.0321773039.pdf)
- OpenGL 4 Shading Language Cookbook, 2nd Edition, pag: 345-370. Disponible online a través de VPN en trobes +, Servei de biblioteques de la UV. (https://ebookcentral.proquest.com/lib/univalencia/detail.action?docID=1441782)

Ejercicios Tema 4 – parte 3

Crea un proyecto en Microsoft Visual Studio e incluye en él los ficheros de la carpeta parte 3; compila y ejecuta el programa. En este ejemplo, se dibuja un conjunto de partículas (inicialmente representadas por puntos de colores), distribuidas de forma aleatoria dentro de un cubo imaginario (el cubo no se dibuja en pantalla), alineado con los ejes cartesianos, que se extiende (en el S.R. del mundo) desde -1 hasta 1 para cada una de las coordenadas x,y,z.

3.1 – Partículas como esferas (impostores).

En este ejercicio hay que representar las partículas como esferas (en lugar de puntos) usando la técnica de los “impostores” implementada en el ejercicio de la semana pasada.

Para ello debemos incluir los shaders de vértice, geometría y fragmento usados en el ejercicio 2.3 de la parte 2 del tema 4 (en lugar de los shaders de vértice y fragmento que aparecen en la carpeta de la parte 3), y debemos modificar el shader de geometría, teniendo en cuenta que:

- En la aplicación de la parte 3 definimos como primitivas de dibujo puntos (en el ejercicio de la semana pasada eran triángulos). Este debe ser ahora, por tanto, la primitiva de entrada del shader de geometría.
- La primitiva de dibujo punto está formada por un solo vértice (y no 3, como en el caso de un triángulo). En el ejercicio de la semana pasada situábamos el plano del impostor centrado en el baricentro del triángulo. Ahora lo centraremos en la posición del punto.

3.2 – Shader de cómputo: animación de las partículas

En este ejercicio vamos a animar las partículas con ayuda de un shader de cómputo.

En el código de la aplicación, a cada partícula se le asigna una posición aleatoria dentro del cubo (imaginario), una velocidad y un color. Utilizaremos esta información en el shader de cómputo (en realidad, solo la posición y la velocidad) para actualizar en cada frame de la animación el estado (posición y velocidad) de la partícula empleando simples ecuaciones de cinemática.

Las partículas están encerradas dentro del cubo imaginario y no pueden atravesarlo. Cada vez que chocan con una de las paredes de dicho cubo salen reflejadas en la dirección del vector reflejo. Además, las partículas sufren una aceleración en dirección al suelo consecuencia de la influencia de la gravedad.

Para conseguir el resultado deseado, debemos modificar el código de la aplicación en los siguientes puntos:

- Dentro de la función “initPoints”: se crearán SSBOs (en lugar de VBO) donde se ubicarán los datos de las posiciones y velocidades de las partículas. Una vez hecho, se activaran asignándoles un índice que posteriormente se empleará en el shader de cómputo para acceder a dichos buffers (para leer y escribir sus datos). Además, en la definición del VAO, se empleará el SSBO que almacena las posiciones de las partículas como VBO (en realidad, se reinterpreta su papel dentro de la pipeline gráfica) que contiene las posiciones de los vértices de los puntos.
- Dentro de la función “init”: se creará un nuevo objeto programa (además del que ya hay creado para los shaders gráficos) donde se ubicará el shader de cómputo.
- Dentro de la función “display”: Antes de activar el objeto programa con los shader gráficos, activaremos el objeto programa con el shader de cómputo y lanzaremos su ejecución.

3.2 – Shader de cómputo: amortiguación de las partículas

Modifica la aplicación y el shader de cómputo para que se amortigüe la velocidad de la partícula cada vez que choca con una de las paredes del cubo. El factor de atenuación será un valor entre 0 y 1, que le pasaremos al shader desde la aplicación como una variable uniform. Así mismo, hay que cambiar el código de la aplicación para que cada vez que el usuario pulse la tecla ‘+’ se incremente dicho factor en 0.01 y cuando pulse la tecla ‘-’ se decremente en 0.01.

3.3 – Shader de cómputo: configuración óptima de los parámetros del shader.

Prueba a modificar el número de partículas de la simulación y el tamaño del grupo de trabajo y observa si varía la frecuencia de refresco. Si es así, busca una configuración óptima del tamaño de grupo para conseguir representar el número máximo posible de partículas a frecuencias de refresco interactivas (por encima de 30 frames por segundo).

Nota: En principio, varía el tamaño del grupo de trabajo, fijando como tal, valores que sean potencia de 2 no superiores a 1024. Esto es porque el número de núcleos de un multiprocesador es también potencia de 2 (normalmente 16 o 32 núcleos por multiprocesador) y el número máximo de hilos que un núcleo pueden gestionar de forma simultanea es del orden de una decena.