Procesadores Gráficos y Aplicaciones en Tiempo Real

Sistema de partículas por GPU

Álvaro Muñoz Fernández
Iván Velasco González

1. Introducción

El objetivo de esta práctica consiste en la implementación de un modelo de partículas básico implementado en shaders de cómputo que se ejecutarán en GPU. Además se proponen tareas adicionales, de las que se ha optado por la implementación de la ordenación de partículas (también realizada en shaders de cómputo) para una correcta visualización de las partículas.

2. Aplicación cliente

La aplicación se ha implementado en C++ partiendo de un proyecto vacío en el que se han añadido las librerías necesarias para el desarrollo. Se ha utilizado *freeGLUT* para la inicialización del contexto, *GLEW* para la inicialización de extensiones, *GLM* para las funciones matemáticas y estructuras de datos algebraicas, y *FreeImage* para la carga de texturas, aunque finalmente no se ha cargado ninguna textura en el proyecto.

El proyecto se divide en funciones de inicialización, funciones de cómputo y funciones de renderizado que son llamadas en los puntos de ejecución donde son necesarias. Además se emplean estructuras de datos para almacenar la información correspondiente a los programas utilizados.

3. Sistema de partículas

El sistema de partículas que se ha implementado es el propuesto por Mike Bailey, que incluye solamente colisiones entre las partículas y una esfera definida mediante su posición y radio. Opcionalmente podía extenderse este sistema, pero nosotros hemos optado por la implementación de ordenación y renderizado con transparencia.

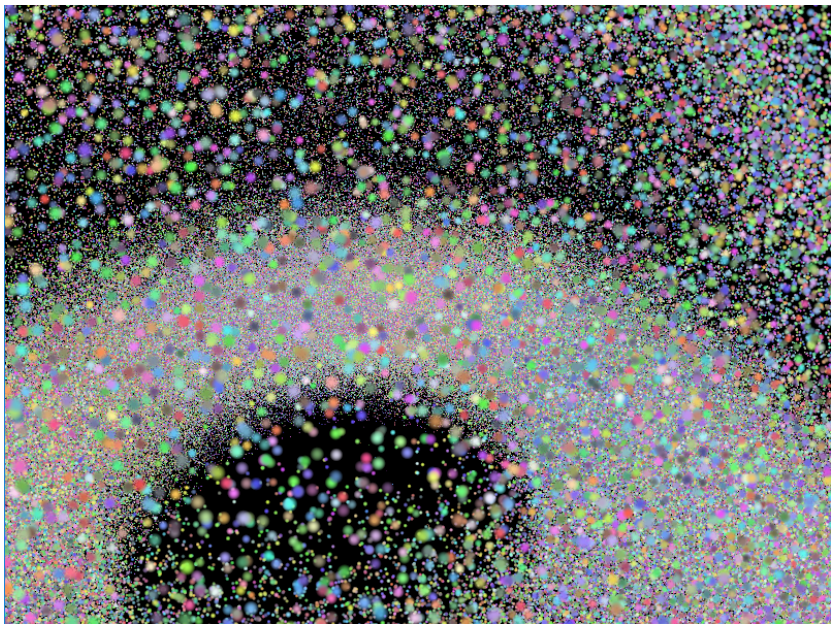

Figura 1: Instantánea del sistema de partículas

El sistema de partículas se almacena en tres buffers que contienen las posiciones,

velocidades y colores de las partículas.

4. Ordenación

4.1. Inicialización

Antes de proceder a la ordenación de las partículas es necesario inicializar dos buffers extra. Uno de ellos almacena los índices que se emplearán en la llamada *DrawElements()* para realizar el dibujado ordenado, y el otro almacenará las distancias de cada partícula a la cámara.

Este paso previo se realiza en una shader de cómputo que exclusivamente almacena, de forma paralela, la distancia de cada una de las partículas en el array de distancias.

4.2. Ordenación

La ordenación se realiza en una nueva shader de cómputo que realiza varias pasadas. Hemos optado por una implementación de *bitonic sort*, que recibe el buffer de distancias y el buffer de índices y realiza la ordenación solamente del buffer de índices.

Con este buffer ordenado puede realizarse el renderizado ordenado a través de un VAO sin necesidad de modificar los buffers con información de partículas. Se ha tomado esta decisión porque aunque incrementa el uso de memoria, simplifica el número de accesos ya que se escribe sobre un solo buffer de valores enteros utilizando la información leída de un buffer auxiliar, en lugar de modificar los 3 buffers de información de cada partícula, cada uno de ellos con 3 elementos float por cada partícula.

5. Renderizado

5.1. Billboards

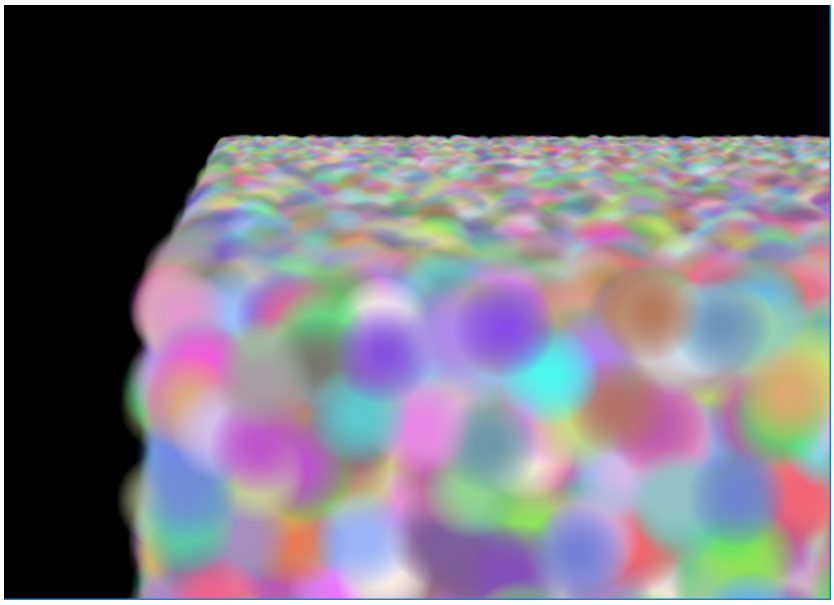

Figura 2: Varios billboards orientados a cámara

Para dibujar cada una de las partículas se ha creado una shader de geometría que recibe las posiciones de las partículas como puntos y las traslada a posiciones en el espacio de cámara. En este espacio el eje Y está orientado perpendicularmente a la cámara, por lo que al generarse un quad siempre estará orientado hacia la cámara. De esta forma se consigue el efecto billboard con facilidad, generándose los quads sobre los que dibujar la partícula de forma eficiente en la shader de geometría. Además se generan coordenadas UV en cada vértice de la partícula que permitirán colorear correctamente la partícula en la shader de fragmentos.

5.2. Texturizado

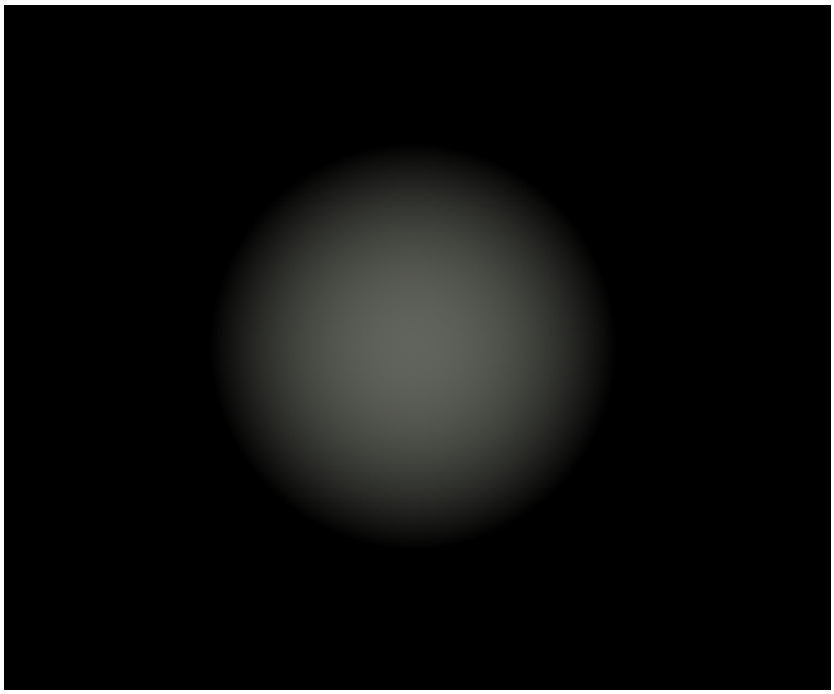

Figura 3: Visualización de una partícula con transparencia

Para colorear las partículas se utiliza el color que se asigna aleatoriamente al inicio de la simulación. En lugar de utilizar una textura como máscara, se ha generado procedimentalmente la máscara de las partículas. Mediante la distancia al centro de la partícula se calcula el porcentaje de opacidad de la partícula, decayendo este cuanto más lejos del centro está un fragmento. Así se consigue una forma esférica suavizada para las partículas, con una transparencia variable que permite que algunas zonas sean translúcidas y se vea con facilidad el resultado de la aplicación del blending.

6. Funcionamiento de la demo

6.1. Tamaño de partícula

Se ha añadido funcionalidad para alterar el tamaño de partícula en tiempo de ejecución. Esto puede hacerse mediante las teclas + y -.

6.2. Pausa de simulación

La simulación se puede pausar y reanudar presionando la tecla **P**.

6.3. Reinicio de simulación

La simulación se puede reiniciar presionando la tecla **R**.

6.4. Alternar ordenado

El ordenado de partículas puede activarse y desactivarse presionando la tecla **T**.

6.5. Estadísticas

Presionando la tecla **L** se muestran por consola las estadísticas de la media de frametime y framerate de los últimos 50 frames.

6.6. Cámara

La cámara se puede manipular mediante las teclas **W**, **A**, **S**, **D** y con el movimiento del ratón mientras se mantiene presionado el botón izquierdo. Además se puede alterar la elevación de la cámara utilizando la tecla **Espacio** para ascender y **Z** para descender.

7. Análisis de resultados

7.1. Metodología

Para las pruebas se han realizado varias ejecuciones en las que se ha alterado el parámetro del tamaño de WorkGroup de las shaders, se han tomado 10 medidas para ejecuciones con cada uno de los parámetros y se han calculado las medias para cada ejecución. Las medidas se han desglosado entre el tiempo dedicado al cálculo de físicas, el tiempo de ordenamiento de las partículas y el tiempo de dibujado de las partículas.

Las pruebas se han realizado con un número de partículas de 1024x1024.

Las especificaciones del equipo de pruebas son las siguientes:

- Procesador: AMD Ryzen 1600X 3.6GHz
- Memoria RAM: 16GB DDR4 3000Mhz
- Tarjeta gráfica: Nvidia GTX 1060 6GB

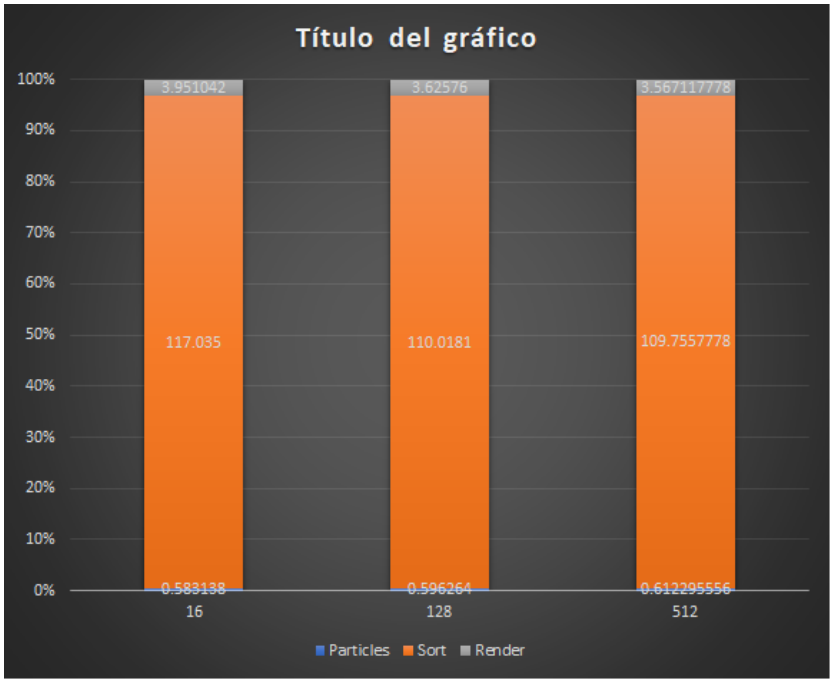

Figura 4: Histograma de las medias de tiempos

7.2. Media de resultados por ejecución

7.3. Tablas de resultados

Frametimes para tamaño de WorkGroup 16			
Iteración	Físicas	Ordenación	Rendering
1	0.35158	117.294	4.47528
2	0.50754	117.188	4.38544
3	0.56222	117.225	4.27188
4	0.55212	117.019	4.16744
5	0.56998	117.021	3.96658
6	0.5778	116.986	3.76458
7	0.62348	116.977	3.73252
8	0.66744	116.97	3.70164
9	0.70276	116.96	3.60998
10	0.71646	116.71	3.43508
Media	0.583138	117.035	3.951042

Frametimes para tamaño de WorkGroup 128			
Iteración	Físicas	Ordenación	Rendering
1	0.49302	110.307	4.32266
2	0.54266	110.307	4.27152
3	0.55578	110.156	4.02068
4	0.5653	110.002	3.75822
5	0.5773	109.968	3.69862
6	0.58566	109.99	3.57418
7	0.62238	110.002	3.38232
8	0.64676	109.803	3.18426
9	0.67342	109.816	3.07414
10	0.70036	109.83	2.971
Media	0.596264	110.0181	3.62576

Frametimes para tamaño de WorkGroup 512			
Iteración	Físicas	Ordenación	Rendering
1	0.56121	109.93	4.07228
2	0.54918	109.948	4.15572
3	0.55176	109.77	3.81522
4	0.55892	109.702	3.72278
5	0.60886	109.705	3.59522
6	0.64248	109.755	3.3695
7	0.6717	109.786	3.2267
8	0.67832	109.58	3.03026
9	0.67306	109.57	2.9145
10	0.57638	109.986	4.27416
Media	0.612295556	109.7557778	3.567117778

7.4. Conclusiones

De los datos obtenidos hemos concluído que el grueso del tiempo de ejecución se debe a la ordenación de partículas. Nuestra implementación de bitonic no es la más óptima, haciendo accesos a memoria global constantes, por lo que el tiempo de ejecución de la ordenación corresponde a más de un 90 % del tiempo total de cada frame.

Además se puede observar que al variar el tamaño del WorkGroup hay ligeras variaciones en el tiempo de ejecución, afectando principalmente a la etapa de ordenación. Cuando el tamaño se incrementa se reducen los tiempos de ordenación, aunque el tiempo de físicas parece no cambiar y los resultados entran dentro del margen de error. En el caso de la etapa de dibujado no hay efecto, ya que el tamaño de grupo no interviene en el dibujado.