CEIG'09, San Sebastián, Sept. 9-11 (2009)
C. Andújar and J. LLuch (Editors)

Simulación de Lluvia sobre Escenas Dinámicas

Nicolau Sunyer[1], Anna Puig-Centelles[2], Oscar Ripolles[2], Miguel Chover[2], Mateu Sbert[1]

[1] Graphics & Imaging laboratory, Institut d'Informàtica i Aplicacions, Girona
{nsunyer,mateu}@ima.udg.edu
[2] Departamento de Lenguajes y Sistemas Informáticos, Universitat Jaume I, Castellón
{apuig,oripolle,chover}@uji.es

Abstract

La lluvia es un fenómeno complejo y su simulación suele ser muy costosa. En este artículo se propone un sistema de simulación de lluvia mediante sistemas de partículas que funciona únicamente en la tarjeta gráfica (GPU). La flexibilidad de CUDA permite incluir, además de la simulación de la precipitación, un sistema de detección y manejo de las colisiones de las partículas contra el escenario que permite simular al mismo tiempo las salpicaduras. En el apartado de resultados mostramos cómo el uso de CUDA permite mejorar el rendimiento que se obtenía utilizando métodos anteriores.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.7]: Virtual reality—; Computer Graphics [I.3.3]: Display algorithms—

1. Introducción

El uso de un sistema de simulación de lluvia eficiente y realista aumenta considerablemente el realismo de escenas de exteriores. La mayor parte de los sistemas propuestos para la simulación de lluvia en tiempo real se basan en el uso de sistemas de partículas [KTT01, WLF*06, Tar, CWZ*08]. Sin embargo, el uso de sistemas de partículas para representar lluvia puede tener importantes limitaciones debido al coste que supone el manejo de grandes cantidades de partículas.

Una posible solución para superar este tipo de limitaciones es el uso del hardware gráfico, cuya constante evolución ofrece nuevas posibilidades para la informática gráfica. Algunos de los últimos trabajos incluyen la utilización del reciente *Geometry Shader* [Tar, PCRC08]. En este sentido, recientemente se presentó CUDA (Compute Unified Device Architecture) como una tecnología cuyo objetivo es aprovechar al máximo la capacidad de proceso de las tarjetas gráficas actuales para resolver problemas con una gran carga computacional [CUD07].

En este trabajo se propone un sistema de simulación de lluvia que se controla y actualiza únicamente en la tarjeta gráfica. La solución propuesta se basa en la utilización de sistemas de partículas. Este sistema obtiene un rendimiento muy elevado gracias al uso de CUDA, puesto que libera considerablemente a la CPU de una enorme carga de operaciones. CUDA ofrece un marco de trabajo de una gran flexibilidad que nos permite incluir la implementación de un sistema de detección y manejo de las colisiones de las partículas de la lluvia contra el escenario para la generación de salpicaduras, como se puede ver en el ejemplo de la Figura 1.

Este artículo presenta la siguiente estructura. La sección 2 resume el trabajo previo en simulación de lluvia, centrándose en aquellos modelos que utilizan el hardware gráfico para acelerar y mejorar la visualización. La sección 3 presenta las características del modelo de lluvia presentado, incluyendo el subsistema encargado del cálculo de las colisiones. Posteriormente, la sección 4 presenta los resultados obtenidos. Finalmente la sección 5 resume las aportaciones de nuestro trabajo y esboza el trabajo futuro.

2. Trabajo previo

Durante los últimos años, los investigadores han desarrollado distintos modelos para la visualización de lluvia dentro del campo de la informática gráfica. Tradicionalmente, la precipitación de las gotas se ha simulado con dos métodos distintos:

- **Texturas deslizantes**

DOI: 10.2312/LocalChapterEvents/CEIG/CEIG09/247-250

Figura 1: *Entorno con lluvia muy intensa.*

Las soluciones que utilizan esta familia de técnicas desplazan una textura de lluvia que cubre toda la escena, simulando le movimiento de caída de la lluvia [WW04]. Estos métodos no son capaces de crear una impresión de precipitación verdaderamente convincente, ya que existe el problema de la falta de sensación de profundidad en el escenario. Para superar esta limitación, algunos autores [Tat06] incluyen diferentes capas o texturas deslizantes para simular la lluvia a diferentes distancias, aunque el problema no se soluciona completamente. Otro problema que cabe destacar es que los movimientos de cámara son limitados; si el observador eleva la vista hacia el cielo, la sensación realista de la lluvia se pierde por completo.

- **Sistema de partículas**
  Las técnicas que utilizan la solución de sistemas de partículas representan eficientemente escenas lluviosas [KTT01, RJG06, WLF*06, Tar, CWZ*08]. El principal problema es el hecho que no son adecuados para escenarios donde el usuario se mueve rápidamente, puesto que ajustar todo el sistema de partículas a las nuevas condiciones de la cámara resulta muy costoso. Los modelos más recientes [Tar, PCRC08] proponen soluciones que maximizan el uso de las posibilidades del hardware gráfico para procesar y simular la lluvia trabajando directamente en la tarjeta gráfica.

3. Simulación de Lluvia

La utilización del hardware gráfico ha permitido acelerar la visualización de la lluvia de forma considerable. De esta manera, han surgido varias soluciones que utilizan *shaders*, no solamente para ofrecer efectos finales, sino también para procesar eficientemente los sistemas de partículas. En este sentido, otros autores han aportado métodos en los que se trasladan las partículas en la propia tarjeta gráfica, guardando las posiciones calculadas utilizando técnicas como el *render-to-vertex* [RJG06] o el más reciente *Stream Output* [Tar, PCRC08] introducido en el Shader Model 4.0.

Tradicionalmente, los modelos de simulación de lluvia en la tarjeta gráfica [RJG06, Tar, PCRC08] realizan dos pasadas de la *pipeline* gráfica para obtener la geometría. La primera pasada traslada las partículas siguiendo el movimiento de caída de las gotas, mientras que en la segunda pasada el sistema se encarga de crear los cuatro puntos que forman el *quad* utilizado para simular cada una de las gotas.

Mediante CUDA somos capaces de optimizar el uso de la tarjeta gráfica. Obviamente, seguirá siendo necesario que realicemos dos pasadas. La primera de ellas se realizará con CUDA y la segunda con OpenGL, ya que no es posible realizar la actualización y la visualización del sistema de partículas en una única pasada de GPU, puesto que CUDA no es una librería gráfica que permita visualizar geometría.

3.1. Uso de CUDA

El uso de CUDA resulta muy interesante por la flexibilidad que permite respecto al uso de los métodos basados en librerias gráficas. En CUDA se genera un hilo de ejecución para cada elemento del problema a tratar (en nuestro caso para cada gota). Cada hilo puede generar un número variable de resultados de salida en posiciones que no tienen porque seguir ningún patrón fijo (*scatter*). Además pueden generar estos resultados de salida en los vectores de salida que se decida para cada hilo. Finalmente CUDA permite operaciones atómicas, cosa que los shaders no permiten. Todas estas características nos permiten, por ejemplo, generar un vector dinámico de colisiones en el mismo paso en el que se genera la simulación de las gotas de lluvia y la detección de estas colisiones. Algo que no es posible con el uso de los métodos gráficos anteriores.

3.2. Nuestro Modelo de Lluvia

En nuestro trabajo proponemos la utilización de CUDA para optimizar la actualización de las partículas del sistema de simulación de lluvia. La utilización de este marco de trabajo nos permite, por una parte, reducir el tiempo empleado en el cálculo de las nuevas posiciones y, al mismo tiempo, incorporar nuevas funcionalidades que, sin CUDA, serían muy complejas de obtener.

La principal ventaja que ofrece nuestra solución es que todos los cálculos se realizan en un único *kernel* de CUDA. La forma en que CUDA trabaja con las posiciones de memoria permite guardar información en varios buffers al mismo tiempo, utilizar buffers de lectura/escritura y decidir en qué posición de los buffers guardar la información generada. Así pues, podremos almacenar al mismo tiempo las posiciones transformadas y los *quads* orientados a la cámara.

La Figura 2 resume la distribución de los datos en la memoria de la tarjeta gráfica que proponemos para la utilización de CUDA. Podemos ver cómo será necesario almacenar en memoria las posiciones originales de las gotas y su velocidad. Además, necesitamos un buffer para almacenar la posición actual de las partículas y otro para los *quads* generados a partir de la posición y la orientación de la cámara. Así, el vector de posiciones se utiliza para realizar los

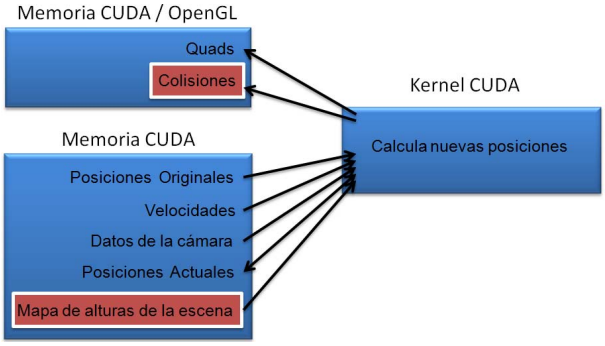


Figura 2: *Diagrama del uso de memoria con colisiones.*

cálculos de traslación y colisiones sobre una única posición mientras que el cálculo de los *quads* permite la visualización del sistema de partículas. Es importante comentar que si la partícula sufre una colisión es necesario recolocarlas sobre el campo de visión del usuario para continuar con la simulación de la lluvia, como veremos en la sección siguiente. Así, esta figura incluye, sombreadas en rojo, las estructuras que son necesarias para realizar el cálculo de las colisiones y la simulación de las salpicaduras.

3.2.1. Colisiones

Uno de los objetivos de nuestro sistema propuesto es ofrecer mayores prestaciones y, en este sentido, consideramos que el cálculo de las colisiones es un requisito muy importante. Alguno de los modelos que hemos introducido en la sección de trabajo previo incluyen la detección de colisiones. Feng et al. [FTDC06] presentan un método de detección de colisiones de las partículas e incluye un subsistema para simular las salpicaduras. Más recientemente, el método introducido en [Tar] propone la simulación de las salpicaduras repitiendo la aplicación de texturas de relieve sobre el escenario, aunque el resultado es poco preciso.

En nuestro artículo proponemos un sistema sencillo, aunque sería posible utilizar métodos más complejos sin ninguna dificultad técnica. El método aplicado consiste en realizar *mapa de alturas* de la escena que estamos visualizando. Esta captura se realiza para cada imagen que creamos, de manera que el sistema es capaz de adaptar el cálculo de colisiones a los movimientos de personajes y otros objetos del entorno. Este mapa de alturas es almacenado en forma de textura dentro de la memoria de la tarjeta gráfica.

El *kernel* de CUDA que se propone incluye el cálculo de colisiones, de manera que cada vez que desplazamos una partícula comparamos su posición resultante con la altura almacenada en la textura comentada anteriormente. Así, si detectamos una colisión recolocaremos las partículas en la zona superior de la escena y, además, añadiremos una partícula a nuestro subsistema de partículas encargado de las colisiones. Este subsistema almacena las posiciones de los lugares donde se producen colisiones. En la solución propuesta crearemos un *quad* en esta posición que texturizaremos con una imagen que simula la dispersión del agua tras la colisión. Del mismo modo, sería posible crear varias partículas que simularan las gotas creadas a partir de la gota colisionada, que podrían seguir una nueva trayectoria.

4. Resultados

En esta sección analizaremos la solución propuesta desde varias perspectivas ya que, además del rendimiento, nos interesa analizar la calidad visual de la simulación obtenida. La escena utilizada en los diversos *tests* es la presentada en la Figura 1. Todas las pruebas se han realizado sobre Windows XP Professional en un ordenador con un procesador Intel QuadCore Q6600 a 2.4GHz, 4 GB. de RAM y una tarjeta gráfica GeForce GTX-280 con 1 GB. de RAM.

4.1. Colisiones y Salpicaduras

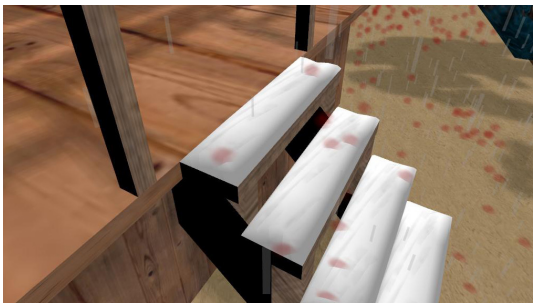
La Figura 3 presenta dos imágenes detalle de la escena. En este caso, las salpicaduras se han coloreado en rojo para que sea más sencillo distinguirlas del entorno. La primera de ellas permite ver cómo el sistema se adapta a la geometría de la escena, ya que se puede observar cómo las salpicaduras tienen lugar sobre los escalones de la cabaña o sobre el guerrero montado en el dragón. Por otra parte, la imagen 3(b) muestra una perspectiva de los escalones, pudiendo ver cómo las salpicaduras nunca se generan dentro de la cabaña, ya que las gotas colisionan sobre su techo. Finalmente, es importante comentar que nuestro método realiza la captura del mapa de alturas en cada *frame* generado, de manera que el cálculo de colisiones se adapta a los movimientos de personajes y objetos en el escenario.

4.2. Rendimiento

La utilización de CUDA para la simulación de la caída de las gotas ofrece importantes ventajas en cuanto a rendimiento. La Tabla 1 presenta un estudio del rendimiento (fps) de nuestro sistema al simular distintas cantidades de lluvia. Además, incluimos por separado el coste del cálculo de las colisiones, para poder analizar cuánto afecta la simulación de las salpicaduras al rendimiento de nuestra propuesta. De los resultados podemos concluir que el sistema de simulación propuesto es capaz de simular una gran cantidad de gotas en tiempo real. Por otra parte, el cálculo y la visualización de las salpicaduras incrementa el tiempo de procesado en un 15% de media. Es importante comentar que, cuantas más partículas maneja el sistema de lluvia, más aumenta el porcentaje de tiempo que es necesario dedicar a las colisiones.

En este apartado también queremos comparar nuestra propuesta con métodos basados en el uso de la *pipeline* gráfica tradicional [Tar, PCRC08]. En este caso, el método con el que nos compararemos no realiza cálculos de colisiones.

(a) Perspectiva de la escena.



(b) Detalle de los escalones de la cabaña.

Figura 3: *Ejemplos de la simulación de lluvia con colisiones.*

Núm. Gotas	CUDA	CUDA (sin col.)	Shaders (sin col.)
500.000	76,43	83,14	127,22
1.000.000	50,34	56,12	76,92
2.000.000	30,12	35,71	43,29
4.000.000	15,62	19,02	21,78

Tabla 1: *Comparación de resultados para entornos de lluvia de distintas intensidades (fps).*

Podemos decir que el rendimiento obtenido con *shaders* aumenta un 30% de media. En este caso, la diferencia entre ambos métodos disminuye conforme visualizamos más partículas, de manera que si queremos simular 4 millones de partículas la diferencia en rendimiento es de un 14%.

Por otra parte, es importante comentar que el uso que el sistema de lluvia hace de la CPU en uno y otro caso es muy diferente. Analizando en el sistema el porcentaje de uso de la CPU durante 30 segundos de ejecución de la simulación, hemos podido ver que la simulación de lluvia basada en *shaders* ocupa un 57% del tiempo de CPU, mientras que con CUDA esta ocupación se reduce a un 26%. Las llamadas a funciones por parte de CUDA son asíncronas, permitiendo que la simulación sea más fluida.

5. Conclusiones

En este artículo hemos presentado un conjunto de técnicas para visualizar eficientemente escenarios con lluvia de diversas intensidades con un aspecto realista. La mejora de la simulación ha sido posibles gracias al uso de CUDA, cuya flexibilidad permite ofrecer técnicas avanzadas como la detección colisiones. Así, la calidad obtenida en las diversas pruebas es debida a la posibilidad de incluir una cantidad elevada de partículas y salpicaduras. Además, CUDA permite que la aplicación gráfica reduzca el uso de CPU un 50%, lo que permite que la aplicación dedique ese tiempo a realizar otros cálculos.

La solución propuesta ofrece ventajas y nos anima a mejorar la simulación. En este sentido, actualmente nos interesa maximizar el uso de las colisiones para ofrecer mejores efectos, como la acumulación de lluvia en el terreno. Además, consideramos interesante estudiar el funcionamiento de la precipitación y las salpicaduras en situaciones de viento, de manera que el sistema resuelva correctamente situaciones donde las gotas no caen de forma totalmente vertical.

Agradecimientos

Este trabajo ha sido subvencionado por el MCT (beca TSI-2004-02940 y proyectos TIN2007-68066-C04-02 y TIN2007-68066-C04-01) y por Bancaja (P1 1B2007-56).

References

[CUD07] Nvidia cuda compute unified device architecture - programming guide, 2007.

[CWZ*08] CHANGBO W., WANG Z., ZHANG X., HUANG L., YANG Z., PENG Q.: Real-time modeling and rendering of raining scenes. *Visual Computer 24* (2008), 605–616.

[FTDC06] FENG Z., TANG M., DONG J., CHOU S.: Real-time rain simulation. In *CSCWD 2005, LNCS 3865* (2006).

[KTT01] KUSAMOTO K., TADAMURA K., TABUCHI Y.: A method for rendering realistic rain-fall animation with motion of view. *IPSJ SIG*, 106 (2001), 21–26.

[PCRC08] PUIG-CENTELLES A., RIPOLLES O., CHOVER M.: Multiresolution techniques for rain rendering in virtual environments. In *Int. Symp. on Comp. and Information Sciences* (2008).

[RJG06] ROUSSEAU P., JOLIVET V., GHAZANFARPOUR D.: Realistic real-time rain rendering. *Computers & Graphics 30*, 4 (2006), 507–518.

[Tar] TARIQ S.: Rain nvidia whitepaper. http://developer.download.nvidia.com/whitepapers/2007/SDK10/RainSDKWhitePaper.pdf.

[Tat06] TATARCHUK N.: Artist-directable real-time rain rendering in city environments. In *ACM SIGGRAPH Courses* (2006).

[WLF*06] WANG L., LIN Z., FANG T., YANG X., YU X., KANG S.: Real-time rendering of realistic rain. In *ACM SIGGRAPH 2006 Sketches* (2006), p. 156.

[WW04] WANG N., WADE B.: Rendering falling rain and snow. In *ACM SIGGRAPH 2004 Sketches* (2004), p. 14.