

Procesamiento de Flujos de Vídeo mediante Hardware Gráfico de Consumo

Antonio Sanz Raúl Cabido Juan José Pantrigo Ángel Sánchez

antonio.sanz@urjc.es rcabido@gmail.com juan jose.pantrigo@urjc.es angel.sanchez@urjc.es

DIET, ESCET, Universidad Rey Juan Carlos

28933 Móstoles (Madrid)

Resumen

En los últimos años las tarjetas gráficas de consumo han evolucionado debido principalmente a la demanda de la industria de los videojuegos. Estas tarjetas son actualmente programables y capaces de procesar grandes cantidades de datos siguiendo un modelo SIMD. En este trabajo se formulan algunas ideas para el procesamiento de vídeo en tiempo real, donde la tarjeta gráfica de consumo de un PC es el principal procesador de vídeo.

1. Motivación

El procesamiento de vídeo en tiempo real es una tarea costosa, que involucra el uso de sistemas de altas prestaciones. Por otra parte, hay una creciente migración de los sistemas de vídeo analógico a digital a causa del interés en las tecnologías inteligentes en multitud de aplicaciones comerciales, industriales o militares [1]. Muchas de estas aplicaciones necesitan hardware de propósito específico [3]. También el mercado multimedia ha evolucionado para incluir pequeños dispositivos de visualización que demandan soluciones de bajo coste. Aunque existen soluciones software económicas, no hay muchas alternativas hardware de uso común.

Sin embargo, la industria de los videojuegos ha fomentado el desarrollo del hardware gráfico para mejorar sustancialmente su funcionalidad y potencia. Dichos procesadores no deben menospreciarse y se ha demostrado que las unidades de procesamiento gráfico (*Graphics Processing Units* o *GPUs*) sobrepasan la potencia de la mayoría de las CPUs más comunes [8, 7, 4]. Asimismo, estas GPUs pueden ser programadas para personalizar sus *pipelines* de renderizado.

Los programadores pueden aprovechar la potencia las GPUs incluso en aplicaciones distintas de la visualización o renderizado. Así, las GPUs pueden llegar, al menos de forma teórica, a comportarse a modo de coprocesador de la CPU. En la literatura se han propuesto aplicaciones sobre GPUs para realizar cálculos algebraicos [6], simulaciones de procesos físicos [4], procesamiento de imágenes y volúmenes 3D [5] entre otras [2].

En este trabajo aplicamos la tecnología más propia del campo de la informática gráfica al tratamiento de imagen y vídeo.

2. Hardware gráfico

Tradicionalmente, las GPUs han implementado un *pipeline* fijo para el procesamiento de descripciones primitivas, pero sus etapas fijas se han reemplazado por componentes programables. Éstos son: la etapa de transformación e iluminación (T&L) y la de multitexturas, que ofrecen gran versatilidad al programador.

La aceleración hardware de las GPUs es expuesta a los desarrolladores para la creación de programas especializados llamados *shaders*. Éstos se cargan en la tarjeta gráfica para reemplazar la funcionalidad fija, existiendo dos clases, respectivamente *vertex* y *fragment shaders*. Los *shaders* son utilizados principalmente para renderizar complejos efectos especiales sobre escenas fotorrealistas 3D en tiempo real, sin embargo la computación de la GPU al nivel del *fragment shader* encaja muy bien con el modelo de computación sobre flujos de datos (*stream computation*). De esta manera, una operación se ejecuta sobre un número elevado de fragmentos en forma SIMD [9]. Estas operaciones son muy eficientes cuando no existen interdependencias entre los datos

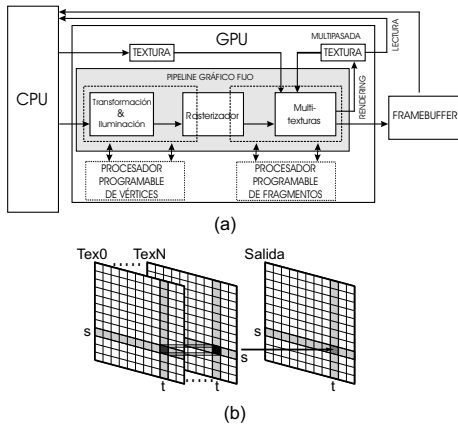


Figura 1: (a) Modelo de programación CPU/GPU. Una vez habilitados, los caminos de ejecución de vértices y fragmentos pasan a través de los nuevos *vertex* y *fragment shaders*. (b) Esquema simplificado de un *fragment shader* aplicado a unas texturas (Tex_0 - Tex_N).

y pocas ramificaciones, permitiendo una alta coherencia de caché. En la Figura 1(a) se esquematiza el modelo básico de una arquitectura CPU/GPU.

El programador, por lo tanto, es responsable de organizar los datos en una estructura reticular y convertirlos en una textura. Para conseguir la máxima eficiencia es deseable llenar los cuatro canales de color de la textura, pues el coste de procesamiento de una única componente es similar al de la cuadrupla (RGBA). Los datos se fijan a una retícula con el propósito de operar sobre sus fragmentos. Para ello, se crea un *fragment shader* personalizado que realiza un procesamiento sobre las texturas de entrada. Forzando una renderización se aplica la operación del *shader* sobre cada uno de los fragmentos de las texturas habilitadas. Una vista esquemática de este proceso se muestra en la Figura 1(b). El resultado puede ser redirigido a la entrada en un modo multipasada para continuar las tareas de procesamiento.

3. Procesamiento de vídeo

En esta sección se exponen dos aplicaciones de procesamiento de vídeo usando la GPU como procesa-

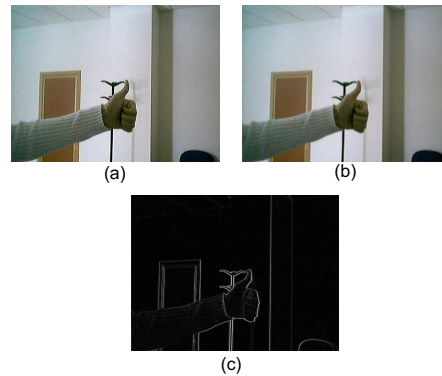


Figura 2: (a) Cuadro original para el siguiente procesamiento. (b) Suavizado con máscara de tamaño 7×7 y (c) detector de bordes Sobel 3×3 .

dor de datos. Para ambas se considera una etapa de inicialización, en la que los frames son transferidos como texturas a la memoria de vídeo.

3.1. Filtrado de vídeo en tiempo real

En las más recientes familias de GPUs, es posible acceder de forma directa a los *texels* a través de índices (s, t) , por lo que el filtrado de una imagen resulta una tarea trivial.

Dado el cuadro actual del vídeo como una textura (Tex_0), un *fragment shader* personalizado se habilita para procesarlo, y la simple renderización fuerza a realizar la operación sobre los datos de la imagen.

En la Figura 2 se muestran ejemplos de procesamiento en tiempo real sobre secuencias de dimensiones 640×480 . En concreto, se aplican filtros de suavizado y detección de bordes basados en convoluciones. Para el filtro de Sobel es necesario realizar una conversión RGB a escala de grises, aunque se aprovecha el mismo *fragment shader*.

3.2. Desentrelazado de vídeo

En un formato de vídeo entrelazado, los cuadros se dividen en campos pares e impares, que se transfieren secuencialmente. El desentrelazado de vídeo es la técnica para reconstruir el cuadro completo a partir de cada campo, o grupo de campos. Los métodos más sencillos se basan en interpolaciones en tiempo

o espacio. Las técnicas temporales son efectivas en escenas estáticas mientras que las espaciales trabajan mejor para las dinámicas. Sin embargo, ambas pueden ser combinadas inteligentemente.

3.2.1. Repetición espacial

Las líneas que faltan son completadas con información contenida en el campo actual. Así, la paridad del campo se pasa como parámetro de la aplicación al *fragment shader*. Entonces, la coordenada vertical de cada texel es evaluada para comprobar si está en una línea sin información. Si el fragmento (x, y) pertenece a una línea sin información se aplica la repetición espacial a través de la adyacente $(x, y - 1)$. Una limitación intrínseca de estos métodos es la pérdida de resolución vertical en el cuadro resultante (ver Figura 3a).

3.2.2. Repetición temporal

Mediante este método, las líneas no transferidas se completan directamente con información del campo anterior. Así, en este caso se necesitan 2 texturas (*Tex0* y *Tex1*) que contienen información de ambos campos. De nuevo, se pasa la paridad al *fragment shader* para determinar si el fragmento pertenece o no a una línea sin información. Los fragmentos que faltan $((x, y)$ en tiempo t) son reemplazados con información previa $((x, y)$ en tiempo $t - 1$) generando defectos en la imagen en aquellas zonas donde ha existido movimiento (ver Figura 3b).

3.2.3. Desentrelazado adaptativo por movimiento

Como combinación de las técnicas anteriores se propone un desentrelazado adaptativo por movimiento. Para ello se utiliza una combinación lineal en función de la cantidad de movimiento detectada:

$$I_s(i, j, t) = (1 - \alpha)I_t(i, j, t) + \alpha I_s(i, j, t) \quad (1)$$

donde α es la cantidad de movimiento por píxel que viene dada por la diferencia de intensidades en escala de grises entre campos consecutivos de igual paridad. Para mejorar la estimación de movimiento se suaviza el resultado en una segunda pasada mediante el filtrado descrito anteriormente.

En consecuencia, se hace necesario el uso de 4 pasadas para recoger los resultados del factor de

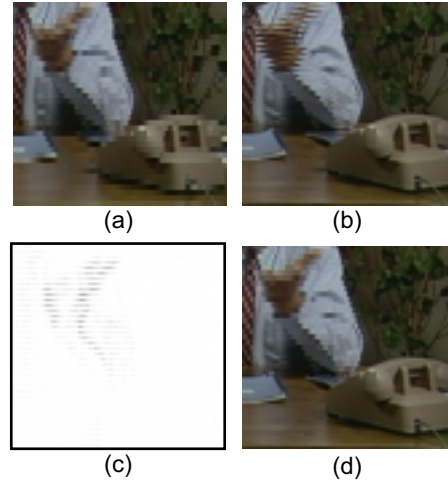


Figura 3: Detalle de una escena del vídeo *Salesman* para los diferentes métodos expuestos. (a) Repetición espacial. (b) Repetición temporal. (c) Factor α (representado en intensidades invertidas para acentuar el contraste) y (d) combinación adaptativa.

movimiento α , el suavizado, el cálculo de I_t y el de I_s , lo que genera una carga computacional muy elevada.

4. Resultados experimentales

Para este trabajo se ha usado una de las últimas GPUs lanzadas al mercado, la Nvidia 6800GT (NV40, drivers v66.93) en un Pentium 4 a 2.8GHz, 512 MB RAM, AGPx8, bajo Windows XP Profesional SP2.

En la Figura 3 se muestra un detalle de cada método de desentrelazado para el vídeo test de *Salesman*. Se observa que el método espacial hace perder resolución vertical, y los detalles quedan pixelados. La repetición temporal degrada mucho las zonas dinámicas, aunque la reconstrucción de las estáticas es perfecta (compárese la zona del teléfono). El cálculo del factor α se representa en (c), utilizado por la combinación espacio-temporal que se muestra en (d). Claramente, se consigue una mejora respecto a las anteriores técnicas, tanto para las zonas estáticas como para las dinámicas.

El sistema se alimenta de los cuadros de vídeo

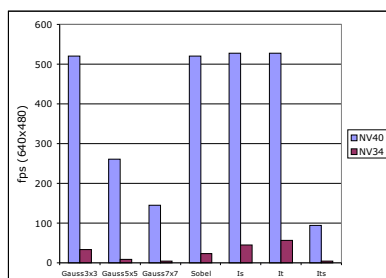


Figura 4: Rendimiento de las GPUs NV40 y NV34 para los diferentes experimentos realizados.

que llegan a 30 fps. Sin embargo, el procesamiento que se hace sobre ellos se realiza en torno a 500 fps para resoluciones espaciales de 640x480 (ver Figura 4). Por supuesto, el filtrado de vídeo es dependiente del tamaño de la máscara de convolución, pero estos números dan una estimación de la enorme potencia de cálculo de este hardware de consumo. Con el propósito de demostrar la rápida evolución de este tipo de hardware hemos realizado pruebas con una tarjeta Nvidia GeForce FX5200 (NV34), lanzada al mercado a finales de 2003. Esta GPU no era de alta gama dentro de su familia GeForce FX, pero en algunos casos puede ofrecer resultados muy aceptables.

En la Figura 4 se detallan los resultados obtenidos en los que se observa la enorme diferencia entre ambas tarjetas. La capacidad de procesamiento de la NV40 es mucho mayor que la de cualquier tarjeta de la familia anterior NV3X, fundamentalmente debido al mayor número de *pipelines* dedicados en la etapa del *fragment shader*, a la mayor integración de transistores y frecuencia de reloj. Por otra parte, las técnicas usadas en este trabajo, tales como acceso directo a fragmentos y bifurcación condicional en el *fragment shader*, son favorecidas en mayor medida en la NV40 que en las anteriores. Para el problema de desentrelazado adaptativo se obtienen 94 fps en la NV40 frente a los inaceptables 4 fps de la NV34 que, sin embargo, logra filtrar a 40 fps.

5. Conclusión

El procesamiento de vídeo en tiempo real es una tarea computacionalmente costosa. En este trabajo

se ha demostrado que las GPUs de bajo coste pueden ser explotadas para ejecutar procesos que poco tienen que ver con la visualización y el renderizado. En concreto, se han aplicado a problemas de filtrado y desentrelazado de vídeo en tiempo real. La idea clave que subyace es el aprovechamiento del alto poder computacional que adquiere este hardware para procesamientos basados en el modelo de computación de *streams*.

Referencias

- [1] Dockstader, S.L., Tekalp, M., *On the Tracking of Articulated and Occluded Video Object Motion, Real-Time Imaging*, 7: 415–432, 2001.
- [2] *GPGPU Website*, <http://www.gpgpu.org>
- [3] Haritaoglu, I., Harwood, D., Davis, L.S., *W4: Real-Time Surveillance of People and Their Activities, IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22: 809–830, 2000.
- [4] Harris, M. J., *Real-Time Cloud Simulation and Rendering*, Ph. D Thesis, Univ. of North Carolina at Chapel Hill, 2003.
- [5] Krueger, J., Westermann, R., *Acceleration Techniques for GPU-based Volume Rendering. In Proc. IEEE Visualization 2003*.
- [6] Larsen, E. S., McAllister, D., *Fast Matrix Multiplies using Graphics Hardware, In Proc. Supercomputing 2001*.
- [7] Purcell, T., *Ray Tracing on a Stream Processor*, Ph. D Thesis, Univ. of Stanford, 2004.
- [8] Thompson, C.J., Hahn, S., Oskin, M., *Using Modern Graphics Architectures for General-Purpose Computing: A Framework and Analysis, Int. Symposium on Microarchitecture (MICRO)*, 2002.
- [9] Venkatasubramanian, S., *The Graphics Card as a StreamComputer, Workshop on Management and Processing of Data Streams*, San Diego, California, USA, 2003.