Flujo óptico variacional en plataformas paralelas GPU

F. Naveros, T. Díaz, J. Ralli, E. Ros, J. Díaz

Resumen— **En este trabajo se evalúa la capacidad de una arquitectura GPU para calcular el flujo óptico de una secuencia de imágenes a una velocidad superior o igual a tiempo real y cómo afecta al rendimiento el tamaño de la imagen. Utilizamos la técnica *combined-local-global* (CLG), un modelo variacional que combina la creación de un flujo óptico denso con la aproximación de *Horn and Schunck* y la robustez frente al ruido del método de *Lucas-Kanade*. Para abordar este método, hay que resolver un sistema de $2N_xN_y$ ecuaciones de *Euler-Lagrange*, donde N_x y N_y representan el tamaño de la imagen. Utilizando un algoritmo *Multigrid* que utiliza el método iterativo *Successive Over-Relaxation* (SOR) con *Jacobi* para calcular el pre- y postrelajación y el valor del residual, cuya velocidad de convergencia no es excesivamente elevada, pero con una estructura óptima para ser paralelizado en una GPU, se consigue procesar el flujo óptico entre dos imágenes de la secuencia de Grove de 640 x 480 pixel en 8,79 ms, lo que supone una velocidad de 114 imágenes por segundo, muy superior a las 25 imágenes por segundo que generalmente se entiende por tiempo real.**

Palabras clave— **CUDA, GPU, flujo óptico, Multigrid, tiempo real.**

I. Introducción

Uno de los retos más importantes en la visión por computador es la extracción automática de la información del movimiento entre secuencias de imágenes de forma rápida y precisa. Sin tener información a priori de la escena a evaluar, ser capaz de calcular un patrón que describa el movimiento nos puede dar la información necesaria para calcular la trayectoria, velocidad o aceleración de un objeto o de la propia cámara si esta está en movimiento en relación a la escena observada.

Los métodos diferenciales son una de las técnicas más ampliamente utilizadas para la estimación del flujo óptico en secuencias de imágenes y una de las que obtiene los mejores rendimientos [1][8]. Estos métodos se basan en el cálculo de las derivadas temporales y espaciales de la imagen. Se pueden clasificar en dos grandes grupos:

- Métodos locales: tratan de optimizar alguna función de energía local. Son métodos muy robustos frente al ruido, pero con el inconveniente de que no generan un campo denso de descripción del movimiento. Algunos de estos métodos son *Lucas-Kanade* [11] o la aproximación del tensor de estructura de *Bigün* [2].

Dirección de los autores. E-mail: {fnaveros, tdiaz, jarnor, eros, jdiaz}@atc.ugr.es .

- Métodos globales: son métodos variacionales que tratan de minimizar una función global de energía. Estos sí generan un campo denso de descripción del movimiento, pero se sabe mediante técnicas experimentales que son más sensibles al ruido [1] [8]. Algunos de estos métodos son el de *Horn and Schunck* [10] o el de *Nagel and Enkelmann* [13].

El método utilizado en este trabajo es el CLG [3][15], una técnica variacional que combina las ventajas de los dos tipos de métodos descritos anteriormente: la aproximación variacional de *Horn and Schunck* [10] para la creación de un campo denso de descripción del movimiento y su alto rendimiento y la robustez frente al ruido del método local de *Lucas y Kanade* [11].

Se puede observar que los métodos empleados para llevar a cabo esta tarea tienen una convergencia razonablemente rápida al comienzo, pero rápidamente se ve degradada y a menudo es necesario realizar cientos de iteraciones para alcanzar el resultado deseado, lo que conlleva una alta carga conputacional que hace que su implementación en tiempo real sea compleja. Para imágenes de reducido tamaño es posible implementar estos algoritmos en ordenadores convencionales, sin embargo, cuando la resolución de las imágenes necesita ser mayor o la aplicación debe ser integrada como parte de otra, estas arquitecturas no tienen suficiente potencia y es necesaria la utilización de arquitecturas de alto rendimiento. Se pueden ver trabajos en esta línea en *cluster* de ordenadores [6], en el procesador *Cell Broadband Engine* de la *Playstation 3* [9] y también en procesadores gráficos o GPU [12].

Este trabajo se centra en las plataformas GPU como alternativa de computación de alto rendimento. Primeramente se desarrolló un modelo en C para ser ejecutado en una CPU y posteriormente se adaptó para ser ejecutado en una GPU, usando el lenguaje de programación CUDA (Compute Unified Device Architecture) del fabricante NVIDIA.

Este trabajo se organiza como sigue. En la Sección 2 se hace una breve descripción del método CLG y el sistema de ecuaciones lineales que se han de resolver. En la Sección 3 se presenta la implementación paralelizada del método *Multigrid* que se va a utilizar, indicando los criterios de diseño tenidos en cuenta para mejorar el rendimiento en una GPU. El algoritmo es evaluado en la Sección 4, analizando su rendimiento para distintas secuencias y tamaños de imagen, comparándolo con la versión serie en CPU. Se concluiye con un resumen en la Sección 5.

II. Flujo óptico con el método CLG

A. Descripción del método CLG

En [3] y [15] se describe el método *combined local-global* (CLG) para el cálculo del flujo óptico. Esta técnica combina las ventajas de la aproximación global de *Horn and Schunck* [10] y el método local de *Lucas y Kanade* [11]. Para describir este algoritmo, supongamos que $f(x, y, t)$ es una secuencia de imágenes en escala de grises, donde (x, y) describe la posición dentro de la imagen rectangular de tamaño Ω y t es el tiempo. El método CLG calcula el campo del flujo óptico $(u(x, y), v(x, y))^\top$, en un instante de tiempo t, mediante la minimización del funcional de energía.

$$E(u, v) = \int_\Omega (\omega^\top J_\rho(\nabla_3 f)\omega + \alpha(|\nabla u|^2 + |\nabla v|^2))dxdy \tag{1}$$

En la ecuación (1), el vector de campo $w(x, y) = (u(x, y), v(x, y), 1)^\top$ describe el desplazamiento de cada pixel entre imágenes de la secuencia, ∇u es el gradiente espacial $(u_x, u_y)^\top$, y $\nabla_3 f$ describe el gradiente espaciotemporal $(f_x, f_y, f_t)^\top$. La matriz $J_\rho(\nabla_3 f)$ es el tensor estructural dado por $K_\rho * (\nabla_3 f \nabla_3 f^\top)$, donde * describe una operación de convolución y K_ρ es una *gaussiana* con desviación estándar ρ. El peso $\alpha > 0$ define como de suave ha de ser la solución.

Analizando la ecuación (1), se puede determinar que si $\rho \to 0$, la aproximación CLG se reduce al método de *Horn and Schunck*, mientras que si $\alpha \to 0$, entonces se transforma en el algoritmo de *Lucas-Kanade*.

Para poder calcular el flujo óptico de la secuencia, debemos minimizar la expresión $E(u, v)$. Esta tarea se puede llevar a cabo resolviendo la ecuación de *Euler-Lagrange* [7].

$$\alpha\triangle u - (J_{11}(\nabla_3 f)u + J_{12}(\nabla_3 f)v + J_{13}(\nabla_3 f)) = 0 \tag{2}$$

$$\alpha\triangle u - (J_{12}(\nabla_3 f)u + J_{22}(\nabla_3 f)v + J_{23}(\nabla_3 f)) = 0 \tag{3}$$

Donde $\triangle$ representa el Laplaciano y se aplica la condición de que la derivada normal se anula en la frontera. En la expresión (4), n representa el vector normal perpendicular al borde de la imagen.

$$0 = \mathtt{n}^\top \nabla u, \qquad 0 = \mathtt{n}^\top \nabla v \tag{4}$$

B. Discretización

Una vez definidas las expresiones (2) y (3) que reflejan las ecuaciones de *Euler-Lagrange*, se puede utilizar una técnica de discretización diferencial finita para obtener un sistema lineal de ecuaciones que ha de ser resuelto para la obtención definitiva del campo denso del flujo óptico.

C. Las ecuaciones discretas de Euler-Lagrange

Se quiere calcular el valor desconocido de las funciones $u(x, y, t)$ y $v(x, y, t)$, para ello, se discretiza su valor en una malla rectangular de tamaño N_x x N_y con una separación entre pixel h_x x h_y y se utiliza la nomeclatura $u_{i,j}$ para expresar la aproximación de $u(x, y, t)$ en una posición (i, j) con $1 \le i \le N_x$ y $1 \le j \le N_y$. La convolución con la *gaussiana* K_ρ se lleva a cabo mediante una convolución discretizada con una *gaussiana* truncada y renormalizada, donde el truncamiento se realiza a tres veces la desviación estándar. Las derivadas espaciales de la secuencia $f(x, y, t)$ se han aproximado utilizando un esquema de diferencias finitas entre imágenes, mientras que las derivadas temporales se aproximan utilizando una plantilla de dos puntos. También usaremos la notación $[J_{nm}]_{i,j}$ para representar la componente (n, m) del tensor de estructura $J_\rho(\nabla_3 f)$ en un pixel (i, j). Por último, usamos la expresión $\mathcal{N}_l(i, j)$ para denotar los vecinos de un pixel (i, j) en la dirección del eje l. Entonces, una aproximación diferecial finita de las ecuaciones de *Euler-Lagrange* (2)(3) con las condiciones de nulidad en el contorno de la imagen se pueden expresar así:

$$\begin{aligned} 0 = {} & [J_{11}]_{i,j}u_{i,j} + [J_{12}]_{i,j}v_{i,j} + [J_{13}]_{i,j} - \\ & -\alpha \sum_{l\in\{x,y\}} \sum_{(m,n)\in\mathcal{N}_l(i,j)} \frac{u_{m,n}-u_{i,j}}{h_l^2} \end{aligned} \tag{5}$$

$$\begin{aligned} 0 = {} & [J_{12}]_{i,j}u_{i,j} + [J_{22}]_{i,j}v_{i,j} + [J_{23}]_{i,j} - \\ & -\alpha \sum_{l\in\{x,y\}} \sum_{(m,n)\in\mathcal{N}_l(i,j)} \frac{v_{m,n}-v_{i,j}}{h_l^2} \end{aligned} \tag{6}$$

Donde (i, j) toman los valores $i = 1, ..., N_x$ y $j = 1, ..., N_y$ y forman un sistema de $2N_xN_y$ ecuaciones con respecto a $u_{i,j}$ y $v_{i,j}$.

D. Implementación

Para resolver este sistema extenso de ecuaciones lineales, debemos tener en cuenta el tipo de arquitectura que va a ser empleada, en nuestro caso, una GPU. La gran ventaja de este tipo de plataformas es la capacidad de ejecutar miles de hebras de forma totalmente paralela mediante un conjunto de operaciones SIMD (Single Instruction, Multiple Data). Seleccionando el algoritmo iterativo adecuado, podemos eliminar las interdependencias entre ecuaciones y de esta forma calcular el siguiente valor de la iteracion de u y v para todas las posiciones de forma simultánea.

Teniendo en cuenta esta información, se propone un algoritmo *multigrid* que utiliza un método *Successive Over-Relaxation* (SOR) con *Jacobi* como algoritmo iterativo (aunque la convergencia es un poco más lenta que para el algoritmo SOR basado en

Gauss-Seidel y se necesitan más iteraciones, se elimina la interdependencia entre ejecuciones y se pueden calcular todos los valores de u y v de una misma iteración en paralelo).

III. Algoritmo multigrid

Los algoritmos iterativos tradicionales pueden eliminar las componentes de alta frecuencia del error en pocas iteraciones, pero necesitan muchas iteraciones para eliminar las componentes de baja frecuencia, lo que reduce considerablemente su velocidad de convergencia. El método *multigrid* soluciona este problema mediante la creación de una sofisticada jerarquía fino-a-grueso de sistemas de ecuaciones para la reducción del error. Las componentes de baja frecuencia en un sistema fino reaparecen como componentes de alta frecuencia en los siguientes sistemas gruesos, donde pueden ser eliminadas rápidamente.

Este algoritmo pertenece a una de las clases más poderosas para la resolución númerica de sistemas de ecuaciones lineales y es ampliamente utilizado en algoritmos de visión por computador. En [4] y [5] se ha demostrado que estas técnicas permiten el cálculo del flujo óptico en tiempo real para imágenes pequeñas en ordenadores convencionales. El algoritmo implementado en la GPU será una versión paralelizada del descrito en [4], pero utilizando *Jacobi* en vez de *Gauss-Seidel* en el método SOR. También se implementará una versión serie en CPU para así poder comparar los resultados.

A. Definición del método

El funcionamiento básico de este método se basa en la ejecución encadenada de ciclos V (existe la alternativa de trabajar con ciclos W, que pueden obtener mejores resultados, pero para ello se necesita realizar muchas más operaciones con los sistemas más gruesos y como se verá posteriormente, el rendimiento de una GPU es menor cuanto menor es el sistema de ecuaciones, por lo tanto, se descarta esta opción). Un ciclo V se puede dividir a su vez en dos etapas.

- Etapa de bajada: en un sistema fino de ecuaciones (escalas más grandes de la imagen), se realizan I iteraciones del algoritmo SOR para obtener una aproximación de la solución. Con esta aproximación, se calcula un residual del error y se pasa al siguiente sistema de ecuaciones más grueso (escala inferior de la imagen) mediante submuestreo. Con este valor del residual, se ejecutan de nuevo I iteraciones para obtener una nueva aproximación de la solución en el nuevo sistema de ecuaciones más grueso (los errores de baja frecuencia del sistema fino se convierten en errores de alta frecuencia en el sistema grueso al hacer el submuestreo y por tanto se pueden eliminar fácilmente). Se repite este proceso, calculando el residual y pasándoselo al siguiente sistema grueso, hasta llegar al último sistema de ecuaciones (el de menor tamaño).
- Etapa de subida: una vez que estamos en la última escala, se pasa la aproximación al sistema fino más próximo mediante interpolación. Con este valor, se corrige la solución del sistema fino y se realizan I nuevas iteraciones para obtener la nueva aproximación. Se repite este proceso hasta llegar a la escala inicial.

Con estas dos simples etapas se recorre un ciclo V completo. Se realizan V ciclos V por cada escala y cuando se terminan, si la escala en la que se está no es la mayor, se pasa la solución al siguiente sistema fino de ecuaciones y se vuelven a realizar los V ciclos V para esa escala, así hasta llegar a la superior.

A continuación se muestra un ejemplo de la concatenación de ciclos V con cuatro escalas de la imagen y un ciclo V por cada escala.

Fig. 1. Un ciclo V por cada escala.

B. Estructuración de las hebras de procesamiento

Como se ha mencionado anteriormente, la gran ventaja de una GPU reside en su gran número de unidades de procesamiento divididas en una jerarquía de multiprocesadores multihebra. Las funciones a ejecutar se dividen en bloques de hebras que son asignados mediante un planificador automático a uno de los multiprocesadores disponibles. Estos bloques tienen asociados un conjunto de hebras que se planifican de forma automática en cada procesador multihebra en bloques de un *warp* (32 hebras). Si todas las hebras de un mismo *warp* ejecutan el mismo código, se planifican en paralelo. Por el contrario, si dentro del código hay sentencias *if - else* que bifurcan el código a ejecutar por cada hebra, son planificadas y ejecutadas una a una, con lo que disminuye considerablemente el rendimiento. Este tipo de situaciones se presentan muy a menudo y hay que tratar de evitarlas en lo posible. Por ejemplo, en el cálculo de las derivadas y en las convoluciones con la *gaussiana*, para calcular un valor de una posición es necesario utilizar los vecinos. Si la posición a calcular se encuentra en un borde y no tiene vecinos, una posible alternativa es utilizar únicamente la información disponible mediante una serie de sentencias *if - else* que verifiquen cuántos datos tenemos disponibles (totalmente desaconsejable en una GPU, pues las operaciones serían serializadas). La opción más recomendable es añadir información redundante en los bordes de la imagen, copiando el valor más cercano y de esta forma eliminar las sentencias *if - else*

(el redundar la información de los bordes añade un poco de carga computacional pero se ve compensada al poder eliminar las sentencias *if - else*).

Igualmente, se necesita un método iterativo para calcular la pre- y postrelajación y el valor residual. Ya se descartó anteriormente el método de *Gauss-Seidel* para llevar a cabo esta tarea, debido a la interdependencia en las ejecuciones (se podría utilizar la técnica *Red-Black* para paralelizar las ejecuciones, pero esto afectaría seriamente a la jerarquía de memorias de la GPU, impidiendo agrupar los accesos a memoria global y empeorando la gestión de la caché). Por el contrario, utilizamos el método de *Jacobi*, cuya convergencia es un poco más lenta, es decir, requiere más iteraciones, pero con una estructura que elimina las interdependencias de datos y permite que se pueda planificar paralelamente una hebra por cada posición a calcular en cada iteración.

C. Gestión de la memoria

La forma en la que una GPU accede a los datos en memoria depende de cómo se haga la lectura. Si las hebras de un *warp* acceden a posiciones consecutivas de la memoria, se unifican todos los accesos y se realiza una única transferencia de datos, con lo que se consigue optimizar el ancho de banda. Teniendo en cuenta este efecto, se ha intentado en lo posible dividir las tareas a realizar en bloques y que cada bloque procese una fila de la imagen (la forma en que se almacena una imagen en memoria es por filas, por tanto, si las hebras acceden a datos de una misma fila, todos los accesos se unificarán en uno solo).

IV. Resultados

A. Pruebas realizadas

En este capítulo se evalúa el rendimiento del modelo desarrollado, tanto la versión serie para la CPU como la paralelizada y optimizada para la GPU. Se utilizarán dos imágenes de dos secuencias distintas, la de *Grove* y la de *Yosemite* sin nubes para medir la precisión alcanzada (error angular) en función del tiempo de ejecución que deseemos utilizar (número de escalas, ciclos V e iteraciones empleadas). Por último se verá cómo afecta el tamaño de la imagen y el valor de los parámetros al rendimiento total del sistema, tanto en tiempo de ejecución como en la mejora conseguida.

B. Caracteristicas del equipo de pruebas

Las mediciones se han llevado a cabo en un equipo con las siguientes características técnicas:

- CPU Intel core i7 920 a 2,66GHz, con placa base Asus P6T SE y 12 GB de memoria RAM DDR3 a 1066 MHz.
- GPU NVIDIA GTX 470, basada en una arquitectura CUDA con capacidad computacional 2.0. Contiene 448 núcleos CUDA y 1280 MB de memoria GDDR5. La velocidad de los relojes es de 1215 MHz para el procesador central, 607 MHz para el procesador gráfico y 1674 MHz para la memoria, lo que supone un ancho de banda máximo de 133,9 GB/s.

C. Secuencia de Yosemite

Se evalúan dos imágenes de la secuencia de *Yosemite*[20] modificada sin nubes que tiene una resolución de 316 x 252 pixel. Se realizan una serie de ejecuciones variando el número de escalas, ciclos V e iteraciones utilizadas y buscamos los valores óptimos de los parametros α y ω que dan el menor error angular posible, promediando el tiempo obtenido de diez ejecuciones. Con estos datos, generamos un gráfico de dispersión del error angular en función del tiempo de ejecución. Una vez que tenemos todos los resultados, seleccionamos los parámetros, TABLA I, con los que obtenemos los mejores resultados, TABLA II y Fig. 2.

TABLA I
Parámetros de ejecución.

Escalas	Ciclos V	Iter.	Omega	Alpha
3	3	1	0,75	4,51E-5
3	3	3	0,8	1,78E-4
3	5	3	0,8	3,59E-4
3	7	3	0,85	4,32E-4
3	9	3	0,85	5,93E-4
3	11	3	0,75	6,07E-4
3	13	3	0,8	7,50E-4
3	13	5	0,75	1,18E-3
3	15	7	0,5	1,28E-3
3	15	9	0,5	1,26E-3

TABLA II
Error angular (°) y tiempo de ejecución (*ms*) en la secuencia de Yosemite sin nubes para CPU y GPU.

Error	Tiempo CPU	Tiempo GPU
5,18	276	7,23
4,23	389	10,27
3,36	640	17,07
2,88	889	23,57
2,59	1142	30,24
2,46	1396	36,86
2,30	1649	43,52
2,18	2130	59,95
2,12	3005	81,82
2,08	3563	97,18

Como puede verse, cuanto mayor es el tiempo de ejecución (mayor número de escalas, ciclos V o iteraciones usadas), mejor es el resultado con respecto al error conseguido, pero llega un momento en el

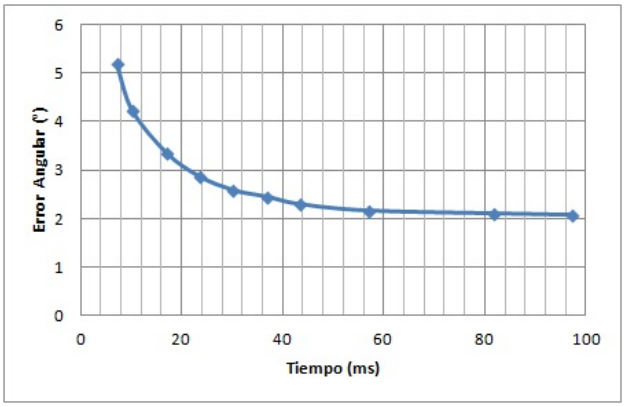

Fig. 2. Error angular vs Tiempo de ejecución en GPU.

que aumentar estos factores no mejora el resultado, puesto que se llega al límite de funcionamiento del algoritmo.

La mejora obtenida en la GPU respecto de la CPU está en torno a un factor x37 y se consigue una velocidad máxima de 138 imágenes por segundo y un error mínimo de 1,97°para un tiempo de ejecución considerablemente grande. El mínimo error conseguido para esta secuencia en los trabajos publicados hasta el momento es de 1,25°para el algoritmo *StereoFlow* y nuestro algoritmo se encuentra entre los 10 mejores para el estudio comparativo del estado del arte realizado en [14].

D. Secuencia de Grove

El procedimiento seguido en este apartado es el mismo que el del apartado anterior, pero usando dos imágenes de la secuencia de *Grove*[20] cuya resolución es 640 x 480 pixel, casi 4 veces más grande. Los parámetros utilizados se pueden ver en la TABLA III, con cuyos valores se obtienen los resultados mostrados en la TABLA IV y en la Fig. 3.

TABLA III
Parámetros de ejecución.

Escalas	Ciclos V	Iter.	Omega	Alpha
5	1	1	0,35	5,73E-6
5	2	1	0,3	9,28E-6
5	3	1	0,2	9,39E-6
5	3	2	0,5	5,61E-5
5	4	2	0,6	8,30E-5
5	4	3	0,45	8,62E-5
5	4	4	0,35	8,86E-5
3	15	3	0,5	5,62E-5
3	15	5	0,35	5,83E-5
4	15	7	0,2	9,25E-5

La tendencia observada para esta secuencia es la misma que para la de *Yosemite*, produciéndose una zona plana en el error si se aumenta mucho el tiempo de ejecución.

TABLA IV
Error angular (°) y tiempo de ejecución (ms) en la secuencia de Grove sin nubes para CPU y GPU.

Error	Tiempo CPU	Tiempo GPU
4,90	390	8,78
4,50	749	15,93
4,36	1123	23,02
4,27	1341	28,82
4,18	1779	37,95
4,16	2091	45,26
4,15	2418	52,10
3,97	5880	127,83
3,90	7852	170,70
3,76	11249	244,54

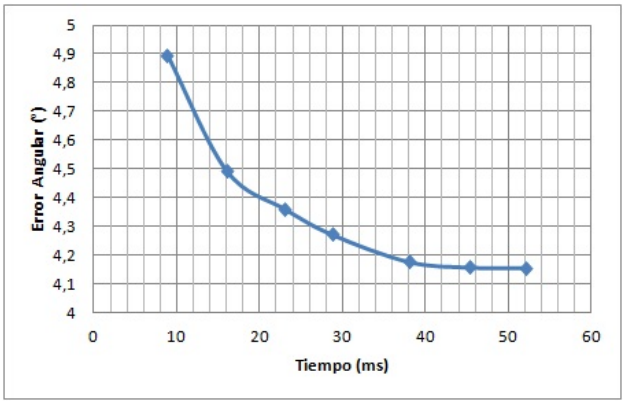

Fig. 3. Error angular vs Tiempo de ejecución en GPU.

En este caso, el tamaño de la imagen es considerablemente mayor, sin embargo, este algoritmo sigue siendo capaz de procesarlas a una velocidad máxima de 114 imágenes por segundo, con un factor de mejora en torno a x46 respecto de la CPU y consigue un error mínimo de 3,76°para un tiempo de ejecución considerablemente grande. El mínimo error conseguido para esta secuencia en los trabajos publicados hasta el momento es de 2,35°para el algoritmo *Layers++* y nuestro algoritmo se encuentra entre los 35 mejores para el estudio comparativo del estado del arte realizado en [14].

E. Dependencia con el tamaño de la imagen

Ya se ha demostrado la validez del algoritmo para dos secuencias de imágenes ampliamente utilizadas para la evaluación de modelos de flujo óptico, consiguiendo tiempos de ejecución muy inferiores a lo que requiere tiempo real para errores angulares relativamente pequeños.

En este apartado, vamos a caracterizar el comportamiento del algoritmo en función del tamaño de la imagen a procesar y del número de ciclos V e iteraciones que deseemos utilizar, mostrando en sendas gráficas tanto el tiempo de ejecución en la GPU

(Fig.4), como el factor de ganancia obtenido con respecto a la CPU (Fig.5).

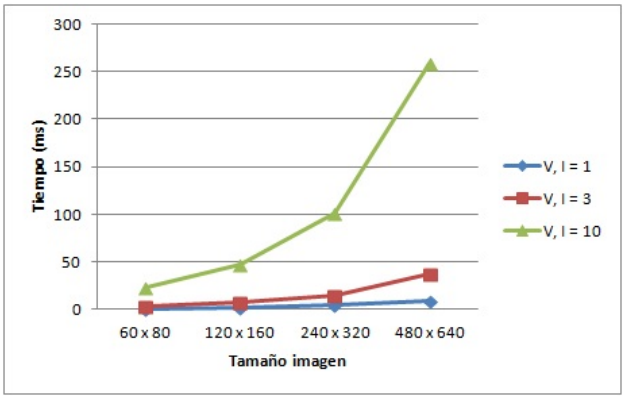

Fig. 4. Tiempo de ejecución en GPU.

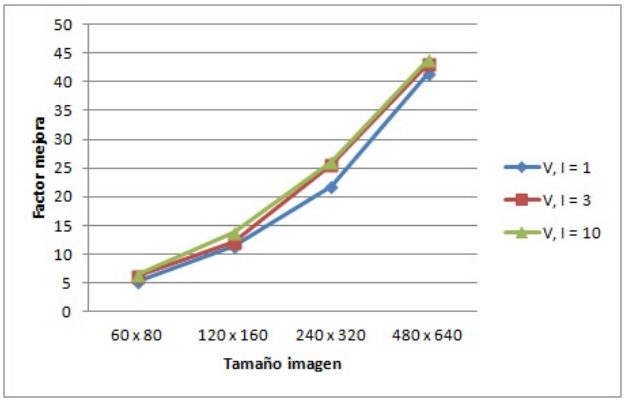

Fig. 5. Factor de mejora GPU vs CPU.

Como puede verse en Fig.4, los tiempos de ejecución crecen conforme aumenta el tamaño de la imagen, como era de esperar. También se puede ver que cuantos más ciclos V e iteraciones deseemos realizar, mayor será el retardo.

Con respecto al factor de ganancia, Fig.5, se puede ver cómo este tiene un comportamiento casi lineal con respecto al número de filas de la imagen, pues la mayoría de los métodos utilizan un bloque de hebras por cada fila de la imagen a procesar, por tanto, el doble de filas corresponden aproximadamente al doble de bloques a procesar y al doble de tiempo en la GPU mientras que el doble de filas y columnas suponen una cuadruplicación del tiempo en la CPU.

Mientras que la CPU sólo consigue tiempo real (es decir, capacidad de procesamiento a más de 25 imágenes por segundo) para los tamaños de imagen más pequeños, la GPU consigue resultados muy superiores a tiempo real para todos los tamaños de imagen analizados.

Ha quedado demostrado que el algoritmo es más eficiente con respecto a la versión CPU conforme aumenta el tamaño de la imagen a procesar, pues la carga de los procesadores es cada vez mayor y están funcionando durante más tiempo, mientras que para imágenes pequeñas, el factor de ganancia se ve seriamente degradado.

V. Conclusiones

En este trabajo se ha presentado un algoritmo de alto rendimiento para el cálculo del flujo óptico en GPU. Combinando técnicas de procesado en paralelo, reduciendo al mínimo las sentencias *if - else* (de esta forma se planifican todas las hebras simultáneamente), utilizando un método que minimiza la dependencia de datos y planificando un acceso coalescente a los datos para la minimización del número de accesos a memoria global se consigue un factor de mejora, con respecto al modelo serie, que oscila entre 5 y 45, dependiendo del tamaño de la imagen a procesar. En términos de imágenes por segundo, se consigue calcular el flujo óptico de una secuencia de imágenes de 640 x 480 a 114 imágenes por segundo.

También se ha visto que aumentar mucho el número de escalas, ciclos V e iteraciones empleadas aumenta considerablemente el tiempo de ejecución mientras que el error obtenido apenas mejora, pues se llega al límite de precisión del algoritmo. Es por tanto fundamental fijar el valor de estos parámetros al valor adecuado para conseguir las prestaciones de tiempo de ejecución y error cometido deseados.

Para modelos de GPU de mayores prestaciones se pueden conseguir resultados aún mejores en términos de tiempo de ejecución, consiguiendo factores de mejora aún mayores y con imágenes más grandes.

Agradecimientos

Este trabajo ha sido parcialmente financiado con el proyecto Europeo TOMSY (FP7-270436) [19], el proyecto Nacional ARC-VISION (TEC2010-15396) y los proyectos Autonómicos MULTIVISION (TIC-3873) [18] e ITREBA (TIC-5060) [17].

Referencias

[1] J. L. Barron, D. J. Fleet, and S. S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1):43:77, Feb. 1994.

[2] J. Bigüi, G. H: Granlund, and J. Wiklund. Multidimensional orientation estimation with applications to texture analysis and optical flow. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 13(8):775-790, Aug. 1991.

[3] A. Bruhn, J. Weickert, and C. Schnörr. Combining the advantages of local and global optic flow methods. In L. Van Gool, editor, *Pattern Recognition. Lecture Notes in Computer Science:* 2449: 454-462. Springer, Berlin, 2002.

[4] A. Bruhn, J. Weickert, C. Feddern, T. Kohlberger, and C. Schnörr. Variational optical flow computation in real-time. *IEEE Transactions on Image Processing*, 14(5):608-615, May 2005

[5] A. Bruhn, J. Weickert, T. Kohlberger, and C. Schnörr. A multigrid platform for real-time motion computation with discontinurity-preserving variational methods. *International Journal of Computer Vision*, 70(3):257-277, Dec. 2006

[6] M. El Kalmoun, H. Köstler, and U. Rüde. 3D optical flow computation using a parallel variational multigrid scheme

with application to cardiac C-arm CT motion. *Image and Vision Computing*, 25(9):1482-1494, 2007.
[7] L. E. Elsgolc. *Calculus of Variations.* Pergamon, Oxford, 1961.
[8] B. Galvin, B. McCane, K. Novins, D. Mason, and S. Mills. Recovering motion fields: an analysis of eight optical flow algorithms. In *Proc. 1998 British Machine Vision Conference*, Southampton, England, Sept. 1998.
[9] P. Gwosdek, A. Bruhn and J. Weickert. High Performance Parallel Optical Flow Algorithms on the Sony Playstation 3. In *O. Deussen, D- Keim, D. Saupe (eds.) VMV 2008: Proceeding of Vision, Modeling, and Visualization.* 253-262. AKA, Konstanz, Germany, 2008.
[10] B. Horn and B. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185-203, 1981.
[11] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proc. Seventh International Joint Conference on Artificial Intelligence*, 674-679, Vancouver, Canada, 1981
[12] Y. Mitsukami and K. Tadamura. Optical flow computation on Compute Unified Device Architecture. In *Proc. 14th International Conference on Image Analysis and Processing*, 179-184. IEEE Computer Society Press, 2007.
[13] H.-H. Nagel and W. Enkelmann. An investigation of smoothness constraints for the estimation of displacemente vector fields from image sequences. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 8:565-593, 1986.
[14] D. M. Young. *Iterative Solution of Lagrange Linear System.* Academic Press, New York, 1971.
[15] J. Weickert, A. Bruhn, and C.Schnörr. Lucas/Kanade meets Horn/Schunck: Combining local and global optic flow methods. Technical Report 82, Dept. of Mathematics, Saarland University, Saarbrücken, Germany, Apr. 2003.
[16] Optical flow evaluation results *[En línea]* http://vision.middlebury.edu/flow/eval/results/results-a1.php
[17] Proyecto Autonómico ITREBA *[En línea]* http://atc.ugr.es/itreba/
[18] Proyecto Autonómico MULTI-VISION *[En línea]* http://atc.ugr.es/mvision/
[19] Proyecto Europeo TOMSY *[En línea]* http://www.cas.kth.se/tomsy/
[20] Secuencias de imágenes *[En línea]* http://vision.middlebury.edu/flow/data/