Aceleración de procedimientos de mejora local en hardware gráfico para seguimiento visual

Raúl Cabido[1], Antonio S. Montemayor[1], Juan José Pantrigo[1], Bryson Payne[2]

Resumen—**Los sistemas de seguimiento tienen mucha relevancia en Visión Computacional, y poseen multitud de aplicaciones en vigilancia, interacción persona-ordenador, etc. Las tarjetas gráficas (GPUs) han experimentado una evolución extraordinaria tanto en rendimiento computacional como en programabilidad, permitiendo su aplicación a cómputo general. En este trabajo, se propone un algoritmo de seguimiento de objetos en tiempo real, basado en la hibridación de un filtro de partículas (PF) y una búsqueda local multiescala (MSLS). El algoritmo ha sido implementado y probado sobre arquitecturas CPU y GPU. El sistema desarrollado, proporciona resultados precisos en lo que respecta a la calidad del seguimiento de varios objetos. Además, se ejecuta en tiempo real a 70 fotogramas por segundo en la GPU, un 1100 % más rápido que la versión CPU del algoritmo.**

Palabras clave— **Videovigilancia, Seguimiento en tiempo real, Aceleración de procesos de mejora local, Filtros de partículas, Procedimientos multiescala.**

I. Introduction

El seguimiento eficiente de objetos es una tarea requerida por una gran cantidad de aplicaciones de Visión Computacional, como Videovigilancia o Robótica. El proceso de seguimiento está relacionado con la estimación de variables del espacio de estados del sistema y su predicción en el tiempo.

Como otros problemas bien establecidos, el seguimiento visual se puede categorizar de diferentes formas. Atendiendo de las características del objeto, el problema puede consistir en seguimiento de siluetas, de múltiples objetos, objetos articulados, deformables, o combinaciones de éstos [18]. Teniendo en cuenta las características de los dispositivos de captura, el problema puede ser 2-D, 2 1/2-D o 3-D. Además, en un enfoque *top-down*, se evalúa un modelo paramétrico sobre la observación. Este enfoque necesita modelos descriptivos del objeto, que pueden ser difíciles de crear, dependiendo del problema considerado. Las estrategias *bottom-up* son menos costosas computacionalmente, y tratan de encontrar los objetos en la escena sin un modelo explícito preestablecido [7]. Se basan en la extracción de características de las imágenes, tales como estimación del movimiento o segmentación. Este tipo de estrategias les hacen más generales y aplicables a diferentes escenarios.

Una de los enfoques más populares al problema del seguimiento en los últimos años es el algoritmo probabilista basado en el filtro de partículas (PF). Este tipo de algoritmos fue propuesto por Gordon et al. en 1993 para resolver problemas de estimación secuencial en sistemas no lineales y no Gausianos [8]. PF se basa en representaciones discretas (llamadas partículas) de la función de densidad de probabilidad (*pdf*) que describe la evolución del sistema a lo largo del tiempo [3]. PF combina estrategias de adaptación y predicción. Por otro lado, la independencia de las evaluaciones sobre el conjunto de partículas les convierte en algoritmos muy adecuados para plataformas paralelas.

En este artículo, proponemos un enfoque basado en características para el seguimiento 2D de múltiples objetos utilizando filtros de partículas. Para hacer frente a la carga computacional del método, proponemos dos estrategias: una optimización algorítmica y una implementación *hardware*. Para la primera, planteamos la hibridación de un PF con métodos de búsqueda local dirigidos por un procedimiento multiescala (MSLS). El algoritmo resultante (MSLSPF) lleva a cabo una búsqueda local sobre la mejor solución encontrada por PF. Para la segunda, utilizamos un procesador gráfico. Por lo tanto, se lleva a cabo un enfoque paralelo tanto de PF como de las etapas de optimización subsecuentes. Los algoritmos propuestos han sido adaptados cuidadosamente a la GPU, de modo que se minimice la transferencia de datos entre la CPU y la GPU.

II. Trabajo previo

A. Filtro de Partículas

Los filtros de partículas (en adelante PF) estiman distribuciones teóricas en el espacio de estados, aproximándolas a través de medidas aleatorias (llamadas partículas) [3]. El modelo de espacio de estados está formado por dos procesos: (i) proceso de observación $p(Z_{1:t}|X_t)$ donde X_t es el estado del sistema y Z_t denotan las observaciones en el instante t, y (ii) proceso de transición $p(X_t|X_{t-1})$. Asumiendo que se dispone de observaciones $\{Z_0, Z_1, \ldots, Z_t\}$ a lo largo del tiempo, el objetivo es estimar el estado del sistema en cada instante. La *pdf* posterior se estima en dos etapas:

(a) Evaluación: la *pdf* posterior $p(X_t|Z_{1:t})$ se calcula utilizando el vector de observación $Z_{1:t}$:

$$p(X_t|Z_{1:t}) = \frac{p(Z_t|X_t)p(X_t|Z_{1:t-1})}{p(Z_t|Z_{1:t-1})} \quad (1)$$

(b) Predicción: la *pdf* posterior $p(X_t|Z_{1:t-1})$ se propaga en el tiempo utilizando la ecuación de Chapman-Kolmogorov:

$$p(X_t|Z_{1:t-1}) = \int p(X_t|X_{t-1})p(X_{t-1}|Z_{1:t-1})dX_{t-1} \quad (2)$$

[1]Departamento de Ciencias de la Computación. Universidad Rey Juan Carlos. E-mail:{raul.cabido, antonio.sanz, juanjose.pantrigo}@urjc.es
[2]Mathematics and Computer Science Department. North Georgia College and State University. Email: bpayne@ngcsu.edu

Usualmente se utiliza un modelo predefinido para obtener un conjunto de partículas actualizado en cada instante.

Los algoritmos PF se encargan de dirigir la evolución de un conjunto de partículas. Las partículas en PF son dirigidas por el modelo del sistema y se multiplican o eliminan de acuerdo a su peso o *fitness*, determinado por la *pdf* [3]. En problemas de seguimiento visual, esta *pdf* representa la probabilidad de que el objeto esté en una determinada posición y/u orientación en un instante determinado.

B. Filtros de partículas en seguimiento visual

La primera adaptación del esquema PF a un problema de seguimiento visual se debe a Isard and Blake [11] y data de mediados de los '90. Posteriormente, propusieron el algoritmo CONDENSATION (*CONditional DENsity propagaTION*) [12]. Desde entonces, los filtros de partículas han sido ampliamente aplicados en este ámbito. Existen dos categorías de PF, atendiendo al modelo de medida: basados en modelos y basados en características. En este trabajo nos centramos en los segundos. Estos, son menos costosos computacionalmente y, con cierta información acerca de la geometría del objeto, pueden resolver oclusiones [13]. Sin embargo, tienen el inconveniente de que son más sensibles al ruido y las medidas inexactas.

Para aplicar un PF a un problema de seguimiento visual, se deben tener en cuenta diferentes modelos y definiciones:

- **Espacio de estados**: Define el número de variables consideradas en el problema de seguimiento. Las soluciones almacenan el conjunto de variables necesarias para describir el estado del sistema y el peso asociado con esa configuración. Por ejemplo, en el seguimiento de M objetos en 2D, la estructura de solución es un vector $\mathbf{s}^t=[(x_1^t,y_1^t),\ldots,(x_M^t,y_M^t)]$, en el que cada par de variables (x_i^t,y_i^t) representa la posición del objeto i en el sistema de referencia de la imagen. Adicionalmente, se pueden considerar otras variables, tales como orientación y escala.
- **Modelo Geométrico**: En algunas aplicaciones es útil definir un modelo geométrico que describa la forma del objeto. El objetivo de éste modelo es relacionar las soluciones en el espacio de estados multidimensional (o espacio de soluciones) con las caraceríticas extraídas de la imagen 2D. Por ejemplo, en el seguimiento de múltiples objetos en 2D, se suelen utilizar ventanas de interés (*bounding boxes*).
- **Modelo de Observación**: Especifica las caracterñisticas de la imagen que pueden extraerse para proporcionar una medida de los objetos de interés. Por ejemplo, se suelen utilizar técnicas de segmentación como sustracción de fondo, selección de puntos de interés, detección de bordes, detección de piel, etc.
- **Función de ponderación**: Establece una relación entre las características detectadas por el modelo de observación y el peso asignado a al solución. Por ejemplo, en el seguimiento de múltiples objetos en 2D, se suele utilizar la suma de los píxeles correspondientes a cada objeto que pertenecen a cada ventana de interés.
- **Modelo del Sistema**: Describe la regla de transición temporal del estado del sistema entre dos fotogramas consecutivos [19]. En otras palabras, este modelo determina el modo en el que las partículas evolucionan en el tiempo.

III. Etapa de mejora local

El algoritmo propuesto por Isard y Blake [12] no es efectivo en aquellos problemas que presentan una alta dimensionalidad. A esta categoría pertenecen problemas como el seguimiento de múltiples objetos y objetos articulados. En el algoritmo de condensación, el número de partículas necesarias crece con el tamaño del espacio de estados, como se demuestra en [14]. Para abordar este dificultad, se han propuesto distintas optimizaciones del algoritmo de filtro de partículas. Muchas de ellas utilizan estrategias para mejorar el rendimiento del algoritmo convencional. En esta sección se propone la aplicación de una búsqueda local dirigida por una estrategia multiescala.

A. Métodos de Búsqueda Local

Los métodos de búsqueda local (en adelante LS) son procedimientos de optimización que parten de una solución inicial y la mejoran progresivamente. En cada paso, LS realiza un movimiento de desplazamiento desde la solución inicial para obtener una nueva solución. Si la solución obtenida mejora el valor de la función objetivo, el movimiento se acepta y el se repite el procedimiento. El método termina cuando no hay mejores soluciones accesibles [1]. Hemos implementado dos métodos de búsqueda local diferentes, denominados LS1 y LS2. A continuación se describe su funcionamiento:

- **Búsqueda local aleatoria (LS1)**: dada una solución inicial s se establece una vecindad $N(s)$ de s caracterizada por una región del espacio de estados cercana a s (ver la figura 1.a). A continuación se generan y evalúan un conjunto de soluciones aleatorias en $N(s)$. Si la mejor solución encontrada, $s_{best} \in N(s)$, es mejor que s, esta solución se reemplaza por s_{best} y se repite el procedimiento. El método termina cuando s_{best} tiene un peso menor que s. La mejor solución encontrada es el valor devuelto por el método.
- **Búsqueda local sistemática (LS2)**: este método es similar al anterior, pero realiza un procedimiento sistemático para obtener la solución en una vecindad dada $N(s)$ (ver figura 1.a). Dada una solución inicial s, se genera un conjunto de soluciones realizando movimientos predefinidos en s. Si alguna de estas soluciones tiene un peso mejor que s, entonces se reemplaza la solución inicial y se repite el proceso. El método termina cuando no se encuentran en la vecindad mejores soluciones que la de partida.

Hemos adaptado estas dos estrategias a problemas de seguimiento visual. En la Fig. 1.b se muestra un ejemplo en el que queremos encontrar la posición de una ventana de interés que localice el objeto (círculo blanco). Sea una solución inicial $s_i = [x_i, y_i]$, donde x_i e y_i son las posicio-

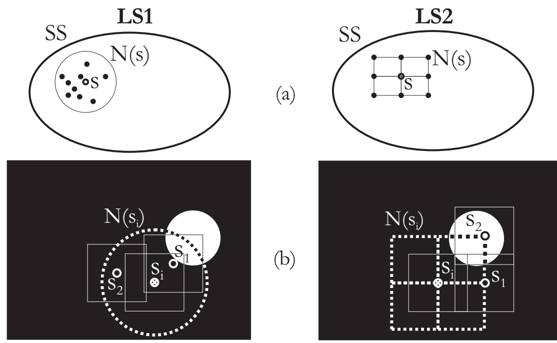

Fig. 1. a) LS1 (izq) y LS2 (dcha) y (b) seguimiento de 1 objeto

nes horizontal y vertical de la ventana de interés. El método LS1 (Fig. 1.b izquierda) define una vecindad $N(s_i)$ de la solución s_i y genera soluciones aleatorias en esa región. El método LS2 (Fig. 1.b derecha) define una malla de soluciones alejadas una distancia predefinida y las explora.

Fundamentalmente, existen dos estrategias que dirigen una LS, denominadas "*first improvement*" y "*best improvement*" [6]. La primera, es una estrategia secuencial, que busca en la vecindad actual hasta que encuentra la primera solución que mejora la actual. La segunda explora todas las posibles soluciones de una vecindad y elige la mejor en cada iteración. En este trabajo, nos centramos en esta segunda, puesto que se adapta mejor a la arquitectura de las GPUs.

B. Multiescala

Las características en una imagen pueden mostrarse en diferentes escalas, y estas escalas pueden cambiar a lo largo del tiempo debido a las variaciones en el tamaño que tienen lugar cuando los objetos se mueven en relación a la cámara [2]. En los problemas de seguimiento visual, estas características están relacionadas con la caracterización del objeto. Por ejemplo, un objeto caracterizado por su color producirá una silueta (o *blob*) representativo en la imagen de medida. Si el objeto se mueve hacia la cámara o se aleja de ella, el tamaño (y probablemente la forma) de la silueta cambiará. Los métodos multiescala (MS) afrontan esta variabilidad cambiando y seleccionando la escala mas apropiada para encontrar la característica de interés. Estas estrategias son muy interesantes para este trabajo, ya que permiten dirigir el procedimiento LS de una forma muy efectiva.

IV. Arquitectura propuesta para filtros de partículas y seguimiento visual

Para este trabajo se proponen las unidades de procesamiento gráfico (GPUs) o procesadores gráficos. Debido a la rápida expansión de los juegos por computador y las tecnologías multimedia, las GPUs se han convertido en un hardware con un alto poder de cómputo y de bajo coste, doblando su rendimiento cada 9 meses. La propuesta del uso de este tipo de arquitecturas para el seguimiento visual 2D se fundamenta en los resultados obtenidos en trabajos como [16] y [17]. En [16], los autores demostraron una mejora de un 30 % frente a soluciones optimizadas para CPU en problemas de segmentación mediante la manipulación directa de los píxeles con una tarjeta gráfica. Para aquellas operaciones que requieran de la manipulación directa de los píxeles, la GPU es muy eficiente y ofrece un elevado rendimiento debido a que esta básicamente diseñada para este propósito en su contexto de gráficos generados por computador. El procesamiento de imágenes y la visión por computador son dos de los mï¿½s importantes campos que pueden beneficiarse del rendimiento de los procesadores gráficos.

A. Procesadores gráficos

Estos procesadores se han convertido en unidades de cómputo vectorial programables, con precisión en coma flotante, y un gran ancho de banda a memoria, lo que ha favorecido que investigadores de todo el mundo los haya empezado a utilizar como coprocesadores de altas prestaciones y bajo coste en estaciones de consumo, creando una disciplina cada vez más reconocida como GPGPU (*General Purpose computation using Graphics Processing Units*) [15].

Tradicionalmente, los procesadores gráficos incluían tres etapas de computación en serie, que en su conjunto se denominaba *pipeline de renderizado*. Este *pipeline* dividía conceptualmente el cómputo en procesamiento de primitivas (vértices), una etapa de rasterización para la creación de fragmentos, y una etapa final de combinación de fragmentos y texturizado para dotar de un mayor realismo a las escenas fotorrealistas. Estas etapas podían ser utilizadas como máquina de estados desde un API (Application Programming Interface) de acceso al hardware gráfico como OpenGL o Direct3D.

A partir del año 1999, los mayores fabricantes de GPUs liberan las etapas de procesamiento de vértices y combinación de fragmentos haciéndolos programables para poder personalizar el procesamiento de dichas etapas a través de programas independientes para el procesamiento de vértices o de fragmentos. Estos programas, denominados sombreadores (*shaders*), permiten acceder a los recursos hardware de una manera relativamente transparente, utilizando un lenguaje de *shading* o lenguaje de alto nivel para la programación de *shaders*. Entre los lenguajes más populares para esta tarea se encuentran Nvidia Cg [5], Microsoft HLSL[10] y OpenGL Shading Language [9].

Bajo este modelo de cómputo basado en el uso del API gráfica, el programador debe enviar los datos de entrada al procesador gráfico a través de texturas. En el contexto de la informática gráfica, una textura es una imagen que puede mapearse sobre una estructura poligonal para dotar de realismo al modelo 3D. Al igual que una imagen, en la textura se pueden representar 4 valores (canales de color RGB y transparencia A) por cada posición accesible, denominada *texel* (*texture element*). Estas texturas son equivalentes a arrays bidimensionales de hasta 4 capas y tamaños de hasta 4096x4096(x4) elementos, que se guardan en memoria de vídeo accesible desde el procesador gráfico con anchos de banda mucho mayores que

los respectivos entre la CPU y la memoria principal. Sin embargo, para sacar partido del ancho de banda a memoria, es necesario que el acceso sea sistemáticamente en bloque, tratando de ejecutar el mismo programa sobre una colección completa almacenadas en las texturas participantes, en lugar de poder realizar accesos directos más propios del modelo de cómputo tradicional en CPU. En definitiva, la plataforma GPU permite realizar un cómputo en paralelo sobre grandes colecciones de datos de manera muy eficiente y escalable en el tiempo. Esta escalabilidad en el tiempo responde a la manera en que la tecnología asociada al hardware gráfico evoluciona. De forma general, por la facilidad de la integración de transistores dedicados a cómputo así como por el empuje de la industria de los videojuegos, la tendencia de los procesadores gráficos llega a duplicar e incluso triplicar la Ley de Moore de los procesadores generales, y una solución que actualmente no consiga un gran rendimiento en GPU puede ser significativamente superior en pocos meses.

V. Nuestro enfoque

En esta sección se detallan las decisiones tomadas y algunos detalles de implementación.

A. Búsqueda Local

La inclusión de un algoritmo LS dentro de PF da lugar a un algoritmo más flexible. Por ejemplo, el espacio de estados definido en la etapa PF puede ser diferente del empleado en la etapa LS. Esto es muy útil en problemas de seguimiento. Por ejemplo, consideremos el seguimiento de un objeto móvil y deformable, del que se desea estimar tanto su posición como su tamaño. La formulación de este problema en PF requiere que la estructura de una partícula venga dada por un vector de cuatro componentes $[x, y, S_x, S_y]$. Sin embargo el mismo problema puede formularse bajo el marco de LSPF mediante una estimación en dos fases. Durante la etapa PF, se estima sólamente la posición del objeto utilizando ventanas de tamaño fijo (resultando un espacio de estados 2D). A continuación, LS se encarga de mejorar la calidad de esta estimación y ajusta el tamaño de la *bounding box* para que el objeto quede encuadrado dentro de ésta. La principal ventaja de esta aportación es la reducción de carga computacional.

La segunda idea es que LS es un método computacionalmente costoso, pero que puede proporcionar buenos resultados en la estimación de PF. Dada una estimación inicial de PF, la etapa de LS proporciona un refinamiento de la calidad de esta estimación más precisa que aquéllas proporcionadas por otras estrategias como el remuestreo. LS se adecúa perfectamente a la arquitectura propuesta si se utiliza una estrategia *best improvement* sobre una vecindad predeterminada.

Se han desarrollado dos búsquedas locales: una aleatoria (LS1) y otra sistemática (LS2). Éstas se diferencian en la manera de generar las regiones de interés dentro del área de búsqueda, tal y como se muestra en la Fig. 2.

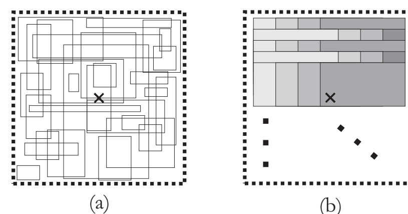

Fig. 2. generación de ROIs (a) aleatoria y (b) sistemática

B. Búsqueda Local Dirigida por un Procedimiento Multiescala

Los procesos de mejora local son computacionalmente costosos y consume mucho tiempo en implementaciones CPU (donde la generación y evaluación de las soluciones se realiza secuencialmente). La carga computacional es significativa cuando la vecindad considerada es lo suficientemente grande y los objetos de interés pueden cambiar sus dimensiones proyectadas de forma considerable durante la secuencia. El uso de la GPU nos permite generar y evaluar soluciones en paralelo, disminuyendo significativamente el tiempo de ejecución.

Para aprovechar las características de la GPU, se propone un estrategia MS. Este método evalúa diferentes escalas de una solución inicial con el fin de dirigir de forma más eficiente el proceso LS. Así, el procedimiento MS comienza con la mejor solución encontrada por PF y evalúa regiones de interés más pequeñas y más grandes con centro en la estimación inicial dada. El número de escalas puede ser variable. Para la plataforma propuesta se han considerando dimensiones potencia de dos para las regiones de interés, modificando los tamaños de 8×8 a 128×128. Esta colección de niveles de escala asegura la cobertura del objeto seguido en las situaciones prácticas consideradas. Se evalúa cada nivel de resolución y el mejor de todos ellos actúa como semilla para la fase LS. La evaluación de la calidad de los niveles de escala debe considerar el hecho de que los niveles de escala mayores pueden contener objetos pequeños que ya eran contenidos en niveles de escala menores. Por esta razón, se debe tener en cuenta el ratio entre los píxeles etiquetados como objeto y como fondo. Se ha determinado empíricamente un buen rendimiento cuando se calcula el peso correspondiente a cada nivel de escala como:

$$W_{LOS_j} = \sum_{i \in LOS_j} w_i \qquad (3)$$

donde i itera sobre todos los píxeles de la j-ésima LOS, y w_i:

$$w_i = \begin{cases} 1, & \text{si } i \in \text{objeto;} \\ -0{,}075, & \text{si } i \in \text{fondo.} \end{cases} \qquad (4)$$

La evaluación de las ventanas en LS sigue la misma estrategia que la seguida en el procedimiento MS, ponderando cada ventana como se describe en las ecuaciones (3) y (4). Sin embargo, en esta ocasión se penalizan más los píxeles etiquetados como fondo (-0.25) por ser la última etapa de refinamiento.

Juntos, los métodos MS y LS establecen una etapa de optimización para PF especialmente adecuada para la

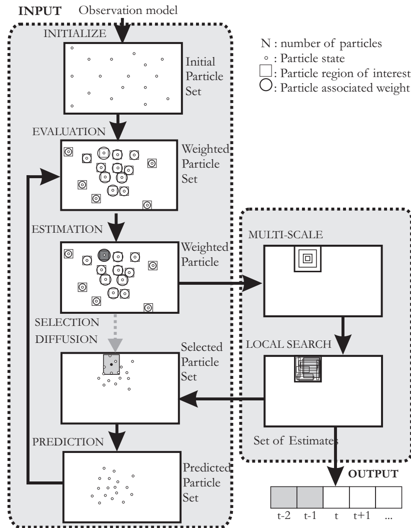

Fig. 3. PF (izq) y adaptación MSLS (dcha)

GPU. Tal y como puede verse en la Fig. 3 la etapa MS y LS se localiza justo después de la etapa de estimación dentro del flujo de PF.

C. *Detalles de implementación GPU*

PF es un método especialmente adecuado para su implementación en tarjetas gráficas. Sin embargo, los cambios algorítmicos que hay que realizar para portar su ejecución a esta plataforma no son triviales. Por claridad se expondrá el caso de seguimiento de un único objeto con una estructura de solución dada por $\mathbf{s}=(x,\ y)$. El proceso comienza con una etapa de inicialización, en la que se reserva memoria en el host y GPU, y algunas de las texturas de datos predefinidos se cargan en memoria de vídeo.

C.1 Modelo de Observación e inicialización

El fotograma en el instante t se carga en memoria de vídeo y, antes de que las etapas del PF se ejecuten, se realiza un preprocesamiento dependiente de la aplicación con objeto de extraer la medida. En nuestro caso, hemos utilizado detección del color de la piel y sustracción de fondo. Una vez que se dispone de medida, se procede a seguir las siluetas obtenidas, utilizando PF. La etapa de inicialización del PF se ejecuta sólo en la primera iteración y consiste en la inicialización de un conjunto de partículas sobre todo el espacio de estados. Desde el punto de vista de la imagen, esto equivale a dispersar regiones de interés (ROIs) sobre el modelo de observación. Esta información inicial se guarda en una textura (*textura solución*), donde los dos primeros canales de un píxel RGBA contienen el estado de una partícula **s**. Como la GPU es muy eficiente en el manejo de texturas 2D, se puede incluir la información de las N partículas en una textura 2D de dimensiones $\sqrt{N} \times \sqrt{N}$.

C.2 Etapa de evaluación

En la etapa de evaluación se asignan peso a cada partícula en función del número de pï¿½xeles etiquetados que

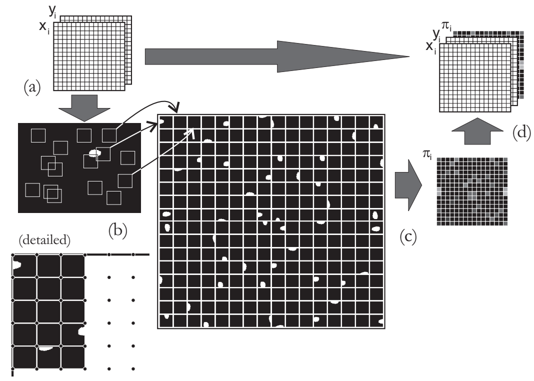

Fig. 4. Etapa de evaluación en GPU.

pertenecen a la región de interés asociada a cada partícula. La estrategia paralela propuesta para la arquitectura es la de construir una *textura de medidas* donde representar el conjunto de partículas y sus respectivas ROIs, tal y como se muestra en las Figs. 4.a y 4.b, donde se representa gráficamente el proceso de creación de una textura a través de la renderización *offline* de manera mucho más eficiente que la creación en CPU para su posterior subida a memoria de vídeo.

Como puede verse en Fig. 4, el número de partículas queda limitado por el espacio de memoria. Asumiendo una textura de medida de tamaño 1024×1024, sería adecuada cualquier combinación que la cubra (por ejemplo, 4096 partículas de 16×16). Las últimas tarjetas permiten texturas con tamaños por encima de los 8192×8192 píxeles (unos 268×10^6 valores), que es suficientemente grande para cualquier tipo de configuración.

C.3 Etapa de estimación

En esta etapa se selecciona la estimación mas probable del conjunto de partículas. Para ello se establece una correspondencia entre los pesos de las partículas y su estado. El estado de cada partícula se encuentra codificado en la textura solución, mientras que el peso de cada una de ellas se encuentra almacenado en la textura de pesos, resultante de la etapa de evaluación. Ambas texturas son del mismo tamaño y pueden combinarse en una sola. Para ello se emplean tres de los cuatro canales RGBA, dos para almacenar el estado de cada partícula y un tercero para almacenar su peso (ver Fig. 4.d)). A continuación se realiza una reducción "máximo" de la textura generada, obteniendo así el estado más probable, que es la estimación de PF para el instante t.

C.4 Método de multiescala

La estimación anterior, es la solución inicial de la etapa multiescala (MS). Aquí se realizan los pasos descritos en la Sección V-B para los diferentes niveles de escala (LOS) considerados. Como resultado, se obtiene el mejor nivel de escala, que se utilizará en la LS.

C.5 Búsqueda local

Una vez conocido el mejor nivel de escala, éste se utiliza para limitar el espacio de búsqueda en LS. Un *frag-*

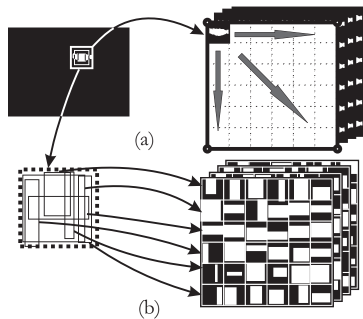

Fig. 5. Creación y cómputo de la textura MSLS

ment shader calcula el solapamiento existente entre cada ventana de la LS y el nivel de escala calculado. Esto se lleva a cabo de forma paralela, mediante grandes texturas de datos, la textura de nivel de escala y las textura de LS. En lugar de repetir el array CPU que representa el nivel de escala seleccionado para su posterior subida a memoria de vídeo como una textura, se hace uso de la característica auto-repetir de la GPU (`GL_REPEAT`). De esta forma se consigue mantener la carga de trabajo en la GPU, evitando los cuellos de botella producidos por las transferencias CPU-GPU (ver Fig. 5.a)

Se crean diferentes texturas RGBA de LS *offline* y se cargan en memoria de vídeo en la etapa de inicialización. Estas texturas difieren en los tamaños de las ventanas que contienen, los cuales se corresponden con los distintos niveles de escala. A continuación, ambas texturas son examinadas en un *fragment shader* que calcula el solapamiento mediante un producto binario fragmento a fragmento. A continuación se realiza una operación de reducción para obtener la mejor configuración. Nuevamente las dimensiones de estas texturas están limitadas por el máximo tamaño de textura soportado por el hardware gráfico. En una configuración típica, con un nivel de escala de 64×64 y texturas RGBA de 1024×1024, se evaluan 1024 ventanas diferentes en cada pasada. La mejor estimación conseguida por LS se convierte en el estimado del proceso, reemplazando a la que fue establecida como mejor partícula en la etapa PF.

C.6 Etapa de selección

La etapa de selección elige con probabilidad proporcional a sus pesos las partículas que sobrevivirán en el siguiente instante de tiempo. Para ello, se crea una textura de igual tamaño que la textura de soluciones con valores aleatorios [0, `maxWeight`], donde `maxWeight` es el máximo valor de pesos en cada instante. Los valores de esta textura de aleatorios se utilizan para seleccionar aquellas partículas que contienen pesos superiores al correspondiente aleatorio en la misma posición. Si el peso de una partícula no es superior al correspondiente valor de esta textura de aleatorios, se reemplaza su estimación por la mejor del conjunto. Este procedimiento remplazará la mayor parte de las partículas de menor calidad. Sin embargo, existirá cierta probabilidad de que algunas partículas con pesos bajos sobrevivan, lo cual proporcionará diversidad al conjunto. Con el fin de no incluir cierta predisposición espacial en la etapa de selección, se realiza un desplazamiento, en las coordenadas de textura que se asocian al *quad* utilizando dos valores escalares aleatorios, simulando así el efecto de una nueva selección aleatoria. Así, se evita generar una nueva textura de aleatorios en cada instante de tiempo t.

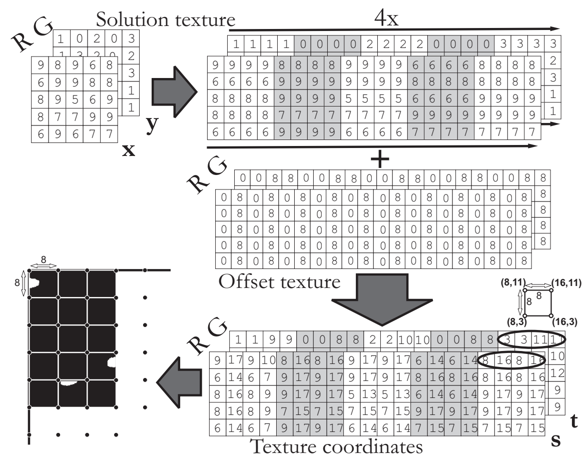

Fig. 6. Realimentación OpenGL.

C.7 Etapas de difusión y predicción

La etapa de difusión previene de la pérdida de diversidad producida en la etapa anterior. Para ello, se aplica ruido gausiano a cada partícula de la población. Una forma eficiente de hacer esto en la GPU es mediante otra textura de valores aleatorios uniformes en el rango $[-\Delta x, +\Delta x]$ y $[-\Delta y, +\Delta y]$, donde Δx y Δy son los desplazamientos máximos para cada coordenada. Esta textura será utilizada en cada iteración.

C.8 Realimentación al instante siguiente

Después de aplicar el modelo de movimiento, el conjunto de partículas contiene la estimación a priori para el siguiente instante de tiempo $t+1$. Con el fin de mantener todo el cómputo en la GPU, se requiere convertir la salida del pipeline de renderizado en la entrada para el siguiente instante. Para ello, se extiende la textura solución por cuatro en el eje horizontal y se aplica un *fragment shader* que añade desplazamientos Δs y Δt (o coordenadas relativas) a cada nueva solución (x, y). Esta textura incorpora las coordenadas de nuevos *quads* para la creación de la textura de medidas y se lee desde la primera etapa del PF en GPU gracias al uso de extensiones FBOs y PBOs.

VI. Resultados

Los experimentos fueron realizados en un Pentium 4 Dell XPS600 3.6GHz con 1GB RAM. En GPU se contó con una Nvidia GeForce 7800 GTX (G70) de 256MB y una GeForce 8800 GTS (G80) con 640MB, ambas sobre un Pentium 4 3.0GHz y 1 GB RAM usando controladores Nvidia v169.21. Todas los programas se han escrito en C y, para las versiones en GPU, se utilizó OpenGL y Nvidia Cg 2.0.Algunas variables de configuración fueron:

TABLA I

PRECISIÓN Y EXACTITUD DE PF, MSLS1 Y MSLS2)

Alg.	♯evals.	A_x	A_y	P_x	P_y
PF	800	3.13	3.10	3.64	3.57
MSLS1	703	1.49	1.28	1.93	1.76
MSLS2	703	0.79	0.94	0.43	1.19
PF	1300	2.96	3.01	3.49	3.53
MSLS1	1203	1.23	1.09	1.66	1.49
MSLS2	1203	0.39	0.17	0.61	0.44
PF	1800	3.12	3.03	3.60	3.52
MSLS1	1703	1.10	0.98	1.60	1.36
MSLS2	1703	0.33	0.11	0.48	0.39

- Número de partículas: potencias de 2 para empaquetar mejor las soluciones en las texturas de vídeo (64-16384).
- Niveles de Escala (LOS): se utilizaron 3 niveles (LOS), por ejemplo, para ROIs de 16×16, se tomaron LOS de 8×8, 16×16 and 32×32.
- Método de búsqueda local (LS): se probaron búsquedas locales aleatorias (LS1) y sistemáticas (LS2).
- Número de objetos: define la longitud del vector de características de la partícula.

A. Medidas de Calidad

Para medir la calidad de la estimación hemos utilizado las definiciones de precisión y de exactitud. Exactitud es el grado de concordancia de una medida con su valor real. La precisión o reproducibilidad es el grado de similitud de diferentes medidas en las mismas condiciones. La exactitud (A_x) en la coordenada x se ha calculado del siguiente modo:

$$A_x = \frac{1}{F} \sum_{f=1}^{F} |x_A^f - x_E^f| \quad (5)$$

donde x_A y x_E son los valores verdadero y estimado de x, respectivamente, y F es el número total de fotogramas. Para calcular A_y se utiliza la misma ecuación.

Como medida de la precisión en cada coordenada P_x y P_y, hemos calculado el promedio de la desviación estándard de las estimaciones en 10 experimentos bajo las mismas condiciones. La Tabla I muestra los resultados obtenidos cuando se sigue un objeto de 20×20 en una secuencia sintética. La Tabla compara PF, MSLSPF con MSLS1 y MSLSPF con MSLS2 y diferente número de evaluaciones en cada método: PF con 800, 1300 y 1800 partículas, y MSLS con 200 partículas, 3 niveles de escala y 500, 1000 y 1500 evaluaciones en LS (es decir, 703, 1203, 1703 evaluaciones totales). Los resultados obtenidos muestran la mejor precisión y exactitud de MSLSPF (especialmente con MSLS2) comparado con PF. Además, MSLSPF permite ajustar la ventana de interés al tamaño del objeto.

También hemos probado el algoritmo sobre secuencias reales de la colección CAVIAR [4]. La Fig. 7 muestra los resultados obtenidos en el seguimiento de una persona.

Fig. 7. Seguimiento de una persona con (a) PF y (b) MSLSPF.

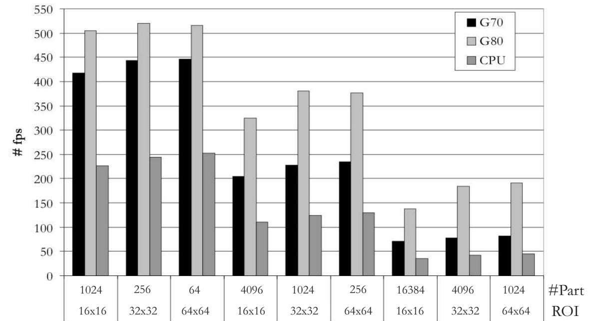

Fig. 8. PF-GPU Vs. PF-CPU en el seguimiento de un objeto

Tanto PF como MSLSPF describen la posición del objetivo, pero sólo MSLSPF ajusta la ventana de interés a las dimensiones del mismo.

B. Rendimiento de la arquitectura propuesta

Hemos probado el rendimiento de la GPU en comparación con la CPU para PF y MSLSPF, en el seguimiento de uno y varios objetos, utilizando diferentes resoluciones de vídeo. Además hemos probado las diferencias entre distintas GPUs. Algunas consideraciones sobre la experimentación GPU son:

- Dimensiones de la ROI: 16×16, 32×32 and 64×64 píxeles.
- Número de evaluaciones de LS: utilizamos una textura RGBA de 1024×1024 que permite empaquetar 1024 ROIs de 64×64 (o 4096 de 32×32).

B.1 Seguimiento de un objeto

En este primer experimento se compara una implementación estándar de PF sobre diferentes plataformas. Los resultados sobre 780 fotogramas (26 segundos) de un vídeo de 320×240 se muestran en la Fig. 8. La reproducción de video se lleva a cabo a 30 *fps*. Aquí se muestra que la tasa de procesamiento de la sustracción de fondo para la ejecución del algoritmo para todas las configuraciones de PF es más rápida que la restricción de tiempo real. Sin embargo, las implementaciones GPU son mucho más eficientes que la versión CPU, hasta un 390 % y 137 *fps* en las configuraciones más costosas para el modelo G80. Las configuraciones más eficientes para la CPU en comparación con la GPU son aquellas con menor número de partículas, en las que el ratio entre la transferencia de datos y el cómputo es muy alto.

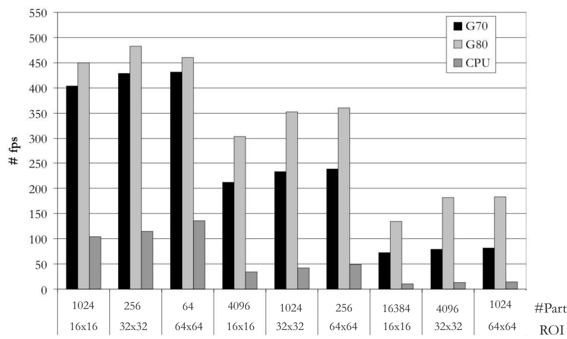

Fig. 9. PF-GPU Vs. PF-CPU en el seguimiento de 4 objetos

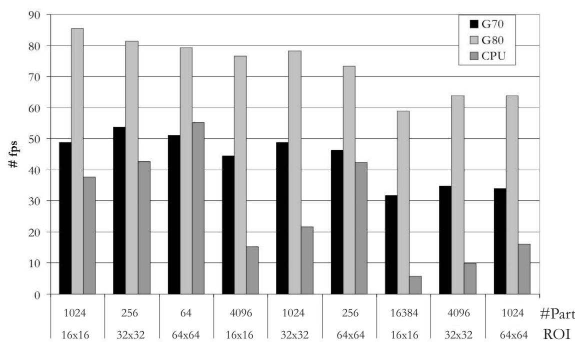

Fig. 10. MSLS-GPU Vs. MSLS-CPU en el seguimiento de 4 objetos

B.2 Seguimiento de múltiples objetos

Para el seguimiento de múltiples objetos, las diferencias entre las implementaciones CPU y GPU del PF estándard son mucho más pronunciadas. Los resultados dela comparativa para ROIs de tamaños desde 16×16 a 64×64 y un número de partículas entre 64 y 16384 se muestran en la Fig. 9. En la gráfica se aprecia claramente que la GPU mantiene tasas de procesamiento similares al seguimiento de 1 objeto. Esto es razonable, ya que la implementación previa sobre GPU utilizaba un sólo canal incluso en los casos más costosos. Aprovechando los 4 canales (RGBA), podemos codificar el mismo número de partículas en la misma configuración sin ninguna penalización en rendimiento. En esta implementación, la GPU es entre 3.39 (64 partículas y ROIs de 64×64) hasta 14.14 (4096 partículas y ROIs de 32×32) veces más rápida que la CPU.

Finalmente, en la Fig 10 se muestran los resultados experimentales para el MSLSPF en el caso de seguimiento de 4 objetos. La GPU procesa más rápido que la restricción de tiempo real, ya que ambos modelos ofrecen tasas superiores a 30 *fps* para cada configuración probada y, en algunos casos, el modelo G80 es 10 veces más rápido que la configuración CPU.

VII. Conclusiones

En este trabajo se ha presentado una búsqueda local multiescala (MSLS) aplicada como extensión de un filtro de partículas (PF) para mejorar su rendimiento en el seguimiento de múltiples objetos en 2D. MSLS representa una mejora significativa respecto a PF, tal y como muestran los resultados obtenidos. Los procedimientos de búsqueda local se han demostrado muy efectivos en la resolución de este problema. Además, la aceleración de los distintos procedimientos mediante el uso de GPUs, especialmente los de mejora local, ha proporcionado un algoritmo un 1100 % más rápido que la versión CPU. Los resultados obtenidos nos permiten ser optimistas en lo que respecta al uso de este tipo de arquitecturas en la aceleración de heurísticos de mejora y metaheurísticas aplicados a otros problemas de optimización.

Agradecimientos

Este trabajo ha sido financiado parcialmente por la Comunidad de Madrid con el proyecto URJC-CM-2007-CET-1724, y por *Nvidia Professor Partnership Program*.

Referencias

[1] Aarts, E., Lenstra, J.K. Local Search in Combinatorial Optimization. Princeton University Press (2003)

[2] Bretzner, L., Lindeberg, T.: Feature tracking with automatic selection of spatial scales. Computer Vision and Image Understanding, **71**, 385–392 (1998)

[3] Carpenter, J., Clifford, P., Fearnhead, P.: Building robust simulation based filters for evolving data sets. *Tech. Rep., Dept. Statist., Univ. Oxford, Oxford, U.K* (1999)

[4] EC Funded CAVIAR project/IST 2001 37540, found at URL: http://homepages.inf.ed.ac.uk/rbf/CAVIAR/

[5] NVIDIA, Cg Toolkit: Cg 1.4 rc 1, Release Notes, 2005.

[6] Chiang W-C. The application of a tabu search metaheuristic to the assembly line balancing problem. Annals of Operations Research 77, 209–227 (1998)

[7] Curio, C., Giese, M.: Combining View-Based and Model-Based Tracking of Articulated Human Movements. Proc. of the IEEE Workshop on Motion and Video Computing **2:2**, 261–268 (2005)

[8] Gordon, N.J., Salmond, D.J., Smith, A.F.M.: Novel approach to nonlinear/non-Gaussian Bayesian state estimation. IEE Proc. F Radar & Signal Processing **140:2** (1993)

[9] R. J. Rost, OpenGL Shading Language, Pearson Education, 2004.

[10] Microsoft, Documentación HLSL (High Level Shading Language), DirectX SDK, 2008.

[11] Isard, M., Blake, A.: Visual tracking by stochastic propagation of conditional density. *In Proc 4th European Conf. Computer Vision*, 343–356, (1996)

[12] Isard, M., Blake, A.: Condensation - conditional density propagation for visual tracking. *International Journal of Computer Vision*, **29:1**, 5–28 (1998)

[13] Lanvin, P., Noyer, J.-C., and Benjelloun, M.: An hardware architecture for 3D object tracking and motion estimation. In Proc. of Intl. Conf. on Multimedia and Expo (ICME) (2005)

[14] MacCormick, J., Blake, A.: Partitioned sampling, articulated objects and interface-quality hand tracking. Proc. 7th European Conf. on Computer Vision **2**, 3–19 (2000)

[15] Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E., Purcell, T. J.: A Survey of General-Purpose Computation on Graphics Hardware, Computer Graphics Forum, **26:1**, 80–113 (2007)

[16] Yang, R., Welch, G.: Fast image segmentation and smoothing using commodity graphics hardware. Journal of Graphics Tools, **7:4**, 91–100 (2002)

[17] Yang, R., Pollefeys, M.: Multi-resolution real-time stereo on commodity graphics hardware. Computer Vision and Pattern Recognition, **1**, 211–217 (2003)

[18] Yilmaz, A., Javed, O., Shah, M.: Object tracking: A survey. ACM Comput. Surv. **38:4**, (2006)

[19] Zotkin, D., Duraiswami, R., Davis, L.: Joint Audio-Visual Tracking Using Particle Filters. EURASIP Journal on Applied Signal Processing, 11, 1154–1164 (2002)