Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXXIII, págs. 1721-1731 (artículo completo)
Graciela Bertolino, Mariano Cantero, Mario Storti y Federico Teruel (Eds.)
San Carlos de Bariloche, 23-26 Setiembre 2014

UN MÉTODO EFICIENTE PARA LA SUSTRACCIÓN DE FONDO EN VIDEOS USANDO GPU

L. Gervasoni[a,b], J.P. D'amato[a,b,c], R. Barbuzza[a,b] and M. Vénere[a,b]

[a]*Facultad de Cs. de Exactas, Univ. Nac. del Centro de la Prov. de Bs.As. , Paraje Arroyo Seco S/N, 7000, Tandil, Buenos Aires, Argentina, gervasoni.luc@gmail.com*

[b]*Instituto PLADEMA, Paraje Arroyo Seco S/N, 7000, Tandil, Buenos Aires, Argentina*

[c]*Consejo Nacional de Investigaciones Científicas y Técnicas, Av. Rivadavia 1917, CABA, Argentina*

Keywords: Background subtraction, foreground extraction, motion detection

Abstract. La sustracción de fondo es una técnica ampliamente utilizada para detectar objetos en movimiento a partir de cámaras estáticas. En el campo de la vídeo-vigilancia, es una técnica indispensable, ya que permite discriminar rápidamente cuando ha surgido un evento y permite a su vez realizar algún tipo de estudio de movilidad. En los últimos años, se han propuesto muchos métodos con diferentes fortalezas y requerimientos para realizar la sustracción de fondo. Uno de los métodos más novedosos es el Visual Background Extractor (ViBe), el cual utilizando análisis estocástico multimodelo permite discriminar objetos del fondo con una alta tasa de confianza. En este trabajo se propone una variante de este algoritmo para el estudio de objetos en situaciones particulares a partir de videos de vigilancia. Dado que se desea trabajar con videos de alta resolución, se presenta una implementación en GPU utilizando OpenCL para lograr un análisis en tiempo real. Presentamos la funcionalidad en casos de estudios típicos en para este área.

1 INTRODUCCIÓN

La sustracción de fondo es una etapa fundamental de una gran cantidad de aplicaciones en las que se necesita detectar movimiento, identificar y/o seguir objetos en secuencias de imágenes dinámicas. Entre estas aplicaciones se pueden nombrar la vídeo-vigilancia, como la detección y captura del flujo de movimiento, la estimación de pose, la interacción hombre-computadora o la codificación de vídeo basado en el contenido, entre otros. Para detectar objetos en movimiento, cada *frame* de entrada se compara con un modelo de fondo obtenido de los *frames* anteriores, y se clasifica la escena en zonas de fondo o estáticas (*background*), y zonas dinámicas que se corresponden con el primer plano (*foreground*).

Dado que el fondo no es invariante en el tiempo, es necesario realizar un modelado dinámico del mismo para adaptarlo a las variaciones que se puedan producir. En el comienzo de una secuencia, cualquier cambio significativo en una región de la imagen respecto del modelo de fondo será considerado *foreground*. En general, cada elemento del modelo se actualiza por cada cierto intervalo de tiempo, siempre que el mismo haya perdido significancia.

En el contexto de la vídeo-vigilancia existen ciertos desafíos que aún no han sido resueltos con éxito, debido a múltiples factores que inciden en la complejidad de sustraer el fondo en este tipo de secuencias. En la práctica, el ambiente tiende a ser complejo debido a la diversidad de alteraciones que pueden presentarse. Se pueden destacar los cambios en la iluminación, la existencia de varios objetos en movimiento, la superposición de éstos, cambios en las estructuras estáticas y la presencia de ruido en la captura de las imágenes, entre otras.

En la actualidad existe una tendencia de los medios de captura hacia obtener imágenes a una velocidad y resolución cada vez mayores. Es por esto que el formato HD (720p) y FullHD (1080p) son un estándar, que se ha extendido incluso en las cámaras de los celulares y en lo pronto el formato 4K también será soportado en estos dispositivos. Es importante destacar nuevamente que no sólo ha aumentado la resolución en la captura, sino también la velocidad de captura. Gracias a cámaras comerciales como las *goPro*, es posible contar con 120 *frames* por segundo en formato HD. Seguramente no siempre es necesario procesar todos los *frames* obtenidos, pero de todas maneras existe una tendencia hacia la búsqueda de estrategias cada vez más eficientes para procesar la información proveniente de ellas.

En este trabajo se presenta una técnica de sustracción de fondo implementada en GPU, para lograr el procesamiento de imágenes de alta resolución en tiempo real. De esta manera se presenta una solución para mejorar tanto la tasa de detección (aciertos) como el tiempo computacional, para su aplicación en áreas que requieren simultáneamente eficiencia y robustez como la vídeo-vigilancia. Se prestó especial atención a un método de extracción de contornos, que es generalmente implementado en forma secuencial y debió ser replanteado para su ejecución en paralelo. En la siguiente sección se discuten brevemente trabajos actuales relacionados, en la sección 3 se describen el método con los módulos de mejoras propuestos. En la sección 4 se detalla la implementación y a continuación los resultados y conclusiones obtenidas.

2 TRABAJOS RELACIONADOS

Existe una gran cantidad de contribuciones a la sustracción de fondo que han sido propuestas, así como también revisiones, comparativas y *data-sets* (Vacavant et al., 2013; Goyette et al., 2012; Benezeth et al., 2008; Piccardi, 2004). En una gran mayoría, los métodos realizan un modelado y análisis basado a nivel de píxeles. Entre ellos, algunos consideran que los valores de intensidad de un píxel puede ser modelado con una distribución de probabilidad Gaussiana o una mezcla de ellas (*Mixture of Gaussians*), y determinan para cada valor de entrada que prob-

abilidad tiene de pertenecer al fondo. A su vez, utilizan parámetros que permiten actualizar el modelo para ir ajustándolo a lo largo de una secuencia. La principal desventaja radica en que la suposición del comportamiento Gaussiano no siempre se sostiene, sobre todo en escenas complejas (KaewTraKulPong and Bowden, 2002; Zivkovic, 2004; Zivkovic and van der Heijden, 2006), por lo que han surgido otras técnicas que han mostrado mejorar la tasa de detección de estos métodos.

En particular, en este trabajo nos centramos en el desarrollo de un sustractor de fondo basado en píxel propuesto en (Barnich and Van Droogenbroeck, 2009, 2011; Van Droogenbroeck and Paquot, 2012). El análisis estocástico que utiliza este método para modelar el fondo ha mostrado altas tasas de detección en entornos complejos, tal cual suelen ser típicos en la vídeo-vigilancia. Entre las virtudes se destacan el bajo tiempo de cómputo, las altas tasas de detección y la robustez ante la existencia de ruido, las cuales son necesarias en capturas de cámaras de supervisión, pero que no son consideradas en conjunto en las técnicas mencionadas. Estas características son imprescindibles en el contexto de la vídeo-vigilancia. Por otro lado, tienen ciertas falencias reconocidas, como su falla en la detección cuando los objetos son similares al fondo.

La técnica que se utiliza como base en este trabajo utiliza información de color para determinar la detección del fondo. El modelado de fondo está basado en muestras reales obtenidas de la historia de cada píxel. La actualización del modelo es aleatoria, con cierta probabilidad de reemplazar uno de los valores guardados anteriormente por el nuevo valor. La aleatoriedad sobre este mecanismo de actualización permite que valores introducidos para modelar el fondo tengan un tiempo de vida mayor, reduciendo gradualmente con el tiempo la probabilidad de que persistan en el modelo. Contiene una política de actualización conservativa, ingresando al modelo de fondo únicamente valores clasificados como *background*.

Además, existen algunas técnicas de *Background Substraction* implementadas en placas gráficas. Tal como se conoce, el estándar de OpenCL de programación en paralelo para procesadores multi-núcleo y GPUs propuesto por *Khronos Group*, ha ganado popularidad y está siendo adoptado incluso en navegadores Webs. Este estándar utiliza un lenguaje de programación tipo Ansi C, el cual es cargado y compilado de forma dinámica en un dispositivo. Cada programa se instancia en un método o *kernel*, el cual corre en múltiples *threads* dentro de una unidad de cómputo. Cada *kernel* accede a espacios de memoria distribuidos en 3 niveles.

Existen varios métodos de los nombrados que han sido portados a su versión en placas gráficas y que hoy se incluyen en el paquete de OpenCV. Estos se encuentran implementados en CUDA que es propietaria de NVidia. Igualmente, para este tipo de implementaciones es necesario tener en cuenta algunas características que hagan un mejor aprovechamiento del hardware. En (Schreiber and Rauter, 2009) y en (Yin Li, 2012) se muestra una variación del método de *Mixture of Gaussians* adaptado en GPU, alcanzando un buen tiempo de procesamiento.

3 SUSTRACTOR DE FONDO SOBRE CÁMARAS ESTÁTICAS

El método ViBe (Barnich and Van Droogenbroeck, 2009, 2011) utiliza un modelo del fondo que almacena para cada píxel, N muestras seleccionadas aleatoriamente de *frames* ya procesados. Si consideramos m=$\{m_1, m_2, ..., m_N\}$ las muestras correspondientes al píxel actual p_t, cada m_i para i=1..N, consiste en un vector m_i=(r,g) que usa dos componentes del espacio RGB. Esto puede adaptarse fácilmente para extenderlo al espacio RGB, escala de grises u otro. El hecho de excluir el componente azul se debe a que en las secuencias de vídeo-vigilancia este componente suele encontrarse más saturado que los otros componentes. Luego, se define la condición de intersección I(p_t, m_i) igual a 1, cuando la distancia euclídea evaluada entre el valor del píxel actual y una muestra del modelo m_i es menor o igual a un determinado valor de

radio R, es decir:

$$I(p_t, m_i) = \begin{cases} 1 & \text{si } \sqrt{(p_t(r) - m_i(r))^2 + (p_t(g) - m_i(g))^2} \leq R \\ 0 & \text{en otro caso} \end{cases} \quad (1)$$

La clasificación consiste en comparar cada píxel de la imagen actual con las N muestras del mismo que se encuentran almacenadas en m, y mediante un indicador determinar si es *background* o *foreground*. Luego, la siguiente función

$$Clasificar(p_t, m) = \begin{cases} Background & \text{si } \sum_{i=1}^{N} I(p_t, m_i) \geq \#Min \\ Foreground & \text{en otro caso} \end{cases} \quad (2)$$

define una máscara binaria de clasificación para cada píxel del *frame* de entrada, #Min es la cantidad mínima de veces que debe ser verdadera la Ec.(1) para considerarse *background*. Como consecuencia, por cada imagen del video, se genera una salida binaria correspondiente a la clasificación de cada píxel.

Luego de la detección, se actualiza el modelo en forma estocástica, para adaptar gradualmente la representación a los diferentes cambios que ocurren a lo largo de una secuencia. Es deseable que sólo aquellos píxeles clasificados como *background* se inserten en el modelo, ya que la inserción de píxeles que pertenecen a objetos en movimiento, o que resultan inciertos en la clasificación, pueden alterar significativamente los resultados de la detección. Esto último es muy importante ya que ha permitido la introducción de mejoras al método original logrando mayor tasa de detección. Entre ellas podemos citar, el módulo que no permite insertar en modelo píxeles que alternan permanentemente su clasificación entre *background* y *foreground* (*Blink Discard*), el módulo de postprocesamiento (*Extract Contour + Fill Object*) que corrige la clasificación de aquellos píxeles clasificados según Ec.(2) como *background*, pero que en realidad pertenecen al interior de objetos en movimiento. Además, se implementó una mejora al módulo de actualización del modelo (*Update*) para permitir o inhibir la propagación, de objetos *foreground* que se detienen, en el modelo de fondo. En este trabajo, se propone mejorar el tiempo computacional del algoritmo con sus módulos para su aplicación en áreas que requieren procesamiento en tiempo real como la vídeo-vigilancia.

La Figura 1 muestra el algoritmo y los módulos de mejoras mencionados. En las subsecciones siguientes se describen las etapas del algoritmo.

3.1 El modelo de fondo y la actualización estocástica

Para representar el modelo de fondo se mantiene una estructura donde para cada píxel se almacenan N muestras representativas del fondo (Figura 2). Al inicio, se debe adoptar alguna estrategia de inicialización rápida de esta estructura, por ejemplo utilizando sólo la información espacial del primer *frame*. Luego, la metodología de actualización del modelo del píxel que se está procesando, consiste en reemplazar el nuevo valor por una de las muestras que ya existía en su modelo, según una probabilidad definida como $1/\varphi$. Es decir, en promedio se realizará un reemplazo en el modelo cada φ píxeles que se corresponden con el fondo. La decisión sobre cuál de los N valores del modelo correspondiente a un píxel reemplazar es definido mediante un mecanismo aleatorio con distribución equi-probable. Por este efecto, el módulo de actualización del modelo es eficiente, ya que no debe reemplazar un valor específico ni realizar ningún procesamiento adicional de estimación sobre el modelo.

También se puede realizar una propagación espacial de cada píxel al modelo de alguno de sus vecinos con el mismo esquema probabilístico de reemplazo (se utiliza el mismo factor φ).

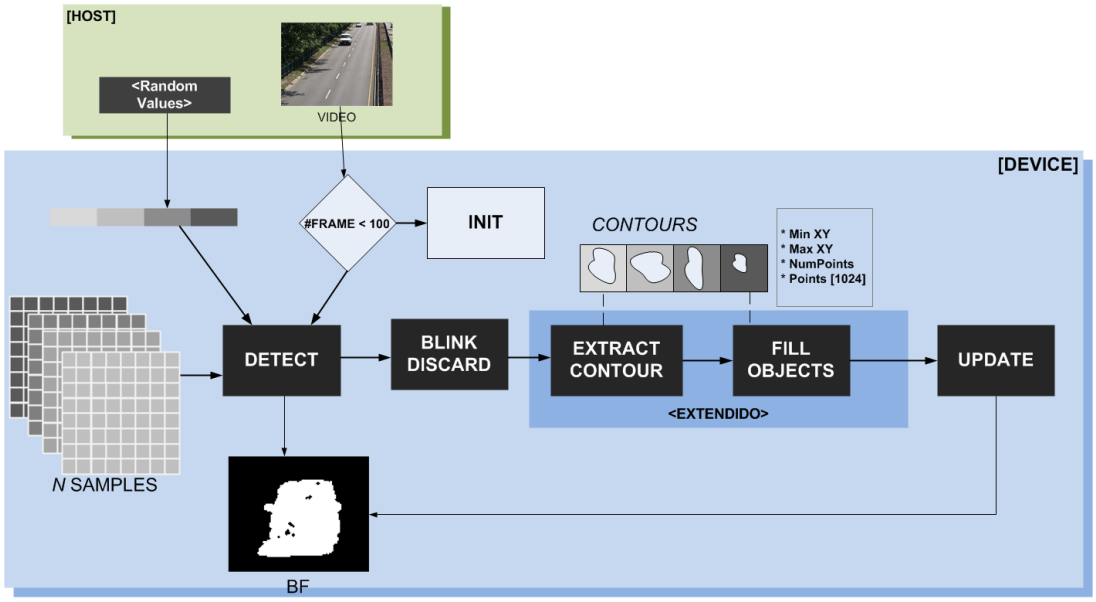


Figure 1: Etapas del proceso de detección del fondo.

Se considera vecino a cualquier píxel conectado en las ocho posibles direcciones respecto a su posición. En la Figura 2 se ejemplica el caso de actualización con el píxel actual a la cuarta muestra de su modelo y también la propagación del píxel actual que reemplaza a la tercera muestra del modelo de uno de sus 8 vecinos elegido al azar.

Es importante que en el modelo de fondo siempre se incluyan muestras que corresponden al *background* de la escena, para tener un modelo confiable y mejorar la detección durante el procesamiento de los *frames* siguientes. Sin embargo, para algunos píxeles resulta difícil identificar acertadamente si pertenecen al *background* o no. Por esta razón, se incluyen módulos de pos-procesamiento de la clasificación inicial para quedar sólo los píxeles *background* y *no blinking* como lo indica la (Figura 2).

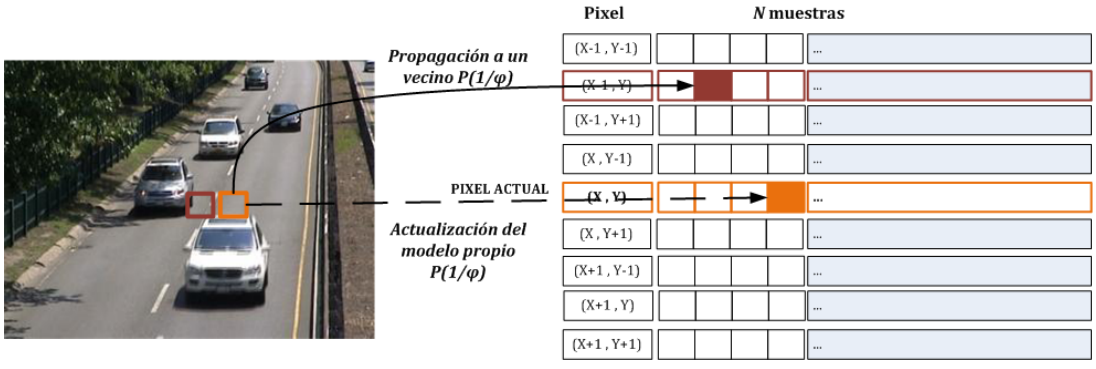


Figure 2: Modelo del fondo, actualización y propagación del píxel.

Se puede notar además que si se modifica la probabilidad φ de reemplazo, se puede acelerar o no, la incorporación de muestras al modelo. Esta estrategia se utilizó en la etapa inicial de inicializacíon de la estructura, aumentando la probabilidad de reeemplazo para incorporar rápidamente información en el modelo de los primeros *frames* procesados, también permitió una rápida absorción de los “fantasmas" iniciales que se producen por la falta de información inicial sobre el modelo de fondo, principalmente para aquellas zonas que tienen movimiento, lo

cual es uno de los problemas de la técnica original. En otros casos, se modificó la probabilidad de reemplazo para retrasar la incorporación en el modelo de píxeles que corresponden a objetos que están en movimiento, y se detienen por un período breve (continuando su condición de *foreground*), o por el contrario si se detiene por un período prolongado es necesario incorporarlo al modelo de fondo.

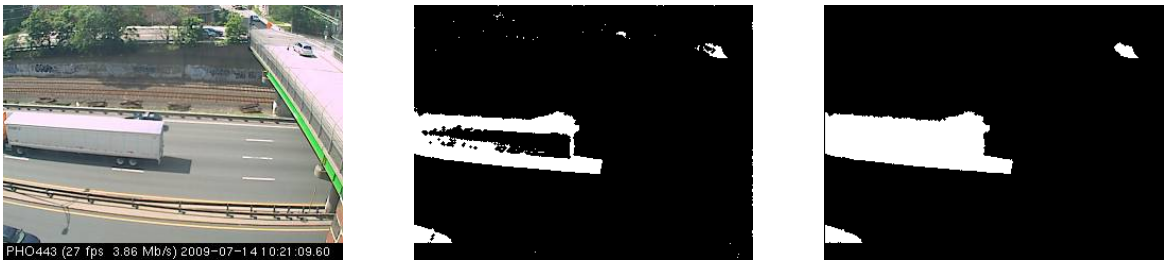
El hecho de que la permanencia de los distintos valores del modelo no sea ponderado según el tiempo es un factor decisivo en el mecanismo de actualización. De esta manera, se permite la persistencia de modelos que tengan un tiempo de vida relativamente mayores a otros. Resulta de utilidad en una gran diversidad de situaciones, principalmente cuando el fondo es frecuentemente superpuesto por objetos en movimiento (por ej. veredas en las que transitan continuamente personas, o calles en las que pasan muchos autos).

3.2 Modificación de Clasificación

En la heurística que estudia los objetos, se realizan modificaciones sobre la segmentación inicial producida según la Ec. Ec.(2). En primer lugar, se realiza la operación morfológica de *cierre* sobre esta salida binaria permitiendo que componentes que se encuentran muy cercanos entre sí se terminen de conectar en el caso de que no lo estén. Se efectúa utilizando un *kernel* de diámetro 3. Esto permite mejorar el proceso de llenado de agujeros posterior.

Por otra parte, resulta importante descartar posibles falsos positivos para proveer una mejora en el proceso de sustraer el fondo. Éstos existen frecuentemente en forma de objetos muy pequeños aislados de cualquier otro objeto. Por lo tanto, se realiza un filtrado de aquellos objetos que contengan un área menor o igual a 10. Dicho valor ha demostrado filtrar una gran cantidad de objetos detectados que se correspondían con el fondo, mientras que los objetos de interés que se encuentran en movimiento no los filtra debido a que, en la vasta mayoría de secuencias, su tamaño es mayor al valor dado para filtrarlo.

Figure 3: Efecto del módulo *Fill Update*



(a) Imagen original (b) Previo al módulo (c) Salida

Finalmente, se rellenan los contornos de los objetos que se detectaron como *foreground*. Resulta intuitivo analizar que si se detectó un contorno de un objeto como movimiento y, a su vez su interior se encuentra en su mayoría detectado como parte del objeto, pequeños sectores dentro del mismo que inicialmente hayan sido detectados como fondo se corresponderían con el movimiento del mismo objeto. En la Figura 3 se puede observar el efecto de aplicar la heurística.

Como consecuencia del análisis de objetos, algunos píxeles modifican su clasificación original de la Ec. 2, por el efecto del cierre o filtrado, generando una máscara *update*, por lo que sólo aquellos que queden luego de este proceso como *background* podrán actualizar el modelo de la Figura 2.

3.3 Descarte de píxeles en parpadeo

Frecuentemente la clasificación de un píxel de una imagen alterna entre *background* y *foreground* en cuadros continuos. Cuando un determinado píxel realiza este intercambio de estado en cuadros sucesivos, se lo considera que se encuentra *parpadeando*. Dicho caso se relaciona con la existencia de fondos multi-modales, el cual presenta para un determinado píxel la posibilidad de que existan múltiples fondos. En la práctica se guarda una estructura que indica el nivel de parpadeo de cada píxel. Si un determinado píxel pertenece al límite entre el objeto y el fondo, y además la clasificación para dicho píxel es diferente en los últimos dos *frames*, entonces se sospecha que puede ser un caso de parpadeo y se aumenta el nivel del píxel en 15. En caso contrario, el nivel se reduce en 1. El rango de trabajo de análisis aceptado se encuentra en [0..150]. Consideramos que si un píxel contiene un nivel de parpadeo mayor o igual a 30, se debe impedir que actualicen el modelo si presentan un parpadeo constante a lo largo de los cuadros más recientes. No ingresar píxeles clasificados como *foreground* o en parpadeo, mejora la detección en cuadros posteriores. Por ejemplo, ante una inminente estimación de pose puede que se realice un refinamiento de siluetas de los objetos, lo cual sirve para no actualizar en el modelo píxeles que no pertenecían al fondo. Los píxeles en parpadeo son descartados en la máscara *blinking* para no ser considerados como fondo en la actualización del modelo.

4 IMPLEMENTACIÓN DEL ALGORITMO EN GPU

Para que el algoritmo pueda funcionar en placas gráficas, utilizando la API de OpenCL, se deben considerar ciertos cambios al algoritmo original. En su mayoría, cada etapa del procesamiento es trivialmente paralelizable; es decir, cada píxel puede ser procesado de forma independiente, excepto por el algoritmo de detección del borde de objetos que describiremos luego. Cada una de las estructuras necesarias se corresponde con las indicadas en el proceso (*samples*, máscara *blinking*, máscara *update*) y se deben generar en memoria del dispositivo. Estas estructuras se alojan en memoria al inicio y sólo se actualizan por cada *frame* nuevo. Cada etapa del proceso se implementa en un *kernel* independiente, que se paraleliza a nivel de cada píxel y se ejecutan en forma secuencial. El único dato que se transfiere nuevamente al *host*, es la máscara resultante de *foreground*.

Por otra parte, la plataforma OpenCL no ofrece funciones para la generación de números aleatorios, necesarias en la etapa de actualización del modelo. Para resolver esta limitación, se utiliza una tabla pre-generada en CPU de números, que luego son accedidas pseudo-aleatoriamente por cada *thread*. Se utilizaron 2048 valores aleatorios, que consideramos suficientes para obtener un grado de aleatoriedad razonable.

Para el método de rellenado de objetos en GPU, se debieron implementar los siguientes métodos, cada uno en distintos *kernels* y con distinta granularidad.

En la Figura 4 se esquematiza la idea de la implementación en paralelo.

Reconstrucción del contorno: Se utilizó una implementación serial del algoritmo de *Moore-Neighbor Tracing* (Rajashekar Reddy et al., 2006). La imagen original se particiona en bloques de igual tamaño y se distribuye entre distintos *threads*. Cada *thread*, obtiene una lista de contornos o *blobs*, que luego son filtrados por área. Dado que no se conoce de antemano la cantidad y precisión de los contornos, se aloja memoria fija con un un número proporcional a la resolución de la imagen. Por ejemplo, para una imagen HD de 1080p, se distribuyen en 64 bloques, cada uno con 16 *blobs*, con 1024 puntos cada uno. A continuación, mostramos el pseudo-código de este método:

Dibujo de contornos: Se dibuja cada contorno continuo utilizando el algoritmo de *Brehnam*

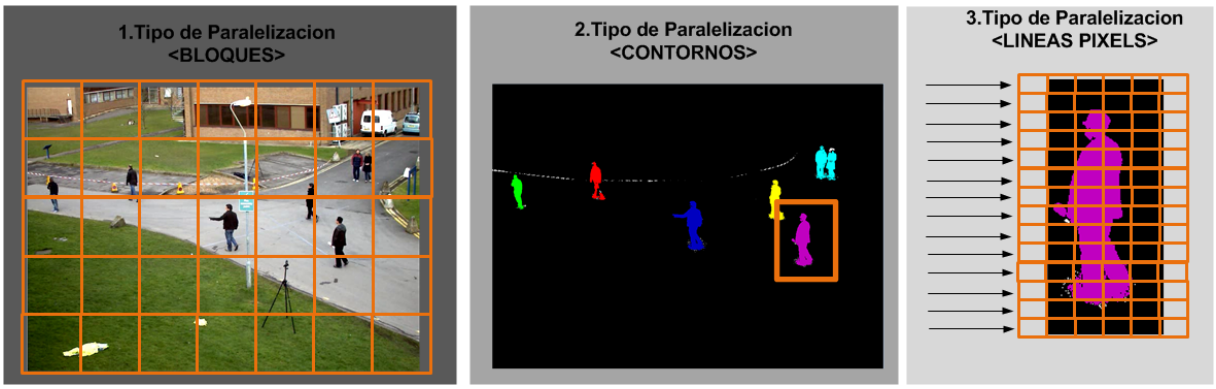


Figure 4: Etapas en paralelo del relleno de objetos. (izq.) Búsqueda de contornos (centro) Contornos detectados y delineados (der.) Distribución vertical de los rayos entre los *threads*.

Algoritmo de Moore-Neighbor Tracing por bloques

```
regions = split(image) // Divide la imagen en regiones cuadradas
parallel for eachregion r
        for( each pixel in r.bound)
        // Buscar un pixel WHITE
            if(image[pos] == WHITE ) // Nuevo pixel de borde
            // Recorrer en el vecindario
            while(true)
                pos = nextPos();
                if (image[pos] == WHITE )
                  storePoint(pos);
                if(checkPosition == startPos) break;
                mark[checkPosition] = WHITE; // Set pixel borde
        end for
end parallel for
```

(Rajashekar Reddy et al., 2006). Dado que los contornos son independientes, el método se realiza en paralelo por cada contorno. En cada píxel de la imagen se almacena el ID del *blob*, que será utilizado en el siguiente método de rellenado.

Rellenado del polígono por rasterización: Se implementó una variante del algoritmo del rayo, el cual computa los bordes encontrados, y supone que todo píxel que se encuentra entre dos píxeles bordes, es interno. El algoritmo se paraleliza con dos niveles: por contorno, y a su vez, por cada contorno, se distribuye por líneas de celdas en horizontal. Para asegurar que cada *thread* visite un único contorno, se verifica que el encontrado se corresponda con el ID del *blob*. De otra forma, si existen *blobs* coincidentes o uno interno a otro, fallaría la condición de rellenado.

En *Algoritmo de rellenado* mostramos el pseudo-código de este método.

5 RESULTADOS

En esta sección se presentan las métricas resultantes y los tiempos de cómputo necesarios para ejecutar el algoritmo sobre los distintos *data-sets* analizados. El conjunto de pruebas se

Algoritmo rellenado de la imagen

parallel for eachblob b
    if (b.numPoints==0) discard;
    // recorro el bounding box
    parallel for (each y in b)
        int $rmnx = \infty, rmxx = -\infty, ultBorde = 0$, rowx[64];
        for (each x in b) // si el pixel pertenece al contorno que visito
            if (input[ry*width + rx]==b.ID)
            rowx[ultBorde++] = rx;
        // Ordeno los bordes
        sort(rowx,ultBorde)
        // relleno las celdas como blancas, de a pares
        for (int i = 0; i <= ultBorde; i++)
            for (each rx between rowx[2*i] and rowx[2*i])
            output[ry*width + rx] = 255;
    end parallel for ry
end parallel for

realizó sobre una computadora portátil con un procesador AMD FX 6100 3.3 GHz y una placa gráfica NVIDIA GTX 550TI. La implementación fue realizada en C++ y OpenCL.

5.1 Evaluación de la precisión del algoritmo

Para realizar un estudio de la eficacia del algoritmo, se utilizaron los *data-sets* provistos por (Goyette et al., 2012), en la cual disponen del *ground-truth* para cada secuencia. Se eligieron tres secuencias bien características del contexto de video-vigilancia con 1800 *frames* cada una, denominadas *Highway*, *Street light* y *Parking*. Se utilizaron las métricas de *precision* y *recall*, ampliamente utilizadas en la sustracción de fondo. El resultado es mejor, cuanto más cercana a 1 es la métrica.

Se lo compara contra las técnicas de sustracción de fondo basadas en *Mixture of Gaussians* (MoG) implementadas en la funcionalidad de sustracción de fondo de la librería OpenCV. En la Tabla 1 se presentan los resultados de estas métricas.

Casos	Propuesto(Ext)		Vibe		GMM-Z		GMM-KTP	
	Pr	Rc	Pr	Rc	Pr	Rc	Pr	Rc
Highway	0.91 -	**0.97**	0.92 -	0.85	0.92 -	0.89	0.91 -	0.65
Street light	**0.99** -	**0.99**	**0.99** -	0.72	0.92 -	0.34	**0.99** -	0.23
Parking	0.91 -	0.52	0.82 -	0.25	0.74 -	**0.69**	0.66 -	0.37
Correlación	0.80255		0.6165		0.5291		0.3549	

Table 1: Valores de *Precision -Recall* por cada caso y para cada método

Se puede observar, que en la mayoría de los casos se mejora la métrica *recall* con el método propuesto, sin perjudicar la métrica *precision* y esto hace que el valor de correlación sea más alto. Incluso en secuencias complejas con objetos que se mueven, la inserción propuesta de objetos acelera la absorción de píxeles fantasmas, reduciendo efectivamente los falsos positivos; mientras que el algoritmo base que lo hace de forma lenta.

5.2 Evaluación de tiempos de cómputo

En la Tabla 2 se presentan las métricas de tiempo de procesamiento, medida en cuadros por segundo *(FPS)*. Se utilizó uno de los casos anteriores, el cual fue redimensionado en cuatro resoluciones distintas. En cualquiera de ellas, se mantiene la relación de eficacia. Se implementaron 3 versiones del algoritmo: la versión en CPU de múltiples núcleos, la versión en GPU y la versión extendida (con los módulos *Extract Contour+Fill Object*, *Blink Discard* y *Update Model* mejorados) también en GPU. Las dos primeras se utilizan para comparar con respecto al método ViBe original.

Para la comparación de rendimiento se utilizó la implementación de los dos algoritmos basados en *Mixture-of-Gaussians* disponibles en la biblioteca OpenCV. Se debe notar que para ambos se utilizó la misma GPU. También se utilizó el SDK de ViBe corriendo en CPU, el cual tiene una velocidad muy alta de procesamiento, y con funcionalidad equivalente a la versión normal de nuestra propuesta. En la Tabla 2 se muestran los resultados de procesamiento de un cuadro medido en *(FPS)*. Estos datos son los efectivos de procesamiento (incluyendo tanto el pasaje de datos como la sección de la ejecución misma).

Casos	GMM-Z	GMM-K	VIBE	Propuesto		
	GPU	GPU	CPU	CPU	GPU	GPU(ext)
576p	120	105	122	79	240	109
720p	92	89	75	51	160	55
1080p	48	52	37	27	91	35
1440p	25	27	18	13	47	23

Table 2: Frames por segundo procesadas en diferentes arquitecturas y con diferentes algoritmos

Se puede observar que todos los métodos presentan una buena cantidad de frames procesados por segundo, siendo de esta manera potenciales técnicas a utilizar para un sistema de tiempo real. Presentan una tasa de FPS altas incluso en videos de alta resolución.

6 CONCLUSIONES

Se presentó un algoritmo de sustracción de fondo multi-modelo que utiliza reemplazo estocástico. Se proveyeron distintas técnicas y variantes que permitieron aumentar la tasa de detección, basándose en ciertas características presentes en el contexto de la vídeo-vigilancia. Se realizó una implementación en paralelo utilizando OpenCL que permitió mejorar el rendimiento y lograr procesamiento en tiempo real en imágenes de alta resolución. Seguramente estos tiempos puedan mejorarse aún más, aprovechando los accesos a memoria local del dispositivo. Los bajos tiempos de cómputo permiten realizar la sustracción de una gran cantidad de cuadros por segundo, por lo que se puede llegar a utilizar en arquitecturas embebidas (como un ARM).

El algoritmo tiene algunas dificultades para discriminar objetos similares al fondo (problema denominado de camuflaje) por lo que se continuará estudiando y mejorando esta propuesta. Como comentario adicional, el código fuente de este algoritmo se comparte solicitándolo a los autores.

REFERENCES

Barnich O. and Van Droogenbroeck M. Vibe: a powerful random technique to estimate the background in video sequences. pages 945–948, 2009.

Barnich O. and Van Droogenbroeck M. Vibe: A universal background subtraction algorithm for video sequences. *Image Processing, IEEE Transactions on*, 20(6):1709–1724, 2011.

Benezeth Y., Jodoin P.M., Emile B., Laurent H., and Rosenberger C. Review and evaluation of commonly-implemented background sub. algorithms. pages 1–4, 2008.

Goyette N., Jodoin P.M., Porikli F., Konrad J., and Ishwar P. Changedetection. net: A new change detection benchmark dataset. pages 1–8, 2012.

KaewTraKulPong P. and Bowden R. An improved adaptive background mixture model for real-time tracking with shadow detection. pages 135–144, 2002.

Piccardi M. Background subtraction techniques: a review. 4:3099–3104, 2004.

Rajashekar Reddy P., Amarnadh V., and Bhaskar M. Evaluation of stopping criterion in contour tracing algorithms. *Int. J. of Computer Science and Inf. Technologies*, 3(3):3888–3894, 2006.

Schreiber D. and Rauter M. Gpu-based non-parametric background subtraction for a practical surveillance system . pages 870–877, 2009.

Vacavant A., Chateau T., Wilhelm A., and Lequièvre L. A benchmark dataset for outdoor foreground/background extraction. pages 291–300, 2013.

Van Droogenbroeck M. and Paquot O. Background subtraction: experiments and improvements for vibe. pages 32–37, 2012.

Yin Li Guijin Wang X.L. Three-level gpu accelerated gaussian mixture model for background subtraction. *Proc. SPIE 8295, Image Processing: Algorithms and Systems X; and Parallel Processing for Imaging Applications II*, 8295, 2012.

Zivkovic Z. Improved adaptive gaussian mixture model for background subtraction. 2:28–31, 2004.

Zivkovic Z. and van der Heijden F. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern recognition letters*, 27(7):773–780, 2006.