Algoritmo de visión estereo en tiempo real implementado en GPGPU

Andrés J. Demski[1], Andrés L. Di Donato[1], Santiago F. Maudet[1], Alejandro Furfaro[1]

Universidad Tecnológica Nacional, Facultad Regional Buenos Aires, Dpto. Ing. Electrónica, Laboratorio de Procesamiento Digital
dplab@electron.frba.utn.edu.ar,
http://www.electron.frba.utn.edu.ar/dplab

Abstract. En el este trabajo se presenta una implementación en tiempo real de un sistema de visión artificial en tres dimensiones sobre GPGPU. Se logra optimizar el procesamiento de visión estéreo mediante el algoritmo de Suma de Diferencias Absolutas (SAD), seleccionado en virtud de su costo computacional moderado. Se aplican sobre la imagen resultante sendos kernels de sustracción de fondo y suavizado, con el objeto de minimizar los errores de procesamiento intrínsecos de SAD, obteniéndose en forma experimental los parámetros óptimos para cada kernel. Se aplica esta cadena de procesamiento sobre una plataforma basada en GPGPU, y sobre un equipo basado en CPU convencional de arquitectura x86. Se comparan los resultados con el mismo procesamiento ejecutado en una CPU convencional y se demuestra una mejora 13x en el tiempo de ejecución, y por consecuencia, la factibilidad de aplicar el método para procesamiento de video en tiempo real utilizando GPGPUs como plataforma de ejecución, hecho que permite proyectar futuros trabajos en aplicaciones industriales de tiempo real que se valgan de la visión 3D para verificación de ensamble de productos sobre líneas de montaje.

Keywords: Visión Artificial - Reconocimiento de Patrones, GPGPU, visión stereo, SAD

1 Introducción

Los sistemas de procesamiento de imagen (SPI) extienden continuamente su aplicación en áreas cada vez mas diversas: entretenimiento, aplicaciones industriales, agropecuarias, visión en robótica, y salud, entre las mas destacadas. Este auge se debe al crecimiento sostenido de la capacidad de computo de los procesadores modernos, de la capacidad de los medios de almacenamiento, y de la resolución de las imágenes que pueden capturarse con las tecnologías actuales.
Si bien se han desarrollado una gran cantidad de técnicas para la detección de patrones, seguimiento y clasificación de objetos, existe en estos últimos años un interés creciente en determinar, no solo la posición del objeto en la imagen, sino

también la distancia a la cual están respecto de la cámara, los diferentes objetos que componen la escena. Esto implica obtener una descripción tridimensional del objeto y del ambiente en el que éste se encuentra, a partir de la información relevada con los sensores y las cámaras.
Para lograr esta referencia de profundidad se han planteado diversas soluciones, entre las cuales se encuentra el concepto de visión estero, cuya denominación se debe a que obtiene la componente de distancia entre el objeto y la cámara a partir de dos imágenes del mismo, tomadas desde ángulos diferentes. Este comportamiento se asemeja al de la visión humana en donde se integran las imágenes binoculares de ambos ojos. Así se obtiene la imagen tridimensional a partir de dos imágenes planas con sutiles diferencias entre sí. Dentro del concepto de visión estero, existen metodologías de aplicación que requieren una gran capacidad de procesamiento para obtener una resolución de distancia adecuada (como se puede ver en [1]), especialmente cuando la representación buscada se desea obtener en tiempo real. Por este motivo, las implementaciones que reúnen este tipo de requisitos se implementan sobre plataformas de hardware que cuentan con gran capacidad de paralelización. Tal es el caso de FPGA, SOCs y GPGPU. Algunas comparaciones entre las distintas implementaciones sobre estas plataformas están descriptos en [2]

La metodología mas implementada para implementar visión estero es la de suma de diferencias absolutas (SAD por Sum of Absolute Differences), que consiste básicamente en hacer la diferencia entre ambas imágenes y así obtener un mapa de costos, que no es otra cosa que una referencia en el plano de cuán iguales son las imágenes.

Una vez obtenido el mapa de costos se procede al calculo de la distancia entre un punto en particular y las cámaras.

Para implementar SAD hay una serie de derivaciones que resultan en mejores resultados a costa de mayor complejidad de implementación y procesamiento. Estas son Zero Mean Sum of Absolute Difference(ZSAD), Sum of Squared Difference(SSD), Zero Mean Sum of Squared Difference(ZSSD), Normalized Cross Correlation(NCC) Zero Mean Normalized Cross Correlation(ZNCC).

Como se mencionó anteriormente, este algoritmo permite obtener la distancia cámara / objeto. Sin embargo su costo computacional resulta elevado. Por lo también, si se pretende obtener resultados acordes a un frame rate de video en tiempo real, se requiere implementarlo sobre hardware que se adapte a las exigencias. En tal sentido las GPGPUs han demostrado poner en juego una capacidad de cómputo muy superior a la de las CPUs convencionales, fundamentalmente debido a que se han diseñado en base a clusteres de procesadores digitales de señal que tiene únicamente los recursos necesarios para procesar este tipo de algoritmos, y que al ser mas simples que una CPU de propósito general pueden

disponerse una cantidad muy alta de procesadores de señales en una GPGPU.

Este trabajo plantea un algoritmo de visión estéreo utilizando la técnica SAD,implementado en GPGPU con el objetivo de analizar una escena de video en tiempo real.Por otra parte el mismo procedimiento se aplicará a CPU, con el objetivo de obtener un análisis comparativo del rendimiento de cada una de las arquitecturas, y se cuantificará la ganancia que se tiene al implementarlo en GPGPU.

2 Materiales Y Metodología

2.1 Materiales

En esta sección detallaremos los recursos de hardware y software utilizados para el desarrollo del proyecto. La plataforma de hardware empleada tiene la siguiente configuración:

- CPU: Procesador Intel Core i7-3770
- Memoria RAM: 16GB
- GPGPU: NVIDIA GeForce GTX 670
  - CUDA Driver Version 6.5.
  - Runtime Version: 6.5.
  - CUDA Capability Major/Minor version number: 3.0
  - Global Memory: 2048 MB.
  - Multiprocessors: 7
  - CUDA Cores per Multiprocessor: 192
  - Warp Size: 32

Se utilizó un sistema operativo Linux Ubuntu x86_64 Kernel v3.11.0-26. Para manejo de archivos de imágenes, acceso a las cámaras, gestión de ventanas de muestra y demás operaciones auxiliares con imágenes se utilizó la biblioteca OpenCV versión 2.4.9.

El tolchain utilizado para los algoritmos de GPGPU se desarrollaron con CUDA ToolKit 6-5 provisto con el IDE Nsight versión 6.5. Para el resto se ha utilizado gcc 4.8.1, y OpenMP 3.1[5] para mediciones de performance.

El sistema de visión estéreo utilizado para adquirir las imágenes, fue implementado con dos cámaras web Genius Modelo FaceCam 1000 de resolución HD 720p,foco manual, ambas con puertos USB 2.0. Para emular una cámara estéreo se montó un soporte para fijar ambas a una distancia entre objetivos de 2cm.

2.2 Metodología

Concepto de SAD El algoritmo base implementado (SAD) requiere de una función costo que mide la similitud entre los píxeles de la imagen tomada con la cámara izquierda (IL) y los de la imagen tomada con la cámara derecha (IR).

Como principio básico del algoritmo SAD, es necesario medir la diferencia absoluta entre la intensidad de píxeles entre una imagen y la otra, como expresa la ecuación 1.

$$AD(x.y, d) = |I_L(x.y) - I_R(x + d, y)| \tag{1}$$

En la ecuación 1, d representa la disparidad o corrimiento de píxeles entre una imagen y la otra. Esta diferencia es función de la separación entre las cámaras y de la posición en profundidad de los objetos en cuestión. De este modo, para cada píxel (x,y) se obtendrá un valor de disparidad d, tal que minimice el costo AD. Para tal fin, proponemos un algoritmo de decisión del tipo "winner-take-all", que genere un valor de d tal que aplicado en ecuación 1, permita calcular el valor mínimo posible del costo AD.

Sin embargo, trabajar píxel a píxel, no provee el resultado óptimo, ya que producto de la adquisición de la señal original es factible tener ruido de origen que es necesario neutralizar. Para obtener una mejor relación señal a ruido, la solución propuesta es en lugar de comparar píxeles de a pares, comprar áreas de píxeles en cada imagen, por estar suficientemente probado su efecto de suavizado respecto del ruido, al promediar el efecto individual de cada píxel con el de sus vecinos. De este modo, se obtiene un costo acumulado, asociado a una ventana de píxeles, como expresa la ecuación 2.

$$SAD(x.y, d) = \sum_{(x,y)\in W} |I_L(x.y) - I_R(x + d, y)| \tag{2}$$

SAD es un método de cálculo de profundidad basado en la comparación de un área. Como expresa la ecuación 2, se considera una ventana alrededor de un determinado píxel, y se busca la correspondencia buscando en la otra imagen la ventana de igual tamaño cuya similitud sea mayor, o lo que es lo mismo, donde la disparidad encuentre su mínimo (winner-take-all), considerando siempre una disparidad máxima admisible. Este valor de disparidad máxima estará condicionado por el montaje de las cámaras, así como por el tamaño de los objetos que se deseen caracterizar. Si la disparidad máxima es baja, entonces se contará con pocos niveles de profundidad distinguibles, situación que producirá resultados pobres, dado que los saltos de profundidad serán demasiado grandes entre nivel y nivel. Por otra parte, si se analizaran objetos que por su posición tuvieran un gradiente de profundidad en las imágenes, se observarán variaciones grandes y con poco detalle. Por el contrario, elegir una disparidad máxima excesivamente alta (y por lo tanto, muchos niveles de profundidad) provocará resultados muy ruidosos y consecuentes errores en el resultado final.

El SAD es un método muy interesante para implementar utilizando GPGPU, dado que emplea un algoritmo simple, paralelizable y con bajos requerimientos de memoria, hecho este último que lo hace especialmente atractivo ya que la capacidad de memoria, resulta habitualmente un condicionante en las implementaciones sobre esta plataforma.

Sustracción de fondo Nos referimos al fondo de la imagen como el conjunto de píxeles que no aportan información que permita determinar su posición en profundidad a partir de sus píxeles vecinos. Por ello el SAD no sacará provecho de procesar los mismos, y por el contrario, es posible que termine aportando ruido, hecho que obviamente resultará contraproducente en el resultado final. Por lo tanto, utilizamos un algoritmo de sustracción de fondo para limpiar la imágenes tomadas por ambas cámaras, y evitar efectos de degradación no deseados .

Para estimar el fondo de la imagen, hemos utilizado un filtro pasa bajos, que permite eliminar los detalles para luego, con una simple diferencia entre la imagen original y la filtrada, obtener los píxeles con información de la imagen, eliminando la información de fondo.

Suavizado SAD suele experimentar un aumento del error en los bordes de las imágenes, que aumentan cuanto mas abruptos son dichos bordes. Por tal motivo, una vez aplicado SAD, se aplica un algoritmos de suavizado a la imagen resultante, con el objeto de enmascarar algunos de estos errores.

Como metodología de suavizado se utiliza un filtro de media en dos dimensiones. En este tipo de filtrado, cada píxel se reemplaza por la media de sí mismo y sus vecinos. Se trata de una convolución discreta en dos dimensiones a coeficientes iguales para todos los píxeles. Cuando más píxeles vecinos se incluyan en el promedio, más agresivo será el suavizado, pero en contrapartida se pierde nitidez en la imagen original, de modo que seleccionar la cantidad de píxeles vecinos a incluir en el algoritmo de suavizado no es una decisión menor.
Para la elección del tamaño del kernel de suavizado se realizaron pruebas preliminares iniciando con el el mínimo admisible por el método: tres píxeles. Al ensayar los kernels de tres y cinco píxeles de radio, se obtuvieron errores muy notorios en los contornos de la imagen. Por otra parte, para kernels que utilizaron mas de nueve píxeles de radio, los resultados finales resultaron demasiado borrosos, perdiéndose detalles significativos para la imagen. En base a estos resultados preliminares hemos adoptado siete píxeles como radio ideal del kernel para el filtro de suavizado.
El filtro de media puede implementarse utilizando convolución separable [6], que sugiere reemplazar la convolución en dos dimensiones por una convolución fila seguida por una convolución columna. Dado que todos los elementos de la matriz de convolución utilizada en el filtro de media son iguales, existirá una gran redundancia en los productos y acumulaciones al realizar la convolución en dos dimensiones, razón que justifica la aplicación de la técnica de convolución separable, cuyo resultado permitirá mejorar de manera notable la eficiencia la

implementación del suavizado, evitando que este post-procesamiento empeore el rendimiento general de la implementación de la visión 3D.

3 Resultados y Discusión

3.1 Validación del algoritmo

Para validar la efectividad del algoritmo exclusivamente, es decir, sin involucrar el montaje de cámaras para tomar fotografías, lo hemos testeado empleando imágenes estándar utilizadas en visión 3D[7]. Esto nos permitió realizar el ajuste de los parámetros propios de la implementación para obtener los mejores resultados, sin involucrar la adquisición de las imágenes ni las cámaras empleadas.

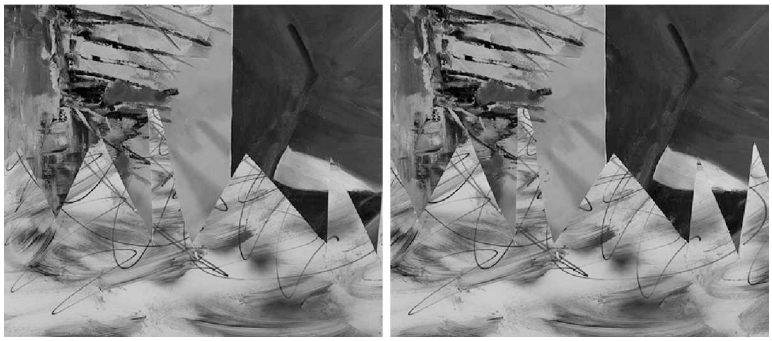

Fig. 1. Imágenes patrón izquierda y derecha respectivamente

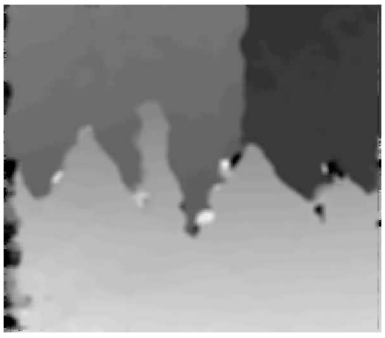

Fig. 2. Mapa de disparidad obtenido para las imágenes patrón

3.2 Validación del sistema

Las imágenes mostradas en la figura 3 han sido tomadas con el montaje de cámaras. Posteriormente, se logró construir el mapa de disparidad mostrado en la figura 4 a partir de dichas imágenes.

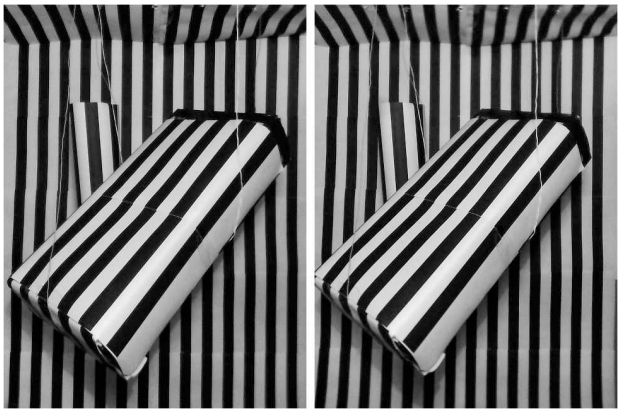

Fig. 3. Imágenes capturadas por el sistema

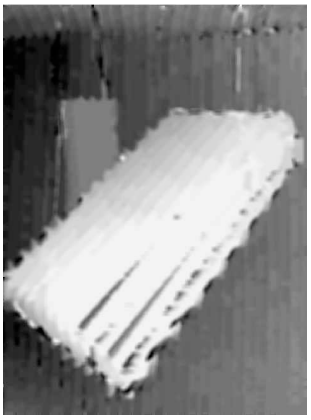

Fig. 4. Mapa de disparidad obtenido para las imágenes anteriores

3.3 Comparación de rendimiento de CPU y GPU

A continuación se presenta el tiempo medio necesario para obtener los resultados anteriores en ambas plataformas: Analizando una imagen de $640x480$, con un kernel de costo $11x11$, kernel de sustracción de fondo de $21x21$ y disparidad máxima de 21, utilizando CPU.
Luego de varias pruebas, el tiempo de ejecución convergió a 0.515 segundos, valor que corresponde a menos de 2 cuadros por segundo
Bajo las mismas condiciones anteriormente descriptas pero utilizando GPGPU, se obtuvo un resultado considerablemente mejor, tardando en procesar la totalidad de la imagen 0.039 segundos (25 cuadros por segundo).

4 Conclusiones

4.1 Implementación en GPU vs. CPU

Al contrastar los resultados obtenidos en ambos casos, observamos que el tiempo necesario de procesamiento con GPU es 13 veces menor al que se obtendría por la ejecución en la CPU detallada anteriormente, siendo los tiempos 0.039 y 0.515 respectivamente. Esta mejora del rendimiento justifica la utilidad de elegir esta implementación, y a su vez, permite una velocidad de trabajo de hasta 25 cuadros por segundo, impensados para la ejecución del SAD en CPU. De este modo en aplicaciones de video en real time, el uso de GPGPU abre un campo de aplicación para algoritmos SAD.

4.2 Posibles mejoras y trabajo futuro

En base a estos auspiciosos resultados, nos proponemos continuar trabajando en aplicaciones con SAD con el objeto de mejora aún mas estos resultados. En tal sentido, nos proponemos emplear filtrado bilateral y correlación asistida por detección de bordes [8]. Estas mejoras permitirán la detección de detalles con mejor precisión y obtener información de profundidad de elementos mucho más pequeños, hechos que nos permitirán utilizar el sistema en aplicaciones industriales. Una aplicación directa es la verificación de montaje de circuitos impresos, verificando que todos los componentes de un PCB estén presentes en el montaje final y estén además ocupando una posición adecuada. La velocidad obtenida en la implementación resulta interesante para la implementación automatizada de este sistema de validación en tiempo real sobre una línea de montaje que produzca circuitos impresos en serie.

5 Bibliografía

1. *Navab, N., Unger, C.* - "Stereo Matching" - Technische Universität München
2. *Lazaros, N., Sirakoulis, G., Gasteratos, A.* - "Review of stereo vision algorithms: from software to hardware" - International Journal of Optomechatronics
3. Open Source Computer Vision - http://docs.opencv.org/
4. "NVIDIA CUDA C Programming Guide" - NVIDIA - 2012
5. "OpenMP API specification for parallel programming"- http://openmp.org
6. *Podlozhnyuk, V.* - "Image Convolution with CUDA" - NVIDIA - 2007
7. "Middlebury Stereo Datasets" - http://vision.middlebury.edu/stereo/data/
8. *Ansar, A., Huertas, A., Matthies, L., Goldberg,S.* - "Enhancement of Stereo at Range Discontinuities" - California Institute of Technology