García Blas, J., Brox, P., Carretero, J., Desco, M., Abella, M. (25-27 de noviembre, 2020). *Herramienta de pegado de múltiples camas para tomografía computarizada en 3D mediante dispositivos GPU* [Comunicación en Congreso]. XXXVIII Congreso Anual de la Sociedad Española de Ingeniería Biomédica, CASEIB 2020. Recuperado de http://caseib.es/2020/wp-content/uploads/2020/12/CASEIB2020_LibroActas.pdf

Herramienta de pegado de múltiples camas para tomografía computarizada en 3D mediante dispositivos GPU

Javier Garcia Blas[1], Pablo Brox[1], Jesus Carretero[1], Manuel Desco[2,3,4,5] and Monica Abella[2,3,5]

[1] *Departamento de Informática, Universidad Carlos III de Madrid, Madrid, España*

[2] *Instituto de Investigación Sanitaria Gregorio Marañón (IiSGM), Madrid, España*

[3] *Departamento de Bioingeniería e Ingeniería Aeroespacial, Universidad Carlos III, Madrid, España*

[4] *Centro de Investigación en Red de Salud Mental (CIBERSAM, CIBER CB07/09/0031), 28007 Madrid, España*

[5] *Centro Nacional de Investigaciones Cardiovasculares Carlos III (CNIC), Madrid, España*

Resumen

En sistemas de tomografía axial computarizada (TAC) es común hacer varias adquisiciones tomográficas consecutivas para distintas posiciones de la cama, que posteriormente se combinan para ampliar el campo de visión en la dirección longitudinal. Para esta combinación es necesaria la calibración geométrica del movimiento de la cama para evitar dobles bordes en la zona de pegado. Esta calibración se realiza de forma periódica usando un maniquí específico. Este trabajo presenta una herramienta novedosa de pegado de cama automático para TAC basada en correlación. Nuestra propuesta explota el paralelismo masivo que ofrecen las GPUs y cuenta con un modelo de memoria optimizado que permite pegar grandes volúmenes en tiempo casi real. La evaluación en estudios de roedor demuestra, no solo que la implementación ofrecida es capaz de pegar los estudios tomográficos en un tiempo reducido, sino también que hace un uso eficiente de los recursos de memoria disponibles.

1. Motivación

Con los nuevos avances en el campo de la imagen médica, la calidad de los detectores en tomografía computarizada ha aumentado significativamente al obtener un mayor número de proyecciones por estudio. Esta mejora se traduce en una mayor resolución y cantidad de datos para procesar [3], exigiendo cada vez más potencia de computación. Este desafío, combinado con la necesidad de obtener los resultados en poco tiempo [4] motiva la necesidad de aceleradores para mejorar los cálculos. El enfoque más empleado es el uso de dispositivos de GPU, que ofrecen un buen equilibrio entre rendimiento y uso. Además, se pueden acoplar al hardware existente con poco esfuerzo.

La tomografía axial computarizada (TAC) es uno de los procedimientos de diagnóstico y evaluación de imágenes médicas más importantes. Se basa en la combinación de múltiples imágenes de rayos X tomadas desde diferentes ángulos alrededor del objeto para obtener un volumen tridimensional [1]. Cuando el campo de visión es insuficiente para abarcar la estructura anatómica de interés, es necesario generar múltiples volúmenes (camas) solapados, que se concatenan una sola salida. Este paso de concatenación se denomina pegado de cama e implica la fusión de la parte común de las dos camas, que denominamos "zona de solape". En este sentido, en [2] se introdujo un software de reconstrucción basado en FDK que incluye pegado de camas basado en una calibración previa.

Hay dos contribuciones principales presentadas en este trabajo. La primera es portar una aplicación de pegado automático de múltiples camas, proporcionando una nueva implementación paralela de los algoritmos incluidos utilizando dispositivos GPU. Se presenta un enfoque basado en módulos, que incluye un nuevo modelo de memoria que reduce significativamente los recursos necesarios, lo que permite el pegado de mayores volúmenes en menos tiempo. Además, el diseño elegido abre la puerta a una mayor implementación de nuevos algoritmos y modos de operación. El segundo es integrar esta aplicación en un software de reconstrucción basado en FDK [6].

Las principales contribuciones de este trabajo están relacionadas con la mejora e integración del prototipo dado y pueden dividirse en tres. Primero, identificar los diferentes procesos que conforman el prototipo y estructurarlos en diferentes módulos, facilitando la adición de nuevos algoritmos o modos de ejecución. Segundo, redefinir el modelo de memoria de la aplicación reduciendo la memoria consumida por los algoritmos implementados. Por último, presentar la implementación y evaluación de una versión de la aplicación de pegado de camas que aproveche la potencia intrínseca de los dispositivos GPU para optimizar los cuellos de botella, reduciendo el tiempo global de ejecución y reduciendo la posibilidad de introducir errores de forma manual.

2. Método propuesto

Para obtener un volumen final una vez solapadas cada una de las camas, tenemos que abordar principalmente dos obstáculos: los cortes superpuestos y la desviación de ángulo entre camas consecutivas. En la Figura 1 muestra un ejemplo de dos camas no alineadas del mismo volumen, el área sombreada representa los cortes que aparecen en ambos sub-volúmenes.

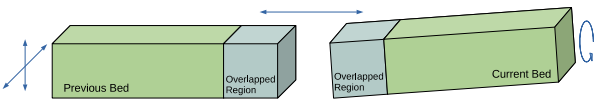

Figura 1.*Diferentes camas tomadas del mismo sujeto.*

ISBN: 978-84-09-25491-0, pags. 85- 88

Las etapas principales del método son dos. Primero, a través de tres pasos de correlación entre camas adyacentes se caracterizan los parámetros de registro: (i) desplazamiento axial (eje z), que corresponde con el corte de comienzo de la porción superpuesta en la cama actual; (ii) rotación alrededor del eje axial; (iii) desplazamiento coronal y (iv) desplazamientos sagital. Segundo, la fusión de los sub-volúmenes o camas.

El flujo de ejecución se presenta en la Figura 2. El proceso se divide en cuatro submódulos.

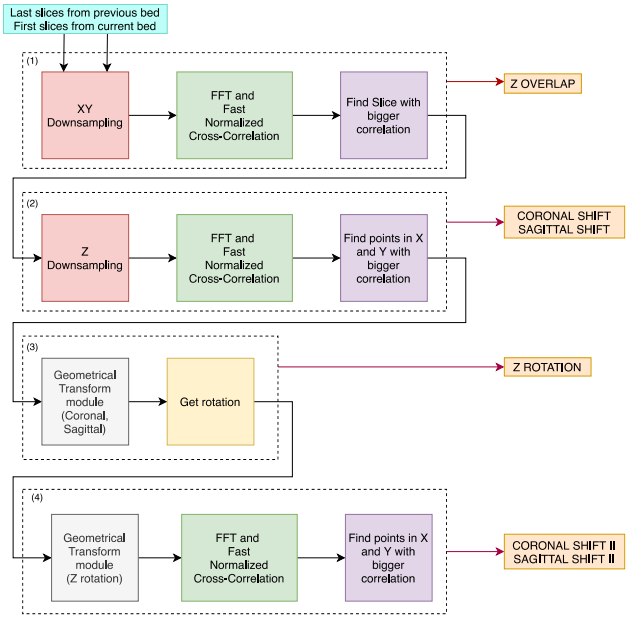

Figura 2. *Flujo de ejecución de pegado automático de camas.*

En el Módulo 1, a través de una primera correlación se caracteriza el desplazamiento en el eje Z, que se corresponde con el comienzo de la zona de solape con la cama anterior. Para poder acelerar el proceso se permite realizar un submuestro en el plano XY. Una vez aplicado dicho desplazamiento, el Módulo 2 busca mediante una segunda correlación el desplazamiento en los ejes X e Y. Una vez más, se puede aplicar un submuestro en el eje Z para acelerar este proceso. Estos desplazamientos se aplican mediante una transformación geométrica, para en un siguiente paso, calcular el ángulo de rotación entre camas. Finalmente, en el Módulo 4 se refinan los valores de desplazamiento en X e Y una vez aplicada la rotación anterior.

En la última etapa, se realiza una transformación geométrica a la cama actual con los parámetros calculados anteriormente. Para ello, se hace la fusión de la parte común de las dos camas tomando el valor del vóxel que sea máximo entre ambas camas.

La escritura al archivo de salida se hace de forma gradual. Al principio, el archivo de salida sólo contiene el primer volumen completo, a continuación, al final de cada iteración, el sistema coloca el puntero del archivo en el sub-volumen donde comienza la superposición y escribe la correspondiente cama con la zona de solape corregida.

3. Optimizaciones aplicadas

El objetivo de este trabajo no sólo es ofrecer una nueva implementación del prototipo, más fácil de usar e integrable con otros sistemas de reconstrucción, sino también disminuir el tiempo de ejecución y el consumo de memoria. Como estamos trabajando principalmente con volúmenes gráficos, la computación en GPU nos permite paralelizar los algoritmos usados aliviando los cuellos de botella con una baja sobrecarga.

La solución seleccionada para interactuar con la GPU es NVIDIA CUDA. No sólo por su flexibilidad y facilidad de uso, sino también porque proporciona algunas librerías útiles (cuFFT [7] o NPP [6]) que implementan algoritmos útiles para el sistema, como se verá en las siguientes secciones.

3.1. Fast Normalized Cross-Correlation

El algoritmo de Correlación Cruzada Rápida Normalizada introducido por Lewis [9] requiere que sus entradas estén en el dominio de la frecuencia, y para lograrlo es necesario aplicar varias FFT (en ambas direcciones) a los volúmenes. El proceso de implementación de FFT en la GPU es complejo, sin embargo, CUDA ofrece la librería cuFFT que proporciona funciones para realizar estas transformaciones dentro de la GPU, reduciendo la complejidad del código.

A pesar de la mejora en el tiempo de computación obtenido con esta librería, el algoritmo que aplica la FFT consume mucha memoria. Como la memoria de los dispositivos GPU es inferior a la memoria principal del computador, se ha implementado un mecanismo de control que comprueba la memoria disponible antes de llamar a la librería y transfiere a la memoria principal los resultados anteriores en caso de que no haya suficiente espacio.

3.2. Aceleración de transformaciones geométricas

A pesar del esquema de paralelización aplicado al procedimiento de interpolación, los accesos se realizan a vóxeles vecinos, realizando un uso ineficiente de la caché. Los accesos a la memoria de texturas son a través de una caché de sólo lectura que proporciona localidad espacial, por lo que los vóxeles se almacenan cerca de sus vecinos en lugar de en posiciones consecutivas en memoria.

Para solventar los anteriores problemas, hemos empleado la librería de primitivas de rendimiento de NVIDIA, que proporciona funciones de procesamiento de imágenes y señales aceleradas por la GPU e interpola utilizando la memoria de texturas. Esta solución permite obtener los valores interpolados directamente del hardware del dispositivo, evitando su cálculo.

3.3. Gestión de memoria

Con el objetivo de reducir el consumo de memoria del sistema, se aplicaron varios pasos de submuestreo de los volúmenes en el módulo de parámetros de pegado. Para ello, utilizamos la librería de primitivas de rendimiento de NVIDIA, que aceleran el proceso, aprovechando la potencia de GPU. Estas transformaciones mejoran no sólo el uso de la memoria sino también la velocidad del sistema.

4. Evaluación experimental

El hardware utilizado para llevar a cabo los experimentos consiste en un ordenador personal equipado con un procesador Intel Core i7-7700, 32 GB de RAM y una GPU NVIDIA GeForce GTX TITAN Black.

En este trabajo comparamos el rendimiento del prototipo inicial basado en MatLab con la versión del sistema implementada en lenguaje C. Un prototipo del sistema planteado en este trabajo se ha integrado en RAPTOR, un algoritmo basado en FDK [5].

Para ello, ambas versiones se ejecutan con dos estudios de roedor descritos en la Tabla 1.

Estudio	Camas	Dimensiones (vóxeles)	Resultado (vóxeles)
1	3	512×512×572	512×512×1170
2	2	516×516×574	516x516×970

Tabla 1. *Descripción de los conjuntos de datos empleados.*

4.1. Consumo de memoria

Para cada ejecución, se midió el consumo de memoria y el tiempo de ejecución. Las Figuras 3 y 4 muestran los resultados del experimento para cada conjunto de datos para la versión en Matlab y la propuesta en este trabajo.

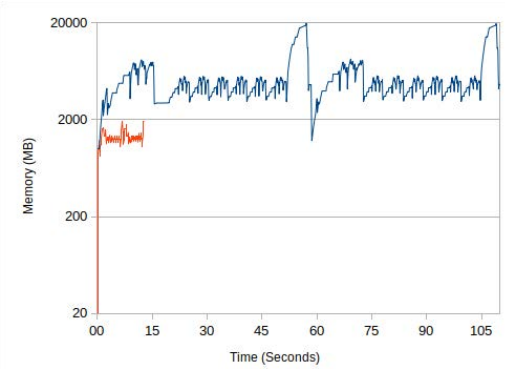

Figura 3. *Tiempo de ejecución en comparación del consumo de memoria del Estudio 1 (3 camas). El color azul corresponde a la versión basada en Matlab y el naranja a la implementada para este trabajo.*

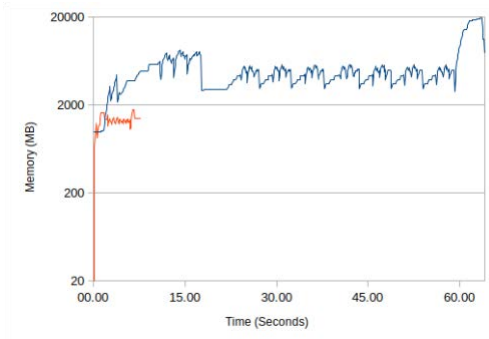

Figura 4. *Tiempo de ejecución en comparación del consumo de memoria del Estudio 2 (2 camas). El color azul corresponde a la versión basada en Matlab y el naranja a la implementada para este trabajo.*

Podemos observar que la versión de MatLab utiliza una media de 4 veces más memoria RAM que la versión propuesta para ambos estudios, comportamiento que es reproducible durante toda la ejecución. Además, en los experimentos hemos observado picos de consumo de memoria a lo largo de la ejecución. Esos picos son más significativos para la versión MatLab y aparecen al final del proceso de cada cama (el primer estudio tiene tres camas, y por lo tanto, dos regiones superpuestas e iteraciones) indican que un proceso está mal optimizado.

En términos generales, se emplea en promedio un 23,5% del total de la memoria RAM disponible en la versión de MatLab y el porcentaje cae a un 9,3% cuando se observa el pico de consumo de memoria. En cuanto al tiempo de ejecución, la solución propuesta obtiene una aceleración de 6,85× para el primer estudio y 8,18× para el segundo, resultando en un promedio de aceleración de 7,51×.

4.2. Tiempo de ejecución

El propósito de este experimento es determinar esa sobrecarga.

Estudio	Pegado	Tiempo de ejecución (seg)
1	Manual	20,020
	Auto	26.101
2	Manual	17,540
	Auto	19,645

Tabla 2. *Comparativa del tiempo de ejecución en modos manual y automático.*

Observamos en la Tabla 2 que el pegado de camas automático presenta una sobrecarga de 6,08 segundos para el primer estudio (30% más lento) y 2,105 segundos para el segundo (12% más lento). La principal diferencia entre el primer y el segundo conjunto de datos es el número de camas, lo que afecta al número de iteraciones realizadas en la fase de pegado automático y explica la diferencia porcentual. En promedio, la integración del mecanismo automático añade un tiempo de sobrecarga del 21% frente al tiempo total de ejecución.

5. Conclusiones y discusión

En este trabajo hemos presentado un nuevo sistema de pegado automático de camas que aprovecha la capacidad de cómputo de los dispositivos GPU. La versión inicial de la solución implementada en MatLab hacía inviable la puesta en producción de dicho método. A pesar de contar con un alto número de optimizaciones, MatLab carece de mecanismos de alto rendimiento para el procesamiento de transformadas de Fourier y el submuestreo de volúmenes. El prototipo presentando en este trabajo acelera de forma significativa este proceso mediante es uso de bibliotecas especializas adaptadas a la plataforma hardware, exprimiendo todo su potencial de cómputo.

La evaluación preliminar demuestra que la implementación propuesta no sólo es hasta 8 veces más rápida que la versión implementada en Matlab, sino que su consumo de memoria es un 76,5 % menor en promedio y

no presenta picos notables. Además, la integración con software de reconstrucción implica una sobrecarga aceptable (21% en promedio) en comparación con el ahorro de tiempo, esfuerzo y la eliminación de artefactos producidos por un pegado basado en una etapa previa de calibración.

El pico de mayor consumo de memoria de la aplicación se alcanza dentro de la llamada de la biblioteca de cuFFT, lo que hace que la aplicación pueda tener dificultades a la hora de procesar volúmenes de gran tamaño. Una posible solución es gestionar el consumo de memoria dentro de la biblioteca mediante el particionamiento de cada una de las camas en varios subvolúmenes.

En promedio, la integración del mecanismo automático añade un tiempo de sobrecarga del 21% frente al tiempo total de ejecución. No obstante, elimina la calibración geométrica necesaria en un esquema convencional, que debe realizarse con mucho cuidado para obtener resultados válidos.

En este trabajo no se ha considerado el empleo de más de un dispositivo de GPU a la vez. Implementar un enfoque de multi-GPU que permita aprovechar todos los dispositivos instalados redundará en una disminución del tiempo de ejecución.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia, Innovación y Universidades (Instituto de Salud Carlos III, proyecto DTS17/00122; Agencia Estatal de Investigación, proyecto DPI2016-79075-R-AEI/FEDER, UE), cofinanciado por Fondos de la Unión Europea (FEDER), "A way of making Europe". Además, ha sido financiado por el Programa de apoyo a la realización de proyectos interdisciplinares de I+D para jóvenes investigadores de la Universidad Carlos III de Madrid 2019-2020 en el marco del Convenio Plurianual Comunidad de Madrid- Universidad Carlos III de Madrid (proyecto DEEPCT-CM-UC3M) y por CRUE Universidades, CSIC y el Banco Santander (Fondo Supera, proyecto RADCOV19). El CNIC está financiado por el Ministerio de Ciencia, Innovación y Universidades y la fundación PRO-CNIC y es un centro de excelencia Severo Ochoa (SEV-2015-0505).

Referencias

[1] T. B. R. Company. (Jul. 2018). Veterinary industry outlook: Top global trends and statistics, [Online]. Available: https://blog.marketresearch.com/veterinary-industry-outlook-top-global-trends-and-statistics.

[2] M. Abella et al., "Software architecture for multi-bed fdk-based reconstruction in x-ray ct scanners", Computer methods and programs in biomedicine, vol. 107, no. 2, pp. 218–232, 2012.

[3] N. De Briyne and D. Iatridou, "Challenges seen with treatment of exotic pets in veterinary practice exotic pets in veterinary practice", Dec. 2016.

[4] H. Turbell, "Cone-beam reconstruction using filtered backprojection", PhD thesis, Linköping University Electronic Press, 2001.

[5] J. G. Blas, M. Abella, F. Isaila, J. Carretero, and M. Desco, "Surfing the optimization space of a multiple-gpu parallel implementation of a x-ray tomography reconstruction algorithm", Journal of Systems and Software, vol. 95, pp. 166–175, 2014.

[6] J. Ghorpade, J. Parande, M. Kulkarni, and A. Bawaskar, "GPGPU processing in CUDA architecture", arXiv preprint arXiv:1202.4347, 2012.

[7] CUDA Toolkit 4.2 CUFFT Library programming guide, NVIDIA Corporation, Mar. 2012. [On- line]. Available: https://developer.download.nvidia.com/compute/DevZone/docs/ html/CUDALibraries/doc/CUFFT_Library.pdf.

[8] NVIDIA 2D Image and Signal Performance Primitives (NPP), [Online]. https://docs.nvidia.com/cuda/npp/index.html.

[9] J. Lewis, "Fast normalized cross-correlation, 1995", in *Vision Interface*, vol. 2010, 2010, pp. 120–123.