3D GrabCut: Una segmentación de volúmenes basada en la técnica GrabCut utilizando la GPU

Pablo Temoche[1], Esmitt Ramírez[1], Rhadamés Carmona[1]

[1]Centro de Computación Gráfica, Universidad Central de Venezuela. {pablo.temoche | esmitt.ramirez | rhadames.carmona}@ciens.ucv.ve

RESUMEN

La segmentación de imágenes consiste en obtener una región de interés dentro de un espacio más amplio. GrabCut es una técnica reciente de segmentación 2D que obtiene excelentes resultados. Esta se basa en representar la imagen como un grafo de flujo, y luego aplicar un algoritmo de corte mínimo en el grafo y separar la imagen de fondo (background) y la región de interés (foreground). Este artículo presenta un enfoque novedoso para segmentar volúmenes basado en GrabCut empleando un esquema de algoritmo paralelo que se ejecuta en la GPU denominado 3D GrabCut. El grafo es creado completamente en la GPU y se ejecuta sobre la arquitectura CUDA de NVIDIA®. Las pruebas demuestran excelentes resultados con los volúmenes de prueba, arrojando tiempos inferiores que su ejecución en la CPU y una separación adecuada background/foreground.

ABSTRACT

The image segmentation consists in obtaining a region of interest within a larger area. GrabCut is a recent technique of 2D segmentation which presents excellent results. This is based on representing the image as a flow network, and then apply a minimum cut algorithm on the graph and separate the background image (background) and the region of interest (foreground). This article presents a novel volume segmentation approach based on GrabCut. It implements a parallel algorithm on the GPU called 3D GrabCut. The graph is created entirely on the GPU and runs on NVIDIA's CUDA architecture®. Tests show excellent results with test volumes, reducing the computing time with an adequate separation background/foreground.

Palabras claves: segmentación de volúmenes, GPU, GrabCut, *max-flow/min-cut*.

1. Introducción

El proceso de extraer ciertas zonas de interés de una imagen, se conoce como segmentación. La segmentación de imágenes consiste en separar una imagen en al menos dos regiones con características diferentes. Cada región debe ser homogénea con respecto a ciertos criterios de similitud. En el caso de imágenes 3D, a dicho proceso se le conoce como segmentación de volúmenes.

Las técnicas empleadas para la segmentación de volúmenes aplican algoritmos que requieren de un tiempo de procesamiento considerable [Ban08, PRH10]. Dicho procesamiento generalmente se realiza como una etapa previa al manejo del volumen, siendo dependiente de su tamaño y características.

Recientemente, una serie de algoritmos en la GPU (*Graphics Processing Unit*) han sido desarrollados con el objetivo de explotar su paralelismo y conseguir los mismos resultados que los algoritmos convencionales en la CPU (*Central Processing Unit*) pero en menor tiempo. Este crecimiento se debe principalmente a la facilidad de adquisición de un hardware gráfico y a la cantidad de herramientas de programación existentes. Un ejemplo notable se observa con la arquitectura CUDA desarrollada por NVIDIA Corporation [NVI11], la cual provee un entorno de desarrollo de algoritmos que se ejecutan en los núcleos (*cores*) de la GPU.

Como parte de los algoritmos que son implementados en la GPU para mejorar los tiempos de obtención sus resultados, se encuentra la segmentación de volúmenes. En este trabajo, se plantea un algoritmo basado en GrabCut [RKB04] para la segmentación de volúmenes en la GPU empleando la arquitectura de CUDA. La idea principal, es seleccionar una región de interés dentro del volumen a estudiar, luego aplicar una versión modificada del algoritmo de GrabCut y así obtener 2 sub-volúmenes: *foreground* (zona de interés) y *background* (resto del volumen).

Este trabajo presenta un enfoque novedoso para la segmentación en volúmenes basados en el algoritmo de GrabCut. El algoritmo construye un grafo con peso basado en los vóxeles del volumen y su color asociado de acuerdo a una función de transferencia. También se construye un algoritmo paralelo de Push-Relabel ejecutado y almacenado en el *hardware* gráfico. Al mismo tiempo, se presenta un esquema para el tratamiento de hilos inactivos en la GPU. Por último, se estudian los resultados obtenidos y son comparados con

una implementación en la CPU del mismo algoritmo en su versión secuencial.

La siguiente sección presenta una introducción a la teoría de grafos asociada con la segmentación de imágenes empleada en nuestra propuesta. En la sección 3 se explican todos los detalles de nuestra propuesta basada en el algoritmo de GrabCut para la segmentación de volúmenes. La sección 4 muestra las pruebas y resultados aplicados con el fin de comprobar la eficacia de nuestra propuesta: resultados visuales, tiempo de ejecución y consumo de memoria (tanto en la CPU como en la GPU). Finalmente, la sección 5 muestra las conclusiones y trabajos futuros.

2. Segmentación como un problema de grafos

El problema de segmentación de imágenes puede verse como un problema de grafos, al asociar a cada píxel un nodo del grafo. Boykov y Kolmogorov [BK04] plantean un enfoque para la minimización de energía empleando grafos. La minimización de energía dentro de un grafo permite separar componentes que tengan algún tipo de relación bajo un cierto criterio (e.g. color, gradiente, etc.).

Recientemente, se han empleado técnicas basadas en dividir una imagen, representada como un grafo, en dos o más sub-grafos, donde cada uno de ellos representa un subconjunto de este. Entre los algoritmos existentes para dividir un grafo, se encuentra el corte de un grafo.

2.1. Corte de un grafo

Sea $G = (V,A)$ un grafo dirigido que posee un conjunto de nodos y un conjunto de aristas que enlazan dichos nodos. El corte de un grafo consiste en eliminar un conjunto de aristas A', con el fin de obtener dos grafos disjuntos G_1 y G_2 tal que $G = G_1 \cup G_2$. El peso w del corte de un grafo es la suma de todos los pesos de las aristas eliminadas, $w = \sum w_e$. Por otro lado, el corte mínimo es el corte con el menor peso de todos los posibles cortes en el grafo.

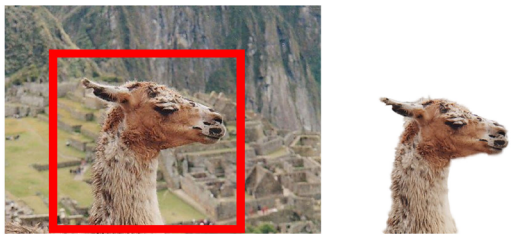
De acuerdo con el teorema de *max-flow/min-cut* [CLRS01], el corte mínimo en un grafo puede ser obtenido utilizando el algoritmo *max-flow*. Un algoritmo de *max-flow* consiste en determinar la máxima capacidad de flujo que puede ingresar a través de un nodo fuente (*source*) e ir hacia un nodo destino (*target*). En el algoritmo, las aristas saturadas (aristas con $w_e = 0$) son eliminadas por el algoritmo para obtener el corte mínimo. Entre los algoritmos más empleados para calcular el corte mínimo están los propuestos por Ford y Fulkerson [FF62] y el enfoque de Goldberg y Tarjan [GT88] llamado Push-Relabel.

Boykov y Jolly [BJ01] proponen una técnica denominada GraphCut, donde la imagen se representa como un grafo y se divide el grafo con un algoritmo de *max-flow/min-cut*. Posteriormente, Rother et al. [RKB04] introducen un enfoque novedoso llamado GrabCut, el cual se divide un grafo en dos regiones: *background* y *foreground* empleando una función de energía basada en similitud de los colores. Este enfoque emplea unas funciones de minimización de energía de forma eficiente utilizando un algoritmo de corte mínimo.

La siguiente sección muestra nuestro enfoque, 3D GrabCut, basado en la propuesta de Rother et al. [RKB04], aplicando modificaciones sobre el cálculo de los valores del grafo para ser usado en un volumen.

3. 3D GrabCut

GrabCut es un algoritmo iterativo compuesto de varias etapas que combina valores estadísticos y el enfoque de GraphCut [BJ01] con el fin de segmentar una imagen. En GrabCut, los datos de entrada son seleccionados por el usuario al seleccionar una región a segmentar dentro de la imagen original utilizando un rectángulo. En la Figura 1a se presenta un ejemplo, donde el usuario dibuja un rectángulo de color rojo que representa la región donde se encuentra el objeto a extraer. La Figura 1b muestra el resultado obtenido después de aplicar el algoritmo de GrabCut.



(a) Imagen original con el área seleccionada por el usuario

(b) Imagen segmentada

Figura 1: Segmentación de una imagen empleando la técnica de GrabCut. Imagen extraída de [RKB04].

Nuestra propuesta denominada 3D GrabCut, el proceso de segmentación se realiza sobre un espacio tridimensional. Por ello, el usuario debe seleccionar un cubo o sub-volumen en vez de utilizar un rectángulo, para seleccionar el objeto de interés.

El algoritmo de 3D GrabCut crea un grafo de flujo en redes [CLRS01] donde cada vóxel representa a un nodo del grafo. Cada nodo es enlazado con sus vecinos utilizando aristas dirigidas con peso, denominadas *N-Links*. El número de vecinos posibles puede ser de máximo 26 (todos los vóxeles que están a distancia 1). En el grafo de flujo en redes, existen 2 nodos especiales: fuente s y destino t. El nodo s está conectado con cada vóxel que pertenece al sub-volumen de selección (*foreground*). Igualmente, el nodo t está conectado con cada vóxel fuera de la selección (*background*).

Cada nodo del grafo está conectado con una arista con peso a la fuente y por medio de otra al destino. Estas aristas son denominadas *T-Links*, y su peso es calculado empleando los modelos mixtos gaussianos (GMM) [CCSS01] para los grupos *foreground* y *background*. En nuestro enfoque, un GMM está formado por 5 componentes para el *foreground* y 5 componentes para el *background*.

Un componente perteneciente a un GMM es derivado de las estadísticas de color en cada región de la imagen. Con el fin de separar los componentes que contengan a un grupo de vóxeles con características de color similares, se buscan los de menor valor en su varianza. Existen diversos métodos para realizar esta búsqueda. En 3D GrabCut, se empleó la técnica de cuantización de color descrita por Orchad y Bouman [OB91].

Luego de calcular los pesos de los *T-Links*, siguiendo el esquema de GrabCut, se aplica el algoritmo del corte mínimo en el grafo. El trabajo original de GrabCut aplica el algoritmo de Ford y Fulkerson. En esta investigación, el corte

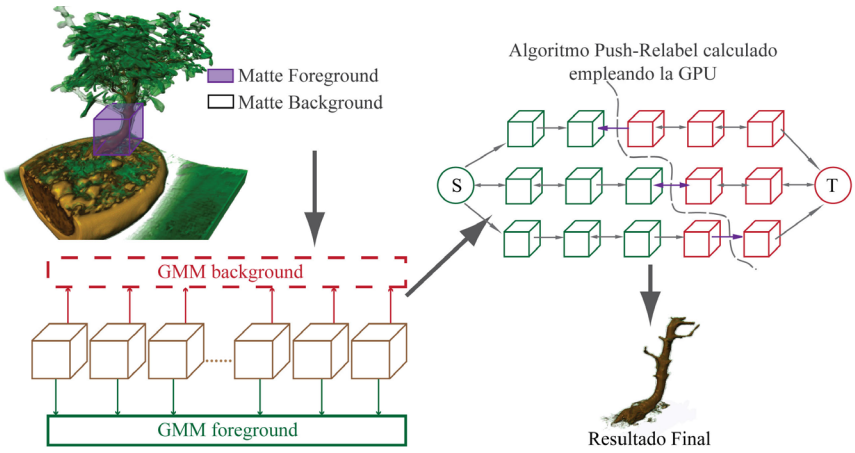


Figura 2: Esquema general del flujo de acciones realizadas en 3D GrabCut.

mínimo se realiza con la técnica de *Push-Relabel*, debido a que puede ser paralelizada eficientemente en la GPU.

En la Figura 2 se muestra el algoritmo aplicado en 3D GrabCut. Primero, el usuario selecciona el área donde se encuentra el objeto de interés, caja de color morado. Con esta selección se crea una lista de valores que representan a cada vóxel. Cada uno de estos valores se denomina *matte*. Si un vóxel se encuentra dentro de la selección, entonces pertenece al conjunto *matte foreground*, en caso contrario pertenece a *matte background*. Los valores de matte son modificados a lo largo del algoritmo y se emplean para construir el resultado final. Después, se calculan los valores de los *N-Links*.

Luego, a cada vóxel se le asigna un valor de probabilidad de pertenecer a cada componente del *GMM background* y *GMM foreground*. Una vez con estos valores, se asignan los valores de *T-Links* para los vóxeles. Entonces, con el grafo construido se ejecuta el algoritmo de corte mínimo para separar el grafo en dos sub-grafos (verde para el *foreground* y rojo para el *background*), ver Figura 2. El algoritmo de corte mínimo se basa en la propuesta de Goldberg y Tarjan [GT88] denominada Push-Relabel y ejecutada completamente en la GPU.

Finalmente, se eliminan los vóxeles que son parte del *background* para obtener el objeto de interés. A continuación, se explicará detalladamente el cálculo realizado para obtener el valor de los *N-Links*, *T-Links* y el algoritmo de Push-Relabel.

3.1. N-Link

Para el cálculo del valor del *N-Link*, $N(m,n)$, entre dos vóxeles m y n, se plantea una fórmula basada en el trabajo de Mortensen y Barrett [MB95] colocando la distancia entre vóxeles siempre igual a 1, ver Ec. 1. El valor de la constante 50 es un valor sugerido en la propuesta de Blake et al. [BRB*04] en el cálculo de modelos GMMRF (*Gaussian Mixture Markov Random Field*).

$$N(m,n) = 50 \times e^{-B||C_m - C_n||^2} \quad (1)$$

El valor de $||C_m - C_n||$ representa la distancia euclideana en el espacio de color RGB para los vóxeles m y n. El valor de B es una constante para asegurar una diferencia en las distancias de color. La Ec. 2 muestra como se calcula B, la cual es una pequeña variación de la fórmula original presentada por Boykov y Jolly [BJ01].

$$B = \frac{1}{\frac{2}{P} \times \sum_{m=0}^{P} \sum_{n=0}^{V} \|C_m - C_n\|^2} \quad (2)$$

El valor de la constante P representa el número de vóxeles en el volumen y V indica el número de vecinos para cada vóxel. En la implementación de 3D GrabCut, solo se utilizaron seis vecinos por cada vóxel en las siguientes direcciones: $\longrightarrow +X, \longleftarrow -X, \uparrow +Y, \downarrow -Y, \swarrow +Z, \nearrow -Z$. Entonces, $V = 6$ para cada vóxel dentro del volumen, a excepción de los bordes.

Como mencionamos anteriormente, los *N-Links* conectan un vóxel m con un vóxel vecino n. Además, un vóxel m tiene que estar conectado por medio de una arista *T-Link* con el *foreground* y con el *background*. En la siguiente sección muestra nuestro enfoque para construir los *T-Links*.

3.2. T-Link

Por cada vóxel del volumen, se almacena el número del GMM al cual pertenece. Adicionalmente, se crean tres marcas temporales para cada vóxel denominadas: *background*, *foreground* ó *unknown* (desconocido). Estas son calculadas por el algoritmo de acuerdo a la selección inicial del usuario, y no varían a través en la ejecución del algoritmo.

Los vóxeles marcados como *foreground* serán parte del *matte foreground*. Los vóxeles seleccionados como *background* siempre están definidos como *matte background*. Igualmente, los vóxeles que modificarán su valor de *matte* en la ejecución del algoritmo serán los seleccionados inicialmente como *unknown*.

Cada vóxel posee un *T-Link* conectado con el *foreground* llamado T_{fore}, y con el *background* llamado T_{back}. Si un usuario selecciona un vóxel m como *foreground*, entonces el corte mínimo no lo desconectará del *foreground*. Por lo tanto, para el vóxel m es asignado un valor de $T_{fore} = K_{max}$, donde K_{max} representa el máximo valor posible para el peso de una arista en el grafo; adicionalmente el valor para $T_{back} =$

0. El mismo enfoque se aplica cuando se selecciona un vóxel como *background* (i.e. $T_{back} = K_{max}$ y $T_{fore} = 0$).

Cuando un vóxel m posee la marca de unknown, $T_{fore} = P_{fore}(m)$ y $T_{back} = P_{back}(m)$. El valor de $P_{fore}(m)$ y $P_{back}(m)$ indican la probabilidad de m de pertenecer al GMM *foreground* y *background* respectivamente.

La probabilidad $P(m)$ del vóxel m se define en la Ec. 3:

$$P(m) = -log \sum_{i=1}^{k} P(m, i) \quad (3)$$

donde k representa el número de componentes en el GMM. El valor de $P(m, i)$ es la probabilidad de un vóxel m para pertenecer a un componente i. Como se mencionó anteriormente, se emplea el valor $k = 5$.

Para un m-vóxel se calcula la probabilidad $P_{back}(m, i)$ y $P_{fore}(m, i)$ en cada componente de los GMMs. La probabilidad $P(m)$ se expresa como $P(m) = -log \sum_{i=1}^{5} P(m, i)$, donde $P(m, i)$ se calcula de la misma forma como se propone en el trabajo de Talbot y Xu [TX06].

Una vez calculado los valores de *N-Link* y *T-Link* para todos los nodos del grafo, se aplica el algoritmo de corte mínimo. En 3D GrabCut se emplea el algoritmo de Push-Relabel el cual se explica a continuación.

3.3. Push-Relabel

Push-Relabel es un algoritmo eficiente para el cálculo del flujo máximo propuesto por Golberg y Tarjan [GT88] en el año 1986. A continuación se explicarán algunos conceptos básicos empleados en el algoritmo.

3.3.1. Conceptos básicos

Sea $G = (V, E)$ un grafo, con un nodo fuente s y un nodo destino t. El algoritmo Push-Relabel construye y almacena un grafo en cada iteración. Un grafo residual G_f del grafo G tiene la misma topología con la diferencia que está compuesto por aristas las cuales pueden admitir más flujo.

La capacidad residual $c_f(u, v)$ es la cantidad adicional de flujo que puede ser enviado desde el nodo u hacia v después de enviar $f(u, v)$, y calcula como $c_f(u, v) = c(u, v) - f(u, v)$ donde $c(u, v)$ es la capacidad de la arista (u, v).

El algoritmo almacena básicamente 2 valores: el exceso de flujo $e(v)$ y la altura $h(v)$ de cada nodo. El valor $e(v)$ indica la diferencia entre el flujo total entrante con el flujo total saliente para un nodo v. La altura $h(v)$ es un valor estimado de la distancia desde el nodo v hacia t. Al inicio, todos los nodos tienen $h(v) = 0$ excepto la fuente s cuya altura es $h(s) = n$, donde n representa el número de nodos en el grafo.

Existen dos operaciones básicas en el algoritmo, Push y Relabel. Una operación Push desde el nodo u hacia v consiste en enviar una parte del exceso de u, $e(u)$, hacia v. Para realizar esta operación se tienen que cumplir 3 condiciones:

1. $e(u) > 0$; debe existir exceso de flujo en u.
2. $c(u, v) - f(u, v) > 0$; debe existir capacidad disponible para enviar flujo de u hacia v.
3. $h(u) > h(v)$; el flujo solo puede ser enviado hacia un nodo destino con menor valor de altura.

Entonces, es posible deducir que el flujo enviado es igual a $min(e(u), c(u, v) - f(u, v))$.

En la operación Relabel, la altura $h(u)$ es incrementada hasta que sea al menos mayor en 1 que la altura mínima entre todos los nodos a los cuales se puede enviar flujo. Las condiciones para realizar esta operación son:

1. $e(u) > 0$; debe existir exceso de flujo en u.
2. $h(u) \leq h(v)$ para cada v que cumpla $c(u, v) - f(u, v) > 0$; solo los nodos con capacidad disponible para enviar flujo, serán considerados.

El incremento de la altura de u se calcula como $h(u) = 1 + min(h(v) : (u, v) \in E_f)$

Para iniciar el algoritmo, se invoca una operación llamada $Preflow(G, s)$. En la operación, la fuente s envía su exceso (el cual inicialmente es ∞) hacia todos los nodos con capacidad disponible. Una estructura para representar un algoritmo genérico de Push-Relabel se muestra a continuación:

```
1: Preflow(G, s)
2: while push or relabel do
3:     Push();
4:     Relabel();
5: end while
```

Para aplicar el algoritmo de Push-Relabel en un ambiente paralelo, es necesario modificar algunos detalles para ser ejecutado en la arquitectura CUDA. A continuación se presenta una implementación de Push-Relabel en la GPU.

3.4. Push-Relabel en la GPU

Un algoritmo desarrollado para ser ejecutado enteramente en la GPU debe ser paralelizable para explotar la capacidad máxima del mismo. Un primer enfoque de paralelismo aplicado al algoritmo de Push-Relabel fue presentado por Anderson y Setubal [AS92]. Alizadeh y Golberg [AG92] presentaron una implementación de un corte mínimo en paralelo empleando una máquina de conexión masiva en paralelo CM-2. En el 2005, Bader y Sachdeva [BS05] desarrollaron una optimización basada en *cache-aware* para la paralelización de dicho algoritmo.

En la literatura, existen pocas versiones paralelas del algoritmo *max-flow/min-cut* sobre la GPU, siendo las más relevantes las presentadas por Paragios [DKP05] y Varshney y Davis [HVD07]. En el 2008, Vinet y Narayanan [VN08] implementaron el algoritmo de GrabCut para imágenes 2D en la GPU bajo CUDA.

En 3D GrabCut se presenta un nuevo algoritmo paralelo en la GPU para Push-Relabel. En el algoritmo de Push-Relabel presentado en la sección previa, las etapas de $Preflow(G, s)$, $Push()$, $Relabel()$ fueron modificadas con la finalidad de ser paralelizadas. Adicionalmente, se creó una cola de hilos activos para tener acceso directo a estos. A continuación se describirán estas etapas.

3.4.1. Operación Push

La operación Push es aplicada localmente en cada nodo, donde cada uno de estos nodos envía flujo hacia sus vecinos. La finalidad de esta operación es reducir el exceso de flujo en cada nodo. Además, un nodo puede recibir flujo de sus vecinos. Cuando esta operación se realiza en paralelo,

esto puede ocasionar errores cuando el flujo y exceso son actualizados (escritura/lectura simultánea).

La arquitectura CUDA permite evitar estos posibles errores al utilizar funciones atómicas [vid09]. Las funciones atómicas permiten ejecutar operaciones de lectura-escritura-actualización en memoria de video local o global. Dichas operaciones garantizan su completa ejecución sin la interferencia de otros sub-procesos.

3.4.2. Operación Relabel

En la etapa de Relabel local en la versión original del algoritmo Push-Relabel [GT88], si un nodo no tenía la posibilidad de enviar su flujo hacia el destino entonces debe incrementar su altura h hasta que sobrepase la altura de la fuente en 1 y retornar el exceso de flujo que posee.

Dado que el peso de la fuente es igual a N (número de nodos), el algoritmo debe efectuar en el peor de los casos N operaciones de Relabel. En nuestro enfoque, las imágenes volumétricas poseen valores muy grandes para N. Por ejemplo, para un volumen de de dimensiones $128 \times 128 \times 128$ vóxeles, el valor de N sería $N = 2.097.152$ vóxeles.

En 3D GrabCut, la operación Relabel se hace de manera global utilizando la distancia hacia el nodo destino. Esta operación global se realiza basándose en los nodos con capacidad disponible hacia el destino t. Estos nodos tendrán una altura h igual a la altura del destino h_t más 1, $h = h_t + 1$. Al comienzo, el valor h de los nodos es igual a 1. En cada iteración, los nodos que posean capacidad disponible con otros nodos (previamente etiquetados), tendrán el valor h igual a la altura de los nodos conocidos más 1. Entonces, en iteraciones posteriores, algunos nodos podrían ser aislados de t y solo estar conectados a la fuente. Esto causa que la operación de Relabel global no etiquete estos nodos.

Cuando la operación de Relabel global es aplicada en nodos que tienen exceso y están aislados, estos nunca podrán enviarlo y el algoritmo no se detendrá. Con la finalidad de resolver esto, se aplica el mismo enfoque pero en vez de usar el nodo t, será utiliza el nodo fuente s. Con esto, si existe un nodo no etiquetado con capacidad disponible hacia s, entonces su altura h será $h = N + 1$.

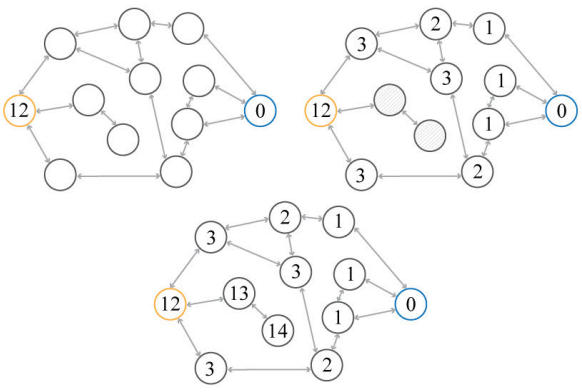


Figura 3: Ejemplo de la operación Relabel global.

En la Figura 3 se muestra un ejemplo de esta operación. Primero, se inicializan las alturas del nodo fuente y nodo destino. La altura del nodo fuente será $h_s = N$, donde $N = 12$, y la altura del nodo destino es $h_t = 0$. Luego, se actualizan los valores de altura de los nodos de acuerdo a su distancia al nodo t; en la Figura 3 se calcula la altura de 8 nodos (altura máxima de 3). Se puede observar como existen 2 nodos que quedaron aislados con respecto al destino. Finalmente, las alturas de los nodos aislados se calculan con respecto a la distancia al nodo s (altura 13 y 14).

Un problema con este enfoque, es que durante la ejecución del algoritmo, existen hilos que por no poseer exceso no ejecutan ninguna instrucción. Esto genera hilos ociosos en ejecución. Para solventar este problema, en cada iteración se crea una cola que contiene el índice de los nodos que van a ejecutarse en la próxima iteración. Con ello, solamente se crean los hilos necesarios y no habrá hilos inactivos, utilizando así toda la capacidad del hardware gráfico.

A continuación se muestra el algoritmo pseudo-formal *Push-Relabel* en la GPU:

1: *Relabel_Global*()
2: *Preflow*(G,s)
3: **while** *Exceso* **do**
4: *Push*()
5: *Relabel_Global*()
6: *Crear_Cola*()
7: **end while**

La instrucción de la línea 1 ejecuta la operación de *Relabel_global*(), tal como se muestra en la Figura 3. La siguiente instrucción ejecuta la operación de *Preflow*(G,s), explicada en la sección 3.3.1. Entonces, mientras exista exceso en algún nodo del grafo, se procede a hacer la operación de *Push*(), seguida del *Relabel_Global*() y se crea la cola de nodos activos para la siguiente iteración.

La función *Crear_Cola*() construye un arreglo lineal donde se almacenan los índices de los hilos, que representan a un nodo activo del grafo, a emplear en la próxima iteración. La cola se encuentra en memoria de video y creada bajo un acceso concurrente.

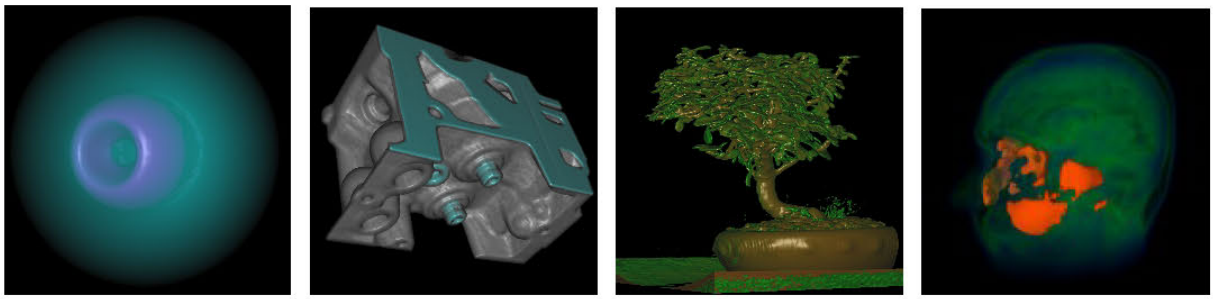
4. Pruebas y Resultados

Con el objetivo de comprobar la efectividad de 3D GrabCut, se realizaron diversas pruebas de segmentación a un grupo de volúmenes. La implementación fue realizada sobre el lenguaje de programación CUDA C en la GPU bajo la arquitectura CUDA [vid09]. La visualización de los volúmenes se realizó en OpenGL®. Los dispositivos de hardware gráfico utilizados poseían una tarjeta NVIDIA con soporte de capacidad de CUDA versión 1.1 (*capability*).

Las pruebas fueron ejecutadas en un computador con procesador Intel i5 3.20 GHz, memoria RAM de 4 Gb y una tarjeta gráfica NVIDIA GTX 470 (NV_1) de 448 cores; y otro computador con las mismas características pero con una NVIDIA GTX 240 (NV_2) de 96 cores. El sistema operativo de las pruebas es Windows 7 64 bits. Los volúmenes empleados en las pruebas tienen una precisión de muestreo de 8 bits. En la Tabla 1 se muestra las características de los volúmenes y en la Figura 4 se observa un despliegue de estas.

4.1. Tiempos de ejecución

La segmentación de los volúmenes fue realizada empleando un algoritmo secuencial de 3D GrabCut en la CPU y un al-



(a) Simulación de la distribución de probabilidad de 2 cuerpos de un nucleón en el núcleo atómico 16O

(b) Tomografía computarizada de un motor

(c) Tomografía computarizada de un árbol bonsai

(d) Resonancia magnética de una cabeza humana

Figura 4: Volúmenes empleados para las pruebas, de izquierda a derecha Vol_1, Vol_2, Vol_3, y Vol_4. Todos los volúmenes fueron extraídos de `http://www.volvis.org`

Volumen	Vol_1	Vol_2	Vol_3	Vol_4
Dimensiones	41^3	$256^2 \times 128$	256^3	256^3
Tamaño en Mb	0.06	8	16	16
# de vóxeles	68.921	8.388.608	2^{24}	2^{24}

Tabla 1: Características de los volúmenes empleados.

goritmo paralelo en la GPU. Para cada volumen, se consideran 3 funciones de transferencias distintas para determinar los resultados de la segmentación. La primera de ellas representa a la función identidad (FT_1), la segunda representa a una función que trata de eliminar el ruido del volumen (FT_2) y la tercera función fue generada de manera de resaltar un objeto dentro del volumen (FT_3).

El algoritmo empleado (CPU y GPU) se dividió en dos etapas primordiales: creación del grafo y ejecución del algoritmo de *max-flow*. La construcción del grafo ocupa en promedio un 5% del tiempo total del algoritmo, y el *max-flow* ocupa el 95% de toda la ejecución.

El número de vóxeles que están dentro del cubo de selección, representan un factor importante en el tiempo y cantidad de memoria ocupada por el algoritmo. Por ejemplo para un mismo sub-volumen de selección, empleando el volumen Vol_1, el algoritmo en la CPU se ejecutó en $72s$ y algoritmo en la GPU en $68s$ y $113s$ para las tarjetas NV_1 y NV_2 respectivamente. Los resultados arrojaron que la implementación en la CPU fue más rápida que la implementación en la GPU en la tarjeta NV_2 debido a que el volumen es pequeño, y al ejecutar un algoritmo en la GPU se debe considerar el tiempo de transferencia de datos desde la GPU hasta la CPU.

Al emplear un volumen de mayor tamaño, Figura 8b, los tiempos de ejecución varían. Los tiempos de ejecución del algoritmo paralelo fueron $\sim 10min$ y $\sim 12min$ (para las 3 funciones), y en la CPU tardó $\sim 89min$ para una función de transferencia y $\sim 4.5h$ para las otras 2 funciones de transferencia. La forma de la función de transferencia influye en el tiempo de ejecución del algoritmo propuesto, debido al número de iteraciones necesarias para completar el algoritmo de *max-flow*.

Empleando el volumen Vol_3 y Vol_4, la tarjeta NV_2 no pudo ejecutar el algoritmo satisfactoriamente. Esto se debe porque la capacidad de memoria de video (1024 Mb) no soporta almacenar la estructura del grafo. Para el caso del volumen Vol_3, el algoritmo en la GPU de la tarjeta NV_1 arrojó un tiempo de ejecución de $\sim 19min$ y en la CPU el tiempo mínimo fue $\sim 2.8h$ y máximo $\sim 6.3h$.

El volumen Vol_4 posee el mismo tamaño que el volumen Vol_3; pero hay una variación en los tiempos de ejecución. El algoritmo en la GPU se ejecutó en promedio en un tiempo de $\sim 17min$ y en la CPU el tiempo mínimo fue de $\sim 7.5h$ y máximo de $\sim 14h$. Esto se debe a la influencia del número de hilos creados en las primeras iteraciones del algoritmo, de acuerdo a la dimensión del sub-volumen de selección.

4.2. Espacio requerido en memoria

Para el cálculo de la memoria requerida por el algoritmo, se realizó una medición de acuerdo a las estructuras de datos que consumen espacio: arreglo de vóxeles, listas que guardan el tipo de nodo en un instante de tiempo (i.e. *matte*), cola de hilos activos y lista que guarda el valor de la conexión entre la fuente y cada nodo. La cantidad de memoria empleada se puede observar en la Tabla 2.

Volumen	Vol_1	Vol_2	Vol_3	Vol_4
Mem. necesaria	3.5Mb	424 Mb	848Mb	848Mb
Mem. total	3.56Mb	432Mb	864Mb	864Mb

Tabla 2: Memoria requerida por cada volumen de prueba.

En la tabla se observa la memoria necesaria que se crea durante el algoritmo y la memoria total que representa la suma de la memoria necesaria más el espacio que ocupa el volumen, ver Tabla 1.

4.3. Generación de hilos

En la primera iteración del algoritmo, se ejecuta una serie de hilos o *threads* igual al número de nodos en el grafo. Si este esquema se mantiene para las siguientes iteraciones, el número de hilos "inactivos" (sin carga de procesamiento alguno) se incrementa, haciéndolo un esquema ineficiente.

Como se mencionó en la sección 3.4, para evitar la

creación de hilos inactivos se construyó una cola para almacenar los hilos activos para la siguiente iteración del algoritmo. La existencia de hilos inactivos genera una carga en la GPU por su creación, manejo y contexto de ejecución la cual es innecesaria. En la Figura 5 se muestra el número de hilos utilizados en cada iteración de 3D GrabCut al ser aplicado al volumen Vol_3 aplicando las 3 funciones de transferencia (FT_1, FT_2 y FT_3) para una selección arbitraria del volumen.

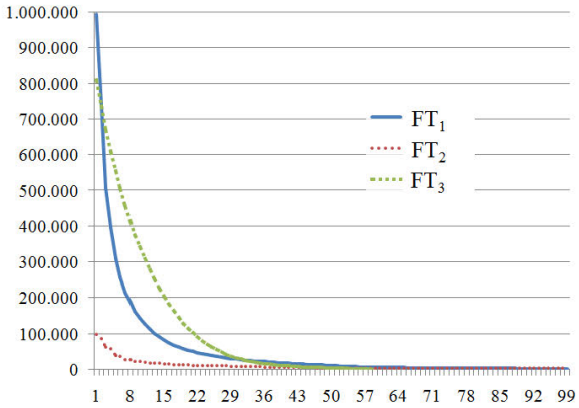


Figura 5: Número de hilos utilizados por iteración en la ejecución del algoritmo para el volumen Vol_3 con 3 funciones de transferencia distintas.

Se puede observar como en la primera iteración, el número de hilos es considerable ($\sim$ 1.000.000). Pocas iteraciones posteriores, el número de hilos va decrementando drásticamente hasta emplear solamente unas cuantas decenas de hilos, logrando así la segmentación. La gráfica muestra solo 100 iteraciones; pero en el ejemplo se requirió $\sim$ 650 iteraciones para la FT_1, FT_2, y $\sim$ 120 para FT_3.

4.4. Calidad del resultado

Como se mencionó anteriormente, las pruebas consistieron en aplicar 3 funciones de transferencias distintas (FT_1, FT_2 y FT_3). Las pruebas realizadas para el volumen Vol_1 buscaron segmentar el anillo que se encuentra en un extremo del volumen en dirección del eje Z. Para las 3 funciones de transferencia y aplicando el mismo sub-volumen de selección, se consiguió la adecuada separación foreground/background.

El proceso de segmentación de Vol_2 busca separar dos piezas cilíndricas y dos anillos presentes en el volumen dentro de su estructura. Los resultados arrojaron una buena segmentación al aplicar las 3 funciones de transferencia. La segmentación con la función FT_3 se obtuvo en un tiempo menor al empleado por las otras 2 ($\sim$ 35% menor), debido a su característica de la clasificación de los vóxeles.

En la segmentación del volumen Vol_3, se consiguió resultados adecuados al aplicar las 3 funciones transferencias para separar el tallo del árbol del resto. Sin embargo, en los 3 casos, el sub-volumen producto de la segmentación presentó vóxeles que pertenecían a la tierra donde está sembrada dicho árbol. Esto se debe a la similitud de intensidad y color de estos con el tallo del árbol. En la Figura 6 se observa la selección del usuario sobre el volumen que representa a una tomografía computarizada de un árbol bonsai al aplicar la función de transferencia FT_2.

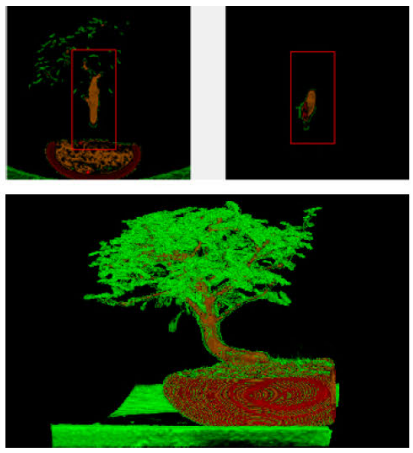


Figura 6: Selección del usuario para la segmentación del tallo en un volumen que representa una tomografía computarizada de un árbol bonsai.

Se observa en la parte superior de la figura dos ventanas, las cuales representan la selección hecha bajo una vista en dirección del plano de corte coronal (izquierda) y transversal (derecha). El resultado visual se muestra en la Figura 7 donde se muestra la representación del sub-volumen obtenido.

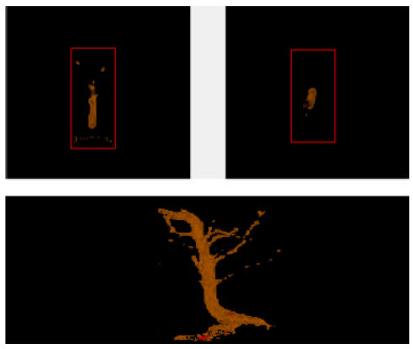


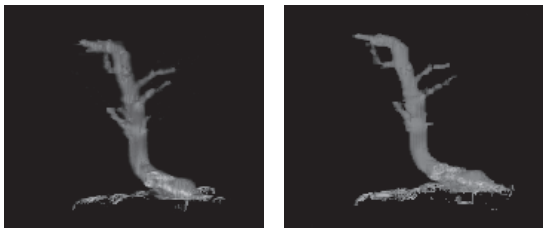
Figura 7: Resultado de la segmentación empleando 3D GrabCut del volumen mostrado en la Figura 6.

En el volumen Vol_4 se busca realizar la segmentación del cerebro de la cabeza humana. Empleando la FT_1 se genera más ruido que las otras 2 funciones, debido a la poca variación de intensidad entre los vóxeles del *foreground* y *background*. Sin embargo, la solución obtiene el cerebro con pequeños fragmentos de corteza cerebral que no deben estar presentes. Al utilizar las funciones FT_2 y FT_3, se obtienen mejores resultados a pesar de generar un sub-volumen poco real de acuerdo a los colores de la anatomía cerebral.

4.5. CPU vs. GPU

Los volúmenes obtenidos luego de la segmentación fueron los mismos al emplear cualquiera de las tarjetas gráficas antes descritas. Ahora, los resultados visuales obtenidos con el algoritmo ejecutado en la CPU difieren ligeramente. En la Figura 8 se observan los resultados obtenidos.

Los resultados muestran que los datos de salida pueden ser distintos de acuerdo al orden de ejecución del algoritmo.



(a) Empleando la GPU (b) Empleando la CPU

Figura 8: Comparación entre un resultado aplicando el algoritmo en la CPU y la GPU.

En la GPU, el orden en que cada nodo actualiza su exceso modifica de manera diferente el grafo que la ejecución en la CPU, porque un grafo puede tener más de un corte mínimo. Ambos resultados son correctos desde el punto de vista visual; pero en diversas corridas en la GPU se pueden obtener segmentaciones distintas. Una forma de verificación de este fenómeno, es realizar un mecanismo de comparación (e.g. diferencia de vóxeles e intensidades, *heatmap*, etc.) para tener control sobre los resultados obtenidos.

5. Conclusiones y Trabajos Futuros

En este trabajo se presenta un enfoque novedoso para la segmentación de volúmenes empleando la GPU basado en la técnica GrabCut denominado 3D GrabCut. En 3D GrabCut se emplean técnicas de paralelismo sobre el hardware gráfico bajo la arquitectura CUDA. Las pruebas realizadas muestran excelentes resultados tanto visuales como en tiempo.

El tiempo de ejecución y el resultado visual después de la segmentación depende del tamaño de la selección y de la función de transferencia. Si el sub-volumen de selección es grande en relación al tamaño del volumen, el número de nodos e hilos activos será considerable, implicando un mayor tiempo para obtener el corte mínimo. La función de transferencia determina el color que se asigna a un valor de intensidad de un vóxel, si el color es similar dentro y fuera de la selección, entonces el algoritmo asumirá ciertos vóxeles del *foreground* como del *background* y viceversa.

Una desventaja del algoritmo propuesto radica en el espacio requerido en memoria de video para almacenar las estructuras de datos a emplear. Nuestras pruebas fueron realizados con volúmenes de un tamaño máximo de $256 \times 256 \times 256$ vóxeles tomando en cuenta dicho factor. Al poseer tarjetas gráficas de mayor capacidad de memoria, se podrá manipular volúmenes de mayores dimensiones.

En el trabajo presentado por Delong y Boykov [DB08], se presenta una modificación del algoritmo Push-Relabel basado en un enfoque paralelo llamado region push-relabel. En el futuro, se propone implementar este enfoque en nuestra propuesta y comparar los resultados obtenidos.

Otra propuesta radica en reducir el espacio ocupado por las estructuras de datos, y así permitir segmentar volúmenes de un mayor tamaño. Por último, se propone aplicar otros algoritmos de corte mínimo en el grafo que puedan ser paralelizados en la GPU eficientemente para realizar comparaciones de tiempo de ejecución.

Referencias

[AG92] ALIZADEH F., GOLDBERG. A.: *Implementing the pushrelabel method for the maximum flow problem on a connection machine*. Stanford University, 1992.

[AS92] ANDERSON R. J., SETUBAL J. A. C.: On the parallel implementation of Goldberg's maximum flow algorithm. In *Proceedings of the fourth annual ACM symposium on Parallel algorithms and architectures* (1992), SPAA '92, ACM, pp. 168–177.

[Ban08] BANKMAN I. N.: *Handbook of Medical Image Processing and Analysis*, 2 ed. Academic Press, New York, 2008.

[BJ01] BOYKOV Y., JOLLY M.-P.: Interactive Graph Cuts for Optimal Boundary & Region Segmentation of Objects in N-D images. In *Proceedings of the 8th IEEE International Conference on Computer Vision ICCV* (2001), vol. 1, pp. 105–112.

[BK04] BOYKOV Y., KOLMOGOROV V.: An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26* (2004), 1124–1137.

[BRB*04] BLAKE A., ROTHER C., BROWN M., PÉREZ P., TORR P. H.: Interactive image segmentation using an adaptive GMMRF model. In *European Conference on Computer Vision* (2004), pp. 428–441.

[BS05] BADER D. A., SACHDEVA V.: A cache-aware parallel implementation of the push-relabel network flow algorithm and experimental evaluation of the gap relabeling heuristic. In *ISCA International Conference on Parallel and Distributed Computing Systems* (2005), pp. 41–48.

[CCSS01] CHUANG Y.-Y., CURLESS B., SALESIN D. H., SZELISKI R.: A bayesian approach to digital matting. In *Proceedings of IEEE CVPR 2001* (2001), vol. 2, IEEE Computer Society, pp. 264–271.

[CLRS01] CORMEN T. H., LEISERSON C. E., RIVEST R. L., STEIN C.: *Introduction to Algorithms*, 2 ed. MIT Press, New York, 2001.

[DB08] DELONG A., BOYKOV Y.: A scalable graph-cut algorithm for n-d grids. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2008), pp. 1–8.

[DKP05] DIXIT N., KERIVEN R., PARAGIOS N.: GPU-Cuts: Combinatorial optimisation, graphic processing units and adaptive object extraction, 2005.

[FF62] FORD L. R., FULKERSON D. R.: Flows in networks. *Princeton University* (1962).

[GT88] GOLDBERG A., TARJAN R.: A new approach to the maximum flow problem. *Journal of the ACM 35* (1988), 921–940.

[HVD07] HUSSEIN M., VARSHNEY A., DAVIS L.: On implementing Graph Cuts on CUDA, 2007.

[MB95] MORTENSEN E. N., BARRETT W. A.: Intelligent scissors for image composition. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (1995), SIGGRAPH '95, ACM, pp. 191–198.

[NVI11] NVIDIA CORPORATION: NVIDIA homepage. `http://www.nvidia.com`, Diciembre 2011.

[OB91] ORCHARD M., BOUMAN C.: Color quantization of images. *IEEE Transactions on Signal Processing 39* (1991), 2677–2690.

[PRH10] PRASSNI J.-S., ROPINSKI T., HINRICHS K.: Uncertainty-aware guided volume segmentation. *IEEE Transactions on Visualization and Computer Graphics 16* (Nov. 2010), 1358–1365.

[RKB04] ROTHER C., KOLMOGOROV V., BLAKE A.: GrabCut: Interactive Foreground Extraction using Iterated Graph Cuts. *ACM Transactions on Graphics 23* (2004), 309–314.

[TX06] TALBOT J., XU X.: Implementing GrabCut, 2006. Brigham Young University.

[vid09] CUDA Programming Guide 2.3. `http://developer.download.nvidia.com`, 2009.

[VN08] VINEET V., NARAYANAN P.: CUDA cuts: Fast graph cuts on the GPU. In *Computer Vision and Pattern Recognition Workshops* (2008), IEEE Computer Society, pp. 1–8.