Rev. Téc. Ing. Univ. Zulia. Vol. 31, Nº 3, 266 - 274, 2008

BVR: a storage format for navigation in volumetric data

***Leonardo Zúñiga*[1] *y Francisco Hidrobo*[2]**

[1]*Facultad de Ingeniería, Telefax: 0274-2402811. leoz@ula.ve.*
[2]*Facultad de Ciencias, elefax: 0274-2401284. hidrobo@ula.ve*
Universidad de Los Andes, Mérida 5101, Venezuela,

Abstract

In this paper we present the design and implementation of a format for volumetric data storage, along with the corresponding resolution algorithms of requests. This format allows the implantation of rendering algorithms with capacities to respond fast and efficiently to navigation requests with arbitrary trajectories. The proposal is based on grouping of *voxel* data in blocks (clusters) which match with the disk blocks size. Beside, we propose a multilayer model and develop a library that allows the usage of our format. The proposed model can be used in the visualization of huge volumetric data and its usefulness can be improved to support navigation based on data properties. To evaluate our proposal, we developed a visualization application which tests stability and performance of the implemented library and shows like using it.

Key words: Volumetric data, 3D visualization, image storage.

BVR: un formato de almacenamiento para la navegación en datos volumétricos

Resumen

En este artículo presentamos el diseño e implementación de un formato de almacenamiento de datos volumétricos, junto con los correspondientes algoritmos de resolución de peticiones. Este formato permite la implantación de algoritmos de visualización con capacidades de respuestas rápidas y eficientes a peticiones de navegación en trayectorias arbitrarias. La propuesta se basa en la agrupación de *voxels* de datos en bloques (clusters) que coinciden con el tamaño de los bloques de disco. Además, se propone un esquema de funcionamiento basado en capas y se implanta una librería que permite el uso de dicho formato. El esquema propuesto puede utilizarse en la visualización de datos volumétricos donde el tamaño de los datos es varias veces mayor al disponible en memoria principal y puede ampliarse su utilidad para soportar la navegación basada en propiedades del conjunto de datos. Para evaluar el esquema propuesto, se desarrolló una aplicación de visualización que prueba la estabilidad y el rendimiento de la librería implantada, y sirve de ejemplo de uso.

Palabras clave: Datos volumétricos, visualización 3d, almacenamiento de imágenes.

1. Introducción

En la actualidad existen aplicaciones de visualización que usan grandes volúmenes de datos, estas aplicaciones son utilizadas en diversas áreas como medicina, geofísica, climatología, entre otras. En estas aplicaciones se obtienen, a través de mediciones, o a partir de modelos, una gran cantidad de datos que luego deben ser visualizados mediante alguna clase de visualización 2D/3D. El volumen de los datos ha aumentado significativamente por al aumento en las re-

soluciones de los dispositivos de medición y por la mayor precisión en las técnicas de modelado. Además, los medios de almacenamiento se diversifican y abaratan, haciendo que se obvien las restricciones de espacio en beneficio de la calidad en los datos.

Es común que se requieran despliegues animados y dinámicos de información, por vías diferentes, siendo una de las más resaltantes la animación 3D en tiempo real, en cuyos casos el sistema de visualización debe responder con prontitud para garantizar la ilusión óptica de movimiento. La animación de volúmenes de datos tiene que confrontar la representación discreta de un dato por cada uno de los puntos en el espacio, a los que denominaremos *voxel*[1] *(volumetric element)* o elementos volumétricos, de la misma manera que la palabra *pixel* se corresponde a elementos pictóricos *(picture element)*. Estos *voxels* deben poseer un valor de transparencia que permita visualizar toda la información del volumen, dicho valor también es útil para la conformación de iso-superficies y primitivas geométricas adicionales en la visualización, realzadas del resto de los datos dándoles a éstos una menor transparencia. Es por esto que los algoritmos de visualización son costosos tanto en procesamiento como en almacenamiento. El primer problema es abordado por las tecnologías de procesamiento de imágenes basado en hardware; el segundo recurre al campo de las tecnologías de almacenamiento y manejo de datos.

En cuanto a almacenamiento, dos aspectos son fundamentales: el tamaño de memoria disponible y la velocidad de acceso a los datos en disco. El objetivo es que los datos estén en memoria cuando sean requeridos, por lo cual es deseable algún mecanismo de predicción eficiente cuando la trayectoria es desconocida a priori.

En este artículo presentamos el diseño e implementación de un formato de almacenamiento de datos volumétricos, junto con los correspondientes algoritmos de resolución de peticiones, que proveen flexibilidad y rapidez para algoritmos de visualización permitiendo mejorar la velocidad de respuestas a las solicitudes de navegación en trayectorias arbitrarias dentro del volumen a visualizar.

2. Marco de Referencia

En esta sección presentamos algunos elementos que sirven de base a la propuesta y que nos permiten describir el marco referencial en los aspectos de almacenamiento y visualización.

2.1. Esquemas de almacenamiento

Los esquemas de almacenamiento de datos volumétricos son diversos, son variados los campos del conocimiento que hacen uso de estos, pero se destacan principalmente aquellos que los emplean como un medio para "ver" dentro de ciertas estructuras, principalmente los servicios de imagenología médica [1-3] y los estudios geológicos [4-6].

Existen soluciones basadas en el funcionamiento de los dispositivos, como el formato SEGY [7] que es específico al formato secuencial propio de las cintas magnéticas; aunque ha sido adaptado a dispositivos no secuenciales [4, 6, 8] sin optimización para visualización 3D. Otras propuestas como NetCDF (Network Common Data Form) [9, 12] incluyen tanto el formato como una biblioteca para acceder a los datos. Además, NetCDF ha servido de base para propuestas especializadas como MINC [2], específico para data neurológica. *Hierarchical Data Format* (HDF5) [10-12] es una biblioteca, con formato genérico, que puede almacenar dos objetos primarios: Conjuntos de datos (arreglo multidimensional de elementos) y Grupos (estructura para organizar objetos).

Por otro lado, han surgido propuestas basadas en sistemas distribuidos de almacenamiento [13], y compresión de datos [14-16], con las consiguientes restricciones de costos en hardware para el primer caso, y de procesamiento en el segundo. Esto ha llevado a buscar soluciones que permitan obviar partes no importantes de los volúmenes como propone Mröz en [17], sin resolver el 100% de los casos de uso, puesto

1 Si bien la definición de voxel implica un subespacio volumétrico con 8 puntos, estamos usando un único punto para representarlo.

que precisa de algún criterio para reconocer y descartar las partes menos importantes del volumen de datos.

En [18], proponen un herramienta propietaria que presenta un esquema de fragmentación de datos que permita su manipulación de forma eficiente; representando una posibilidad de responder a peticiones no alineadas con los ejes de coordenadas, la mayoría de los otros esquemas presenta serias dificultades para resolver peticiones de regiones arbitrarias en secuencias arbitrarias.

Los esquemas tradicionales solamente usan localidad espacial de los datos en una o dos dimensiones; mientras que en nuestra propuesta se pretende explotar la localidad espacial en 3D.

2.2. Visualización volumétrica

Aunque existen diversos estrategias e implementaciones para realizar visualización volumétrica [19], el algoritmo específico de visualización no es relevante para nuestro esquema, lo que interesa es la manera como ese algoritmo accede a los datos almacenados.

Para mostrar datos volumétricos como superficies podemos recurrir a la búsqueda de iso-superficies dentro del volumen, tal como el algoritmo de cubos marchantes [20]. La desventaja de este esquema es que las superficies obtenidas son un sub-muestreo de los datos iniciales, y por lo general no se tiene acceso a la información contenida en la parte interna de dichas superficies, así como no se pueden llevar a cabo cálculos sobre el volumen, salvo aquellos que hagan uso de la información de contorno obtenida por el algoritmo.

Otra forma de construir la representación espacial de los datos volumétricos es usar visualización directa, asignando capa por capa, junto con sus valores de transparencia, el cubo de datos a un conjunto de superficies 2D perpendiculares a la línea de visión del usuario [21]. Sin embargo, este esquema tiene como restricción la carga de trabajo que significa construir las superficies en memoria, sin aprovechar las potencialidades de la GPU [22]. Aún cuando se cuenten con lo servicios de la GPU, es necesario hacer llegar los datos desde el dispositivo de almacenamiento, con lo cual el esquema propuesto sigue siendo de utilidad en los casos donde el volumen no cabe en memoria (CPU o GPU).

3. El formato BVR

3.1. Descripción general

El objetivo general del formato "***Block-Voxel Representation*** (**BVR**)" (en español *Representación de Bloques de Voxels)* es aumentar la capacidad de visualización, es decir, permitir el recorrido de un volumen de datos mostrando un subvoulmen en cada instante con trayectorias arbitrarias. Esto no excluye el empleo del esquema para otros fines tales como el acceso a los datos para realizar cálculos sobre ellos y definir trayectorias en función de los resultados obtenidos. De esta manera, se tienen en consideración los patrones de acceso de los algoritmos de visualización buscando optimizar el acceso a datos para hacerlo más fluido. Una consideración importante acerca del formato a implementar es la forma más efectiva de leer los datos desde el dispositivo de almacenamiento. En general, los sistemas de archivos leen un lote de datos, aún cuando la petición sean unos pocos bytes. También se conoce que los tiempos de búsqueda dentro del disco constituye una de las causas más importantes de retardo en las operaciones de lectura, así que es deseable ajustar las peticiones para que aprovechen al máximo esta lectura por lotes, y estén ordenados de forma de aprovechar capacidades de los discos modernos como *read-ahead* y *disk caching*.

3.2. Esquema de agrupación de *voxels*

Proponemos un esquema de almacenamiento basado en grupos (clusters) de 10x10x10 *voxels* de 4 bytes, como ilustra la Figura 1, los cuales serán denominados bloques.

Los bloques en sí ocupan 4000 bytes, así que se agrega un relleno de 96 bytes, totalizando 4096 bytes, que es el tamaño estándar de la mayoría de sistemas de archivos. Este conjunto de bloques serán precedidos de una cabecera para almacenar información global de los bloques.

La cabecera y la la información de los bloques son los componentes principales del formato ***BVR***. Como muestra la Tabla 1, la parte de datos está compuesta por una serie de bloques, los

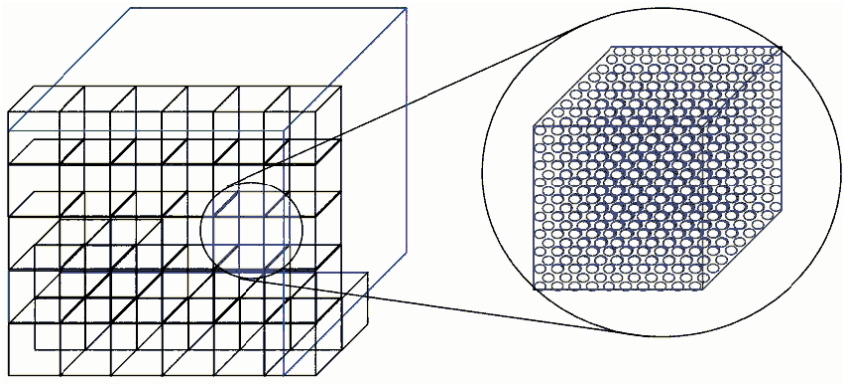

Figura 1. Agrupación de datos.

Tabla 1
Disposición de la información en un archivo en formato BVR

Cabecera				Datos		
Coordenadas	dimensiones	promedios	Relleno	Bloque 0	Bloque i...	Bloque n
96	12	n*4	<4096	4096	4096	4096

cuales almacenan la información del volumen junto con información adicional en los rellenos, tal como las posiciones espaciales [x,y,z] de los vértices del bloque, sus promedios, desviación estándar, mínimos y máximos, direcciones de bloques con mayor nivel de detalle, o cualquier otra información relevante que permita extender la funcionalidad del formato.

Antecediendo el área de datos, se hace uso de una cabecera que almacena las coordenadas de los vértices del cubo (96 bytes), la cantidad de *voxels* en cada dimensión (12 bytes), los promedios de cada uno de los bloques del archivo principal (**n** Bloques x 4 bytes) y, al final, un relleno que ajusta el tamaño de la cabecera para que éste sea múltiplo de 4096 bytes, asegurando que cada bloque de datos esté alineado con los bloques del sistema de archivos. Los promedios de cada bloque están contemplados con dos finalidades de índole gráfica: poder construir de forma inmediata una vista de baja resolución del cubo y para responder rápidamente a peticiones tan grandes que hagan excesivamente largo su tiempo de carga, de manera similar a como trabaja el formato JPEG-progresivo. Opcionalmente, la cabecera puede contener el desplazamiento de cada bloque (medido en bloques), en el archivo de datos, y cualquier otra información que sea de importancia para el tratamiento del cubo principal como índices de valores, paletas de colores, datos de transformaciones de escala, etc.

3.3. Buffering

El otro componente importante de la propuesta es el esquema de manejo de los bloques en memoria. Este esquema está basado en un buffer (caché de bloques), desde donde se atienden las peticiones para el proceso de armado del cubo de visualización. Nótese que este buffer no forma parte del proceso de visualización, sino de la carga ordenada y cacheada de los datos, ya que se actualiza de forma predictiva (también puede implementarse un esquema correctivo para mejorar la calidad de la visualización una vez realizada la petición). Dentro del buffer tendrán prioridad los bloques que se encuentran en el primer plano de visualización, de manera que si se llena y no se ha resuelto la petición por completo, se puede recurrir, para los planos mas lejanos, a los promedios. Este buffer se mantiene indexado de forma de poder determinar las peticiones que deben realizarse al disco, éste índice posee tres campos por cada bloque en el archivo BVR:

- Número del bloque: permite reordenar el índice luego que se han determinado las peticiones.

- Número de petición: por omisión –1, se va incrementando a medida que un bloque es solicitado para dar prioridad a los bloques más cercanos al plano de visualización.
- Apuntador al bloque: es el responsable del direccionamiento al bloque almacenado en memoria, si el bloque no está su valor es –1.

Al llegar una petición al buffer, éste actualiza el segundo elemento del índice para preservar el orden en que se realizan las peticiones. Al finalizar las peticiones, se actualiza con los siguientes pasos:

1. Ordenar ascendentemente las peticiones, dejando al final los bloques no solicitados (cuyo valor por defecto es el máximo valor entero sin signo).
2. Reorganizar hasta el tamaño del buffer por número de bloque, asegurando que las peticiones a disco sean siempre incrementales.
3. Recorrer esta parte recargando y sustituyendo bloques solicitados por los que no lo están, dejando en su sitio los bloques que han sido solicitados y se encuentran en el buffer.
4. Reordenar el resto del vector según el número de peticiones, de forma que se resuelva el mayor número posible de ellas.
5. Reordenar de forma que el índice posea nuevamente acceso directo.

Este algoritmo garantiza la carga ordenada de los datos en el buffer de acuerdo a un criterio que da prioridad a aquellas partes de la visualización en primer plano, siempre buscando no cargar repetidamente el mismo bloque sino reutilizándolos en peticiones subsiguientes.

3.4. Capa de entrada/salida

Esta capa es la que realiza las operaciones de entrada/salida, sólo se encarga de atender las solicitudes realizadas por el componente de manejo del buffer, sin ninguna responsabilidad en el orden ni el momento en que son solicitados los bloques. En principio, esta capa solamente atiende solicitudes de lectura; sin embargo, pueden incluirse operaciones de escritura puesto que el direccionamiento a bloques está implantado. Esta característica sería de utilidad para aplicaciones que realizan cálculos sobre el volumen de datos.

4. Pruebas y Resultados

Para mostrar la efectividad del esquema de almacenamiento y acceso propuesto se desarrolló una aplicación de prueba, la cual puede obtenerse en: http://code.google.com/p/bvrlib/ Además, se tomó un archivo de datos, y se definió un conjunto de trayectorias de acceso que permitieran medir la capacidad de respuesta de las capas de manejo de buffer.

4.1. Aplicación

Como muestra la Figura 2, el desarrollo de la biblioteca y la aplicación que la utiliza están conformados por 5 capas:

Interface. Esta capa contiene todos los elementos que interactúan con el usuario, y aquellos que le permiten realizar peticiones de visualización. Con la finalidad de probar el sistema en un escenario real de visualización 3D dinámica, se ha recurrido a la implementación de un esquema para la navegación dentro del volumen de datos. Este esquema propone una acción de activar y 5 movimientos (Avanzar, Girar Arriba, Girar Abajo, Girar a la izquierda y Girar a la Derecha).

Rendering. Esta capa consta de dos vistas implementadas usando el módulo OpenGL de la librería QT. La primera está destinada a la visualización del sub-cubo y orientada siempre a lo largo de la línea de visión del usuario. La otra capa es una miniatura del volumen entero de datos, junto con una representación del sub-volumen que está siendo mostrado. Se hace uso de la memoria del GPU para alojar las superficies que van a ser visualizadas, llevando la resolución y el número de superficies al límite del almacenamiento de dicha memoria.

También se pueden implementar filtros de visualización, paletas y herramientas de visualización usando elementos geométricos de OpenGL, quedando abierta la posibilidad de ampliación de la aplicación.

Resolve. Permite al algoritmo de visualización realizar peticiones al sistema de almacenamiento. Existen 5 casos de uso: 1.- Un dato puntual del cubo, en este caso el vector de vértices contiene ocho puntos iguales. 2.- Una línea, con 4 juegos de elementos iguales. 3.- Un plano, los 4 primeros valores representan sus límites y los

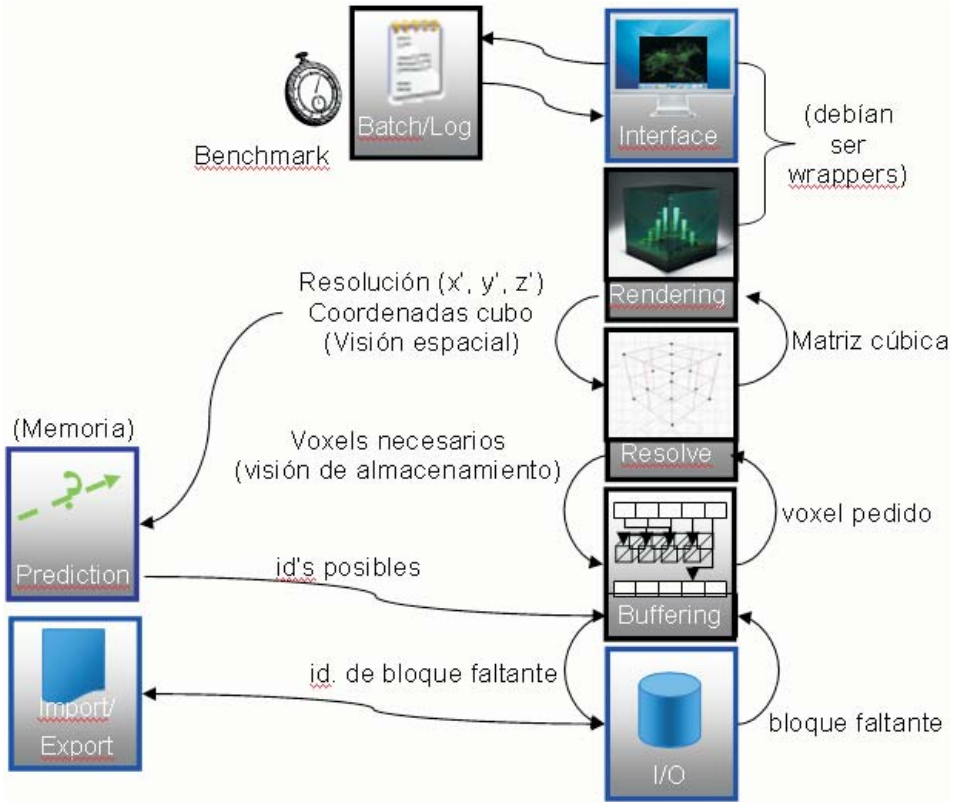

Figura 2. Arquitectura de capas para la implementación propuesta.

otros 4 son iguales. 4.- Un sub-cubo, con cada vértice representado por un punto diferente. 5.- Cualquier otro espacio geométrico delimitado por 8 vértices en el espacio.

Buffering. A partir de este módulo se comienza la implementación de la propuesta, puesto que es desde aquí, y no desde el disco, que se resolverán las peticiones de la capa superior.

I/O. Esta módulo hace el I/O e interactúa directamente con la capa *Buffering*.

4.2. Metodología

La aplicación permite mostrar un sub-volumen arbitrario comprendido entre 8 vértices, editar cada vértice por separado, presentar una pre-visualización de todo el volumen para brindar al usuario su localización dentro del volumen y, almacenar caminos y volverlos a recorrer.

Para comprobar la efectividad del formato propuesto, se realizaron pruebas funcionales y de desempeño. Para este fin, se usó un subconjunto de datos de proyecto Mujer Visible (http://www.nlm.nih.gov/research/visible/visible_human.html), en particular se uso la cabeza, en formato RGB. Este formato se transformó a su representación en BVR.

Para las pruebas funcionales se comprobó la capacidad de respuesta para peticiones inusuales, tales como sub-volúmenes muy grandes, muy pequeños, deformados, vértices intercambiados, o por fuera del cubo de datos. La aplicación respondió de manera satisfactoria, no llegando a bloquearse o terminar inesperadamente. La responsabilidad de hacer peticiones coherentes recae en el usuario, así que la librería se limita a resolver y visualizar lo que ha sido pedido. La Figura 3 muestra una visualización de un subvolumen de la imagen.

En el caso de las pruebas de desempeño se usa un algoritmo de predicción basado en las dos últimas peticiones. Este algoritmo calcula el vector dirección con esas peticiones, supone que será el mismo para la próxima petición y estima la nueva petición. Además, se utilizaron ejecuciones previamente almacenadas, luego de ejecutar

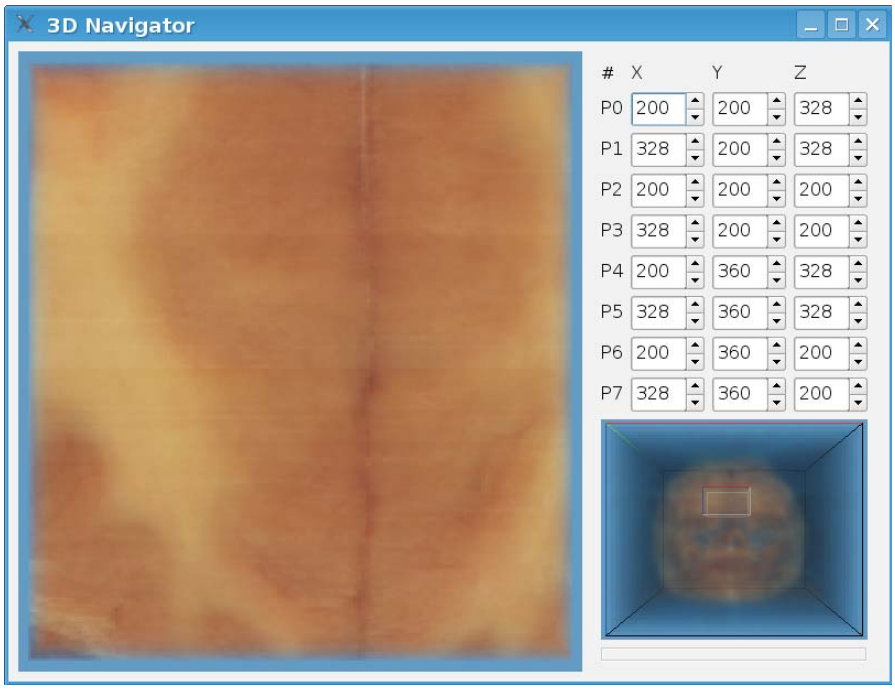

Figura 3. Aplicación mostrando una petición de sub-volumen.

la aplicación con una posición y tamaño de la visualización iniciales, y almacenando la trayectoria descrita. Mediante este mecanismo se almacenaron los conjuntos de peticiones para 4 caminos: **Lineal**, **Suave**, **Brusco** y **Circular**.

Para cada uno de estos caminos, se midió la respuesta del esquema predictivo variando los tamaños de buffer desde 100 hasta 10000 bloques. El patrón seguido por las peticiones, el esquema de actualización del buffer y el tamaño de éste conforman las distintas condiciones de uso que miden el desempeño de la propuesta. Mediante una serie de ejecuciones de la aplicación y usando los caminos almacenados se fueron generando estadísticas de cada una de éstas.

4.3. Resultados

El conjunto de datos usado está compuesto por 797 capas con una resolución de 570x670, cada punto almacenado en su valor RGB dado por tres valores de 8 bits, lo que representa un tamaño de 872 Mb. Estos archivos, una vez convertidos a un archivo ***BVR*** ocupa un tamaño de 1.17 Gb., en nuestro caso este archivo no cabe en memoria principal y es suficiente para las pruebas a realizar. De este tamaño, los primeros 299 bloques de 4 Kb representan la cabecera y los 305520 bloques restantes representan los datos en sí; esto parecería un aumento de espacio de 37.18% respecto al archivo original; sin embargo, el 33% representa la adición de un componente necesario de transparencia (8 bits por cada *voxel*). Eso significa que el incremento debido al cambio de formato es inferior al 5%.

La Figura 4 muestra el porcentaje de éxitos en bloques (el porcentaje promedio de bloques que ya están en el buffer cuando son solicitados) versus el tamaño del buffer (en bloques). Con el esquema utilizado, las solicitudes que tienen una trayectoria lineal son las más fáciles de predecir, mientras que las que hacen movimientos bruscos son las más difíciles y por tanto tienen un porcentaje de éxito más bajo. En estos casos, puede utilizarse algoritmos de predicción más complejos que aumenten el porcentaje de éxito. En cuanto al tamaño del buffer se puede implementar políticas adaptativas según los requisitos de la aplicación y el comportamiento del algoritmo de predicción. Es claro que a mayor tamaño del buffer mayor porcentaje de éxito porque el algoritmo puede cargar mayor cantidad de bloques en cada solicitud a disco.

Como muestra la Figura 5, los resultados son similares si se toma en cuenta el porcentaje

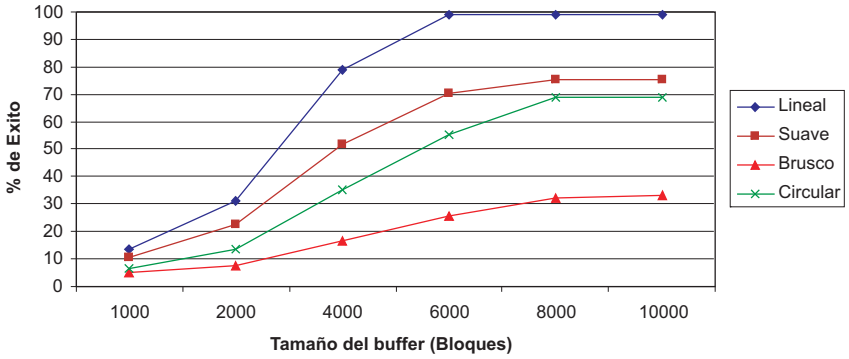

Figura 4. Porcentaje de éxito (Bloques) vs. Tamaño del buffer.

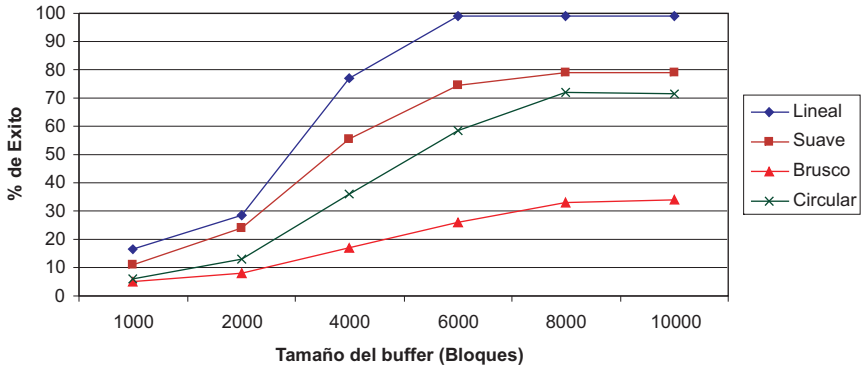

Figura 5. Porcentaje de éxito (*Voxels*) vs. Tamaño del buffer.

de éxitos en *voxels*, a pesar de que pueden solicitarse bloques donde no todos los *voxels* serán parte de la visualización. En esta aplicación cada cubo de visualización tenía aproximadamente 3500 bloques, las Figuras 4 y 5 muestran que con un buffer del doble de este tamaño se pueden lograr buenos porcentajes de éxito. Además, este valor representa un espacio de memoria aceptable para las configuraciones actuales puesto que se usan menos de 30 MB de la memoria principal.

5. Conclusiones

La librería desarrollada es confiable y brinda libertades en las peticiones con trayectorias arbitrarias lo cual representa una capacidad novedosa en herramientas de visualización dinámica de grandes volúmenes de datos. Por otra parte, el uso de información agrupada (promedios) permite dar respuestas en corto tiempo para los bloques que no están en el buffer. La capacidad de ajustar la dimensión del buffer permite que equipos con mayor capacidad de memoria principal tengan mayor cantidad de datos disponibles para garantizar porcentajes de éxito mayores, al mismo tiempo permite que aplicaciones basadas en esta librería puedan ser ejecutadas en equipos de prestaciones limitadas. El esquema predictivo, a pesar de su sencillez, permitió obtener buenos resultados. Representando un buen punto de partida para implantar esquemas más complejos que tengan mejor rendimiento en casos especiales. El modelo basado en capas resultó satisfactorio, puesto que facilita el mantenimiento y depuración, permitiendo la actualización de una capa de manera sencilla, sin afectar las funcionalidades del resto.

Referencias

1. Lehmann T.M, Gonner C., Spitzer K. “Survey: Interpolation Methods in Medical Image Processing”. IEEE Transactions on Medical Imaging. Volume: 18, Issue 11(1999) 1049-1075.
2. MINC: Medical Image NetCDF. http://www.bic.mni.mcgill.ca/software/minc/

3. H. D. Tagare, C. C. Jaffe, and J. Dungan, "Medical Image Databases: A Content-Based Retrieval Approach," Journal of the American Medical Informatics Association 4, pp. 184-198, 1997.
4. Carrion, P.M. "Enhanced Migration of Seismic Data". Geophysical Prospecting, v 38, n 7, Oct. 1990, p 689-704.
5. LaTraille, S., Gettrust, J.; Simpson, M. "ROSE Seismic Data Storage and Exchange". Journal of Geophysical Research, v 87, n B10, Oct 10, 1982, p 8359-8363.
6. B. G. Nickerson, P. A. Judd, L. A. Mayer. "Data Structures for Fast Searching of SEG-Y Seismic Data". Computers & Geosciences. Volume 25, Issue 2 (1999) 179-190.
7. Celniker G., Chakravarty I., Moorman J. "Visualization and Modeling of Geophysical Data". In Proceedings of the 4th conference on Visualization (93). IEEE Computer Society. Ca. PP 362-365.
8. Murillo, A.E.; Bell, J. "Distributed Seismic Unix: A Tool for Seismic Data Processing" Concurrency: Practice and Experience, v 11, n 4, 10 April 1999, p 169-187.
9. Rew R., Davis G. "NetCDF: An interface for scientific data access". IEEE Computer Graphics and Applications, v 10, n 4, Jul, 1990, p 76-82.
10. Elena Pourmal, Mike Folk, Bob McGrath. "HDF Software Process-Lessons Learned". In Proceeding of NOBUGS Conference (2004), October, 18-20, 2004 at PSI, Switzerland.
11. Folk, M.; McGrath, R.E.; Yeager, N. "HDF: An Update and Future Directions" IEEE 1999 International Geoscience and Remote Sensing Symposium. p 273-285 vol. 1.
12. Rew, Russ "Merging NetCDF and HDF5". Bulletin of the American Meteorological Society, Hartnett,. Ed. 84th American Meteorological Society (AMS) Annual Meeting, 2004, p 1457-1460.
13. Lippert L., Gross M.H., Kurmann C. "Compression Domain Volume Rendering for Distributed Environments". Computer Graphics Forum, Volume 16, n 3 Conference, Sep. 1997, pp. 95-107.
14. James E. Fowler, Roni Yagel. "Lossless Compression of Volume Data".In Proceedings of the symposium on Volume visualization (1994). Virginia, United States. PP 43-50.
15. Paul Ning, Lambertus Hesselink. "Fast Volume Rendering of Compressed Data". In Proceedings of the 4th conference on Visualization (93). San Jose, California. PP 11-18.
16. Tzi-cker Chiueh, Chuan-kai Yang, Taosong He, Hanspeter Pfister, Arie Kaufman."Integrated volume compression and visualization". In Proceedings of the 8th conference on Visualization (1997). Phoenix, Arizona, United States. PP 329 - 336.
17. Lukas Mroz. "Real-Time Volume Visualization on Low-End Hardware". PHD-Thesis. Institute of Computer Graphics and Algorithms, Vienna. University of Technology (2001).
18. The Grid DataBlade - A Database Extension for Manipulating Gridded Data. http://www.barrodale.com/grid_Demo/.
19. Boucheny, C.; Bonneau, G.; Droulez, J.; Thibault, G. ; Ploix, S. "A Perceptive Evaluation of Volume Rendering Techniques". ACM Proceedings of Symposium on Applied Perception in Graphics and Visualization, 2007, p 83-90. Germany.
20. William E. Lorensen, Harvey E. Cline. "Marching cubes: A high resolution 3D surface construction algorithm" Computer Graphics (ACM), v 21, n 4, Jul, 1987, p 163-169.
21. Levoy, M. "Display of Surfaces from Volume Data". IEEE Computer Graphics and Applications, v 8, n 3, Feb, 1987.
22. Kruger, J. ; Westermann, R. "Acceleration Techniques for GPU-based Volume Rendering". IEEE Visualization, 2003, p 287-292.

Recibido el 16 de Julio de 2007
En forma revisada el 02 de Junio de 2008