CEIG - Spanish Computer Graphics Conference (2012)
Isabel Navazo and Gustavo Patow (Editors)

High Resolution Medical 3D Data Sets on Mobile Devices and WebGL

J.R. Jiménez and J.M. Noguera

Dpto. de Informática, Escuela Politécnica Superior, Universidad de Jaén
Campus Las Lagunillas, Edificio A3, 23071 Jaén, Spain
{rjimenez, jnoguera}@ujaen.es

Abstract

Nowadays, mobile devices and the web are being used to deliver 3D graphics to mass users. However, applications such as visualization of high resolution medical models are still impossible to handle in such platforms due to texture limitations, mainly the lack of 3D texture support. In this paper we propose a software architecture and a novel texture storage technique that overcome these limitations. In addition, our proposal allows us to adapt existing direct volume rendering techniques based on 3D textures to mobile devices and WebGL. Our experiments demonstrate the feasibility and validity of our proposal to render high resolution volumetric models on both platforms.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introducción

El hardware y las capacidades gráficas de los dispositivos móviles (teléfonos móviles, tabletas, ...) se mejoran continuamente, e investigaciones recientes han demostrado su validez en algoritmos complejos de gráficos por ordenador, incluido el campo de la visualización de volúmenes [MW08, Ver10, Ima11, CSK*11, NJOS12].

Sin embargo, y a pesar de los importantes avances en la computación móvil, sus prestaciones quedan muy por debajo de las ofrecidas por los ordenadores de sobremesa contemporáneos. Ello se debe a que, por definición, estos dispositivos funcionan con baterías, por lo que su hardware y arquitectura están diseñados para favorecer la eficiencia de energía en detrimento de la potencia de cálculo puro. Adicionalmente, la especificación del estándar gráfico de este tipo de dispositivos es OpenGL ES 2.0 [MGS08], que difiere en varios aspectos del estándar correspondiente a ordenadores de sobremesa. En concreto, las texturas 3D estan contempladas como una extensión opcional de OpenGL ES 2.0*, que no está soportada por la mayoría de los dispositivos móviles.

Por estas razones, es un error suponer que se pueden obtener los mismos resultados en dispositivos móviles mediante una traducción literal de los algoritmos clásicos originalmente destinados a funcionar en PCs estándar o en estaciones de trabajo.

Además, los dispositivos móviles más recientes cuentan con pantallas de grandes dimensiones y alta resolución, que requieren grandes modelos volumétricos para lograr la calidad esperada. Por ejemplo, la resolución de la pantalla del nuevo iPad3 supera el estándar Full HD que se utiliza en la mayoría de los monitores y pantallas de televisión. Sin embargo, la capacidad de su GPU y su memoria están todavía limitadas y no permiten visualizar modelos de gran tamaño de manera directa. Esta problemática es también aplicable al caso de WebGL** dado que al estar basado en OpenGL ES 2.0 sufre de sus mismas carencias y además los monitores de los ordenadores de sobremesa suelen ser de un tamaño considerable con lo que la calidad y respuesta esperadas suelen ser elevadas.

En este trabajo, estudiamos en profundidad esta problemática en el contexto de la visualización de volúmenes. Se presenta una propuesta para modelos volumétricos de gran

* GL_OES_texture_3D – http://www.khronos.org/registry/gles/

** http://www.khronos.org/registry/webgl/

DOI: 10.2312/LocalChapterEvents/CEIG/CEIG12/021-029

tamaño con el fin de satisfacer las expectativas de los usuarios en cuanto a la calidad y al rendimiento esperado en los dispositivos actuales y WebGL. Por otra parte, se propone una arquitectura software que permite adaptar los técnicas actuales de visualización de volúmenes basadas en texturas 3D a las plataformas que sólo admiten texturas 2D.

Además, hemos implementado esta solución tanto en móviles como en WebGL, y hemos llevado a cabo una serie de experimentos para probar el comportamiento de estas plataformas en las condiciones de máxima utilización de su capacidad de almacenamiento de volúmenes.

El resto del artículo se organiza del siguiente modo. La Sección 2 presenta los trabajos previos en el contexto de la visualización de volúmenes en dispositivos móviles y WebGL. En la Sección 3, se explica nuestra técnica y la arquitectura software propuesta. En la Sección 4 se evalúan y analizan los resultados obtenidos. Finalmente, en la Sección 5 se indican las conclusiones de este trabajo.

2. Trabajos Previos

En el contexto de la visualización científica y de volúmenes, un volumen se representa como un conjunto de imágenes o lonchas paralelas y uniformemente distribuidas a lo largo del mismo. La Ecuación 1 representa el cálculo del color final de un determinado píxel como la acumulación de los colores y opacidades a lo largo de una línea que atraviesa el volumen [Lev88] :

$$C_\lambda(x) = \sum_{k=0}^{n} c_\lambda(x+r_k)\alpha(x+r_k) \prod_{l=k+1}^{n} (1-\alpha(x+r_l)) \quad (1)$$

donde $C_\lambda(x)$ es el color final en la posición x para una longitud de onda λ, $c(x+r_k)$ es el color de la muestra k-ésima en la posición $x+r_k$ dentro del volumen y $a(x+r_k)$ su opacidad.

Es relativamente reciente el uso de dispositivos móviles para algoritmos de visualización de volúmenes. Las primeras soluciones evitaban el cálculo de la visualización en estos dispositivos utilizando una arquitectura cliente-servidor. Esta arquitectura consta de un servidor dedicado a la visualización y un cliente en el que se muestran las imágenes transmitidas a través de la red [LS05, JK07]. También siguiendo un esquema cliente-servidor, Zhou et al. [ZQWyC06] utilizan un servidor remoto para precalcular isosuperficies, que se envían al dispositivo móvil y permiten una visualización más eficiente. Moser y Weiskopf [MW08] introdujeron una técnica interactiva para la visualización de volúmenes en dispositivos móviles basada en un conjunto de imágenes guardado en una textura 2D. Noguera et al. [NJOS12] proponen un algoritmo que obtiene resultados interactivos mediante el almacenamiento en caché de la geometría de las lonchas en un búfer de vértices (VBO).

ImageVis3D [Ima11] es una aplicación iOS que utiliza el enfoque de texturas 2D [Wei07]. Cuando el usuario está interactuando, el número de lonchas se reduce drásticamente. Cuando la interacción finaliza, se genera una imagen de mayor calidad empleando el conjunto total de lonchas. El proceso de visualización se puede realizar tanto en el dispositivo móvil como en un servidor externo en caso de tratar con modelos grandes o complejos. Campoalegre [Ver10] también propone un esquema cliente-servidor principalmente para la visualización de estructuras óseas. En este caso el modelo se comprime en el servidor mediante una función *wavelet* Haar y se reconstruye en el cliente. Por otro lado, Congote et al. [CSK*11] implementaron una técnica basada en trazado de rayos para el estándar WebGL.

Todas las técnicas mencionadas anteriormente comparten la misma limitación: la falta de texturas 3D en OpenGL ES 2.0 y WebGL, lo que restringe en gran medida el tamaño y la resolución de los modelos volumétricos que pueden ser utilizados. El problema se agrava en WebGL dado que estas aplicaciones suelen ejecutarse en ordenadores de sobremesa equipados con monitores de gran tamaño. Entre los citados trabajos, cabe destacar que [NJOS12], [CSK*11], pese a que se centran en describir técnicas de visualización directa de volúmenes, resuelven parcialmente la citada limitación almacenando las sucesivas lonchas en una única textura 2D con una configuración de mosaico. En el presente trabajo nos centramos exclusivamente en resolver la limitación del tamaño del modelo volumétrico de una manera más genérica y efectiva, aplicable a cualquier método de visualización de volúmenes.

Las siguientes propuestas hacen referencia al problema de los modelos volumétricos de gran resolución pero en el contexto de ordenadores de sobremesa o de estaciones de trabajo, cuyas características difieren de las de los dispositivos móviles y WebGL. Gunthe y Straßer [GS01] utilizaron una compresión *wavelet* con el fin de procesar este tipo de volúmenes de modo interactivo en un PC estándar. Tomandl et al. [THRS*01] proponen una visualización 3D cliente-servidor híbrida combinando una parte local y otra remota para conseguir una visualización 3D rápida pero de alta calidad. Schneider y Westermann [SW03] también solucionan el problema de la limitación de memoria de textura mediante la compresión a gran escala de conjuntos de datos volumétricos. Su solución tiene en cuenta la coherencia temporal en entornos animados. Thelen et al. [TMEH11] introdujeron un sistema dinámico de multi-resolución basado en *wavelets* para visualizar de manera interactiva conjuntos de datos con varios gigabytes distribuidos en varios clusters. Por último, Xie et al. [XY08] subdividieron el conjunto de datos volumétricos en un conjunto de bloques de tamaño uniforme combinado con la técnica de terminación rayos de manera temprana.

3. Metodología

Esta sección explica el algoritmo y la arquitectura software que proponemos para visualizar volúmenes 3D de grandes dimensiones en dispositivos móviles y WebGL. Además, se proporcionan detalles de la implementación.

Normalmente, los volúmenes se almacenan como un conjunto de lonchas, donde cada una de ellas contiene una imagen 2D que representa la intersección de dicha loncha con el volumen. Las técnicas más usuales guardan estas lonchas en texturas 3D. Lamentablemente, ni los dispositivos móviles ni WebGL soportan texturas 3D. Esta limitación se puede evitar guardando las lonchas en una textura 2D en configuración de mosaico [CSK*11, NJOS12]. Hemos de observar que, normalmente, las texturas 2D tienen un menor número de téxeles que las texturas 3D. Por tanto, el tamaño del volumen que es posible almacenar es menor.

En este trabajo extendemos la configuración en mosaico hasta explotar la máxima capacidad de textura de la GPU para poder manejar volúmenes más grandes. Nuestra idea se basa en maximizar la capacidad de almacenamiento utilizando todas las unidades de textura disponibles (multitexturas) así como todos los canales de color. Usualmente, la capacidad de los actuales dispositivos es de 8 texturas RGBA de 2048^2 téxeles. Estas dimensiones nos permiten visualizar un volumen con un tamaño máximo de 512^3 vóxeles usando nuestra técnica, lo que supone un tamaño considerablemente mayor a lo utilizado hasta ahora en plataformas móviles.

La técnica propuesta consiste en considerar los cuatro canales de color RGBA de una textura como cuatro texturas independientes de un solo canal. Para cada uno de los canales, y comenzando por el primero, se almacenan las sucesivas lonchas del volumen 3D una al lado de la otra en una configuración de mosaico. Una vez el canal de la textura se completa, se comienza con el siguiente. Si se completan los cuatro canales de color, se continúa con la siguiente unidad de textura, y así sucesivamente, hasta almacenar el volumen en su totalidad o hasta alcanzar la máxima capacidad del dispositivo.

La Figura 1 muestra un ejemplo de textura 2D donde cada canal de color guarda un subconjunto del total de lonchas en forma de mosaico. De este modo, cada color RGBA representa los valores en 4 lonchas no consecutivas del volumen.

La técnica propuesta de almacenaje en textura puede combinarse con cualquier algoritmo estándar de visualización de volúmenes basado en texturas 3D. La Figura 2 ilustra nuestra propuesta de arquitectura para algoritmos de visualización de volúmenes diseñados para OpenGL ES 2.0. Esta arquitectura se divide en dos partes: la *memoria de textura* y el *shader*. La memoria de textura se usa para almacenar tanto el modelo 3D como la función de transferencia. El volumen se guarda usando multitexturas como se ha descrito anteriormente. La función de trasferencia, por otro lado, es una textura empleada habitualmente por las técnicas de visualización de volúmenes a fin de asignar un color a cada vóxel [Wei07]. De este modo, es posible destacar diferentes elementos del modelo (huesos, músculos, etc.) modificando de manera interactiva dicha función de transferencia.

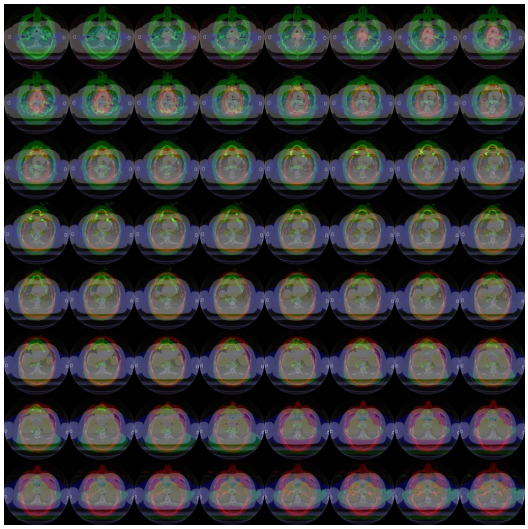

Figura 1: *Textura RGBA donde se almacenan 4 mosaicos.*

En la parte del shader de la Figura 2 se pueden ver dos módulos: el Adaptador de Coordenadas de Textura (ACT) y la técnica Directa de Visualización de Volúmenes (*Direct Volume Rendering*, DVR) elegida.

Las técnicas DVR habitualmente emplean coordenadas de textura 3D para acceder directamente al modelo volumétrico almacenado en la textura a fin de dibujarlo. Nuestro módulo ACT actúa de mediador entre la técnica de DVR y la representación del modelo en textura. De esta forma, la técnica de DVR ignora cómo se almacena el modelo. Simplemente proporciona las coordenadas de textura 3D al módulo ACT y éste accede a la textura y le devuelve el tono de gris correspondiente. Este tono se convierte en un color RGBA de acuerdo con la función de transferencia, con el que se calcula el color del fragmento correspondiente. Dado que la técnica DVR es independiente del método de almacenamiento del modelo, nuestra arquitectura proporciona un mecanismo eficaz para adaptar técnicas existentes de visualización de volúmenes basadas en texturas 3D a plataformas que solo soporten texturas 2D.

El módulo ACT transforma las coordenadas de textura 3D suministradas por la técnica DVR en un formato entendible según la organización del volumen y calcula el tono de gris correspondiente. El valor devuelto incluye la interpolación trilineal con los vecinos más cercanos del volumen. El proceso llevado a cabo por este módulo se descompone en dos tareas.

La primera tarea calcula la unidad de textura u_1, el ca-

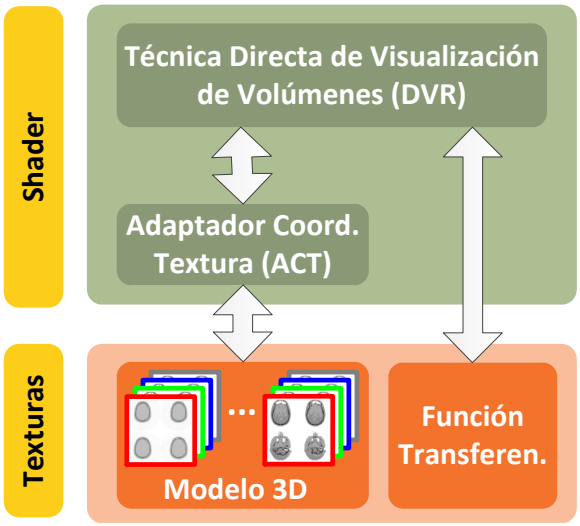

Figura 2: *Arquitectura software propuesta.*

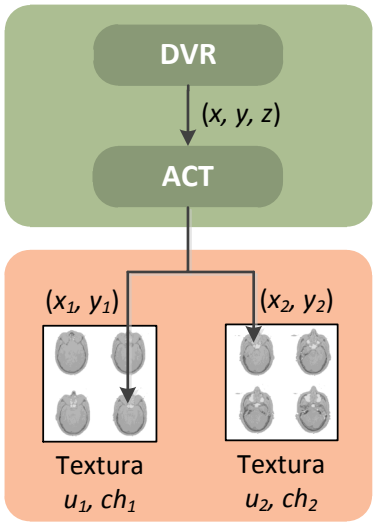

Figura 3: *Ejemplo de cálculo de coordenadas de textura 2D de dos téxeles vecinos en mosaicos consecutivos.*

nal de color ch_1 y las coordenadas de textura 3D locales (x, y, z_{res}) a partir de las coordenadas originales (x, y, z). El Listado 1 muestra cómo calcular estos valores. S es el número de lonchas en que se divide el volumen, $M_x \times M_y$ es el número máximo de lonchas en un mosaico guardado en un canal de color y unidad de textura determinados, ver Figura 1, y Max_{ch} es el número de canales de color por textura. Como las lonchas se almacenan de forma consecutiva a lo largo de canales de color y unidades de textura, cada mosaico almacena un intervalo de lonchas de la componente z. El valor z_{res} es el z residual definido como la componente z de las coordenadas de textura (x, y, z) proporcionadas por la técnica DVR menos la componente z del vóxel almacenado en el primer téxel del mosaico elegido.

La segunda tarea del módulo ACT es utilizado para calcular un par de 4-tuplas (u_1, ch_1, x_1, y_1), (u_2, ch_2, x_2, y_2) que definen dos téxeles del conjunto de texturas 2D, donde u_i, ch_i hacen referencia a la unidad de textura y al canal de color, respectivamente, y x_i, y_i representan la coordenada de textura 2D del téxel elegido en el mosaico correspondiente guardado en la unidad de textura u_i, véase Figura 3. Estos dos téxeles son vecinos en la dirección z y se sitúan en el mismo o en mosaicos consecutivos. Los tonos de gris correspondientes a los téxeles indicados se combinan simulando una interpolación trilineal. Hemos de observar que la interpolación bilineal se obtiene de manera automática a partir del propio interpolador implementado en la GPU. El Listado 2 muestra el algoritmo que desarrolla estas operaciones. Aquí, T es un vector de texturas 2D que almacenan el modelo volumétrico en su totalidad.

Los shaders representados en los Listados 1 y 2 han sido cuidadosamente diseñados para minimizar las operaciones de control de flujo y maximizar el uso de funciones propias de GLSL como por ejemplo, la función *step*. Las operaciones de control de flujo tienen un coste considerable en la GPU de los dispositivos móviles.

4. Resultados

Se han implementado dos prototipos con el fin de medir la eficacia y el rendimiento de nuestra técnica. El primero de ellos es una aplicación móvil basada en OpenGL ES 2.0, mientras que el segundo es una aplicación de escritorio que utiliza WebGL. La técnica empleada para el módulo DVR es un trazado de rayos ejecutado enteramente en la GPU [KW03, HLSR09]. En resumen, esta técnica consiste en un bucle que se ejecuta en el procesador de fragmentos. El bucle itera n pasos, recorriendo el volumen a lo largo de la dirección de un rayo dado. A lo largo de esta iteración se acumula tanto el color como la opacidad del volumen, y estos valores acumulados se emplean para colorear al fragmento. Se utiliza una estrategia de terminación temprana de rayos para ahorrar cómputo cuando la opacidad acumulada alcanza el máximo permitido.

Recordemos que nuestra arquitectura es independiente de la técnica de visualización utilizada. Por tanto, podríamos haber seleccionado alguna técnica basada en texturas, que suelen proporcionar mayor rendimiento que las basadas en trazado de rayos [NJOS12]. No obstante, nuestro objetivo en este trabajo no es medir el rendimiento de la técnica de DVR en la que nos basamos, sino estudiar el rendimiento y escalabilidad al incrementar la resolución del modelo 3D y al utilizar múltiples texturas.

En nuestros experimentos hemos utilizado el conjunto de datos *CTHuman*, proporcionados por el *Visible Human Pro-*

```
//vector de texturas con el modelo
uniform sampler2D T[6];

//número total de lonchas del modelo
uniform float S;

//dimensiones de cada mosaico
uniform float Mx;
uniform float My;

//número máximo de canales de color
float Maxch = 4.0; //RGBA

float ACT(vec3 p)
{
  float Mxy = Mx * My;
  float aux1 = floor( S * p.z );

  //primera loncha de la que leer
  float aux2 = floor( aux1 / Mxy );

  //porcentaje de Z dentro de cada textura
  float depth = min( 1.0, Mxy / S );

  //Z del primer texel de la loncha a leer
  float zini = aux2 * depth;

  //Z residual de p dentro del mosaico actual
  float zres = (p.z - zini) / depth;

  int u1 = int( floor(aux2/Maxch) );
  int ch1 = int( mod(aux2, Maxch) );

  vec3 p2 = vec3( p.xy, zres );

  return getValue( p2, ch1, u1 );
}
```

Listado 1: *Cálculo de la unidad de textura u_1, el canal de color ch_1 y las correspondientes coordenadas de textura $p2(x,y,z_{res})$ a partir de las coordenadas 3D originales $p(x,y,z)$.*

```
float getValue( vec3 p2, int ch1, int u1 )
{
  float Stex = min( Mx * My, S );

  float aux3 = floor( p2.z * Stex );
  float aux4 = mod( aux3+1.0, Stex );

  vec2 texPos1, texPos2;

  float texPos1.x = aux3/Mx + p2.x/Mx;
  float texPos1.y = floor(aux3/My)/My + p2.y/My;

  float texPos2.x = fract(aux4/Mx) + p2.x/Mx;
  float texPos2.y = floor(aux4/My)/My + p2.y/My;

  //control desbordamiento 2º acceso textura
  float next = float( ch1+int(step(aux4,aux3)) );

  int u2 = u1 + int( step(Maxch, next) );
  int ch2 = int( mod(next,Maxch) );

  //leer colores con interpolacion bilineal
  float v1 = texture2D( T[u1], texPos1 )[ch1];
  float v2 = texture2D( T[u2], texPos2 )[ch2];

  //simular interpolacion trilineal
  return mix( v1, v2, (p2.z*Stex)-aux3 );
}
```

Listado 2: *Cálculo del tono de gris de la posición $p2(x,y,z_{res})$.*

ject[***]. Este conjunto de datos tiene una resolución total de $512^2 \times 1877$ vóxeles.

4.1. Resultados en Dispositivos Móviles

Los experimentos se llevaron a cabo en una tableta *iPad2*. Este dispositivo dispone de una GPU PowerVR SGX543MP2 de doble núcleo, así como el sistema operativo iOS 5. La aplicación utilizada en nuestra experimentación se desarrolló como un software nativo, escrito en C++ y GLSL ES 2.0. Según las especificaciones técnicas de Apple, este dispositivo soporta texturas 2D con un tamaño máximo de 2048^2 téxeles y dispone de ocho unidades de textura.

Los experimentos se han diseñado con el objetivo de cubrir todo el rango de resoluciones posibles soportadas por nuestra técnica. Para ello, hemos utilizado un subconjunto del modelo CTHuman, ilustrado en la Figura 4, a diferente resolución. Los experimentos consistieron en generar una animación de la cámara rotando alrededor del modelo. Cada animación constó de 100 fotogramas, y se tomó el tiempo requerido por el dispositivo móvil para generar cada uno de dichos fotogramas.

La GPU del dispositivo utiliza una arquitectura conocida como "visualización diferida basada en celdas" (TBDR, o "*tile-based deferred rendering*") [Pow11], que provoca que la ejecución de las llamadas a OpenGL ES puedan retrasarse hasta que la escena se presenta en pantalla. Para garantizar unas mediciones de tiempo exactas, en nuestros experimentos forzamos a que el dibujado se terminara completamente mediante una llamada a *glFinish*.

Dado que nuestro interés no es evaluar el rendimiento de la técnica DVR de rayos, se fijaron unos parámetros para la

*** http://www.nlm.nih.gov/research/visible/visible_human.html

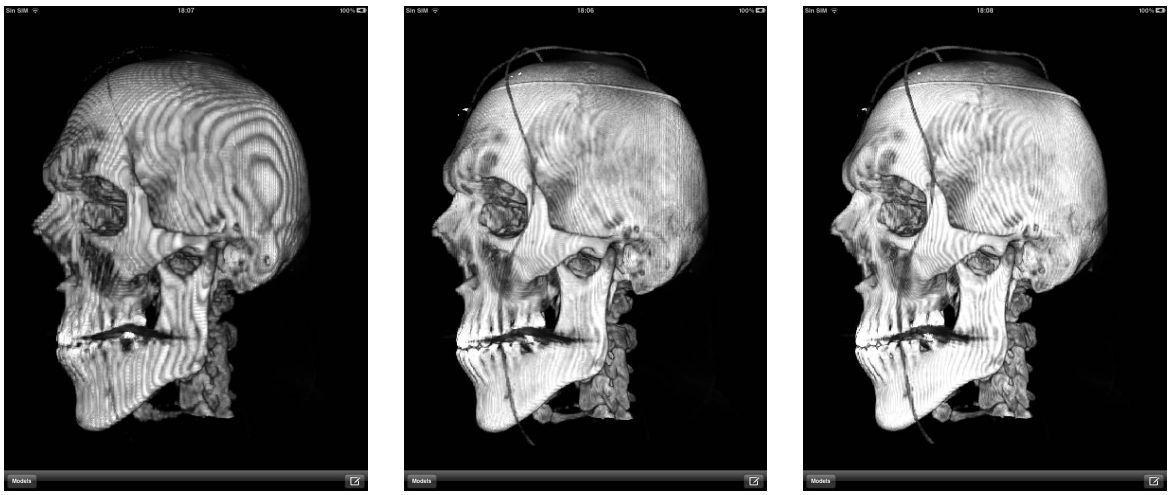

Figura 4: *El modelo CTHuman en un dispositivo móvil iPad2. a)* $128^3 A$*, b)* $256^3 A$*, c)* $512^2 \times 384$*.*

misma que permanecieron constantes durante todos los experimentos. Concretamente, el trazado de rayos se configuró para efectuar 128 pasos, y la resolución de pantalla fue de 480×320 píxeles. La función de transferencia también fue constante, tal y como se ilustra en la Figura 4.

La Figura 5 muestra gráficamente los tiempos medios en milisegundos requeridos para dibujar cada imagen. En todos los casos se han empleado texturas de 2048^2 píxeles sin compresión para almacenar el modelo. Se incluyen resultados para los siguientes experimentos:

- 128^3A: almacenado usando una textura con un único canal.
- 128^3B: mismo modelo que en experimento anterior.
- 256^3A: almacenado usando una textura RGBA.
- 256^3B: almacenado usando cuatro texturas de un único canal.
- $512^2 \times 384$: almacenado usando seis texturas RGBA.

Para el experimento 128^3A se utilizó una versión simplificada del módulo ACT que solamente era capaz de manejar un mosaico. Esta versión simplificada es similar a la propuesta de Congote et al. [CSK*11]. En cambio, en el resto de experimentos se utilizó nuestra propuesta de módulo ACT, la cual puede gestionar modelos 3D de mayor resolución.

Hay que reseñar que la máxima resolución que se puede conseguir con nuestra técnica de ACT en este dispositivo es de $512^2 \times 384$ vóxeles. Ello se debe a que una unidad de textura tiene que ocuparse con la función de trasferencia, mientras que otra es requerida por la técnica de visualización de trazado de rayos.

Los resultados de la Figura 5 revelan que los tiempos de dibujado son casi constantes en todos los experimentos que comparten el mismo módulo ACT. Esto sugiere que la resolución del modelo volumétrico no tiene un impacto significativo en el rendimiento del proceso de dibujado, siempre y cuando el modelo quepa en la memoria del dispositivo.

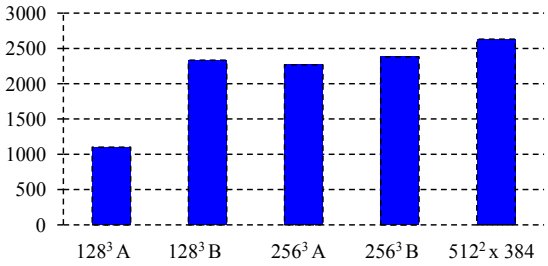

Figura 5: *Tiempos de dibujado (ms) para el dispositivo móvil iPad2. Resolución de pantalla: 320×480 píxeles. Trazado de rayos: 128 pasos.*

Tal y como se dijo con anterioridad, la pareja de experimentos 128^3A y 128^3B comparten el mismo conjunto de datos, pero difieren en el modulo ACT utilizado. Concretamente, en el experimento 128^3B se utilizó nuestra técnica para calcular las coordenadas de textura, mientras que en 128^3A se recurrió a la versión simplificada. Fue posible utilizar esta versión simplificada porque el modelo es lo suficientemente pequeño como para caber en un único mosaico. Tal y como se muestra en la Figura 5, los tiempos de dibujado del segundo experimento fueron el doble de largos que los del primero. Nótese que nuestra técnica para calcular las coordenadas de textura no incrementa el número de accesos a textura ni tampoco añade instrucciones de control al shader. No obstante, sí que incrementa su longitud en

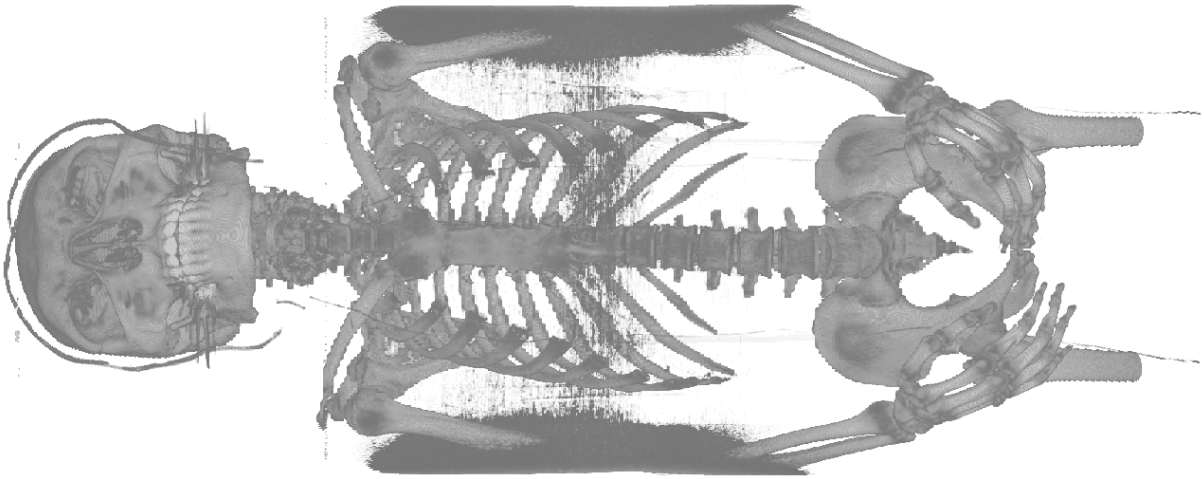

Figura 6: *Modelo CTHuman dibujado en WebGL, resolución:* $512^2 \times 1024$ *vóxeles.*

aproximadamente una docena de líneas, que son necesarias para gestionar múltiples canales de color y unidades de textura, ver Listados 1 y 2. Pese a que estas líneas solo efectúan cómputos simples, su ejecución se repite una vez por cada paso efectuado por el algoritmo de trazado de rayos, lo que explica el incremento en el tiempo de dibujado.

Los experimentos 256^3A y 256^3B utilizan un conjunto de datos cuyo tamaño es excesivo para un mosaico de 2048^2 píxeles. El almacenamiento de dicho modelo puede abordarse mediante dos estrategias diferentes: o bien usar una textura RGBA (256^3A), o bien usar cuatro texturas de un único canal (256^3B). Los resultados muestran que no hay ninguna diferencia significativa entre ambas estrategias en términos de velocidad de visualización. Por tanto, podemos seleccionar aquella que más convenga a nuestra aplicación particular.

Para terminar, también hemos confirmado que nuestra técnica permite maximizar el uso de los recursos de textura disponibles en el dispositivo móvil a fin de dibujar modelos muy grandes. En concreto, hemos sido capaces de visualizar modelos de hasta $512^2 \times 384$ vóxeles sin con ello sacrificar velocidad de dibujado.

4.2. Resultados en WebGL

A continuación, realizamos un conjunto de experimentos similar para estudiar la eficiencia de nuestra solución en WebGL. Los experimentos se ejecutaron en un PC equipado con Intel Core2 Quad CPU Q6600, 4 GB de RAM, una GeForce 8800GT y Windows 7 SP1 64 bits. El navegador web empleado fue el Opera 11.50 labs (build 24661).

La GPU que hemos utilizado soporta texturas 2D de hasta 8192^2 téxeles y equipa 16 unidades de texturas. Dado que el tamaño de textura es considerablemente superior al proporcionado por el iPad2, hemos utilizado un subconjunto de mayor tamaño del modelo CTHuman. Este subconjunto se muestra en la Figura 6.

Para poder medir los tiempos de dibujado, tuvimos que forzar a la ventana de WebGL para que se redibujara constantemente. Entonces generamos una animación consistente en rotar la cámara alrededor del modelo durante 5 segundos, y se contó cuántos fotogramas se dibujaron durante dicho intervalo de tiempo. La Figura 7 representa gráficamente los tiempos medios requeridos para dibujar cada fotograma en milisegundos. La Figura incluye resultados para los siguientes experimentos:

- $512^2 \times 256$A: almacenado usando una textura de un único canal.
- $512^2 \times 256$B: mismo modelo que en experimento anterior.
- $512^2 \times 1024$: almacenado usando una textura RGBA.

El experimento $512^2 \times 256$A utilizó la versión simplificada del módulo ACT descrita en la Sección 4.1, mientras que los experimentos $512^2 \times 256$B y $512^2 \times 1024$ emplearon nuestro módulo ACT propuesto.

En este caso, se emplearon texturas sin comprimir de 8192^2 téxeles, y el trazado de rayos efectuó 128 pasos. La resolución de la ventana de WebGL fue de 800^2 píxeles.

Durante nuestros experimentos hemos encontrado algunas limitaciones inesperadas. Por ejemplo, cada vez que intentábamos cargar más de una textura de 8192^2 téxeles, el navegador Opera dejaba de responder y generaba un mensaje de error. La razón argumentada por el navegador era que el controlador gráfico de NVIDIA estaba excediendo el límite de tiempo de dibujado impuesto por Windows, fijado en dos segundos. Esto limitó nuestra experimentación a un modelo de $512^2 \times 1024$ vóxeles, pues éste es el mayor tamaño que podemos codificar en una sola textura RGBA de 8192^2 téxeles.

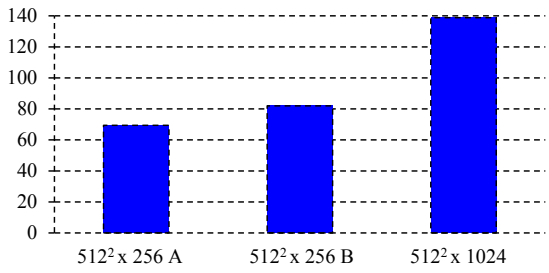

Figura 7: *Tiempos de dibujado (ms) para un PC de escritorio con una GPU NVIDIA Geforce 8800GT. Resolución de pantalla:* 800^2 *píxeles. Trazado de rayos: 128 pasos.*

Es interesante destacar que no hemos encontrado este tipo de problemas cuando estábamos experimentando con dispositivos móviles, a pesar de que ambas plataformas comparten la misma especificación de OpenGL ES 2.0. Según nuestra experiencia, las implementaciones de WebGL están todavía relativamente inmaduras. De hecho, el dispositivo móvil empleado en nuestros experimentos ha resultado ser una plataforma mucho más predecible y estable. Al contrario que la implementación WebGL de Opera, el iPad2 fue capaz de completar todos nuestros experimentos, incluyendo aquellos que explotaban todos los recursos de textura disponibles.

A pesar de estas limitaciones, una comparación directa entre las Figuras 5 y 7 permite apreciar que la velocidad de WebGL es más de un orden de magnitud superior a la del iPad2 al visualizar modelos volumétricos de dimensiones similares. Pese a las limitaciones de WebGL como plataforma software, el hardware subyacente es mucho más poderoso que el incluido en un dispositivo móvil de alta gama. Téngase en cuenta además que el PC utilizado en nuestros experimentos tiene unas especificaciones técnicas discretas.

En los experimentos $512^2 \times 256$A y $512^2 \times 256$B (ver Figura 7), se utilizó un modelo volumétrico que cabe en un solo mosaico. Por tanto, nuestra propuesta de módulo ACT no es estrictamente necesaria para un modelo tan pequeño. Como dijimos anteriormente, estos dos experimentos difieren entre sí en el módulo ACT. La diferencia en tiempos entre ambos experimentos muestra el coste de incluir nuestro módulo para gestionar modelos grandes. Los resultados muestran claramente que dicha diferencia en tiempos es muy pequeña. Es decir, la GPU es capaz de procesar las operaciones adicionales sin degradar apreciablemente la velocidad de dibujado.

El último experimento ($512^2 \times 1024$) utiliza un modelo que es demasiado grande para almacenarse en un mosaico de la forma convencional. Por tanto, es obligatorio usar nuestro módulo ACT para dibujarlo. En este caso, se utilizó los cuatro canales de color de una textura RGBA. Los resultados muestran un incremento en el tiempo de dibujado en comparación con los experimentos anteriores. Este resultado es inesperado, puesto que tanto las dimensiones de la textura, como los números de accesos a la misma y de operaciones son los mismos. De hecho, la única diferencia es el número de canales de color. Por tanto, la causa parece ir relacionada con un aumento de los fallos en la caché de la memoria de textura.

5. Conclusiones

Pese al increíble aumento en prestaciones que han experimentado los dispositivos móviles durante los últimos años, sus capacidades gráficas se encuentran muy por debajo de los ordenadores de escritorio. El limitado almacenaje de texturas ha imposibilitado visualizar modelos volumétricos de calidad. Por ejemplo, en los teléfonos y tabletas de Apple, el tamaño máximo que hasta ahora era posible representar en 3D era de 128^3 vóxeles. Este tamaño resulta totalmente insuficiente para mostrar imágenes médicas con la calidad requerida.

En este artículo hemos propuesto una técnica novedosa que permite solventar esta limitación, permitiendo visualizar volúmenes de mucha mayor resolución en este tipo de dispositivos. Nuestra técnica se basa en utilizar las capacidades multi-textura de la GPU para codificar modelos volumétricos mediante un conjunto de texturas 2D RGBA. La técnica propuesta es también muy adecuada para WebGL, dado que este estándar comparte las mismas limitaciones que los dispositivos móviles, principalmente la carencia de texturas 3D.

También hemos propuesto una arquitectura sencilla y fácil de implementar que permite adaptar técnicas ya existentes de visualización directa de volúmenes basadas en texturas 3D a plataformas que carezcan de dichas texturas, como los dispositivos móviles y WebGL. Nuestros experimentos demuestran que es posible visualizar volúmenes de hasta $512^3 \times 384$ vóxeles en un dispositivo móvil sin penalizar la velocidad de dibujado.

Como trabajos futuros, nos planteamos estudiar el uso de técnicas de compresión de texturas para reducir los problemas de caché y mejorar la eficiencia. También tenemos previsto estudiar los problemas de rendimiento que puedan surgir al trasmitir grandes volúmenes a través de Internet con WebGL. Queremos explorar el uso de técnicas multirresolución que permitan optimizar el consumo de ancho de banda. Creemos que técnicas de refinamiento progresivo pueden resultar muy interesantes en este contexto para mejorar la experiencia del usuario al interactuar con el modelo.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el Ministerio de Ciencia e Innovación del Reino de España y la Unión Europa (fondos FEDER) mediante el proyecto de investigación TIN2011-25259; por la Consejería de Innovación, Ciencia y Empresa de la Junta de Andalucía y la Unión

Europea (fondos FEDER) mediante el proyecto de investigación P07-TIC-02773; y por la Universidad de Jaén mediante el proyecto PID441012.

Referencias

[CSK*11] CONGOTE J., SEGURA A., KABONGO L., MORENO A., POSADA J., RUIZ O.: Interactive visualization of volumetric data with WebGL in real-time. In *Proceedings of the 16th International Conference on 3D Web Technology* (New York, NY, USA, 2011), Web3D '11, ACM, pp. 137–146. URL: http://doi.acm.org/10.1145/2010425.2010449, doi:http://doi.acm.org/10.1145/2010425.2010449.

[GS01] GUTHE S., STRASSER W.: Real-time decompression and visualization of animated volume data. *Proceedings of the IEEE Visualization Conference* (2001), 349–356.

[HLSR09] HADWIGER M., LJUNG P., SALAMA C. R., ROPINSKI T.: Advanced illumination techniques for GPU-based volume raycasting. In *ACM SIGGRAPH 2009 Courses* (New York, NY, USA, 2009), SIGGRAPH '09, ACM, pp. 2:1–2:166. URL: http://doi.acm.org/10.1145/1667239.1667241, doi:10.1145/1667239.1667241.

[Ima11] IMAGEVIS3D: ImageVis3D: A real-time volume rendering tool for large data. scientific computing and imaging institute (sci), 2011. [accessed 29 September 2011]. URL: http://www.imagevis3d.org.

[JK07] JEONG S., KAUFMAN A. E.: Interactive wireless virtual colonoscopy. *The Visual Computer 23*, 8 (2007), 545–557. doi:http://dx.doi.org/10.1007/s00371-007-0117-8.

[KW03] KRUGER J., WESTERMANN R.: Acceleration techniques for GPU-based volume rendering. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (Washington, DC, USA, 2003), VIS '03, IEEE Computer Society, pp. 38–. URL: http://dx.doi.org/10.1109/VIS.2003.10001, doi:10.1109/VIS.2003.10001.

[Lev88] LEVOY M.: Display of surfaces from volume data. *IEEE Comput. Graph. Appl. 8* (May 1988), 29–37. URL: http://dl.acm.org/citation.cfm?id=44650.44652, doi:10.1109/38.511.

[LS05] LAMBERTI F., SANNA A.: A solution for displaying medical data models on mobile devices. In *SEPADS'05* (Stevens Point, Wisconsin, USA, 2005), World Scientific and Engineering Academy and Society (WSEAS), pp. 1–7.

[MGS08] MUNSHI A., GINSBURG D., SHREINER D.: *OpenGL(R) ES 2.0 Programming Guide*, 1 ed. Addison-Wesley Professional, 2008.

[MW08] MOSER M., WEISKOPF D.: Interactive Volume Rendering on Mobile Devices. In *Workshop on Vision, Modelling, and Visualization VMV '08* (2008), pp. 217–226.

[NJOS12] NOGUERA J., JIMÉNEZ J., OGÁYAR C., SEGURA R.: Volume rendering strategies on mobile devices. In *International Conference on Computer Graphics Theory and Applications (GRAPP 2012). Rome (Italy)* (2012), pp. 447–452.

[Pow11] POWER VR: PowerVR Series5 Graphics SGX architecture guide for developers, 2011.

[SW03] SCHNEIDER J., WESTERMANN R.: Compression domain volume rendering. *Proceedings of the IEEE Visualization Conference* (2003), 293–300.

[THRS*01] TOMANDL B., HASTREITER P., REZK-SALAMA C., ENGEL K., ERTL T., HUK W., NARAGHI R., GANSLANDT O., NIMSKY C., EBERHARDT K.: Local and remote visualization techniques for interactive direct volume rendering in neuroradiology. *Radiographics 21*, 6 (2001), 1561–1572.

[TMEH11] THELEN S., MEYER J., EBERT A., HAGEN H.: Giga-scale multiresolution volume rendering on distributed display clusters. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 6431 LNCS* (2011), 142–162.

[Ver10] VERA L. C.: *Volumetric medical images visualization on mobile devices*. Master's thesis, Polytechnic University of Catalonia, 2010.

[Wei07] WEISKOPF D.: *GPU-based interactive visualization techniques*. Mathematics and visualization. Springer, 2007.

[XY08] XIE K., YANG J., ZHU Y.: Real-time visualization of large volume datasets on standard pc hardware. *Computer Methods and Programs in Biomedicine 90*, 2 (2008), 117–123.

[ZQWyC06] ZHOU H., QU H., WU Y., YUEN CHAN M.: Volume visualization on mobile devices. In *14th Pacific Conference on Computer Graphics and Applications* (2006), pp. 76–84.