Título VExplore - Videojuego de exploración 3d de imágenes médicas

Tipo de Producto Material didáctico

Autores Banquiero, Mariano y Ortiz, Edgar

Código del Proyecto y Título del Proyecto

A16T11 - Realtime Voxel Rendering

Responsable del Proyecto

Banquiero, Mariano

Línea

Automatización y Procesamiento de Imágenes

Área Temática

Realtime Rendering- Computer Graphics

Fecha

2016

INTEC
Instituto de Tecnología

UADE
FUNDACIÓN UADE

Caso de Estudio

VExplore - Videojuego de exploración 3D de imágenes médicas

1. Introducción

El presente trabajo está orientado al desarrollo de un videojuego de exploración de modelos 3D creados a partir de imágenes médicas. Dichas imágenes están compuestas por una gran cantidad de elementos tridimensionales de forma cúbica llamados vóxeles. En tal sentido, un voxel, es el equivalente tridimensional del concepto de píxel en imagines bidimensionales.

Los vóxeles son ampliamente utilizados para graficar y representar objetos volumétricos dando origen a un conjunto de técnicas comúnmente llamadas *Voxel Rendering*.

Para el desarrollo del videojuego se ha creado un motor de renderizado de vóxeles. Los algoritmos para renderizar vóxeles usualmente requieren una gran cantidad de accesos a memoria y ejecutar una cantidad muy elevada de operaciones matemáticas complejas. Además, no existe hardware específico para graficar vóxeles. Sin embargo, si existe hardware de aceleración 3D preparado para graficar triángulos en forma performante utilizando una arquitectura de procesamiento paralelo (GPU), y por lo tanto, los métodos de renderizado deben adaptarse al mismo. Debido a que se pretende crear un videojuego, uno de los factores más importantes en este caso es la velocidad de renderizado y el tiempo de respuesta. Para lograr esto, se adaptaron técnicas estándar de renderizado de vóxeles con el objetivo de buscar la mayor performance posible y visualización en tiempo real del resultado. En consecuencia, se prepondera la velocidad de renderizado a expensas de la calidad gráfica. Esto implica que podrían aparecer ciertos artifacts en la imagen imperceptibles para el jugador, o dentro de ciertos límites aceptables, que no tienen un impacto visual en la estética del videojuego. Por ejemplo, el color de un área de tejido puede no coincidir exactamente con el tejido real, y aunque esta información es importante a nivel médico, no es así en el caso del videojuego.

En los últimos años ha crecido rápidamente el interés de aplicar las técnicas de renderizado de datos volumétricos para graficar fenómenos atmosféricos, fuego,

humo y nubes en tiempo real. Gracias a la evolución del hardware de gráficos programable, estas técnicas tienen una gran variedad de aplicaciones, desde visualizaciones científicas hasta efectos avanzados de iluminación en juegos de computadora (Engel y otros, 2006).

En el campo de la visualización de imágenes médicas, cada vez es más común renderizar un modelo tridimensional a partir de escaneos de Rayos X, Tomografías Computarizadas, Resonancias Magnéticas y Tomografías por Emisión de Positrones. Estos modelos permiten interpretar mejor el tamaño, orientación y otras relaciones espaciales de la anatomía de una persona, lo que facilita la realización de diagnósticos, intervenciones y terapias.

El Engine que se desarrolla en este trabajo permite utilizar las técnicas de *Ray Casting* y *Texture Based Volume Rendering*. Se utiliza la API gráfica OpenGL y el lenguaje de *shading* de la misma.

En cuanto al videojuego, el mismo es del tipo 3D de exploración en primera persona. El jugador comienza ubicado dentro de un volumen 3D y debe navegar por el mismo en cualquier dirección buscando y eliminando anomalías dentro del mismo.

1.1. Engine de Renderización

El Engine está posee la siguiente estructura:

- Soporte de textura 3D: posee soporte para cargar archivos tipo RAW (un formato de vóxels en el cual los datos están organizados uno a continuación del otro sin ningún metadato). El motor carga el archivo, lo transforma de una textura 3D compatible con la placa de video y le agrega información adicional para usar en el videojuego (como por ejemplo anomalías) utilizando canales adicionales de la textura.
- Soporte de textura 2D: el Engine tiene implementado un *loader* de archivos bmp de 24 bits, el cual le añade un *color key* al mismo que permite transformar un bmp estándar en una máscara binaria (transparente - opaca). Esto es utilizado por ejemplo para dibujar la mira.

- Soporte de primitivas graficas 2D de alto nivel: el engine tiene soporte para dibujar distintos componentes gráficos 2D (líneas con gradientes, rectángulos con bordes, rectángulos con gradientes, círculos, etc.). Esto se utiliza para dibujar la GUI.
- Soporte de textos simples: el engine permite cargar un *font* de Windows y transformar la geometría a primitivas de OpenGL. Este módulo permite dibujar textos con geometrías lineales, aunque no colorea el interior de las mismas.
- Método de renderizado volumétrico por Ray Casting: es el *core* central del motor de rendering. Consiste en un Vertex Shader que implementa un *Full Screen Quad*. El Full Screen Quad es un rectángulo que ocupa toda la pantalla a efectos de llamar un pixel shader por cada pixel de pantalla en paralelo. En este caso, el punto de vista se encuentra dentro del volumen que se quiere dibujar y la técnica estándar no es efectiva. A medida que el rayo se aleja del punto de partida, la contribución de cada vóxel que se evalúa decrece en forma exponencial. Para un videojuego, sólo es necesario tomar algunos puntos de muestra en la dirección del rayo. Cuántas más muestras se computen más lento es el proceso, considerando que el acceso a textura es muy costoso en la arquitectura de GPU. El engine implementa la ecuación de Volume Rendering, muestrea la textura volumétrica y calcula un promedio ponderado de todas las muestras. Eventualmente puede aplicar una serie de filtros específicos para resaltar ciertas partes del tejido, agregar contrastes, distintos efectos especiales, etc. El algoritmo de ray casting está diseñado para dibujar los objetos observándolos desde el exterior, por lo tanto se realizaron las siguientes adaptaciones para este caso, en que el punto de vista se encuentra dentro del objeto:
    - Distancia inicial: resuelve el problema que el observador está dentro del volumen.

    - Cantidad de pasos: cantidad de accesos a textura.
    - Paso: distancia en vóxeles a avanzar en la dirección del rayo en cada iteración.
- Método de renderizado volumétrico por Texture VR: desde la aplicación se dibujan una serie de Quads, uno delante de otro, de atrás hacia adelante. Luego, se llama un Fragment Shader especial que realiza un muestreo de la textura volumétrica simulando la inclinación y orientación del Quad. Se realizó una modificación a la técnica original, que consiste en la aplicación de un factor de escala al quad, disminuyendo el tamaño de los Quads posteriores, dando una sensación de perspectiva. Además, en el Fragment Shader se aplica una función heurística que permite generar un efecto de transparencia. Si bien desde el punto de vista de médico estas modificaciones no son correctas, las mismas se implementaron para mejorar la estética para el videojuego.

1.2. Videojuego

El videojuego que se desarrolla utiliza el engine de renderización para cargar un escenario pre-optimizado. El mismo comienza con el jugador situado dentro de un volumen 3D (una cabeza) que contiene unas esferas rojas distribuidas aleatoriamente por todo el escenario. El objetivo del videojuego es eliminar todas las esferas antes que finalice el tiempo disparándoles con el clic del mouse. Por cada esfera eliminada se suman 100 puntos y al finalizar el juego, debido a eliminación de las esferas o el fin del tiempo, se muestra una pantalla indicando el puntaje total alcanzado.

Los dos procesos más lentos del videojuego son los relacionados con la visualización del escenario y del mapa. Las optimizaciones realizadas para conseguir la cantidad de FPS deseada son las siguientes:

- Para el visor de información del videojuego:
    - Se ajustó la cantidad de *slices* dibujados a la cantidad mínima posible manteniendo una calidad aceptable.

    - Se aplicaron funciones de suavizado que si bien modifican la imagen a nivel médico, favorecen la estética del videojuego.
- Para el escenario:
    - Se limitó la cantidad de accesos a textura del método Ray Casting a una cantidad configurable (cant_steps). Cuanto menor distancia, mayor precisión, pero la distancia de visión disminuye.
    - Distance Fields: se realizó una adaptación a esta técnica, en la cual, como todos los vóxels que atraviesa el rayo de visión se encuentran ocupados, se considera como espacio vacío a todos los vóxels contiguos al voxel actual cuya diferencia de color este dentro de un rango predefinido. Esto fue realizado utilizando WebGL, y al no tener soporte para texturas volumétricas, hubo problemas detectando los límites de los slices, generando una cantidad de artifacts que no era aceptable.

1.2.1. Implementación de Distance Fields

En las imágenes médicas la mayor parte de los vóxels tienen como mínimo una cierta intensidad, o sea que prácticamente no hay vóxels vacíos, un campo de distancia usual no tendría utilidad real. En consecuencia, se define un campo de variaciones S que almacena en cada elemento de la textura volumétrica la distancia máxima donde los vóxels vecinos no superan cierta variación con respecto al valor del vóxel central.

$$S(p) = d \text{ / si } |p-q| < d \Rightarrow |I(p)-I(q)| < E$$

Ecuación 1

Dónde:

- d es la distancia en vóxels almacenada en el mapa de variaciones S.
- I es la intensidad del volumen que se está representando.
- E es el valor máximo de variación permitido en el entorno del vóxel.

Durante el *Ray Marching* se utiliza el mapa de variaciones para reemplazar N accesos a texturas por el valor del vóxel. Es decir, en lugar de iterar N veces a intervalos regulares, directamente se utiliza el valor del vóxel.

1.2.1.1. Algoritmo de Distance Field

El algoritmo comienza en el punto de partida *r0* (cuando *t* vale cero) y accede a la textura volumétrica para recuperar el valor del vóxel actual y al mapa de variaciones que devuelve la distancia máxima sobre la cual no hay mayores variaciones de intensidad en los vóxels vecinos. La idea principal es que al no haber variaciones significativas en un entorno de distancia d del punto actual, se pueden reemplazar los siguientes pasos por el mismo valor sin demasiada pérdida de precisión. Para determinar la cantidad de pasos que representa la distancia se divide por el paso fijo almacenado en el parámetro *step*. Esa cantidad, llamada *s* en el algoritmo, representa la cantidad de pasos ahorrados, así la distancia *t* se incrementa según el valor de *s* y a su vez la contribución del vóxel, o sea el valor *I*, tiene que ser evaluado *s* veces, como si fuese muestreado varias veces.

```
1. C = 0
2. t = 0
3. i = 0
4. while(i<cant_steps)
5.{
6.      p = r(t)
7.      I = tex3D(p)
8.      d = S(p)
9.      s = d / step
10.     C = C + VR(I)*s
11.     t = t + d
12.     I = i + s
```

13.}

- C representa el color acumulado sobre cada rayo.
- t es la distancia recorrida sobre la ecuación del rayo, que es r(t) = r0 + t * rd.
- i es el paso actual.

La eficacia de la técnica depende de la elección del valor de épsilon *E*. Valores elevados permiten una gran aceleración, pero introducen una serie de artifacts en la imagen, mientras que valores demasiados pequeños no generan beneficios. La elección del valor es un parámetro configurable de diseño.

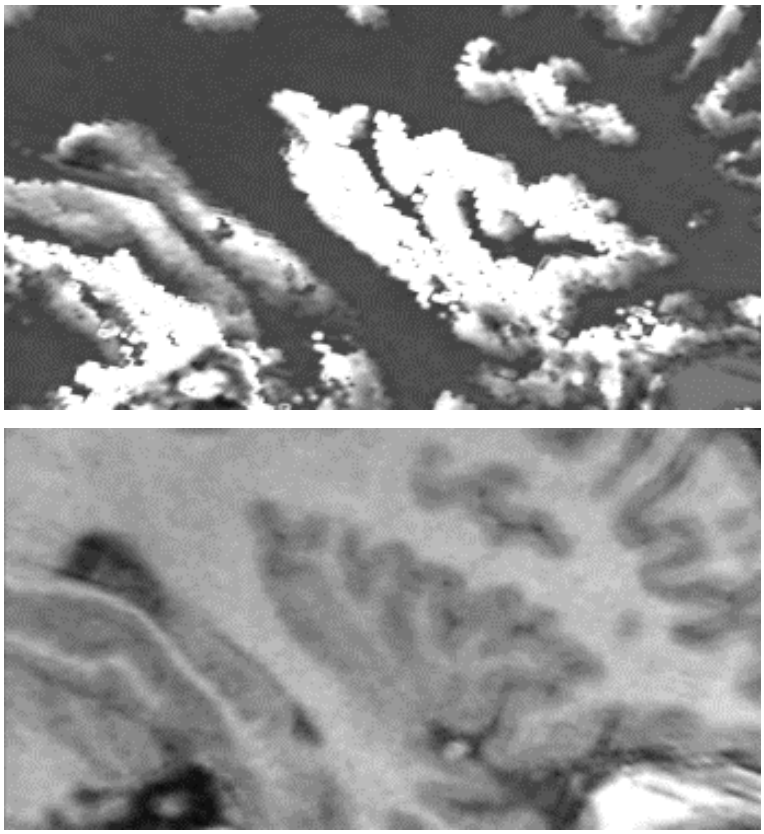

Ilustración 1 - Arriba: artifact por E demasiado grande (E=10px), abajo (E=4px)

1.3. Capturas de pantalla

Ilustración 2

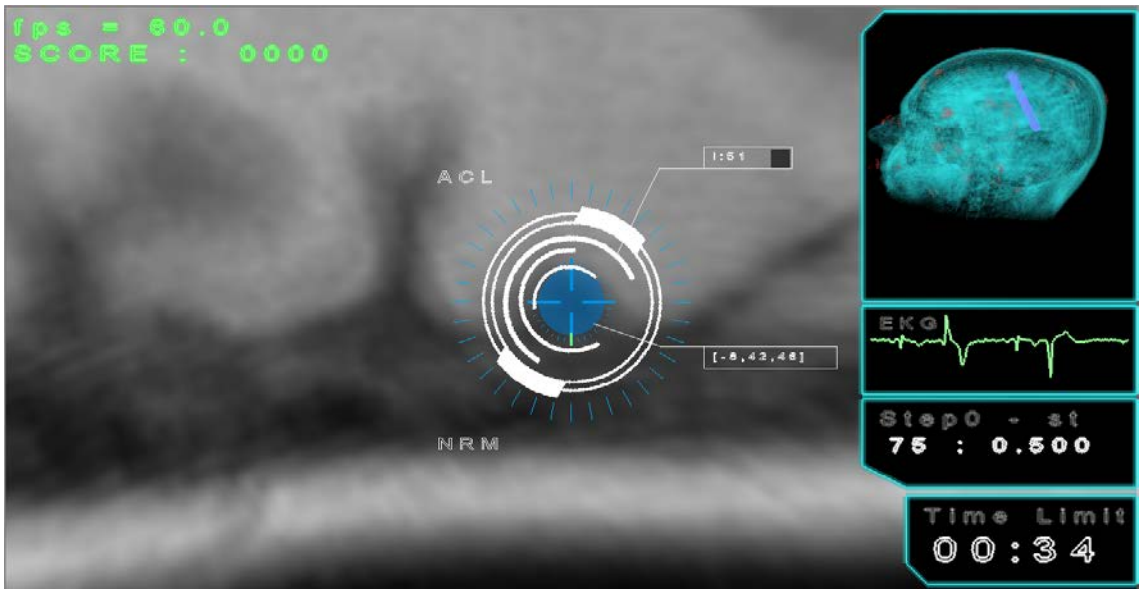

Ilustración 3

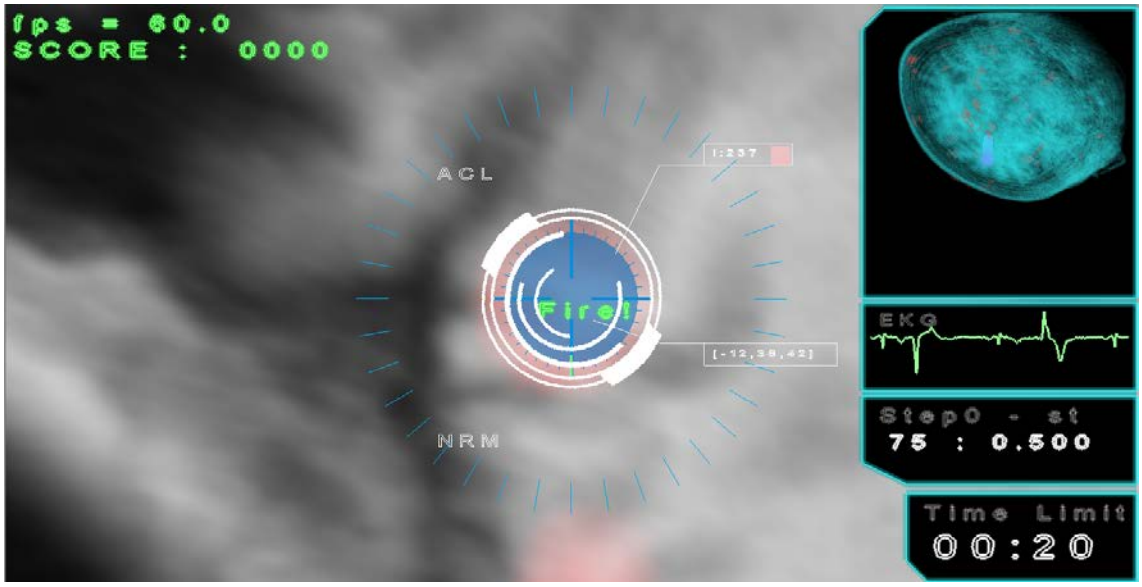

Ilustración 4

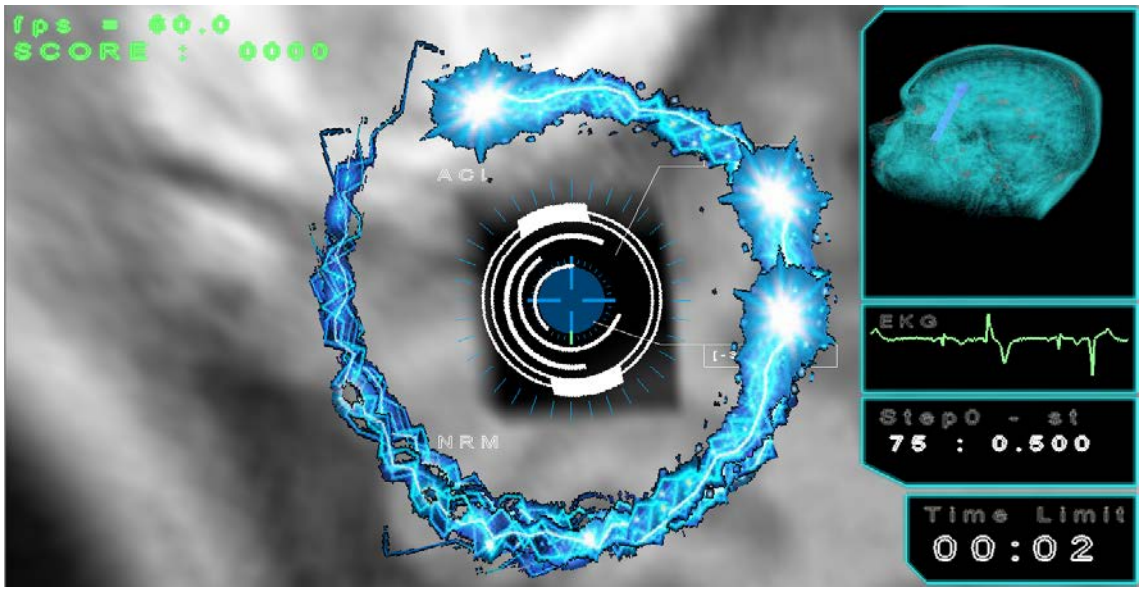

Ilustración 5

Ilustración 6

Bibliografía

- AMIGO, J.C., 2012. Diseño y desarrollo de un juego en WebGL [en línea]. S.l.: Universidad Politécnica de Catalunya. Disponible en: http://upcommons.upc.edu/bitstream/handle/2099.1/15478/82457.pdf?sequence=1&isAllowed=y.
- BASTOS, Thiago; CELES, Waldemar. GPU-accelerated adaptively sampled distance fields. En Shape Modeling and Applications, 2008. SMI 2008. IEEE International Conference on. IEEE, 2008. p. 171-178. 13
- BASTOS, Thiago; CELES, Waldemar. GPU-accelerated adaptively sampled distance fields. En Shape Modeling and Applications, 2008. SMI 2008. IEEE International Conference on. IEEE, 2008. p. 171-178. 13
- COLMENA DE CELIS, J.M., 2010. Un entorno virtual con clientes remotos sobre la plataforma XNA [en línea]. S.l.: Universidad Carlos III de Madrid. Disponible en: http://e-archivo.uc3m.es/bitstream/handle/10016/7585/PFC_Jose_Miguel_Colmena.pdf?sequence=1.
- ENGEL, Klaus, y otros, Real-time volume graphics. CRC Press, 2006.
- GE HEALTHCARE, 2015. Clinical Image Library - Optima NM/CT 640. [en línea]. [Consulta: 9 diciembre 2015]. Disponible en: http://www3.gehealthcare.com/en/products/categories/nuclear_medicine/spect_and_spect-ct/optima_nm-ct_640/optima_nm-ct_640_clinical_image_library#tabs/tab725DE12C6A0A4100955712C9C1BE8766.
- HADWIGER, Markus, y otros, High-quality volume graphics on consumer pc hardware. En IEEE Visualization. 2002.
- HONG, Sunpyo; KIM, Hyesoon. An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. En ACM SIGARCH Computer Architecture News. ACM, 2009. p. 152-163.

- HOSPITAL UNIVERSITARIO AUSTRAL, [sin fecha]. Tomografía Axial Computada (TAC) Y Tomografía por Emisión de Positrones (PET). [en línea]. [Consulta: 13 noviembre 2015]. Disponible en: http://www.hospitalaustral.edu.ar/pruebas-diagnosticas/tomografia-axial-computada-tac-y-tomografia-por-emision-de-positrones-pet/.
- IKITS, Milan, y otros, Volume rendering techniques. GPU Gems, 2004, vol. 1.
- KESSENICH, John; BALDWIN, Dave; ROST, Randi. The opengl shading language. Language version, 2004, vol. 1.
- KHRONOS, 2011. What Is WebGL. [en línea]. [Consulta: 7 marzo 2016]. Disponible en: https://www.khronos.org/webgl/wiki/Getting_Started.
- LEFOHN, Aaron E., y otros, A streaming narrow-band algorithm: interactive computation and visualization of level sets. En ACM SIGGRAPH 2005 Courses. ACM, 2005. p. 243.
- LEVOY, M. 1988. "Display of Surfaces from Volume Data." IEEE Computer Graphics & Applications 8(2), pp. 29–37.
- LINDHOLM, Erik, et al. NVIDIA Tesla: A unified graphics and computing architecture. IEEE micro, 2008, vol. 28, no 2, p. 39-55.)
- MAX, N. 1995. "Optical Models for Direct Volume Rendering." IEEE Transactions on Visualization and Computer Graphics 1(2), pp. 97–108.
- MOZILLA DEVELOPER NETWORK. Tutorial Canvas [en línea]. 2016. S.l.: s.n. [Consulta: 1 Febrero 2016]. Disponible en: https://developer.mozilla.org/es/docs/Web/Guide/HTML/Canvas_tutorial
- NOON, Christian John. A volume rendering engine for desktops, laptops, mobile devices and immersive virtual reality systems using GPU-based volume raycasting. 2012.
- NOVO, Rodríguez Javier; PORTO, Díaz Iago; SUÁREZ, Casal Pedro; VALENCIA, Almansa José. Lenguajes de Shading de Alto Nivel, 2005.

- NVIDIA, 2007. The Cg Tutorial. [en línea]. [Consulta: 06 octubre 2016]. Disponible en: http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter01.html
- REZK-SALAMA, Christoph, y otros, Interactive volume on standard PC graphics hardware using multi-textures and multi-stage rasterization. En Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware. ACM, 2000. p. 109-118.
- RHADAMÉS, C., 2015. Introducción al Rendering Directo de Volúmenes. [en línea]. Caracas: Disponible en: https://www.researchgate.net/profile/Rhadames_Carmona2/publication/272493263_Introduccin_al_Rendering_Directo_de_Volmenes/links/54e65fdf0cf2cd2e028ec239.pdf.
- SCHWABER, Ken. Scrum development process. In Business Object Design and Implementation, pp. 117-134. Springer London, 1997.
- SIEMENS, 2015. Magnetic Resonance Imaging - DICOM Images [en línea]. 2015. S.l.: s.n. [Consulta: 1 Febrero 2016]. Disponible en: http://www.healthcare.siemens.com/magnetic-resonance-imaging/magnetom-world/clinical-corner/protocols/dicom-images
- TOMCZAK, Lukasz Jaroslaw. GPU Ray Marching of Distance Fields. Technical University of Denmark, 2012. 8
- TOMCZAK, Lukasz Jaroslaw. GPU Ray Marching of Distance Fields. Technical University of Denmark, 2012. 8
- WEB3D CONSORTIUM, 2014. X3D Medical. [en línea]. [Consulta: 13 noviembre 2015]. Disponible en: http://www.web3d.org/wiki/index.php/X3D_Medical.
- WONG, Henry, et al. Demystifying GPU microarchitecture through microbenchmarking. En Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on. IEEE, 2010. p. 235-246 15