Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Centro de Computación Gráfica

ACTA DEL VEREDICTO

Quienes suscriben, Miembros del Jurado designado por el Consejo de la Escuela de Computación para examinar el Trabajo Especial de Grado, presentado por el Bachiller Fernando Álvarez C.I.: 18.899.353, con el título Sistema colaborativo en red para el despliegue de datos médicos, a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído el trabajo por cada uno de los Miembros del Jurado, se fijó el día 22 de Mayo de 2014, a las 2:00 pm, para que su autor lo defendiera en forma pública, en el Centro de Computación Gráfica, lo cual se realizó mediante una exposición oral de su contenido, y luego respondió satisfactoriamente a las preguntas que les fueron formuladas por el Jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo.

En fe de lo cual se levanta la presente acta, en Caracas a los 22 días del mes de Mayo del año dos mil catorce, dejándose también constancia de que actuó como Coordinador del Jurado el Profesor Esmitt Ramírez.

Prof. Esmitt Ramírez

Prof. David Perez

Prof. Walter Hernández

Universidad Central de Venezuela

Facultad de Ciencias
Escuela de Computación

Centro de Computación Gráfica

Sistema colaborativo en red para el despliegue de datos médicos

Trabajo Especial de Grado en la Lic. de Computación

Autor: Fernando José Álvarez Diez

Tutor: Esmitt Ramírez

Mayo 2014

Resumen

El despliegue de datos médicos ha sido constantemente estudiado en los últimos años. Actualmente, los sistemas expertos orientados al estudio de la medicina presentan grandes beneficios a la hora de la enseñanza, facilitando la comprensión y visualización por parte de los usuarios, sin embargo, usualmente son muy costosos y difíciles de implementar, además muy complicados y con una gran curva de aprendizaje.

En este trabajo se implementa un sistema colaborativo desarrollado para ser visualizado vía web orientado a la discusión médica, facilitando el acceso a un grupo de personas a datos, de manera grupal y transparente a la posición física del usuario con resultados los más fieles posibles.

Palabras claves: DICOM, Ray Casting GPU, imágenes médicas, aplicación web.

Abstract

The deployment of medical data has been continually studied in recent years. Currently, expert systems oriented to medicine have great benefits when it comes to education, understanding and facilitating viewing by users, however, are usually very expensive and difficult to implement, along with a very complicated and big learning curve.

This paper developed a collaborative system for viewing via oriented medical discussion site, providing access to a group of people to group data and transparently to the physical position of the user, with the most results possible faithful implemented.

Keywords: DICOM, Ray Casting GPU, medical imaging, web app.

Índice de Contenidos

Índice de Ilustraciones

Índice de Tablas

Introducción

En la actualidad, la computación ha tenido una gran influencia en la medicina, facilitando la realización de muchos procesos relacionados con el análisis de enfermedades. Con el desarrollo de sistemas expertos o programas que se nutren de diferentes áreas de la computación, muchos procesos de toma de decisiones, como el caso del diagnóstico y tratamiento de enfermedades, se han hecho más rápidos y especializados, permitiendo optimizar la calidad del servicio en materia de salud [**1**].

En este sentido, existen diferentes maquinarias y equipos médicos que permiten la extracción, edición y visualización de datos médicos, que pueden tener orígenes variados y usualmente se despliegan al usuario final en formato de imágenes y/o volúmenes. Estos formatos son utilizados por sistemas expertos que facilitan y mejoran la experiencia de examinar la anatomía humana y por extensión permitir el diagnóstico de enfermedades.

Por esta razón, es necesario el desarrollo de técnicas y herramientas para el tratamiento de imágenes y volúmenes. Así como también el estudio de diversas técnicas que permitan una integración homogénea entre modelos de datos y técnicas de despliegue de imágenes y volúmenes, con el objeto de desarrollar aplicaciones que faciliten el trabajo a los diferentes usuarios, haciendo transparente los temas relacionados con la obtención y despliegue de este tipo de datos.

Actualmente existen diversos desarrollos orientados al despliegue de volúmenes médicos, integrando diferentes técnicas de almacenamiento de datos y de despliegue de los mismos desde un servidor. Un ejemplo de dichos desarrollos son las investigaciones de [**2**] y [**3**], en las que se presentan unas aplicaciones web que permiten la carga de datos DICOM y facilitando el despliegue de imágenes y volúmenes, utilizando como técnicas de despliegue texturas 3D y las bibliotecas ITK respectivamente. A pesar de que estas investigaciones sirvieron como referencia para el desarrollo de este trabajo especial de grado, se descartaron estos métodos de despliegue, y se implementó el Ray Casting en GPU presentado en [**3**], como la técnica de despliegue para obtener mejores calidades de imagen y menores tiempo de despliegue

De esta manera, estos antecedentes de investigaciones sirvieron de punto de partida para la presente investigación, la cual se plantea como objetivo desarrollar una aplicación de despliegue de imágenes y volúmenes médicos de manera colaborativa, que permitan a los usuarios el manejo practico y transparente de los datos desde cualquier navegador, mientras es posible la comunicación entre ellos de manera no presencial.

El presente documento se estructura de la siguiente forma: El Capítulo 1 especifica el tema de los datos médicos. Sus orígenes, técnicas de refinamiento y técnicas utilizadas para el despliegue y almacenamiento de dichos datos. El Capítulo 2, se enfoca en la implementación del sistema, ilustrando las diferentes partes que conforman el sistema y como es su funcionamiento. Ya en el Capítulo 3, se evalúan los diferentes resultados arrojados de las pruebas realizadas al sistema, que involucran los tiempos de carga y de envío de los modelos, realizando diferentes aproximaciones al problema para las tomas de decisiones realizadas para el desarrollo final del sistema. El último capítulo plantea una serie de limitaciones asociadas al trabajo de grado. Así como una serie de recomendaciones y aspectos a tomar en cuenta para trabajos futuros sobre la plataforma.

Capítulo 1. Imágenes Médicas

Los avances de las imágenes médicas se han visto favorecidas por los avances de la tecnología, siendo ampliamente influenciada por el desarrollo de áreas como el sector militar, sus avances se aplicaron a la medicina por su potencial de detección y diagnóstico de enfermedades humanas. Otro ejemplo de estos desarrollos incluyen el ultrasonido, inicialmente utilizado para Sonares en submarinos y dispositivos orientados a la detección de pérdida de sangre, utilizados en el campo de batalla.

Los laboratorios de investigación también han producido muchas tecnologías orientadas al tratamiento de imágenes, que también fueron migradas a la medicina de manera exitosa, los ejemplos incluyen todos los avances en la matemática necesaria para la reconstrucción de imágenes por tomografía, las técnicas de laboratorio en la resonancia magnética que evolucionaron en lo que hoy en día se conoce como MRI (*Magnetic Resonance Imaging*), la espectrometría y otros avances útiles para la medicina actual [**6**].

Todos estos avances, favorecieron también al campo de la medicina conocido como imagenología médica, que se puede definir como un conjunto de modalidades y técnicas de adquisición de imágenes y datos, diferenciables entre sí en cuanto a la naturaleza de los principios físicos involucrados en el proceso de adquisición. Adicionalmente, existen también diferencias en cuanto a su aplicación médica. Entre las modalidades más comunes se encuentran los rayos X, la tomografía computadorizada, la resonancia magnética nuclear, la imagenología nuclear y la imagenología por ultrasonidos.

Este tipo de datos o imágenes médicas captadas por las diferentes modalidades (MRI, CT, etc.), típicamente se obtienen como proyecciones en dos dimensiones, captando imágenes individuales o una secuencia de ellas por examen, este último caso permite ver dichas imágenes de manera individual o una por una y facilita el despliegue de un modelo en tres dimensiones [**7**].

Aunque es posible llevar a cabo la imagenología médica con las imágenes sin recibir tratamiento, muchos detalles pueden perderse por la capacidad del ojo humano de interpretar los datos y de la calidad de los equipos de despliegue. Es por eso que para llevar a cabo la imagenología médica se requieren de diferentes técnicas de procesamiento digital de las imágenes, estas técnicas operan sobre la representación digital de una imagen, con el

objeto de destacar algunos de los elementos que conforman la escena, de modo que se facilite su posterior análisis, bien sea por parte de un usuario o un sistema de visión artificial. En general, las técnicas de procesamiento de imágenes son aplicadas cuando resulta necesario realzar o modificar una imagen para mejorar su apariencia o para destacar algún aspecto de la información contenida en la misma, o cuando se requiere, medir, contrastar o clasificar algún elemento contenido en la misma. También se utilizan técnicas de procesamiento, cuando se requiere combinar imágenes o porciones de las mismas o reorganizar su contenido [**8**].

1.1 Técnicas de Procesamiento de imágenes

La meta del refinamiento de imágenes es percibir mejor toda la información relevante del diagnóstico presente en la imagen. Un ejemplo, en la radiología digital, las imágenes de 12 bits pueden almacenar 4096 posibles tonalidades de gris, lo que hace imposible para el ojo humano distinguir tal cantidad de valores de gris contenidos en una imagen [**7**] es entonces cuando se puede utilizar las siguientes técnicas para la facilidad de visualización.

1.1.1 Histograma

Una manera de simplificar el análisis de las imágenes médicas es el uso de histogramas, que consisten en crear una representación del número de píxeles en la imagen al agruparlos en función de su intensidad. Gracias a esta herramienta se facilita el análisis de imágenes, usualmente reduciendo la dificultad de tareas como identificar el color de fondo o la representación de vacío en la imagen, así como contabilizar el ruido que aparece dentro de la misma. Un ejemplo de un histograma se puede ver en la Ilustración 1.

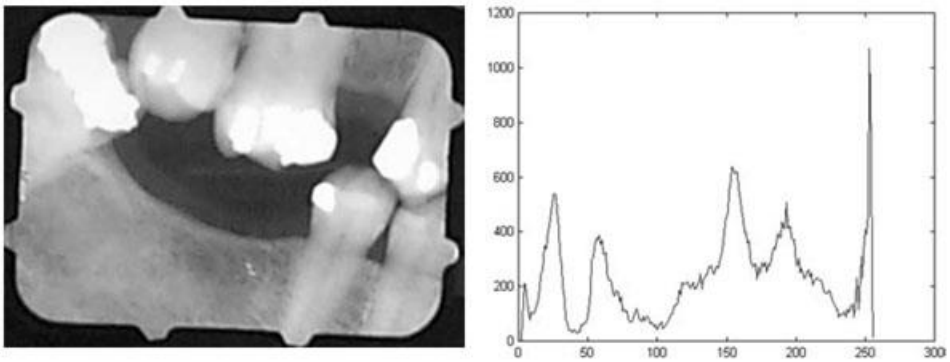

Ilustración 1: Imagen obtenida por rayos X con su respectivo histograma
Obtenido de [**7**]

1.1.2 Transformación de los valores de Gris

Este tipo de transformaciones parte de que en cada píxel (*x*, *y*) de la imagen, tiene un valor en la escala de grises, al que se le aplica una función que transforma ese valor de gris contenido en el píxel (*x*, *y*) a un valor nuevo independiente de su posición, tarea que ocurre para todos los píxeles en la imagen. Es importante destacar que esta transformación puede ser llevada a cabo también en una imagen a color, pero será necesario un mayor número de transformaciones para llevar un color a su nueva interpretación.

Un ejemplo particular de este tipo de transformaciones es la operación de ventana y nivel (*window/level operation*). En esta operación expande el contraste dentro de un rango de ventana, utilizando un valor L para indicar el medio de la ventana y un valor W para el tamaño de la misma. El resultado final es una imagen donde todos los valores dentro de la ventana reciben una interpolación lineal, mientras que toda intensidad menor a (L - (W/2)) es igual a cero y toda intensidad mayor al valor (L + (W/2)) se interpreta como valor máximo (Ver Ilustración 2).

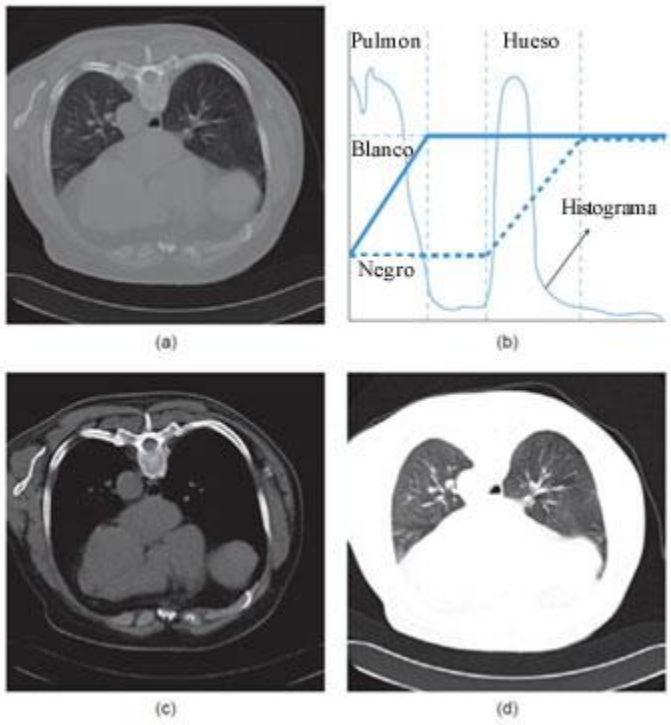

Ilustración 2: Aplicación del refinamiento *window/level*
Obtenido en [**6**]

Una operación más simple de este tipo de transformaciones se conoce como umbral, en ella, todos los valores de grises encontrados hasta cierto valor de umbral, se interpretan como cero, y todos los valores sobre este umbral se consideran máximo valor de gris. De igual manera hay aplicaciones más avanzadas como la función de transferencia.

1.1.3 Función de Transferencia

Esta técnica asigna un conjunto de colores para distinguir ciertos atributos que no son fáciles de percibir en la imagen original. El modo más común de función de transferencia son aquellas que asignan opacidad y color transformando la imagen monocromática en una imagen a color, asignándole un color a un píxel con respecto a su intensidad al aplicarle una función, un ejemplo de esta implementación puede ser observado en la Ilustración 3.

Ilustración 3: Aplicación de diferentes funciones de transferencia en una imagen
Obtenida en [**9**]

Esta técnica permite modificar la salida del despliegue, pudiendo obtener información importante de los datos. Sin embargo, esta operación es manual y no existen soluciones completamente automatizadas. Sin embargo, trabajos como el de los investigadores Gordon Kindlmann y James W. Durkin [**10**], buscan crear una aproximación a una función de transferencia. Para lograrlo analizan el histograma, realizan una detección de bordes y calculan los diferentes niveles de opacidad, utilizando estos resultados para la creación de esta función de transferencia.

1.1.4 Filtros

Otro tipo de transformaciones que se pueden llevar al cabo son las operaciones locales o de filtros, donde se busca modificar la el color final de cada uno de los píxeles de una manera

que dependan solo por el valor de los píxeles vecinos alrededor del mismo. La selección de los píxeles vecinos involucrados en la transformación se les conoce como máscara, la cual es usualmente la misma y se mueve con respecto a la posición de los píxeles. Un ejemplo de la aplicación de esta técnica es el uso de promedio de la imagen, que utilizando una máscara de 3x3 píxeles donde cada uno de los píxeles vecinos influye en un noveno de valor final del píxel. El resultado final de la imagen es un acabado suavizado que elimina cierto nivel de ruido en la imagen (la Ilustración 4 muestra el uso como se implementa esta técnica).

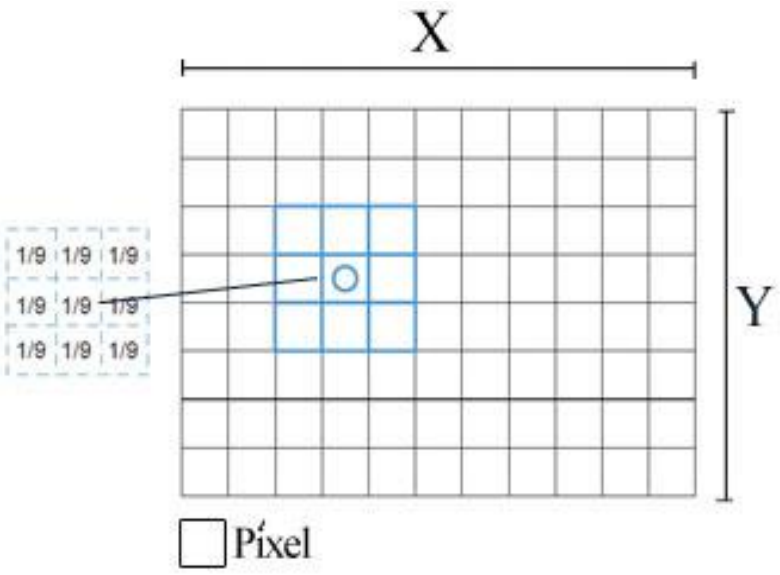

Ilustración 4: Aplicación de una máscara en un píxel dentro de la imagen

Todas estas técnicas y operaciones facilitan la visualización de las imágenes, facilitando las tareas del usuario. Sin embargo, para poder usar estas herramientas y poder visualizar sus resultados, son necesarios métodos precisos que permitan el despliegue de estos datos en el mundo real. Para lograr esto, son menesteres métodos de despliegue que permitan representar los datos de una manera comprensible para el usuario. Sin embargo, en el presente trabajo se omitirá el despliegue de datos en dos dimensiones, por no considerarse pertinente para el desarrollo del objetivo de investigación.

1.2 Despliegue de Volúmenes

En la actualidad, existen dos aproximaciones fundamentales al despliegue de volúmenes, extracción de superficie (*Surface rendering*) y despliegue directo del volumen (*Volume rendering*). La primera técnica surge a finales de la década de los setenta y consiste en

realizar un pre-cálculo para crear una representación intermedia del volumen, que pueda ser tratada por el hardware gráfico. Es decir, la visualización y manejo de datos multidimensionales se basa únicamente en los límites del objeto, el despliegue de la superficie del objeto de interés está aislado del resto del volumen. Dentro de esta aproximación existen diversas técnicas como técnicas como Marching Cubes [**11**] y Tetra Cubes [**12**] .

Estos métodos tienen la limitante de que los datos desplegados presentan artefactos (desperfectos en la representación), causados por la clasificación binaria de los vóxeles, esto quiere decir que un vóxel solo puede aportar como un todo, donde cada uno de ellos puede ser parte o no del despliegue final, esta clasificación vuelve a estas técnicas de despliegue poco precisas. No obstante, siguen siendo eficientes para el despliegue rápido de datos volumétricos arbitrarios, pero en aplicaciones orientadas al campo de la medicina, la necesidad de alisamiento o las existencias de artefactos causados por estas técnicas las hacen poco confiables para el despliegue de estos tipos de datos [**13**].

Una segunda aproximación desarrollada una década más tarde, consiste en utilizar los datos volumétricos directamente, sin llevar a cabo ningún tipo de representación intermedia, asignando propiedades ópticas directamente a los vóxeles [**14**]. En otras palabras, a cada vóxel se le asocia un nivel de opacidad y se asume que el valor de cada uno está correlacionado con el tipo de material que lo integra. En este sentido, existen diversas técnicas de despliegue de volumen como: Ray Casting [**15**], Splatting [**16**], ShearWarp [**17**], Texturas 2D [**18**] y Texturas 3D [**19**], a continuación se describirán algunas de estas técnicas.

1.2.1 Ray Casting

La forma más básica de Ray Casting parte del lanzamiento de un rayo por cada píxel del dispositivo de salida, que entrará en contacto con los datos volumétricos e indicará el color y opacidad de cada píxel en dicho dispositivo [**20**]. Para lograr esta tarea, se traza un rayo por cada píxel de la imagen final tomando como punto origen el punto de visión y cruzándose con todos y cada uno de los píxeles encontrados en el plano de visión, buscando identificar si existe colisión de estos rayos con los datos volumétricos y conocer cuál es el aporte de estos datos a cada píxel en la imagen final.

Por ser una representación discreta de los datos, es poco probable cada uno de estos rayos emitidos estén alineados con los vóxeles del volumen, por lo que es necesaria la

interpolación de valores de los vóxeles más cercanos al paso del rayo para obtener el valor tentativo. Cuando se realiza este cálculo de interpolación, se calcula el aporte de este vóxel, se indica cuanto es el color y opacidad de cada uno de los píxeles intersectados.

Cuando termina el proceso de conocer el color y opacidad de los vóxeles seleccionados y se empiezan a acumular para lograr el color final de cada uno de los píxeles, esta tarea inicia acumulando color desde los vóxeles más cercanos a la cámara, para conseguir de esta manera, que vóxeles más lejanos a ella influyan de menor manera al píxel resultante, todo el proceso de coloreado puede observarse en la Ilustración 5.

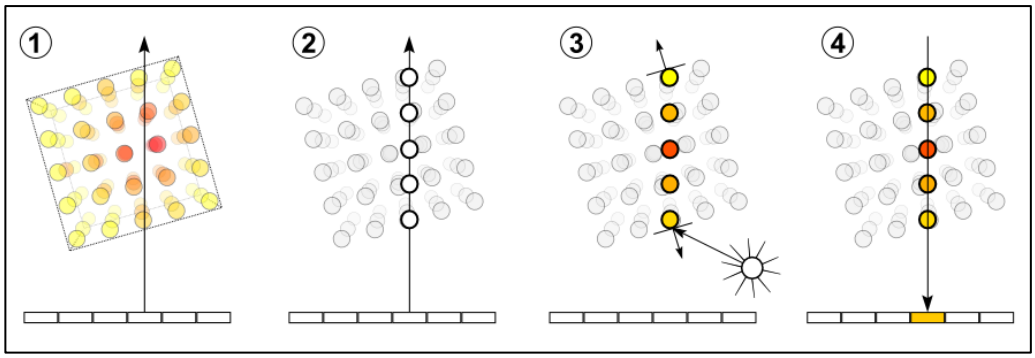

Ilustración 5: Proceso de Ray Casting: 1) Ray Casting 2) Muestreo 3) Coloreado 4) Composición.

Esta técnica se reconoce por generar una gran calidad de imagen, ya que simula las propiedades físicas de la luz, generando imágenes precisas, suavizadas y de alta calidad. Sin embargo, para la mayoría de los usuarios, o donde es necesario llevar a cabo el despliegue de muchos modelos de manera simultánea. El uso de esta técnica está limitado a volúmenes de menor tamaño, dado al alto nivel de poder computacional requerido para desplegar un volumen [**21**]. Sin embargo, existen diferentes técnicas de optimización, que disminuyen el número de cálculos realizados por el Ray Casting. Entre las técnicas de optimización más conocidas se encuentran la terminación temprana del rayo, el salto de los espacios blancos y el refinamiento progresivo [**13**].

1.2.2 Texturas 2D

La texturas 2D es una técnica que utiliza la capacidad de los dispositivos para desplegar texturas, trasparencias y geometría, utilizando cortes planos del volumen [**9**]. Cuando se utilizan los planos alineados a los ejes del objeto, el volumen puede ser pensado como secuencias de cortes que se almacenan en texturas 2D, que serán interpretados finalmente como un volumen por las capacidades de transparencia del hardware. No obstante esta

técnica requiere la existencia de tres conjuntos de planos del volumen (ver Ilustración 6) para cuando el conjunto de datos se encuentra muy cercano a encontrarse perpendicular al plano de visión, intercambiarlo convenientemente con un conjunto cuyos planos se encuentren más cercanos a estar paralelos con el plano de visión (ver Ilustración. 7).

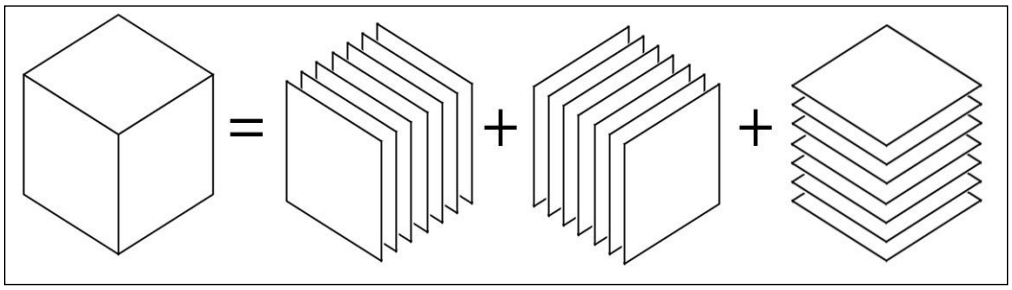

Ilustración 6: Cortes para texturas 2D

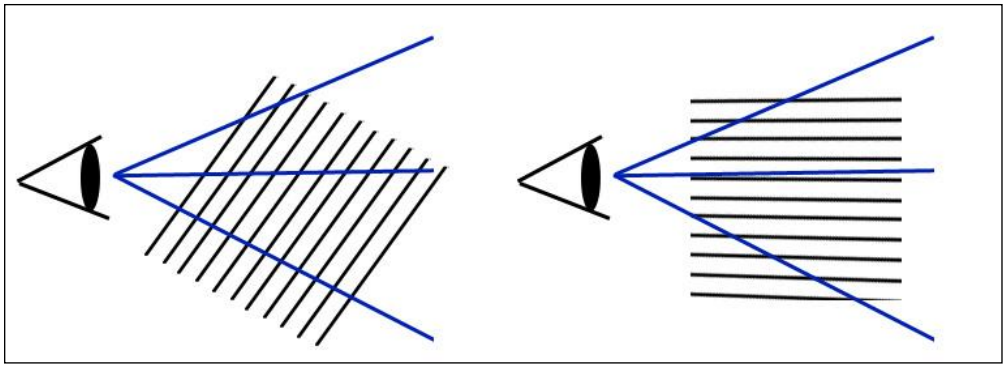

Ilustración 7: El modelo es visible (izquierda) visión incorrecta del modelo (derecha)

Entre las ventajas de este método, se consideran su fácil implementación y excelentes tiempo de respuesta y la portabilidad de las imágenes creadas, debido a que son soportadas por la mayoría de las tarjetas gráficas. Sin embargo, puede considerarse inconveniente la molestia visual cuando se cambia el conjunto de datos por el uso de las rotaciones [**9**].

1.2.3 Texturas 3D

Otra técnica que requiere de geometría y texturas intermedias es el uso de texturas 3D, donde el modelo de los cortes coronales, sagitales y axiales, son transformados directamente en una textura 3D, que servirá para hacer una correspondencia entre los planos paralelos con el plano de visión directamente desde el volumen (ver Ilustración 8).

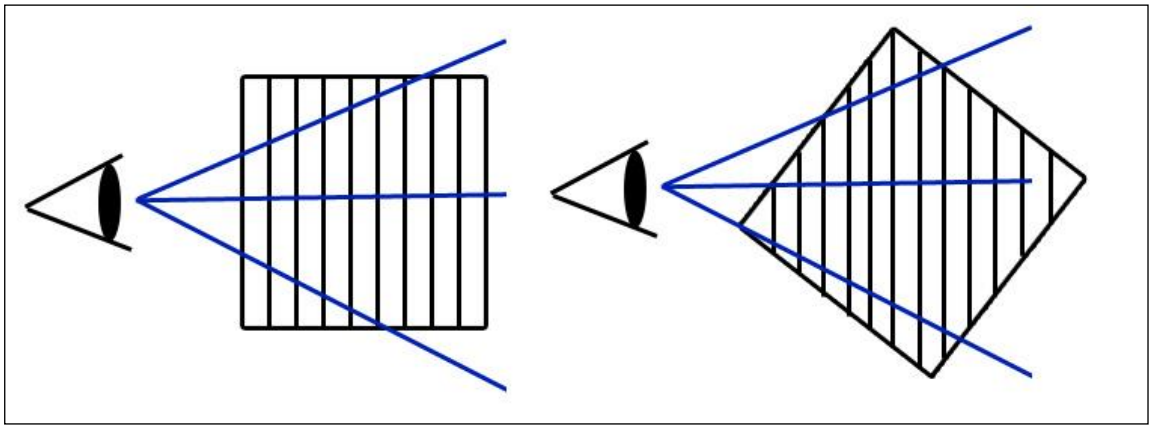

Ilustración 8: Diferentes disposiciones de los cortes en texturas 3D

Este método utiliza comúnmente interpolación trilineal para reconstruir las muestras requeridas. Para el caso de proyección paralela, la distancia entre muestras interpoladas es siempre la misma, independientemente del ángulo de rotación. Para la proyección perspectiva, la distancia entre muestras varía rayo a rayo [**9**].

Una ventaja de la técnica de texturas 3D en contraste con las texturas 2D, es el uso de un solo conjunto de datos, lo que disminuye significativamente el espacio ocupado por el volumen en memoria, pero este es más difícil de implementar, además de volver necesario el soporte de las texturas 3D por el hardware gráfico [**13**].

En resumen, existen numerosas técnicas de despliegue de volúmenes, sin embargo con el uso de éstas se vuelve difícil conseguir tiempos interactivos en computadores convencionales, por lo que es necesario el uso del hardware gráfico que permita mejorar considerablemente los tiempos de carga, solucionando el problema usualmente asociado al uso de técnicas por software.

Adicionalmente es importante destacar, que motivado a la evolución del hardware gráfico, la demanda de estrategias de despliegue de volúmenes con este tipo de arquitecturas se encuentra en constante crecimiento [**22**]. Así, el poder del hardware impulsa a los investigadores a desarrollar técnicas que exploten las cualidades de las tarjetas gráficas, mejorando la calidad y tiempos de carga de cualquier tipo de aplicación, logrando mayores tasas de cuadros por segundo realizando baja inversión.

Esta breve reseña de las diferentes técnicas de despliegue de datos volumétricos, sirve para dar un esbozo de los diversos métodos utilizados por diferentes compañías para el

despliegue de volúmenes. Además de su visualización, es importante considerar que el manejo y almacenamiento de volúmenes, es otro tópico a considerar en el desarrollo de una herramienta computacional orientada al apoyo médico. Así, surge la necesidad del almacenaje ordenado de volúmenes e imágenes para su fácil acceso sin importar la situación geográfica. Por tal motivo, son necesarios sistemas especializados de almacenamiento de datos, como los incluidos en un sistema PACS (*Picture Archiving and Communication System*) [**23**].

1.3 Sistemas de almacenamiento de imágenes y volúmenes

Una de estas tecnologías de almacenamiento imágenes es el PACS, que es un sistema compuesto por hardware y software dedicado al almacenamiento, obtención, manejo, distribución y presentación de imágenes. Los PACS son utilizados frecuentemente en hospitales, con el propósito de reemplazar las cintas físicas por imágenes digitales, que pueden ser usados y vistos por una variedad de médicos y profesionales simultáneamente, sin importar su posición geográfica [**24**].

Los componentes básicos que constituyen a un PACS son: equipos médicos de adquisición de imágenes, interfaces de conversión analógico-digital o video-digital, red de comunicación, bases de datos, estaciones de diagnóstico y visualización, como también sistemas de almacenamiento a largo y corto plazo [**25**]. La adquisición de imágenes médicas tienen diversos orígenes según el fabricante y modelo, estos sistemas pueden ser digitales o sobre película. Sin embargo estos últimos, necesitan diferentes interfaces que permitan la transformación de datos al formato digital.

Un sistema PACS requiere de una red de comunicación, que permite el intercambio de información entre las diferentes estaciones, haciendo transparente las distancias geográficas y velocidades de transferencia. Esta red tiene un papel fundamental en este tipo de sistemas.

Otro elemento requerido por las PACS, es el diseño de la base de datos, que resulta de gran importancia para el almacenamiento de imágenes, voz y texto. Además se necesita una estrategia que consista en el manejo correcto de la información basada en la frecuencia de uso de los datos, por lo que es preciso el uso de algoritmos que determinen cuales imágenes son más necesarias, para así ser recuperarlas de servidores secundarios.

Por su parte, otro aspecto relevante utilizados por las PACS son las estaciones de diagnóstico, que deben cumplir con una serie de normas básicas de resolución y una serie

funciones de modificación de contraste, acercamiento, análisis de texturas, histogramas, filtrado y despliegue en 3D, para lograr la imagen.

Finalmente, un último componente a destacar son los sistemas de almacenamiento, los cuales deben soportar una gran cantidad de terabytes, que permitan usualmente el almacenamiento de años de información.

Con el objetivo de que los sistemas PACS funcionen correctamente en dispositivos de fabricantes diferentes, existen una serie de estándares que consideran la transferencia y el formato de imagen digital para optimizar los PACS en dispositivos como estos. En este sentido, DICOM es actualmente el estándar mundialmente reconocido para el intercambio, manejo y almacenamiento de imágenes (*Digital Imaging and Communication in Medicine*), que se ha convertido en el estándar líder usado por grandes compañías médicas y equipos de diagnóstico, en diferentes ramas de la medicina [**26**] . A continuación se presenta una pequeña instrucción al estándar DICOM.

1.3.1 DICOM

Como se describe en [**27**], DICOM es el estándar universal y fundamental en la imagen médica digital. Como tal, proporciona todas las herramientas necesarias para el diagnóstico preciso y el tratamiento de imágenes médicas. Además, DICOM no es sólo una imagen o un formato de archivo, abarca la transferencia, almacenamiento y protocolo de visualización. Construido y diseñado para cubrir todos los aspectos funcionales de la imagen médica digital.

1.3.2 Funcionamiento de DICOM

Todos los datos del mundo real (pacientes, estudios, equipos médicos) son interpretados por DICOM como objetos, con sus respectivas propiedades y atributos. Esta definición de objetos se encuentra estandarizada por IODs (*Information Object Definitions*), donde se describen todas las características particulares de cada objeto [**27**]. Un ejemplo de esto en la información de un paciente, como sexo, código de identificación, edad, peso, alergias entre muchas que permitan conseguir información clínicamente relevante.

Todas estas características se encuentran en un listado estandarizado con más de 2000 atributos incluidos en un diccionario que facilita la consistencia entre el formato y el nombre del atributo. Es importante destacar que todos estos atributos deben presentar un formato acorde con los 27 valores representación VR especificados en DICOM.

En el momento que los datos son capturados como atributos, pueden ser transmitidos y procesados entre diferentes dispositivos DICOM o AE (*Application Entities*) como son conocidas en DICOM. Este proceso cumple con el esquema cliente-servidor donde las aplicaciones proveen servicio entre ellas que usualmente con lleva un intercambio de datos, por eso, se puede considerar natural asociar servicios particulares con los tipos de datos (IOD) que procesan. DICOM nombra estas asociaciones SOP (*Service-Object Pairs*), agrupando estas asociaciones en clases SOP. Por ejemplo, el almacenamiento de una imagen CT por un escáner a un PACS digital corresponde a un SOP de almacenamiento CT [**27**].

Para DICOM diferenciar los diferentes proveedores de servicios así como quienes los solicitan, DICOM los identifica como SCP (*Service Class Provider*) y SCU (*Service Class User*) respectivamente, En el ejemplo que involucra al CT, el escáner CT actúa como el SCU y el archivo digital cumple de SCP para el almacenamiento CT, este intercambio de datos es conocido como asociación.

Cada una de estas asociaciones inicia con un "*DICOM handshake*" donde dos aplicaciones intercambian información entre ellas, esta información se conoce como "*Presentation Context*" sin ambas aplicaciones pueden unir sus contextos pueden iniciar un proceso SCU-SCP.

El estándar DICOM, se encuentra en constante cambio desde sus orígenes en el año 1983 cuando se empieza a gestar la primera versión del estándar.

1.3.3 Historia

El comité ARC/NEMA [**28**] menciona que en sus orígenes, el estándar DICOM nace como un trabajo conjunto de ACR (*American College of Radiology*) y NEMA (*National Electrical Manufacturers Association)*, que en año 1983 empiezan a desarrollar un estándar para la comunicación de dispositivos de visualización y equipos de obtención de imágenes de diferentes fabricantes facilitando la expansión de la imagen digital y PACS, creando los documentos que servirán como base al estándar [**28**], como los puntos explicados a continuación.

Así, la primera y segunda versión de este estándar, se publicaron en 1985 y 1988 respectivamente, siendo nombradas como ARC/NEMA. Estas versiones presentaban fallos y no contaba con todas las funcionalidades requeridas, por ejemplo, en este paradigma, el usuario podía enviar una imagen a un dispositivo, pero no especificaba que debía hacer el dispositivo con las imágenes. Esto representó un problema para las redes comunicaciones, forzando al grupo de desarrollo a crear una nueva versión que soportara el crecimiento de

dispositivos y redes. Para 1992 se presenta la nueva versión del protocolo con el nombre DICOM 3.0, que propone utilizar protocolos de red estándar TCP/IP, para optimizar las redes de comunicaciones, sin dejar a de lado la compatibilidad con ARC-NEMA 2.0.

Hoy en día DICOM 3.0 no ha sido reemplazado, sin embargo, este estándar se encuentra en constante desarrollo y revisión, de manera que se actualiza y agregan nuevos suplementos, todas estas revisiones se encuentran publicadas en el sitio oficial de DICOM [**29**] para su descarga. A continuación se explicarán las características básicas del estándar.

1.3.4 Características

Los datos clínicos pueden tener diversos orígenes y formatos, sean números, cadenas de caracteres, formato de medidas y fechas, además de diferentes orígenes. en esta sección se explicaran como se manejan estos datos, así como también el intercambio de información entre entidades.

VR (*Value Representation*)

Para manejar esta multitud de formatos, DICOM define 27 tipos de datos básicos conocidos como VR que permiten encapsular todo tipo de datos clínicos, es importante destacar que cualquier dato escrito en un paquete DICOM tiene que coincidir con alguno de estos tipos. Cada VR tiene su propia abreviación de dos letras (Tabla 1).

Nombre del VR	Contenido	Caracteres	Longitud de datos	Formato
CS - Code String	Cadena de caracteres	Mayúsculas, números, "_" y espacio	16 máximo	"CD123_5"
PN - Person Name	EL nombre de la persona separado por "^"	N/A	64 máximo	Perez^Pablo
UI - Unique Identifier (UID)	Cadena de Caracteres que contiene un UID	Números y "."	64	"1.2.840.10008.1.1"
SS - Signed Short	Entero binario 16 bits	N/A	2	N/A
UN – Unknow	Cadena de caracteres donde la codificación es desconocida	N/A	N/A	N/A

Tabla 1 Ejemplos de Valores de Representación en DICOM

Así el VR toma un papel esencial en la estructura de datos de DICOM conectándolo con el mundo real. VR son palabras que DICOM entiende y habla. En otras palabras, el diccionario de datos DICOM transforma los elementos del mundo real en VR [**27**].

Diccionario de datos DICOM

Según [**27**] el diccionario de datos DICOM es un registro de todos los atributos estándar utilizados en la medicina digital, éstos deben utilizar un formato incluido dentro de los 27 VR.

Al contener una gran lista (más de 2000), todos estos se encuentran divididos en grupos numerados basados en las semejanzas de su contenido. Cada grupo de estos se encuentra organizado por elementos individuales, en consecuencia, cada ítem está numerado con su propia dupla grupo, elemento conocida como etiquetas "*tags*", el elemento etiquetado se el conoce como atributo o simplemente elemento DICOM.

De esta manera, una etiqueta únicamente corresponde a un nombre de atributo, lo que hace posible referirse a un elemento de datos por su etiqueta (*0010,0010*) o por el nombre del atributo "*Patient Name*" en la siguiente tabla podemos ver unos ejemplos de etiquetas.

(Grupo, Elemento)	Atributo	Atributo (Español)	VR
(0010,0010)	Patient Name	Nombre Paciente	PN
(0010,0020)	Patient ID	ID del Paciente	LO
(0010,0021)	Issuer of Patient ID	Emisor ID del Paciente	LO
(0010,0030)	Patient's Birth Date	Fecha nacimiento del Paciente	DA
(0010,0032)	Patient's Birth Time	Hora nacimiento del Paciente	TM
(0010,0040)	Patient's Sex	Sexo del Paciente	CS

Tabla 2 Ejemplos de *Tags* en DICOM

De la misma manera, existen atributos privados, que son utilizados ampliamente por las diferentes compañías para intercambiar información entre dispositivos propios, este tipo de información, se encuentra en los grupos impares (ej. 0009 y 0007), que son reservados para información propietaria [**27**].

Utilizando el diccionario de datos en conjunto con los VR, es posible realizar un objeto DICOM. Por ejemplo [**27**], al partir de los elementos siguientes, Luis Alfonzo, que nació el día 9 de octubre de 1985. Con el uso de los VR, etiquetas y aplicación del formato VR conseguimos "(0010,0010) Alfonzo^Luis (0010,0030) 19851009 (0010,0040) M" que incluye los datos del mundo real en elementos DICOM que quiere decir que se almacenara la información para el paciente Luis Alfonzo, sexo masculino, nacido el 09 de Octubre de 1985.

Al conocer como es la sintaxis básica, es posible componer un paquete DICOM. Sin embargo, surge la pregunta de cómo hacer para diferenciar los diferentes paquetes e identificarlos inequívocamente y para esto se utilizan los Identificadores únicos UID.

UID (*Unique Identifiers*)

DICOM UID es una cadena de caracteres como "1.2.840.10008.1.2" como es definida en el VR UI con capacidad de 64 caracteres, esta cadena se busca que sea globalmente única, evitando coincidencias con equipos de diferentes países, instituciones, etc. Este se encuentra codificado en formato "<Organización>.<sufijo>". <Organización> contiene un código único que idealmente debe ser solicitado por las diferentes organizaciones a NEMA para obtener su código único, y el sufijo es una cadena única usualmente de mayor tamaño que la <Organización> que busca personalizar y volver único el UID.

Imágenes en DICOM

En el diccionario de datos DICOM hay una serie de propiedades que contienen la información básica de cualquier imagen, como alto, ancho y bits por píxel, pero lo realmente interesante es la imagen en sí que se encuentra almacenada en (7FE0, 0010) "*Pixel Data*".

DICOM soportan un amplio rango de formato de imágenes para almacenar, los específicos de DICOM muy similares al BMP sin compresión, pero con ciertas variaciones a la hora de empaquetar el píxel. Así como una serie de formatos estándar aceptados por DICOM, que se encuentran JPEG, RLE (*Run-Length Encoding*), ZIP, JPEG2000 y JPEG-LossLess. Donde además soporta sus técnicas de compresión. Sin embargo DICOM no es solo un formato de archivo, sino que también es un protocolo de transferencias de red.

DIMSE (*DICOM service exchange*)

De la misma manera que los humanos, los diferentes AE envían mensajes de servicio entre ellos. Estos mensajes se encuentran controlados por el set de protocoles DIMSE, siendo el corazón de la red DICOM, cada servicio DIMSE usualmente tiene un mensaje de solicitud y de respuesta, estos mensajes son: comprobar, almacenar, solicitar, cancelar, obtener y mover (*ECHO, STORE, FIND, CANCEL, GET, MOVE*) paquetes DICOM.

Luego de esta breve descripción de los aspectos principales del estándar, se puede resumir que DICOM es una serie de reglas que permiten la interoperabilidad entre equipos médicos, definiendo protocolos de red, en una serie de comandos conocidos, además de una sintaxis definida para la comunicación. Sin embargo dejan libertad con respecto a los

detalles del desarrollo y el número funciones implementadas. Es por esto, que se trata del estándar de facto utilizado en el área médica en la actualidad.

Capítulo 2. Marco Metodológico

En este Capítulo se expone el problema a resolver, los objetivos propuestos y la solución. Igualmente, se indicaran los objetivos generales y específicos logrados y finalmente, se expondrán las herramientas a utilizar para este desarrollo

2.1 Definición del problema

En el área de la imagenología médica, el despliegue de imágenes se ha vuelto una práctica de uso extendido, debido a que permite modelar y obtener resultados en menor tiempo. Existe una gran variedad de estudios y trabajos realizados con respecto al despliegue de volúmenes. La mayoría de estos análisis, están destinadas a equipos de alta potencia o para sistemas operativos específicos. Es por esto que resulta atractiva la creación de sistemas basados en web de bajo costo, que permita el despliegue de volúmenes e imágenes, en cualquier navegador web.

2.2 Propuesta de Solución

Como solución al problema planteado, se propone un sistema colaborativo en red de despliegue de datos médicos bajo el nombre DCMPal, que permite el acceso rápido y libre a datos médicos, así como la visualización de imágenes 3D de manera grupal. Permitiendo la interacción entre un expositor y sus oyentes para la discusión y análisis de los despliegues, que pueden estar ubicados en diferentes computadores. A continuación se presentan los objetivos que se cumplieron en este desarrollo.

2.3 Objetivo General

Desarrollar una aplicación web que sea capaz de desplegar volúmenes en formato DICOM de manera colaborativa desde un sistema de almacenamiento remoto, que este enfocado al uso en dispositivos que posean un navegador web con capacidades HTML5.

2.4 Objetivos Específicos

- Desarrollar una aplicación web para el despliegue de volúmenes específicamente con carga del volumen el servidor.

- Facilitar la visualización a múltiples usuarios de un mismo volumen en formato DICOM de manera colaborativa.
- Implementar sistema de intercambio de mensajes entre los oyentes
- Implementar un algoritmo de Ray Casting eficiente para el despliegue del volumen.
- Crear un diseño para la presentación de las herramientas de manera correcta en diferentes navegadores.

En las próximas secciones se procederá a explicar cómo fue implementado el sistema DCMPal y como llevo el desarrollo los objetivos propuestos.

2.5 Metodología

La aplicación Web que tiene como nombre DCMPal, utiliza el patrón de diseño MVC (Modelo-Vista-Controlador), que fue desarrollado utilizando ASP.NET MVC en su versión 4 y OpenGL para la gestión y despliegue de volúmenes. El hardware requerido para su ejecución fue un equipo de plataforma x86 con las siguientes características y librerías.

- Windows 8 Como sistema operativo.
- Microsoft Visual Studio 2012 como ambiente de trabajo
- Lenguaje C# en su versión 5.0.
- OpenTK 1.1 librería para utilizar OpenGL sobre C#.
- DCMTK 2.4.2 librería con soporte a diferentes lenguajes de desarrollo para manejo de red y archivos con el protocolo DICOM.
- SignalR 2.0 soporte en tiempo real para ASP.NET.
- DCM4CHEE 3.3.2 como almacén y gestión de modelos de pruebas.
- IIS 8.0 Para Albergar el sitio WEB.

En el siguiente capítulo se procederá a explicar todos los detalles pertinentes con respecto al desarrollo de la herramienta DCMPal, así como la explicación de sus características mas relevantes.

Capítulo 3. Diseño e Implementación de DCMPal

Este capítulo pretende exponer como se realizó la implementación del trabajo especial de grado, tomando como referencia los trabajos [**2**] [**3**] [**4**] [**22**] y [**5**], sobre el despliegue de volúmenes con cálculos realizados en el servidor. Primero se explicaran las características básicas de la solución, después se procederá a explicar las herramientas de terceros utilizadas en la implementación. Luego se expondrá los detalles y funcionamiento de las clases involucradas con el despliegue y obtención de modelos de los servidores PACS. Finalmente, se explicarán las pantallas, los controles y los JavaScript utilizados.

3.1 DCMPal

La solución propuesta como trabajo especial de grado DCMPal es una aplicación web, realizada utilizado un patrón de desarrollo MVC (Model-View-Control) [**30**], donde como modelo se encuentran las tareas la recuperación y creación de imágenes de despliegue, de controlador una serie de acciones orientadas a monitorear los eventos de los usuarios en modo de clases y finalmente como vistas, un sitio web que permite la interacción del usuario con la aplicación.

Este proyecto que tiene como finalidad el despliegue de datos médicos contenidos en un administrador de imágenes y archivos (Ej. Dcm4chee) de manera colaborativa en forma semi-masiva, que utiliza como lenguaje del servidor C# para las clases y el código oculto; y Razor/HTML como motor de despliegue. A continuación se expondrá el diagrama de red en conjunto de la configuración utilizada por defecto.

3.2 Características Básicas

El sistema web, está pensado para ser utilizado dentro de una red interna, ya que las necesidades de ancho de banda pueden aumentar a más de un megabyte por segundo, lo cual puede ser inviable para ser utilizada de manera remota. Esta es la configuración inicial utilizada por el sistema, asumiendo direcciones IP internas (Ilustración 9).

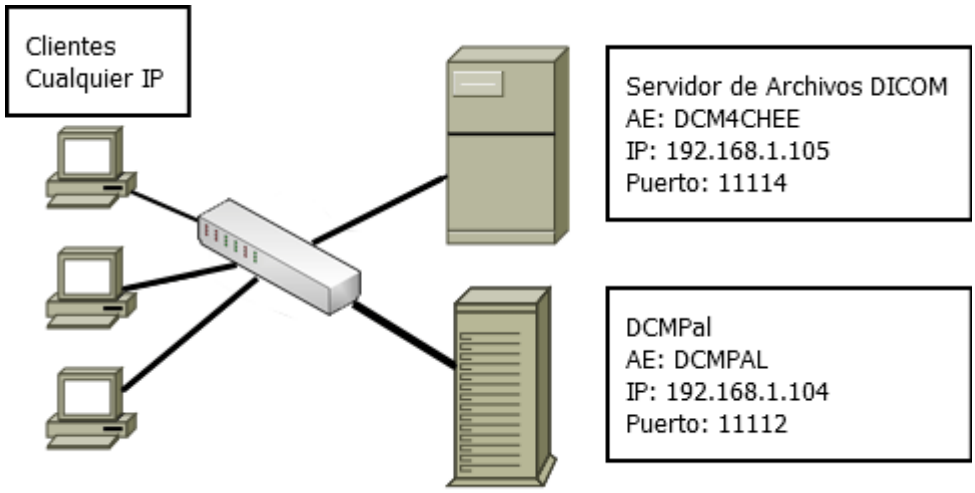

Ilustración 9 Diagrama de Red utilizado en DCMPal

Primero es necesario un sistema gestor de archivos DICOM, para la configuración inicial, se encuentra ubicado en la dirección de red 192.168.1.105, y tenga como nombre de AE "DCM4CHEE", donde el este asignado el puerto 11112 para las solicitudes. Además, que esten habilitada las conexiones desde un AE con el nombre "DCMPAL", que reciba las respuestas por el puerto 11114 (no es necesario para DCM4CHEE conocer el IP del DCMPal).

Ahora por parte de la aplicación DCMPal, se encuentra ubicada en el IP 192.168.1.104, la misma, tiene configurada en el servidor IIS el puerto 3224 para las conexiones remotas a la aplicación, las mismas pueden venir de cualquier subred, por esta razón, los clientes utilizando el navegador de su preferencia deben solicitar la página ubicada en http://192.168.1.104:3224.

Es importante destacar, que las direcciones de red y puertos asignados, no son fijos y pueden ser modificados desde la aplicación modificando el archivo de configuración básica del sistema Web.Config (archivo que contiene todas las configuraciones necesarias para el despliegue de la aplicación web), ubicado en la raíz del programa. A continuación se pasaran a explicar las librerías externas utilizadas en este desarrollo.

3.3 Bibliotecas Externas

La aplicación Web cuenta con una serie de librerías de terceros que facilitan diferentes tareas que deben ser realizadas dentro del sistema, como la comunicación entre el servidor de imágenes y la aplicación, así como el despliegue de las imágenes en tiempo real.

3.3.1 Grassroots DICOM (GDCM)

GDCM [**31**] es una implementación de una parte del Estándar DICOM diseñado bajo el modelo de desarrollo open source, para permitir a los desarrolladores acceder a datos clínicos directamente. GDCM incluye la definición del formato de un archivo DICOM y parte de los protocolos de comunicación, para la creación de herramientas completas de visualización. Entre los tipos de codificación de imagen que soporta se encuentran. RAW, JPEG lossy 8 y 12 bits, JPEG lossless 8 y16 bits, JPEG 2000 y RLE.

Siendo enteramente open source, se encuentra escrito en lenguaje C++.Sin embargo los desarrolladores ofrecen un envoltorio (*wrap*) para los lenguajes, C#, Python y Java. No obstante, este debe ser creado por el usuario.

Creación del envoltorio para C#

Prerrequisitos: Tener instalado CMAKE [**32**], SWIG [**33**] y los binarios de GDCM.
Proceso:

1. Ejecutar CMAKE y seleccionar el paquete descargado de GDCM
2. Indicar los siguientes parámetros:
    a. GDCM_BUILD_SHARED_LIBS: True
    b. GDCM_BUILD_EXAMPLES: True
    c. GDCM_WRAP_CSHARP: True
    d. DESIRED_CSHARP_COMPILER_VERSION: 3
    e. GDCM_BUILD_TESTING: False
3. Configurar en CMAKE
4. Aceptar

Después de esto se generará un directorio con una solución de Visual Studio, una vez abierta la solución, se procede a generar la solución en *Release*. Esta operación creará un conjunto de *dll* y entre ellas se encontrará gdcm-sharp.dll que será la utilizada como referencia en los proyectos (es importante para el correcto funcionamiento de la *dll* copiar de igual manera todas las otras *dll* creadas dentro del nuevo proyecto).

3.3.2 SignalR

SignalR es una biblioteca para desarrollo en ASP.NET que facilita el proceso de añadir funcionalidad en tiempo real a una aplicación Web. Esta funcionalidad se puede definir como la capacidad que tiene un servidor de enviar información a los clientes en el momento

que esta se encuentra disponible, eliminando la necesidad del servidor de esperar una petición de los clientes.

Esta biblioteca provee un API de desarrollo para la creación de llamadas del servidor al cliente (RPC), que se comunica con funciones JavaScript incluidas en la plataforma del cliente, además permite manejo de la conexión como agrupación conexiones.

Para lograr la comunicación entre el cliente y el servidor, la biblioteca se adapta a las capacidades del cliente, utilizando *websocket* si este está disponible, o por el contrario utilizando las tecnologías de socket disponibles en los casos que no es posible Ajax y JavaScript para funcionar, creando una capa de abstracción de las implementaciones propias de cada sistema del cliente (Ver Ilustración 10).

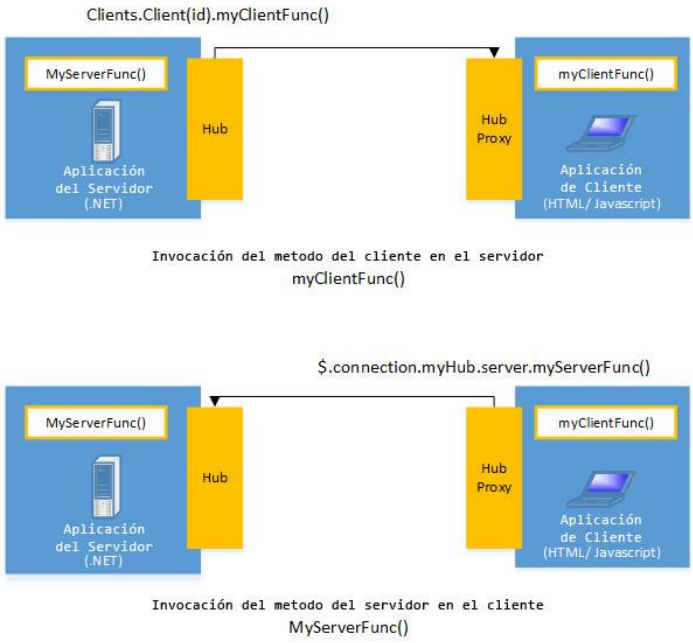

Ilustración 10: Funcionamiento de SignalR

Comunicación entre los clientes y el servidor

En la API de desarrollo existen dos modelos de comunicación entre el cliente y el servidor: conexión persistente y hub. La primera de estas representa una herramienta para el envío sencillo a un solo destinatario, grupo o mensajes de difusión. Este tipo de comunicación da acceso de bajo nivel incluido en la biblioteca. El hub por su parte presenta un mayor nivel de abstracción de la conexión persistente, permitiendo a los clientes llamar a métodos en el servidor como si estos fueran métodos locales.

Estas bibliotecas de terceros antes comentadas son utilizadas por las diferentes clases dentro del programa, encargadas de la obtención, manejo y despliegue de imágenes, todas estas clases conforman el modelo de la aplicación.

3.4 Estructuras de Datos

El siguiente diagrama explica cómo está compuesta la aplicación.

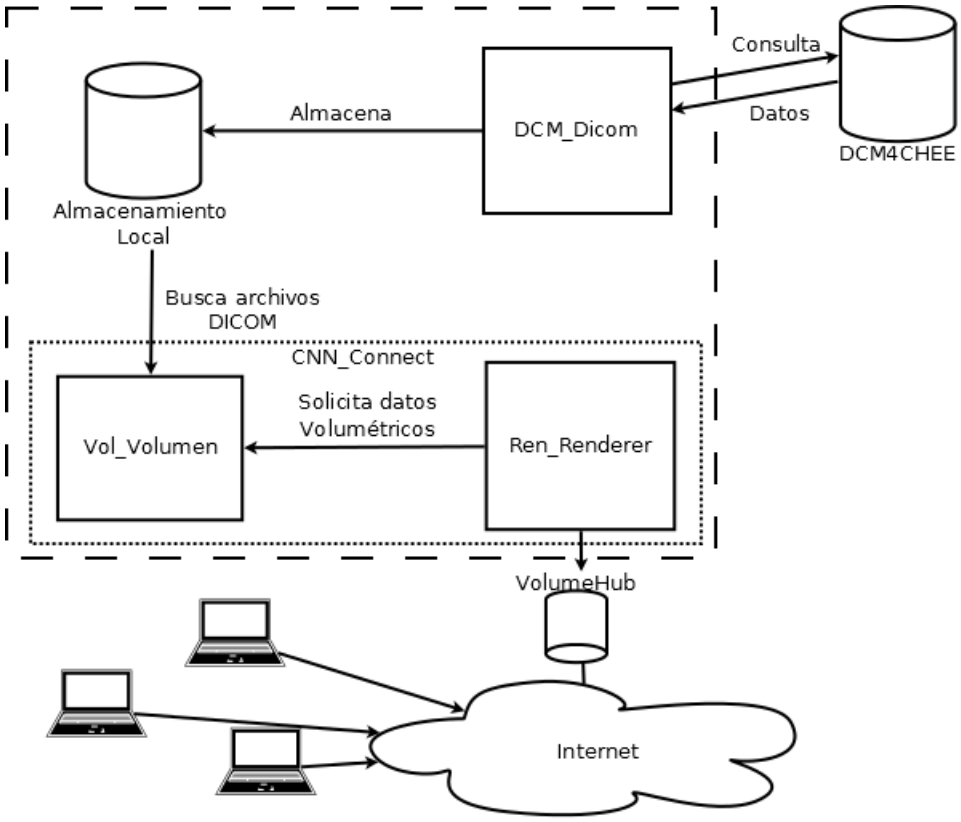

Ilustración 11: Diagrama de clases de despliegue de la aplicación

En la Ilustración 11 se puede ver el diagrama completo del trabajo de las clases dentro de la aplicación, el mismo se ejemplifica el funcionamiento de petición de modelos al servidor de almacenamiento, esto ocurre cuando un cliente solicita a través de la página web que se realice el despliegue de un volumen, la página invoca a las funciones encontradas dentro de *Ren_Renderer* para cargar el volumen.

Una vez iniciado el proceso de carga, *Ren_Renderer* solicita una copia del modelo a *Vol_Volumen* para poder realizar el despliegue, es entonces cuando *Vol_Volumen* verifica la existencia del modelo dentro del archivo local. De no existir una copia, se invoca a *DCM_Dicom* para que se conecte al almacén remoto de archivos para obtener una copia local de los archivos solicitados para la creación del modelo. En la siguiente sección se explicaran los detalles de las clases orientadas a la obtención y manejo de datos con mayor detalle.

3.4.1 Clase DCM_Dicom

Esta clase contiene un conjunto de los campos contenidos dentro de un archivo DICOM, además de contener los métodos de lectura y desempaquetado de este tipo de archivos. Otra de las características de esta clase es la capacidad de gestionar la comunicación entre la aplicación DCMPal y un PACS (*dcm4chee*). Para lograr esta tarea, la clase contiene dos métodos públicos.

Método ReadFromServer/DCMconnect

Este método contiene una serie de argumentos que serán usados como filtros para la solicitud que se creara para el servidor. Esta solicitud es hecha utilizando el método *CFIND* incluido dentro de la clase *CompositeNetworkFunctions* que forma parte de la librería *GDCM*. Este herramienta solicita una dirección de Red indicando la ubicación del PACS, el puerto, un argumento BaseRootQuery (pertenece a la librería *GDCM*), el cual se le agregaron todos los argumentos indicados como filtros y que serán enviados al servidor y como último incluye el AE del servidor del SCU creado por el mismo. Además retorna un tipo de dato listado de *DCM_Dicom* como resultado de la consulta al servidor y cada uno de estos indica los parámetros solicitados en la consulta.

Método ReadSeries

Gracias a este método es posible solicitar al servidor PACS los archivos DICOM solicitados y poder crear una copia local que será utilizada para el despliegue de volúmenes e imágenes, este método también utiliza la librería GDCM y utiliza el método incluido CMOVE que crea tanto un SCP como un SCU para crear la solicitud y poder gestionar la obtención de los archivos, entre los argumentos que solicita se encuentran el IP del servidor, el puerto del servidor, el puerto del SCP, BaseRootQuery indicando la serie a solicitar, el AE del servidor y como último parámetro la carpeta destino dentro del equipo. Entre los argumentos que este método solicitan se encuentran el UID de la serie a solicitar, el nombre del usuario realizando la solicitud, y un booleano que permite indicar que en

caso de existir la carpeta solicitada ya en el archivo local, si se tiene que sustituir o dejar intacto.

Método ReadFile

Este método es ampliamente utilizando dentro de DCMPal y permite leer una serie de datos incluido dentro de un archivo DICOM, retorna un *DCM_Dicom* con esos campos.

3.4.2 Clase DCF_Folder

La Clase *DCF_Folder* cumple con la tarea de administración y manejo de los archivos locales, incluye tres métodos orientados a la lectura del archivo local, listado de carpetas que forman parte a un espacio de trabajo de un usuario y despliegue de imágenes. Entre los métodos que esta incluye están los siguientes.

Método DCF_GetList

Este método utiliza el *namespace de System.IO* para poder acceder a las carpetas dentro del servidor, donde utilizando la Carpeta indicada en el Web.Config y el UID de la serie, obtiene el nombre de los archivos dentro de esta carpeta y los agrega a un listado de cadena de caracteres (*String*) una vez completada esta tarea retorna ese listado.

Método DCF_GetFolder

El Corazón de la selección del modelo a desplegar es el método *DCF_GetFolder*, este lee el espacio de trabajo de un usuario. Retorna un listado de *DCM_Dicom* con las características de las series que se encuentran alojadas dentro de su espacio de trabajo.

Método DCF_GetImg

Pilar del listado de imágenes, *DCF_GetImg* gestiona en conjunto con la librería GDCM el despliegue de la imagen contenida dentro de los espacio de trabajo de un usuario en la aplicación. La misma obtiene un arreglo de bytes por parte de un *gdcm.Image*, que en conjunto con la clase *gdcm.ImageReader* lee un archivo DICOM y obtiene los aprometeros básicos, como alto, ancho y el tipo de dato en que se encuentra dentro del arreglo de bytes. Estos serán utilizados para crear una imagen y almacenarla en un Bitmap.

3.4.3 Clase CNN_Connect

Esta clase estática, contiene instanciadas los hilos necesarios para la ejecución de las diferentes instancias de *Ren_Renderer*. También incluye las instancias de los semáforos de procesos utilizados en la aplicación para controlar las peticiones al servidor PACS.

3.4.4 Clase Log_Logger

La clase *Log_Logger* fue utilizada dentro del desarrollo como medio de obtención de datos para análisis de las pruebas del sistema en forma de archivo plano, la misma contiene solo los métodos *Append* y *writetolog* el primero de ellos agrega una nueva línea a una cadena de caracteres incluida dentro de la clase, mientras la segunda escribe esos datos a un archivo plano. Los datos contenidos dentro de estos archivos fueron utilizados para optimización de varios aspectos del sistema.

3.4.5 Clase Roo_Room

Esta clase cumple con la tarea de informar sobre los estados de las salas de exposición, dando a conocer si la misma se encuentra en uso y quien es el expositor. La clase, contiene un solo método nombrado *getRooms* que permite saber el estado de todas las salas en forma de listado.

3.4.6 Clase Gen_Generic

Esta clase estática que contiene todos los métodos que pueden considerarse de uso común en la aplicación pero que no forman parte de ninguna clase en sí. Facilita tareas como conversión del formato de fechas entre DICOM y C# y métodos transformación de datos básicos.

Todas las clases mencionadas anteriormente, cumplen con la tarea de obtener datos y manejo lógico del sistema web sin involucrarse de ninguna manera en la creación ni despliegue del volumen. A continuación se expondrá los aspectos involucrados en el despliegue del volumen en conjunto con las clases involucradas.

3.4.7 Clase Shader_Sha

Utilizando la Clase *Shader_Sha*, es posible la compilación y creación de vínculos del programa con OpenGL. Los métodos para estas tareas son *compileShader y Attachshader.*

Método compileShader

Cumple la función de compilar el código GLSL, recibe como argumentos el id del indicado por OpenGL para almacenar el *shader* y un cadena de caracteres con el programa a compilar. Además de esto, en caso de estar activada la consola de depuración, facilita ver las advertencias y errores en tiempo de compilación.

Método Attachshader

El método *Attachshader* se encarga de recuperar los diferentes *shaders*, obtener un ID en OpenGL para ellos y asociarlos a un programa de OpenGL. Para lograr esto, utiliza el método *compileShader* mencionado anteriormente, en conjunto con *GLAttachshader* y *GLLinkProgram* que pertenecen a OpenGL.

3.5 Carga del volumen

La carga del volumen es necesaria para el despliegue, ya que permite crear el modelo intermedio a partir de los diferentes archivos DICOM involucrados. Sin embargo, estos archivos contienen una serie de parámetros importantes a tomar en cuenta a la hora obtener la imagen, que son necesarios para la correcta visualización del modelo. Dentro de DCMPal, estas tareas son llevadas a cabo por la clase *Volume_Vol*.

Es importante destacar que los archivos DICOM en su mayoría, solo contienen una imagen por archivo, por lo que es necesario leer un grupo de ellos para crear el volumen. Por esto, *Volume_Vol* necesita crear una representación intermedia tridimensional de los datos para crear el volumen En la Ilustración 12 puede ver el proceso completo realizado en *Volume_Vol*.

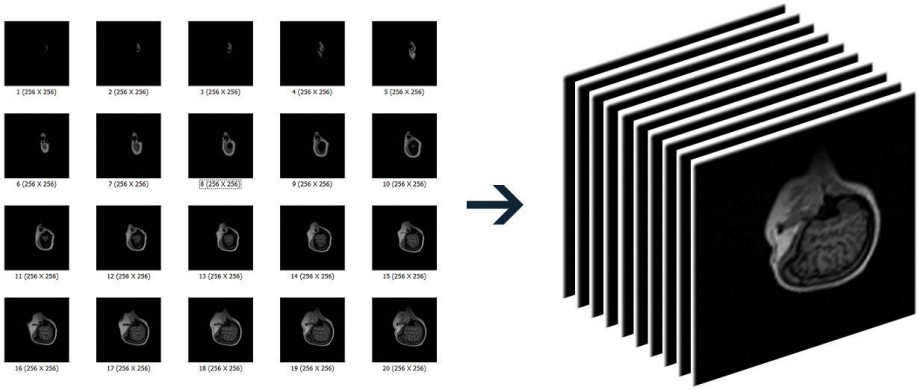

Ilustración 12: Proceso de carga del volumen, paso de imágenes a volumen.

Para poder obtener las imágenes (Ilustración 12) *Volume_Vol* busca todos los archivos DICOM alojados en una carpeta indicada (todos estos forman parte de una Serie), para realizar un recorrido a cada uno buscando extraer el *PíxelData* (el arreglo de píxeles de la imagen) dentro. Una vez obtiene estos datos, realiza una serie de transformaciones a estos datos para la correcta visualización, la primera que se necesita en conocer el tipo de escalar.

Como se habló en el Capítulo 1, un archivo DICOM puede almacenar distintos tipos de formato de imagen y aunque DCMPal este limitado a archivos sin compresión, el tipo de escalar que conforma el arreglo dentro del *PíxelData* (*tag* de la imagen dentro del archivo) puede ser variado, por lo que se valida y realiza la extracción.

Una vez obtenido el arreglo unidimensional, es recorrido para obtener cada una de las columnas de la imagen final, utilizando los valores *row* (filas) y *columns* (columnas) del archivo DICOM, una vez finalizada la creación de columnas, se llevan a cabo las transformaciones tomando en cuenta cual es el método de despliegue sugerido por el archivo DICOM (MONOCHROME1, MONOCHROME2, RGB, etc.).

Es importante destacar que *Volume_Vol*, lleva todos los tipos de escalar a un valor decimal entre 0 y 1, utilizando el valor máximo dentro del arreglo del escalar y dividiendo cada uno de los escalares internos por ese valor.

Cuando termina todas estas transformaciones, copia este arreglo al modelo intermedio tridimensional, que utiliza para la creación de la textura. La manera que ocurre esto es directa ya que las transformaciones fueron realizadas dentro del arreglo tridimensional.

En el último paso, *Volume_Vol* crea el ID en OpenGL y lo enlaza a la textura, utilizando como textura el arreglo tridimensional creado. Una vez que termina este proceso, se almacena como parámetro publico el *ID* de la textura generada.

Como ya se realizaron la textura para el despliegue del volumen, ya es posible desplegar el volumen. En la siguiente sección se explicará el algoritmo utilizado.

3.6 Algoritmo de Despliegue

Para el despliegue del volumen dentro de DCMPal se utilizó Ray Casting en GPU presentado por Kruger y Westermann [**22**], por calidad de imagen sin comprometer los tiempos interactivos, aún cuando exista una serie de despliegues simultáneos.

La clase encargada de este despliegue recibe el nombre de *Ren_Renderer*, que hereda de *GameWindow* incluida dentro de la librería OpenTK para enlazar C# con OpenGL, y utiliza los siguiente eventos: *Load y RenderFrame*.

3.6.1 Método Load

Evento que ocurre antes de realizar el primer despliegue, es utilizado para la inicializar los componentes principales como lo son: carga de modelo, compilación de *shaders* (esto incluye obtener los ID de las variables uniformes), creación de la ventana, creación de los *framebuffer* necesarios y habilitar diferentes capacidades dentro de OpenGL.

3.6.2 Método RenderFrame

Siendo el bucle principal de OpenGL, se encarga de la creación de los cuadros, es aquí donde se implementa Ray Casting de la siguiente manera. Primero, se realizan las traslaciones, rotaciones y además se eliminan todos los elementos dibujados en el dispositivo de salida. Una vez realizado esto, inicia el cálculo de las texturas de intersección. Las mismas son creadas utilizando dos buffers enlazados a texturas donde se dibuja un cubo unitario centrado en el eje que además se le asignan como colores de los vértices la posición el eje. (Como puede observarse en la Ilustración 13). El primer buffer, pintan solamente las caras delanteras (Entradas de los rayos), mientras que el segundo, se pintan las traseras (Salidas de los rayos), estas texturas son utilizadas por el *shader* para el cálculo del rayo que será pintado en una textura creada en la pantalla inicial.

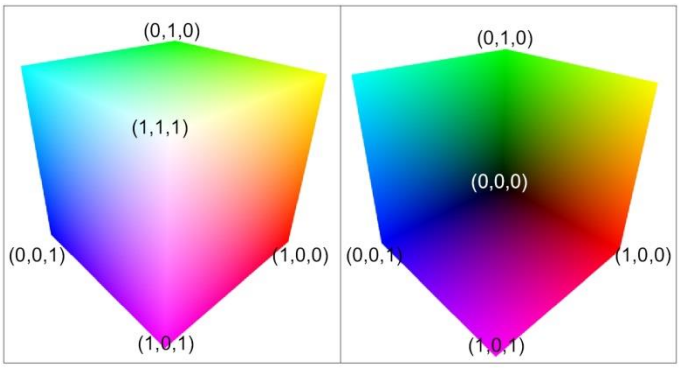

Ilustración 13 Texturas de entrada y Salida

Una vez obtenidas las dos texturas, las mismas son enviadas al *shader* en conjunto con otros parámetros para realizar el despliegue. Estas variables son las siguientes

```
uniform sampler2D back;
uniform sampler2D front;
uniform sampler3D volume;
uniform sampler2D transfer;
uniform float mylower;
uniform float myupper;
```

El siguiente cuadro explica cada una de ellas

Uniform	Uso
back	Textura de terminación de rayo
front	Textura de inicio del rayo
volume	Textura 3D del volumen a desplegar
transfer	Incluye la textura utilizada para implementar la función de transferencia
mylower	Valor mínimo de la ventana en para la implementación *windows/level*
myupperr	Valor máximo de la ventana en para la implementación *windows/level*

Usando este conjunto de variables es posible llevar a cabo el Ray Casting, así como también refinar el modelo final utilizando *windows/level* y una función de transferencia. A continuación se explicara cómo funciona el *fragment shader.*

Primero se obtienen los valores de entrada y salida del rayo dentro del volumen y se guardan en las variables *frontPos* y *backPos* respectivamente

```
vec3 frontPos = texture2D(front, gl_TexCoord[1].st).rgb;
vec3 backPos = texture2D(back, gl_TexCoord[1].st).rgb;
```

Una vez obtenido el rayo se invoca la función *directRendering* que tiene como argumentos punto de entrada, punto de salida, valor mínimo y valor máximo de la ventana, el resultado de esta función se retorna como el valor del fragmento final.

```
gl_FragColor = directRendering(frontPos, backPos, mylower, myupper);
```

La función *directRendering* primero calcula la dirección del rayo restando el punto final con el punto inicial almacenándolo en la variable *direction*. De ser el resultado de esta operación igual a cero, el cálculo rayo no es llevado a cabo para ese fragmento y se retorna el valor de fondo.

```
vec3 direction = last - first;
if(length(direction) == 0)
	return vec4(0.0,0.0,0.0,1.0);
```

Un ver realizada la verificación y el rayo realiza una intercepción con el volumen, se pasa a calcular cuantas muestras se van a tomar así como el la separación entre cada una, estas variables se almacenaran en *diff1* y *step* respectivamente.

```
int steps = int(floor(Samplings * length(direction)));
vec3 diff1 = direction / float(steps);
```

Se crea la variable donde se almacenará el valor final del píxel además el valor por el cual multiplicar el valor de la muestra que se encuentre dentro de la ventana

```
float scale = 1.0/(upperThreshold - threshold);
vec4 result = vec4(0.0);
```

Una vez creadas las variables antes mencionadas, se inicia el recorrido en cada una de las muestras, donde se toma el valor de cada una de las muestras y se verifica en donde se encuentra el punto, de esta manera si su valor es menor al umbral mínimo de la ventana es descartado y si es mayor es igualado al valor máximo. En caso de encontrarse dentro del umbral se realizan la operación necesaria de interpolación.

```
for (int i=0; i<steps; i++){
       value = texture3D(volume, first);
       first += diff1;
       if (value.r>= threshold) {
              if(value.r >=upperThreshold) value.r = 1.0;
              value.r = ((value.r-threshold)*scale);
```

Ya modificado el valor de intensidad, ahora es necesario aplicarle la función de transferencia, l amanera de hacer esto es buscando dentro de la textura la posición en la intensidad obtenida por el cálculo de las ventanas la ilustración XX explica el funcionamiento de la función de transferencia y de *window/level*.

```
vec4 texVal = texture2D(transfer, value.rr);
```

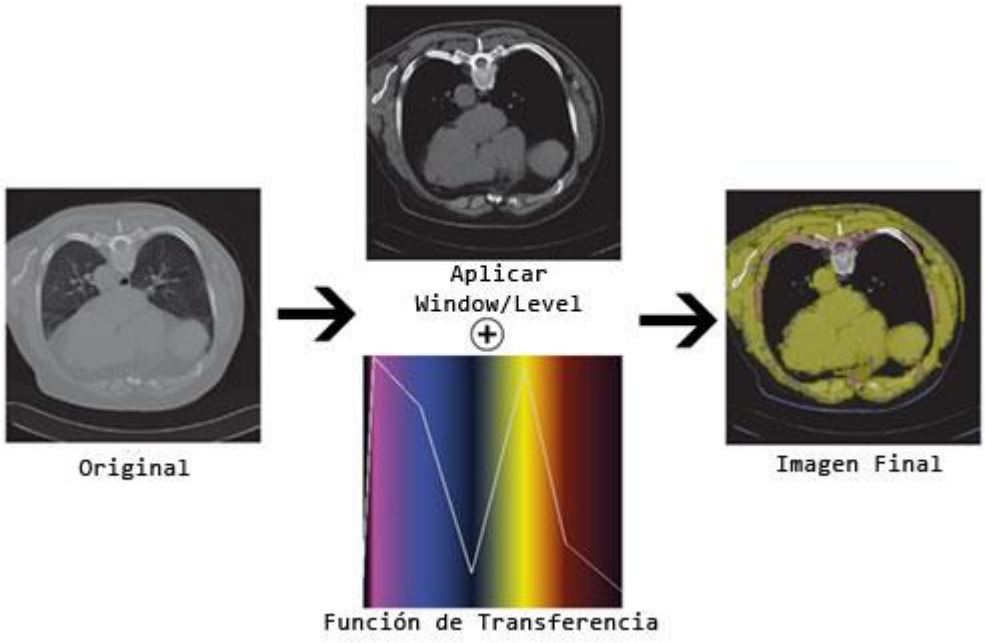

Ilustración 14: Aplicación de la función de transferencia en el *fragmentsahder*

En la Ilustración 14 se puede observar como la imagen original (izquierda), se le realizan las operaciones de *window/level* pertinentes, obteniendo la imagen del centro que al aplicarle la función de transferencia, da como resultado la imagen final (derecha).

Ya realizadas todas las operaciones necesarias sobre la muestra, se realizan los cálculos pertinentes para obtener cuanto influye esa muestra en el valor final del fragmento.

```
result.rgb += (1.0-result.a)*texVal.a*texVal.rgb;
result.a += (1.0-result.a)*texVal.a;
```

Todos valores finales de intensidad son acumulados en la variable *result* , que indica el color final del fragmento, sin embargo en algoritmo, utiliza una terminación temprana del rayo para cuando pasa el umbral del 90% de la intensidad máxima retorna ese valor como el final del rayo.

```
if (result.r >= 0.9) {
	return result;
}
```

El algoritmo anterior, despliega el volumen y además realiza todas las operaciones necesarias para obtener las zonas de interés, pero no lleva a cabo ninguna creación de la función de transferencia. A continuación se explica cómo se crea la función de transferencia

3.7 Textura para la función de transferencia

La textura función de transferencia esta implementada en la clase *Transfer_Tra,* la cual permite la creación, edición y limpieza de la función de transferencia, los métodos son los siguientes

3.7.1 Método GenerateIdentity

Este método genera la primera textura utilizada como función de transferencia por OpenGL, la misma se puede nombrar como la función identidad, donde el punto en *x*, indica el valor *rgba,* en la Ilustración 15a se ejemplifica esta textura (la línea verde es el valor de *alpha* entre 0 y 1).

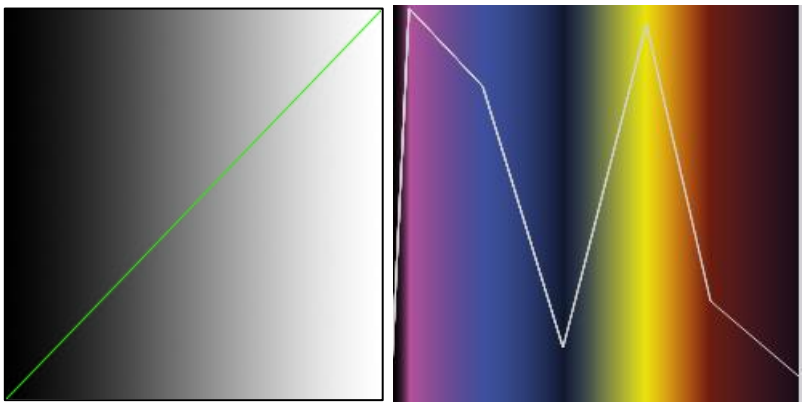

Ilustración 15: Texturas de función de transferencia a) identidad b) diferentes zonas de interés

3.7.2 Método Clearfunc

Utilizada para llevar al estado inicial la textura de función de transferencia, limpia el arreglo de puntos dentro de *Ren_Renderer* y solicita al bucle principal volver a calcular la textura.

3.7.3 Método Addpoint

Añade nuevos puntos a la función de transferencia, reemplazando si este ya existe y en el listado. Además, indica que es necesario en el bucle principal llamar a la función *Calculatepoints***.**

3.7.4 Método Calculatepoints

Método también utilizado dentro del bucle principal, es llamado siempre que sea necesario rehacer la textura de la función de transferencia. Para llevar a cabo esta tarea interpola puntos almacenados dentro de la variable *GL_toidentity*, que utiliza para crear un arreglo donde almacena los valores de todos los puntos RBGA de la textura. El resultado final puede asemejarse a (Ilustración 15b).

La textura creada, necesita ser aplicada a la imagen final. Para lograr esto, existen dos maneras de implementarlo, vía CPU o GPU. La primera, descartada para el proyecto, busca reescribir todas las texturas de los tres conjuntos cada vez que se agrega un punto, esta aproximación no solamente es lenta, y además desperdicia mucho espacio de RAM para almacenar las copias originales, es por eso que se optó realizar por GPU, utilizando *shaders*.

Ya con todas las clases y métodos expuestos es posible la carga de los volúmenes, sin embargo, por ser una aplicación web, se necesita una clase que facilite que esta representación sea posible enviarla a los diferentes clientes.

3.8 Envío de cuadro a los clientes

Cada vez que un cuadro (imagen con el despliegue completado) es terminado en *Ren_Renderer* es necesario obtener el arreglo de bytes del mismo y transformarlo en un formato de imagen que sea desplegable de manera general en todos los navegadores, para lograrlo se utilizan los métodos *GrabScreenshot y ToBase64String* para la creación de la imagen y *Sendimage* para enviarla a los clientes.

3.8.1 Método GrabScreenshot

Este método retorna un bitmap con un los píxeles leídos del cuadro, se le deben pasar como argumento argumentos el alto y ancho de la imagen de salida, que utiliza para indicar cuantos píxeles debe capturar de la pantalla de OpenGL.

3.8.2 Método ToBase64String

Este método crea una cadena de caracteres en base64 de una imagen. Para lograr esto, se pasan como argumentos un bitmap y el formato de imagen, que son usados para crear la cadena de caracteres de un tipo de imagen específico, entre los posibles a utilizar se encuentra *gif*, *jpeg*, y *png*.

La razón por la que se utiliza *base64* para enviar a los clientes, es por las características propias de *signalR*, donde los parámetros son enviados como arreglo de caracteres para ser interpretados por *JavaScript*, por lo que se necesita otro método de interpretación de imágenes. Los navegadores actuales permiten la representación de imágenes en formato *base64* y se aprovecha esta característica para desplegar la imagen en los clientes.

3.8.3 Método Sendimage

Esta clase es utilizada para enviar la imagen a los clientes conectados en una sala de exposición. Logra esto, enviando el arreglo correspondiente a una imagen a un grupo de la sala de expo a un evento JavaScript (*responserfirt*), este listado es manejado por *GlobalHost*, que forma parte de la librería *SignalR.*

Es importante destacar que SignalR, no solo es utilizado para el envío en tiempo real de la imagen a los clientes, también es necesario para obtener cambios realizados al modelo por parte del cliente.

3.9 Intercambio de datos cliente-servidor

Para lograr ese intercambio de manera transparente, se utilizó un Hub (forma parte de la librería *SignalR*), que permite la comunicación “directa” entre JavaScript y la aplicación del servidor como si pertenecieran a la misma aplicación. Gracias a esto, el cliente puede enviar comandos para la modificación de los aspectos del volumen, así como también comunicarse por el chat. Los métodos de interconexión entre la web del cliente y el servidor pueden observarse en el siguiente cuadro.

Getwl	Recibe del cliente un valor para el window/*level* que son asignados a la sala de exposición específica para el refinamiento del volumen.
Scale	Permite ampliar o disminuir le modelo presentado en una sala de exposición.
Rotate	Se encarga de realizar las rotaciones de *x, y, z*. Para hacerlo se le pasan como argumentos dos booleanos, el primero que indica si va aumentar o disminuir la rotación, el segundo quien de los tres ejes es el afectado y por último la sala de exposición del modelo.
Changeprogram	Indica cual programa de *shader* se utilizara en el despliegue del volumen de la sala de exposición específico, indicando si usar umbral o *window/level* en conjunto con función de transferencia.
Newvalue	Añade un nuevo punto a la función de transferencia, se le indican como argumento los valores de la posición, los valores *rgba* y la sala de exposición afectada.
Clearfunc	El método limpia la función de transferencia, llevándola al estado inicial.

Tabla 3 Métodos JavaScript Cliente-Servidor

Todas las clases anteriormente mencionadas, forman parte del código del servidor. Gracias a ellas es posible desplegar el modelo de manera correcta en un navegador. En la siguiente sección se explicara los aspectos más importantes del código *JavaScript*.

3.10 Ejecución del lado del cliente

Como antes se mencionó, gracias a la clase Hub incluida en la librería SignalR, es posible la comunicación entre el cliente y el servidor de manera transparente. Par lograr esto es necesario es necesario crear una variable de la siguiente manera

```
var hub = $.connection.volumehub;
```

Esta línea solicita y conecta todos los métodos encontrados en *volumehub* con el cliente, además de gestionar le pase de mensajes entre ellos. Sin embargo, es buena práctica no permitir el uso del mismo hasta que sea totalmente cargado, es por eso que todos los eventos que ocurren del cliente al servidor están encapsulados dentro del siguiente código.

```
$.connection.hub.start().done(function () { … }
```

De esta manera los eventos que son generados desde el cliente no van a poder ser invocados hasta que no sea realizada la conexión completamente. Un ejemplo de una llamada del cliente al servidor es la siguiente:

```
hub.server.addme(room, username);
```

El método anterior, podemos ver que llama una función dentro de server, en este caso para agregar un usuario al chat. Podemos ver cómo se maneja como una llamada dentro del sistema. La siguiente llamada es del servidor al cliente.

```
hub.client.responsefirt = function (image)
{
 $('#img').attr("src", "" + image);
}
```

Se seleccionó el ejemplo anterior para ejemplificar la llamada del servidor al cliente, ya que esta es la llamada involucrada con la descarga de la imagen, se puede apreciar como utiliza la respuesta para modificar le atributo de la imagen a *base64.*

Además de estos códigos antes expuestos, existen un conjunto de funciones que controlan el funcionamiento del *windows/level* y función de transferencia, así como también para manejo de los datos a visualizar, sin embargo, no son necesarios para la compresión de esta implementación. En la siguiente sección se explicará la lógica de los controles de la aplicación.

3.11 Controles de vistas

Los controles en una aplicación web basada en el patrón MVC responden a los eventos (acciones del usuario) e invoca peticiones de datos cuando los eventos hacen solicitud de información. Además intercambia información con las páginas asociadas, para desplegar diferentes tipos de información, en resumen sirve como intermediario entre el modelo y la vista.

3.11.1 Clase DCMIMGController

Este Controlador, maneja las páginas orientadas al despliegue del listado de imágenes con un conjunto de los datos contenidos dentro de los archivos DICOM, indica que modelos deben ser obtenidos del PACS, y se encarga de entregar las imágenes con las transformaciones necesarias para ser visualizadas por el navegador. Las acciones que contiene son las siguientes.

Solicitud de imágenes (*Imgcontainer*)

En conjunto con su vista asociada, sirve como espera del contenedor para el listado de imágenes, utiliza el evento .load de JavaScript para obtener de manera asíncrona las imágenes (utilizando el control *imglist*) y el número de páginas asociadas a esas imágenes.

Listado de imágenes (*Imglist*)

Solicita a DCM_Dicom Obtener una serie desde el servidor de DICOM, después obtiene el listado de las imágenes obtenidas que pasa a la vista para ser desplegadas. Sin embargo es importante destacar, que este listado solo obtiene los nombres de las imágenes, es necesaria la acción *getimg* para obtener la imagen.

Despliegue de Imágenes (*Getimg*)

Acción encargada del despliegue de la imagen, requiere como parámetros indicar el UID de la imagen a desplegar, parámetro que utiliza para invocar al método DCF_GetImg de la clase DCF_Folder, utilizando el valor de retorno como respuesta del servidor a este control (obtienen una imagen como resultado final).

3.11.2 Clase DCMPalController

Es el control destinado al despliegue de la información contenida en el servidor de imágenes, permite obtener un listado de los datos del servidor con la posibilidad de utilizar filtros para obtener la información, contiene una serie de acciones para obtener los diferentes niveles información (Pacientes, Estudios, Series).

Listado de Pacientes (*Patient*)

Obtiene los datos de los pacientes que se encuentra en el servidor, los datos pueden ser filtrados por ID del paciente o por parte del nombre.

Listado de Estudios (*Study*)

Acción que recibe un *PatientID* para poder solicitar a DCM_Dicom que obtenga todos los Estudios Realizados a ese paciente.

Listado de Series (*Series*)

Acción que recibe un *PatientID* y un *StudyInstanceUID* para poder solicitar a la clase DCM_Dicom que obtenga todas los las series realizados en un estudio a un paciente

3.11.3 Clase DCMVOLUMEController

Este control, maneja toda la lógica involucrada en la selección de los modelos así como realización de las redirecciones necesarias para brindar las pantallas de despliegue de expositores y oyentes.

Verificación de Disponibilidad (*Index*)

La acción por defecto del control, valida si la sala de exposición a utilizar se encuentra disponible para exponer, de encontrarse disponible, crea nuevas instancias de Ren_Renderer, y el da al usuario solicitando la sala el puesto de oyente y pasa el estado de la sala a "en exposición".

Sala de Oyentes (*Volumeloader*)

Acción para el despliegue de la pantalla de oyentes, indica cual el nombre de usuario para utilizar en el Chat.

Selección de Volumen (*Volumeselector*)

Acción para el despliegue del listado de modelos disponible en el espacio de trabajo del usuario que solo puede ser accedida por el expositor de la sala. Utiliza el método *DCF_GetFolder* que forma parte de *DCF_Folder* para recuperar el listado de modelos.

Sala de Administrador (*Volumegenerator*)

Acción únicamente accesible por un usuario expositor, pasa los parámetros necesarios para el despliegue del volumen y reinicia el hilo de ejecución del despliegue.

Todos estos controles funcionan como intercambio entre las vistas de usuario y los modelos, pero requieren de un despliegue.

3.12 Vistas de usuario

El siguiente apartado incluye las vistas del usuario, que permiten la interacción y la visualización por parte del cliente. Las vistas tiene el siguiente mapa de navegación (Ilustración 16).

.

Ilustración 16 Mapa de Navegación en DCMPal

A continuación se explican en mayor detalle cada una de las vistas presentadas.

3.12.1 Vista de Inicio.

La pantalla inicial del sistema, mostrando un *changelog*, que se actualiza cada nueva versión (Ilustración 17).

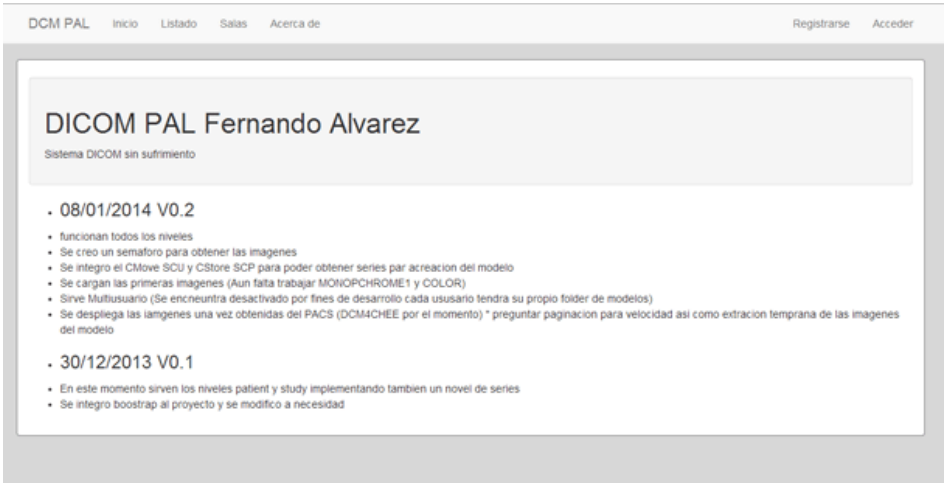

Ilustración 17: Vista de inicio

3.12.2 Vista de Ingreso.

La pantalla permite ingresar el usuario a la aplicación, el da acceso al usuario a la zona con restricción (Ilustración 18).

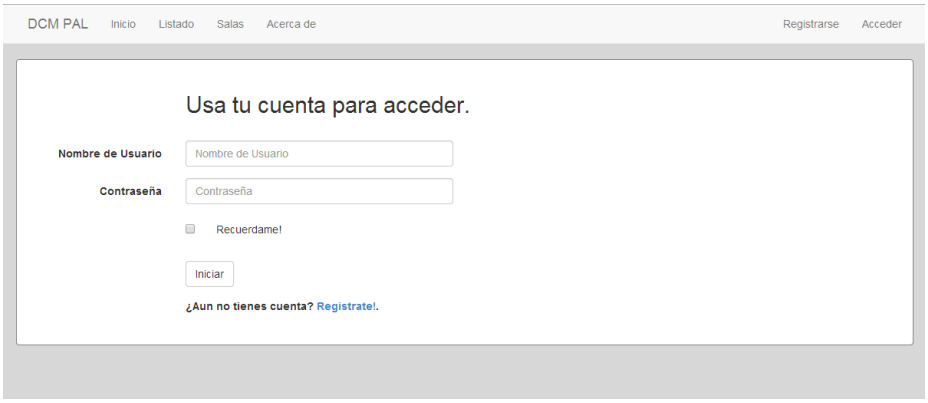

Ilustración 18 Vista de ingreso

3.12.3 Vista de Creación Cuenta.

Permite ingresa al sistema de cuentas incluido en ASP.NET un nuevo usuario para el sistema. Para realizar el registro solo es necesario un usuario y una contraseña (Ilustración 19).

Ilustración 19: Vista de registro de usuario

3.12.4 Vista de selección de sala de exposición.

Esta pantalla permite seleccionar entre las diferentes salas de despliegue, indicando si ya se encuentra un expositor impartiendo o la sala está libre para iniciar una exposición (Ilustración 20).

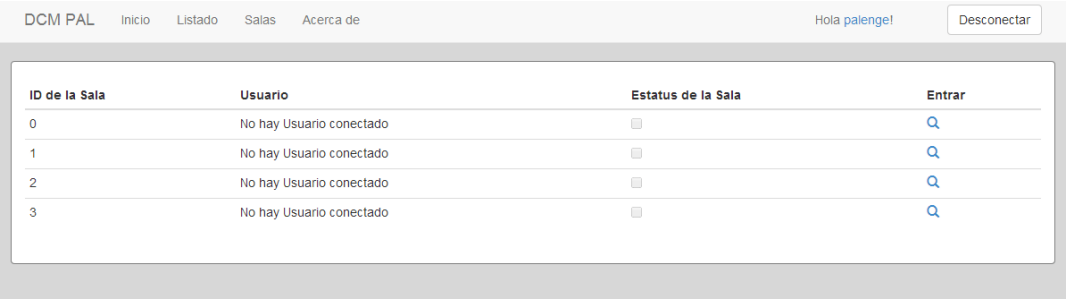

Ilustración 20: Selección de sala de exposición

3.12.5 Vista de selección de Modelo.

Pantalla que aparece en caso de ser expositor, despliega la información de todos los volúmenes descargados por el usuario usando el sistema de listar. Es importante destacar que la información aquí desplegada esta filtrada por series, la selección puede ser realizada por series únicamente (Ilustración 21).

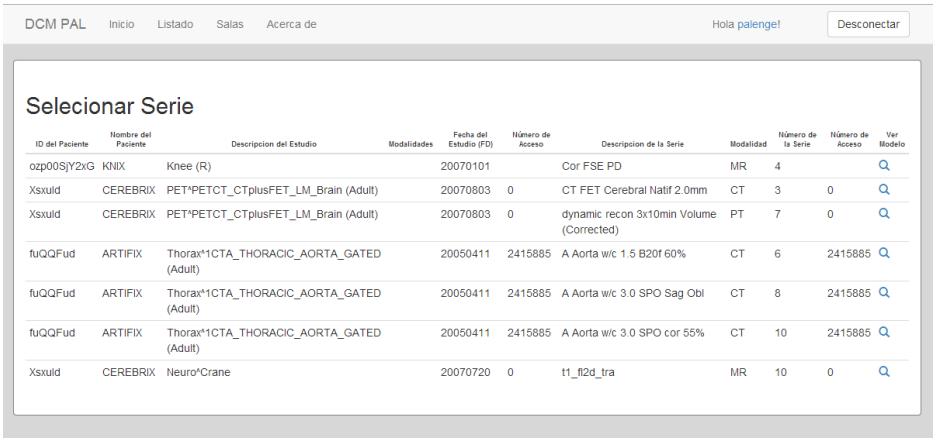

Ilustración 21: Vista de selección de serie

3.12.6 Vista de oyente.

Pantalla que aparece al acceder a una sala de exposición como oyente, en la misma se puede ver el chat para comunicación entre los oyentes y el expositor, mientras el expositor selecciona el modelo, un símbolo de carga aparece en la zona de la imagen (Ilustración 22).

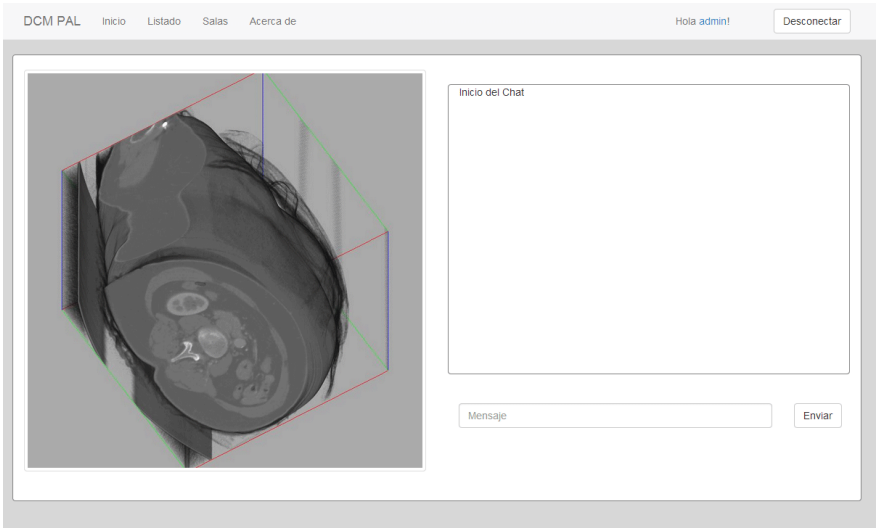

Ilustración 22: Vista de oyente

3.12.7 Vista de expositor.

Pantalla principal de despliegue de volumen, la misma solo puede ser accedida por un expositor, en ella se controlan todas herramientas para la modificación del volumen como lo son: rotación, escalamiento, *window/level* y función de transferencia (Ilustración 23).

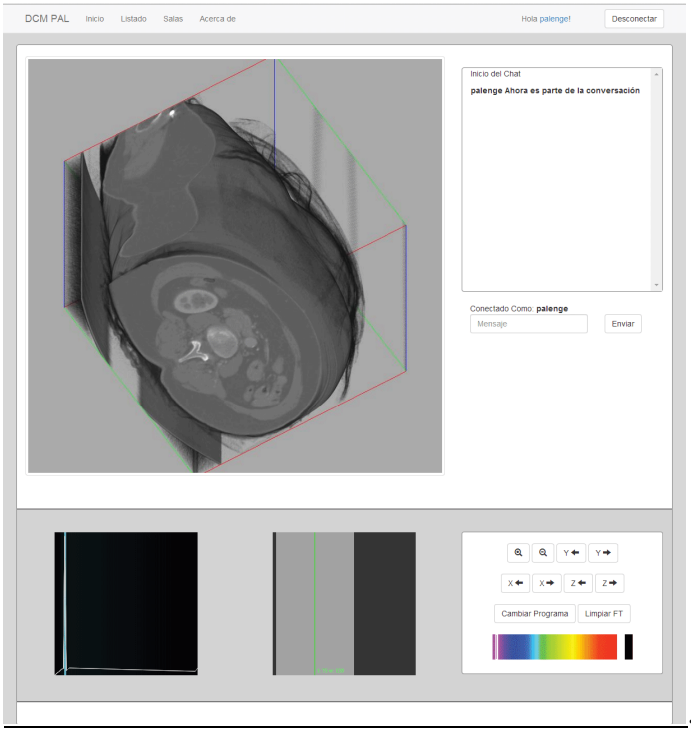

Ilustración 23: Vista de Exponente

Una vez revisado todos los aspectos incluidos dentro de DCMPal, podemos observar que es un sistema orientado a cualquier tipo de cliente, ya que todos los cálculos son realizados directamente en el servidor. Gracias a esto el sistema es fácilmente utilizado en cualquier navegador con pocas prestaciones, haciéndolo ideal como un sistema para instituciones de pocos recursos. A continuación se mostraran las pruebas realizadas al sistema.

Capítulo 4. Pruebas y Resultados

En este capítulo se analizarán los resultados obtenidos al aplicar diferentes pruebas al sistema de despliegue de DCMPal. Tomando en cuenta que el gran peso de la aplicación recae mayormente sobre el despliegue del volumen, las mediciones se llevarán a cabo analizando el despliegue del mismo. Primero se detallarán las características de los modelos y ambientes utilizados para realizar las pruebas. Luego se estudiarán los diferentes factores que afectan para el despliegue de volumen, como son el tamaño y el número de cortes que contenga. Finalmente se analizará el desempeño del envío de la imagen a los clientes utilizando diferentes formatos (*jpeg, gif, png*) y tamaños de imagen.

4.1 Ambiente de Pruebas

Los modelos contienen datos obtenidos de archivos DICOM, que contienen información de cada corte del modelo. Se utilizaron tres modelos diferentes tamaños: pequeña (Ilustración 24a), mediana (Ilustración 24b) y grande (Ilustración 24c).

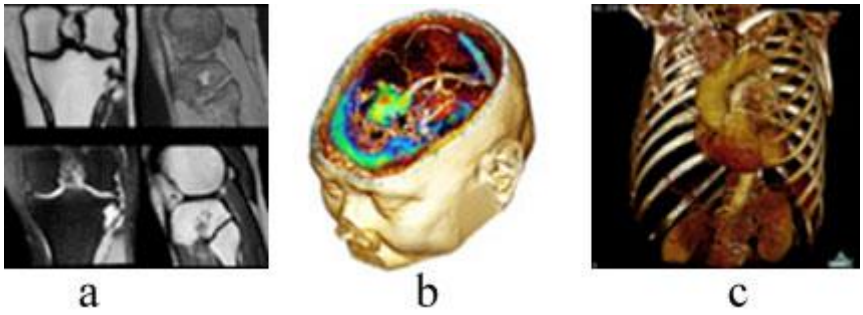

Ilustración 24: Modelos utilizados a) KNIX pequeño b) CEREBRIX Mediano c) ARTIFIX Grande

En la siguiente tabla se muestra el nombre de los modelos, la cantidad de píxeles y el número de cortes que contienen, los modelos provienen de los repositorios ofrecidos por el proyecto Osirix [**32**], estos fueron seleccionados por la cantidad de cortes involucrados en cada uno de ellos.

Nombre	Ancho en píxeles	Alto en píxeles	Nro. de Cortes
KNIX	512	512	20
CEREBRIX	512	512	174
ARTIFIX	512	512	347

Tabla 4 Modelos de Pruebas de la aplicación DCMPal

Las pruebas fueron realizadas en dos ambientes diferentes con las siguientes especificaciones:

Característica	Ambiente 1	Ambiente 2
Sistema Operativo	Windows 8	Windows 8
Procesador	i7 2.4Ghz	Core2Duo 2.6Ghz
RAM	16GB DDR3	4GB DDR3
Velocidad de HDD	7200RPM	Raid0 7200RPM
Tarjeta(s) de video	Nvidia 650m SLI	Nvidia 9800m GTS

Tabla 5 Ambientes de pruebas de la aplicación DCMPal

Los modelos antes mencionados, serán probados en los dos ambientes de trabajo, buscando analizar cuanto influyen las prestaciones del equipo en la respuesta final del sistema. Además de esto, se analizarán los tiempos involucrados de la creación de texturas, creación de cuadros y envío al cliente, tomando en cuenta los tamaños de ventana y el tamaño de los volúmenes de prueba, y como estos influyen en los tiempos de ejecución.

4.2 Creación de Texturas 3D

El despliegue de volumen requiere de texturas 3D, siendo necesario crear esta textura de un conjunto de archivos DICOM. Los tiempos asociados a esta tarea están estrechamente vinculados con el tamaño de las imágenes. Por ejemplo, es necesario tomar en cuenta los diferentes tipos de datos utilizados para la representación de la imagen dentro de un paquete DICOM, además tomar en cuenta el tiempo necesario para pasar este escalar a un valor decimal que se encuentre entre los valores 0…1 para poder tomar los tiempos de esta tarea.

La Ilustración 25 se presenta el gráfico en microsegundos (*ms*) asociado al tiempo de creación de las texturas para los tres modelos. Como puede observarse, a medida que aumenta la resolución, el tiempo tiende a aumentar. Se puede apreciar además, en el caso del modelo ARTIFIX el crecimiento de tiempo es considerable en comparación con los otros modelos. El último aspecto importante a tomar en cuenta, es la diferencia en tiempo de ejecución entre los dos equipos. Como es de esperar el ambiente 1, presenta unos tiempos mucho mejores con respecto al ambiente 2, debido principalmente a que tanto la

RAM del ambiente 1 como el procesador (Para la tarea de creación de las texturas solo se ve involucrado el CPU), tienen una velocidad mayor.

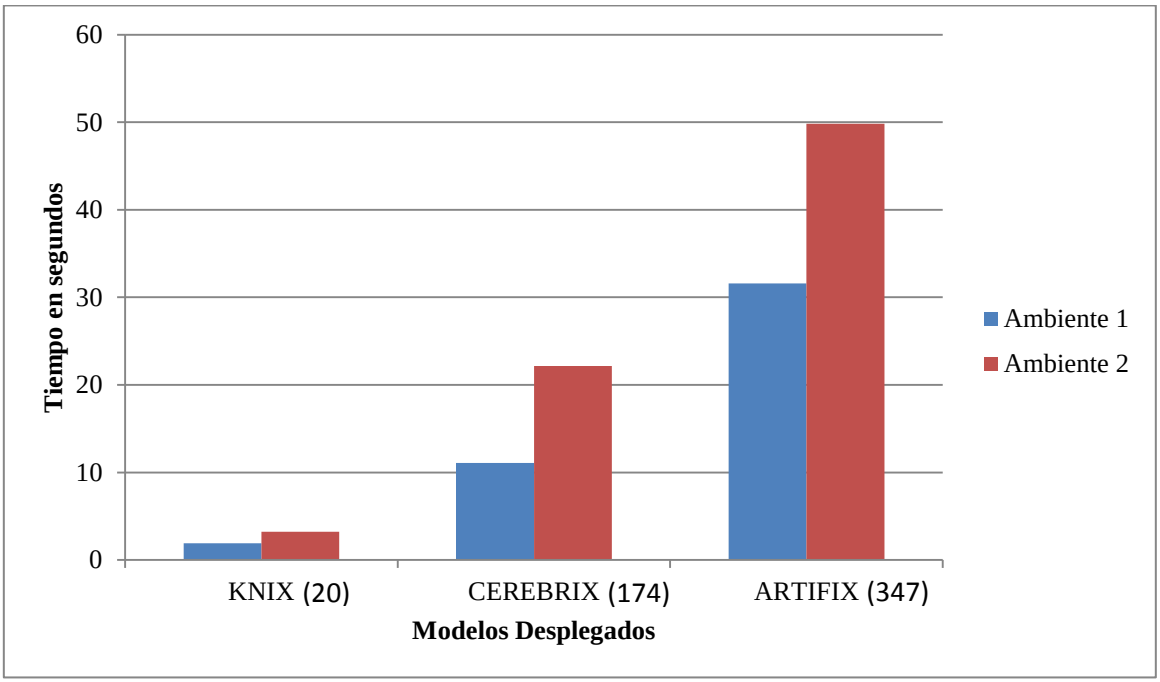

Ilustración 25: Gráfico de tiempos en segundos de creación de la textura 3D

Tomando en cuenta que las imágenes digitales provenientes de exámenes médicos [**33**] tienen tamaños muy diversos, por ejemplo un CT son archivos 512x512*px*, un MRI es más variado y puede partir de los 64x64*px* hasta archivos de más de 1024x1024*px* (en este caso no necesariamente debe ser cuadrado), donde además pueden realizarse entre 50 cortes hasta más de 2500 cortes, se puede considerar que lo modelos seleccionados pueden ser una muestra aproximada representativa a los tiempos de carga necesarios para otros tipos de volúmenes.

4.3 Desplegar un volumen en diferentes resoluciones

El despliegue del volumen en sí, se puede considerar el eje principal del sistema, este puede recibir drásticos impactos de tiempo por dos factores, el tamaño del volumen y el tamaño de la ventana. El tamaño de la ventana puede ser muy variado, pero para realizar las pruebas tomaremos en cuenta las dimensiones 400x400*px*, 600x600*px* y 800x800*px* en la Ilustración 26 Podemos observar los tiempos de creación del imagen en los dos ambientes y el modelo CEREBRIX.

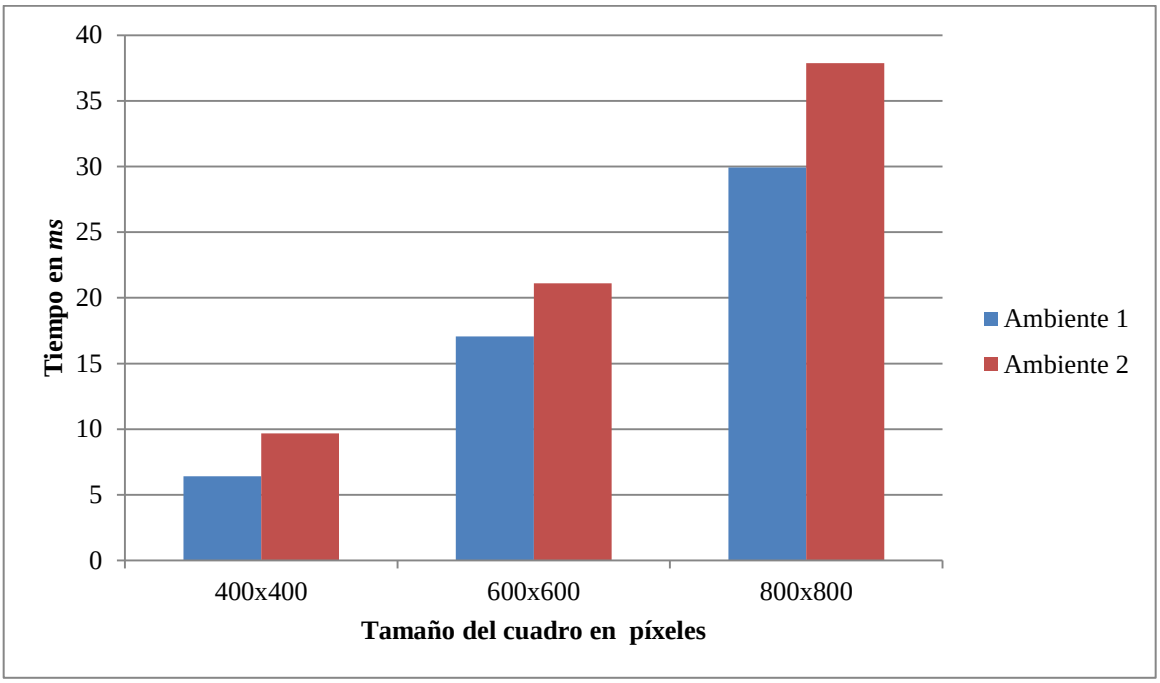

Ilustración 26: Gráfico de tiempos en ms de la creación de un cuadro en diferentes tamaños de ventana

Como se puede apreciar en la Ilustración 26 el tiempo de creación del cuadro aumenta con respecto al tamaño la ventana, ya que son más píxeles que hay que obtener por cada cuadro a realizar. A partir de ahora se asumirá un tamaño de pantalla de 600x600*px* para las pruebas futuras.

4.4 Despliegue de los diferentes volúmenes

La siguiente prueba pretende confirmar los diferentes tiempos de carga de un cuadro para los diferentes modelos. Para esto se realizó pruebas de despliegue de un cuadro en los dos ambientes con una resolución de 600x600px, para confirmar como pueden influir las capacidades del equipo en los tiempos de carga de un cuadro del volumen.

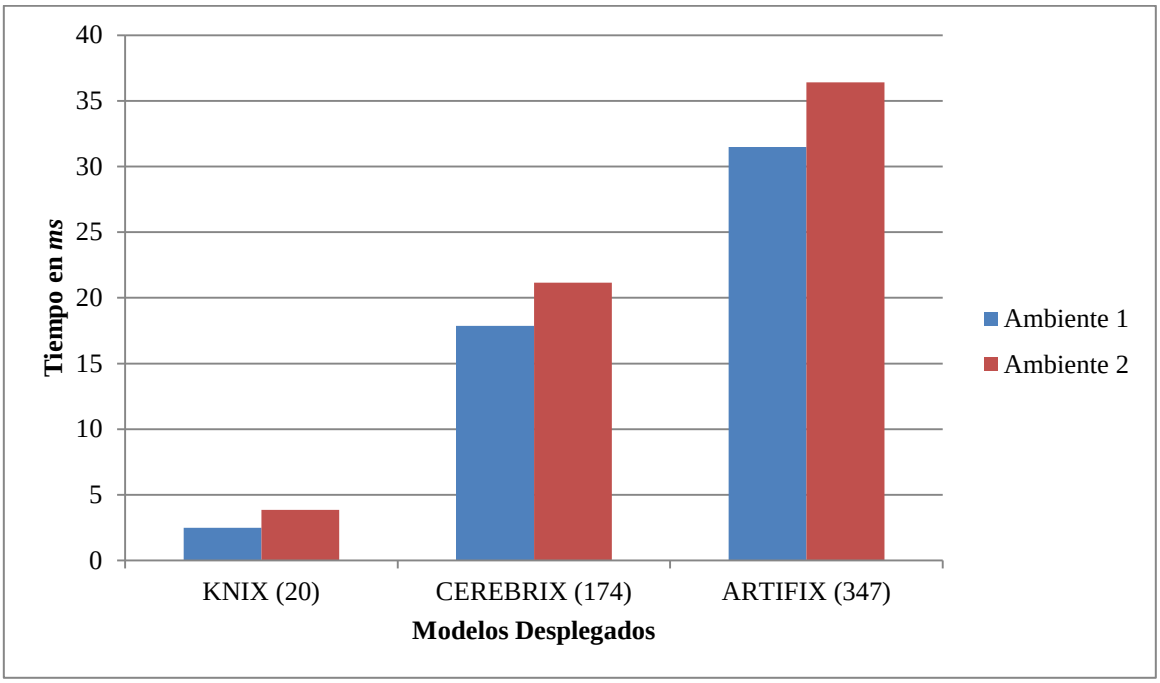

Ilustración 27: Gráfico de tiempos en ms de la creación de un cuadro de diferentes modelos en un tamaño de ventana 600x600*px*

En esta Ilustración (Ilustración 27), podemos observar los tiempos en ms para la creación del cuadro en una pantalla de 600x600*px* con los tres modelos utilizados de pruebas, puede observarse que el tiempo para la creación del cuadro del volumen más grande (ARTIFIX) es mayor al tiempo requerido en los otros modelos para crear un cuadro, debido a que este modelo utiliza un mayor número de texturas para la representación del volumen.

La aplicación hasta este momento, solo tendría la capacidad de desplegarlos resultados en el servidor, no se podría desplegar a los diferentes usuarios, ya que debemos realizar una serie de transformaciones para que sea posible enviar la imagen por el protocolo HTTP.

4.5 Transformación y envío de imagen

En el sistema DCMPal, cada vez que finaliza el proceso de generar un cuadro de imagen, pasa a la tarea de crear una representación intermedia de cuadro en una imagen, esta imagen puede ser de diferentes formatos (*jpeg*, *png*, *gif*), para poder crear un arreglo de caracteres (*String*) de representación a base64 de la imagen.

Recordando que la imagen debe ser creada en tiempo de real, es necesario verificar los tiempos requeridos para la creación de las mismas en diferentes formatos, el siguiente cuadro ilustra los tiempos de creación de los formatos *jpeg*, *png* y *gif* para el modelo CEREBRIX, estos son los resultados (Ilustración 28).

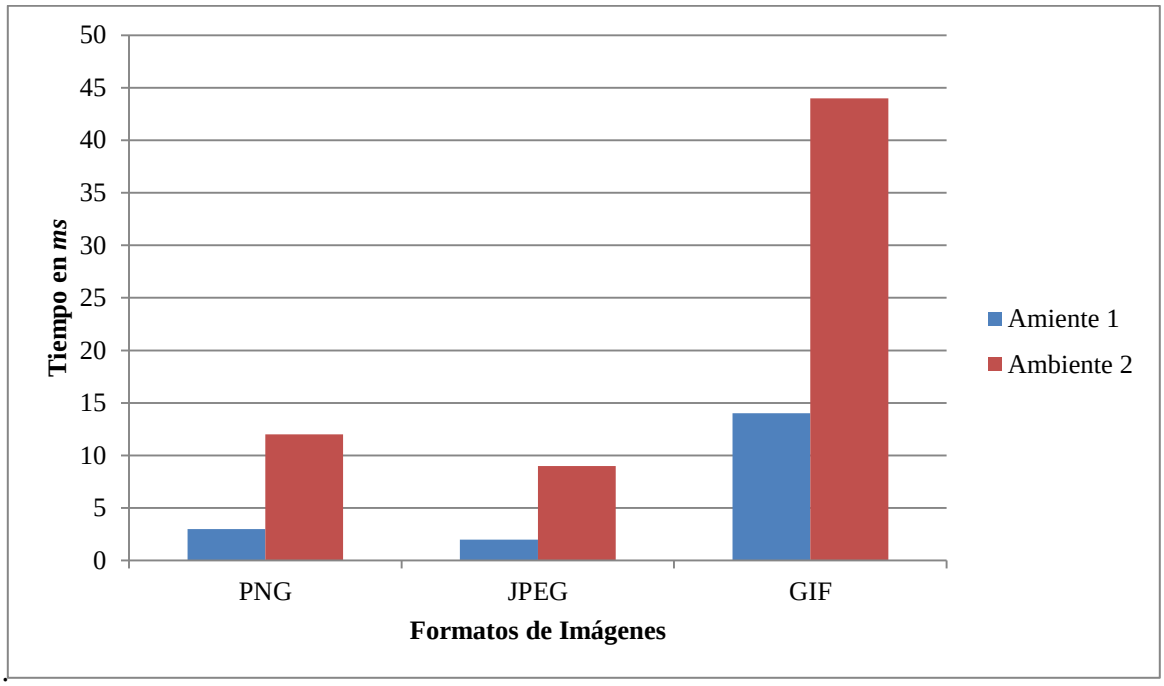

Ilustración 28: Gráfico de tiempos en ms de la creación de una imagen en diferentes formatos

Cuando observamos los resultados, podemos concluir que para esta implementación, los tiempos asociados a la creación de imagen en formato *gif* son muchos mayores en comparación a los tiempos requeridos en los formatos *jpeg* y *png*. Sin embargo, esta sola prueba no influye en el tiempo final del envío de la imagen al cliente, ya que los tamaños de las mismas influyen en la fluidez del sistema.

En la Ilustración 29 podemos observar los tiempos involucrados en la creación de la cadena de caracteres en base64 y el tiempo requerido para el envío de las imágenes a los múltiples clientes.

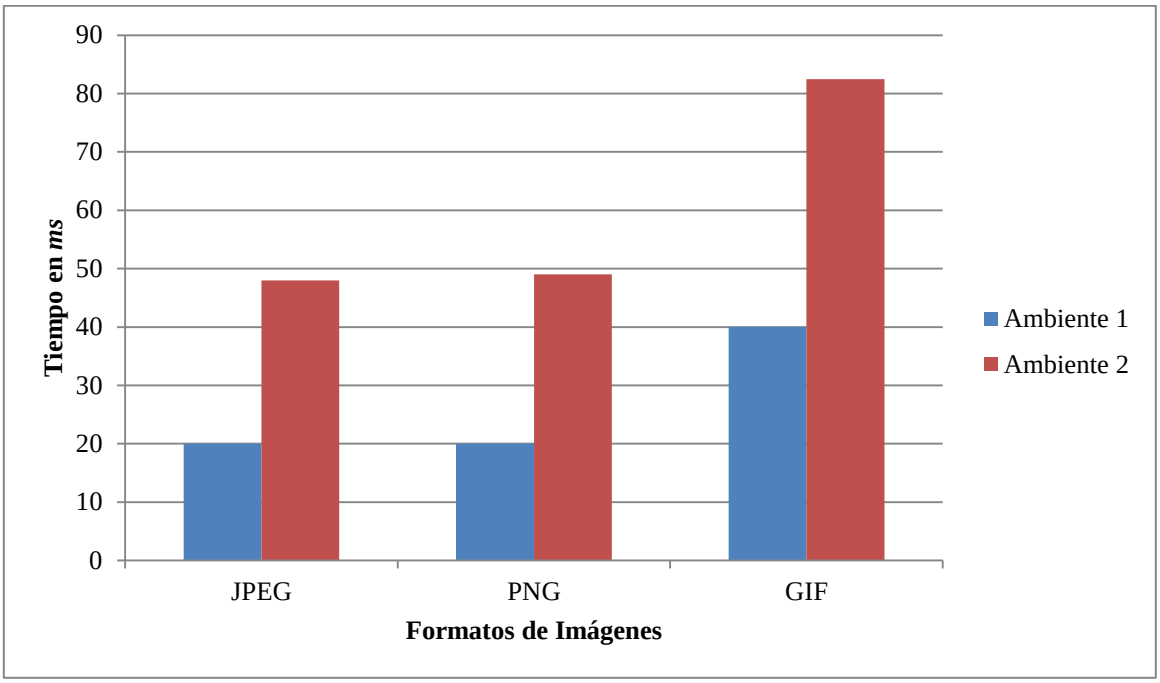

Ilustración 29: Tiempo en *ms* en enviar las imágenes a los clientes

Una vez más, podemos observar que los tiempos involucrados utilizando el formato de compresión *gif* son mayores que los tiempos necesarios para los otros formatos. Además, la diferencia de tiempo entre los otros dos formatos de compresión (*png* y *jpeg*) puede considerarse despreciable, es por ello que puede utilizarse cualquiera de los dos formatos para la solución final. Ahora bien, logrado estos cuadros, es necesario evaluar cuantos pueden ser enviados al cliente dependiendo directamente de los cuadros realizados en el servidor.

4.6 Numero de cuadros por segundo en cliente

Dentro de las configuraciones de DCMPal es posible asignar en número de cuadros máximos a realizar en un segundo (*fps*), sin embargo esto se encuentra fuertemente asociado a los tiempos necesarios para la creación de cada una de las imágenes, así como también a los tiempos requeridos para el envío de la misma al cliente, en la Ilustración 30, podemos ver como estos influyen.

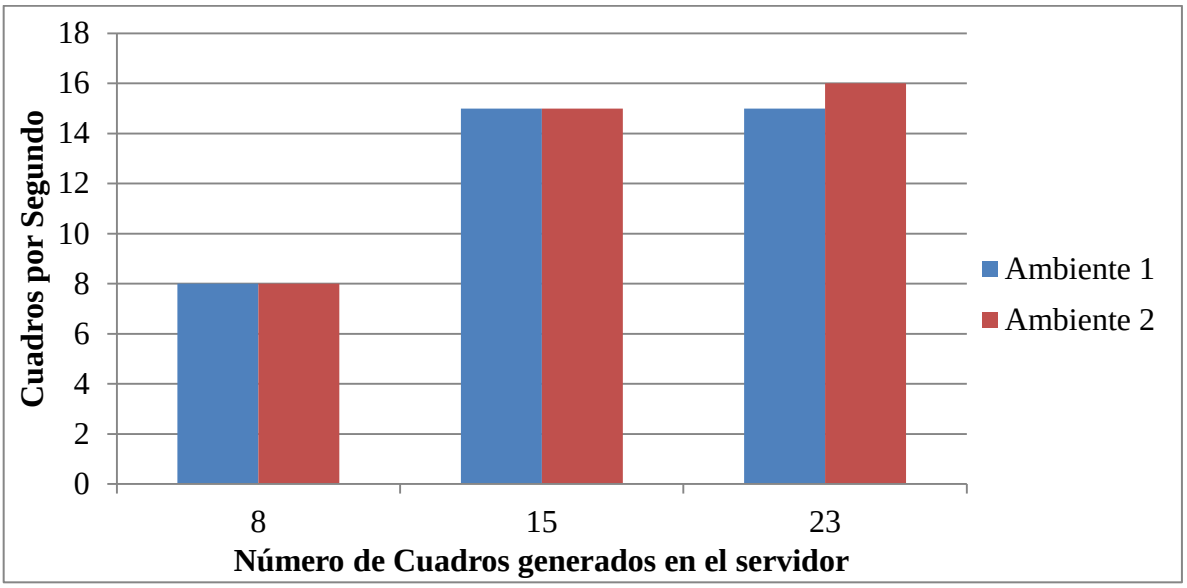

Ilustración 30 Número de Cuadros por Segundo (*fps*)

Como podemos observar, los tiempos de la creación de la imagen pueden ser controlados, sin embargo hay un tiempo mínimo requerido para el envío de la imagen del servidor al cliente, por lo que a mayor número de cuadros creados en el servidor durante un segundo, mayor es la pérdida de cuadros y mayor es el tiempo que tardan en ser mostrados en el cliente.

4.7 Tiempos de despliegue en diferentes navegadores

Esta última prueba realizada, pretende comparar los tiempos requeridos por los diferentes navegadores para desplegar descargar y desplegar el modelo CEREBRIX en base64, para lograrlo se utilizaron los diferentes modos de depuración incluidos en los navegadores Chrome [**36**], Firefox [**37**] e Internet Explorer [**38**], para obtener los tiempos necesarios para lograr la tarea.

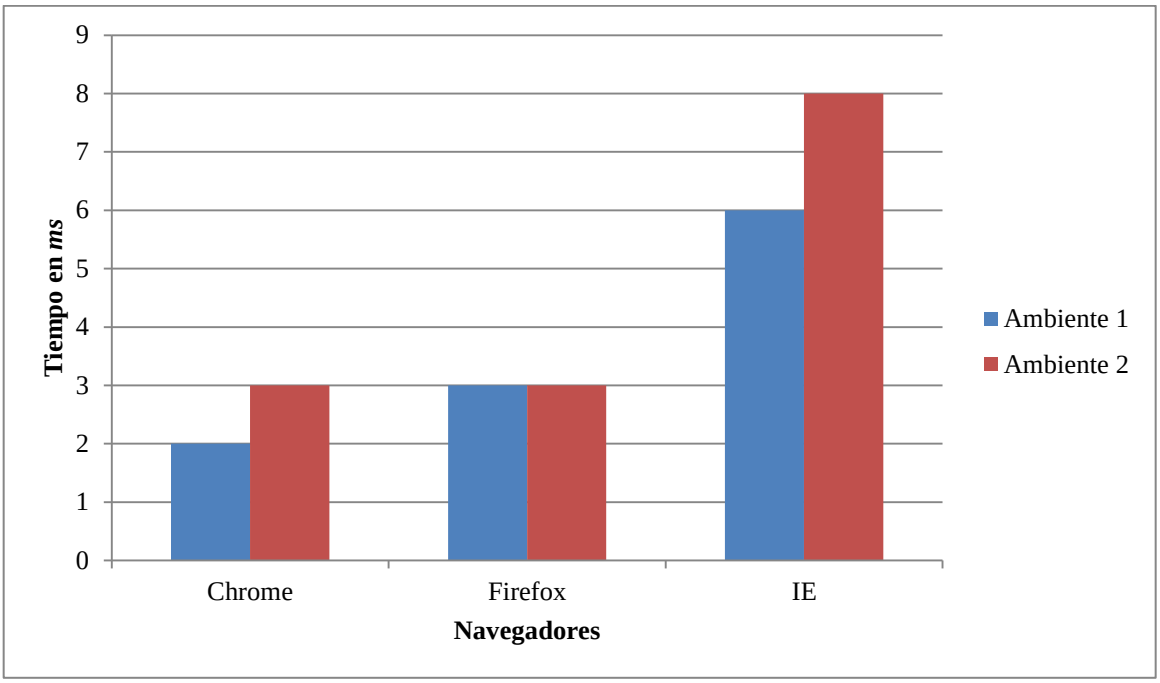

Ilustración 31 Tiempos de descarga de imágenes en diferentes navegadores

Como se puede observar en la gráfica anterior, Chrome presenta mejores prestaciones que los navegadores IE y Firefox, por lo que se puede considerar ideal a la hora de realizar el uso de esta aplicación. A continuación se presentaran imágenes finales de los tres modelos utilizados.

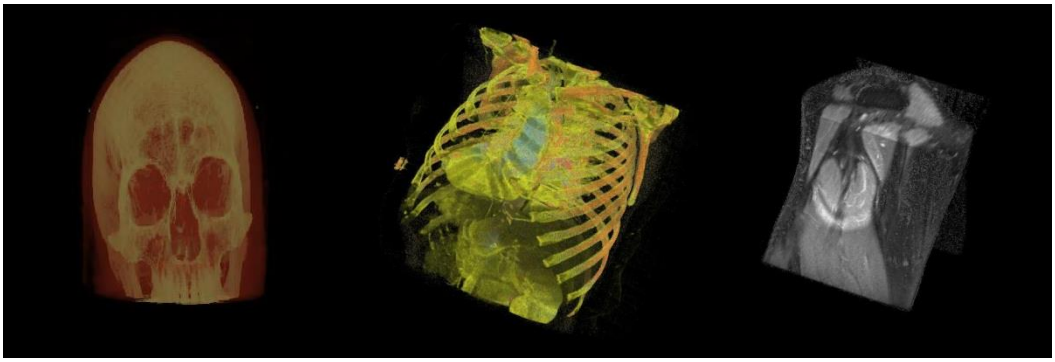

Ilustración 32 Despliegue de los modelos utilizados

En el siguiente capítulo se discutirán las recomendaciones y expondrán diferente ideas para trabajos futuros sobre la plataforma DCMPal.

Capítulo 5. Conclusiones

En este trabajo especial de grado, se muestra la implementación de un sistema colaborativo para el despliegue de imágenes médicas en la red, tomando ideas de diferentes trabajos similares presentados en [**2**] [**3**] [**4**] y [**5**]. Fue desarrollada implementado un sistema vía web y la carga del volumen llevada a cabo enteramente en el servidor para que pueda ser visualizada por clientes de todo tipo.

Gracias a las pruebas realizadas a la aplicación, se pudo seleccionar tanto el formato de imagen como el sistema de despliegue óptimo comparando en los primeros pasos del desarrollo los resultados de usar texturas 2D o Ray Casting como método de despliegue, además se buscó el tamaño ideal para el envío de la imagen. Por último, se confirmó como el uso de un servidor de mayores prestaciones, puede aumentar drásticamente el rendimiento del sistema.

Además el sistema es capaz de cargar volúmenes de gran tamaño de una manera eficiente, limitado solamente por la capacidad máxima de la tarjeta de video, las utilizadas para las pruebas ambas tenían un límite de 512^3 bytes y podían cargar modelos de tales tamaños sin ningún inconveniente, tomando en cuenta las limitaciones impuestas sobre el sistema con respecto al tipo de archivos que pueden ser desplegados. Sin embargo, a mejor rendimiento y memoria de la tarjeta este mayor es el tamaño del volumen que puede ser cargado.

DCMPal es un sistema rápido y elegante, que permite el despliegue con alta calidad de imagen en sistemas de pocas prestaciones, haciéndolo ideal para métodos de enseñanza y discusión, donde el acceso a herramientas costosas no sea un método viable. Adicionalmente DCMPal, por ser un desarrollo vía web, es altamente manipulable, por lo que la revisión de errores y sugerencias pueden ser implementadas rápidamente de manera transparente a todos los usuarios.

Otro gran beneficio de ser un sistema vía web, es la facilidad de despliegue en diferentes sistemas operativos, sin necesidad de realizar nuevas compilaciones o sistemas dedicados, disminuyendo los costos de implementación al requerir equipos con de propósito general.

Conociendo los impedimentos y cualidades antes mencionados, es posible realizar una lista de desarrollos que se pueden realizar en un futuro. El primero de ellos, consiste en aumentar la compatibilidad del sistema con los archivos DICOM, ya que es un estándar con

muchas características donde es necesario conocer hasta el más mínimo detalle para la reproducción perfecta de todo tipo de archivos.

Otro desarrollo futuro pertinente, es extender las funcionalidades propias del sistema, mientras que DCMPal, tiene la capacidad de desplegar volúmenes e imágenes, no cuenta con un módulo para la carga de archivos desde el navegador, que facilite el almacenamiento de los mismos directamente en un servidor remoto, que permita además la edición de los datos del archivo aprovechando todas las ventajas de los sistemas web, así como la creación de un sistema robusto de despliegue de los archivos 2D, que contenga un kit completo de herramientas para el usuario avanzado.

Del desarrollo de esta aplicación es solamente un pequeño acercamiento de un proyecto más grande, un visualizador web de imágenes y volúmenes avanzado, que en conjunto con un archivador de imágenes DICOM, permita la implementación de un PACS en clínicas y hospitales, donde la implementación sea simple y el código sea libre para desarrollos futuros.

Abreviaciones y siglas

CT	Computed Tomography
DICOM	Digital Imaging and Communications in Medicine
HIS	Hospital Information System
MRI	Magnetic Resonance Imaging
PACS	Picture Archiving and Communication Systems
RIS	Radiology Information System
TCP/IP	Transmission Control Protocol/Internet Protocol
CT	Computer Tomography
PET	Proton Emition tomography
SPET	Singular Photon Emition Tomography
MRI	Magnetic Resonance Imaging
ARC	American College of Radiology
NEMA	National Electrical Manufacturers Association
UID	Unique Identifiers
CLS	Common Language Specification
IL	Intermediate Language
CLR	Common Language Runtime
JIT	Just In Time
JAI	Java Advanced Imaging
LWJGL	Lightweight Java Game Library
JOGL	Java OpenGL
GUI	Graphic User Interface

Bibliografía

[1] Enrique Freer Bustamante and Johnny Chavarría Cerdas, "El desarrollo de la computación y su influencia en la medicina / Computation development and its influence in medicine," *Revista costarricense de ciencias médicas*, pp. 59-70, 1992.

[2] K Szostek and A Piórkowski, "OpenGL in Multi-User Web-Based Applications," *Innovations in Computing Sciences and Software Engineering*, pp. 379-383, 2010.

[3] Kuleesha Yadav and Lin Feng, "Biomedical Image Visualization as a Web Application," *Journal of Next Generation Information Technology*, pp. 17-27, 2012.

[4] Ilmi Yoon and Ulrich Neumann, "Web-Based Remote Rendering with IBRAC," *Computer Graphics Forum Volume 19, Issue 3*, pp. 321-330, 2001.

[5] David Koller et al., "Protected Interactive 3D Graphics Via Remote Rendering," *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 695-703, Agosto 2004.

[6] Paul Suetens, *Fundamentals of medical imaging*, 2nd ed. New York, UNited States of America: Cambridge University Press, 2009.

[7] Isaac Bankman, *Handbook of medical imaging*, 4th ed. San Diego, United States of America: Academic Press, 2000.

[8] Jesús Bellera and Rubén Medina, "Bases del Procesamiento de Imágenes Médicas," , Merida, pp. 1-12.

[9] Carmona Rhadamés, "Visualización Multi-Resolución de Volúmenes de Gran Tamaño," Universidad Central de Venezuela, Caracas, Venezuela, Tesis Doctoral 2008.

[10] Gordon Kindlmann and James W. Durkin, "Semi-automatic generation of transfer functions for direct volume rendering," in *IEEE symposium on Volume visualization (VVS '98)*, New York, 1998, pp. 79-86.

[11] William Lorensen and Harvey Cline, "Marching Cubes: A high resolution 3D surface construction algorithm," *Computer Graphics, Vol. 21, Nr. 4*, 1987.

[12] Bernardo Piquet, Silva Claudio, and Arie Kaufman, "Tetra-Cubes: An algorithm to generate 3D isosurfaces based upon tetrahedra," *Brazilian Symposium on Computer Graphics and Images*, pp. 205 - 210, 1996.

[13] Marcus Jonsson, "Volume rendering," Umea University, Suecia, Maestría de Ciencias de Computación 2005.

[14] B. Csébfalvi and E. Gröller, "Interactive Volume-Rendering Techniques for Medical Data Visualization," Vienna University of Technology, Vienna, Tesis de Doctorado Mayo 2001.

[15] M. Levoy, "Efficient ray tracing of volume data," *ACM Transations on Graphics, Vol.9, No.3*, pp. 245-261, 1990.

[16] Lee Westover, "Interactive Volume Rendering," *en Proceedings of the 1989 Chapel Hill workshop on Volume visualization*, pp. 9 - 16, 1989.

[17] G Lacroute Phillippe, "Fast Volume Rendering Using Shear-Warp Factorization of the Viewing Transformation," *Reporte Técnico CSL-TR-95-678*, 1995.

[18] Ericm LaMar and Joy Kenneth, "Multiresolution Techniques for Interactive Texture-based Volume Visualization," *en Proceedings Visualization 1999*, pp. 355-361, 1999.

[19] Akeley Kurt, "Reality Engine Graphics," *Computer Graphics, Vol. 27*, pp. 109-116, 1993.

[20] Hege Hans-Christian, Höllerer Tobias, and Stalling Detlev., "Volume Rendering - Mathematical Models and Algorithmic Aspects," *Reporte Técnico, TR 93-7*, 1993.

[21] Jusub Kim and Joseph Jaja, "Streaming model based volume ray casting implementation for Cell Broadband Engine.," *Scientific Programming*, vol. 17, no. 1-2, pp. 173-184, 2009.

[22] J Krüger and R Westermann, "Acceleration Techniques for GPU-based Volume rendering," *IEEE Visualization* , pp. 287-292, 2003.

[23] Armando. Jiménez Herrera, "Sistema PACS mínimo basado en el estándar DICOM.," *Maestría en Ciencias de la Computación*, 2006.

[24] Margaret Rouse. (2010, Junio) SearchHealthIT. [Online]. http://searchhealthit.techtarget.com/definition/picture-archiving-and-communication-system-PACS

[25] Josefina Gutiérrez Martínez et al., "Sistema PACS-CNR, una propuesta tecnológica," *Revista mexicana de ingeniería biomédica*, pp. 77-85, 2003.

[26] Mario Mustra, Kresimir Delac, and Mislav Grgic, "Overview of the DICOM Standard," *50th International Symposium ELMAR*, pp. 39-44, 2008.

[27] Oleg S. Pianykh, *Digital Imagingand Communications in Medicine (DICOM) A Practical Introduction and Survival Guide*, 3rd ed. Berlin, Alemania: Springer-Verlag, 2008.

[28] DICOM Standards Committee. (2011, Noviembre) NEMA. [Online]. ftp://medical.nema.org/medical/dicom/2011/

[29] ARC-NEMA. (2012, Dec.) DICOM estándar. [Online]. http://medical.nema.org

[30] Microsoft. (2014, Mayo) asp.net. [Online]. http://www.asp.net/mvc/tutorials/older-versions/overview/asp-net-mvc-overview

[31] GDCM team. (2014, Mayo) GDCM : Grassroots DICOM library. [Online]. http://gdcm.sourceforge.net/wiki/index.php/Main_Page

[32] (2014, Mayo) Cross Platform Make. [Online]. http://www.cmake.org/

[33] (2014, Mayo) Simplified Wrapper and Interface Generator. [Online]. http://www.swig.org/

[34] Osirix Team. (2013, Jan.) Osirix. [Online]. http://www.osirix-viewer.com/datasets/

[35] Anthony Seibert. (2014, 24) Society for Imaging Informatics in Medicine. [Online]. http://siim.org/books/archiving/chapter-2-medical-image-data-characteristics-capicity-requirements

[36] (2014, Mayo) Google. [Online]. https://www.google.com/intl/en/chrome/browser/

[37] (2014, Mayo) Firefox. [Online]. http://www.mozilla.org/en-US/firefox/new/

[38] (2014, Mayo) Microsoft. [Online]. http://windows.microsoft.com/en-us/internet-explorer/download-ie

[39] N Max, P Hanrahan, and R Crawfis, "Area and volume coherence for efficient visualization of 3D scalar functions. In Computer Graphics," *San Diego Workshop on Volume Visualization, vol. 24*, pp. 27-33, 1990.

[40] H. Munch, U. Engelmanna, A. Schroeter, and H.P. Meinzera, "Web-based distribution of radiological images from PACS to EPR," *Elsevier Science B.V*, pp. 873 – 879, 2003.

[41] Ioana M. Martin, "Adaptive Rendering of 3D Models Over Networks Using Multiple Modalities," *Technical Report RC21722, Watson Research Center*, 2000.

[42] F Evesque, S Gerlach, and R Hersch, "Building 3D anatomical scenes on the Web," *Journal of Visualization and Computer Animation, vol. 13, num. 1, p. ,* pp. 43-52, 2002.

[43] Robert Choplin, J Boebme, and C Douglas, "Picture Archiving and Communication Systems: An Overview," *RadioGraphics*, pp. 127-129, 1992.

[44] LaMar Ericm Hamann Bernd and Joy Kenneth I., "Multiresolution Techniques for Interactive Texture-based Volume Visualization," *Proc. Visualization '99*, pp. 355-361, 1999.

[45] Mark Levoy, "Display of surfaces from volume data," *IEEE Computer Graphics and Applications*, vol. 8, no. 3, pp. 29 - 37, Mayo 1989.

[46] Jusub Kim and Joseph Jaja, "Streaming model based volume ray casting implementation for Cell Broadband Engine.," *Science Program*, vol. 17, pp. 173-184, 2009.

[47] Julian Templeman and David Vitter, *Visual Studio.Net: The.Net Framework Black Book*.: Coriolis

Group Books, 2002.

[48] Thuan Thai and Hoang Lam,.*NET Framework Essentials*, 2nd ed.: O' Reilly & Associates, Inc., 2002.

[49] Dino Esposito, *Introducing ASP.NET 2.0*. Redmond, USA: Microsoft Press, 2004.

[50] V Spitzer, M J Ackerman, A L Scherzinger, and D Whitlock, "The visible human male: a technical report," *Journal of the American Medical Informatics Association : JAMIA* , vol. 3, pp. 118-30, Marzo 1996.

[51] Klaus Engel and Thomas Ertl, "Texture-based Volume Visualization for Multiple Users on the World Wide Web," in *5th Eurographics Workshop on Virtual Environments*, Erlangen, 1999, pp. 115-124.

[52] khronos group. (2013, Jan.) WebGL - OpenGL ES 2.0 for the Web. [Online]. http://www.khronos.org/webgl/

[53] Web3D Consortium. (Enero, 2013) Web3D Consortium - VRML Archives. [Online]. http://www.web3d.org/x3d/vrml/index.html

[54] Inc. Kitware. (2013, Febrero) ITK - Segmentation & Registration Toolkit. [Online]. http://www.itk.org/

[55] James Painter and Kenneth Sloan, "Antialiased Ray Tracing by Adaptive Progressive Refinement," *Computer Graphics*, vol. 23, no. 3, pp. 281-288, 1989.

[56] Julian Templeman and David Vitter, *Visual Studio.Net: The.Net Framework Black Book*, 1st ed.: Paraglyph Press, 2001, 2002.

[57] Carlos Cabal Mirabal, Evelio González Dalmau, and Henry Blanco Lores, "Hacia una red de imágenes médicas. Conceptos y bases.," *Universidad, Ciencia y tecnología Volumen 11, N° 43*, pp. 87-90, 2007.

[58] Fernando Florez and Luigi Bolaños, "¿DICOM? una aproximación a los formatos de las imágenes radiológicas.," *Revista colombiana radiol*, pp. 1748-1752, 2005.

[59] Josep Fernàndez-Bayó, Octavio Barber, and Carles Rubies, "Distributing Medical Images with Internet Technologies: A DICOM Web Server and a DICOM Java Viewer," *Radiographics*, vol. 20, pp. 581-590, 2000.

[60] Sergio Hernández Sánchez and Fernando Blázquez Román,.*NET Framework*, Departamento de Informática y Automática ed. Salamanca, Salamanca, 2005.

[61] John Congote et al., "Interactive visualization of volumetric data with WebGL in real-time," in *Proceedings of the 16th International Conference on 3D Web Technology*, 2011, pp. 137-146.

[62] Blog Actual Med. (2010, Oct.) ActualMed. [Online]. http://www.actualmed.com/blog/2010/10/20/servidor-pacs-dicom-server/

[63] Jorge Carrasco Romero, "JCRayTracer: Una Herramienta para la Síntesis de Imágenes de Alta Resolución empleando Ray Tracing (RT)," *Liscenciatura de ingeniería en sístemas computacionales*, Enero 2003.

[64] Gordon Kindlmann, "Transfer Functions in Direct Volume Rendering: Design, Interface, Interaction," *Course notes of ACM SIGGRAPH*, 2002.

[65] Joe Kniss, Gordon Kindlmann, and Charles Hansen, "Multidimensional transfer functions for interactive volume rendering," *en IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 3, pp. 270-285, Jul-Sep 2002.

[66] Joe Kniss, Gordon Kindlmann, and Charles Hansen, "Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets," *en Proceedings of the conference on Visualization '01 (VIS '01)*, pp. 255-262, 2001.

[67] J. Martí, X. Lladó C. Mata, and A. Oliver, "Mamodb: A web-based tool for training radiologists in the diagnosis of digital mammography," *en Proceedings of EDULEARN 11 Conference*, pp. 4 - 6, Julio 2011.

[68] Sagar Saladi and J Meyer, "Wavelets And Textures With Illumination For Web-based Volume Rendering," *en High-Performance Computing Symposium 2003*, 2003.

[69] James F Blinn, "Light reflection functions for simulation of clouds and dusty surfaces," *en Proceedings of the 9th annual conference on Computer graphics and interactive techniques*, pp. 21-29, 1982, Computer Graphics.

)