Metodología para la evaluacion de tecnicas de renderizado 3D en un sistema de visualización de imágenes medicas

Ana María Quintero Gomez

Universidad Tecnológica de Pereira

Facultad de Ingenierías Eléctrica, Electrónica, Física y Ciencias de la Computación
Pereira, Colombia
2010

Metodología para la evaluacion de tecnicas de renderizado 3D en un sistema de visualización de imágenes medicas

Ana María Quintero Gomez

Tesis para optar por el título de
Ingeniera Física

Director
M.Sc. Jorge Hernando Rivera Piedrahita.

Universidad Tecnológica de Pereira
Facultad de Ingenierías Eléctrica, Electrónica, Física y Ciencias de la Computación
Pereira, Colombia
2010

A

mi hermosa familia que han sido fuente de inspiración, amor y gratitud, a ellos que siempre han estado a mi lado dandome las fuerzas necesarias para culminar esta etapa de mi vida. A mis maestros y compañeros por las enseñanzas recibidas.

Índice general

Índice de figuras

Índice de tablas

Introducción

En este proyecto se busca diseñar una metodología para la evaluación de las diferentes técnicas de renderizado 3D a partir de imágenes bidimensionales, se realizará una revisión del estado del arte con el fin de determinar las herramientas computacionales que permiten desarrollar estas técnicas. Después de identificadas las herramientas más utilizadas actualmente, se desarrollara un marco experimental que permita la evaluación de las técnicas de renderizado bajo unos mismos criterios. Los criterios que se utilizarán para la escogencia de las técnicas son: La complejidad computacional y la precisión en la representación.

La principal motivación para el desarrollo de este proyecto es que no existe una técnica única para generar imágenes tridimensionales en un sistema de visualización medica, puesto que cada técnica depende de la aplicación, lo que nos lleva a buscar como generar una serie de criterios que permitan la escogencia de las técnicas.

Objetivos

General

Diseño de una metodología para la evaluación entre técnicas de renderizado 3D en un sistema .

Específicos

- Identificar las diferentes técnicas, para implementar el renderizado 3D que se está utilizando.
- Estudio de las herramientas actúales para la implementación del renderizado (librerías de computador).
- Diseño e implementación de las métricas para la evaluación de costo computacional y la precisión.
- Implementación del software para la evaluación de las técnicas.

Capítulo 1

Marco teórico y estado del arte

1.1. DICOM

DICOM (Digital Imaging and Comunications in Medicine) es un estándar propuesto y administrado por la National Electrical Manufacturers Association (NEMA). El propósito principal del estándar es garantizar la igualdad de condiciones desde el momento de la adquisición de un estudio imagenológico hasta el momento de ser desplegado en pantalla o impreso en papel radiográfico, después de un posible procesamiento de las imágenes.

Los archivos Dicom surgen con la idea de dar mayor flexibilidad a los sistemas de almacenamiento de imágenes y ademas facilitar la creación y consulta a sistemas de diagnóstico.

Estructura de un Archivo DICOM

Un archivo DICOM contiene información de la imágen, tambien contiene la información del contexto en el que se ha tomado la imagen. En el contexto de las imagenes DICOM podemos encontrarnos con datos del paciente (nombre, apellidos, edad,), tambien con datos del doctor, del centro médico donde se realiza la prueba, de la prueba médica a la que corresponde la imagen, de la máquina que ha realizado la toma (parámetros de configuración de la máquina como por ejemplo la posición del paciente en cada toma), de las imágenes tomadas(número de tomas realizadas, separación entre cada imagen, dimensión de las imágenes).[1]
Los archivo DICOM suelen ser reconocidos por su extensión *.dcm,tambien hay otras maneras de diferenciarlo y es por medio del HEADER o cabecera que consta de 128 bytes de archivos de preámbulo y 4 bytes de prefijo ”DICM”. El preámbulo puede estar en blanco o contener información sobre la aplicación principal con la que debe ser

ejecutado.

DICOM tiene una estructura organizada por etiquetas (tags) donde cada una representa un dato dististinto. Los tags están formados por dos identificadores, que son el grupo y el elemento por ejemplo (0028, 0010). El identificador de grupo nos indica a qué grupo pertenece el dato que para este caso seria el 0028 que corresponde con el grupo de datos relacionados con las características de la imagen y el identificador de elemento que para este caso es el 0010 indica qué elemento dentro del grupo al que representa corresponde con 'Rows', que nos indica el número de filas que contiene la imagen.[1]

Cada dato que compone un archivo es almacenado en una estructura llamada Data Set (conjunto de datos) donde cada dato almacenado está definido por el tag, el tipo de dato y la longitud del dato.

El conjunto formado por el tag, el tipo de dato, la longitud y el dato en si es llamado Data Element. Dependiendo del tipo de archivo DICOM y del dato almacenado la estructura del Data Element podrá variar

Tipos de visores DICOM

Existen varios tipos de visores DICOM entre ellos:

- Osirix.
- Biomedical Image suite.

Osirix

OsiriX es un programa de código abierto escrito por Antoine Rosset, M.D., que transforma un Apple Macintosh en una estación de trabajo PACS DICOM para procesar y visualizar imágenes médicas.

OsiriX es un programa de manejo de imagen dedicado a imágenes DICOM (".dcm/ extensión ".DCM") creadas por equipos médicos (MRI, CT, PET, PET-CT, ...) y microscopio confocal (formatos LSM y BioRAD-PIC). Puede leer igualmente varios formatos de archivos : TIFF (8,16, 32 bits), JPEG, PDF, AVI, MPEG y Quicktime. Es totalmente compatible con el estándar DICOM para comunicación y archivo de imagen. OsiriX es capaz de recibir imágenes transferidas mediante el protocolo de comunicación DICOM desde cualquier PACS o modalidad de imagen médica (STORE SCP - Service Class Provider, STORE SCU - Service Class User, y Query/Retrieve).

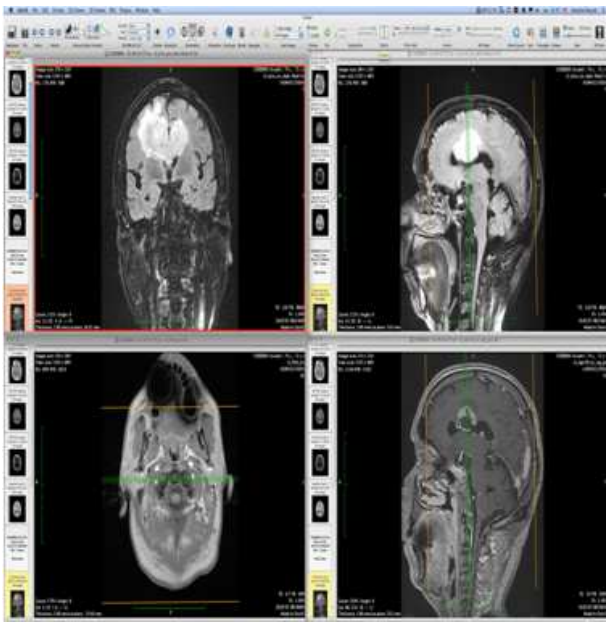


Figura 1.1: entorno del visor OsiriX, fuente Osirix Viewer

OsiriX fue diseñado especificamente para la navegación y visualización de imágenes multimodalidad y multidimensionales: Visualizador 2D, Visualizador 3D, Visualizador 4D (series 3D con dimensión temporal, por ejemplo: Cardiac-CT) y Visualizador 5D (series 3D con dimensiones temporal y funcional, por ejemplo: Cardio-PET-CT). El visualizador 3D permite todos los modos modernos de renderización: Reconstrucción multiplanar (MPR), Renderización de Superficie, Renderización de Volumen y Proyección de Intensidad Máxima (MIP). Todos estos modos aceptan datos 4D y pueden producir una fusión de imágenes entre dos series diferentes (por ejemplo: PET-CT).

Osirix es a su vez una estación PACS DICOM de visualización y renderización de imagen médica para búsqueda científica médica (radiología y imaginología nuclear), imagenología funcional, imagenología 3D, microscopio confocal y imagenología molecular.

Biomedical image suite

Tiene capacidades neuro/cardiacas, como para el análisis de imágenes abdominales. Muchos paquetes están disponibles que es sumamente extensible, y proporciona la funcionalidad para la visualización de imagen y el registro, edición de superficie, visualización cardiaca 4D, el procesamiento de imágenes de tensor de difusión, y mucho más. Puede ser integrado con otro software de procesamiento de imágenes biomédico, como FSL Y SPM. Está disponible para windows, macOx y linuX.

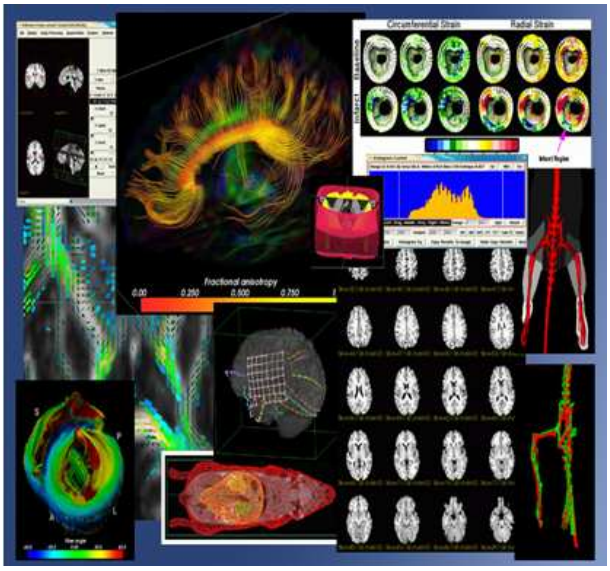


Figura 1.2: entorno de Biomedical Image Suite fuente: http://www.bioimagesuite.org/

Funcionalidad BioImage Suite

BioImage Suite cuenta con instalaciones para:[2]

- ***Pre-procesamiento*** : Maneja la imagen estándar. Además, corrige los cortes de campo, que incluye una reimplementación del método de corrección del corte ámbito de Styner, que incorpora ajustes de histogramas automatizado para la determinación del número adecuado de clases y limitaciones espaciales.

- ***Voxel Clasificación***: Método para la clasificación voxel, está disponible mediante simple histograma, un solo canal MRF y los métodos de ajuste exponencial.[2]

- ***Segmentacion de superficies deformables***: BioImage Suite tiene una fuerte y única herramienta interactiva deformable que es el editor de superficie que permite la segmentación de diferentes estructuras anatómicas. [2]

- ***Inscripción***: BioImage Suite incluye una reimplementación de los trabajos de Studholme et al. Estos métodos se han utilizado con éxito para alinear serie de datos RM, así como datos multimodales (por ejemplo, CT / PET / SPECT para RM). También incluye un complemento completo de los métodos de registro.[2]

- ***Difusión RM de Análisis de Imágenes***: BioImage Suite incluye métodos para el cálculo y visualización de medidas básicas de voxel de imágenes. [2]

1.2. VTK

Introducción

The Visualization Toolkit (VTK) es un conjunto de librerías de código libre basadas en la programación orientada a objetos. Están destinadas a la visualización y el procesamiento de imágenes, así como a la creación de objetos gráficos en 2D y 3D.

VTK esta constituido por dos subsistemas, una librería de clases compilada en C++ y varios ”interpretes”que permiten la manipulación de estas clases compiladas en lenguajes Java, Tcl/Tk y Python. Por lo tanto las aplicaciones de VTK pueden ser escritas directamente en cualquiera de estos lenguajes.

Este software es un sistema de visualización que no sólo nos permite visualizar geometría, sino que además soporta una amplia variedad de algoritmos de visualización.

Debido a su buen funcionamiento, se hacen necesarios amplios recursos de memoria en el computador para poder aprovechar en su totalidad sus funcionalidades.[3]

Funcionamiento en C++ de VTK

CMake

Para poder ejecutar un código fuente escrito en C++ con VTK es necesario el uso de la aplicación CMake.

El CMake es una herramienta multiplataforma de código libre empleada para configurar y dirigir el proceso de construcción de aplicaciones. Ficheros independientes llamados CMakeLists.txt se usan para describir el proceso de construcción y establecer las dependencias. Cuando ejecutamos CMake, se generan los ficheros necesarios, dependiendo del compilador y sistema operativo que estemos utilizando. Esto sirve para compilar VTK fácilmente, y trabajar con herramientas propias de la plataforma en la que estemos trabajando.[3]

CMake es un sistema extensible, de código abierto que gestiona el proceso de construcción de un sistema operativo y de manera independiente por el compilador. A diferencia de muchos sistemas multi-plataforma, CMake está diseñado para ser usado en conjunto con la construcción del ambiente nativo.La configuración de los archivos

colocados en cada directorio de origen (llamado CMakeLists.txt archivos) se utilizan para generar los ficheros de construcción estándar (p.ej., makefiles en Unix y proyectos / áreas de trabajo en Windows MSVC) que se utilizan en la forma habitual. CMake puede compilar el código fuente, crear bibliotecas, generar contenedores y crear ejecutables en combinaciones arbitrarias. Una característica distintiva de CMake es que genera un archivo de caché que está diseñado para ser utilizado con un editor gráfico. Por ejemplo, cuando se ejecuta CMake, localiza los archivos de inclusión, bibliotecas y archivos ejecutables, y puede crear directorios de encuentro opcional.Esta información se recoge en la memoria caché, que podrán ser modificadas por el usuario antes de la generación de los ficheros de construcción de origen.[4]

CMake está diseñado para soportar complejas jerarquías de directorios y aplicaciones que dependen de varias bibliotecas.Por ejemplo, CMake consistente en kits de herramientas múltiples (es decir, las bibliotecas), donde cada conjunto de herramientas puede contener varios directorios, y la aplicación depende de los conjuntos de herramientas más el código adicional. CMake también puede manejar situaciones en las que los ejecutables deben ser construidos con el fin de generar código que se compilan y enlazan con una aplicación final. [4]

Arquitectura de VTK

The Visualization Toolkit es un sistema orientado a objetos. La clave para utilizar VTK eficientemente es tener un buen conocimiento de los modelos de objetos fundamentales.[3]

VTK está constituido por dos modelos de objetos: Graphics Model y Visualization Model.

Graphics model

Los objetos principales que componen el graphics model son los siguientes:

- vtkActor, vtkActor2D, vtkVolume (subclases de vtkProp y vtkProp3D)
- vtkLight
- vtkCamera
- vtkProperty, vtkProperty2D
- vtkMapper, vtkMapper2D (subclases de vtkAbstractMapper)

- vtkTransform
- vtkLookupTable, vtkColorTransferFunction (subclases de vtkScalarsToColors).
- vtkRenderer
- vtkRenderWindow
- vtkRenderWindowInteractor

Al combinar estos objetos creamos una escena.

Los props representan las cosas que vemos en escena, los que son utilizados en 3D son del tipo vtkProp3D y los representados en 2D son del tipo vtkActor2D. Los props no representan directamente su geometría, sino que ésta es referida a mappers, los cuales son responsables de la representación de datos (entre otras cosas), los props también se refieren a una propiedad del objeto. [3]

La propiedad del objeto controla la apariencia del prop (color, efecto de luces, representación de la renderización, etc).

Los actores y volúmenes tienen un objeto de transformación interna (vtkTransform). Este objeto encapsula una matriz de transformación 4x4 que controla la posición, orientación y escala del prop.

Las luces (vtkLight) se usan para representar y manipular la iluminación de la escena. Solo se emplean en 3D.

La cámara (vtkCamera) controla cómo la geometría 3D es proyectada en imagen 2D durante el proceso de renderización. Tiene varios métodos para posicionar y orientar. Además controla la perspectiva de la proyección y la visión estéreo. Esto no es necesario en 2D.

El mapper (vtkMapper) junto con el lookup table (vtkLookupTable) son usados para transformar y renderizar geometría. El mapper proporciona la interfaz entre el pipeline de visualización y el graphics model. VtkLookupTable es una subclase de vtkScalarsToColors, también lo es vtkColorTransferFunction, la cual se usa para renderizar volúmenes. Las subclases de vtkScalarsToColors son responsables de mapear los valores de los datos a color.

Los renderers (vtkRenderer) y las render windows (vtkRenderWindow) se usan para dirigir la interfaz entre la máquina gráfica y el sistema de ventanas del ordenador. La render window es la ventana del ordenador donde el renderer crea el objeto. Varios renderers pueden actuar sobre una misma ventana de renderización. Además se pueden

crear múltiples render windows.[3]

Una vez creados los objetos en la ventana de renderización, existen varios métodos en VTK para interactuar con los datos de la escena. Uno de ellos es el objeto vtkRenderWindowInteractor, que es una herramienta simple para manipular la cámara, mover objetos, etc.

Muchos de estos objetos tienen subclases que especializan el comportamiento del objeto.[3]

Visualization model

La función del pipeline gráfico consiste en transformar datos gráficos en imágenes, y la del pipeline de visualización en crear esos datos gráficos a partir de la información necesaria; es decir, el pipeline de visualización es el encargado de construir la representación geométrica que será renderizada por el pipeline gráfico. VTK emplea dos tipos básicos de objetos en esta tarea:

- vtkDataObject
- vtkProcessObject

Los data objects representan datos de varios tipos. La clase vtkDataObject puede interpretarse como un conjunto genérico de datos. A los datos que tienen una estructura formal se les llama dataset (de la clase vtkDataSet).[3]

Los process objects, también llamados filtros, operan en los data objects para generar nuevos data objects. Representan los algoritmos del sistema. Process y data objects se conectan para formar los pipelines de visualización.[3]

Existen varios tipos importantes de process objects:

Los fuentes son objetos que generan datos leyendo (reader objects) o construyendo uno o más data objects (procedural source objects).[3]

Los filtros pueden tener varios data objects en la entrada, y generar uno o más data objects en la salida.[3]

Los mappers transforman los data objects en datos gráficos, los cuales son renderizados por la máquina gráfica.[3]

Es necesario llevar a cabo varios pasos para la construcción del pipeline de visualización:

Primero, la topología se construye usando variaciones de los métodos que asignan a la entrada de un filtro la salida de otro filtro.[3]

Segundo, debemos tener mecanismos para controlar la ejecución del pipeline.
Solo necesitaremos ejecutar las partes del pipeline necesarias para actualizar la salida.[3]

Tercero, el ensamblaje del pipeline requiere que sólo aquellos objetos compatibles entre sí puedan enlazarse con los métodos SetInput() y GetOutput().[3]

Finalmente, debemos decidir si conservar o no los data objects una vez que el pipeline es ejecutado. Los dataset de visualización suelen ser bastante grandes, lo que es necesario tener en cuenta para la aplicación con éxito de las herramientas de visualización.[3]

Procesamiento de la imagen

VTK tiene un extenso número de métodos para el procesamiento de imágenes y renderización de volúmenes. Los datos de imágenes 2D y 3D vienen dados por la clase vtkImageData. En un dataset de imagen los datos son ordenados en un vector regular alineado con los ejes. Mapas de bits y mapas de píxeles son ejemplos de datasets de imágenes 2D, y volúmenes (pilas de imágenes 2D) lo son de datasets de imágenes 3D.

Los process objects en un pipeline de imagen siempre tienen como entradas y salidas data objects de imagen. Debido a la naturaleza regular y simple de los datos, el pipeline de imagen tiene otros rasgos importantes. La renderización de volumen se usa para visualizar objetos 3D de la clase vtkImageData, y visores especiales de imágenes se usan para ver objetos 2D. La mayoría de los process objets en el pipeline de imagen están multiensamblados y son capaces de hacer fluir los datos por partes (para hacer un uso satisfactorio del límite de memoria). Los filtros automáticamente detectan el número disponible de procesos en el sistema y crean el mismo número de uniones durante la ejecución; igualmente, separan automáticamente los datos en partes que fluyen a través del pipeline.[3]

1.3. ITK

ITK es un conjunto de herramientas de software de código abierto para realizar el registro y la segmentación. La segmentación es el proceso de identificación y

clasificación de los datos encontrados en una representación digital de la muestra. Normalmente la representación muestra es una imagen obtenida de la instrumentación médica tales como TC o escáneres de resonancia magnética. El registro es la tarea de alinear el desarrollo de las correspondencias entre los datos.Por ejemplo, en el entorno médico, la TC puede ser alineado con una resonancia magnética a fin de combinar la información contenida en ambos.[5]

ITK es implementado en C + +. ITK es multi-plataforma, utilizando el CMake entorno de desarrollo para gestionar el proceso de configuración. Además, un proceso de ajuste automático genera interfaces entre el C + + y lenguajes de programación interpretado, como Tcl, Java y Python (usando CableSwig). Esto permite a los desarrolladores para crear software utilizando una variedad de lenguajes de programación. C + + estilo de ITK aplicación se conoce como programación genérica (es decir, utilizando el código de plantilla). Tal C + + significa que el código de plantillas es muy eficiente, y que muchos problemas de software que se descubren en tiempo de compilación, en vez de en tiempo de ejecución durante la ejecución del programa.

El Juego de herramientas de Insight es un sistema software open-source. Lo que esto significa es que la comunidad de usuarios y diseñadores de ITK tiene una gran contribución en la evolución del software. Los usuarios y diseñadores pueden hacer aportaciones significantes a ITK proporcionando informes de fallos, ajustes de fallos, tests, nuevas clases, y otras regeneraciones.

Hay dos grandes categorías de usuarios de ITK. Primero están los diseñadores de las clases, aquellos que crean las clases en C++. En segundo lugar están los usuarios, que emplean C++ y las clases existentes para construir las aplicaciones. Los diseñadores deben ser hábiles en C++, y si pretenden extenderse o modificar ITK, también deben estar familiarizados con las estructuras interiores de ITK. Los usuarios pueden o no pueden usar C++, ya que también se pueden usar otros lenguajes como Tcl o Python. Sin embargo, como usuarios debemos entender el interfaz externo a las clases de ITK y las relaciones entre ellas.[6]

1.4. MITK

La intención original de MITK es proporcionar a la comunidad un conjunto de herramientas de imágenes médicas coherente que incluye la función de segmentación, registro y visualización.El estilo de la MITK es muy similar al estilo de la VTK. Además el estilo coherente de la MITK también trae algunas características nuevas. El propósito de desarrollar MITK es enriquecer los conjuntos de herramientas disponibles y proporcionar otra opción para los investigadores y desarrolladores relacionados.

Diseño objetivos para MITK

Para el diseño de software, especialmente el diseño de complejos programas informáticos de dominio específico, un objetivo claro diseño se debe establecer por adelantado. Desde el diseño inicial, MITK siempre persigue los siguientes objetivos de diseño de alto nivel.[7]

Consistente estilo de diseño

La programación requiere una sintaxis muy detallada. En la decisión de que método de diseño MITK se debe utilizar, se insiste en utilizar la orientada a objetos método tradicional, es decir, árbol de herencia y función virtual, para formar el estilo de diseño principal, en tiempo de ejecución. El desempeño de la mejora con la optimización de los principales algoritmos. Además, se utilizan patrones de diseño para obtener una respuesta coherente.[7]

Metas limitadas

MITK sólo se centra en el dominio específico de la imagen médica, procesamiento y análisis. Por ejemplo, MITK sólo es compatible con la visualización del conjunto de datos regular, que es el tipo de datos obtenidos por el dispositivo de imágenes médicas. Esta regla puede simplificar el diseño de MITK, y mantener la MITK a una escala moderada.

Portabilidad

Para un conjunto de herramientas obtener la aplicación más amplia, la portabilidad es un factor muy importante.MITK no utiliza las características avanzadas, basadas en plantillas de C + +, por lo que el requisito del compilador es muy bajo. Actualmente MITK puede ser compilado en la mayoría de los principales compiladores de C + + y se puede ejecutar en Microsoft Windows, Unix y los sistemas operativos Linux.[7]

Algoritmo de optimización

Muchos algoritmos en la imagen médica procesamiento y análisis, especialmente en algoritmos de visualización 3D , requieren computación intensiva y rápida respuesta de la interacción del usuario.Debido a la herencia y la función virtual en el objeto del método de diseño orientado puede causar una sobrecarga adicional, la optimización de los algoritmos clave es muy importante. Mantener MITK como un conjunto de

herramientas a escala moderada hace que se tenga la oportunidad de optimizar especialmente algunos algoritmos.[7]

El marco computacional de MITK

Igual que VTK y ITK, MITK también utilizan el modelo de flujo de datos para formular el marco de cómputo.

Flujo de modelo de datos

El modelo de flujo de datos, es el centro del procesamiento de datos y el algoritmo. Es muy importante para el campo de aplicación de imágenes médicas procesamiento y análisis, en el que hay diferentes tipos algoritmos y diferentes datos a tratar.

El flujo de datos del modelo adoptado por MITK es una versión simplificada del modelo en VTK. Cada uno de los datos y el algoritmo son representados por un objeto. Un dato se extrae de una clase de datos, mientras que un algoritmo se abstrae de una clase de filtro, que recibe un objeto de datos de entrada y genera un objeto de datos de salida. Una serie de algoritmos pueden ser conectados en una tubería y forman un marco coherente de cómputo. Esto es igual al modelo aplicado en la visualización VTK, pero la diferencia aquí es que MITK no proporciona el apoyo de la topología de red, redes y retroalimentación.MITK es un conjunto de herramientas de desarrollo de software clásica.[7]

Modelo de datos

Teniendo en cuenta la característica de los datos procesados por el procesamiento de imágenes médicas y algoritmos de análisis, obtenemos los datos del modelo de MITK, volumen y acoplamiento de dos subclases concretas de datos que representa dos diferentes tipos de datos.

El volumen es un dato concreto para representar un conjunto de datos de imágenes médicas obtenidas por un dispositivo de imágenes médicas. Proporciona un resumen de la dimensión (1, 2, 3), multi-modal (TAC, RMN) y regula el conjunto de datos múltiples. Los datos internos y los atributos están expuestos al algoritmo a través de la interfaz de volumen. El volumen es uno de los objetos del kernel en MITK.

Capítulo 2

Visualización

2.1. Visualización

La visualización se puede definir como el proceso de explorar, transformar y mostrar datos en forma de imágenes para comprender y apreciar adecuadamente las características de los mismos.

Se entiende por procesamiento digital de imágenes la manipulación de las mismas a través de un computador, de modo que la entrada y la salida del proceso sean imágenes. Por otro lado, la elaboración de gráficos por computador envuelve la creación de imágenes a partir de descripciones de las mismas.

Esta área ha generado un gran interés en las dos últimas décadas. Tanto la evolución de la tecnología de computación, como el desarrollo de nuevos algoritmos para tratar señales bidimensionales y tridimensionales están permitiendo una gama de aplicaciones cada vez mayor.[3]

Algunas herramientas disponibles para la visualización de imágenes mediante la informática serán brevemente explicadas a continuación:

Matlab

MATLAB (abreviatura de MATrix LABoratory, ”laboratorio de matrices”) es un lenguaje de alto nivel que incluye herramientas de cálculo numérico y visualización de imágenes. Es un programa de Mathworks orientado para realizar todo tipo de cálculos con vectores y matrices. También presenta la posibilidad de realizar gráficos en dos y tres dimensiones.[3]

El paquete MATLAB dispone de dos herramientas adicionales que expanden sus prestaciones, a saber, Simulink (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario - GUI). Además, se pueden ampliar las capacidades de MATLAB con las cajas de herramientas (toolboxes); y las de Simulink con los paquetes de bloques (blocksets).

VTK

El modelo gráfico de VTK posee un nivel de abstracción mucho mayor que el de otras librerías de renderización de imágenes como OpenGL o PEX. Esto se traduce en una mayor sencillez a la hora de implementar aplicaciones gráficas o de visualización con VTK. Además, las aplicaciones creadas empleando VTK pueden ser escritas directamente en Tcl, Java, Pitón o C++, lo que aumenta y facilita la posibilidad de implementar aplicaciones en poco tiempo.

Por otra parte, este software es un sistema de visualización que no solo nos permite visualizar geometría, sino que además soporta una amplia variedad de algoritmos de visualización, incluyendo métodos escalares, vectoriales, tensores, de textura y volumétricos, además de otras modernas técnicas de modelado, como la reducción poligonal, el contorneado, la técnica de marching cubes, etc.[3]

Paraview

ParaView es una aplicación diseñada debido a la necesidad de visualizar archivos con gran cantidad de datos. Los objetivos del proyecto de ParaView, incluyen lo siguiente:

- Desarrollar un código abierto para la visualización multiplataforma.
- Soportar lo modelos de programación distribuida para procesar conjuntos de datos complejos.
- Crear una interface de usuario abierta, flexible e intuitiva.
- Desarrollar una arquitectura extensible basadas en estándares abiertos.

ParaView utiliza VTK como base de procesamiento de datos y motor de la renderización y visualización. Posee una interfaz escrita mediante una mezcla única de Tcl/Tk y de C++ .[3].

Técnicas de Rendering

las técnicas de rendering son basicamente dos:

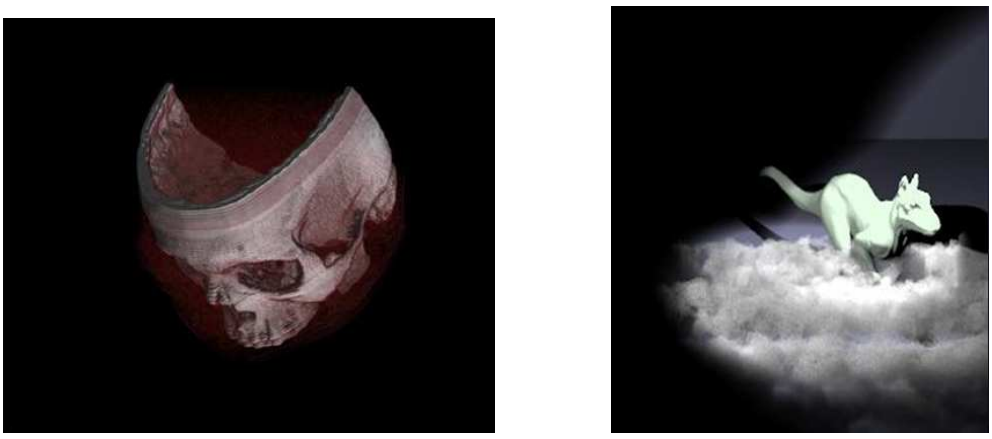
- Rendering de volumen.
- Rendering de superficie.

Rendering de Volumen

Rendering Volumétrico (o Volume Rendering, en inglés) es una técnica para hacer proyecciones bidimensionales a partir de datos tridimensionales discretos.Cada uno de estos datos representa información escalar o vectorial de un fenómeno, proceso u objeto que se quiere visualizar. Por ejemplo, densidad, presión, carga eléctrica, o cualquier otra propiedad mensurable.[8]

Las aplicaciones principales de esta técnica está en la visualización de datos médicos, geológicos, fenómenos naturales como la niebla, fuego, viento o como medio para hacer representaciones de dinámica de fluidos.[9]

La carga computacional de volume rendering depende directamente del tamaño del volumen de datos de entrada tanto en espacio como en tiempo. Un volumen de datos puede llegar a ocupar desde unos pocos megabytes como volúmenes de datos médicos hasta unos cuantos gigabytes volúmenes para la exploración de datos geológicos.



(a) (b)

Figura 2.1: Volumen rendering [9]

El rendering de volumen construye objetos con voxeles(unidad de volumen en el espacio 3D Volumetric + Pixel = Voxel.).

La idea general consiste en tomar cada una de las muestras y con esto, construir un volumen sobre el cual se hará la visualización. Por el hecho de ser un volumen de datos, la cantidad de procesamiento necesaria para representar estos datos aumenta en la medida en que las dimensiones de este volumen lo hacen.

La representación de las primitivas tridimensionales también es un proceso de proyección de datos sobre una superficie. La diferencia de complejidad reside en que las primitivas componen solamente superficies con valores de color y opacidad bien definidos que por dentro están vacías. En Volume Rendering no importa únicamente la parte externa del volumen, por el contrario, es necesario conocer los datos que componen el interior del volumen para dar el resultado.[8]

Algunas de las características es que la luz no sólo reflecta la niebla, sino que también la atraviesa, lo mismo sucede con la visualización de la piel. Tiene volumen, pero nos permite ver la calavera.Se necesita un MODELO DE ILUMINACIÓN ÓPTICO.

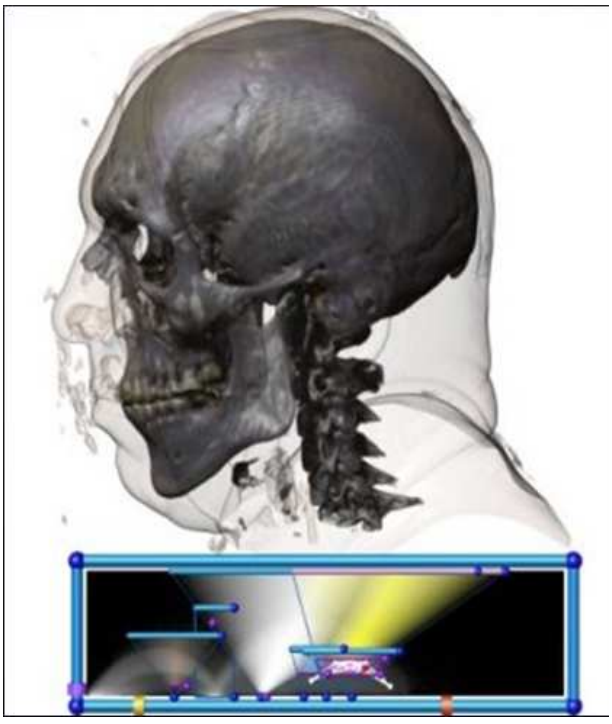


Figura 2.2: Modelo de iluminación óptico[8]

Las características volumétricas pueden ser infinitas o de mayor resolución que la

resolución de la escena (en voxels). SE NECESITA UNA FUNCIÓN DE TRANSFERENCIA.

La función de transferencia asocia un valor volumétrico a un voxel, valores de color y de transparencia para cada canal de color por separado, emisión de luz, parámetros para el modelo de Phong, índice de refracción, etc. Puede ser tan simple o compleja como sea necesario: una tabla de datos o una función polinomial.

Descripcion del proceso Físico

- La luz es absorbida por los elementos del volumen.
- La luz se dispersa en los elementos del volumen.
- La luz se emite desde los elementos del volumen.
- Una mezcla entre los factores anteriores, como puede ser el caso de la fluorescencia o la fosforescencia.

Aunque la idea general de Volume Rendering es simular este proceso, no es tan sencillo en la práctica, pues si se tomaran en consideración todos estos factores, la cantidad de cálculo involucrado sería desmesurada.[8]

Numéricamente consistiría en evaluar por completo la simulación del proceso físico, estando presente la simulación del medio circundante, reflexión, refracción entre otros. Una reducción del modelo óptico general es más que suficiente para hacer una visualización de los datos de manera apropiada.[8]

En el modelo simplificado de Volume Rendering solamente se estudia cómo, en el proceso de transporte, la luz es absorbida y emitida por los elementos del volumen. Se descarta por entero el proceso de dispersión de la luz de los volúmenes a visualizar.[8]

El rendering de volumen tiene varias técnicas:

- ***Ray Casting***: Esta técnica consiste en generar un rayo para cada uno de los pixeles de la imagen que van a mostrarse en pantalla. Los rayos son perpendiculares al plano de visión (ojo) y todos los rayos son paralelos entre si. La idea general es que muestra el volumen a intervalos regulares sobre el rayo, y sobre todas las muestras hacer una composición de colores y opacidades; luego el color final del pixel es el resultado de la composición. La composición simula el proceso de visión humano. Es una suma balanceada de los colores y opacidades de muestras en la trayectoria del rayo. La diferencia es que en la visión humana, esta suma se puede definir como una integral desde 0 a 1 (muestras infinitas), y en VR es una

suma discreta.

El proceso del ray casting se lleva a cabo mediante la interpolación trilinear, lo que hace es generar una muestra balanceada en su espacio tridimensional. Esto se hace tomando los valores de los 8 vecinos que rodean al punto en el espacio de la trayectoria del rayo. [9]

En la naturaleza, una fuente de luz emite un rayo de luz que viaja, eventualmente, a una superficie que interrumpe su avance. Uno puede pensar en este rayo como una corriente de fotones que viajan por el mismo camino. En este punto, cualquier combinación de tres cosas que pueden ocurrir con este rayo de luz: la absorción , reflexión y refracción. La superficie puede reflejar toda o parte de el rayo de luz, en una o varias direcciones. También podría absorber parte del rayo de luz, resultando en una pérdida de intensidad de la reflejada y / o de la luz refractada.

- ***Texture Mapping***:Texture mapping pinta un polígono utilizando la información de una imagen.

  Eso quiere decir que para pintar un polígono podemos elegir cualquier plano dentro del volumen que define la textura. La textura es definida por el volumen. Se utiliza la función de transferencia para obtener la textura.

  Entonces la idea del Renderizado de Volumen con Texture Mapping es representar el volumen con una cantidad finita de planos alineados. Es necesario recordar que esta aproximación del volumen también requiere un modelo de iluminación óptica o volumétrica.

Rendering de Superficie

Esta técnica es un proceso mediante el cual se determinan superficies aparentes en el interior del volumen de datos, obteniéndose una imagen representando las superficies derivadas. Trata el objeto 3D como si fuese totalmente opaco. El valor del sombreado para un vóxel está definido por la orientación original de la superficie y la localización del vóxel. El resultado se asemeja a la adquisición de una fotografía de un objeto con un foco de luz situado en un punto determinado y el valor de la sombra definido por el ángulo de la luz reflejada. Al mismo tiempo se puede modificar la localización del foco de luz y la cantidad de la luz ambiental.

Como consecuencia, la imagen 3D vista con la reconstrucción de superficie muestra sólo la parte externa del objeto, no pudiéndose analizar las estructuras internas del

objeto estudiado. Por tanto, si se representa una estructura ósea, se podra examinar su superficie, pero no el hueso trabecular si se realiza un "corte" sobre la misma. Por lo tanto, es sencillo comprobar cómo al representar únicamente los datos de la superficie del objeto, se está "desperdiciando" una gran cantidad de datos del volumen que disponemos (aquellos que representan las estructuras internas del objeto).

Existen varios algoritmos para generar objetos 3D usando esta técnica, los cuales difieren básicamente en la forma en que ellos establecen los límites del objeto. Para generar estos límites todos ellos usan alguna forma de interpolación, siendo la más común la interpolación lineal. El sistema construido cuenta con dos tipos de algoritmo: el Marching Cubes y el Dividing Cubes. [10]

- ***Marching Cubes***:Marching cubes es un algoritmo para los isosurfaces(Es una superficie que representa puntos de un valor de la constante presión, temperatura, velocidad, densidad dentro del volumen del espacio). La noción básica es que podemos definir un cubo por los valores del pixel en las ocho esquinas del cubo. Si uno o más pixeles de un cubo tienen valores menos que el isovalue (es decir,esta dentro de la superficie) definido por el usuario, y uno o más tienen valores mayores que este valor, sabemos que el cubo debe contribuir un cierto componente del isosurface. Determinando qué los bordes del cubo son intersectados por el isosurface, podemos crear los remiendos triangulares que dividen el cubo entre las regiones dentro del isosurface y las regiones afuera. Conectando los remiendos de todos los cubos en el límite del isosurface, conseguimos una representación superficial.

  Este algoritmo es de uso frecuente extraer la superficie de órganos médicos. Proporciona una manera rápida y fácil de conseguir de secciones seriales a un objeto completo 3D. [11]

- ***Dividing Cubes***: El algoritmo de dividing cubes se subdivide en los voxels, cubos más pequeños que se encuentran en la superficie del objeto y proyectados a la intensidad calculada para cada cubo en el plano de proyección, formando una sombra gradiente de la representación tridimensional del objeto. La escala voxel es elegido para hacer los cubos más pequeños igual al tamaño de pixel en la pantalla. Esta subdivisión aumenta la exactitud de la interpolación. El algoritmo calcula un índice para cada voxel mediante la comparación de los vértices del cubo con el valor de la superficie deseada, como en el algoritmo poligonal. Cada cubo está catalogado dentro de la superficie, fuera de la superficie o la intersección de la superficie.

Dividing Cubes genera puntos (nube de puntos), lo cual lo hace más eficiente a la hora de hacer el renderizado, ya que hacer render de puntos es mucho más eficiente que render de polígonos.

Una desventaja importante es que como la superficie es una nube de puntos sin conexión, hacer zoom revela .ªgujeros"dentro del objeto. Una solución es construir el conjunto conociendo el valor máximo del Zoom, de forma tal de generar puntos lo suficientemente cerca como para que no se noten los agujeros por más que realice el mismo. [10]

Capítulo 3

Diseño Experimental

3.1. Base de datos

La base de datos utilizada en ambas metodologías hacen parte de un estudio realizado a un paciente, el formato de las imágenes es un formato DICOM que contiene la información necesaria en el estudio de cada caso, las imágenes utilizadas son imágenes de resonancia magnética de la cabeza y se han usado un total de 26 imágenes para hacer cada reconstrucción.

3.2. Equipo utilizado

El objetivo principal de la investigación es mostrar como las diferentes técnicas de rendering pueden ser usadas a trávez de dos metodologías diferentes, el uso de la programación se hizo importante y para tomar los diferentes tiempos de ejecución y sacar resultados, se hicieron diferentes pruebas las cuales se realizaron con el siguiente computador:

- COMPAQ Presario F506LA, 2.0GHz, 2 GB RAM.
- Tarjeta de video NVIDIA(clase de rendimiento: 5,serie: GeForce Go 6000, frecuencia de reloj: chip: 425 MHz)

En la siguiente parte vamos a encontrar las metodologías que se usaron para hacer la reconstrucción 3D.

Para ambas metodología se tuvieron en cuenta 25 tiempos, a estos datos se les saco el respectivo promedio.

Para ambas metodologías y su diferentes técnicas primero se hizo un trabajo con la reconstrucción de esferas, para las cuales también se realizo un código que las genera y otro que le diera a dichas esferas la información y características de una archivo DICOM.

3.3. Reconstrucción 3D con matlab

En matlab se creo una interfaz de usuario llamada GUI desde donde se pudieran abrir las imágenes y hacer la reconsrucción 3D.

La interfaz gráfica de usuario, conocida también como GUI (del inglés graphical user interface) es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador.

En esta parte de la programación se trabajo con OpenGL que es un conjunto de estándares para el procesamiento de gráficos 2D y 3D de alto rendimiento en la unidad de procesamiento de gráficos (GPU) para una amplia variedad de aplicaciones. OpenGL proporciona al usuario un procesamiento rápido y de alta calidad para las previsualizaciones y la salida final transfiriendo el procesamiento de la CPU a la GPU de la tarjeta de visualización.[12]

3.4. reconstruccion 3D en visual C#

Para trabajar con visual c# se hizo necesario el uso de la libreria Activiz.

Activiz

ActiViz proporciona una interfaz de gran alcance para el kit de herramientas de visualización, es una interfaz orientada a objetos que abarca los algoritmos que transforman los datos en entornos 3D. ActiViz. NET proporciona a los desarrolladores las aplicaciones interactivas en 3D en la red. Los usuarios son capaces de integrar gráficos 3D de VTK, procesamiento de imágenes, representación volumétrica y visualizaciones en cualquier aplicación para Windows que soporte los controles integrables.[13]

Datos e Imagenes

Pruebas reconstruccion 3D matlab Rendering de Superficie y de volumen

Número de datos	Tiempo(seg)
1	21,2597
2	21,2621
3	21,3313
4	21,2951
5	21,3094
6	21,3124
7	21,3232
8	21,3233
9	21,3998
10	21,3873
11	21,3793
12	21,3966
13	21,3895
14	21,2690
15	21,3819
16	21,3687
17	21,3417
18	21,3566
19	21,3464
20	21,3524
21	21,3469
22	21,3344
23	21,3545
24	21,3333
25	21,3333
PROMEDIO	21,3395

Tabla 3.1: Datos tomados a partir de la reconstruccion 3D usando Rendering de Superficie en matlab

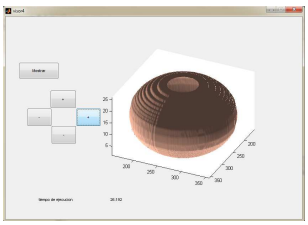


Figura 3.1: Imagenes Esferas Rendering de Superficie

Número de datos	Tiempo(seg)
1	1,4522
2	1,4619
3	1,9057
4	1,8709
5	1,8830
6	1,8875
7	1,9036
8	1,9021
9	1,9021
10	1,8735
11	1,8756
12	1,9026
13	1,9044
14	1,9024
15	1,9031
16	1,9021
17	1,8875
18	1,8756
19	1,9021
20	1,8756
21	1,8733
22	1,9023
23	1,4522
24	1,8733
25	1,8705
PROMEDIO	1,8378

Tabla 3.2: Datos tomados a partir de la reconstruccion 3D usando Rendering de Superficie en matlab

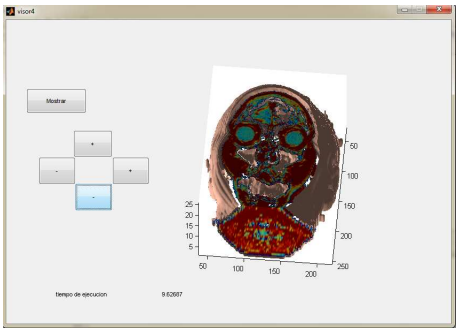


Figura 3.2: Imagenes DICOM Rendering de Superficie

Número de datos	Tiempo(seg)
1	9,2063
2	9,2095
3	9,2097
4	9,2154
5	9,2095
6	9,2063
7	9,2156
8	9,2246
9	9,2263
10	9,2300
11	9,2355
12	9,2316
13	9,2702
14	9,2325
15	9,2394
16	9,2282
17	9,2290
18	9,2097
19	9,2186
20	9,2246
21	9,2506
22	9,2118
23	9,2302
24	9,2935
25	9,2935
PROMEDIO	9,2301

Tabla 3.3: Datos tomados a partir de la reconstruccion 3D usando Rendering de Volumen en matlab

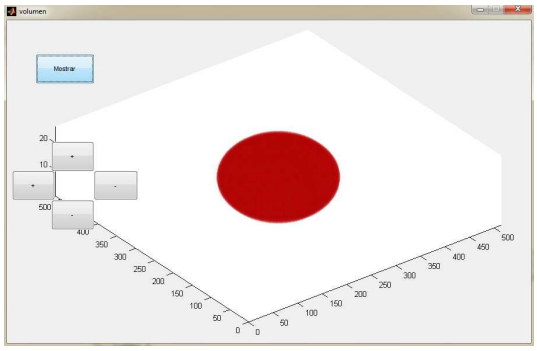
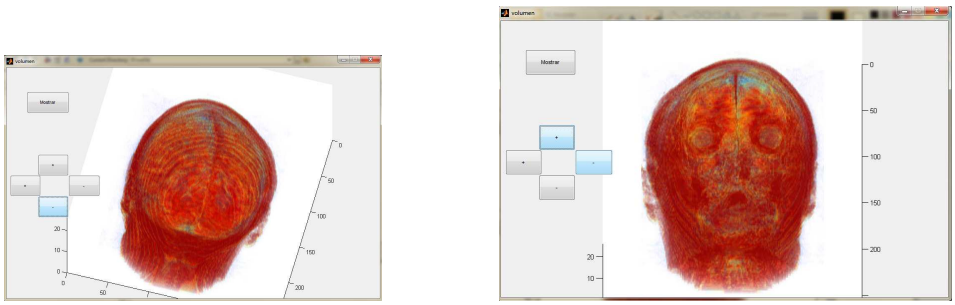


Figura 3.3: Esferas Rendering de Volumen

Número de datos	Tiempo(seg)
1	0,7015
2	0,7028
3	0,7025
4	0,7033
5	0,7055
6	0,7062
7	0,7065
8	0,7060
9	0,7034
10	0,7040
11	0,7051
12	0,7064
13	0,7080
14	0,70801
15	0,7051
16	0,7097
17	0,7121
18	0,7124
19	0,7125
20	0,7146
21	0,7151
22	0,7139
23	0,7151
24	0,7190
25	0,7192
PROMEDIO	0,7087

Tabla 3.4: Datos tomados a partir de la reconstruccion 3D usando Rendering de Volumen en matlab



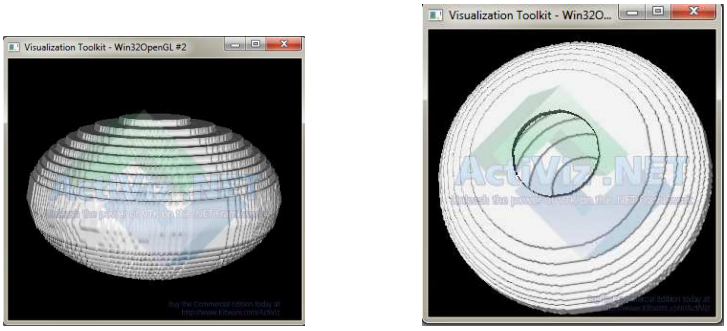
(a) (b)

Figura 3.4: Imagenes DICOM Rendering de Volumen

Pruebas reconstruccion 3D C# Rendering de Superficie y de Volumen

Número de datos	Tiempo(seg)
1	**0,9048**
2	**1,1232**
3	**1,1232**
4	**1,1388**
5	**1,2012**
6	**1,2480**
7	**1,2636**
8	**1,2480**
9	**1,2792**
10	**1,2636**
11	**1,2324**
12	**1,2480**
13	**1,1232**
14	**1,1388**
15	**1,2168**
16	**1,2480**
17	**1,2792**
18	**1,2636**
19	**1,2792**
20	**1,2792**
21	**1,2480**
22	**1,2324**
23	**1,2324**
24	**1,2792**
25	**1,2792**
PROMEDIO	**1,1812**

Tabla 3.5: Datos tomados a partir de la reconstruccion 3D usando Rendering de Superficie en C#



(a) (b)

Figura 3.5: Esferas rendering de Superficie

Número de datos	Tiempo(seg)
1	2,1684
2	2,0592
3	1,8408
4	2,1372
5	2,1684
6	2,2308
7	2,1996
8	2,2776
9	1,9812
10	1,8252
11	2,1528
12	2,1840
13	2,2776
14	2,2776
15	2,2620
16	2,1996
17	2,1840
18	2,2464
19	2,2308
20	2,2464
21	2,2776
22	2,2464
23	2,1996
24	2,2308
25	2,2776
PROMEDIO	2,1753

Tabla 3.6: Datos tomados a partir de la reconstruccion 3D usando Rendering de Superficie en C#

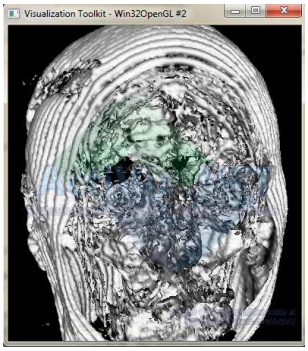


Figura 3.6: Imágenes DICOM Rendering de Superficie

Número de datos	Tiempo(seg)
1	1,8410
2	1,7319
3	1,7319
4	1,7780
5	1,7469
6	1,8719
7	1,7939
8	1,7780
9	1,7940
10	1,8099
11	1,7469
12	1,7000
13	1,7630
14	1,7939
15	1,8880
16	1,7940
17	1,7469
18	1,7939
19	1,7469
20	1,7160
21	1,7939
22	1,7630
23	1,7479
24	1,7160
25	1,7629
PROMEDIO	1,7740

Tabla 3.7: Datos tomados a partir de la reconstruccion 3D usando Rendering de Volumen en C#

Figura 3.7: Esferas Rendering de Volumen

Número de datos	Tiempo(seg)
1	0,8170
2	0,8690
3	0,8810
4	0,8710
5	0,8689
6	0,8719
7	0,8310
8	0,8410
9	0,8260
10	0,8409
11	0,8629
12	0,8739
13	0,8489
14	0,8640
15	0,8419
16	0,8380
17	0,8209
18	0,8570
19	0,8309
20	0,8299
21	0,8459
22	0,8270
23	0,8269
24	0,8269
25	0,8270
PROMEDIO	0,8456

Tabla 3.8: Datos tomados a partir de la reconstruccion 3D usando Rendering de Volumen en C#

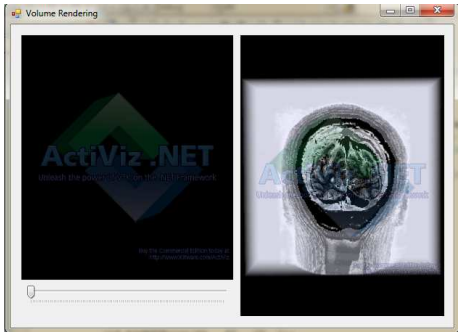


Figura 3.8: Imágenes DICOM Rendering de Volumen

Capítulo 4

Conclusiones

4.1. Conclusiones

- El aporte del trabajo realizado se encuentra en mostrar una comparación entre las técnicas de rendering para imágenes diagnosticas, y su implementación en varios lenguajes de programación, como lo fueron en este caso Matlab y C#.
- El trabajo realizado es de gran ayuda para la carrera de Ingeniera Física y el grupo de investigación en Ingeniera Física GIIF pues con estas tecnicas de rendering se puede realizar software para visualización de diferentes tipos de imágenes diagnosticas.
- Al realizar este trabajo se encontró que las técnicas de rendering de volumen son mas usadas debido a que ofrecen una mejor visualización de lo que se requiere estudiar, debido a que se pueden realizar reconstrucciones de diferentes partes del paciente en este caso de la cabeza, no solo de las capas externas.
- La ventaja del rendering de volumen en matlab en cuanto a forma de visualización es que hace mas rápido el proceso y además la imagen final es de mejor calidad. Aunque cabe resaltar que la forma del algoritmo es mas complicado.
- La ventaja del rendering de volumen en C# en cuanto a forma de visualización es que hace el proceso mas lento, pero la imagen resulta ser de muy buena calidad.
- La ventaja del rendering de superficie en matlab es que el algoritmo se hace mucho mas trabajable y la imagen no resulta tener tan buena calidad, además en cuanto a tiempos es muy lenta.
- la ventaja del rendering de superficie en C# es que la calidad de la imagen es buena pero poco precisa ya que solo muestra el exterior de la imagen y eso hace que se desperdicie espacio de estudio.

- Dependiendo de la necesidad del usuario se realiza la escogencia de la técnica de rendering.

Transmición del conocimiento

Articulo de revista para Scientia et Technica (etapa de evaluación)

Bibliografía

[1] http://www.fox-computer.es/Dig.html. [cited at p. 5, 6]

[2] X. Papademetris, M. Jackowski, N. Rajeevan, R.T. Constable, and L.H Staib. *BIOIMAGE SUITE USER'S MANUAL*. Section of Bioimaging Sciences, Dept. of Diagnostic Radiology. [cited at p. 8, 9]

[3] I. BERZAL MORENO. *DESARROLLO DE ALGORITMOS DE PROCESAMIENTO DE IMAGENES CON VTK*. Universidad politecnica de Madrid. [cited at p. 9, 10, 11, 12, 13, 17, 18]

[4] www.Cmake.org. [cited at p. 10]

[5] www.ITK.org. [cited at p. 14]

[6] D. LLANOS. *IMPLEMENTACION MULTIPLATAFORMA DE ALGORITMOS DE PROCESAMIENTO DE IMAGENES BIOMEDICAS 2D MEDIANTE LIBRERIAS ITK Y C++*. Universidad politecnica de Madrid. [cited at p. 14]

[7] Zhao Mingchang, Tian Jie 1 1, Xun Zhu, Jian Xue, Zhanglin Cheng, Hua Zhao, Zhu Xun, Xue Jian, Cheng Zhanglin, and Zhao Hua. El diseño e implementación de un c + + toolkit para la gestión integrada del procesamiento de imágenes médicas y análisis. [cited at p. 15, 16]

[8] C. DURÁN and F. MORILLO. *RENDERING VOLUMÉRICO ACELERADO BASADO EN MECANISMOS DE MANEJO DE TEXTURAS*. UNIVERSIDAD SIMÓN BOLÍVAR. [cited at p. 19, 20, 21]

[9] F. MORILLO and F. DURAN. Volumen rendering. Noviembre 2005. [cited at p. 19, 22]

[10] A. MARTIN, J. PRECIOZZI, and M. HEROZ. Descripcion de algoritmos de visualizacion 3d. Junio 2001. [cited at p. 23, 24]

[11] D. LINGRAND, A. CHARNOZ, R. GERVOISE, and K. RICHARD. Marching cubes. [cited at p. 23]

[12] www.openGL.org. [cited at p. 26]

[13] www.kitware.com. [cited at p. 26]

Capítulo 5

Anexos

5.1. Programación en Matlab Rendering de Superficie

```
function varargout = visor4(varargin)
% VISOR4 M-file for visor4.fig
%      VISOR4, by itself, creates a new VISOR4 or raises the existing
%      singleton*.
%
%      H = VISOR4 returns the handle to a new VISOR4 or the handle to
%      the existing singleton*.
%
%      VISOR4('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in VISOR4.M with the given input arguments.
%
%      VISOR4('Property','Value',...) creates a new VISOR4 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before visor4_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to visor4_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help visor4
```

```
% Last Modified by GUIDE v2.5 12-Jul-2010 10:14:30

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @visor4_OpeningFcn, ...
                   'gui_OutputFcn',  @visor4_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before visor4 is made visible.
function visor4_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to visor4 (see VARARGIN)

% Choose default command line output for visor4
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes visor4 wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = visor4_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

clc

 load DICOM
%load mas
[M,N,K]=size(D);
for i=1:K
    D(:,:,i)= mejora(D(:,:,i));
end

axis image
x = xlim;
y = ylim;
cm = brighten(jet(length(map)),-.5);

mage_num)
axis ij
xlim(x)
ylim(y)

axes(handles.axes4);
for i=1:1

tic
```

```
Ds = smooth3(D);
hiso = patch(isosurface(Ds,5),...
    'FaceColor',[1,.75,.65],...
    'EdgeColor','none');
isonormals(Ds,hiso)
hcap = patch(isocaps(D,5),...
   'FaceColor','interp',...
  'EdgeColor','none');

set(gcf,'Renderer','OpenGL');

tiempo=toc
end
set(handles.text2,'String',tiempo);
view(30,30)

axis tight
daspect([1,1,.2])
lightangle(90,0);
lighting phong
set(hcap,'AmbientStrength',.4)
set(hiso,'SpecularColorReflectance',0,'SpecularExponent',50)
camlight('left')
camlight(50,70)


guidata(hObject, handles);




% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


handles.u=handles.u+5;
```

```
axes(handles.axes4);

view(handles.u,handles.l)
guidata(hObject, handles);
% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

handles.u=handles.u-5;



axes(handles.axes4);

view(handles.u,handles.l)
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function figure1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
handles.a=30;
handles.u=30;
handles.l=30;

guidata(hObject, handles);


% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

handles.l=handles.l+5;

axes(handles.axes4);
```

```
view(handles.u,handles.l)
guidata(hObject, handles);

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.l=handles.l-5;
%handles.a+10;


axes(handles.axes4);

view(handles.u,handles.l)
guidata(hObject, handles);



% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load mas %imagenes para las Esferas
load DICOM %imagenes DICOM
tic
D = squeeze(D);
figure
 h = vol3d('cdata',D,'texture','2D');
 view(3);
 vol3d(h);


 axis tight;  daspect([1 1 .2 ])
 alphamap('rampup');
 alphamap(.06 .* alphamap);
 toc
guidata(hObject, handles);
```

5.2. Programación en Matlab Rendering de Volumen

```
function varargout = volumen(varargin)
% VOLUMEN M-file for volumen.fig
%      VOLUMEN, by itself, creates a new VOLUMEN or raises the existing
%      singleton*.
%
%      H = VOLUMEN returns the handle to a new VOLUMEN or the handle to
%      the existing singleton*.
%
%      VOLUMEN('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in VOLUMEN.M with the given input arguments.
%
%      VOLUMEN('Property','Value',...) creates a new VOLUMEN or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before volumen_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to volumen_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help volumen

% Last Modified by GUIDE v2.5 14-Jul-2010 11:11:37

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @volumen_OpeningFcn, ...
                   'gui_OutputFcn',  @volumen_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
```

```
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before volumen is made visible.
function volumen_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to volumen (see VARARGIN)

% Choose default command line output for volumen
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes volumen wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = volumen_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
load mas% imagenes de las eferas
```

```
load DICOM%imagenes DICOM
for i=1:1
tic
D = squeeze(D);
 h = vol3d('cdata',D,'texture','2D');
 view(3);
 vol3d(h);
 axis tight;  daspect([1 1 .2 ])
 alphamap('rampup');
 alphamap(.1 .* alphamap);
tiempo= toc
end

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

handles.l=handles.l-5;




view(handles.u,handles.l)
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function figure1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
handles.a=30;
handles.u=-30;
handles.l=30;
guidata(hObject, handles);

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.l=handles.l+5;




view(handles.u,handles.l)
guidata(hObject, handles);

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.u=handles.u+5;




view(handles.u,handles.l)
guidata(hObject, handles);
% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
handles.u=handles.u-5;




view(handles.u,handles.l)
guidata(hObject, handles);
```

5.3. Programación en C# Rendering de Superficie

```
using System;
```

```
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Kitware.VTK;
using System.IO; // para la funcion de crear archivos .txt

namespace pruebactiviz
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        vtkImageViewer viewerdicom = new vtkImageViewer();
        List<String> datos = new List<string>(0);

        private void abrirToolStripMenuItem_Click(object sender, EventArgs e)
        {
            //imageviewer
                vtkDICOMImageReader readerdicom = new vtkDICOMImageReader();


                viewerdicom.SetInput(readerdicom.GetOutput());

                readerdicom.SetDirectoryName("/act");//nombre de la carpeta donde est
                readerdicom.Update();

                int voldata_images = viewerdicom.GetWholeZMax();
                hScrollBar1.Maximum = voldata_images;

            //el siguiente for es para ver los cortes anatómicos
                for (int i = 0; i <=voldata_images; i++)
                {
                    viewerdicom.SetZSlice(i);
                    viewerdicom.SetColorWindow(400);
                    viewerdicom.SetColorLevel(200);
```

```
                viewerdicom.Render();
            }

    //surfacerendering

            progressBar1.Maximum = 1;//como se va a cerrar la venta esta barra pro
                                     //debe ser igual al numero de iteraciones del

            for (int i = 0; i < 1; i++)// numero de veces q se repetirá el renderi
            {
                TimeSpan tiempo = new TimeSpan();

                DateTime timeStart1 = DateTime.Now;


                vtkRenderer renderer = vtkRenderer.New();//renderWindowControl1.Re
                vtkRenderWindow renwin = vtkRenderWindow.New();//renderWindowContr
                renwin.AddRenderer(renderer);

                vtkRenderWindowInteractor iren = vtkRenderWindowInteractor.New();
                iren.SetRenderWindow(renwin);

                vtkContourFilter skinextractor = vtkContourFilter.New();
                skinextractor.SetInputConnection(readerdicom.GetOutputPort());
                skinextractor.SetValue(0, 90);

                vtkPolyDataNormals skinnormals = vtkPolyDataNormals.New();
                skinnormals.SetInputConnection(skinextractor.GetOutputPort());
                skinnormals.SetFeatureAngle(90.0);

                vtkPolyDataMapper skinmapper = vtkPolyDataMapper.New();
                skinmapper.SetInputConnection(skinnormals.GetOutputPort());
                skinmapper.ScalarVisibilityOff();

                vtkActor skin = vtkActor.New();
                skin.SetMapper(skinmapper);

                vtkCamera camara = renderer.GetActiveCamera();
                camara.SetViewUp(0, 0, -1);
                camara.SetPosition(1, 1, 1);
                camara.SetFocalPoint(0, 0, 0);
```

```
            camara.ComputeViewPlaneNormal();

            renderer.AddActor(skin);
            renderer.SetActiveCamera(camara);
            renderer.ResetCamera();
            camara.Dolly(1.5);

            DateTime timeEnd1 = DateTime.Now;
            tiempo = timeEnd1 - timeStart1; //calcula el tiempo q tarda en ha
            datos.Add(tiempo.ToString());




            renderer.ResetCameraClippingRange();
            iren.Initialize();
            iren.Start();



            skinextractor.Dispose();
            skinnormals.Dispose();
            skinmapper.Dispose();
            skin.Dispose();

            camara.Dispose();
            iren.Dispose();
            renwin.Dispose();
            renderer.Dispose();
            progressBar1.Value = i+1;
        }
        readerdicom.Dispose();



    //guardar los datos en un txt
        string fileName = "datos.txt";
        StreamWriter writer = File.CreateText(fileName);

        foreach (string dato in datos)
        {
```

```
                writer.WriteLine(dato);
            }
            datos.Clear();
            writer.Close();
        }

        private void hScrollBar1_Scroll(object sender, ScrollEventArgs e)
        {
            viewerdicom.SetZSlice(hScrollBar1.Value);
            viewerdicom.Render();
        }

        private void hScrollBar2_Scroll(object sender, ScrollEventArgs e)
        {
            viewerdicom.SetColorWindow(hScrollBar2.Value);
            viewerdicom.Render();
        }

        private void hScrollBar3_Scroll(object sender, ScrollEventArgs e)
        {
            viewerdicom.SetColorLevel(hScrollBar3.Value);
            viewerdicom.Render();
        }
    }
}
```

5.4. Programación en C# Rendering de Volumen

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using Kitware.VTK;
namespace VolumeRendering
{
    /// <summary>
    /// Application to load and display a volume
    /// </summary>
```

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }

    private bool MouseDown1 = false;
    private vtkRenderWindowInteractor Interactor = null;
    private vtkRenderWindow RenderWindow = null;
    private vtkRenderer Renderer = null;
    private vtkImageActor ImageActor = null;
    private vtkImageClip Clip = null;
    string fileName = "";
    private vtkImageViewer viewerdicom = new vtkImageViewer();
    vtkRenderer renderer;
    vtkFixedPointVolumeRayCastMapper texMapper;

    /// <summary>
    /// Tell the application when the mouse is being dragged
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    void iren_LeftButtonReleaseEvt(vtkObject sender, vtkObjectEventArgs e)
    {
        this.MouseDown1 = false;
    }
    /// <summary>
    /// Tell the application when the mouse is being dragged
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    void iren_LeftButtonPressEvt(vtkObject sender, vtkObjectEventArgs e)
    {
        this.MouseDown1 = true;
    }

    /// <summary>
    /// Display the render window with the 3D Volume in it
    /// </summary>
    /// <param name="sender"></param>
```

```
/// <param name="e"></param>
private void renderWindowControl2_Load(object sender, EventArgs e)
{
    //agregado
    //imageviewer
    vtkDICOMImageReader readerdicom = new vtkDICOMImageReader();


    viewerdicom.SetInput(readerdicom.GetOutput());

    readerdicom.SetDirectoryName("/act");
    readerdicom.Update();

    int voldata_images = viewerdicom.GetWholeZMax();
    //agregado

    //Create all the objects for the pipeline
    renderer = renderWindowControl2.RenderWindow.GetRenderers().GetFirstRender
    //vtkXMLImageDataReader reader = vtkXMLImageDataReader.New();
    texMapper = vtkFixedPointVolumeRayCastMapper.New();
    vtkVolume vol = vtkVolume.New();
    vtkColorTransferFunction ctf = vtkColorTransferFunction.New();
    vtkPiecewiseFunction spwf = vtkPiecewiseFunction.New();
    vtkPiecewiseFunction gpwf = vtkPiecewiseFunction.New();

    //Read in the file
    //reader.SetFileName(fileName);
    //reader.Update();

    //Go through the visulizatin pipeline
    texMapper.SetInputConnection(readerdicom.GetOutputPort());

    //Set the color curve for the volume
    ctf.AddHSVPoint(0, .67, .07, 1);
    ctf.AddHSVPoint(94, .67, .07, 1);
    ctf.AddHSVPoint(255, 0, 0, 0);
    ctf.AddHSVPoint(139, .28, .047, 1);
    ctf.AddHSVPoint(160, .38, .013, 1);

    //Set the opacity curve for the volume
    spwf.AddPoint(255, 0);
```

```
            spwf.AddPoint(151, .1);
            spwf.AddPoint(255, 1);

            //Set the gradient curve for the volume
            gpwf.AddPoint(0, .2);
            gpwf.AddPoint(10, .2);
            gpwf.AddPoint(25, 1);

            vol.GetProperty().SetColor(ctf);
            vol.GetProperty().SetScalarOpacity(spwf);
            vol.GetProperty().SetGradientOpacity(gpwf);

            vol.SetMapper(texMapper);

            //Go through the Graphics Pipeline
            renderer.AddVolume(vol);

        }


        /// <summary>
        /// Display the render window with the slice in it
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        //private void renderWindowControl1_Load(object sender, EventArgs e)
        //{
        //    //agregado
        //    //imageviewer
        //    vtkDICOMImageReader readerdicom = new vtkDICOMImageReader();


        //    viewerdicom.SetInput(readerdicom.GetOutput());

        //    readerdicom.SetDirectoryName("/A");
        //    readerdicom.Update();

        //    int voldata_images = viewerdicom.GetWholeZMax();
        //    //agregado

        //    //Get the name of the Unsigned Char volume that you want to load
```

```
//    fileName = "../../../head.vti";

//    //Create all the objects for the pipeline
//    //vtkXMLImageDataReader reader = vtkXMLImageDataReader.New();
//    vtkImageActor iactor = vtkImageActor.New();
//    vtkImageClip clip = vtkImageClip.New();
//    vtkContourFilter contour = vtkContourFilter.New();
//    vtkPolyDataMapper mapper = vtkPolyDataMapper.New();
//    vtkActor actor = vtkActor.New();
//    vtkInteractorStyleImage style = vtkInteractorStyleImage.New();

//    vtkRenderer renderer = renderWindowControl1.RenderWindow.GetRenderers().

//    //Read the Image
//    //reader.SetFileName(fileName);

//    //Go through the visulization pipeline
//    iactor.SetInput(readerdicom.GetOutput());
//    renderer.AddActor(iactor);
//    readerdicom.Update();
//    int[] extent = readerdicom.GetOutput().GetWholeExtent();
//    iactor.SetDisplayExtent(extent[0], extent[1], extent[2], extent[3],
//                (extent[4] + extent[5]) / 2,
//                (extent[4] + extent[5]) / 2);

//    clip.SetInputConnection(readerdicom.GetOutputPort());
//    clip.SetOutputWholeExtent(extent[0], extent[1], extent[2], extent[3],
//                (extent[4] + extent[5]) / 2,
//                (extent[4] + extent[5]) / 2);

//    contour.SetInputConnection(clip.GetOutputPort());
//    contour.SetValue(0, 100);

//    mapper.SetInputConnection(contour.GetOutputPort());
//    mapper.SetScalarVisibility(1);

//    //Go through the graphics pipeline
//    actor.SetMapper(mapper);
//    actor.GetProperty().SetColor(0, 1, 0);

//    renderer.AddActor(actor);
```

```
//    //Give a new style to the interactor
//    vtkRenderWindowInteractor iren = renderWindowControl1.RenderWindow.GetI
//    iren.SetInteractorStyle(style);

//    //Add new events to the interactor style
//    style.LeftButtonPressEvt += new vtkObject.vtkObjectEventHandler(iren_Le
//    style.LeftButtonReleaseEvt += new vtkObject.vtkObjectEventHandler(iren_
//    style.MouseMoveEvt += new vtkObject.vtkObjectEventHandler(iren_MouseMov

//    //Update global variables
//    this.trackBar1.Maximum = extent[5];
//    this.trackBar1.Minimum = extent[4];
//    this.Interactor = iren;
//    this.RenderWindow = renderWindowControl1.RenderWindow;
//    this.Renderer = renderer;
//    this.Clip = clip;
//    this.ImageActor = iactor;
//}

/// <summary>
/// Move the slice when the trackbar is moved
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void trackBar1_Scroll(object sender, EventArgs e)
{
    if (ImageActor != null)
    {
        int[] lastPos = this.Interactor.GetLastEventPosition();
        int[] size = this.RenderWindow.GetSize();
        int[] dim = this.ImageActor.GetInput().GetDimensions();

        int newSlice = (int)(trackBar1.Value);

        if (newSlice >= 0 && newSlice < dim[2])
        {
            this.Clip.SetOutputWholeExtent(0, dim[0] - 1, 0, dim[1] - 1, newS
            this.ImageActor.SetDisplayExtent(0, dim[0] - 1, 0, dim[1] - 1, ne
            this.Renderer.ResetCameraClippingRange();
            this.RenderWindow.Render();
```

```
        }
    }
}

/// <summary>
/// Move the slice and the trackbar when the mouse is dragged on the render wi
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
void iren_MouseMoveEvt(vtkObject sender, vtkObjectEventArgs e)
{
    if (this.MouseDown1 && this.ImageActor != null)
    {
        int[] lastPos = this.Interactor.GetLastEventPosition();
        int[] size = this.RenderWindow.GetSize();
        int[] dim = this.ImageActor.GetInput().GetDimensions();

        int newSlice = (int)((double)(dim[2] - 1.0) * (double)(lastPos[1]) / (

        if (newSlice >= 0 && newSlice < dim[2])
        {
            this.trackBar1.Value = newSlice;
            this.Clip.SetOutputWholeExtent(0, dim[0] - 1, 0, dim[1] - 1, newSl
            this.ImageActor.SetDisplayExtent(0, dim[0] - 1, 0, dim[1] - 1, new
            this.Renderer.ResetCameraClippingRange();
            this.RenderWindow.Render();
        }
    }
}

/// <summary>
/// Clean Up any global variables that might still be around
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void Form1_Closed(object sender, FormClosedEventArgs e)
{
    if (this.Interactor != null)
    {
        this.Interactor.Dispose();
    }
```

```
            if (this.ImageActor != null)
            {
                this.ImageActor.Dispose();
            }
            if (this.Clip != null)
            {
                this.Clip.Dispose();
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            this.Text = renderer.GetLastRenderTimeInSeconds().ToString();

            //this.Text = texMapper.GetTimeToDraw().ToString();
        }
    }
}
```