TRABAJO FIN DE GRADO

INGENIERÍA EN INFORMÁTICA

3DCurator

Un visor 3D de TCs de esculturas

Autor
Francisco Javier Bolívar Lupiáñez

Director
Francisco Javier Melero Rus

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, 10 de mayo de 2016

3DCurator

Un visor 3D de TCs de esculturas

Autor
Francisco Javier Bolívar Lupiáñez

Director
Francisco Javier Melero Rus

3DCurator: Un visor 3D de TCs de esculturas

Francisco Javier Bolívar Lupiáñez

Palabras clave: palabra_clave1, palabra_clave2, palabra_clave3, ...

Resumen

Poner aquí el resumen.

Yo, **Francisco Javier Bolívar Lupiáñez**, alumno de la titulación Grado en Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación**, con DNI 75926571Y, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Francisco Javier Bolívar Lupiáñez

Granada a 10 de mayo de 2016.

D. **Francisco Javier Melero Rus**, Profesor del Área de XXXX del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***3DCurator, Un visor 3D de TCs de esculturas***, ha sido realizado bajo su supervisión por **Francisco Javier Bolívar Lupiáñez**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 10 de mayo de 2016.

El director:

Francisco Javier Melero Rus

Agradecimientos

Poner aquí agradecimientos...

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

El objetivo de este proyecto es construir un software con el que poder visualizar e interactuar con los datos DICOM obtenidos al someter a una escultura a una Tomografía Axial Computerizada (TAC o TC).

Para ello se hará uso de VTK, que proporciona una serie de librerías en C++ para facilitar operaciones sobre datos DICOM, y de Qt, para la Interfaz Gráfica de Usuario (GUI).

Antes de empezar con el proyecto en sí, se definirán conceptos como DICOM o TC que se usarán a lo largo de éste y conviene saber lo que son, así como las distintas herramientas que se utilizarán.

1.1. Obtención de datos DICOM mediante una TC

DICOM (*Digital Imaging and Comunication in Medicine*) es el estándar internacional para manejar, visualizar, almacenar, imprimir y transmitir imágenes de pruebas médicas (ISO12052) [1].

Al contrario de lo que se puede pensar en un principio, DICOM es más que un formato de imagen, es un protocolo que abarca la transferencia, el almacenamiento y la visualización [13].

Pese a que su uso está mayoritariamente extendido en el campo en el que nació (la medicina) para obtener imágenes de cortes de partes del cuerpo de un paciente con fines diagnósticos, se puede usar en otros, como el de la restauración de bienes culturales, como es el caso de este proyecto.

En un archivo DICOM hay almacenado, además de metadatos, una imagen [8] (Figura 1.1).

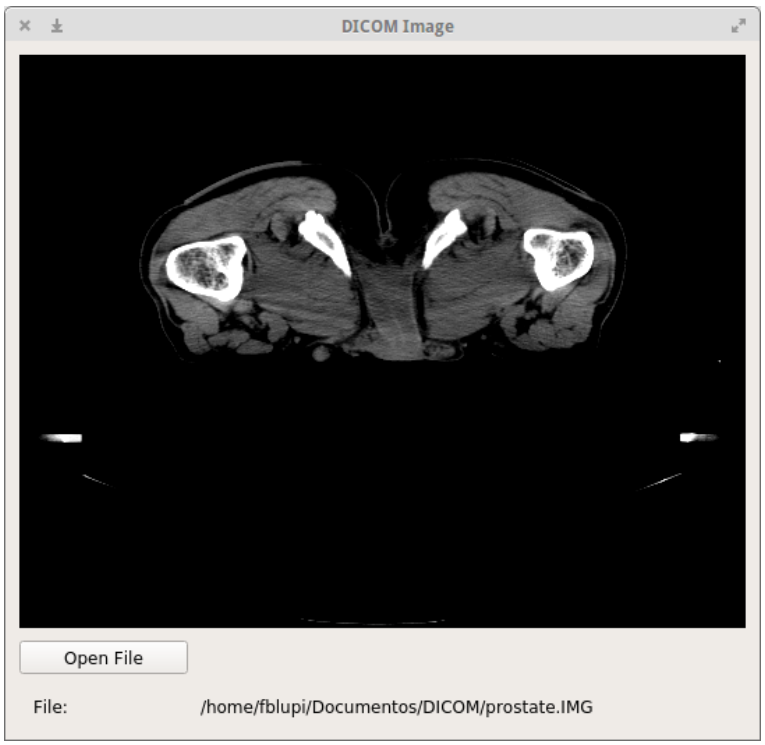

Figura 1.1: Imagen DICOM de una próstata visualizada con un programa diseñado para visualizar archivos DICOM

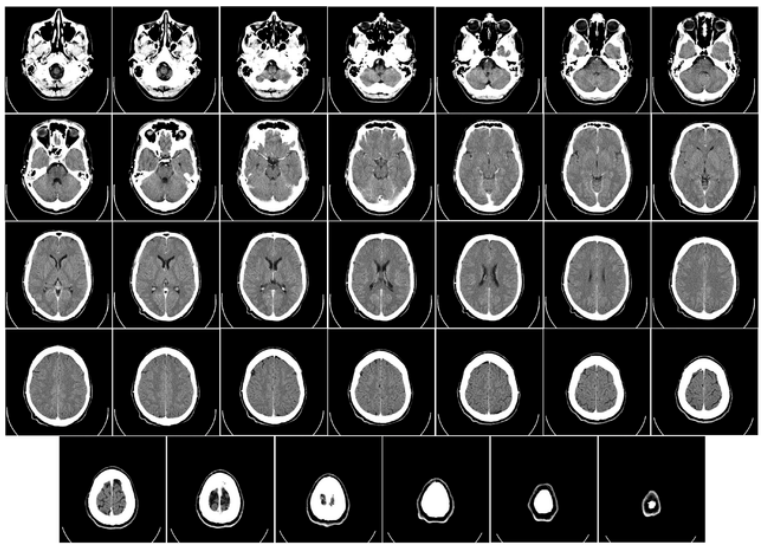

Figura 1.2: Serie de imágenes DICOM extraídas de una TC realizada a un cerebro

Cuando se realiza una TC se obtienen una serie de imágenes 2D (Figura 1.2) de cortes del objeto al que se le realiza el escáner. Éstas imágenes se encapsulan en archivos DICOM, y con todas ellas se puede pasar a un espacio en 3D y llegar a construir un modelo volumétrico en el que para cada voxel (Volumetric Pixel) se tiene el valor de densidad del objeto en ese punto.

El cómo se obtienen las imágenes con una TC no es objeto de estudio de este proyecto, por lo que no se entrará en mucho detalle. En muy resumidas cuentas, el aparato emite un haz de rayos X desde distintos ángulos al objeto y unos sensores recogen la radiación que absorbe en cada una de estas emisiones. Obteniendo el resultado final del promedio de todas las mediciones que realizan los sensores [2].

Para renderizar la imagen con una técnica de Direct Volume Rendering (DVR) hay que darle a cada voxel un valor de color y opacidad. Esto se realizará con lo que se denomina función de transferencia (de la que se hablará exhaustivamente más adelante) que a partir de los valores de densidad y gradiente de cada voxel obtiene un valor de color y opacidad y mediante *ray casting* o cualquier otra técnica de DVR se consigue el color para un pixel. Aquí es donde entrará en juego VTK, una librería que ayudará enormemente en la realización de estas tareas.

1.2. VTK

VTK es una librería gráfica orientada a objetos de alto nivel desarrollada por la compañía Kitware. Permite su uso en lenguajes compilados como C++ y Java o interpretados como Tcl y Python. Debido a los años que lleva desarrollándose, desde 1993 hasta la actualidad, se ha convertido en una librería enorme y compleja, pero al mismo tiempo potente [12]. Esto hace que su curva de aprendizaje tenga un inicio lento. Pero merece la pena su uso porque aunque cueste aprender, el tiempo invertido es mucho menor que al que habría que invertir para realizar las operaciones tan complejas que facilita.

1.3. Trabajos previos

Como ya se ha comentado anteriormente, el uso de las imágenes obtenidas con las TCs está muy extendido en el campo de la medicina, aunque puede ser usado en otros.

Ha tenido bastante repercursión en estudios anatómicos del estado actual de momias [5] y para realizar reconstrucciones de su posible estado anterior [6].

En el caso de estudio de este proyecto, la restauración de bienes culturales, obviamente, también puede aplicarse.

Tradicionalmente, se han utilizado radiografías para examinar el estado de las esculturas, pero el uso de TCs hace que se puedan obtener resultados que permitan realizar exámenes más precisos y exhaustivos a los restauradores.

Se podría hacer uso de herramientas médicas como OsiriX [14], AMILab [10] o RadiAnt; pero proporcionan presets para visualizar materiales de los que están compuestos los organismos humanos y la mayoría de ellas tienen un precio de licencia muy elevado.

También se podrían utilizar herramientas más genéricas como 3DSlicer [7], desarrollada por la misma compañía que VTK. No obstante, tiene muchas funcionalidades que no utilizarían los restauradores y hacen de ella una herramienta demasiado compleja.

Incluso podrían hacer uso de software específico para la visualización de esculturas, aunque hay muy pocas herramientas que tengan funcionalidad para trabajar con conjuntos de datos volumétricos. La más referenciada es Hyper3D [9] pero los resultados obtenidos no son buenos (Figura 1.3).

Figura 1.3: A la izquierda escultura de la Inmaculada Concepción. A la derecha su reconstrucción volumétrica usando Hyper3D y su preset para madera

1.4. Motivación

El resultado insatisfactorio obtenido con las herramientas disponibles ha motivado la idea de desarrollar 3DCurator, un software creado para que los restauradores puedan examinar esculturas de madera policromadas y así conocer los materiales de los que están compuestas, su estructura interna, observar si está dañada o ver cuántos años tiene la madera utilizada gracias a sus anillos.

Incluyendo también una serie de presets para poder visualizar unos u otros materiales sin que el usuario tenga por qué conocer los valores de densidad de estos para crear una función de transferencia que los visualice, pero proporcionando también un editor de funciones de transferencias con el que los usuarios más avezados puedan crear sus propios presets y exportarlos para que puedan ser utilizados por los demás.

Capítulo 2

Especificación de requisitos

Este capítulo es una Especificación de Requisitos Software para el software que se va a realizar siguiendo las directrices dadas por el estándar IEEE830 [3].

2.1. Introducción

2.1.1. Propósito

Este capítulo de especificación de requisitos tiene como objetivo definir las especificaciones funcionales y no funcionales para el desarrollo de un software que permitirá visualizar e interactuar con los datos DICOM obtenidos al someter a una escultura a una TC. Éste software será utilizado principalmente por restauradores.

2.1.2. Ámbito del sistema

En la actualidad los datos DICOM obtenidos tras una TC se utilizan, principalmente, en el campo donde surgieron, la medicina. No obstante, esto no significa que solo se pueda aplicar ahí. Con este software, llamado 3DCurator, se tratará de trasladar esta técnica al campo de la restauración de bienes culturales y poder visualizar e interactuar con los datos DICOM obtenidos con esculturas.

2.1.3. Definiciones, acrónimos y abreviaturas

- **ERS**: Especificación de Requisitos Software.
- **GUI** (*Graphic User Interface*): Interfaz gráfica de usuario.

- **DICOM** (*Digital Imaging and Comunication in Medicine*): Datos de donde se obtienen las imágenes.
- **TAC o TC** (Tomografía Axial Computerizada): Escáner en el que se obtienen los datos DICOM.
- **GPU** (*Graphic Precessor Unit*): Tarjeta gráfica.
- **VTK** (*The Visualization ToolKit*): Librería gráfica que se utilizará.
- **CMake** (*Cross platform Make*): Herramienta para generar código compilable en distintas plataformas.
- **Qt**: Librería que se utilizará para realizar la GUI.
- **Volumen**: Conjunto de datos en los que para cada posición XYZ se tiene un valor determinado.
- **Corte**: Vista de la figura a través de un plano. Por ejemplo, al cortar con una sierra un tronco por la mitad, se puede ver cómo es por dentro en esa posición por donde se ha cortado.
- **Función de transferencia**: Función utilizada para visualizar los datos deseados de un volumen.
- ***Direct Volume Rendering***: Visualización directa de volúmenes en la que cada valor del volumen se mapea con un determinado color y opacidad dado por una función de transferencia.
- ***Ray-Casting***: Técnica de *Direct Volume Rendering* utilizada para la visualización de volúmenes.
- ***Widget***: Elemento de la GUI.

2.1.4. Visión general del documento

Este capítulo consta de tres secciones:

- En la primera sección se realiza una introducción a éste y se proporciona una visión general de la ERS.
- En la segunda sección se realiza una descripción general a alto nivel del software, describiendo los factores que afectan al producto y a sus requisitos y con el objetivo de conocer las principales funcionalidades de éste.
- En la tercera sección se definen detalladamente los requisitos que deberá satisfacer el software.

2.2. Descripción general

2.2.1. Perspectiva del producto

El software 3DCurator tiene como objetivo interactuar con datos DICOM, pero no es el encargado de generarlos. Para generarlos se deberá utilizar algún escáner de TC.

Una vez obtenidos, no se necesitará ningún otro software adicional.

2.2.2. Funciones del producto

Las principales funcionalidades de este sistema serán:

- Cargar datos DICOM.
- Generar un volumen a partir de los datos cargados.
- Visualizar en 3D el volumen.
- Modificar la función de transferencia y cambiar colores asignados a cada material.
- Generar nuevos cortes.
- Visualizar los cortes generados.
- Guardar imagen de lo que se visualiza en la pantalla.

2.2.3. Características de los usuarios

Solo existe un tipo de usuario, que es la persona que desee interactuar con los datos DICOM de una escultura. Esta persona no tiene por qué tener habilidad con un equipo informático, por lo que 3DCurator deberá tener una GUI intuitiva y fácil de utilizar.

2.2.4. Restricciones

Se llevará a cabo un desarrollo evolutivo basado en un prototipo funcional en el que no están definidos todos los requisitos desde un principio y se irán añadiendo conforme se vayan completando y ocurriendo nuevos.

El software será libre, por lo que el código estará accesible en un repositorio de GitHub.

Se programará en C++ usando las librerías VTK para la visualización de gráficos y Qt para la GUI.

Aprovechando que se debe usar CMake para compilar las librerías mencionadas, se utilizará también para generar el proyecto, pues se puede generar código compilable en distintas plataformas.

2.2.5. Suposiciones y dependencias

El software se utilizará para poder visualizar esculturas de madera por lo que se tendrán en cuenta los materiales con los que están hechas la mayoría de estas. Si se introducen los datos DICOM de cualquier otra cosa con materiales distintos a los utilizados en las esculturas no se visualizará correctamente.

2.3. Requisitos específicos

2.3.1. Interfaces

La GUI se construirá con Qt y contendrá los siguientes elementos (Figura 2.1):

- **Barra de menú**: Típica barra de menús (archivo, editar, herramientas...).
- **Barras de herramientas**: Acceso rápido con los botones de las operaciones más utilizadas sobre cada *widget* de visualización.
- ***Widget* de visualización del volumen en 3D**: Con el que se podrá interactuar para girar, hacer zoom y ver la figura desde distintas posiciones.
- ***Widget* de visualización de cortes**: En el que se mostrará el corte que se genera con un plano determinado.
- **Menú de operaciones**: Menú organizado en pestañas donde se podrán realizar distintas operaciones como cambiar la función de transferencia y definir el plano para realizar un corte.

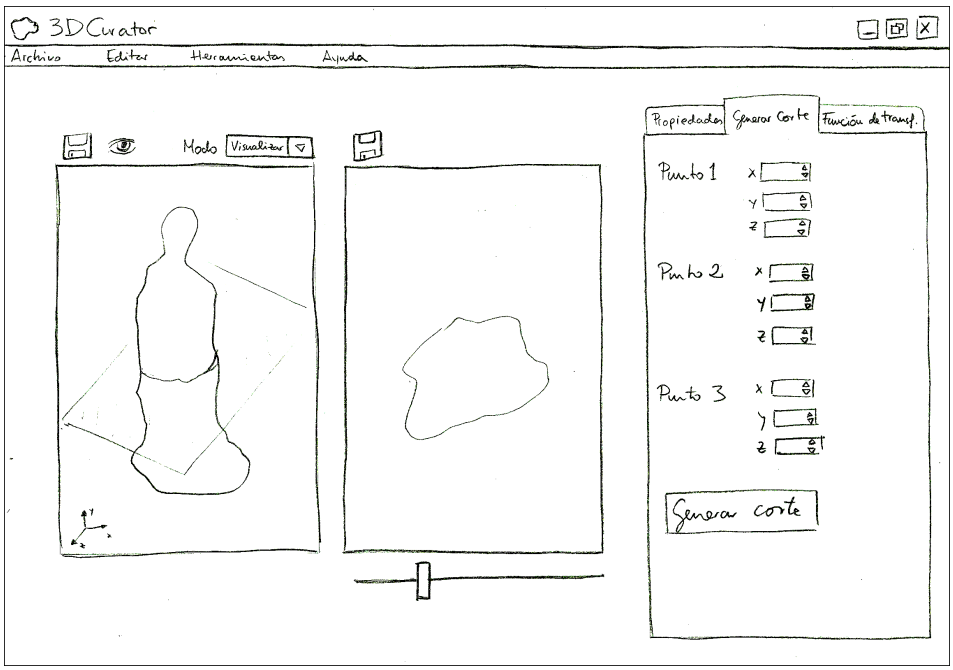

Figura 2.1: Boceto a mano alzada de la posible distribución de los elementos en la GUI

2.3.2. Funciones

El sistema tendrá que realizar distintas funciones que se comentaron anteriormente pero se profundizará en esta sección. Se han estructurado estas funciones por su objetivo separando cuatro subsecciones distintas: Lectura de datos, generación de cortes, visualización y configuración.

Lectura de datos

- **Seleccionar carpeta**: Cuando el usuario quiera cargar datos DICOM, le aparecerá una ventana donde se podrá escoger alguna carpeta de su sistema de forma que solo muestre las carpetas y no los archivos.

- **Verificar que la carpeta contiene datos DICOM**: Se tendrá que verificar que el usuario ha seleccionado una carpeta con datos DICOM y se le avisará si no lo ha hecho.

- **Cargar datos DICOM**: Cuando se haya seleccionado una carpeta correcta, se cargarán los datos y automáticamente los visualizará en 3D con la configuración por defecto.

Generación de cortes

- **Definir plano**: El usuario podrá definir un plano por donde realizar un corte a la figura. Para ello tendrá que introducir tres puntos. Por defecto, se introducirá un plano en el eje XZ a una altura de Y = 0, siendo la dirección de los ejes XYZ la que utiliza VTK (Figura 2.2). El plano se podrá visualizar en el *widget* de visualización 3D para ver gráficamente por dónde pasará.

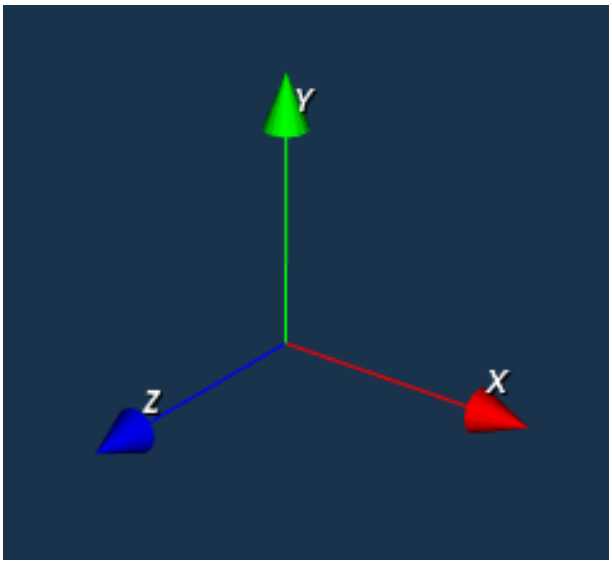

Figura 2.2: Dirección de los ejes XYZ

- **Modificar plano**: El usuario podrá modificar el plano o introduciendo nuevos puntos, o interactuando con la vista previa de éste en el *widget* de visualización 3D. Las operaciones que podrá realizar en este serán:
  - Rotar en cualquiera de los ejes.
  - Trasladar en dirección de la normal.
- **Generar corte**: Una vez se haya creado el plano deseado, se podrá mostrar el corte en el *widget* de visualización de cortes.

Visualización

- **Visualizar en 3D**: Cuando el usuario haya seleccionado una carpeta con datos DICOM, se mostrará en 3D en el *widget* izquierdo pudiendo rotar y hacer zoom interactuando con el ratón. Para visualizar el volumen se utilizarán técnicas de *Direct Volume Rendering* que proporcionan VTK como puede ser el *Ray-Casting*.

- **Visualizar corte**: Cuando el usuario haya cargado los datos DICOM y haya establecido un plano de corte, se podrá generar un corte a la figura por éste. Este corte se visualizará en el *widget* derecho.
- **Guardar imagen**: El usuario en todo momento podrá guardar una imagen (en formatos comunes JPG o PNG) de lo que está viendo en cada uno de los *widgets* seleccionando en una ventana que aparecerá cuando se elija la opción la dirección donde se guardará y el nombre del archivo.

Configuración

- **Cambiar color de fondo**: El usuario podrá cambiar el color de fondo del *widget* donde se mostrará la figura en 3D. Para ello podrá:
  - Elegir entre colores predeterminados.
  - Introducir un color mediante valores RGB.
  - Introducir un color mediante su código de color hexadecimal.
- **Cambiar función de transferencia**: El usuario podrá cambiar la función de transferencia utilizada para poder ver los distintos materiales con una paleta de color distinta a la que se da por defecto.

2.3.3. Requisitos de rendimiento

Al ser una aplicación de escritorio donde no se almacenarán datos sino que se mostrarán, los requisitos de rendimiento no se centrarán en la concurrencia de acceso ni en el almacenamiento, como lo podrían estar en una aplicación web.

Sin embargo, hay que tener en cuenta otros factores, como pueden ser el uso eficiente de memoria y no tener cargadas todas las figuras que se han estado visualizando, desechando la anterior cuando se carga una nueva.

El rendimiento gráfico también es importante, por eso y para obtener imágenes de mayor calidad, se utilizarán técnicas de *Direct Volume Rendering* como el *Ray-Casting*. Estas técnicas necesitan una gran cantidad de procesamiento, pero la velocidad de procesamiento de las GPUs actuales no deberían resultar un problema.

2.3.4. Restricciones de diseño

Al utilizar la librería VTK se seguirá su estructura a la hora de construir el software, y se tendrán restricciones en cuanto a funcionalidad que se pueda

construir con ésta. No obstante es una librería muy completa y no se debería encontrar ninguna restricción viendo otros programas de visualización de datos médicos que se han construido usando esta librería.

2.3.5. Atributos del sofware

Al usar CMake, se podrá crear un software multiplataforma que funcione en cualquier sistema operativo.

El software generado deberá ser fiable, porque aunque no trabaje con datos sensibles cuya pérdida pueda ser grave, siempre resulta molesto utilizar un software con fallos que interrumpan durante su uso.

También se debe tener en cuenta que el software sea mantenible pues, al ser libre, otros desarrolladores pueden colaborar en su desarrollo y debe estar bien documentado para que esto sea una tarea fácil.

Capítulo 3

Planificación

En este capítulo comentaré la planificación inicial de tiempo en la que se llevará a cabo este TFG y la estimación de horas para cada tarea.

3.1. Fechas y aclaraciones

La primera reunión con mi tutor fue el día **11 de Noviembre**, así que se puede dar esa fecha como fecha de inicio. La fecha de entrega, en este momento en el que se realiza la planificación, es desconocida. Pero espero tener el software terminado para la última semana de Mayo o primera de Junio, por lo que he puesto como fecha de fin el **9 de Junio**.

Al llevarse a cabo un desarrollo evolutivo incremental, no están concretados todos los requisitos que satisfará el software, por lo que solo están definidos los requisitos iniciales, aunque si se han planificado horas para las posteriores mejoras. Estos requisitos se descompondrán en tareas una vez se definan y se irán acoplando al espacio temporal reservado.

Al hacerse esta planificación tras la segunda reunión, todas las tareas tanto de la primera como de la segunda no tienen el número de horas estimadas, sino las empleadas realmente.

La estimación para cada tarea ha sido difícil de asignar por ser la primera vez que voy a trabajar con VTK y Qt, pero espero que la experiencia que he adquirido en las prácticas que he realizado en las distintas asignaturas que he tenido durante estos cuatro años me ayude y logre hacer una buena planificación.

3.2. Planificación

He troceado el calendario con un *sprint* de reunión en reunión (cada dos semanas) y el resultado ha sido el siguiente:

- **Reunión 1** (13/11/15 - 26/11/15)
  - Instalación de entorno de desarrollo (12 horas)
  - Aprender VTK (4 horas)
  - Aprender Qt (3 horas)
  - Aprender estructura DICOM (3 horas)
- **Reunión 2** (27/11/15 - 10/12/15)
  - Especificación de requisitos (6 horas)
  - Planificación (2 horas)
  - Visualización de volumen básica (10 horas)
- **Reunión 3** (11/12/15 - 7/1/16)
  - Interacción con la cámara (8 horas)
  - Estudiar función de transferencia (12 horas)
  - Implementar función de transferencia (12 horas)
  - Escribir en la memoria (4 horas)
- **Reunión 4** (8/1/16 - 21/1/16)
  - Modificar función de transferencia (15 horas)
  - Visualizar plano (5 horas)
- **Reunión 5** (22/1/16 - 4/2/16)
  - Modificar plano dando puntos (5 horas)
  - Modificar plano interactuando con éste (15 horas)
- **Reunión 6** (5/2/16 - 18/2/16)
  - Generar corte con el plano (8 horas)
  - Visualizar corte generado (12 horas)
- **Reunión 7** (19/2/16 - 3/3/16)
  - Interactuar con cortes generados (6 horas)
  - Guardar imágenes (8 horas)

  - Testeos intensivos (4 horas)
  - Corrección de *bugs* (6 horas)
  - Escribir en la memoria (4 horas)
- **Reunión 8** (4/3/16 - 17/3/16)
  - Mejora #1 (18 horas)
  - Escribir en la memoria (2 horas)
- **Reunión 9** (18/3/16 - 31/3/16)
  - Mejora #2 (18 horas)
  - Escribir en la memoria (2 horas)
- **Reunión 10** (1/4/16 - 14/4/16)
  - Mejora #3 (18 horas)
  - Escribir en la memoria (2 horas)
- **Reunión 11** (15/4/16 - 28/4/16)
  - Mejora #4 (18 horas)
  - Escribir en la memoria (2 horas)
- **Reunión 12** (29/4/16 - 12/5/16)
  - Mejora #5 (18 horas)
  - Escribir en la memoria (2 horas)
- **Reunión 13** (13/5/16 - 26/5/16)
  - Mejora #6 (18 horas)
  - Escribir en la memoria (2 horas)
- **Reunión 14** (27/5/16 - 9/6/16)
  - Revisar y terminar la memoria (6 horas)
  - Preparar la exposición (10 horas)

En total se estima que se realicen unas **300 horas**. Se llevará a cabo un recuento de horas realizadas para ver, finalmente, cuántas se han necesitado.

Capítulo 4

Análisis

4.1. Técnicas de renderizado

A la hora de renderizar un conjunto de datos volumétricos para obtener una imagen en 3D, se pueden utilizar distintas técnicas y VTK proporciona una serie de clases para su uso:

- ***Marching Cubes***: Con este algoritmo se obtiene una malla poligonal de una isosuperficie a partir de un conjunto de datos volumétrico (Figura 4.1) [11]. Se puede usar en VTK con `vtkMarchingCubes`.

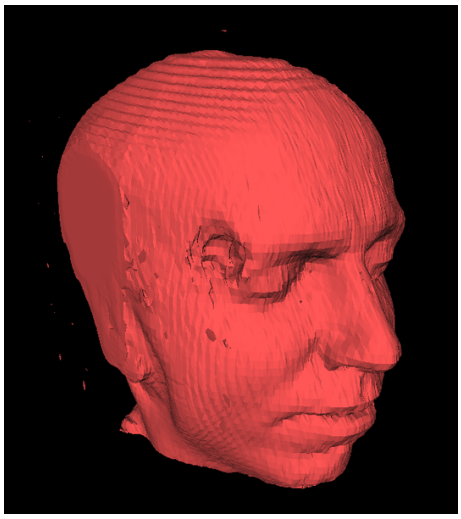

Figura 4.1: Cabeza extraída de 150 cortes obtenidos por una IRM usando *marching cubes* (sobre 150.000 triángulos). Imagen extraída de `https://en.wikipedia.org/wiki/File:Marchingcubes-head.png`

- **Texturas2D**: Se utilizan planos de corte alineados a los ejes de coordenadas. Por lo que se tendría una serie de cortes sobre el plano sagital, otra sobre el coronal y otra sobre el axial. Se realiza una interpolación bilineal para obtener la imagen final (Figura 4.2 [12]). Se puede usar en VTK con `vtkVolumeTextureMapper`.

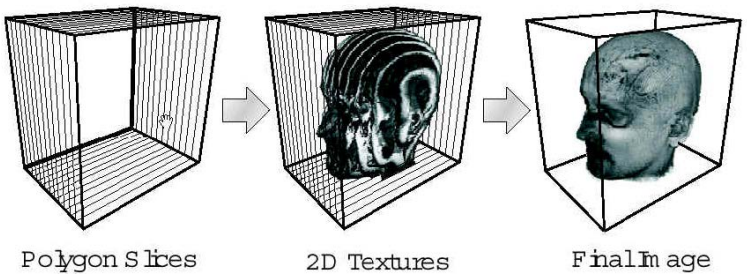

Figura 4.2: Esquema del proceso de renderizado usando texturas 2D. Imagen extraída del apéndice B del libro *An Introduction to Programming for Medical Image Analysis with the Visualization Toolkit* [12]

- **Texturas3D**: Esta técnica es similar a la anterior, pero ahora los datos se cargan en una textura 3D y los cortes se dibujan paralelos a la dirección de vista. A diferencia de las texturas 2D, usa interpolación trilineal y no es necesario tener almacenado en memoria tres copias de los mismos datos (Figura 4.3) [12]. Se puede usar en VTK con `vtkVolumeTextureMapper3D`.

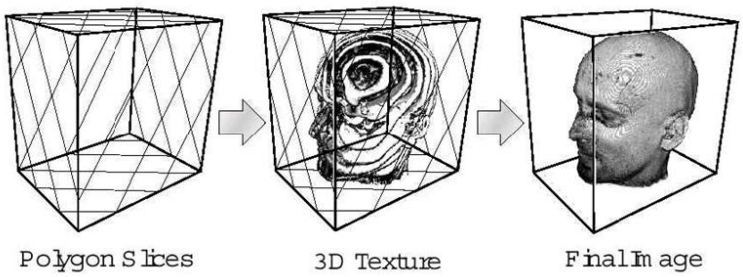

Figura 4.3: Esquema del proceso de renderizado usando texturas 3D. Imagen extraída del apéndice B del libro *An Introduction to Programming for Medical Image Analysis with the Visualization Toolkit* [12]

- ***Volume Ray Casting***: Es una técnica en el que para cada pixel de la imagen se lanza un rayo que atraviesa el volumen. Para cada voxel se obtiene su color y opacidad usando una función de transferencia. Cuando el rayo sale del volumen se calcula el color y opacidad

del pixel como el acumulado por el rayo. Existe una versión de este algoritmo que hace uso de la GPU para acelerar ostensiblemente el tiempo de la operación (Figura 4.4) [12]. Se puede usar en VTK con `vtkFixedVolumeRayCastMapper`, `vtkVolumeRayCastMapper` (usan CPU), `vtkGPUVolumeRayCastMapper` (usa GPU) y `vtkSmartVolumeMapper` (según el contexto usa CPU o GPU).

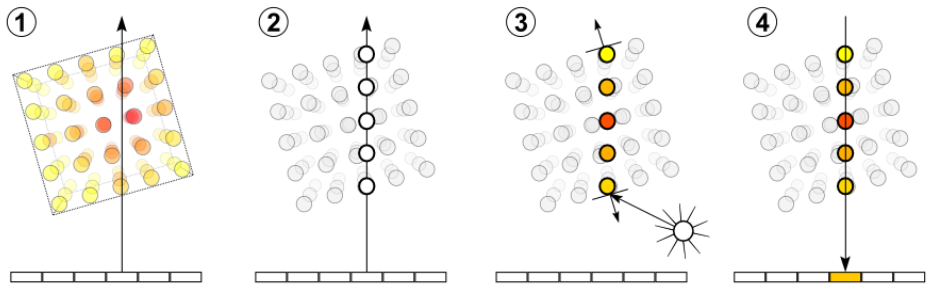

Figura 4.4: Esquema del proceso de *ray casting*. Imagen extraída de `https://en.wikipedia.org/wiki/File:Volume_ray_casting.png`

De entre todas estas técnicas, se podrían descartar rápidamente la de *marching cubes*: pues tan solo trabaja con isosuperficies y la de texturas 2D: pues la opción de texturas 3D es más rápida y usa menos recursos.

Por tanto ya solo habría que elegir entre texturas 3D o *ray casting*. Hasta hace unos años, VTK no proporcionaba un algoritmo de *ray casting* que usase la GPU. Por tanto la opción habría sido sencilla, pero durante los últimos años han trabajado en esto haciendo del *ray casting* la opción preferible.

4.2. Volume Mapper

Para poder visualizar un volumen con VTK mediante Direct Volume Rendering (DVR), necesitamos un *Volume Mapper*. La librería nos ofrece varias alternativas:

- `vtkAMRVolumeMapper`
- `vtkFixedVolumeRayCastMapper`
- `vtkGPUVolumeRayCastMapper`
- `vtkSmartVolumeMapper`
- `vtkVolumeRayCastMapper`

- `vtkVolumeTextureMapper`
- `vtkVolumeTextureMapper3D`

Entre esta lista tenemos algunos que utilizan o texturas o *ray casting*, o la CPU o la GPU. Pero hay uno que es especial con respecto al resto. Se trata de `vtkSmartVolumeMapper`.

Este *Volume Mapper* es una versión mejorada del `vtkGPUVolumeRayCast Mapper` por lo que utiliza la GPU (si el dispositivo cuenta con una) y la técnica de *ray casting*. Además cuenta con nuevas características con respecto al resto, como el poder definir infinitos planos de corte para poder ver el interior del volumen [15].

Por tanto, el *Volume Mapper* utilizado será el `vtkSmartVolumeMapper`.

4.3. Función de transferencia

La función de transferencia es la encargada de dar a un valor de intensidad las propiedades de color y opacidad que le corresponden para la visualización del volumen.

En VTK la función de transferencia forma parte de la clase `vtkVolume Property` [4]. Para ello proporciona otras dos clases:

- `vtkColorTransferFunction`: Para definir el color. Se enlaza a `vtk VolumeProperty` con el método `SetColor`.
- `vtkPiecewiseFunction`: Para definir la opacidad (tanto escalar como gradiente). La opacidad escalar se enlaza a `vtkVolumeProperty` con el método `SetScalarOpacity` y la gradiente con `SetGradientOpacity`.

Podemos, por tanto, diferenciar tres partes fundamentales en la función de transferencia, la encargada de dar la propiedad de color y las dos de dar la propiedad de opacidad. Ambas trabajan de forma independiente. Es decir, cuando se define un punto en una de ellas, no tiene por qué definirse en la otra.

4.3.1. Color

Para definir esta función (`vtkColorTransferFunction`), hay que agregar puntos para valores de intensidad a los que se les asignará un color. VTK se encargará de interpolar entre un punto y otro (Figura 4.5).

Por defecto, cuando no hay ningún punto, a todos los valores de intensidad les corresponderá un color negro. De forma parecida se comporta

cuando solo hay un punto pero en lugar de negro, les corresponderá el color del punto que se ha definido.

VTK permite trabajar tanto con HSV como con RGB y para añadir un punto hay que utilizar `AddHSVPoint` o `AddRGBPoint`. A estos métodos se les pasa un primer parámetro en coma flotante con el valor de intensidad donde se establecerá ese punto y otros tres con las distintas exponentes (*hue*, *saturation*, *brightness* o *red*, *green*, *blue*).

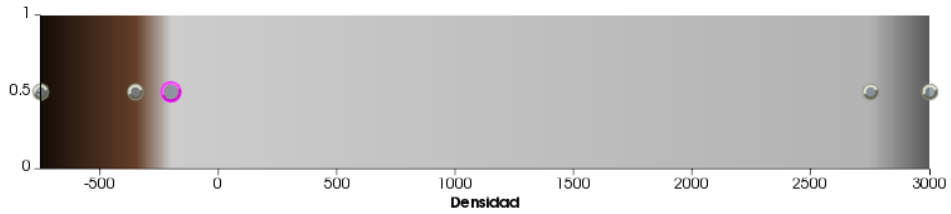

Figura 4.5: Parte de color de la función de transferencia del *preset CT-WoodSculpture* creado para visualizar esculturas de madera policromadas. Dos puntos definen el color. Uno en -750 con un tono más oscuro y otro en -350 con un tono más claro. El estuco se pinta con un color gris claro y también viene definido por dos puntos: -200 y 2750. Finalmente, el metal se verá con un tono gris oscuro definido con un punto en 3000.

4.3.2. Opacidad

El valor de opacidad se obtendría como el **producto de la opacidad escalar por la gradiente**. Si no se define alguna de las dos, se definiría como un valor constante de 1 para que solo se viese el resultado de la que sí está definida.

Opacidad escalar

Con el color no bastaría, pues si comprobásemos ahora añadiéndole tan solo el `vtkColorTransferFunction` al `vtkVolumeProperty` observaríamos que no se pinta nada en pantalla. Esto es porque por defecto, al no tener ningún punto la función de opacidad (`vtkPiecewiseFunction`) es una constante con valor 0 (transparente).

Para definir esta función se trabaja de forma parecida a como se hace con el color, añadiendo puntos. El método que hay que utilizar es `AddPoint` al que se le pasan dos parámetros en coma flotante. El primero con el valor de intensidad y el segundo con la opacidad en ese punto. Para obtener los valores en puntos intermedios, se interpola entre los dos puntos en los que está. De forma que si para el valor de intensidad 100 hemos definido una

opacidad de 0.5 y para el de 200 1, al valor de intensidad 150 le corresponderá 0.75.

Combinando color y opacidad escalar podemos obtener una función de transferencia para visualizar nuestro volumen (Figura 4.6), pero para obtener mejores resultados, habrá que utilizar la opacidad gradiente.

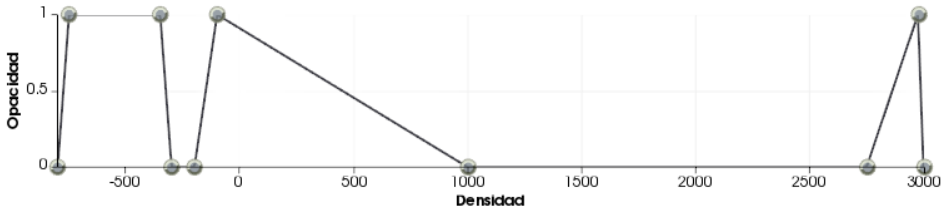

Figura 4.6: Parte de opacidad escalar de la función de transferencia del *preset CT-WoodSculpture* creado para visualizar esculturas de madera policromadas. Se pueden observar tres regiones. La primera corresponde a la madera, la segunda al estuco y la última al metal

Opacidad gradiente

La opacidad gradiente utiliza el vector gradiente para dar el valor de opacidad. Con éste se puede conseguir **dar un mayor valor a regiones de los bordes, así como menor a regiones planas** es decir, donde no varía el valor de intensidad de sus vecinos de alrededor.

El gradiente se mide como la cantidad que varía la intensidad en una unidad de distancia. Este cálculo del gradiente lo realiza VTK cuando genera el volumen de forma transparente sin que haya que añadir nada al código.

Para poder definir la función de la opacidad gradiente, al igual que con las demás, hay que añadir puntos con la misma función que se usaba con la opacidad escalar (`AddPoint`).

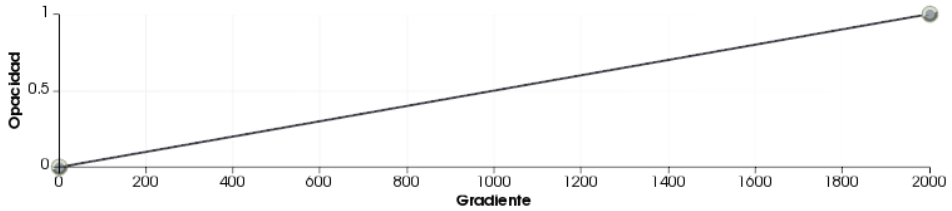

Figura 4.7: Parte de opacidad gradiente de la función de transferencia del *preset CT-WoodSculpture* creado para visualizar esculturas de madera policromadas. Se obtendría un valor cercano a 1 en la opacidad en aquellas zonas más cercanas a los bordes entre materiales pues se ha establecido que para un gradiente 0 la opacidad sea 0, y para 2000, 1.

Bibliografía

[1] About dicom. http://dicom.nema.org/Dicom/about-DICOM.html.

[2] Escáner o tomografía. ¿qué es y cómo funciona? http://www.fisioterapia-online.com/videos/escaner-o-tomografia-que-es-y-como-funciona.

[3] Especificación de requisitos según el estándar de iee 830. https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf.

[4] Vtk example - medical 4. https://github.com/Kitware/VTK/blob/master/Examples/Medical/Cxx/Medical4.cxx.

[5] Federico Cesarani, Maria Cristina Martina, Andrea Ferraris, Renato Grilletto, Rosa Boano, Elisa Fiore Marochetti, Anna Maria Donadoni, and Giovanni Gandini. Whole-body three-dimensional multidetector ct of 13 egyptian human mummies. *American Journal of Roentgenology*, pages 597–606, March 2003. http://dx.doi.org/10.2214/ajr.180.3.1800597.

[6] Federico Cesarani, Maria Cristina Martina, Renato Grilletto, Rosa Boano, Anna Maria Donadoni, Valter Capussotto, Andrea Giuliano, Maurizio Celia, and Giovanni Gandini. Facial reconstruction of a wrapped egyptian mummy using mdct. *American Journal of Roentgenology*, pages 755–758, September 2004. http://dx.doi.org/10.2214/ajr.183.3.1830755.

[7] Andriy Fedorov, Reinhard Beichel, Jayashree Kalpathy-Cramer, Julien Finet, Jean-Cristophe Fillion-Robin, Sonia Pujol, Christian Bauer, Dominique Jennings, Fiona Fennessy, Milan Sonka, John Buatti, Stephen Aylward, James Miller, Steve Pieper, and Ron Kikinis. 3d slicer as an image computing platform for the quantitative imaging network. *Elsevier*, pages 1323–41, November 2012. http://dx.doi.org/10.1016/j.mri.2012.05.001.

[8] David Gobbi. Vtk classes for dicom data. http://dgobbi.github.io/vtk-dicom/doc/vtk-dicom.pdf, 2015.

[9] Min H Kim, Holly Rushmeier, John Ffrench, Irma Passeri, and David Tidmarsh. Hyper3d: 3d graphics software for examining cultural artifacts. *Journal on Computing and Cultural Heritage (JOCCH)*, 7(3):14, February 2014. `http://dx.doi.org/10.1145/2567652`.

[10] Karl Krissian, Francisco Santana-Jorge, Daniel Santana-Cedrés, Carlos Falcón-Torres, Sara Arencibia, Sara Illera, Agustín Trujillo, Claire Chalopin, and Luis Alvarez. Amilab software: Medical image analysis, processing and visualization. In *MMVR*, pages 223–237, January 2012. `http://researchgate.net/publication/221853670_AMILab_software_medical_image_analysis_processing_and_visualization`.

[11] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, August 1987. `http://doi.acm.org/10.1145/37402.37422`.

[12] Xenophon Papademetris and Alark Joshi. *An Introduction to Programming for Medical Image Analysis with the Visualization Toolkit*. BioImage Suite, 2nd edition, 2009.

[13] Oleg S. Pianykh. *Digital Imaging and Communications in Medicine (DICOM). A Practical Introduction and Survival Guide*. Springer, 2nd edition, 2012.

[14] Antoine Rosset, Luca Spadola, and Osman Ratib. Osirix: An open-source software for navigating in multidimensional dicom images. *Journal of Digital Imaging*, pages 205–216, September 2004. `http://dx.doi.org/10.1007/s10278-004-1014-6`.

[15] Aashish Chaudhary Sandy McKenzie, Lisa Avila and Sankhesh Jhaveri. Volume rendering improvements in vtk. *Kitware Blog*, 2014. `https://blog.kitware.com/volume-rendering-improvements-in-vtk/`.