



ugr | Universidad
de Granada

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, 29 de junio 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 29 de junio 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 29 de junio de 2016.

El director:

Francisco Javier Melero Rus

Agradecimientos

Poner aquí agradecimientos...

Índice general

1. Introducción	1
1.1. Obtención de datos DICOM mediante una TC	1
1.2. VTK	3
1.3. Trabajos previos	3
1.4. Motivación	5
2. Especificación de requisitos	7
2.1. Introducción	7
2.1.1. Propósito	7
2.1.2. Ámbito del sistema	7
2.1.3. Definiciones, acrónimos y abreviaturas	8
2.1.4. Visión general del documento	9
2.2. Descripción general	9
2.2.1. Perspectiva del producto	9
2.2.2. Funciones del producto	9
2.2.3. Características de los usuarios	10
2.2.4. Restricciones	10
2.2.5. Suposiciones y dependencias	10
2.3. Requisitos específicos	10
2.3.1. Interfaces	10
2.3.2. Funciones	11
2.3.3. Requisitos de rendimiento	13
2.3.4. Restricciones de diseño	14
2.3.5. Atributos del software	14
3. Planificación	15
3.1. Fechas y aclaraciones	15
3.2. Planificación	16
3.3. Metodología utilizada	17
4. Análisis	19
4.1. Historias de usuario	19
4.1.1. Product backlog	19

4.1.2. Tarjetas de las historias de usuario	20
5. Diseño	31
5.1. Diagrama arquitectónico	31
5.2. Diagramas de clases	31
5.2.1. Application	32
5.2.2. Charts	33
5.2.3. Volume	34
5.3. Diagramas de secuencia	34
5.3.1. OpacityTFChart	35
5.3.2. ColorTFChart	36
5.3.3. ColorTransferControlPointsItem	37
5.3.4. Figura	38
5.3.5. TransferFunction	41
5.3.6. MainWindow	43
5.3.7. InteractorStyleDelete	52
5.3.8. InteractorStyleImage	56
6. Implementación	57
6.1. Plataforma de desarrollo	57
6.2. Instalación y configuración	58
6.2.1. Entorno de desarrollo	58
6.2.2. Compilar bibliotecas	59
6.2.3. Configurar proyecto	60
6.3. Conceptos clave en Volume Rendering	61
6.3.1. Técnicas de renderizado	61
6.3.2. Volume Mapper	63
6.3.3. Función de transferencia	64
6.3.4. Escala Hounsfield	66
6.4. Fases de desarrollo	67
6.4.1. Lectura de imágenes DICOM	67
6.4.2. Reconstrucción volumétrica	68
6.4.3. Generación de cortes	70
6.4.4. Guardar imágenes	70
6.4.5. Editar función de transferencia	71
6.4.6. Importar y exportar función de transferencia	72
6.4.7. Realizar medida	75
6.4.8. Borrar partes innecesarias	76
6.4.9. Exportar malla de triángulos	79
6.4.10. Mostrar valor de densidad	80
6.4.11. Realizar varias medidas	80
6.4.12. Cambiar color de fondo de los visores	81
Bibliografía	84

Índice de figuras

1.1.	Imagen DICOM de una próstata visualizada con un programa diseñado para visualizar archivos DICOM	2
1.2.	Serie de imágenes DICOM extraídas de una TC realizada a un cerebro	2
1.3.	A la izquierda escultura de la Inmaculada Concepción. A la derecha su reconstrucción volumétrica usando Hyper3D y su preset para madera	4
2.1.	Boceto a mano alzada de la posible distribución de los elementos en la GUI	11
2.2.	Dirección de los ejes XYZ	12
5.1.	Diagrama de clases del paquete <i>Application</i>	32
5.2.	Diagrama de clases del paquete <i>Charts</i>	33
5.3.	Diagrama de clases del paquete <i>Volume</i>	34
5.4.	Diagrama de secuencia del constructor de <i>OpacityTFChart</i> .	35
5.5.	Diagrama de secuencia del constructor de <i>ColorTFChart</i> . .	36
5.6.	Diagrama de secuencia del evento de doble click de <i>ColorTransferControlPointsItem</i>	37
5.7.	Diagrama de secuencia del constructor de <i>Figura</i>	38
5.8.	Diagrama de secuencia del método <i>setProperties</i> de <i>Figura</i> .	39
5.9.	Diagrama de secuencia del método <i>connectComponents</i> de <i>Figura</i>	39
5.10.	Diagrama de secuencia del método <i>setDICOMFolder</i> de <i>Figura</i>	40
5.11.	Diagrama de secuencia del método <i>read</i> de <i>TransferFunction</i>	41
5.12.	Diagrama de secuencia del método <i>readData</i> de <i>TransferFunction</i>	41
5.13.	Diagrama de secuencia del método <i>write</i> de <i>TransferFunction</i>	42
5.14.	Diagrama de secuencia del constructor de <i>MainWindow</i> . . .	43
5.15.	Diagrama de secuencia del método <i>connectComponents</i> de <i>MainWindow</i>	44
5.16.	Diagrama de secuencia del método <i>importDICOM</i> de <i>MainWindow</i>	45

5.17. Diagrama de secuencia del método <i>exportImageFromRenderWindow</i> de <i>MainWindow</i>	46
5.18. Diagrama de secuencia del método <i>exportMeshToFile</i> de <i>MainWindow</i>	47
5.19. Diagrama de secuencia del método <i>changeBackgroundColor</i> de <i>MainWindow</i>	48
5.20. Diagrama de secuencia del método <i>addRule</i> de <i>MainWindow</i>	49
5.21. Diagrama de secuencia del método <i>deleteRule</i> de <i>MainWindow</i>	50
5.22. Diagrama de secuencia del método <i>clearAllRules</i> de <i>MainWindow</i>	51
5.23. Diagrama de secuencia del método <i>OnLeftButtonDown</i> de <i>InteractorStyleDelete</i>	52
5.24. Diagrama de secuencia del método <i>deleteByImages</i> de <i>InteractorStyleDelete</i>	53
5.25. Diagrama de secuencia del método <i>deleteImage</i> de <i>InteractorStyleDelete</i>	54
5.26. Diagrama de secuencia del método <i>searchInitialVoxel</i> de <i>InteractorStyleDelete</i>	55
5.27. Diagrama de secuencia del método <i>OnMouseMove</i> de <i>InteractorStyleImage</i>	56
 6.1. Cabeza extraída de 150 cortes obtenidos por una IRM usando <i>marching cubes</i> (sobre 150.000 triángulos). Imagen extraída de https://en.wikipedia.org/wiki/File:Marchingcubes-head.png	61
6.2. Esquema del proceso de renderizado usando texturas 2D. Imagen extraída del apéndice B del libro <i>An Introduction to Programming for Medical Image Analysis with the Visualization Toolkit</i> [17]	62
6.3. Esquema del proceso de renderizado usando texturas 3D. Imagen extraída del apéndice B del libro <i>An Introduction to Programming for Medical Image Analysis with the Visualization Toolkit</i> [17]	62
6.4. Esquema del proceso de <i>ray casting</i> . Imagen extraída de https://en.wikipedia.org/wiki/File:Volume_ray_casting.png	63
6.5. Parte de color de la función de transferencia del <i>preset CT-WoodSculpture</i> 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.	65

6.6. Parte de opacidad escalar de la función de transferencia del <i>preset CT-WoodSculpture</i> 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	66
6.7. Parte de opacidad gradiente de la función de transferencia del <i>preset CT-WoodSculpture</i> 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.	66
6.8. Programa sencillo para visualizar una serie de imágenes DICOM	68
6.9. Primer renderizado sobre la figura de San Juan Evangelista con una función de transferencia que no utilizaba la opacidad gradiente	69
6.10. Barra de herramientas con la que añadir y quitar puntos a la función de transferencia	69
6.11. Primera implementación del pano que corta la figura y renderiza el corte en otro <i>widget</i>	70
6.12. Interfaz para editar la función de transferencia	72
6.13. Primera implementación de regla para medir. Solo se podía utilizar una en el visor de cortes	76
6.14. La camilla tapa la espalda de la figura e impide verla	77
6.15. La camilla ha sido eliminada y permite ver la espalda a diferencia de antes (Figura 6.14)	79
6.16. Malla generada con un valor de isosuperficie de 2976 HU para poder extraer los clavos	80
6.17. A la derecha, barra de las reglas y en los visores cada una de ellas midiendo algo distinto	81

Índice de cuadros

4.1.	Historias de usuario	20
4.2.	Historia de usuario - Cargar datos DICOM	20
4.3.	Historia de usuario - Generar reconstrucción 3D	21
4.4.	Historia de usuario - Cambiar color de fondo	21
4.5.	Historia de usuario - Cambiar material de la figura	22
4.6.	Historia de usuario - Funciones de transferencia por defecto	22
4.7.	Historia de usuario - Editar función de transferencia	23
4.8.	Historia de usuario - Exportar función de transferencia	23
4.9.	Historia de usuario - Importar función de transferencia	24
4.10.	Historia de usuario - Generar y visualizar cortes	24
4.11.	Historia de usuario - Editar plano de corte	25
4.12.	Historia de usuario - Habilitar y deshabilitar el plano de corte	25
4.13.	Historia de usuario - Posiciones del plano de corte por defecto	26
4.14.	Historia de usuario - Guardar imágenes de las ventanas	26
4.15.	Historia de usuario - Realizar medida	27
4.16.	Historia de usuario - Añadir regla	27
4.17.	Historia de usuario - Eliminar regla	28
4.18.	Historia de usuario - Habilitar y deshabilitar regla	28
4.19.	Historia de usuario - Eliminar partes	29
4.20.	Historia de usuario - Generar malla	29
4.21.	Historia de usuario - Mallas de materiales por defecto	30
4.22.	Historia de usuario - Exportar malla	30
4.23.	Historia de usuario - Mostrar valor escalar de un pixel en HU	30
6.1.	Valores en HU de distintos materiales presentes en imágenes de esculturas de madera	67

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 Communication 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 [18].

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 [13] (Figura 1.1).

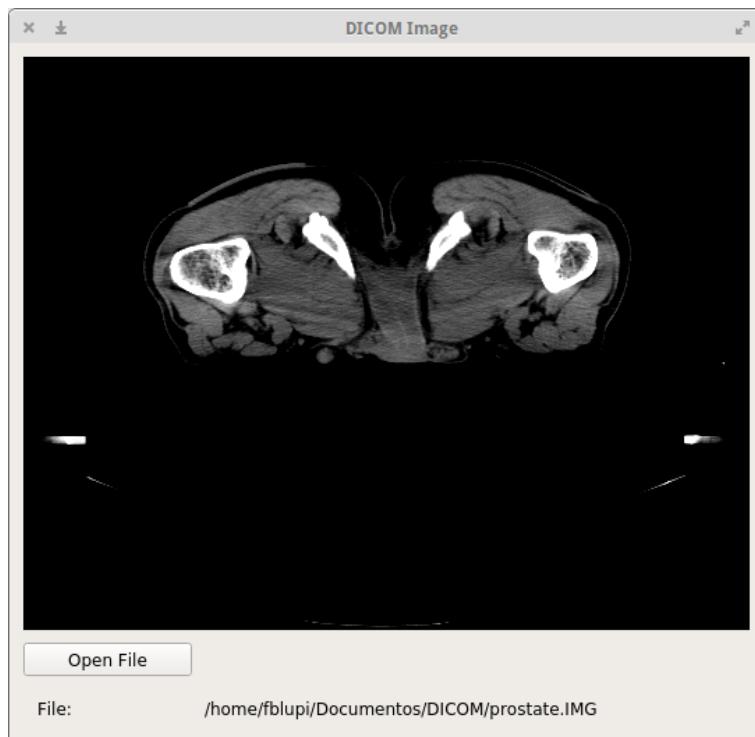


Figura 1.1: Imagen DICOM de una próstata visualizada con un programa diseñado para visualizar archivos DICOM

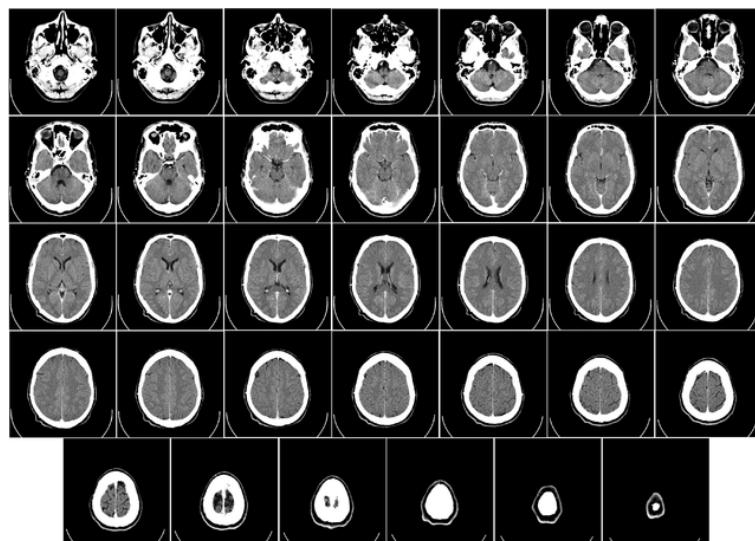


Figura 1.2: Serie de imágenes DICOM extraídas de una TC realizada a un cerebro

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 [4].

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 [17]. 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 repercusión en estudios anatómicos del estado actual de momias [10] y para realizar reconstrucciones de su posible estado anterior [11].

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 [19], AMILab [15] 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 [12], 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 [14] 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 [5].

Esta especificación de requisitos contiene aquellos requisitos iniciales. No obstante se han añadido posteriormente más conforme ha ido avanzando el producto.

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.
- **Widget:** Elemento de la GUI.
- **Volumen:** Conjunto de datos en los que para cada posición XYZ se tiene un valor determinado.
- **Voxel VOlumentric piXEL:** celda en la matriz 3D del conjunto de datos del volumen.
- **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.
- **TF (Función de transferencia):** Función utilizada para visualizar los datos deseados de un volumen.
- **Preset:** Función de transferencia previamente configurada.
- **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.
- **Marching-Cubes:** Técnica para generar malla de polígonos a partir de un volumen y un valor de isosuperficie.
- **HU (*Hounsfield Units*):** Unidad de medida escalar del valor de densidad en un *voxel* del voluen

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.

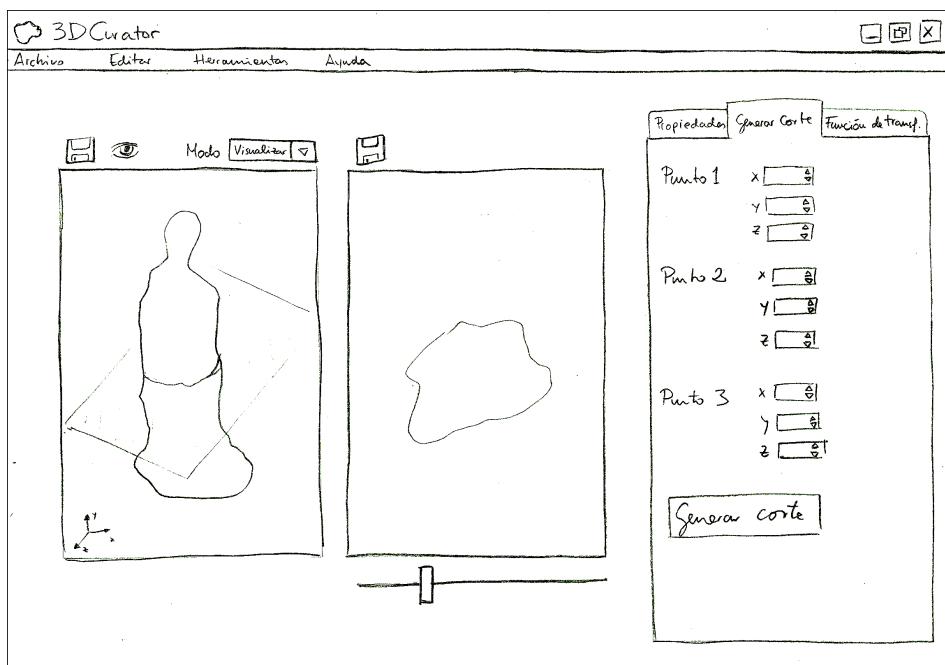


Figura 2.1: Boceto a mano alzada de la posible distribución de los elementos en la GUI

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.

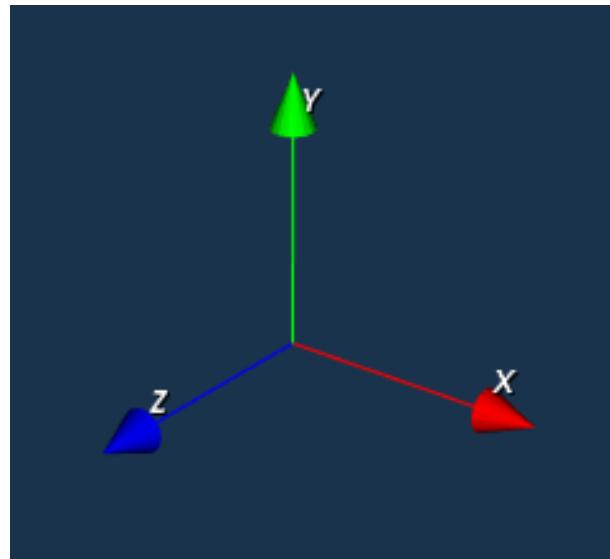


Figura 2.2: Dirección de los ejes XYZ

- **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 software

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 (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.

3.3. Metodología utilizada

Lalala

Capítulo 4

Análisis

En este capítulo se describirán las distintas historias de usuario que han sido implementadas en el software.

4.1. Historias de usuario

4.1.1. Product backlog

Al haberse seguido un ciclo evolutivo de desarrollo, al principio de este no se tenía una lista con todas las historias de usuario, sino que se han ido añadiendo conforme han surgido.

A continuación se muestran el listado de historias de usuario (Product Backlog) completo, y para cada historia de usuario sus dependencias, estimación (en puntos de historia) y prioridad.

#	Descripción	Dep.	Est.	Prio.
1	Cargar datos DICOM	-	2	1
2	Generar reconstrucción 3D	-	6	1
3	Cambiar color de fondo	-	2	5
4	Cambiar material de la figura	-	1	5
5	Funciones de transferencia por defecto	-	5	2
6	Editar función de transferencia	-	7	3
7	Exportar función de transferencia	-	4	4
8	Importar función de transferencia	-	6	4
9	Generar y visualizar cortes	2	6	1
10	Editar plano de corte	9	3	2
11	Habilitar/Deshabilitar plano de corte	9	2	4
12	Posiciones del plano de corte por defecto	9	2	3
13	Guardar imágenes de las ventanas	-	2	2

#	Descripción	Dep.	Est.	Prio.
14	Realizar medida	-	5	2
15	Añadir regla	14	3	2
16	Eliminar regla	14	3	2
17	Habilitar/Deshabilitar regla	14	2	4
18	Eliminar partes	2	8	2
19	Generar malla	-	6	3
20	Mallas de materiales por defecto	19	1	4
21	Exportar malla	19	3	3
22	Mostrar valor escalar de un pixel en HU	9	4	4

Cuadro 4.1: Historias de usuario

4.1.2. Tarjetas de las historias de usuario

A continuación se incluye una descripción completa de las historias de usuario incluyendo una descripción de ésta y sus correspondientes criterios de aceptación.

1	Cargar datos DICOM
Descripción	
Se debe dotar al software de funcionalidad para cargar datos DICOM de un directorio. Para ello el usuario explorará entre los directorios del sistema hasta encontrar aquel con los datos que se desea visualizar.	
Estimación	2
Prioridad	1
Dependencias	-
Pruebas de aceptación	
<ul style="list-style-type: none"> - No se selecciona ningún directorio y no importa los datos. - Se selecciona un directorio sin datos DICOM o mal formateados y se informa del fallo y no importa los datos. - Se selecciona un directorio con datos DICOM y se cargan correctamente. 	

Cuadro 4.2: Historia de usuario - Cargar datos DICOM

2	Generar reconstrucción 3D
Descripción	
A partir de un directorio con datos DICOM se almacenan en un volumen que se renderiza en 3D según una función de transferencia usando <i>ray casting</i> en una ventana.	
Estimación	6
Prioridad	1
Dependencias	-
Pruebas de aceptación	
- A partir de unos datos DICOM y una función de transferencia se genera y se visualiza correctamente el volumen.	

Cuadro 4.3: Historia de usuario - Generar reconstrucción 3D

3	Cambiar color de fondo
Descripción	
El usuario podrá cambiar el color de fondo de cada uno de las ventanas en cualquiera de sus modos. Para ello elegirá un color a partir de un selector con colores predeterminados y funciones para crear el color o mediante RGB o mediante HSV.	
Estimación	2
Prioridad	5
Dependencias	-
Pruebas de aceptación	
<ul style="list-style-type: none"> - El usuario no puede introducir un color incorrecto. - Si el usuario no selecciona ningún color no se realiza ninguna acción. - Cuando el usuario cambia los colores de las ventanas, estas se actualizan con su nuevo color de fondo. 	

Cuadro 4.4: Historia de usuario - Cambiar color de fondo

4	Cambiar material de la figura
Descripción	
El usuario podrá cambiar el material con el que se visualiza la figura cambiando sus componentes ambiental, difuso, especular y potencia espe- cular introduciendo un valor entre 0 a 1 en los tres primeros y uno entre 1 y 50 en el último.	
Estimación	1
Prioridad	5
Dependencias	-
Pruebas de aceptación	
<ul style="list-style-type: none"> - El usuario no puede introducir un valor en ninguna de las componentes fuera del rango de valores. - Cuando se cambia el material la figura pasa a visualizarse con este. 	

Cuadro 4.5: Historia de usuario - Cambiar material de la figura

5	Funciones de transferencia por defecto
Descripción	
Se podrá cambiar la función de transferencia con la que visualizar el volumen eligiendo entre varios presets: uno para tener una vista completa con todos los materiales de la figura, otro en el que solo se muestre la madera, otro en el que solo se muestre el estuco y otro en el que se muestren los clavos y una capa casi transparente de madera para tener una referencia de la figura.	
Estimación	5
Prioridad	2
Dependencias	-
Pruebas de aceptación	
<ul style="list-style-type: none"> - Si el usuario cambia la función de transferencia, se actualiza la visualización del volumen usando esta. 	

Cuadro 4.6: Historia de usuario - Funciones de transferencia por defecto

6	Editar función de transferencia						
Descripción							
El usuario podrá editar la función de transferencia, para ello se le proporcionará un gráfico con cada una de las partes (color, opacidad escalar y gradiente) que podrá modificar añadiendo, quitando y moviendo puntos. Para agregar o cambiar un punto de color se facilitará un selector con colores predeterminados y funciones para crear nuevos mediante RGB o HSV.							
<table border="1"> <tr> <td>Estimación</td> <td>7</td> </tr> <tr> <td>Prioridad</td> <td>3</td> </tr> <tr> <td>Dependencias</td> <td>-</td> </tr> </table>		Estimación	7	Prioridad	3	Dependencias	-
Estimación	7						
Prioridad	3						
Dependencias	-						
Pruebas de aceptación							
<ul style="list-style-type: none"> - Si el usuario intenta borrar un punto cuando solo quedan dos, no se le dejará. - Si el usuario crea un punto en la función de color y no selecciona ningún color se le asigna el color que por interpolación le correspondía antes de que hubiese un punto en esa posición. - Si se edita cualquiera de las partes de la función de transferencia se cambia esta y, por tanto, la visualización del volumen. 							

Cuadro 4.7: Historia de usuario - Editar función de transferencia

7	Exportar función de transferencia						
Descripción							
Se podrá exportar la función de transferencia actual a un archivo XML con un formato específico en el que luego se pueda importar.							
<table border="1"> <tr> <td>Estimación</td> <td>4</td> </tr> <tr> <td>Prioridad</td> <td>4</td> </tr> <tr> <td>Dependencias</td> <td>-</td> </tr> </table>		Estimación	4	Prioridad	4	Dependencias	-
Estimación	4						
Prioridad	4						
Dependencias	-						
Pruebas de aceptación							
<ul style="list-style-type: none"> - Si el usuario guarda sin escribir la extensión, comprobar que se añade automáticamente. - Si el usuario guarda con un nombre, comprobar que efectivamente ocurre así. 							

Cuadro 4.8: Historia de usuario - Exportar función de transferencia

8	Importar función de transferencia
Descripción	
El usuario podrá importar una función de transferencia almacenada en un archivo XML con un formato específico.	
Estimación	6
Prioridad	4
Dependencias	-
Pruebas de aceptación	
<ul style="list-style-type: none"> - Si el usuario carga un archivo XML con un formato distinto al usado para exportar funciones de transferencia, se informará del error. - Si el usuario introduce un archivo correcto, se carga la función de transferencia. 	

Cuadro 4.9: Historia de usuario - Importar función de transferencia

9	Generar y visualizar cortes
Descripción	
A partir de la reconstrucción 3D del volumen y un plano que se podrá mover y girar arbitrariamente, se visualizará el corte que produce este plano con el volumen en una ventana distinta a la utilizada para visualizarlo en 3D.	
Estimación	6
Prioridad	1
Dependencias	2
Pruebas de aceptación	
<ul style="list-style-type: none"> - Cuando no hay ningún volumen cargado no se visualiza nada en la ventana de cortes. - Cuando el plano interseca con el volumen se visualiza correctamente el corte. 	

Cuadro 4.10: Historia de usuario - Generar y visualizar cortes

10	Editar plano de corte
Descripción	
A partir del plano de corte en cualquier posición, se podrá girar en cualquiera de los ejes y mover a través de la dirección de su normal.	
Estimación	3
Prioridad	2
Dependencias	9
Pruebas de aceptación	
<ul style="list-style-type: none"> - Comprobar que se puede girar el plano sobre cualquiera de sus ejes con el centro de éste como punto pivote. - Comprobar que solo se puede mover el plano a través de la dirección de su normal. - Comprobar que no se puede mover el plano más allá de la figura. - Cuando se realiza la transformación en el plano, se actualiza la imagen del corte que produce en la figura. 	

Cuadro 4.11: Historia de usuario - Editar plano de corte

11	Habilitar y deshabilitar el plano de corte
Descripción	
El usuario podrá habilitar y deshabilitar el plano de corte para visualizarlo o no en el visor 3D.	
Estimación	2
Prioridad	4
Dependencias	9
Pruebas de aceptación	
<ul style="list-style-type: none"> - Cuando el plano está habilitado y se ejecuta la acción, este se deshabilita. - Cuando el plano está deshabilitado y se ejecuta la acción, este se habilita. - Cuando el plano está deshabilitado no se puede transformar y si se cambia la posición por defecto, no se actualiza el corte hasta que se vuelve a habilitar. 	

Cuadro 4.12: Historia de usuario - Habilitar y deshabilitar el plano de corte

12	Posiciones del plano de corte por defecto
Descripción	
Se puede colocar el plano directamente en el centro del volumen en posiciones axial, sagital y coronal.	
Estimación	2
Prioridad	3
Dependencias	5
Pruebas de aceptación	
<ul style="list-style-type: none"> - Comprobar que cuando no hay ningún volumen no se puede cambiar la posición del plano e informar al usuario de ello. - El plano se coloca en la posición deseada cuando se realiza la acción. 	

Cuadro 4.13: Historia de usuario - Posiciones del plano de corte por defecto

13	Guardar imágenes de las ventanas
Descripción	
Se podrán guardar imágenes de lo que se visualiza en las ventanas tanto en formato JPG como PNG. Para ello el usuario elegirá dónde almacenar la imagen generada. Por defecto el nombre para la imagen corresponderá con la fecha como cadena de números en formato “AAAAMMDDHHMMSS”.	
Estimación	2
Prioridad	2
Dependencias	-
Pruebas de aceptación	
<ul style="list-style-type: none"> - Si el usuario guarda sin escribir la extensión, comprobar que se añade automáticamente. - Si el usuario guarda con un nombre y una extensión, comprobar que efectivamente ocurre así. 	

Cuadro 4.14: Historia de usuario - Guardar imágenes de las ventanas

14	Realizar medida
Descripción	
El usuario podrá realizar una medida de un punto a otro en cualquiera de las dos ventanas con una regla, para ello tendrá que pinchar en el punto inicial y el final. Una vez realizada puede cambiarla seleccionando el punto inicial o final y arrastrándolo a la nueva posición.	
Estimación	5
Prioridad	2
Dependencias	-
Pruebas de aceptación	
<ul style="list-style-type: none"> - Si el usuario introduce dos puntos se visualiza una línea de uno a otro con la medida correspondiente. - Si el usuario cambia de posición la regla, se actualiza el valor de cuánto mide ésta. 	

Cuadro 4.15: Historia de usuario - Realizar medida

15	Añadir regla
Descripción	
El usuario podrá añadir reglas para realizar medidas en cualquiera de las dos ventanas.	
Estimación	3
Prioridad	2
Dependencias	14
Pruebas de aceptación	
<ul style="list-style-type: none"> - Si el usuario añade una regla en una ventana, se puede realizar la medida en esta. - Si el usuario intenta añadir más reglas que las establecidas por un límite, se le informa y no se añade. 	

Cuadro 4.16: Historia de usuario - Añadir regla

16	Eliminar regla
Descripción	
El usuario podrá eliminar cualquier regla creada con anterioridad.	
Estimación	3
Prioridad	2
Dependencias	14
Pruebas de aceptación	
<ul style="list-style-type: none"> - Si el usuario elimina una regla, se elimina correctamente y deja de visualizarse. - Si el usuario intenta eliminar cuando no hay ninguna regla, se le informa de esto. 	

Cuadro 4.17: Historia de usuario - Eliminar regla

17	Habilitar y deshabilitar regla
Descripción	
El usuario podrá habilitar o deshabilitar cualquier regla creada con anterioridad para mostrarla o no sin llegar a borrarla.	
Estimación	2
Prioridad	4
Dependencias	14
Pruebas de aceptación	
<ul style="list-style-type: none"> - Si el usuario deshabilita una regla habilitada, deja de mostrarse. - Si el usuario habilita una regla deshabilitada, se vuelve a mostrar. - Si el usuario intenta eliminar cuando no hay ninguna regla, se le informa de esto. 	

Cuadro 4.18: Historia de usuario - Habilitar y deshabilitar regla

18	Eliminar partes
Descripción	
El usuario podrá eliminar una parte del volumen separada de las demás seleccionando un punto de esta. Antes de confirmar el borrado se le pregunta al usuario por si ha borrado una parte que no deseaba borrar y poder volver al estado anterior.	
Estimación	8
Prioridad	2
Dependencias	-
Pruebas de aceptación	
<ul style="list-style-type: none"> - Si no se selecciona ningún punto no se realiza ninguna acción. - Si se cancela el borrado se vuelve al estado original. - Si el usuario selecciona un punto y confirma se elimina la parte seleccionada. 	

Cuadro 4.19: Historia de usuario - Eliminar partes

19	Generar malla
Descripción	
El usuario podrá generar una malla de triángulos a partir del volumen dado un valor de isosuperficie mediante el uso de <i>Marching cubes</i> .	
Estimación	6
Prioridad	3
Dependencias	-
Pruebas de aceptación	
<ul style="list-style-type: none"> - El usuario solo puede dar un valor de isosuperficie que se encuentre entre los rangos de valores de una imagen DICOM. - Dado un valor de isosuperficie se genera la malla de triángulos del volumen correctamente. 	

Cuadro 4.20: Historia de usuario - Generar malla

20	Mallas de materiales por defecto
Descripción	
Se facilitarán al usuario los valores de isosuperficie para extraer la madera (que incluye el estuco y los clavos), el estuco (que incluye los clavos) y los clavos.	
Estimación	1
Prioridad	4
Dependencias	19
Pruebas de aceptación	
- Si es usuario cambia el material, se actualiza la malla.	

Cuadro 4.21: Historia de usuario - Mallas de materiales por defecto

21	Exportar malla
Descripción	
A partir de una malla de triángulos generada, se puede exportar a un formato STL.	
Estimación	3
Prioridad	3
Dependencias	19
Pruebas de aceptación	
- Comprobar que se exporta correctamente la malla de triángulos generada.	

Cuadro 4.22: Historia de usuario - Exportar malla

22	Mostrar valor escalar de un pixel en HU
Descripción	
A partir de un corte generado, mostrar el valor escalar en HU del pixel seleccionado.	
Estimación	4
Prioridad	4
Dependencias	9
Pruebas de aceptación	
- Comprobar que cuando se selecciona fuera de la imagen informa al usuario de ello.	
- Comprobar que al colocar el ratón sobre un pixel de la imagen aparece el valor escalar de éste en HU.	

Cuadro 4.23: Historia de usuario - Mostrar valor escalar de un pixel en HU

Capítulo 5

Diseño

En este capítulo se mostrarán los diagramas UML **arquitectónico**, de **clases** y algunos de **secuencia** de la aplicación. Todos han sido generados con la aplicación *Visual Paradigm for UML 13.1 Community Edition* [7].

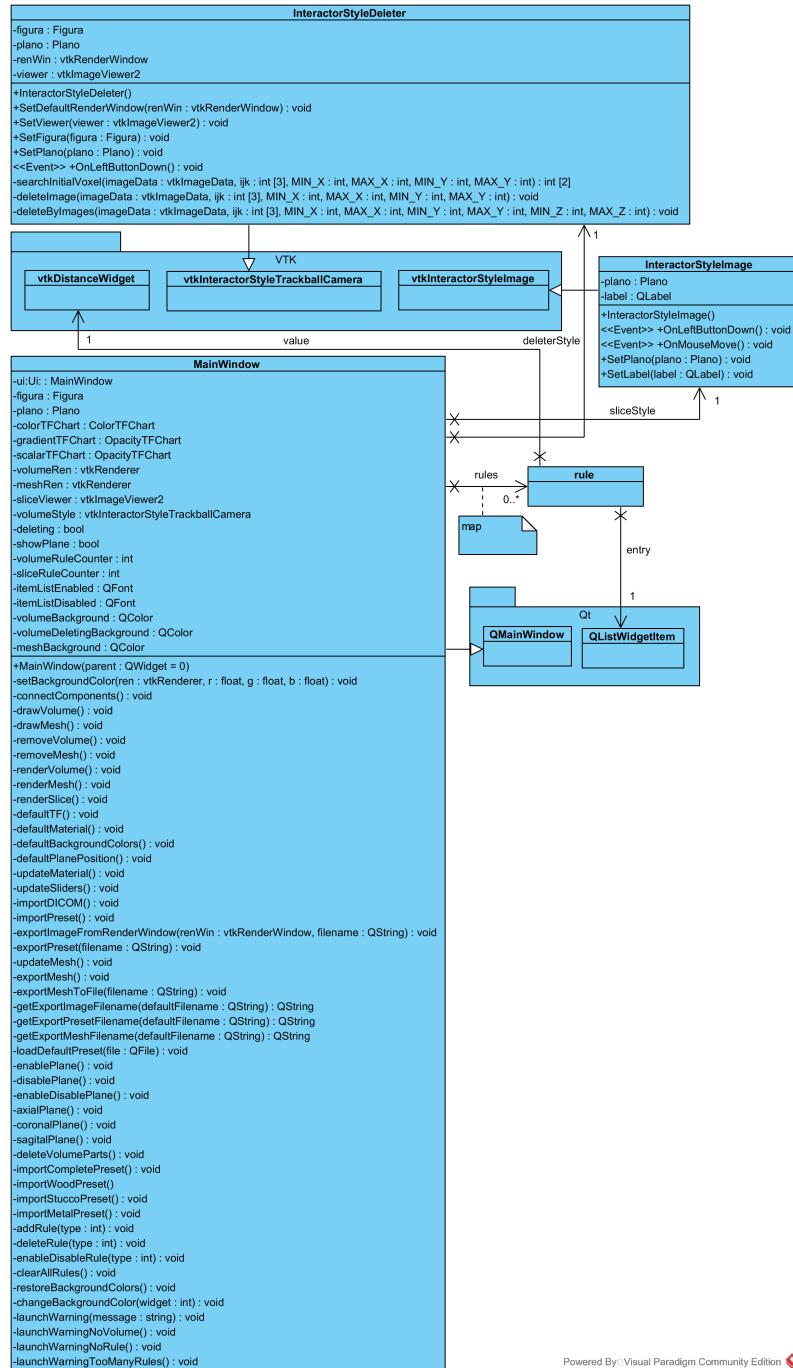
5.1. Diagrama arquitectónico

Aquí el diagrama arquitectónico

5.2. Diagramas de clases

Se presentan las distintas clases en tres diagramas distintos para ser mostrados con más claridad aunque no estén separados en paquetes como tales.

5.2.1. Application



Powered By: Visual Paradigm Community Edition

Figura 5.1: Diagrama de clases del paquete *Application*

5.2.2. Charts

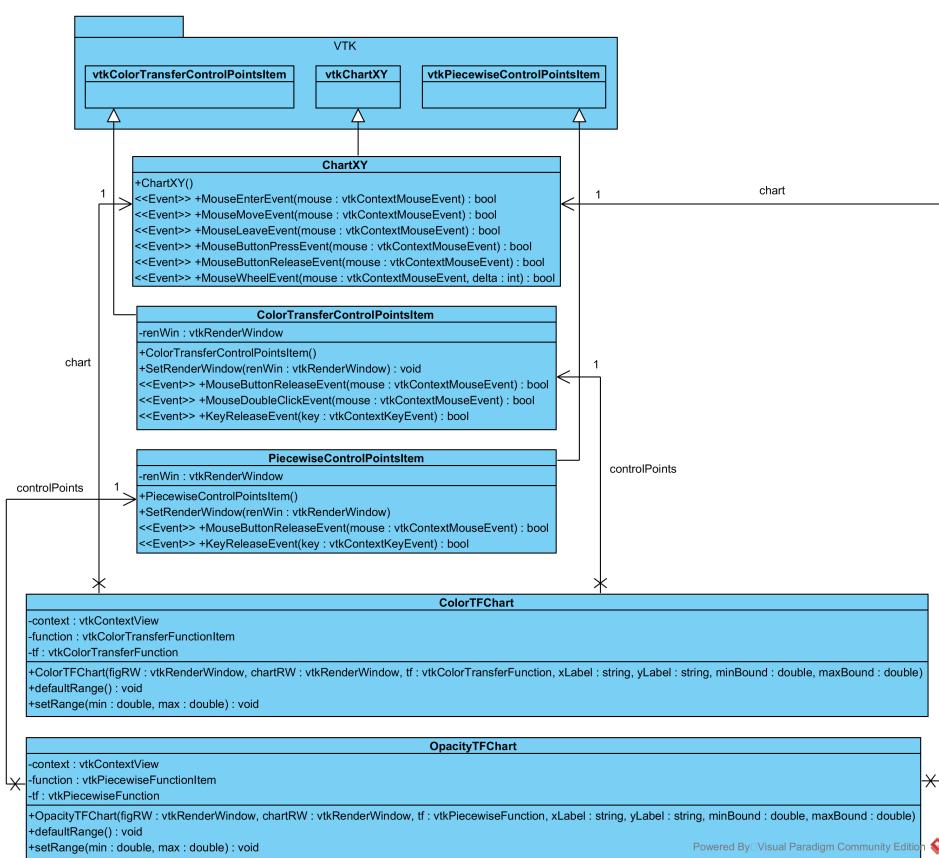


Figura 5.2: Diagrama de clases del paquete *Charts*

5.2.3. Volume

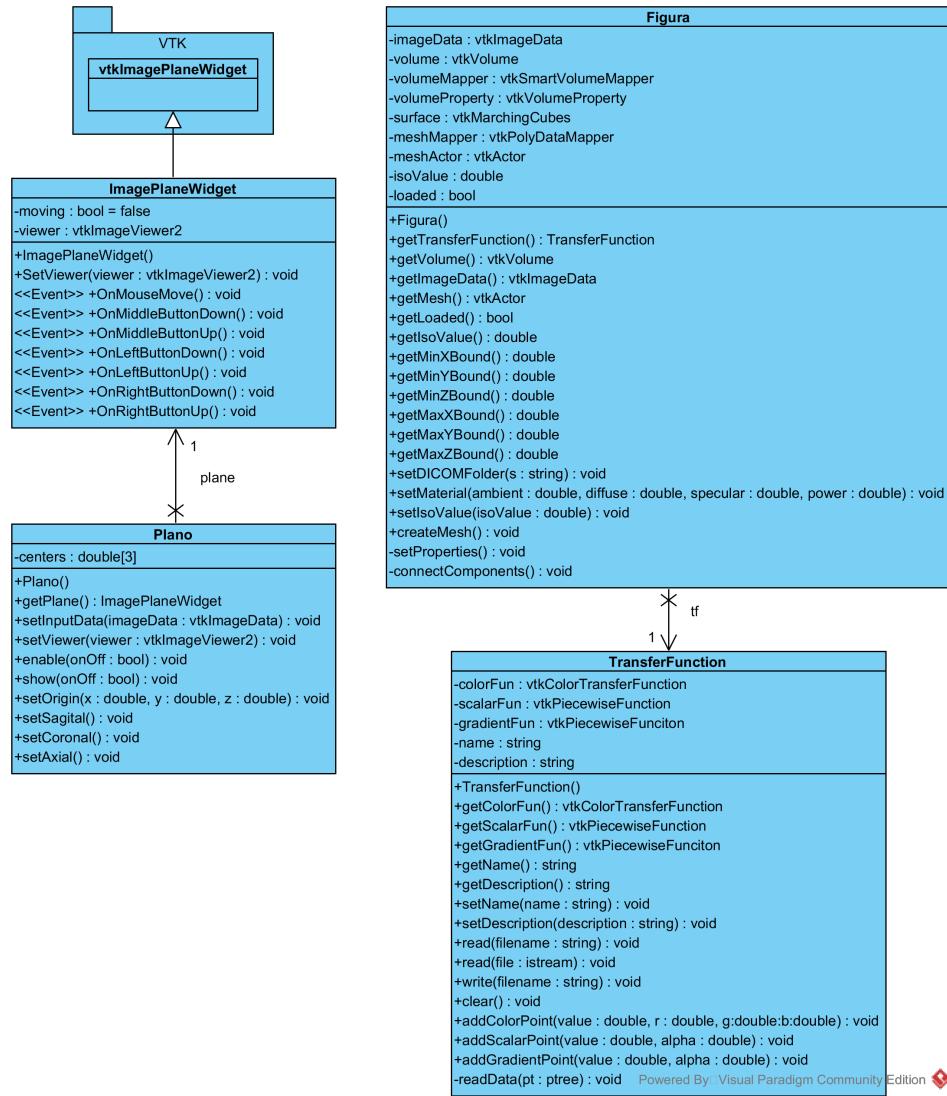


Figura 5.3: Diagrama de clases del paquete *Volume*

5.3. Diagramas de secuencia

A continuación se muestran algunos de los diagramas de secuencia de la aplicación: Aquellos que se han considerado más importantes.

5.3.1. OpacityTFChart

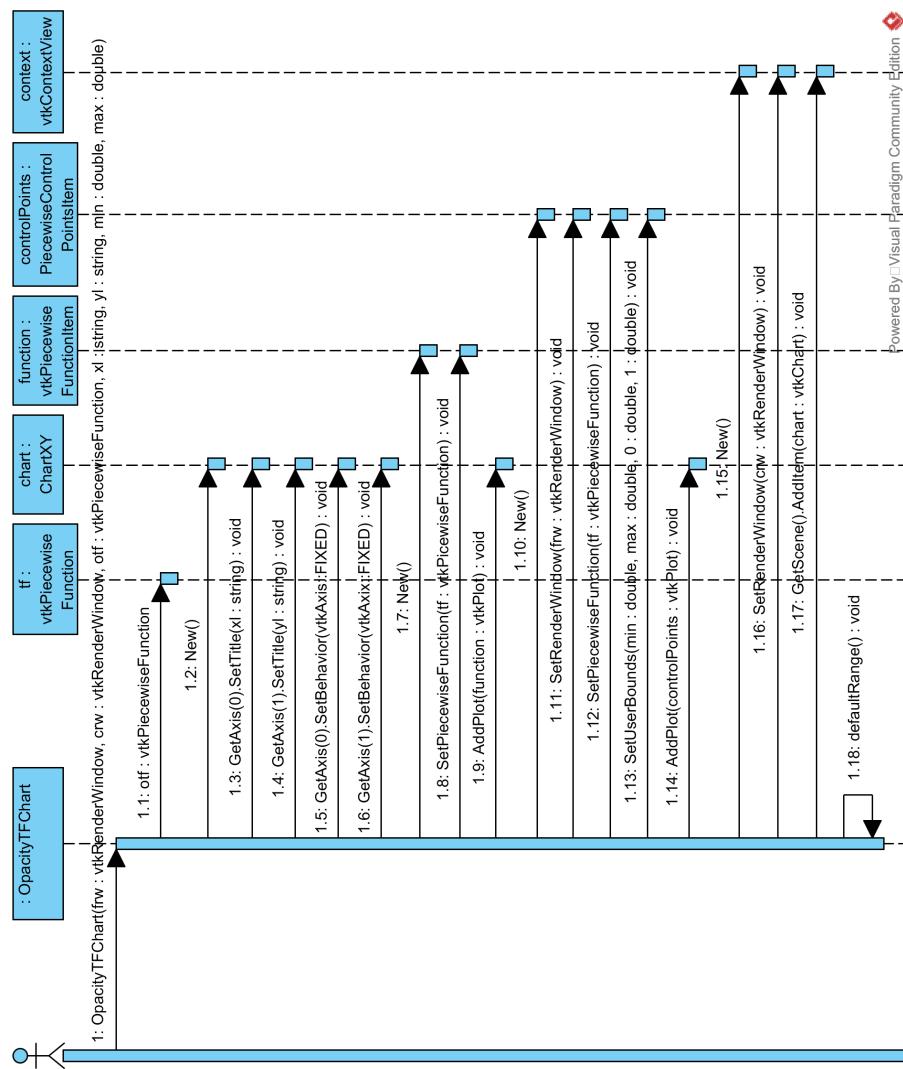


Figura 5.4: Diagrama de secuencia del constructor de *OpacityTFChart*

5.3.2. ColorTFChart

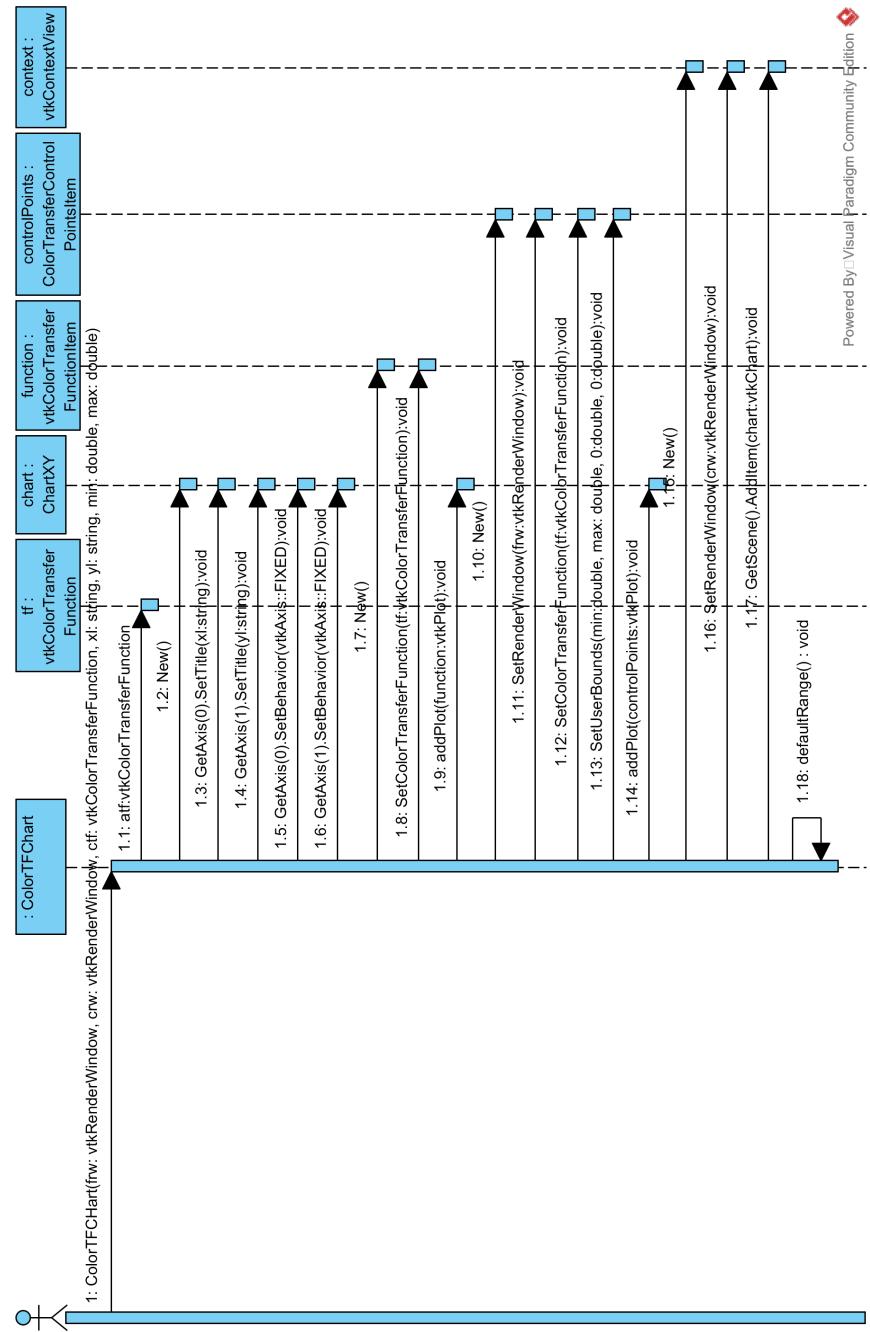


Figura 5.5: Diagrama de secuencia del constructor de `ColorTFChart`

5.3.3. ColorTransferControlPointsItem

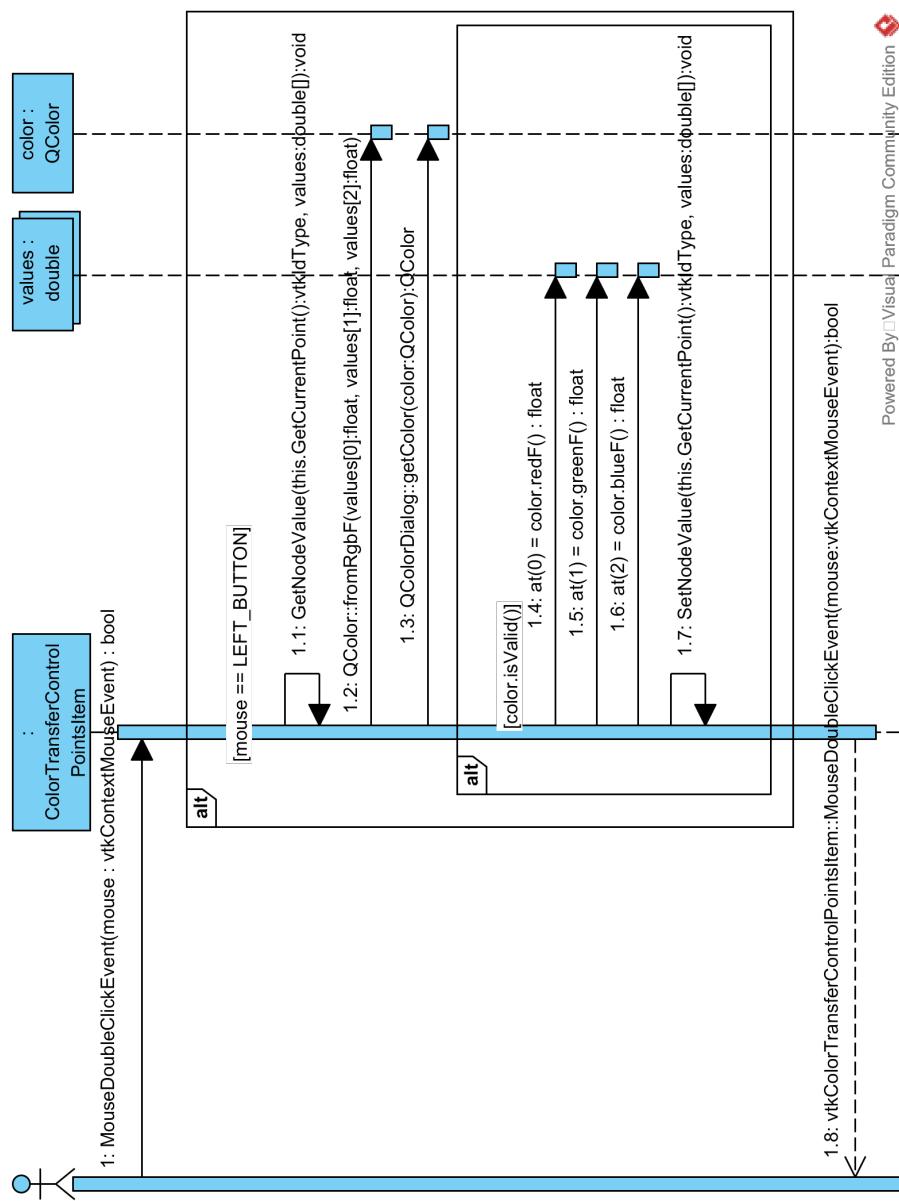
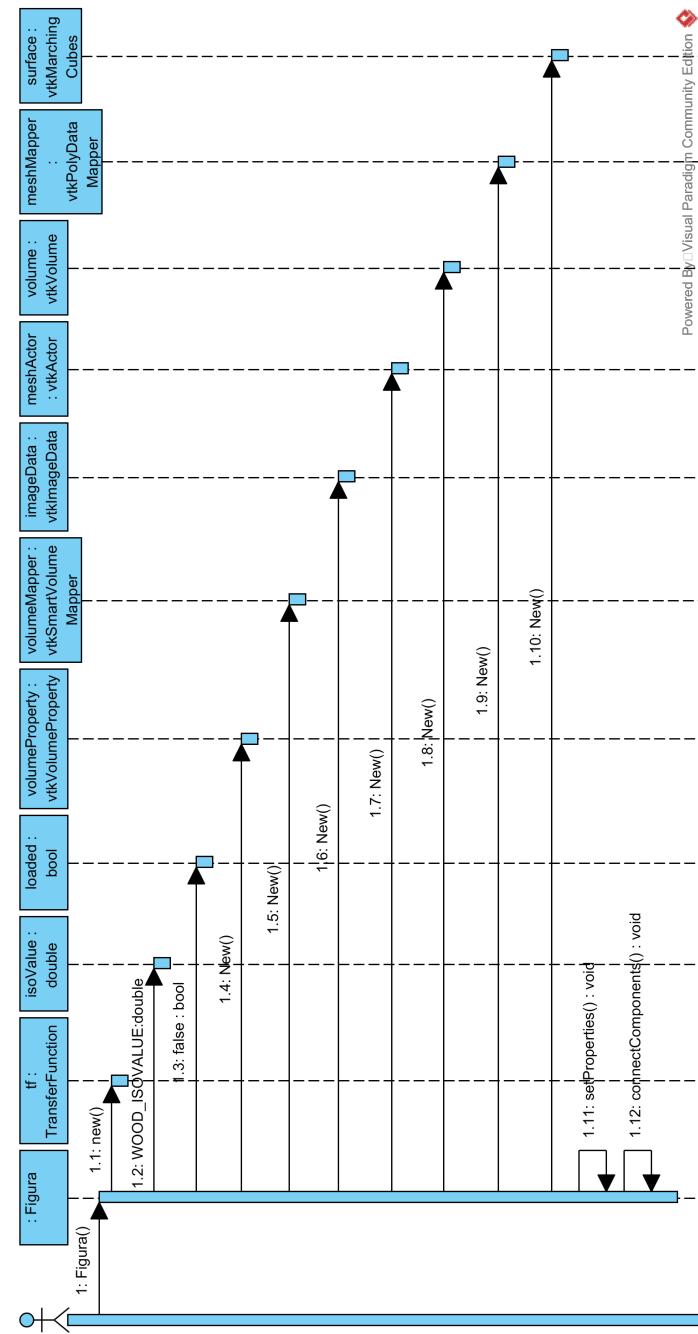


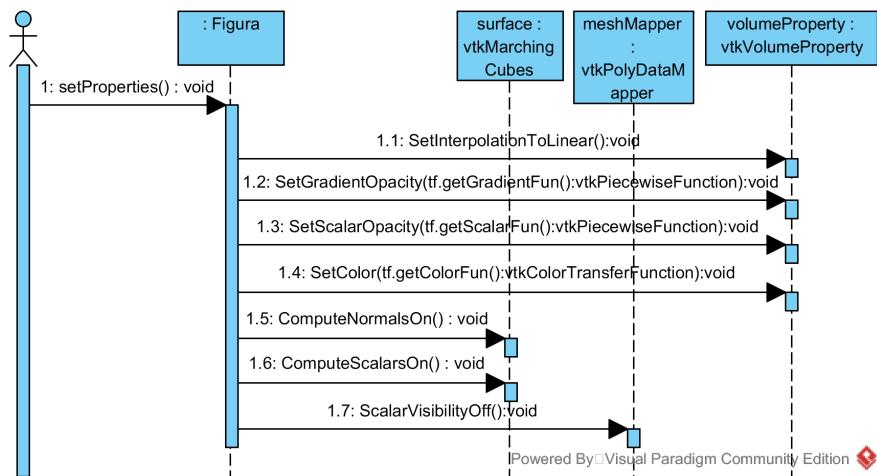
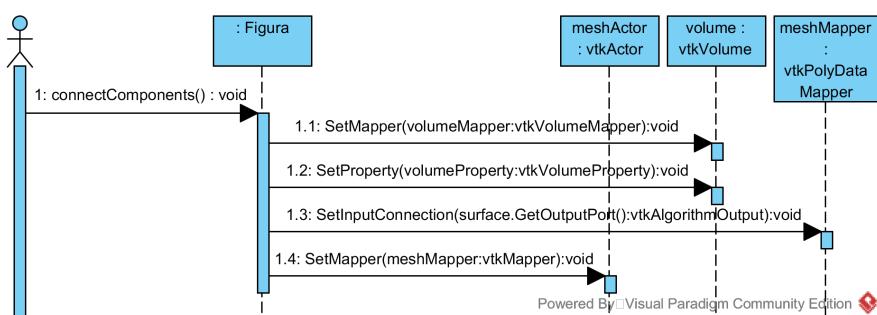
Figura 5.6: Diagrama de secuencia del evento de doble click de `ColorTransferControlPointsItem`



Powered By Visual Paradigm Community Edition

5.3.4. Figura

Figura 5.7: Diagrama de secuencia del constructor de *Figura*

Figura 5.8: Diagrama de secuencia del método `setProperties` de `Figura`Figura 5.9: Diagrama de secuencia del método `connectComponents` de `Figura`

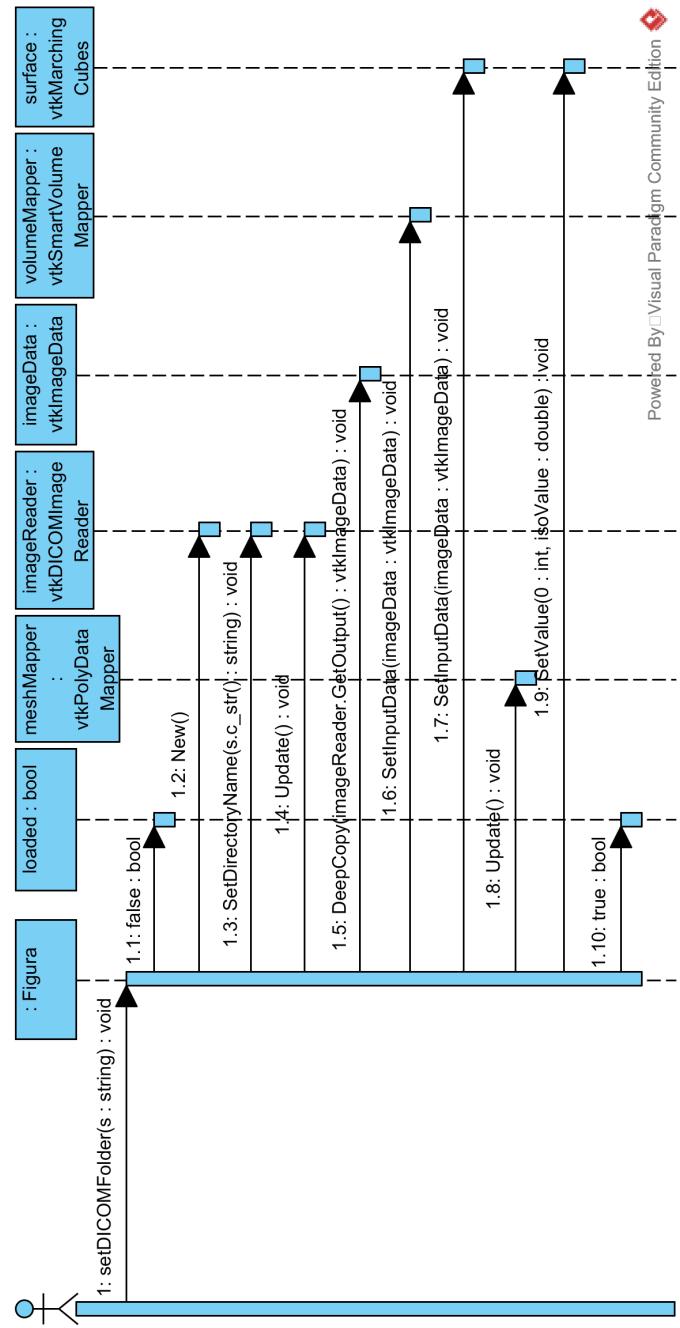


Figura 5.10: Diagrama de secuencia del método `setDICOMFolder` de `Figura`

5.3.5. TransferFunction

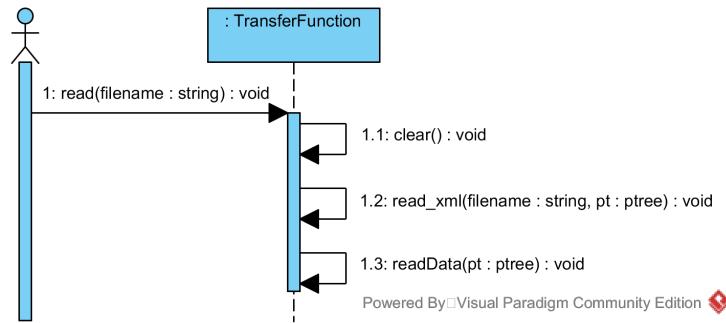


Figura 5.11: Diagrama de secuencia del método `read` de `TransferFunction`

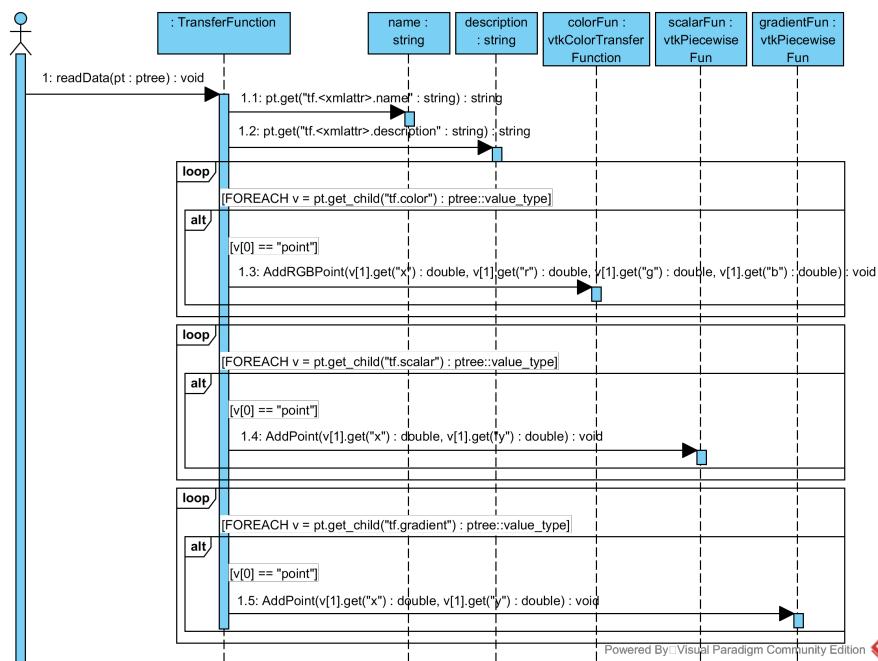


Figura 5.12: Diagrama de secuencia del método `readData` de `TransferFunction`

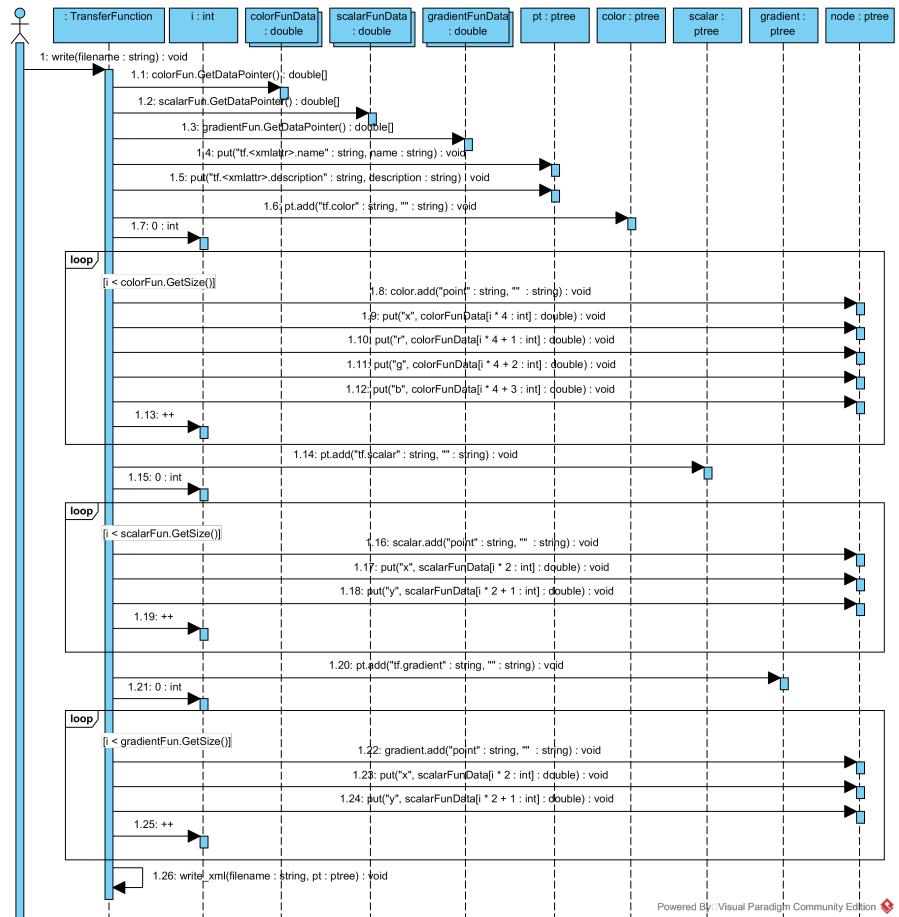


Figura 5.13: Diagrama de secuencia del método `write` de `TransferFunction`

5.3.6. MainWindow

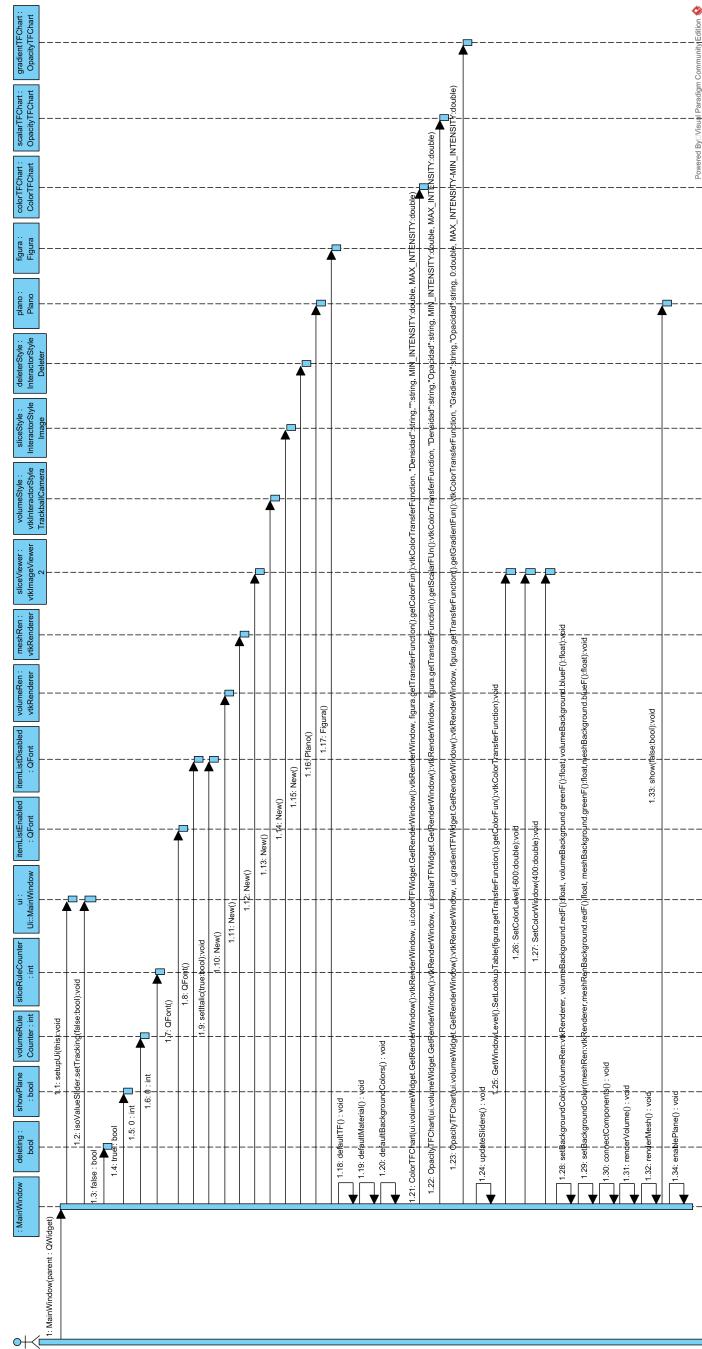


Figura 5.14: Diagrama de secuencia del constructor de *MainWindow*

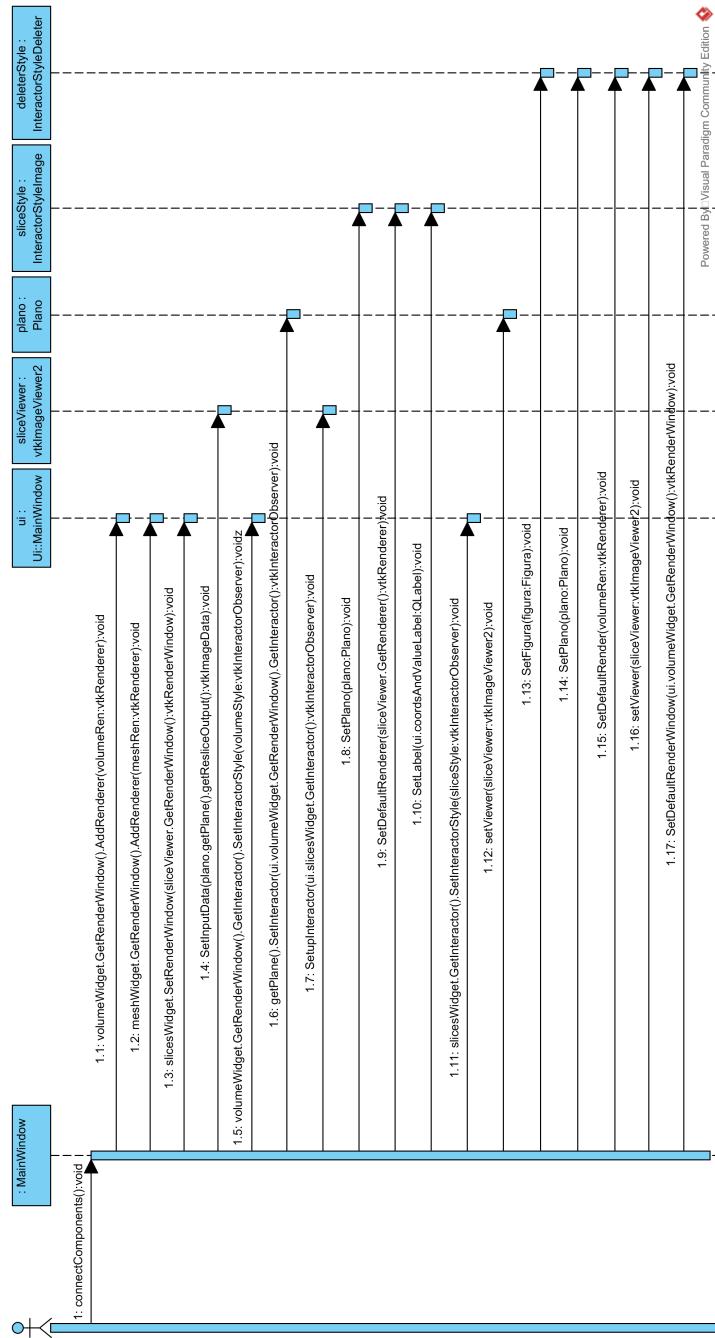


Figura 5.15: Diagrama de secuencia del método `connectComponents` de `Main Window`

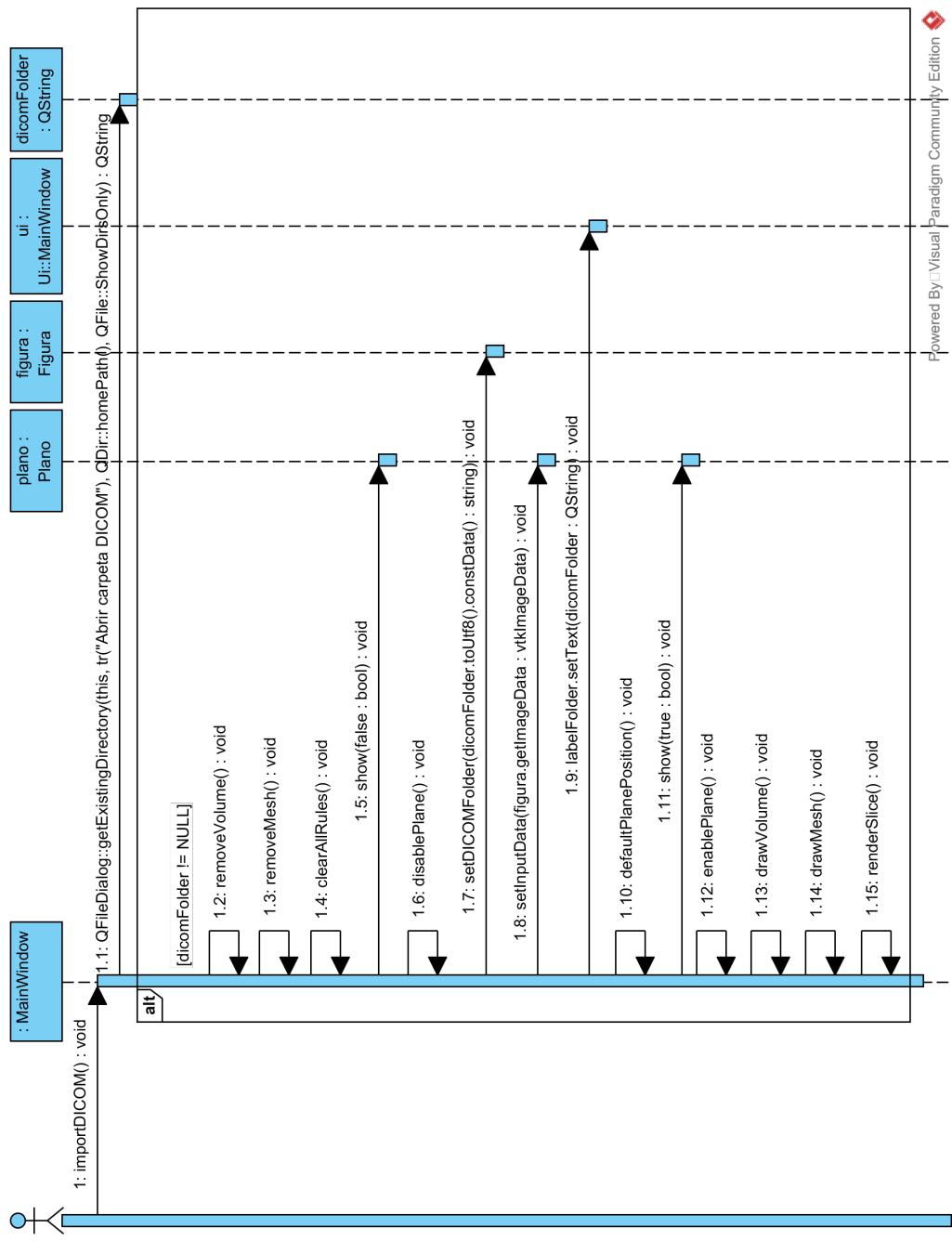


Figura 5.16: Diagrama de secuencia del método `importDICOM` de `MainWindow`

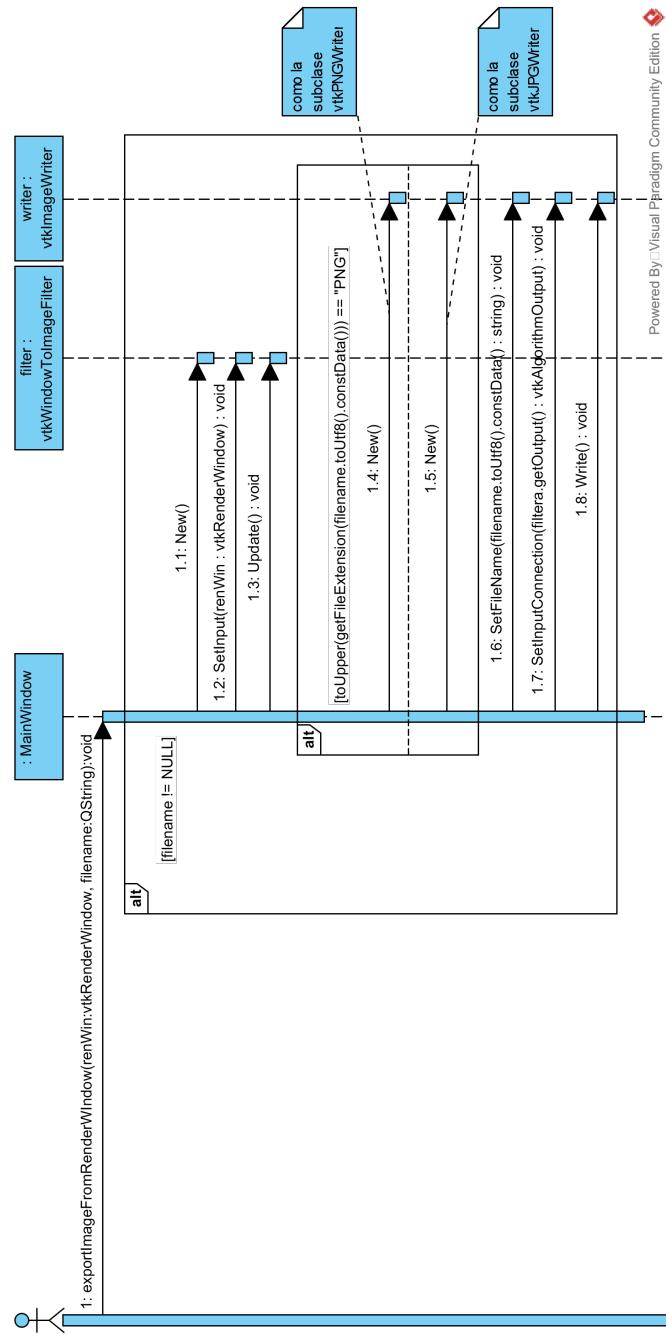


Figura 5.17: Diagrama de secuencia del método `exportImageFromRenderWindow` de `Main Window`

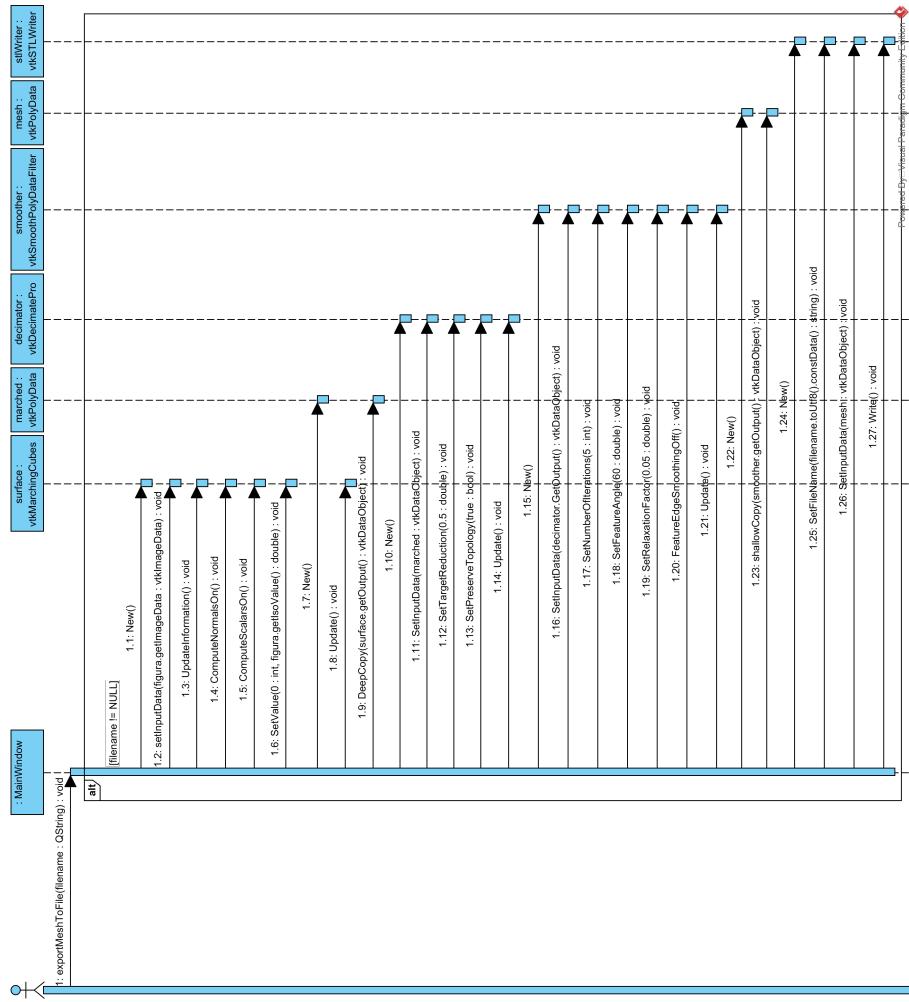


Figura 5.18: Diagrama de secuencia del método *exportMeshToFile* de *Main-Window*

5.3. Diagramas de secuencia

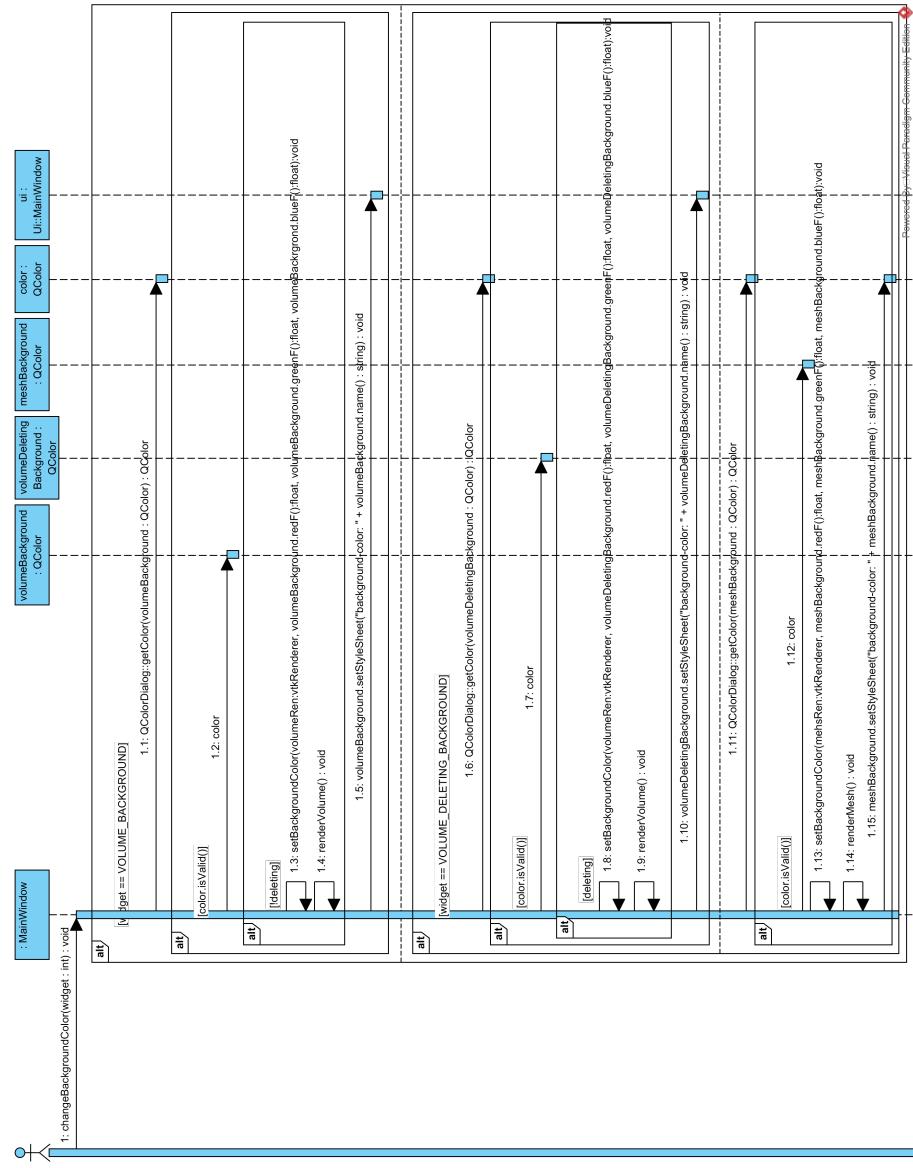


Figura 5.19: Diagrama de secuencia del método `changeBackgroundColor` de `MainWindow`

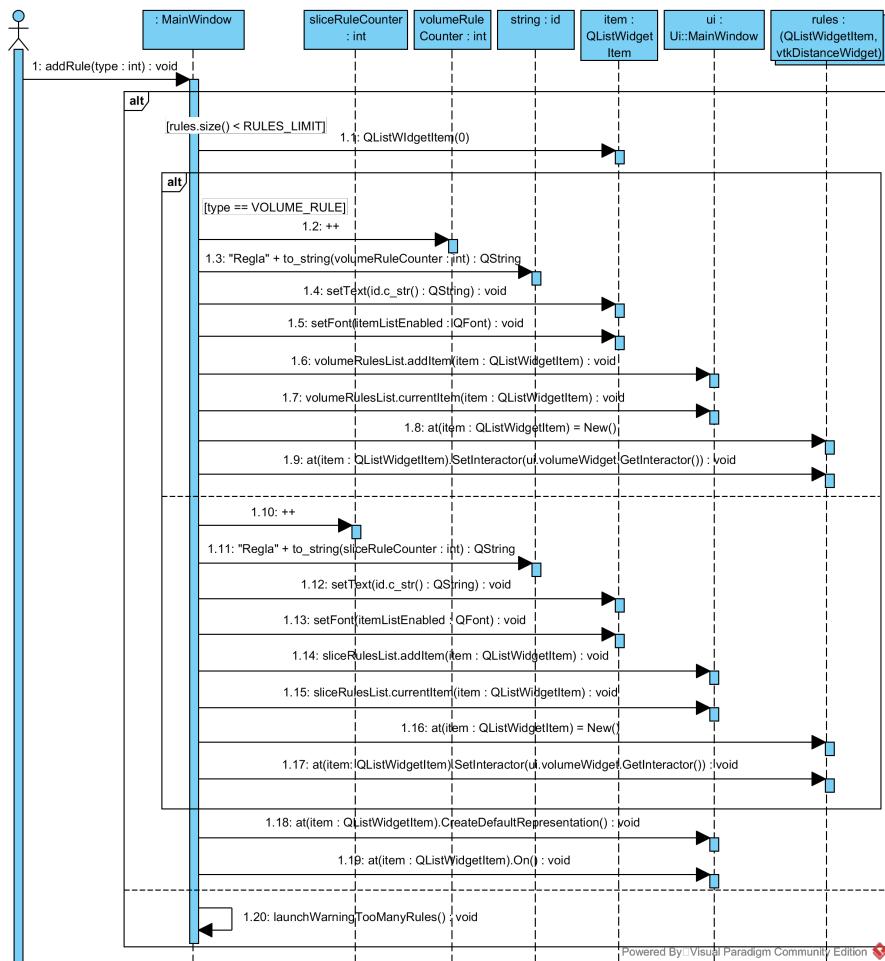


Figura 5.20: Diagrama de secuencia del método `addRule` de `MainWindow`

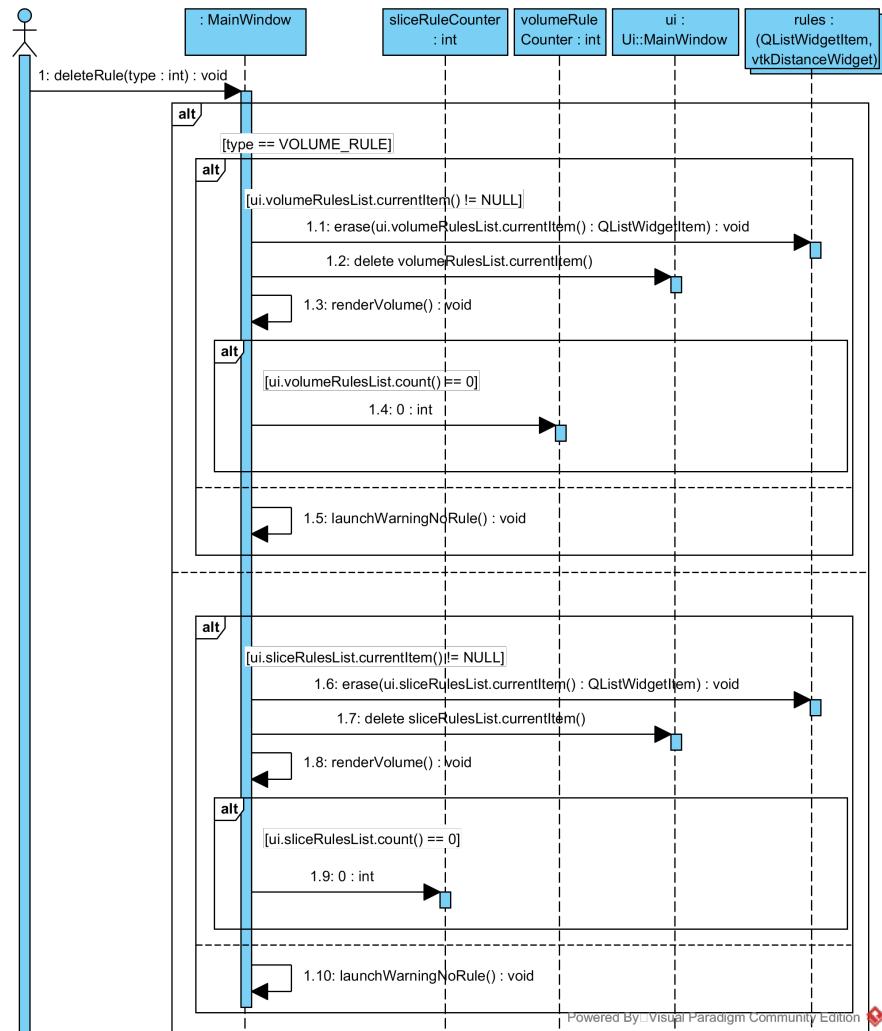


Figura 5.21: Diagrama de secuencia del método `deleteRule` de `MainWindow`

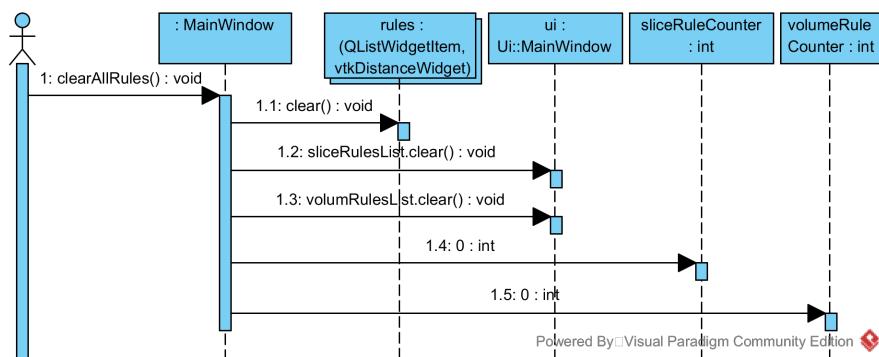


Figura 5.22: Diagrama de secuencia del método `clearAllRules` de `MainWindow`

5.3.7. InteractorStyleDelete

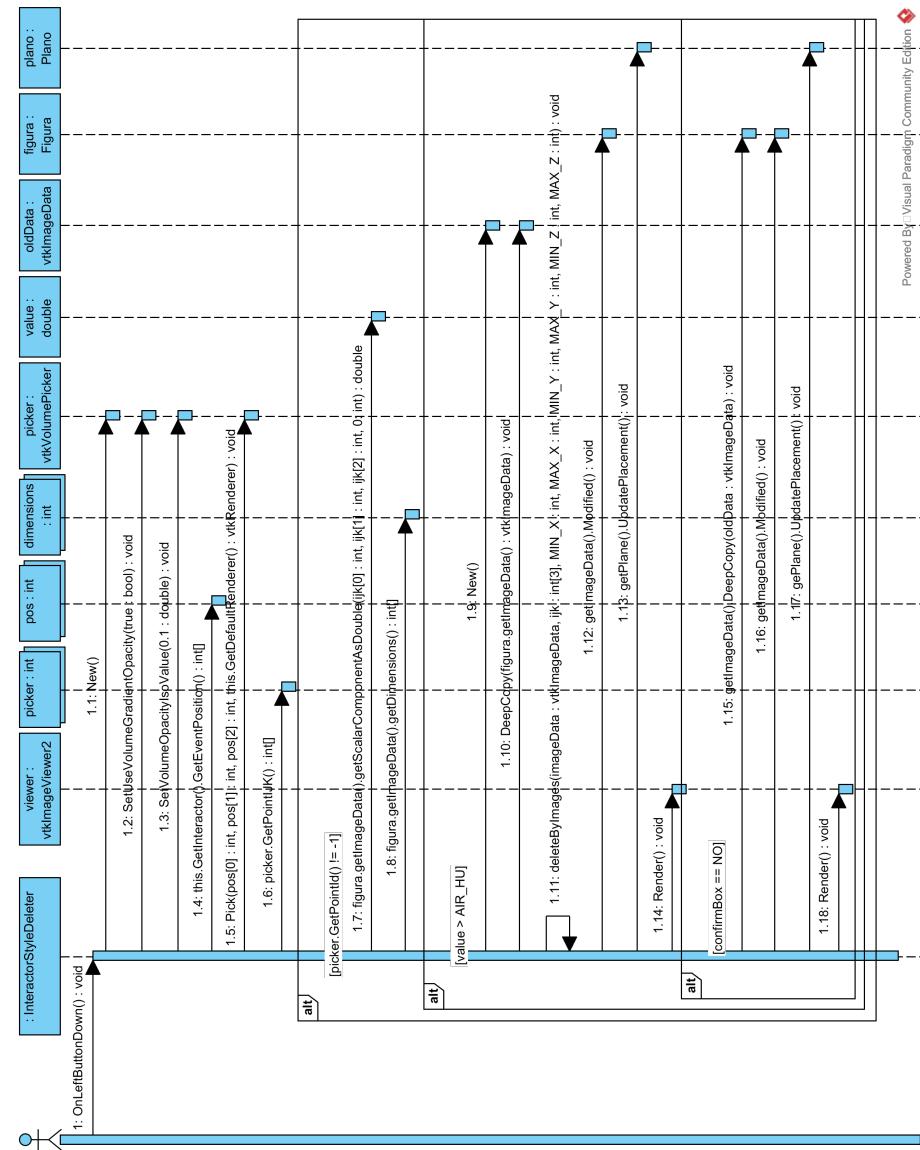
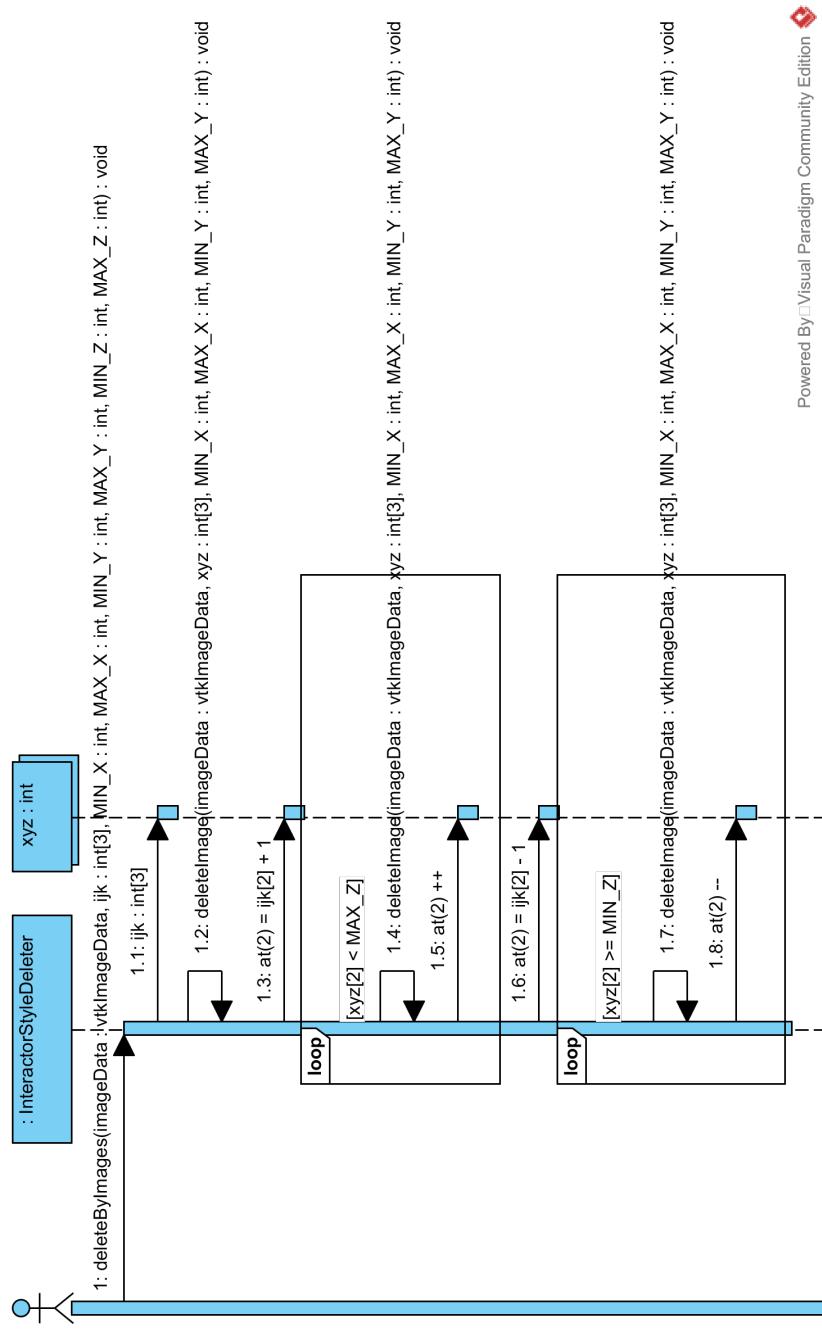


Figura 5.23: Diagrama de secuencia del método `OnLeftButtonDown` de `InteractorStyleDelete`



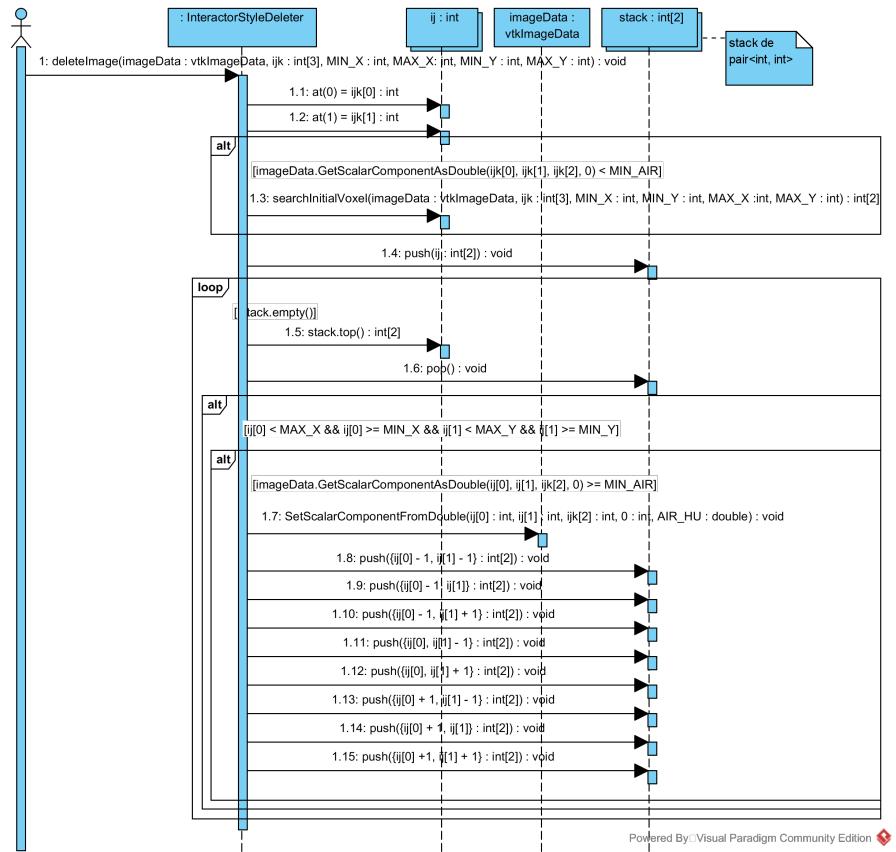


Figura 5.25: Diagrama de secuencia del método *deleteImage* de *InteractorStyleDeleteer*

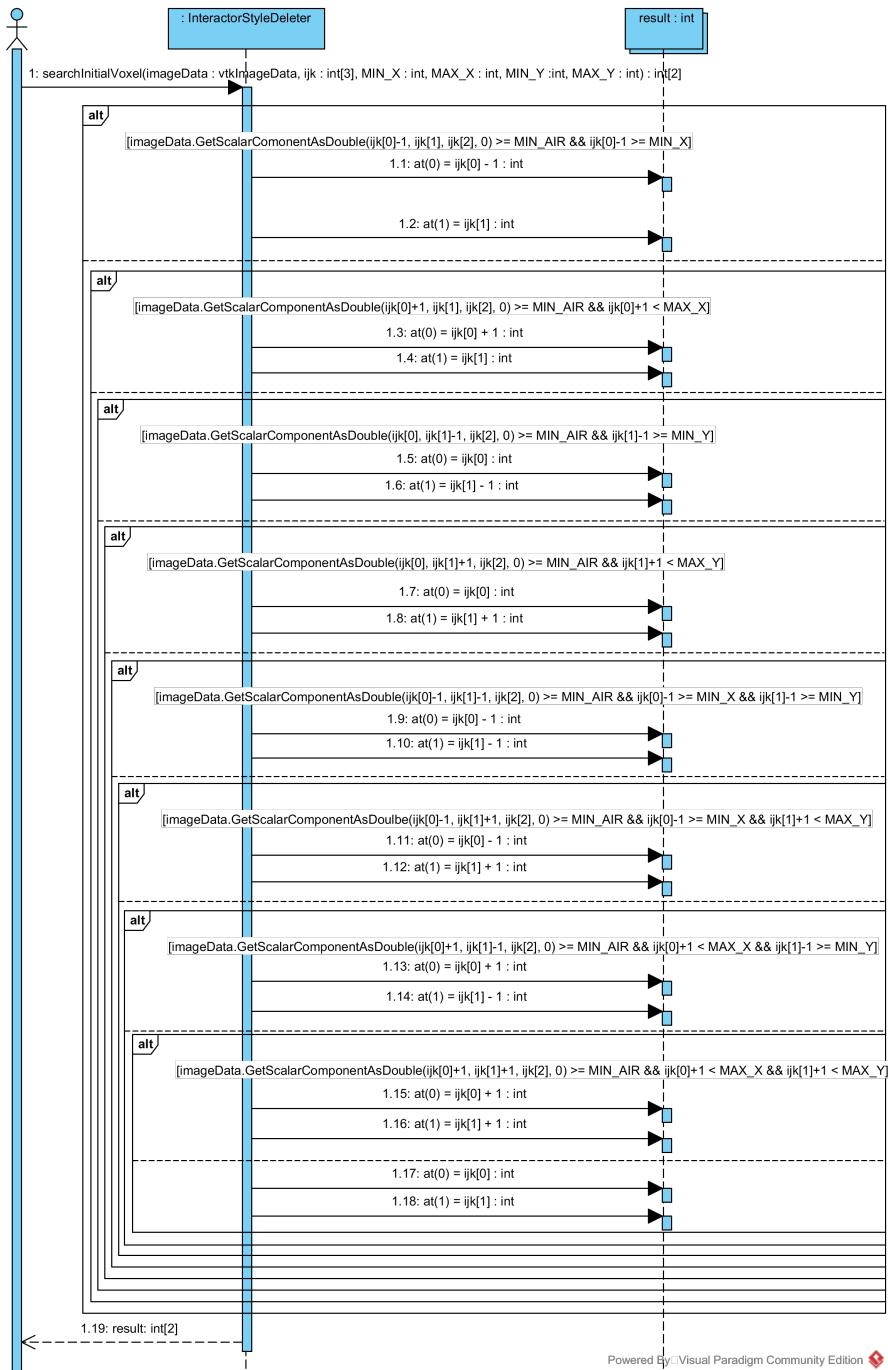


Figura 5.26: Diagrama de secuencia del método `searchInitialVoxel` de `InteractorStyleDelete`

5.3.8. InteractorStyleImage

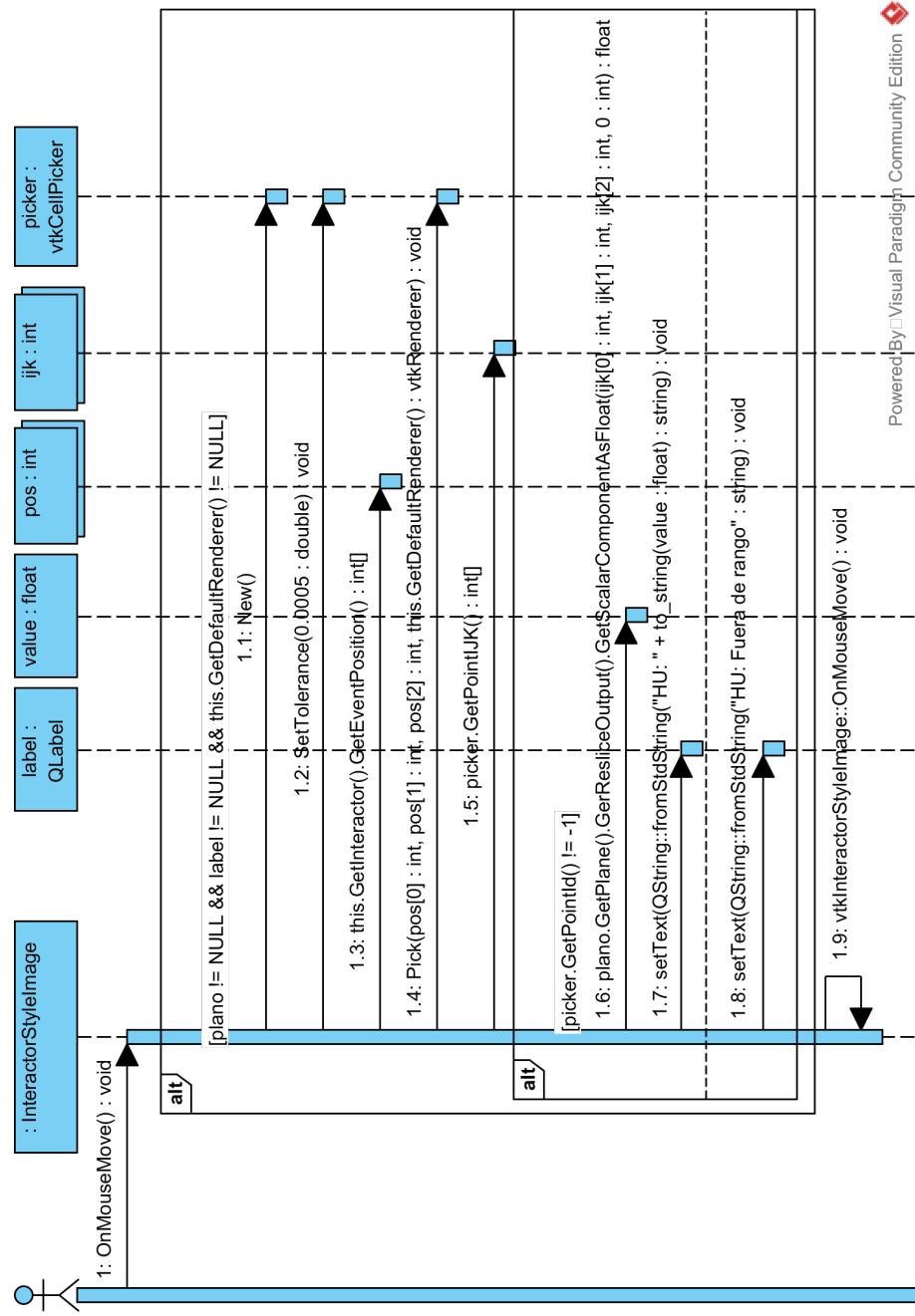


Figura 5.27: Diagrama de secuencia del método `OnMouseMove` de `InteractorStyleImage`

Capítulo 6

Implementación

En este capítulo se hablará del desarrollo del software describiendo la plataforma de desarrollo y despliegue, así como las distintas decisiones tomadas a la hora de implementar y las fases de desarrollo.

6.1. Plataforma de desarrollo

En un principio, la idea fue desarrollar usando una distribución Linux un **software multiplataforma** ayudándose, para ello, de CMake [3] que a partir de los archivos fuentes puede crear *makefiles* para distintas plataformas.

No obstante, unos problemas detectados con los *drivers* de la GPU provocaban que no se pudiese integrar Qt [6] con VTK [8] ya que el *widget* especializado para esta tarea **QVTKWidget** no funcionaba. Tras unos días intentando solucionar los problemas sin éxito se decidió migrar a Windows donde no surgió ningún problema parecido al que se dio en Linux.

Más tarde además de Qt y VTK se decidió utilizar la librería Boost [2] para facilitar el tratamiento de ficheros XML.

Se ha utilizado **Windows 10 Pro** (64 bits) con el siguiente software y librerías:

- CMake 3.4.1
- Visual Studio Community 2013
- Qt5.5.1
- VTK 7.0.0
- Boost 1.60.0

Cuando se empezó a desarrollar el software la versión más nueva de VTK era la **6.3.0** y se empezó utilizando ésta. No obstante, tenía un pequeño fallo que afectaba mucho a la aplicación y es que si se hacía uso de la GPU en el *ray-casting* al renderizar el volumen, **la opacidad gradiente no se computaba**. Por suerte, a principios del mes de Febrero, se lanzó una nueva versión de VTK, la **7.0.0**, que solucionaba este fallo.

Pero los problemas no acababan aquí, y es que, con la nueva versión, el programa no llegaba a funcionar y **se bloqueaba nada más iniciarse**. Detectar el fallo era complicado y al no ser todavía demasiado complejo era más rápido detectar de dónde venía el error si se volvía a crear desde cero agregando uno a uno cada componente.

El fallo lo estaba dando el plano de corte y es que en la versión anterior al activarlo, si no tenía ningún volumen con el que cortar, no producía ningún fallo, pero con la nueva versión era un requisito imprescindible. La solución era bastante sencilla: habilitar el plano cuando se cargase el volumen en lugar de tenerlo habilitado desde su propia construcción.

6.2. Instalación y configuración

6.2.1. Entorno de desarrollo

Visual Studio Community 2013

- Descargar Visual Studio Community 2013 desde su web oficial e instalar.

Qt5.5.1

- Descargar Qt5.5.1 desde este enlace de su web e instalar.
- Crear una nueva variable de entorno con nombre **QTDIR** y valor **C:\Qt\Qt5.5.1** (directorio raíz de la versión instalada).
- Agregar al PATH la siguiente dirección **C:\Qt\Qt5.5.1\5.5\msvc2013\bin**.

CMake 3.4.1

- Descargar CMake 3.4.1 desde este enlace de su web e instalar (al instalar se recomienda marcar la opción de agregar al PATH de todos los usuarios para no tener que hacerlo manualmente).

6.2.2. Compilar librerías

VTK 7.0.0

- Descargar VTK 7.0.0 desde este enlace de su web oficial y extraer en `C:\VTK\7.0.0\src`.
- Abrir CMake y completar:
 - `src: C:\VTK\7.0.0\src`
 - `build: C:\VTK\7.0.0\build\vs12`
- Elegir como generador *Visual Studio 12 2013*.
- Presionar configurar.
- Una vez generado seleccionar los siguientes campos:
 - `BUILD_SHARED_LIBS`
 - `Module_vtkGUISupportQt`
 - `Module_vtkGUISupportQtOpenGL`
 - `Module_vtkGUISupportQtSQL`
 - `Module_vtkGUISupportQtWebKit`
 - `Module_vtkRenderingQt`
 - `Module_vtkViewsQt`
 - `Module_vtkDICOM`
 - `VTK_Group_Qt`
- Agregar dos entradas:
 - `QT_QMAKE_EXECUTABLE:PATHFILE= C:\Qt\Qt5.5.1\5.5\msvc2013\bin\qmake.exe`
 - `CMAKE_PREFIX_PATH:PATH=C:\Qt\Qt5.5.1\5.5\msvc2013\`
- Presionar en configurar y aparecerá un error, habrá que elegir como versión de Qt la 5. Elegirla y volver a configurar.
- Configurar hasta que no aparezca ningún campo en rojo.
- Una vez configurado todo, pulsar en generar. Esto creará una serie de archivos en `C:\VTK\7.0.0\build\vs12`.
- Abrir `VTK.sln`.
- Construir en modo *Release* y esperar unos minutos a que termine.

- Copiar los archivos QVTKWidgetPlugin.lib y QVTKWidgetPlugin.dll que se encuentran en C:\VTK\7.0.0\build\vs12\lib\Release y C:\VTK\7.0.0\build\vs12\bin\Release respectivamente en C:\Qt\Qt5.5.1\5.5\msvc2013\plugins\designer (Si no se encuentran los archivos, comprobar que en CMake se marcó la opción BUILD_SHARED_LIBS). Esto hará que desde Qt Designer se pueda crea un QVTKWidget.
- Construir en modo *Debug*.
- Crear una nueva variable de entorno con nombre: VTK_DIR y valor: C:\VTK\7.0.0\build\vs12.
- Agregar al Path la siguiente dirección: C:\VTK\7.0.0\build\vs12\bin\Release.

Boost 1.60.0

- Descargar Boost 1.60.0 desde este enlace de su web oficial.
- Descomprimir en cualquier lugar, abrir la consola de comandos de Visual Studio y moverse al lugar donde ha sido extraído.
- Escribir bootstrap.bat para generar el Boost.Build.
- Compilar con: b2 toolset=msvc-12.0 --build-type=complete --abbreviate-paths architecture=x86 address-model=64 install -j4.
- Agregar al proyecto de Visual Studio:

6.2.3. Configurar proyecto

Una vez generado el proyecto realizar los siguientes cambios en la configuración:

- En *Project Properties* ir a *Configuration Properties* > *C/C++* > *General* > *Additional Include Directories* y añadir el directorio C:\Boost\include\boost-1_60.
- En *Project Properties* ir a *Configuration Properties* > *Linker* > *Additional Library Directories* y añadir el directorio C:\Boost\lib.
- En *Project Properties* ir a *Configuration Properties* > *Linker* > *System* y:
 - En *Subsystem* seleccionar la opción: Windows (/SUBSYSTEM:WINDOWS).

- En *Enable Large Addresses* seleccionar la opción: Yes (/LARGEADDRESSESAWARE).

6.3. Conceptos clave en Volume Rendering

6.3.1. 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 6.1) [16]. Se puede usar en VTK con `vtkMarchingCubes`.

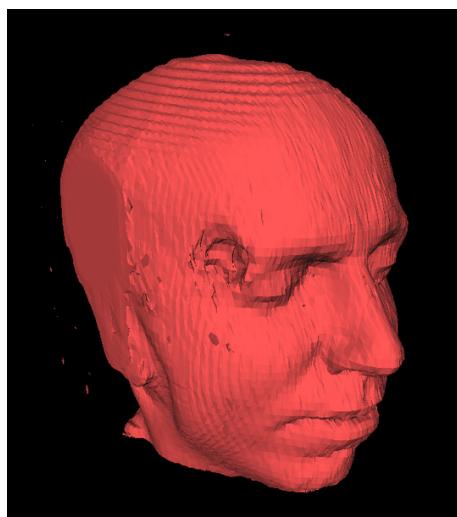


Figura 6.1: Cabeza extraída de 150 cortes obtenidos por una IRM usando *marching cubes* (sobre 150.000 triángulos). Imagen extraída de <https://en.wikipedia.org/wiki/File:Marchingcubes-head.png>

- **Texturas 2D**: 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 6.2 [17]). Se puede usar en VTK con `vtkVolumeTextureMapper`.

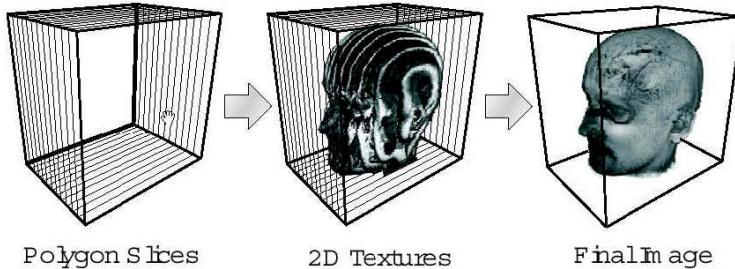


Figura 6.2: Esquema del proceso de renderizado usando texturas 2D. Imagen extraída del apéndice B del libro *An Introduction to Programming for Medical Image Analysis with the Visualization Toolkit* [17]

- **Texturas 3D:** 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 6.3) [17]. Se puede usar en VTK con `vtkVolumeTextureMapper3D`.

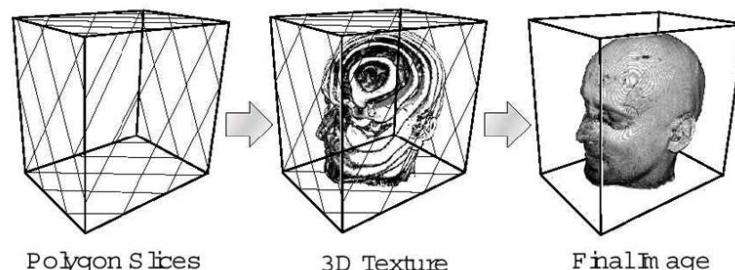


Figura 6.3: Esquema del proceso de renderizado usando texturas 3D. Imagen extraída del apéndice B del libro *An Introduction to Programming for Medical Image Analysis with the Visualization Toolkit* [17]

- **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 transparencia. 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 6.4) [17]. Se puede usar en VTK con `vtkFixedVolumeRayCastMapper`, `vtkVolumeRayCastMapper` (usan CPU), `vtkGPUVolumeRayCastMapper` (usa GPU) y `vtkSmartVolumeMapper` (según el contexto usa CPU o GPU).

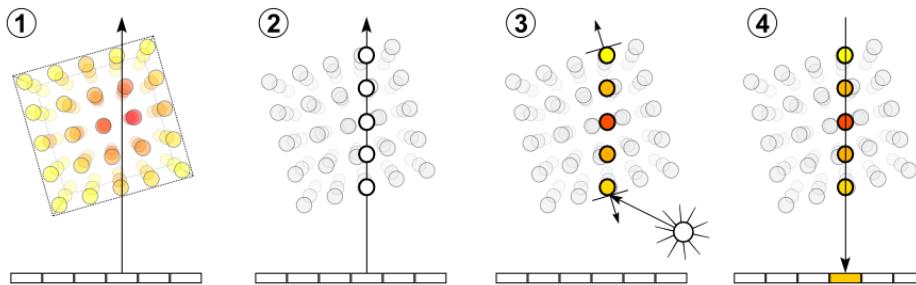


Figura 6.4: Esquema del proceso de *ray casting*. Imagen extraída de https://en.wikipedia.org/wiki/File:Volume_ray_casting.png

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. Sin embargo, la opción de *marching cubes* será útil para poder crear una malla de triángulos que se pueda exportar a un formato con el que luego pueda ser imprimida en 3D.

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.

6.3.2. 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 [20].

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

6.3.3. 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` [9]. 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.

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 6.5).

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*).

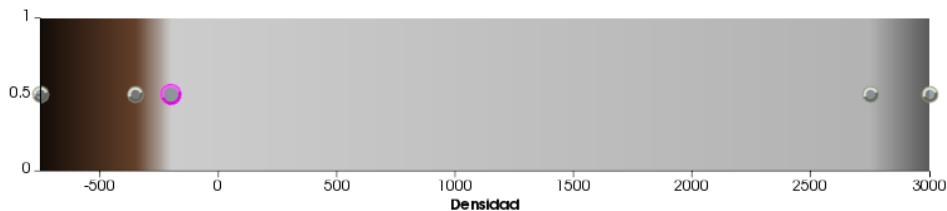


Figura 6.5: Parte de color de la función de transferencia del *preset CT-WoodSculpture* creado para visualizar esculturas de madera policromadas. Dos puntos definen el color. Uno en -750 con un tono más oscuro y otro en -350 con un tono más claro. El estuco se pinta con un color gris claro y también viene definido por dos puntos: -200 y 2750. Finalmente, el metal se verá con un tono gris oscuro definido con un punto en 3000.

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 6.6), pero para obtener mejores resultados, habrá que utilizar la opacidad gradiente.

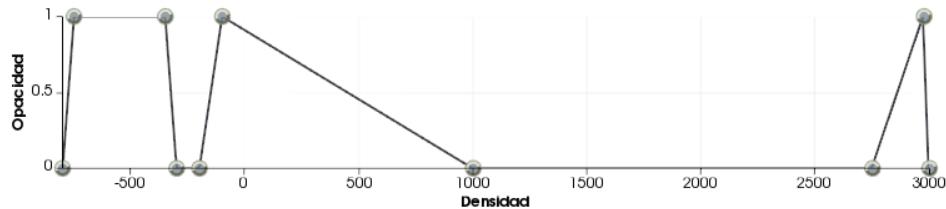


Figura 6.6: Parte de opacidad escalar de la función de transferencia del *preset CT-WoodSculpture* creado para visualizar esculturas de madera policromadas. Se pueden observar tres regiones. La primera corresponde a la madera, la segunda al estuco y la última al metal

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`).

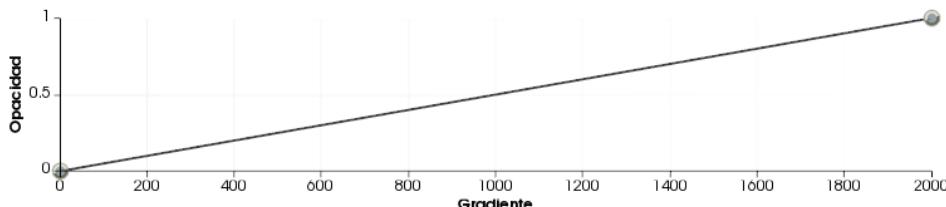


Figura 6.7: Parte de opacidad gradiente de la función de transferencia del *preset CT-WoodSculpture* creado para visualizar esculturas de madera policromadas. Se obtendría un valor cercano a 1 en la opacidad en aquellas zonas más cercanas a los bordes entre materiales pues se ha establecido que para un gradiente 0 la opacidad sea 0, y para 2000, 1.

6.3.4. Escala Hounsfield

Como ya se ha explicado, para desarrollar la función de transferencia con la que se visualiza el volumen, juega un papel muy importante el valor de densidad del material.

Este valor se encuentra en unas unidades conocidas como Unidades Hounsfield (HU) en honor al ingeniero Godfrey Newbold Hounsfield, inventor del primer escáner TAC con el que ganó el Premio Nobel de Fisiología o Medicina en 1979.

La Escala Hounsfield no es más que la transformación de la escala de coeficientes de atenuación lineal de rayos X a una nueva en relación al valor del agua destilada en condiciones normales de presión y temperatura.

El valor de HU de un material viene dado por la siguiente fórmula:

$$HU = 1000 \times \frac{\mu_{mat} - \mu_{agua}}{\mu_{agua}}$$

Donde μ_{mat} es el coeficiente de atenuación lineal del material y μ_{agua} el del agua.

Por tanto, el valor teórico del agua será 0 HU.

El rango de valores de la escala va desde -1024 HU hasta 3071 HU. 4096 valores representados mediante 12 bits.

Material	HU
Aire	-1000
Madera	-750 a -350
Estuco	200 a 1000
Metal	2900 a 3000

Cuadro 6.1: Valores en HU de distintos materiales presentes en imágenes de esculturas de madera

6.4. Fases de desarrollo

Al seguirse un desarrollo evolutivo basado en un prototipo funcional, se ha ido creando poco a poco y añadiendo componentes conforme se iban completando y testando lo que ya se había desarrollado:

6.4.1. Lectura de imágenes DICOM

El primer paso fue poder **leer una imagen DICOM**. Para ello, se utilizó el ejemplo ReadDICOM de la web de ejemplos de VTK que hace uso de `vtkDICOMImageReader` para leer la imagen.

A continuación, se pasó a **leer una serie de imágenes**. En la lista de ejemplos de VTK también había uno que realizaba esta operación: ReadDI-

COMSeries, pero era hora de integrarlo con Qt. Y aquí es donde surgió el problema comentado anteriormente con los *drivers* de la GPU.

Tras migrar a Windows y lograr crear un **pequeño programa en Qt** (Figura 6.8) con el que visualizar una serie de imágenes (desplazándose entre ellas con un slider) pasé a la siguiente fase, una de las más importantes, la de la reconstrucción volumétrica.

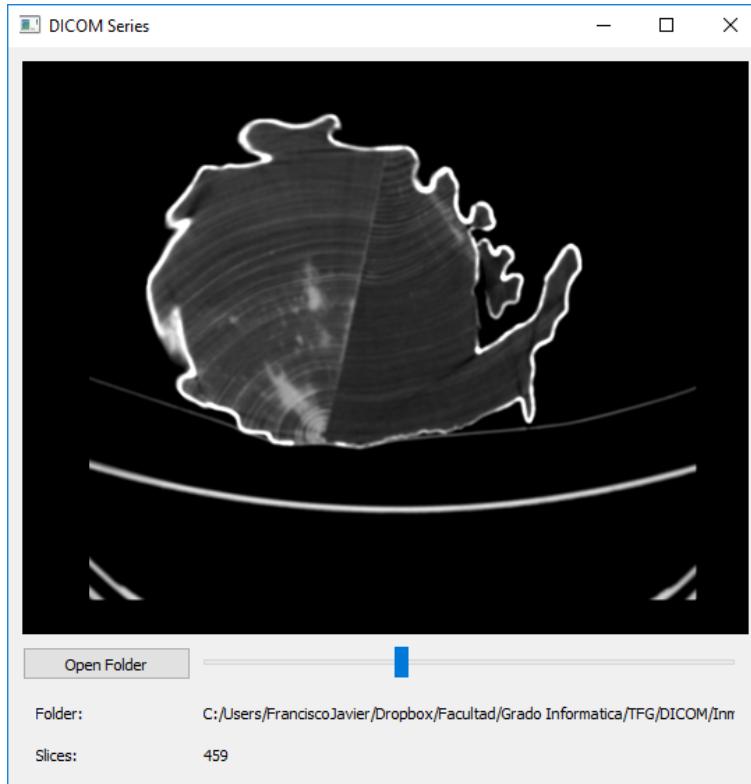


Figura 6.8: Programa sencillo para visualizar una serie de imágenes DICOM

6.4.2. Reconstrucción volumétrica

El primer paso de esta fase fue elegir cómo **renderizar el volumen**. Como ya se ha explicado detalladamente con anterioridad, se eligió el *ray-casting* que usa GPU haciendo uso de *vtkSmartVolumeMapper*.

Una parte fundamental del *ray-casting* es la **función de transferencia** con la que mapea valores. Para poder crearla se ha hecho uso del software 3DSlicer [12] para poder ver los valores de densidad de los distintos materiales de la escultura. Con esto se creó una función de transferencia bastante básica con la que poder hacer el primer renderizado (Figura 6.9).



Figura 6.9: Primer renderizado sobre la figura de San Juan Evangelista con una función de transferencia que no utilizaba la opacidad gradiente

Una vez comprobado que se estaba realizando bien el renderizado, se debía mejorar la función de transferencia. Para ello se creó una **barra de herramientas** (Figura 6.10) con la que poder cambiar esta función de transferencia añadiendo puntos a ésta. Pese a no ser muy amigable para el usuario, era un primer prototipo que ayudaría a crear una función de transferencia con la que poder seguir trabajando.

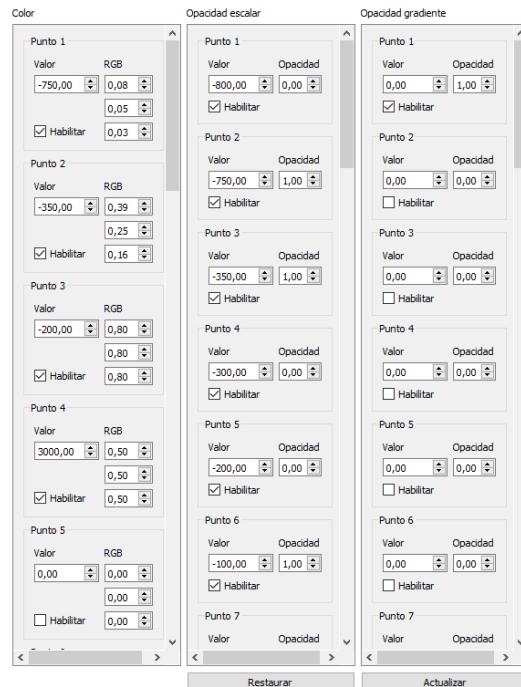


Figura 6.10: Barra de herramientas con la que añadir y quitar puntos a la función de transferencia

6.4.3. Generación de cortes

Antes de mejorar la forma en la que se edita la función de transferencia se pasó a realizar una de las partes más importantes. La de la poder ver cortes de la figura. Para empezar se creó un nuevo *widget* con el que se combinó lo implementado anteriormente para visualizar cortes (Figura 6.8).

Pero de esta forma no se estaban generando cortes sino visualizando las propias imágenes con las que se reconstruía el volumen. La idea era poder **crear un corte en la figura dado por un plano arbitrario**. Para ello se hizo uso de `vtkImagePlaneWidget` modificando esta clase para que se renderizase el corte al mismo tiempo que se movía el plano. Este plano se conectaba con los datos del volumen (`vtkImageData`) para mostrar la salida en un `vtkImageViewer2` (Figura 6.11).

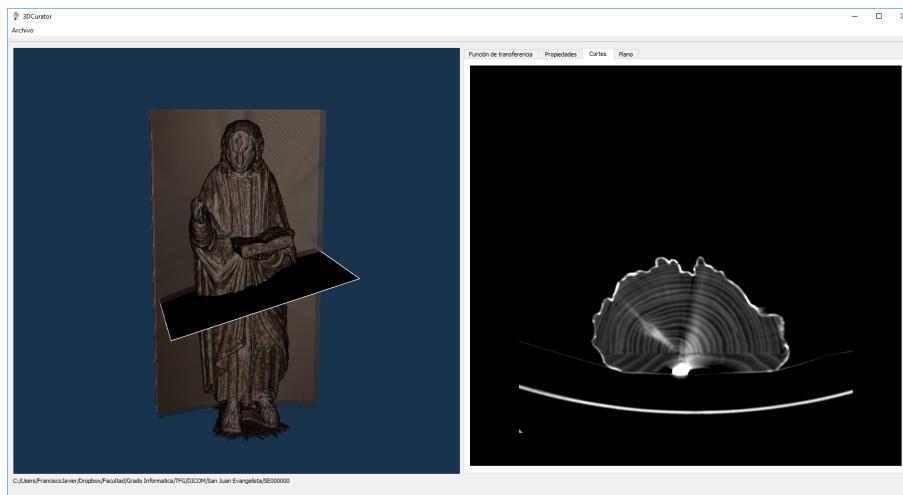


Figura 6.11: Primera implementación del pano que corta la figura y renderiza el corte en otro *widget*

Resultó sencillo añadir a continuación mejoras a este plano, como colocarlo en **posiciones por defecto** (axial, coronal y sagital) y visualizar la salida con el **mismo color que la función de transferencia**.

Además se reescribieron métodos de eventos de ratón para eliminar los innecesarios que ya traían las clases de VTK.

6.4.4. Guardar imágenes

Otra funcionalidad básica era la de poder **guardar imágenes de ambos visores**. VTK ofrece con las distintas subclases de `vtkImageWriter` una serie de clases con las que exportar imágenes en distintos formatos.

Se han utilizado tanto `vtkJPEGWriter` como `vtkPNGWriter` para darle al usuario la posibilidad de guardar la imagen tanto en JPEG como en PNG.

6.4.5. Editar función de transferencia

Con el software cada vez más completo, era el momento de mejorar la **edición de función de transferencia**. La implementación anterior (Figura 6.10) no ofrecía retroalimentación visual al usuario de cómo era la curva de la función o la paleta de colores. Además, añadir y editar puntos era un trabajo tedioso.

Resultaba imprescindible cambiarlo y la forma más cómoda de editar la función era trabajar sobre la misma. Es decir, **mostrar la función en una gráfica** con cada uno de sus puntos y poder mover, añadir o borrarlos interactuando directamente con la gráfica.

Explorando entre las clases que proporcionaba VTK se encontraron:

- `vtkColorTransferFunctionItem`: Muestra la función de transferencia de color como un `vtkPlot` que se puede añadir a una `vtkChartXY`.
- `vtkPiecewiseFunctionItem`: Muestra la función de transferencia de opacidad como un `vtkPlot` que se puede añadir a una `vtkChartXY`.
- `vtkColorTransferControlPointsItem`: Muestra y permite modificar los puntos de la función de transferencia de color como un `vtkPlot` que se puede añadir a una `vtkChartXY`.
- `vtkPiecewiseControlPointsItem`: Muestra y permite modificar los puntos de la función de transferencia de opacidad como un `vtkPlot` que se puede añadir a una `vtkChartXY`.

No obstante, había que reescribir algunos métodos de estas clases para poder adaptarlas al software. Había que eliminar algunos eventos innecesarios y agregar nuevos acciones a otros como volver a renderizar el volumen cada vez que se cambie algún punto.

Llevar a cabo este trabajo fue bastante costoso pero finalmente se logró proporcionar al usuario una interfaz gráfica para editar la función de transferencia intuitiva y fácil de utilizar (Figura 6.12).

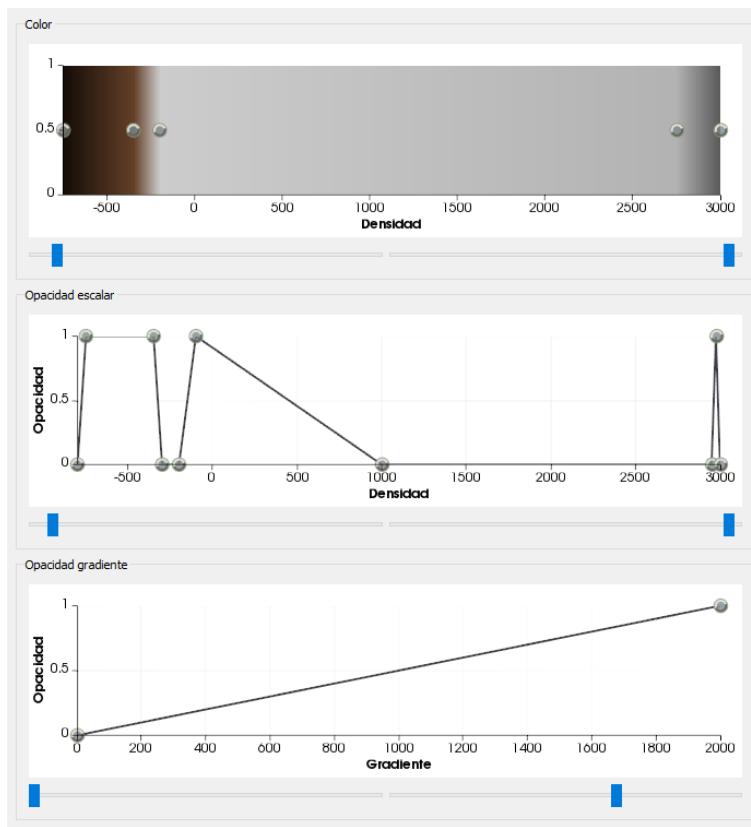


Figura 6.12: Interfaz para editar la función de transferencia

6.4.6. Importar y exportar función de transferencia

Con la nueva interfaz iba a resultar sencillo crear una batería de funciones de transferencia, por tanto, llegó el momento de almacenarlas de alguna forma para luego poder usarlas.

Para esto se decidió utilizar XML almacenándolas con un formato en el que el software pudiese exportar e importarlas (Código 6.1).

Listing 6.1: Descripción de formato XML utilizado usando XML Schema

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <xsschema xmlns:xss="http://www.w3.org/2001/XMLSchema">
3   <xselement name="tf">
4     <xsccomplexType>
5       <xsssequence>
6         <xselement name="color">
7           <xsccomplexType>
8             <xsssequence>
9               <xselement name="point" minOccurs="0" maxOccurs="unbounded">
10              <xsccomplexType>
11                <xsssequence>

```

```

12      <xs:element name="x" type="xs:decimal" />
13      <xs:element name="r" type="xs:decimal" />
14      <xs:element name="g" type="xs:decimal" />
15      <xs:element name="b" type="xs:decimal" />
16    </xs:sequence>
17  </xs:complexType>
18  </xs:element>
19 </xs:sequence>
20 </xs:complexType>
21 </xs:element>
22 <xs:element name="scalar">
23   <xs:complexType>
24     <xs:sequence>
25       <xs:element name="point" minOccurs="0" maxOccurs="unbounded">
26         <xs:complexType>
27           <xs:sequence>
28             <xs:element name="x" type="xs:decimal" />
29             <xs:element name="y" type="xs:decimal" />
30           </xs:sequence>
31         </xs:complexType>
32       </xs:element>
33     </xs:sequence>
34   </xs:complexType>
35 </xs:element>
36 <xs:element name="gradient">
37   <xs:complexType>
38     <xs:sequence>
39       <xs:element name="point" minOccurs="0" maxOccurs="unbounded">
40         <xs:complexType>
41           <xs:sequence>
42             <xs:element name="x" type="xs:decimal" />
43             <xs:element name="y" type="xs:decimal" />
44           </xs:sequence>
45         </xs:complexType>
46       </xs:element>
47     </xs:sequence>
48   </xs:complexType>
49 </xs:element>
50 </xs:sequence>
51 <xs:attribute name="name" type="xs:string" use="required"/>
52 <xs:attribute name="description" type="xs:string" use="required"/>
53 </xs:complexType>
54 </xs:element>
55 </xs:schema>
```

Por ejemplo, la función de transferencia principal utilizada en este formato XML sería la siguiente (Código 6.2)

Listing 6.2: Ejemplo de función de transferencia en el formato XML descrito

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <tf name="CT-WoodSculpture" description="Escultura de madera con
  clavos y una capa de estuco y policromado">
3   <color>
4     <point>
5       <x>-750</x>
6       <r>0.08</r>
7       <g>0.05</g>
8       <b>0.03</b>
9     </point>
```

```
10 <point>
11   <x>-350</x>
12   <r>0.39</r>
13   <g>0.25</g>
14   <b>0.16</b>
15 </point>
16 <point>
17   <x>-200</x>
18   <r>0.8</r>
19   <g>0.8</g>
20   <b>0.8</b>
21 </point>
22 <point>
23   <x>2750</x>
24   <r>0.7</r>
25   <g>0.7</g>
26   <b>0.7</b>
27 </point>
28 <point>
29   <x>3000</x>
30   <r>0.35</r>
31   <g>0.35</g>
32   <b>0.35</b>
33 </point>
34 </color>
35 <scalar>
36   <point>
37     <x>-800</x>
38     <y>0</y>
39   </point>
40   <point>
41     <x>-750</x>
42     <y>1</y>
43   </point>
44   <point>
45     <x>-350</x>
46     <y>1</y>
47   </point>
48   <point>
49     <x>-300</x>
50     <y>0</y>
51   </point>
52   <point>
53     <x>-200</x>
54     <y>0</y>
55   </point>
56   <point>
57     <x>-100</x>
58     <y>1</y>
59   </point>
60   <point>
61     <x>1000</x>
62     <y>0</y>
63   </point>
64   <point>
65     <x>2950</x>
66     <y>0</y>
67   </point>
68   <point>
69     <x>2976</x>
70     <y>1</y>
71 </point>
```

```
72 <point>
73   <x>3000</x>
74   <y>0</y>
75 </point>
76 </scalar>
77 <gradient>
78   <point>
79     <x>0</x>
80     <y>0</y>
81   </point>
82   <point>
83     <x>2000</x>
84     <y>1</y>
85   </point>
86 </gradient>
87 </tf>
```

Para poder **gestionar estos ficheros**, se decidió utilizar una librería. Había muchas opciones, pero lo que se quería gestionar era bastante básico por lo que utilizar una demasiado compleja podría resultar contraproducente ya que se tardaría bastante tiempo en aprender a utilizarla.

Finalmente se optó por **Boost**, una librería muy extensa que cuenta con un pequeño gestor de ficheros XML.

A partir de un fichero XML, Boost transforma la información a un **struct** en forma de árbol de forma que luego se puede recorrer fácilmente obteniendo la información deseada.

6.4.7. Realizar medida

Una de las primeras ideas que surgieron para añadir al software era la de poder realizar medidas. Pues esto podría resultar de mucha utilidad para los restauradores.

En poco tiempo se consiguió, gracias a la clase **vtkDistanceWidget** añadir una regla con la que **realizar medidas** (Figura 6.13).

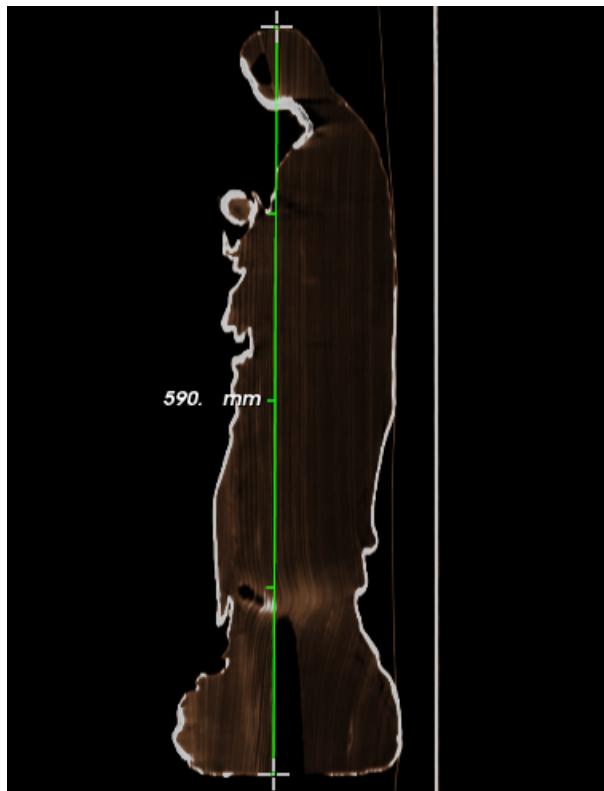


Figura 6.13: Primera implementación de regla para medir. Solo se podía utilizar una en el visor de cortes

Lo ideal sería poder añadir más de una regla, pero antes de llevar esta mejora a cabo, se decidió pasar a otra quizás más importante.

6.4.8. Borrar partes innecesarias

Al hacer *ray-casting* se mapean los datos de la imagen en valores de color y opacidad gracias a la función de transferencia. Esto hace que se puedan diferenciar distintos materiales. Pero hay materiales con compuestos similares que pueden aparecer pues se encontraban en el escáner a la hora de realizarse la tomografía.

Hablo de la **camilla donde está apoyada la figura** que tiene zonas en las que hay un material con una densidad similar al estuco y otra con uno similar a la madera. Por lo que, al visualizarse, aparecen. Aunque no tienen nada que ver con la figura. Y no solo sobran, también molestan pues en el caso de una capa de la camilla, impide ver la figura a las espaldas (Figura 6.14).



Figura 6.14: La camilla tapa la espalda de la figura e impide verla

Es por tanto necesario proveer al software de una herramienta para poder **borrar estas partes**.

En un principio se pensó hacer un borrado en el que a partir de un punto seleccionado por el usuario, se **extendiese en 3D** comprobando si alguno de los puntos de alrededor tiene un valor similar para seguir extendiéndose recursivamente. Pero la camilla pegada a la figura tiene zonas donde las zonas de la camilla y la figura tienen un valor muy similar y esto provocaría que borrarase en el interior de la figura.

La segunda capa de camilla, la más gruesa con valores de estuco (puede verse más blanca en las imágenes), está separada de la figura por un material con valores cercanos a los del aire y que, por tanto, no se muestran.

Esta situación hizo que se cambiase un poco el planteamiento y se borrarase no por zonas con valor parecido sino por *islas*. Entendiendo isla como una zona separada de las demás.

Se empezó a implementar el **algoritmo recursivo** descrito anteriormente, pero usando una pila de puntos a analizar en un bucle que continuase hasta que estuviese vacía en lugar de usar recursividad como tal pues podría agotar la pila de llamadas a funciones.

Sin embargo esto no solucionaba los problemas con el hardware porque, aunque no se agotase la pila de llamadas, la pila de puntos que se utilizaba se hacía demasiado grande. Tanto que agotaba la memoria utilizada por el programa.

Era necesario hacer un cambio y se pasó de extenderse en 3D a extenderse en 2D imagen por imagen. Pasando de introducir 26 puntos en cada iteración a 8 con un tamaño total de $res_x \times res_y$, que hace que las coordenadas de una imagen de $1024px \times 1024px$ tenga espacio de sobra en la memoria de la pila sin desbordarse.

Este cambio no solo era más óptimo en cuanto a utilización de memoria, sino que resultaba bastante más rápido. Y, aunque en ocasiones, no borre la isla con un solo click, el proceso de borrado tarda escasos segundos (Código 6.3).

Listing 6.3: Pseudocódigo del borrado

```

1 void deletByImages(data, point, bounds) {
2     deleteImage(data, point, bounds)
3     z = point.z + 1
4     while (z < bounds.z.max) {
5         point.z = z
6         deleteImage(data, point, bounds)
7         z++
8     }
9     z = point.z - 1
10    while (z >= bounds.z.min) {
11        point.z = z
12        deleteImage(data, point, bounds)
13        z--
14    }
15}
16
17 void deleteImage(data, point, bounds) {
18     z = point.z
19     xy = {point.x, point.y}
20     if (data(point) < AIR_HU) {
21         point = searchInitialVoxel(data, point, bounds)
22     }
23     stack.push(xy)
24     while (!stack.empty()) {
25         xy = stack.pop()
26         if (isInside(xy, bounds)) {
27             point = {xy.x, xy.y, z}
28             if (data(point) >= MIN_AIR) {
29                 data(point) = AIR_HU
30                 stack.push(pointsAround(point))
31             }
32         }
33     }
34 }
```

Gracias a este algoritmo de borrado se puede eliminar fácil y rápidamente la camilla que no está pegada a la figura y, al ser la capa que está pegada muy fina y tener activada la opacidad gradiente, se puede ver las espaldas de la figura a diferencia de antes (Figura 6.15).



Figura 6.15: La camilla ha sido eliminada y permite ver la espalda a diferencia de antes (Figura 6.14)

No obstante, y aunque esta capa que se sigue viendo no molesta tanto, a la hora de realizar el escáner podría resultar útil colocar un material que se sepa que tiene un valor de densidad similar al aire entre la figura y la camilla para que así pueda ser borrada por completa la camilla.

6.4.9. Exportar malla de triángulos

Una de las últimas ideas de funcionalidad a introducir en el software es la de la **generación de una malla de triángulos** del modelo que pudiese ser exportada en formato STL.

Para ello, como se avanzó con anterioridad, se utiliza la técnica de *marching cubes* y la clase `vtkMarchingCubes` que VTK proporciona para ello.

Por lo que se introduce otro *widget* en la aplicación para visualizar la malla generada a partir de un valor de isosuperficie dado por el usuario.

De esta forma se puede obtener, por ejemplo, un modelo de los clavos de una figura 6.16 que luego podrían imprimirse con una impresora 3D.

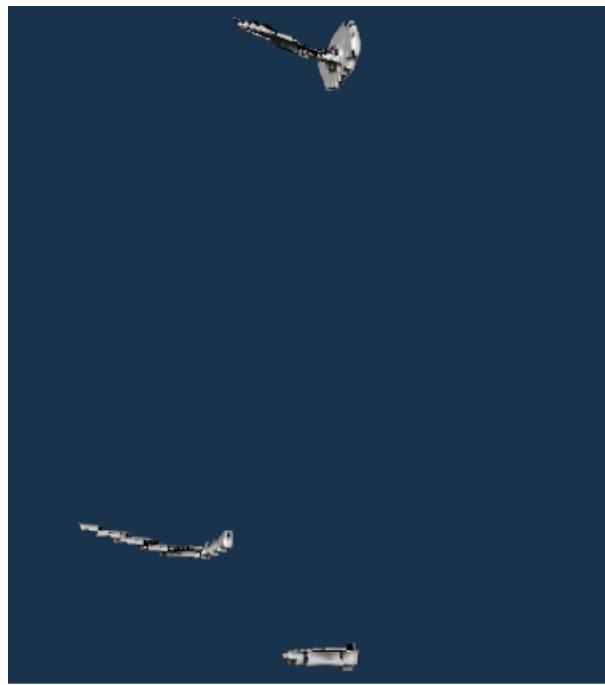


Figura 6.16: Malla generada con un valor de isosuperficie de 2976 HU para poder extraer los clavos

6.4.10. Mostrar valor de densidad

Al empezar a crear la función de transferencia, ya se comentó la utilización de otro software para comprobar el **valor escalar de cada pixel** de la imagen para poder ver entre qué valores se movía cada material. Es por ello que pareció interesante añadir en la interfaz esta información.

Por lo que, como última idea de mejora, y aprovechando lo aprendido de los *pickers* durante la implementación del borrado, se añadió a la interfaz una etiqueta donde se muestra el valor del pixel sobre el que el ratón está posicionado.

6.4.11. Realizar varias medidas

Dada por finalizada la lluvia de ideas que añadir como mejoras, había que realizar todo aquello que se decidió implementar más adelante.

Hablo principalmente de la **gestión de reglas para realizar varias medidas** en cualquiera de los dos visores principales. Para ello había que dotar a la interfaz de un cuadro con las reglas con opciones para añadir, eliminar y habilitar o deshabilitar.

Para ello, dado que contaba con la experiencia justa con Qt, se realizó en primer lugar la gestión de elementos de la interfaz para poder posteriormente mapear cada ítem de las cajas a un `vtkDistanceWidget` distinto.

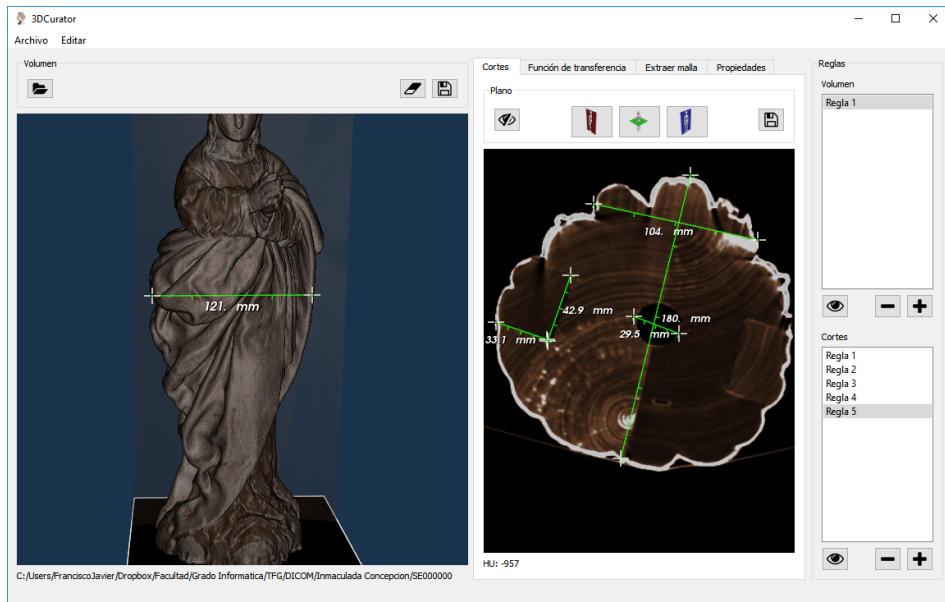


Figura 6.17: A la derecha, barra de las reglas y en los visores cada una de ellas midiendo algo distinto

6.4.12. Cambiar color de fondo de los visores

Una de las primeras ideas que se pensaron fue la de poder permitir cambiar al usuario el **color de fondo de los *widget*** donde se visualizan tanto el volumen como la malla generada. Pero no se realizó hasta al final pues había cosas más importantes que implementar.

Con esta pequeña mejora junto a otras menores en la interfaz, se dio por terminada la fase de desarrollo del software. Cumpliendo los requisitos iniciales y añadiendo múltiples mejoras que han acabado formando un software bastante completo pero al mismo tiempo sencillo de utilizar.

Bibliografía

- [1] About dicom. <http://dicom.nema.org/Dicom/about-DICOM.html>.
- [2] Boost. <http://www.boost.org/>.
- [3] Cmake. <https://cmake.org/>.
- [4] Escáner o tomografía. ¿qué es y cómo funciona? <http://www.fisioterapia-online.com/videos/escaner-o-tomografia-que-es-y-como-funciona>.
- [5] Especificación de requisitos según el estándar de iee 830. <https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf>.
- [6] Qt. <https://www.qt.io/>.
- [7] Visual paradigm. <https://www.visual-paradigm.com/>.
- [8] Vtk. <http://www.vtk.org/>.
- [9] Vtk example - medical 4. <https://github.com/Kitware/VTK/blob/master/Examples/Medical/Cxx/Medical4.cxx>.
- [10] 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>.
- [11] Federico Cesarani, Maria Cristina Martina, Renato Grilletto, Rosa Boano, Anna Maria Donadoni, Valter Capusotto, 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>.

- [12] 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>.
- [13] David Gobbi. Vtk classes for dicom data. <http://dgobbi.github.io/vtk-dicom/doc/vtk-dicom.pdf>, 2015.
- [14] 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>.
- [15] 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.
- [16] 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>.
- [17] Xenophon Papademetris and Alark Joshi. *An Introduction to Programming for Medical Image Analysis with the Visualization Toolkit*. Bio-Image Suite, 2nd edition, 2009.
- [18] Oleg S. Panykh. *Digital Imaging and Communications in Medicine (DICOM). A Practical Introduction and Survival Guide*. Springer, 2nd edition, 2012.
- [19] 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>.
- [20] 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/>.