



UNIVERSIDAD DE GRANADA

TRABAJO FIN DE MÁSTER
INGENIERÍA EN INFORMÁTICA

3DCurator

Sistema gráfico de ayuda al diagnóstico e intervención de esculturas mediante datos médicos volumétricos

Autor

Francisco Javier Bolívar Lupiáñez

Director

Francisco Javier Melero Rus



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, 30 de junio de 2018



3DCurator

Sistema gráfico de ayuda al diagnóstico e intervención de esculturas mediante datos médicos volumétricos

Autor

Francisco Javier Bolívar Lupiáñez

Director

Francisco Javier Melero Rus

3DCurator: Sistema gráfico de ayuda al diagnóstico e intervención de esculturas mediante datos médicos volumétricos

Francisco Javier Bolívar Lupiáñez

Palabras clave: informática gráfica, renderizado de volúmenes, tomografía computarizada, escultura policromada de madera, conservación y restauración, restaurador de arte

Resumen

TODO

3DCurator: Graphic system to aid the diagnosis and intervention of sculptures using volumetric medical data

Francisco Javier Bolívar Lupiáñez

Keywords: Computer Graphics, Volume Rendering, Computed Tomography, Polychromed Wood Sculpture, Conservation-Restoration of Cultural Heritage, Art Curator

Abstract

TODO

Yo, **Francisco Javier Bolívar Lupiáñez**, alumno de la titulación Máster 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 Máster en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Francisco Javier Bolívar Lupiáñez

Granada a 30 de junio de 2018.

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

Informa:

Que el presente trabajo, titulado *3DCurator, Sistema gráfico de ayuda al diagnóstico e intervención de esculturas mediante datos médicos volumétricos*, 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 30 de junio de 2018.

El director:

Francisco Javier Melero Rus

Agradecimientos

TODO

Índice general

1. Introducción	1
1.1. Tomografía Computarizada	2
1.1.1. Historia	2
1.1.2. Generaciones	2
1.2. Esculturas de madera policromadas	7
1.2.1. Historia	7
1.2.2. Maderas más utilizadas	7
1.2.3. Defectos de la madera	8
1.2.4. Proceso de tallado	9
1.3. Estado del arte	10
1.4. Motivación	11
2. Especificación de requisitos	13
2.1. Introducción	13
2.1.1. Propósito	13
2.1.2. Ámbito del sistema	13
2.1.3. Definiciones, acrónimos y abreviaturas	13
2.1.4. Visión general del documento	16
2.2. Descripción general	17
2.2.1. Perspectiva del producto	17
2.2.2. Funciones del producto	17
2.2.3. Características de los usuarios	17
2.2.4. Restricciones	18
2.2.5. Suposiciones y dependencias	18
2.3. Requisitos específicos	18
2.3.1. Interfaces	18
2.3.2. Funciones	20
2.3.3. Requisitos de rendimiento	22
2.3.4. Restricciones de diseño	22
2.3.5. Atributos del software	22

3. Planificación	23
3.1. Planificación inicial	23
3.1.1. Diagrama de Gantt	24
3.2. Metodología de trabajo	24
3.2.1. <i>GitHub</i>	24
3.2.2. <i>Issues</i>	24
3.2.3. <i>Milestones</i>	25
3.2.4. <i>Branches</i>	26
3.3. Resultados	28
4. Análisis	29
4.1. Historias de usuario	29
4.1.1. <i>Product backlog</i>	29
4.1.2. Tarjetas de las historias de usuario	30
5. Diseño	45
5.1. Arquitectura del software	45
5.2. Diagrama de paquetes	46
5.3. Diagramas de clases	46
5.3.1. <i>Chart</i>	47
5.3.2. <i>Core</i>	47
5.3.3. <i>Documentation</i>	48
5.3.4. <i>GUI</i>	49
5.3.5. <i>Interactor</i>	51
5.3.6. <i>Segmentation</i>	51
5.3.7. <i>Util</i>	51
5.3.8. <i>Widget</i>	52
6. Desarrollo del trabajo	53
6.1. Desarrollo profundo del estado del arte	53
6.1.1. Obtención de imágenes	54
6.1.2. Filtrado	57
6.1.3. Segmentación	60
6.2. Plataforma de desarrollo	64
6.3. Instalación y configuración	65
6.3.1. Entorno de desarrollo	65
6.3.2. Generar librerías	65
6.3.3. Compilar librerías	67
6.3.4. Compilar proyecto	68
6.4. Fases de desarrollo	69
6.4.1. Pre-procesamiento de datos	69
6.4.2. Subdivisión de piezas de madera	72
6.4.3. Herramientas de documentación	81
6.4.4. Internacionalización	88

7. Resultados	91
7.1. Filtrado	92
7.1.1. Filtro <i>gaussiano</i>	92
7.1.2. Filtro media	93
7.1.3. Filtro mediana	94
7.2. Segmentación	95
7.2.1. Inmaculada Concepción	95
7.2.2. San Juan Evangelista	96
8. Conclusiones y trabajos futuros	99
Bibliografía	103

Índice de figuras

1.1.	Primera generación de aparatos de tomografía computarizada [18]	3
1.2.	Segunda generación de aparatos de tomografía computarizada [18]	3
1.3.	Tercera generación de aparatos de tomografía computarizada [18]	4
1.4.	Cuarta generación de aparatos de tomografía computarizada [18]	5
1.5.	Cuarta generación de aparatos de tomografía computarizada [18]	5
1.6.	Diferencias entre TC helicoidal multicorte (B) y monocorte (A) [27]	6
1.7.	Equipo TC con doble fuente [27]	7
1.8.	Obtención de datos volumétricos de una escultura posteriormente examinada	12
2.1.	Boceto a mano alzada de la posible distribución de los elementos en la GUI principal	19
2.2.	Boceto a mano alzada de la posible distribución de los elementos en la GUI de selección de filtros	20
3.1.	Diagrama de Gantt con la planificación inicial del proyecto generado con <i>Microsoft Office Project 2016</i>	24
3.2.	Vista con la lista de <i>issues</i> abiertos con sus correspondientes etiquetas y el <i>milestone</i> al que pertenecen	25
3.3.	Resumen de los <i>milestones</i> con su porcentaje de elaboración .	26
3.4.	Ejemplo de <i>commits</i> en la rama <i>develop</i>	27
3.5.	Gráfico de los <i>commits</i> realizados en la rama <i>develop</i> . El gráfico es orientativo porque hay <i>commits</i> con poca funcionalidad y otros con mucha, pero ayuda a ver las etapas donde más se trabajó	28
5.1.	Diagrama de paquetes de 3DCurator	46
5.2.	Diagrama de clases del paquete <i>Chart</i>	47

5.3.	Diagrama de clases del paquete <i>Core</i>	48
5.4.	Diagrama de clases del paquete <i>Documentation</i>	49
5.5.	Diagrama de clases del paquete <i>GUI</i>	50
5.6.	Diagrama de clases del paquete <i>Interactor</i>	51
5.7.	Diagrama de clases del paquete <i>Segmentation</i>	51
5.8.	Diagrama de clases del paquete <i>Util</i>	52
5.9.	Diagrama de clases del paquete <i>Widget</i>	52
6.1.	Escáner TC sin la carcasa, por lo que se puede ver sus componentes internos. T: Tubo rayos X, D: Detectores rayos X, X: Haces de rayos X y R: <i>Gantry</i> . Imagen extraída de: https://en.wikipedia.org/wiki/File:Ct-internals.jpg	54
6.2.	Esquema de un escáner IRM (Sección Longitudinal). Imagen extraída de: https://en.wikipedia.org/wiki/File:Mri_scanner_schematic_labelled.svg	55
6.3.	Esquema de un escáner IRM (Sección Axial). Imagen extraída de: https://en.wikipedia.org/wiki/File:Mri_scanner_schematic_labelled.svg	56
6.4.	A la izquierda gráfica de la relajación longitudinal (crecimiento logarítmico) y a la derecha gráfica de la relajación transversal (crecimiento exponencial) [16]	57
6.5.	Vecindarios 3x3, 5x5 y 7x7 de un píxel (rojo).	58
6.6.	Ejemplo de ruido tipo <i>salt-and-pepper</i> , utilizando un filtro mediana. Imagen extraída de: https://en.wikipedia.org/wiki/File:Medianfilterp.png	59
6.7.	Ejemplo de suavizado usado un filtro <i>gaussiano</i>	60
6.8.	Segmentación de los plexos coirodeos de una imagen de un cerebro usando una segmentación basada en crecimiento usando un umbral entre 210 y 255	62
6.9.	El terreno se indica con una línea continua negra, las distintas regiones están marcadas con líneas discontinuas negras, las flechas los mínimos locales, las líneas discontinuas azules los niveles de agua y las líneas discontinuas azules los distintos niveles de agua para realizar la inundación. Imagen extraída de https://en.wikipedia.org/wiki/File:Watershed_transform-flood_interpretation.svg	63
6.10.	Ejemplo de hígado en un corte segmentado usando <i>livewire</i> [28]	64
6.11.	Botón que hay que pulsar para que aparezca el diálogo de filtrado	71
6.12.	Cuadro de diálogo con los filtros posibles. Se muestran tres imágenes, una por cada pestaña abierta para mostrar el parámetro que hay que establecer para aplicar el filtro	71
6.13.	Ruido producido por los elementos metálicos	72
6.14.	A la izquierda podemos ver continuidad en algunos anillos . .	73

6.15. Corte al que hemos aplicado el algoritmo de detección de bordes de <i>Canny</i> . Podemos apreciar fácilmente la línea recta que separa las dos piezas de madera	74
6.16. Corte al que hemos aplicado el algoritmo de detección de bordes de <i>Canny</i> para aplicar a continuación el algoritmo de <i>Hough</i> para detectar líneas rectas. Se muestran las cinco líneas más largas que se han encontrado	75
6.17. Proceso completo de la segmentación 2D donde se detalla cuándo interviene el ordenador y cuándo lo hace el usuario .	76
6.18. Resultado de segmentación en distintos cortes	78
6.19. Botón que hay que pulsar para cambiar a modo de segmentación	79
6.20. Líneas detectadas en el corte. El usuario tiene que elegir cuál quiere utilizar	80
6.21. Resultado de la segmentación, si se pulsa en OK se pasará a guardar	81
6.22. Listado de ROD y acciones para realizar con ellas	82
6.23. Listado de reglas y acciones para realizar con ellas	83
6.24. Listado de transportadores de ángulos y acciones para realizar con ellos	83
6.25. Listado de notas y acciones para realizar con ellas	84
 7.1. Esculturas utilizadas para realizar las pruebas. Inmaculada Concepción (izquierda) y San Juan Evangelista (derecha) . .	91
7.2. De izquierda a derecha y arriba a abajo: figura original y aplicando el filtro <i>gaussiano</i> con 1, 2 y 3 repeticiones	92
7.3. De izquierda a derecha y arriba a abajo: figura original y aplicando el filtro media con vecindarios de 3x3, 5x5 y 7x7 . .	93
7.4. De izquierda a derecha y arriba a abajo: figura original y aplicando el filtro mediana con vecindarios de 3x3, 5x5 y 7x7	94
7.5. Líneas encontradas en un corte a la altura de la cintura donde se ve la línea (verde) que separa las dos piezas del embón . .	95
7.6. Volumen con la pieza de la derecha y un corte donde se ve dónde ha cortado	96
7.7. Volumen con la pieza de la izquierda y un corte donde se ve dónde ha cortado	96
7.8. Líneas encontradas en un corte a la altura de la cintura donde se ve la línea (azul) que separa las dos piezas del embón . . .	97
7.9. Volumen con la pieza frontal y un corte donde se ve dónde ha cortado	98
7.10. Volumen con la pieza trasera y un corte donde se ve dónde ha cortado	98

Índice de cuadros

3.1. Planificación inicial	24
4.1. Historias de usuario	30
4.2. Historia de usuario - Exportar volumen	31
4.3. Historia de usuario - Importar volumen	31
4.4. Historia de usuario - Filtro <i>gaussiano</i>	32
4.5. Historia de usuario - Filtro media	32
4.6. Historia de usuario - Filtro mediana	33
4.7. Historia de usuario - Segmentar pieza de madera	33
4.8. Historia de usuario - Crear ROD	34
4.9. Historia de usuario - Eliminar ROD	34
4.10. Historia de usuario - Exportar ROD	35
4.11. Historia de usuario - Importar ROD	35
4.12. Historia de usuario - Cambiar ROD	36
4.13. Historia de usuario - Crear regla	36
4.14. Historia de usuario - Eliminar regla	37
4.15. Historia de usuario - Editar regla	37
4.16. Historia de usuario - Ocultar regla	38
4.17. Historia de usuario - Mostrar regla	38
4.18. Historia de usuario - Crear transportador de ángulos	39
4.19. Historia de usuario - Eliminar transportador de ángulos	39
4.20. Historia de usuario - Editar transportador de ángulos	40
4.21. Historia de usuario - Ocultar transportador de ángulos	40
4.22. Historia de usuario - Mostrar transportador de ángulos	41
4.23. Historia de usuario - Crear nota	41
4.24. Historia de usuario - Eliminar nota	42
4.25. Historia de usuario - Editar nota	42
4.26. Historia de usuario - Ocultar nota	43
4.27. Historia de usuario - Mostrar nota	43
4.28. Historia de usuario - Internacionalización	44
5.1. Arquitectura del software	46

Capítulo 1

Introducción

Este trabajo de fin de máster es una continuación del trabajo de fin de grado que realicé en el que se desarrolló un *software* (3DCurator) para visualizar un conjunto de datos volumétricos, en formato DICOM, de esculturas de madera policromada.

Con este *software*, los expertos en la materia como restauradores o historiadores del arte podían inspeccionar el interior de las esculturas sin dañarlas para un posterior proceso de estudio, restauración y/o conservación.

En este trabajo de fin de máster se llevarán a cabo distintas tareas de desarrollo que se integrarán a 3DCurator así como un estudio teórico más completo del proceso de obtención de datos volumétricos en los objetos que abarcan el campo de estudio en cuestión.

Las tareas de desarrollo que se integrarán con el *software* desarrollado se dividen en tres bloques:

- **Pre-procesamiento de datos:** Se estudiarán los distintos filtros disponibles para ver cuáles ofrecen mejores resultados en la tarea de reducción de ruido.
- **Subdivisión de piezas de madera:** Las esculturas suelen estar formadas por distintas piezas de madera. Se estudiará la forma de segmentarla probando, en primer lugar, los algoritmos ya existentes utilizados principalmente en medicina. Si ninguno ofrece los resultados que esperamos, se pasará a desarrollar uno propio.
- **Herramientas de documentación:** Se incluirán herramientas para ayudar a los usuarios en la tarea de documentación permitiendo, por ejemplo, incluir distintas anotaciones en puntos de interés.

A parte se van a realizar mejoras en el código para que sea más legible

y mantenible como la internacionalización pues hasta ahora todos los textos estaban *hard-coded*.

Además de las librerías que ya se utilizaron: VTK [10] (visualización), Qt [7] (GUI) y Boost [1] (XML); se utilizarán las librerías ITK [5] y OpenCV [6] para el análisis de imágenes y la visión por computador respectivamente. Haciendo uso de CMake [2] para pre-compilarlas todas juntas.

Antes de empezar a profundizar en aspectos técnicos se realizará una introducción al proceso de obtención de datos volumétricos usando una Tomografía Computarizada así como de los objetos que se quieren analizar con esta técnica de obtención de datos: las esculturas de madera policromada.

1.1. Tomografía Computarizada

1.1.1. Historia

La tomografía computarizada (TC) es una técnica de obtención de imágenes muy utilizado en el campo de la medicina para, por ejemplo, localizar y ver el tamaño de tumores.

Sus orígenes se remontan a los años 60 cuando en 1967 Goodfrey Newblod Hounsfield propuso la elaboración del que llamó escáner EMI, base para desarrollar el Tomógrafo Axial Computarizado (TAC). El objetivo era “*crear una imagen tridimensional de un objeto tomando múltiples mediciones del mismo con la misma fuente de rayos X desde diferentes ángulos y utilizar un ordenador que permita reconstruir a partir de cientos de ‘planos’ superpuestos y entrecruzados*” [29].

Cuatro años más tarde, en 1971, se realizó con éxito el primer escáner cerebral usando este tomógrafo. En 1972 se instaló permanentemente en el hospital donde realizaron las pruebas y al año siguiente ya era solicitado por hospitales alrededor de todo el mundo.

1.1.2. Generaciones

El sistema de tomografía computarizada ha pasado por cuatro generaciones [27]:

Primera generación

La adquisición de datos en la primera generación se basaba en la geometría del haz de rayos X paralelo y traslación-rotación en un tubo de rayos X y un solo detector (Figura 1.1). El haz de rayos X se colimaba en di-

mensiones de 2 x 13mm. Estos 13mm correspondían al grosor del corte. Se tomaba una medida por cada 160 rotaciones durante 180 translaciones dando un total de 28.800 medidas. El proceso era lento, tardaba unos 5 minutos, y el movimiento del paciente afectaba muy negativamente a la calidad de la imagen, por lo que su uso se veía reducido al escaneo de zonas que podían mantenerse inmóviles como la cabeza.

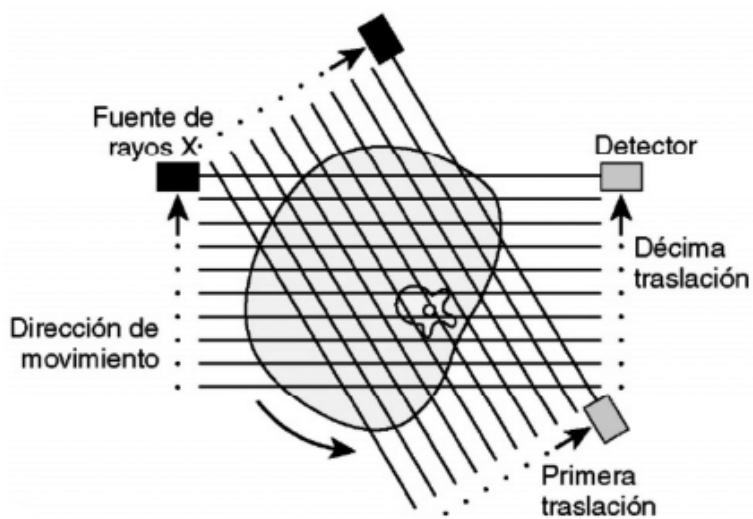


Figura 1.1: Primera generación de aparatos de tomografía computarizada [18]

Segunda generación

En esta segunda generación se aumentó el número de detectores (de 5 a 30) por lo que se vio disminuido el tiempo de la exploración a unos 18 segundos (Figura 1.2).

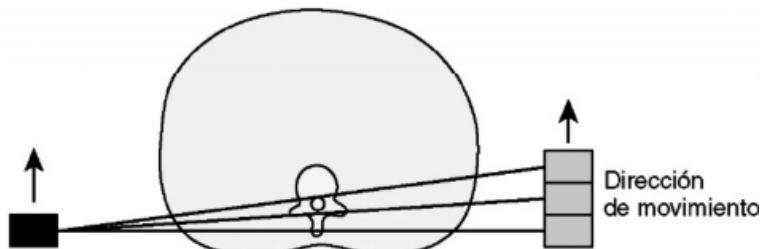


Figura 1.2: Segunda generación de aparatos de tomografía computarizada [18]

Tercera generación

La tercera generación supuso un gran cambio y se ha convertido en la configuración estándar utilizada en la mayoría de los sistemas de escáner. Se utiliza una geometría de haz en abanico de gran angular (50° a 55°), un arco de detectores y un tubo de rayos X. Estos elementos giran 360° alrededor del paciente (Figura 1.3). El número de detectores se encuentra entre 600 y 900. Con este sistema el tiempo de barrido oscila entre 3 y 10 segundos.

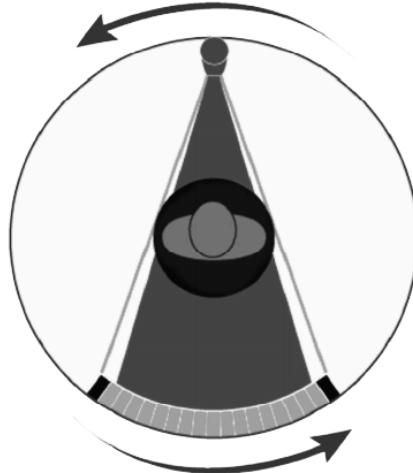


Figura 1.3: Tercera generación de aparatos de tomografía computarizada [18]

Cuarta generación

La cuarta generación es muy parecida a la tercera solo que añade una configuración de giro estacionario (Figura 1.4).

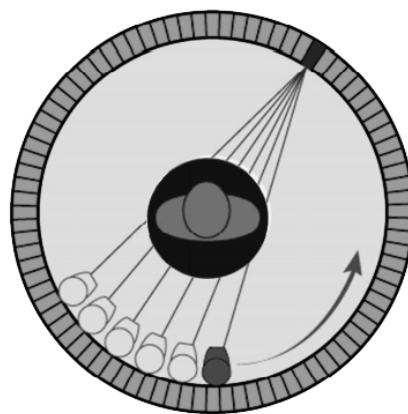


Figura 1.4: Cuarta generación de aparatos de tomografía computarizada [18]

Nuevas tecnologías

- **TC helicoidal (TCH):** Hasta finales de los años 80, los aparatos de TC adquirían los datos en cortes según un método conocido como exploración axial (de ahí el nombre de TAC). Con los sistemas de tipo helicoidal los datos se obtienen de forma continua mientras se avanza la mesa a través del *gantry* haciendo que el tubo de rayos X describa una trayectoria helicoidal (Figura 1.5).

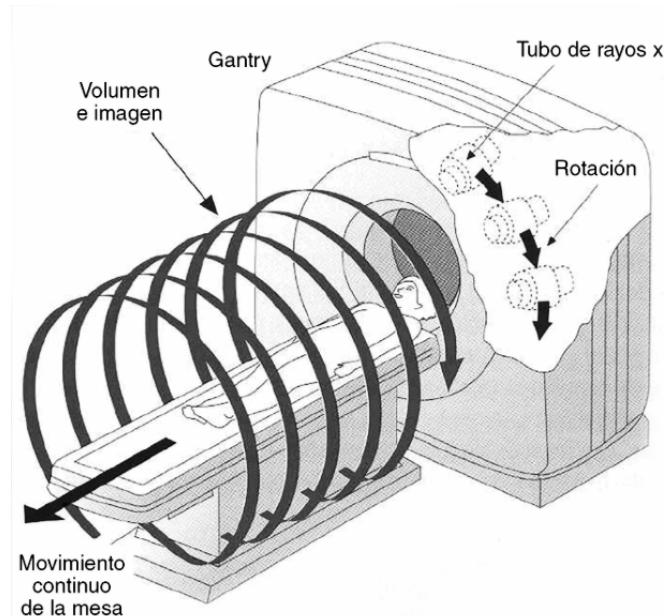


Figura 1.5: Cuarta generación de aparatos de tomografía computarizada [18]

- **TC helicoidal multicorte (TCM):** En el lugar donde había una fila de detectores, se colocan múltiples filas. Los primeros tenían 4 filas contiguas, pero posteriormente se ha pasado a alrededor de 16 y 64 filas (Figura 1.6). Por cada rotación se estudia un mayor volumen aumentando así la velocidad de rotación y por tanto los tiempos de exposición obteniendo imágenes de mayor calidad.

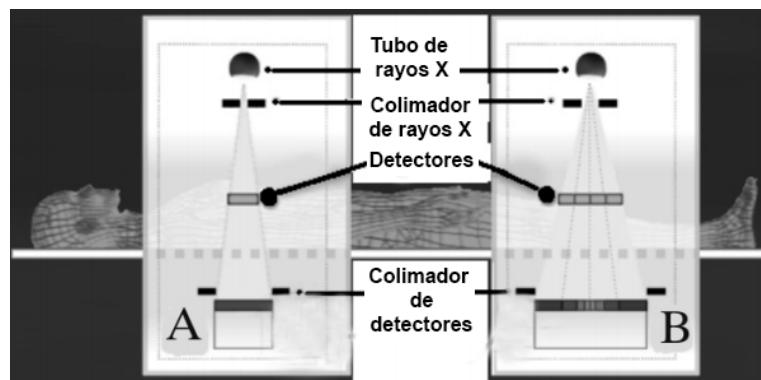


Figura 1.6: Diferencias entre TC helicoidal multicorte (B) y monocorte (A) [27]

- **TC de doble fuente (TCED):** Es uno de los equipos más novedosos pues permiten realizar estudios con diferentes espectros de rayos X. Utilizan dos tubos de rayos X colocados de forma perpendicular en el *gantry* (Figura 1.7). Se obtiene por tanto una resolución temporal equivalente a un cuarto del tiempo de rotación del *gantry*.



Figura 1.7: Equipo TC con doble fuente [27]

1.2. Esculturas de madera policromadas

1.2.1. Historia

El tallado es el método de elaboración de esculturas más antiguo conocido. Se ha tallado en distintos materiales (madera, piedra, marfil...). Pero la madera, por condiciones como su ligereza o la facilidad de ensamblado entre distintas piezas, ha sido uno de los materiales más utilizados.

Se conoce que desde el Antiguo Egipto ya se realizaban esculturas de madera pero es a partir del siglo XI cuando comienza su proliferación. Y desde este momento comienzan a producirse mejoras en las técnicas y herramientas utilizadas durante el proceso del tallado [27].

1.2.2. Maderas más utilizadas

Dependiendo del tipo de escultura se utilizan maderas blandas o duras. Si la escultura es más pequeña y contiene más detalles se utilizará un tipo de madera más dura.

No obstante, en la elección también tiene mucha influencia la situación geográfica al utilizarse maderas autóctonas [27]:

- **Italia:** Álamo y chopo.

- **Francia:** Nogal y castaño.
- **Países Bajos:** Roble y encina.
- **España:** Pino de Flandes, cedro de la Habana, castaño, tejo, álamo, nogal, ciprés, boj, pino silvestre y algunos frutales como el peral.

Además de la madera, una escultura de madera policromada, puede tener varios materiales como el estuco o el metal de los clavos utilizados.

1.2.3. Defectos de la madera

Entre los defectos de la madera se pueden encontrar [27]:

Grietas o fendas

Según la UNE-EN 844-9 se denomina grieta o fenda a “toda separación de las fibras (raja o hendidura) en dirección longitudinal”. Según su origen, pueden ser de distintos tipos:

- **Acebolladuras o colainas:** Hay una discontinuidad entre los anillos de crecimiento.
- **Superficiales o de desecación:** Producidas por el calor, provocan un deterioro en las zonas externas del tronco del árbol dejando la madera desprotegida. Provocan grietas en sentido longitudinal.
- **De heladura:** Producidas por una helada dañan la superficie e interior del tronco. Provocan grietas radiales.
- **De viento:** Originadas por la acción de un fuerte viento. Provocan grietas longitudinales y transversales.

Además de estos procesos naturales, se pueden producir grietas durante procesos como el secado que provocan una separación de las fibras.

Fibras reviradas y entrelazadas

Las fibras se encuentran normalmente orientadas en paralelo al eje principal del tronco, pero en ocasiones pueden presentar nudos que alteran la dirección de éstas.

Nudos

La UNE 56.521 define nudo como “anomalía local de la estructura de la madera producida por la parte inferior de una rama que va quedando englobada en el tronco a medida que se producen los crecimientos de este”. Existen distintos tipos:

- **Adherente, vivo, fijo o sano:** Definido por la UNE 56.521 como “aquel cuyos tejidos son solidarios con los de la madera que los rodea debido a ser formado por una rama viva”.
- **Suelto, saltadizo, muerto o seco:** Definido por la UNE 56.521 como “aquel en que los tejidos de la rama que lo producen no son solidarios con los de la madera que los rodea y suelen separarse”.

Núcleos de resina

Son cavidades entre los anillos de crecimiento producidos frecuentemente por nudos.

Factores de deterioro de tipo biótico

Además de las alteraciones que ya presenta la madera, existen otros factores que también influyen como la humedad y la temperatura o el ataque de insectos xilófagos y hongos.

Los insectos xilófagos se nutren de madera seca y favorecen a su desarrollo una humedad relativa y temperaturas no muy bajas. Estos producen daños rompiendo las fibras de la madera.

Los hongos son microorganismos que pueden desarrollarse en la superficie o en el interior de la madera, haciendo que pierda humedad, reduciendo su tamaño y deformándose. Existen tres tipos distintos de degradación tras un ataque de hongos: pudrición blanca, parda o seca y blanda.

1.2.4. Proceso de tallado

El proceso de tallado se podría definir como una técnica sustractiva en la que a partir de una pieza se obtiene una forma concreta.

Un método puede ser utilizar un único bloque de madera. En ocasiones ahuecado por el reverso para contrarrestar fuerza y movimiento de la madera, y, en cierto modo, aligerando el peso.

A partir del siglo XVI empieza a utilizarse con más frecuencia otro método en el que a partir de un bloque principal se ensamblan diferentes piezas

generando un bloque más grande, denominado embón, con la forma y el tamaño de la imagen a esculpir. A partir de éste se comienza el tallado.

A partir del Barroco se mejora esta técnica realizando ensamblados en hueco para evitar el posterior ahuecado [27].

1.3. Estado del arte

Como ya se ha comentado anteriormente, este trabajo es la continuación del trabajo fin de grado que realicé. Durante este trabajo fin de grado resolví el problema de la visualización de datos volumétricos de esculturas de madera policromada [13]. Pero no realizaba ningún pre-procesamiento, tan solo se visualizaban permitiendo explorar la escultura en su interior. Toda la información que obtenía el experto que usase el programa debía ser anotada a mano a parte pues tampoco se proporcionaba funcionalidad para almacenar esta información para poder ser rescatada posteriormente.

Este trabajo se divide en tres bloques: pre-procesamiento, documentación y segmentación.

Los dos primeros están incluidos en la mayoría de los programas disponibles en la web de visualización de datos volumétricos. Tales como AMILab [20], RadiAnt [8], Slicer [17] o el usado por María Francisca en su estudio [27] OsiriX [26] en el que me he inspirado para realizar mi implementación. Todas estas herramientas son muy completas, pero los resultados obtenidos al visualizar los datos volumétricos de esculturas de madera policromadas dejaban mucho que desear y el gran número de funciones específicas para la medicina los hacen complejos de más, y es que por esta razón surgió la idea de crear un software específico de visualización de datos volumétricos de esculturas.

El tercer bloque, el de la segmentación, es totalmente novedoso pues no hay ningún tipo de segmentación específica para separar distintos trozos de madera. Sin embargo, antes de programar un algoritmo específico habría que probar con los que ya hay disponibles que además son muy usados en el campo de la medicina.

Existen varias aproximaciones para llevar a cabo la segmentación en datos volumétricos. Pueden clasificarse en:

- Segmentación manual
- Segmentación semi-automática
- Segmentación basada en umbrales (*thresholding*)
- Segmentación basada en crecimiento de regiones (*region-growing*)

- Segmentación basada en bordes (*edge*)

La segmentación manual la podríamos descartar automáticamente. Pese a que es un método que siempre se puede aplicar, tener a una persona recortando manualmente las distintas piezas corte por corte puede ser muy costoso en cuanto a tiempo.

Los métodos de segmentación basada en umbrales [25] se basan en definir un umbral en los valores escalares de cada voxel para diferenciar las distintas partes del volumen. Estos no nos son útiles pues las distintas piezas de madera suelen ser del mismo material y coincidir en estos valores de densidad.

Los métodos de segmentación basada en crecimiento de regiones que utilizan umbrales [19] tampoco nos van a servir pues las distintas piezas de madera se encuentran juntas. Otros métodos de segmentación basada en regiones como la transformación divisoria (*watershed*) [11] tampoco nos van a servir pues se definirían muchas cuencas debido a los anillos que presentan los cortes de la madera y si se empiezan a inundar para obtener regiones más grandes obtendremos un resultado similar al que obtenemos con el de crecimiento de regiones con umbrales.

Los métodos de segmentación basados en bordes como el *livewire* [24] podrían resultar efectivos pues el borde que separa dos piezas de madera es visualmente diferenciable. No obstante, este método necesita de una supervisión humana y, aunque sea más preciso y rápido que una segmentación manual, se debería seguir realizando corte a corte.

Como vemos, ninguno de los métodos que no requieren de supervisión humana son efectivos a la hora de resolver nuestro problema. Es por ello que, definitivamente, hay que desarrollar uno propio.

1.4. Motivación

La exploración interna de una escultura mediante los datos obtenidos con una TC 1.8 son mucho más completos que los obtenidos con otras técnicas como las radiografías tradicionales [27]. La necesidad de una herramienta con la que poder visualizar estos datos se vio satisfecha con 3DCurator [13], sin embargo, se trataba de un software todavía incompleto pues no contaba con técnicas de pre-procesamiento ni documentación como otros software del mercado especializados en visualizar datos médicos.

La extracción de los distintos trozos de madera que forman parte de la escultura podrán ayudar enormemente a su estudio por separado además de poder ser extraídos en un formato como el STL para ser imprimidos en 3D.



Figura 1.8: Obtención de datos volumétricos de una escultura posteriormente examinada

Capítulo 2

Especificación de requisitos

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

2.1. Introducción

2.1.1. Propósito

Este capítulo de especificación de requisitos tiene como objetivo definir las especificaciones funcionales y no funcionales para el desarrollo de un software que permitirá pre-procesar datos DICOM obtenidos al someter a una escultura de madera policromada a una TC para que posteriormente los usuarios puedan realizar labores de documentación al explorar la figura así como realizar una segmentación de los distintos trozos de madera que la componen.

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. Con este software llamado 3DCurator, se tratará de trasladar esta técnica al campo de la restauración de bienes culturales y poder pre-procesar, visualizar, interactuar y documentar los datos DICOM obtenidos con esculturas de madera policromada.

2.1.3. Definiciones, acrónimos y abreviaturas

- **ERS** (Especificación de Requisitos Software).

- **GUI** (Interfaz Gráfica de usuario).
- **Widget**: Elemento de la GUI.
- **DICOM** (*Digital Imaging and Communication in Medicine*): Formato de datos volumétricos donde se obtienen las imágenes.
- **CT o TC** (Tomografía Computarizada): Técnica de extracción de datos volumétricos que utiliza radiación X para obtener cortes de objetos.
- **MRI o IRM** (Imagen por Resonancia Magnética): Técnica de extracción de datos volumétricos que utiliza el fenómeno de resonancia magnética nuclear.
- **PET** (Tomografía por Emisión de Positrones): Técnica de extracción de datos volumétricos capaz de medir la actividad metabólica del cuerpo humano.
- **CPU** (*Graphic Processor Unit*): Microprocesador.
- **GPU** (*Graphic Processor Unit*): Tarjeta gráfica.
- **UML** (Lenguaje unificado de modelado): Lenguaje de modelado de sistemas software.
- **Historia de usuario**: Representación de un requisito utilizando el lenguaje común del usuario.
- **Product Backlog**: Listado de historias de usuario del proyecto.
- **C++**: Lenguaje de programación que se usará.
- **XML** (*Extension Markup Language*): Meta lenguaje que se usará para exportar ficheros que después podrán ser importados.
- **CMake** (*Cross platform Make*): Herramienta para generar código compilable en distintas plataformas.
- **Qt**: Librería que se utilizará para realizar la GUI.
- **VTK** (*The Visualization ToolKit*): Librería gráfica que se utilizará para la visualización de volúmenes.
- **ITK** (*Insight Segmentation and Registration Toolkit*): Librería de procesamiento de imágenes que se utilizará.
- **OpenCV**: Librería de visión por computador que se utilizará.
- **Boost**: Conjunto de algoritmos para C++ de los que se usarán la gestión de ficheros XML.

- **Volumen:** Conjunto de datos en los que para cada posición XYZ se tiene un valor determinado.
- **Voxel (Volumetric Pixel):** Celda en la matriz 3D del conjunto de datos del volumen.
- **Vecinadario:** Celdas alrededor de un voxel.
- **Adyacencia de voxels:** Voxels vecinos que satisfacen un criterio de similitud.
- **Conectividad de voxels:** Voxels entre los cuales hay un camino de voxels adyacentes.
- **Malla:** Estructura de datos con la información de una superficie tridimensional.
- **STL (Standard Triangle Language):** Formato que define mallas 3D.
- **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 volumen.
- **ROD (Región de documentación):** Corte en el que se documentará.
- **Regla:** Widget utilizado para realizar mediciones de distancias.
- **Transportador de ángulos:** Widget utilizado para realizar mediciones de ángulos.
- **Nota:** Widget utilizado para realizar anotaciones sobre la figura.

- **Segmentación:** Método por el cual se distinguen distintas partes del volumen.
- **Thresholding:** Segmentación basada en umbrales.
- **Region-growing:** Segmentación basada en crecimiento de regiones.
- **Semilla:** Coordenada donde se comienza el *region-growing*.
- **Watershed:** Técnica de segmentación basada en crecimiento de regiones por inundación.
- **Canny Edge Detection:** Algoritmo para resaltar los bordes de una imagen.
- **Transformada de Hough:** Técnica utilizada para detectar líneas rectas.
- **Filtro gaussiano:** Filtro que utiliza la distribución gaussiana del vecindario.
- **Filtro media:** Filtro que utiliza la media del vecindario.
- **Filtro mediana:** Filtro que utiliza la mediana del vecindario.
- **Embón:** Ensamblado de distintos trozos de madera que hacen de base para la escultura.
- **Estuco:** Pasta de grano fino utilizada para realizar correcciones en la escultura.
- **Policromía:** Capa de pintura de las esculturas.
- **Blanco de plomo:** Pigmento de color blanco creado a partir del plomo.
- **Pan de oro:** Lámina muy fina de oro.

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, se podrán pre-procesar aplicando una serie de filtros, segmentar en los distintos trozos de madera que forman la escultura y documentar añadiendo notas y mediciones de distancias y ángulos.

2.2.2. Funciones del producto

Las principales funcionalidades de este sistema serán:

- Pre-procesar la imagen aplicando filtros de reducción de ruido:
 - Media.
 - Mediana.
 - Gaussiano.
- Documentar la escultura:
 - Agregar anotación.
 - Agregar regla.
 - Agregar transportador de ángulos.
- Segmentar la escultura por los distintos trozos de madera.
- Exportar el volumen.
- Importar el volumen.

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 sobre un software de visualización de datos volumétricos de esculturas de madera ya existente.

Se programará en C++ usando las librerías VTK para la visualización de gráficos, ITK para el filtrado de imágenes, OpenCV para la visión por computador, Boost para la gestión de ficheros XML y Qt para la GUI.

Al usar estas librerías se contará con restricciones funcionales con las que éstas cuentan que se solventarán reescribiendo el código que sea posible y necesario.

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 interactuar con esculturas de madera por lo que se tendrán en cuenta los materiales con los que están hechas la mayoría de estas (madera, estuco y metal). Si se introducen los datos DICOM de cualquier otra cosa con materiales distintos a los utilizados en las esculturas no se visualizará correctamente. Sin embargo se podrá editar la función de transferencia para su correcta visualización.

La única funcionalidad que se vería afectada sería la segmentación en las distintas piezas de madera que la forman pues usaría como dato el valor escalar en HU de la madera.

2.3. Requisitos específicos

2.3.1. Interfaces

Se adaptará la interfaz actual de 3DCurator para agregar las nuevas funcionalidades (Figura 2.1):

- Se crearán botones para importar y exportar volumen, segmentar y aplicar filtro. Éste último lanzará una ventana modal (Figura 2.2) donde se seleccionará el filtro y parámetros deseados.
- A la derecha se colocará la parte de documentación con la que se podrán agregar y eliminar ROD, reglas, transportadores de ángulos y anotaciones.

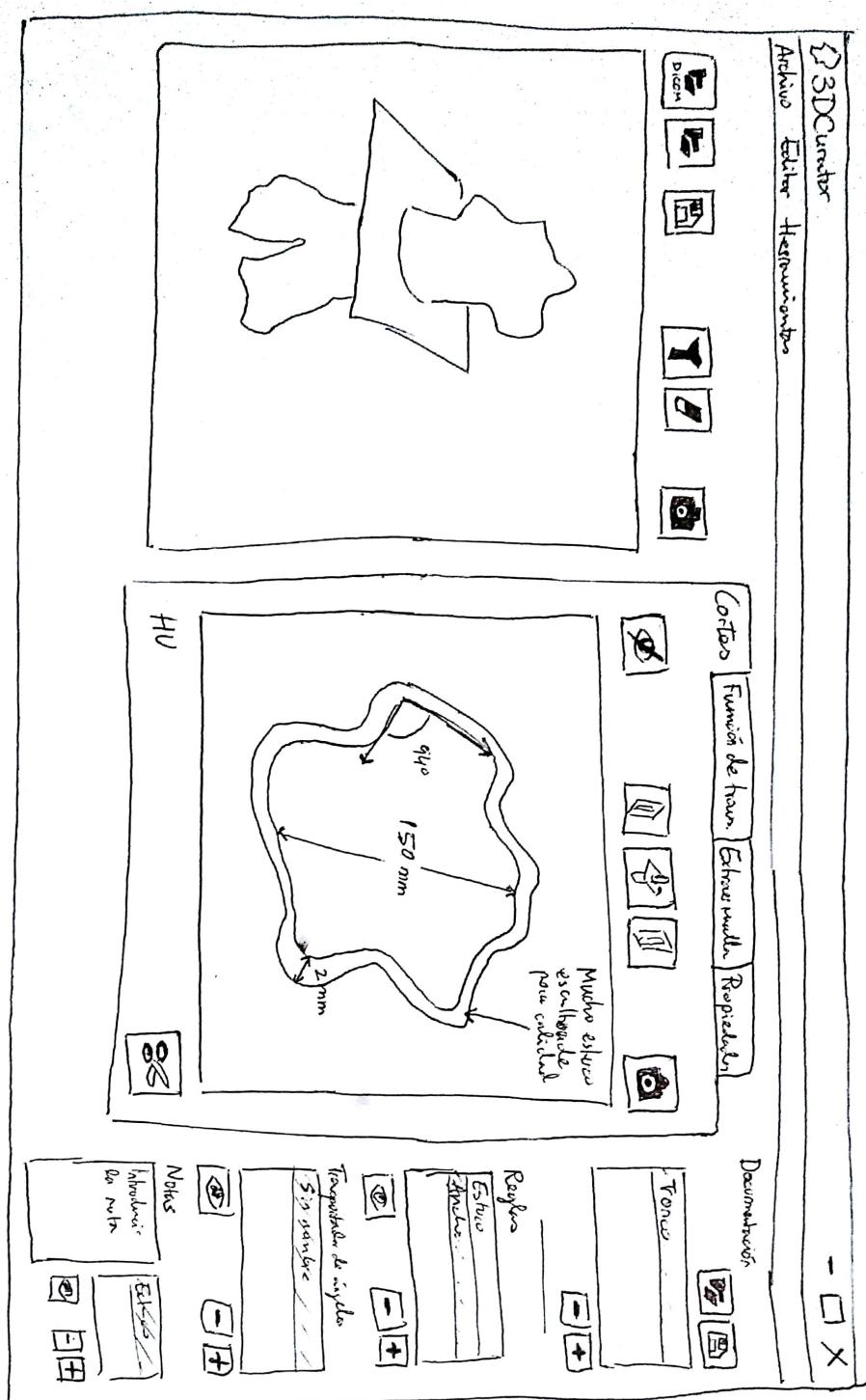


Figura 2.1: Boceto a mano alzada de la posible distribución de los elementos en la GUI principal

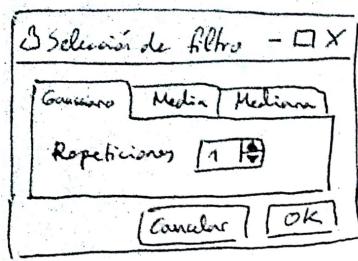


Figura 2.2: Boceto a mano alzada de la posible distribución de los elementos en la GUI de selección de filtros

2.3.2. Funciones

El sistema tendrá que realizar distintas funciones que se comentaron anteriormente pero se profundizará en esta sección. Estas funciones se han estructurado por módulo.

Auxiliar

- **Exportar volumen:** El usuario seleccionará la carpeta y el nombre donde deseará guardar el volumen que está visualizando.
- **Importar volumen:** En lugar de cargar directamente los datos DICOM, el usuario podrá cargar el volumen que haya exportado anteriormente el cual puede tener aplicado los filtros de pre-procesamiento o segmentación.

Pre-procesamiento

- **Filtro gaussiano:** El usuario seleccionará el número de repeticiones que desea aplicar y se aplicará el filtro al volumen cargado.
- **Filtro media:** El usuario seleccionará el tamaño del vecindario y se aplicará el filtro al volumen cargado.
- **Filtro mediana:** El usuario seleccionará el tamaño del vecindario y se aplicará el filtro al volumen cargado.

Segmentación

- **Segmentar:** Desde un corte, el usuario seleccionará la pieza de madera que desea segmentar. Una vez realizado este proceso se mostrará en pantalla el volumen segmentado y se dará la posibilidad de exportarlo.

Documentación

- **Crear ROD:** El usuario podrá agregar un ROD al que podrá dar un nombre.
- **Eliminar ROD:** El usuario podrá eliminar un ROD seleccionado.
- **Exportar ROD:** El usuario podrá exportar el ROD seleccionado.
- **Importar ROD:** El usuario podrá importar un ROD previamente exportado.
- **Cambiar ROD activo:** El usuario podrá cambiar de ROD y se cargará la posición del plano con los elementos de éste. Si el usuario mueve el plano se dejará de visualizar el ROD actual.
- **Crear regla:** El usuario podrá agregar una regla a la que podrá dar un nombre.
- **Eliminar regla:** El usuario podrá eliminar una regla seleccionada.
- **Ocultar regla:** El usuario podrá ocultar una regla seleccionada.
- **Mostrar regla:** El usuario podrá ocultar una regla seleccionada oculta.
- **Cambiar regla activa:** El usuario podrá cambiar la regla activa.
- **Crear transportador de ángulos:** El usuario podrá agregar un transportador de ángulos al que podrá dar un nombre.
- **Eliminar transportador de ángulos:** El usuario podrá eliminar un transportador de ángulos seleccionado.
- **Ocultar transportador de ángulos:** El usuario podrá ocultar un transportador de ángulos seleccionado.
- **Mostrar transportador de ángulos:** El usuario podrá ocultar un transportador de ángulos seleccionado oculto.
- **Cambiar transportador de ángulos activa:** El usuario podrá cambiar el transportador de ángulos activo.
- **Crear nota:** El usuario podrá agregar una nota a la que podrá dar un nombre y agregar un texto.
- **Eliminar nota:** El usuario podrá eliminar una nota seleccionada.
- **Ocultar nota:** El usuario podrá ocultar una nota seleccionada.
- **Mostrar nota:** El usuario podrá ocultar una nota seleccionada oculta.
- **Cambiar nota activa:** El usuario podrá cambiar la nota activa.

2.3.3. Requisitos de rendimiento

Al ser una aplicación de escritorio donde se mostrará y se interactuará con datos, 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 todos los volúmenes que se han estado visualizando, desecharando el anterior cuando se carga uno nuevo.

El rendimiento gráfico también es importante, por eso y para obtener imágenes de mayor calidad, se utilizarán técnicas de DVR como el *ray casting*. Estas técnicas necesitan una gran cantidad de procesamiento, pero la velocidad de procesamiento de las GPU actuales no deberían resultar un problema.

En cuanto a la segmentación que es la función más compleja y costosa computacionalmente, habrá que tener especial cuidado con la gestión de memoria intentando usar los almacenes de datos más adecuados en cada caso.

2.3.4. Restricciones de diseño

Al utilizar librerías como VTK, ITK, OpenCV o Boost, se seguirá la estructura de las mismas a la hora de construir el software, y se tendrán restricciones en cuanto a funcionalidad que se pueda construir con éstas. No obstante son librerías muy completas y no se debería encontrar casi ninguna restricción a excepción de la parte del pre-procesamiento y segmentación donde quizás haya que generar algoritmos propios.

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, en el futuro, otros desarrolladores pueden colaborar y debe estar bien documentado para que esto sea una tarea fácil.

Se tratará de un software de código abierto que se alojará en un repositorio de GitHub.

Capítulo 3

Planificación

En este capítulo se comentará la planificación inicial de tiempo en la que se llevará a cabo este TFM, la estimación en horas de cada módulo de tareas, la metodología de trabajo y los resultados.

3.1. Planificación inicial

La fecha de inicio trabajando en el proyecto que me fijé es el 1 de diciembre de 2016 y se espera llegar a la convocatoria de julio de 2017 por lo que se cuentan con siete meses. No obstante, y como es lógico, no voy a trabajar exclusivamente en este proyecto durante estos meses pues tengo que cursar el resto de asignaturas del máster, además debería tener un mes de margen para pulir la memoria desarrollando algunas secciones y organizando lo que haya ido apuntando durante el resto de meses. Por lo que serían 6 meses de desarrollo.

Para organizarme he decidido dedicar un número de horas semanales al desarrollo de este proyecto. Inicialmente me he fijado 15 horas a la semana dando un total de 360 horas de desarrollo puro excluyendo la redacción de la memoria.

Ordenando los módulos por prioridades y dependencias se tendría la siguiente planificación:

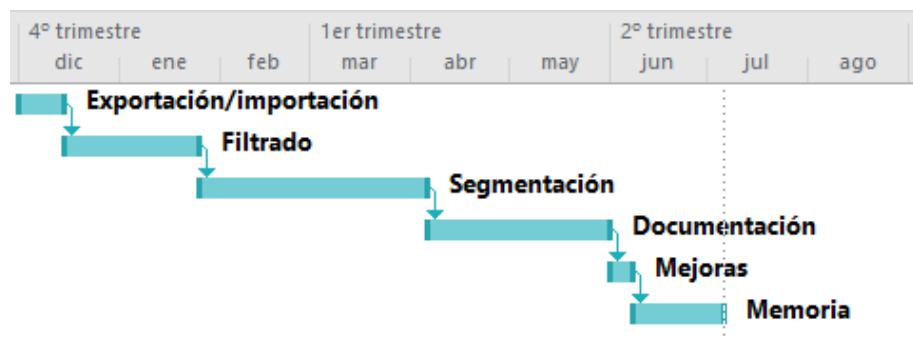
Tarea	Duración	Comienzo	Fin
Exportación/importación	2 semanas	01/12/2016	14/12/2016
Filtrado	6 semanas	15/12/2016	25/01/2017
Segmentación	10 semanas	26/01/2017	05/04/2017
Documentación	8 semanas	06/04/2017	31/05/2017
Mejoras	1 semana	01/06/2017	07/06/2017

Tarea	Duración	Comienzo	Fin
Memoria	4 semanas	08/06/2017	05/07/2017

Cuadro 3.1: Planificación inicial

3.1.1. Diagrama de Gantt

Aunque el diagrama de Gantt tiene más sentido cuando se pueden paralelizar tareas y al haber un único desarrollador esto no es posible, realicé este diagrama porque de un vistazo se ve más clara la planificación que con la tabla anterior.

Figura 3.1: Diagrama de Gantt con la planificación inicial del proyecto generado con *Microsoft Office Project 2016*

3.2. Metodología de trabajo

3.2.1. GitHub

Aprovechando que se está usando *git* como sistema de control de versiones en un repositorio de *GitHub*, voy a aprovechar todos los recursos que nos ofrece esta plataforma para organizarme.

3.2.2. Issues

En lugar de tener por un lado un tablero *kanban* (por ejemplo *Trello*), yo voy a utilizar los *issues* (Figura 3.2) de mi repositorio en *GitHub* para organizar las tareas, así como ir añadiendo tareas nuevas que vaya necesitando y tener un registro de qué he hecho y cuándo, que problemas han surgido, etc.

The screenshot shows a GitHub repository page for 'fblupi / 3DCurator'. The top navigation bar includes links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below the navigation, there are buttons for 'Unwatch' (1), 'Star' (3), 'Fork' (0), and a user icon. The main content area displays a list of 11 open issues. Each issue is represented by a row with a checkbox, a title, labels, and a small profile picture. The issues are as follows:

- Change from QVTKWidget to QVTKOpenGLWidget [enhancement]
- Set-up environment for Mac OS X [enhancement] [resources needed]
- Planes detection [enhancement]
- Lines detection [enhancement]
- Special characters in filenames [bug]
- 3D Region Growing [enhancement] [question]
- Show degree symbol in protractor [enhancement] [help wanted]
- Show histogram in color and scalar opacity charts [enhancement] [help wanted]
- Segmentation using more than one line [enhancement]
- Parse plane coordinates into volume coordinates [enhancement] [help wanted]
- Set-up environment for Linux [enhancement] [help wanted]

At the bottom of the list, there are filters for 'Issues' (11), 'Pull requests' (0), 'Projects' (0), 'Wiki', 'Insights', and 'Settings'. A 'New issue' button is located at the top right of the list.

Figura 3.2: Vista con la lista de *issues* abiertos con sus correspondientes etiquetas y el *milestone* al que pertenecen

Una de las ventajas de utilizar los *issues* de *GitHub* antes que otra plataforma como *Trello* es la posibilidad de referenciarlos en los *commit* agregando `#ref` en el mensaje. Y dentro del propio *issue* se vería una lista de todos los *commit* donde se ha trabajado en éste. Además, si antes de escribir la referencia del *issue* se añade *closes*, el *issue* se cerrará automáticamente sin necesidad de hacerlo de forma manual.

3.2.3. *Milestones*

Voy a clasificar los *issues* por *milestones* (Figura 3.3), uno por cada módulo y al mismo tiempo voy a agregarles *tags* para etiquetarlos según sean *bugs*, mejoras, bloqueantes por necesidad de recursos o por ayuda necesitada, etc.

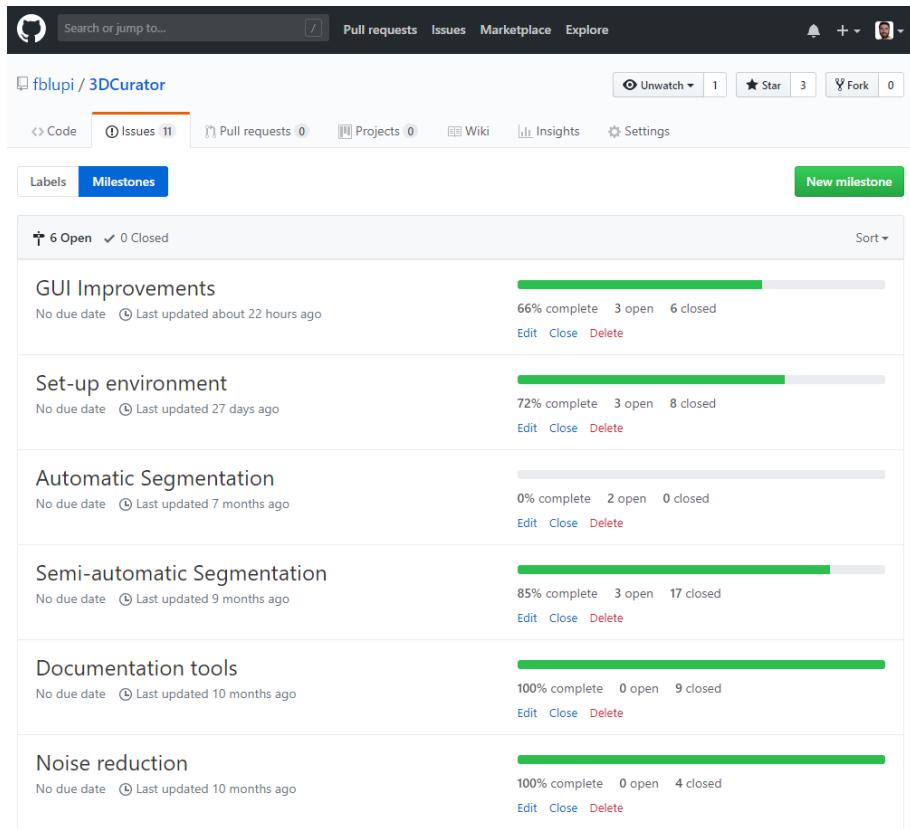


Figura 3.3: Resumen de los *milestones* con su porcentaje de elaboración

3.2.4. *Branches*

La organización que voy a llevar en cuanto a las ramas (*branches*), es la siguiente, basada en *Gitflow* [4]:

- **master**: Esta rama tendrá la última *release* del programa. Debe ser completamente estable, pues es de donde se generan los instaladores finales de la aplicación.
- **develop**: Esta es la rama principal con los últimos cambios desarrollados (Figura 3.4). Cuando se tiene una batería de cambios suficiente como para generar una nueva versión del *software* se realizará un *pull request* a *master* por lo que esta rama debe ser también estable.
- **features**: Estas son ramas de trabajo que no tienen por qué ser estables. Por ejemplo, si se va a agregar un filtro *gaussiano* se podría crear una rama con el nombre *feature/gaussian-filter* y trabajar en esta rama hasta que se validase esta nueva funcionalidad. Entonces se

haría un *rebase* con *develop* por si ha habido algún cambio durante el transcurso de este desarrollo para resolver conflictos y finalmente un *pull request* a *develop* para mezclar los cambios.

Ademásuento con otras ramas auxiliares que no intervienen en el flujo de desarrollo pero donde almaceno información útil.

- **design:** En esta rama se encuentran los diagramas de paquetes y clases actualizados con la versión del software en la rama *develop*.
- **assets:** En esta rama se almacenan recursos como logos e imágenes utilizados en la aplicación.
- **user-manual:** En esta rama se encuentra el manual de usuario de la aplicación.
- **gh-pages:** En esta rama se encuentra la *landing page* de la aplicación. Pues *GitHub* ofrece gratuitamente alojamiento para una página web estática por repositorio.

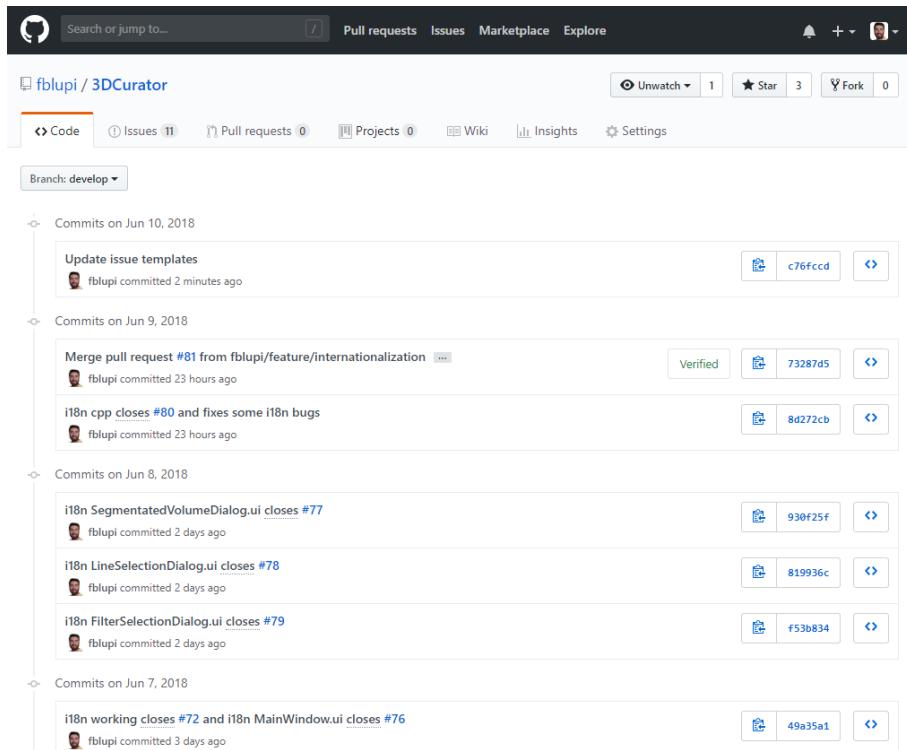


Figura 3.4: Ejemplo de *commits* en la rama *develop*

3.3. Resultados

Los resultados no tienen nada que ver con la planificación inicial que se hizo. La carga del trabajo del resto de asignaturas del máster han hecho que le dedicase muchas menos horas a la semana que las que tenía pensado y durante algunos periodos lo he tenido que tener apartado.

Cuando llegó junio y vi que no llegaba a la convocatoria, recalculeé tiempos para llegar a septiembre. Pero otra vez se vieron trastocados pues en agosto empecé a trabajar a jornada completa y las horas que le pude echar se volvieron a ver reducidas. No obstante llegué a tenerlo todo a excepción de la documentación. Que ya, una vez matriculado de nuevo en la asignatura la tomé con calma.

Además, me permití el lujo de añadir a última hora internacionalización al proyecto para poder tener la aplicación también en inglés además de la versión ya existente en español.



Figura 3.5: Gráfico de los *commits* realizados en la rama *develop*. El gráfico es orientativo porque hay *commits* con poca funcionalidad y otros con mucha, pero ayuda a ver las etapas donde más se trabajó

En total he registrado 312 horas de desarrollo frente a las 360 planificadas. Esto quiere decir que la planificación de tareas no ha fallado como tal. La que ha fallado ha sido la planificación de disponibilidad de recursos humanos.

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*

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

#	Descripción	Dep.	Est.	Prio.
Auxiliar				
1.1.	Exportar volumen	-	12	1
1.2.	Importar volumen	1.1.	8	5
Pre-procesamiento				
2.1.	Filtro <i>gaussiano</i>	-	24	1
2.2.	Filtro media	-	10	3
2.3.	Filtro mediana	-	8	3
Segmentación				
3.1.	Segmentar pieza de madera	1.1.	64	1
Documentación				
4.1.	Crear ROD	-	4	1
4.2.	Eliminar ROD	4.1.	4	3
4.3.	Exportar ROD	4.1.	2	2
4.4.	Importar ROD	4.3.	3	2
4.5.	Cambiar ROD	4.1.	3	1
4.6.	Crear regla	-	3	2

#	Descripción	Dep.	Est.	Prio.
4.7.	Eliminar regla	4.6	3	4
4.8.	Editar regla	4.6	1	5
4.9.	Ocultar regla	4.6	2	6
4.10.	Mostrar regla	4.9	1	6
4.11.	Crear transportador de ángulos	-	3	3
4.12.	Eliminar transportador de ángulos	4.11.	3	5
4.13.	Editar transportador de ángulos	4.11.	1	6
4.14.	Ocultar transportador de ángulos	4.11.	2	7
4.15.	Mostrar transportador de ángulos	4.14.	1	7
4.16.	Crear nota	-	4	2
4.17.	Eliminar nota	4.16.	2	4
4.18.	Editar nota	4.16.	3	5
4.19.	Ocultar nota	4.16.	2	6
4.20.	Mostrar nota	4.19.	1	6
Mejoras en código				
5.1.	Internacionalización	-	6	8

Cuadro 4.1: Historias de usuario

Se han estimado un total de 180 PH (Puntos de historia) y se habían planificado unas 360 horas de trabajo. Por lo que cada PH corresponde aproximadamente a 2 horas.

Al haber registrado 312 horas finalmente, he determinado que mi velocidad ha sido aproximadamente 1,73 horas/PH, un poco más rápido que lo planificado.

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.

Auxiliar

1.1.	Exportar volumen
Descripción	
Se debe proveer a la aplicación de la funcionalidad necesaria para exportar el volumen que se ha cargado y editado para poder utilizarlo de nuevo posteriormente sin tener que volver a editarlos. Para ello se tratará de utilizar el formato propio de VTK para almacenar volúmenes, VTI, que hace uso de XML. Se le mostrará al usuario un diálogo donde elegirá la carpeta y el nombre del archivo.	
Estimación	12
Prioridad	1
Dependencias	-
Pruebas de aceptación	
<ul style="list-style-type: none"> - El usuario realiza la acción de exportar sin ningún volumen cargado y le aparece un mensaje para que lo cargue antes. - El usuario no elige nombre y se guardará el archivo con un nombre por defecto en la carpeta elegida. - El usuario elige el nombre y se guardará el archivo con el nombre elegido en la carpeta elegida. 	

Cuadro 4.2: Historia de usuario - Exportar volumen

1.2.	Importar volumen
Descripción	
El <i>software</i> debe poder leer ficheros en formato VTI como los previamente exportados. Para ello el usuario deberá poder elegir un archivo en un diálogo donde se filtrarán los ficheros para que se muestren solo los que tienen la extensión VTI.	
Estimación	8
Prioridad	5
Dependencias	1.1.
Pruebas de aceptación	
<ul style="list-style-type: none"> - Si el usuario lee un archivo que no es VTI no lo podrá visualizar. - Si el usuario elige un archivo VTI correcto se importará y podrá empezar a utilizarlo. 	

Cuadro 4.3: Historia de usuario - Importar volumen

Pre-procesamiento

2.1.	Filtro <i>gaussiano</i>
Descripción	
Se debe poder aplicar un filtro <i>gaussiano</i> al volumen para reducir el ruido. Para ello el usuario elegirá el número de repeticiones que se realizarán (de 1 a 5). Se utilizará la librería ITK para aplicar este filtro.	
Estimación	24
Prioridad	1
Dependencias	-
Pruebas de aceptación	
<ul style="list-style-type: none"> - El usuario filtra sin haber ningún volumen cargado y le aparece un mensaje de que cargue antes un modelo. - El usuario selecciona el filtrado y los parámetros deseados y se realiza el filtrado en el volumen, el resultado será un volumen más suavizado. 	

Cuadro 4.4: Historia de usuario - Filtro *gaussiano*

2.2.	Filtro media
Descripción	
Se debe poder aplicar un filtro media al volumen para reducir ruido. Para ello el usuario elegirá el tamaño del vecindario (3x3, 5x5 o 7x7). Se utilizará la librería ITK para aplicar este filtro.	
Estimación	10
Prioridad	3
Dependencias	-
Pruebas de aceptación	
<ul style="list-style-type: none"> - El usuario filtra sin haber ningún volumen cargado y le aparece un mensaje de que cargue antes un modelo. - El usuario selecciona el filtrado y los parámetros deseados y se realiza el filtrado en el volumen, el resultado será un volumen más suavizado. 	

Cuadro 4.5: Historia de usuario - Filtro media

2.3.	Filtro mediana
Descripción	
Se debe poder aplicar un filtro mediana al volumen para reducir ruido. Para ello el usuario elegirá el tamaño del vecindario (3x3, 5x5 o 7x7). Se utilizará la librería ITK para aplicar este filtro.	
Estimación	8
Prioridad	3
Dependencias	-
Pruebas de aceptación	
<ul style="list-style-type: none"> - El usuario filtra sin haber ningún volumen cargado y le aparece un mensaje de que cargue antes un modelo. - El usuario selecciona el filtrado y los parámetros deseados y se realiza el filtrado en el volumen. Esto reducirá el ruido de tipo <i>salt-and-pepper</i>. 	

Cuadro 4.6: Historia de usuario - Filtro mediana

Segmentación

3.1.	Segmentar pieza de madera
Descripción	
El usuario debe poder separar las distintas piezas de madera de las que está compuesta la figura. Para ello se desarrollará un método semi-automático guiado por el usuario para partir en dos un volumen, de forma iterativa se podría realizar la descomposición total de la figura. El usuario elegirá en el visor de corte el trozo deseado a segmentar y el sistema le preguntará por cuál de las posibles líneas detectadas divide las piezas de madera, el usuario le responderá y el sistema terminará el proceso. Para realizar esto se deberán combinar filtros, técnicas de visión por computador y de segmentación de volúmenes.	
Estimación	64
Prioridad	1
Dependencias	1.1.
Pruebas de aceptación	
<ul style="list-style-type: none"> - El usuario elige una pieza de madera y la línea que la separa y se devolverá un volumen con esta pieza. 	

Cuadro 4.7: Historia de usuario - Segmentar pieza de madera

Documentación

4.1.	Crear ROD
Descripción	
Se debe poder crear un área de trabajo de documentación, es decir, imágenes donde incluir anotaciones. Para ello se ha creado el concepto ROD (Region of Documentation), que no es más que una posición del plano de corte donde se podrán incluir medidas y anotaciones. El usuario colocará el plano en la posición deseada y creará una ROD a la que podrá dar nombre, si no, cogerá un nombre por defecto.	
Estimación	4
Prioridad	1
Dependencias	-
Pruebas de aceptación	
<ul style="list-style-type: none"> - El usuario crea una ROD sin nombre y se guarda la posición del plano con un nombre por defecto. - El usuario crea una ROD con nombre y se guarda la posición del plano con el nombre indicado. 	

Cuadro 4.8: Historia de usuario - Crear ROD

4.2.	Eliminar ROD
Descripción	
El usuario debe poder eliminar una ROD creada con anterioridad. Para ello la selecciona y pulsa en la opción de eliminar.	
Estimación	4
Prioridad	3
Dependencias	4.1.
Pruebas de aceptación	
<ul style="list-style-type: none"> - No hay ninguna ROD seleccionada, el usuario pulsa en eliminar y le aparece un mensaje de que debe seleccionar antes una. - El usuario selecciona una ROD, la elimina y desaparece del listado de ROD. 	

Cuadro 4.9: Historia de usuario - Eliminar ROD

4.3.	Exportar ROD
Descripción	
Se tiene que proveer al sistema de la funcionalidad necesaria para poder exportar una ROD con todos sus componentes de forma que pueda volver a importarlo. Para ello se exportará en formato XML.	
Estimación 2	
Prioridad 2	
Dependencias 4.1.	
Pruebas de aceptación	
- No hay ninguna ROD seleccionada, el usuario pulsa en exportar y le aparece un mensaje de que debe seleccionar antes una.	
- El usuario exporta una ROD seleccionada, elige el nombre, que por defecto es su nombre y se guarda correctamente en formato XML.	

Cuadro 4.10: Historia de usuario - Exportar ROD

4.4.	Importar ROD
Descripción	
Se debe poder leer un archivo XML con un formato específico con información de una ROD y sus componentes. Para ello se le mostrará al usuario un cuadro de diálogo donde explorará entre sus archivos, realizando un filtrado para mostrar solo los ficheros XML	
Estimación 3	
Prioridad 2	
Dependencias 4.3.	
Pruebas de aceptación	
- El usuario carga un archivo con el formato válido y se importa la ROD con todos sus componentes.	

Cuadro 4.11: Historia de usuario - Importar ROD

4.5.	Cambiar ROD						
Descripción							
El usuario debe poder cambiar la ROD activa seleccionándola dentro de la lista de ROD. Esto cambiará automáticamente la posición del plano para usar la de esta ROD.							
<table border="1"> <tr> <td>Estimación</td><td>3</td></tr> <tr> <td>Prioridad</td><td>1</td></tr> <tr> <td>Dependencias</td><td>4.1.</td></tr> </table>		Estimación	3	Prioridad	1	Dependencias	4.1.
Estimación	3						
Prioridad	1						
Dependencias	4.1.						
Pruebas de aceptación							
<ul style="list-style-type: none"> - El usuario selecciona la misma ROD que está activa y no cambia. - El usuario selecciona una ROD que no está activa y cambia la posición del plano y se muestran los elementos de esa ROD. 							

Cuadro 4.12: Historia de usuario - Cambiar ROD

4.6.	Crear regla						
Descripción							
El sistema debe proveer de una funcionalidad para poder realizar medidas a tamaño real de zonas en un corte. Para ello el usuario solo tendrá que seleccionar dos puntos y le aparecerá la medida en milímetros. Se hará uso de un <i>widget</i> de VTK para ello.							
<table border="1"> <tr> <td>Estimación</td><td>3</td></tr> <tr> <td>Prioridad</td><td>2</td></tr> <tr> <td>Dependencias</td><td>-</td></tr> </table>		Estimación	3	Prioridad	2	Dependencias	-
Estimación	3						
Prioridad	2						
Dependencias	-						
Pruebas de aceptación							
<ul style="list-style-type: none"> - El usuario crea una regla sin seleccionar ninguna ROD y se le informa de que seleccione una. - El usuario selecciona dos puntos y le aparecerá la medida entre ambos puntos. 							

Cuadro 4.13: Historia de usuario - Crear regla

4.7.	Eliminar regla
Descripción	
Se deben poder eliminar medidas de distancias de la pantalla. Para ello el usuario solo tiene que seleccionarla y pulsar en eliminar.	
Estimación	3
Prioridad	4
Dependencias	4.6.
Pruebas de aceptación	
<ul style="list-style-type: none"> - El usuario realiza la acción de eliminar una regla sin que haya ninguna y se le mostrará un mensaje indicando que antes seleccione una. - El usuario selecciona una regla, la elimina y desaparece de la lista y de la vista. 	

Cuadro 4.14: Historia de usuario - Eliminar regla

4.8.	Editar regla
Descripción	
El usuario debe ser capaz de editar una medida por si se ha equivocado. Para ello selecciona y arrastra uno de los puntos de ésta.	
Estimación	1
Prioridad	5
Dependencias	4.6.
Pruebas de aceptación	
<ul style="list-style-type: none"> - El usuario selecciona y arrastra un punto y se actualiza la posición de éste y el valor de la medida de la distancia entre ambos. 	

Cuadro 4.15: Historia de usuario - Editar regla

4.9.	Ocultar regla
Descripción	
El usuario debe ser capaz de ocultar una regla de la vista sin eliminarla. Para ello la selecciona en la lista y pulsa en ocultar. Se mostrará de forma distinta en la lista de reglas para que se tenga claro que se ha ocultado.	
Estimación	2
Prioridad	6
Dependencias	4.6.
Pruebas de aceptación	
<ul style="list-style-type: none"> - El usuario realiza la acción de ocultar una regla sin que haya ninguna y se le mostrará un mensaje indicando que antes seleccione una. - El usuario selecciona una regla, la oculta y desaparece de la vista, pero no de la lista. 	

Cuadro 4.16: Historia de usuario - Ocultar regla

4.10.	Mostrar regla
Descripción	
El usuario debe ser capaz de mostrar una regla que previamente ha ocultado. Para ello la selecciona en la lista y pulsa en mostrar. Se volverá a mostrar como antes en la lista de reglas.	
Estimación	1
Prioridad	6
Dependencias	4.9.
Pruebas de aceptación	
<ul style="list-style-type: none"> - El usuario realiza la acción de mostrar una regla sin que haya ninguna y se le mostrará un mensaje indicando que antes seleccione una. - El usuario selecciona una regla, la muestra, vuelve a aparecer en la vista y se muestra como antes en la lista. 	

Cuadro 4.17: Historia de usuario - Mostrar regla

4.11.	Crear transportador de ángulos
Descripción	
El sistema debe proveer de una funcionalidad para poder realizar medidas de ángulos a tamaño real de zonas en un corte. Para ello el usuario solo tendrá que seleccionar tres puntos y le aparecerá la medida del ángulo en grados. Se hará uso de un <i>widget</i> de VTK para ello.	
Estimación	3
Prioridad	3
Dependencias	-
Pruebas de aceptación	
<ul style="list-style-type: none"> - El usuario crea un transportador de ángulos sin seleccionar ninguna ROD y se le informa de que seleccione una. - El usuario selecciona tres puntos y le aparecerá la medida del ángulo entre esos puntos. 	

Cuadro 4.18: Historia de usuario - Crear transportador de ángulos

4.12.	Eliminar transportador de ángulos
Descripción	
Se deben poder eliminar medidas de ángulos de la pantalla. Para ello el usuario solo tiene que seleccionarlo y pulsar en eliminar.	
Estimación	3
Prioridad	4
Dependencias	4.11.
Pruebas de aceptación	
<ul style="list-style-type: none"> - El usuario realiza la acción de eliminar un transportador de ángulos sin que haya ninguno y se le mostrará un mensaje indicando que antes seleccione uno. - El usuario selecciona una regla, la elimina y desaparece de la lista y de la vista. 	

Cuadro 4.19: Historia de usuario - Eliminar transportador de ángulos

4.13.	Editar transportador de ángulos
Descripción	
El usuario debe ser capaz de editar una medida de ángulos por si se ha equivocado. Para ello selecciona y arrastra uno de los puntos de éste.	
Estimación	1
Prioridad	5
Dependencias	4.11.
Pruebas de aceptación	
- El usuario selecciona y arrastra un punto y se actualiza la posición de éste y el valor de la medida del ángulo entre ambos.	

Cuadro 4.20: Historia de usuario - Editar transportador de ángulos

4.14.	Ocultar transportador de ángulos
Descripción	
El usuario debe ser capaz de ocultar un transportador de ángulos de la vista sin eliminarlo. Para ello lo selecciona en la lista y pulsa en ocultar. Se mostrará de forma distinta en la lista de transportadores de ángulos para que se tenga claro que se ha ocultado.	
Estimación	2
Prioridad	6
Dependencias	4.11.
Pruebas de aceptación	
- El usuario realiza la acción de ocultar un transportador de ángulos sin que haya ninguno y se le mostrará un mensaje indicando que antes seleccione uno. - El usuario selecciona un transportador de ángulos, lo oculta y desaparece de la vista, pero no de la lista.	

Cuadro 4.21: Historia de usuario - Ocultar transportador de ángulos

4.15.	Mostrar transportador de ángulos
-------	----------------------------------

Descripción

El usuario debe ser capaz de mostrar un transportador de ángulos que previamente ha ocultado. Para ello lo selecciona en la lista y pulsa en mostrar. Se volverá a mostrar como antes en la lista de transportadores de ángulos.

Estimación	1
Prioridad	7
Dependencias	4.14.

Pruebas de aceptación

- El usuario realiza la acción de mostrar un transportador de ángulos sin que haya ninguno y se le mostrará un mensaje indicando que antes seleccione uno.

- El usuario selecciona un transportador de ángulos, lo muestra, vuelve a aparecer en la vista y se muestra como antes en la lista.

Cuadro 4.22: Historia de usuario - Mostrar transportador de ángulos

4.16.	Crear nota
-------	------------

Descripción

El sistema debe proveer de una funcionalidad para poder realizar anotaciones en un corte. Para ello el usuario tendrá que introducir un texto y colocar la flecha apuntando al sitio de donde viene la anotación. Se hará uso de un *widget* de VTK para ello.

Estimación	4
Prioridad	2
Dependencias	-

Pruebas de aceptación

- El usuario crea una nota sin seleccionar ninguna ROD y se le informa de que seleccione una.

- El usuario crea una nota sin texto y se le informa de que lo introduzca.

- El usuario introduce un texto, crea la nota y se muestra en la vista.

Cuadro 4.23: Historia de usuario - Crear nota

4.17.	Eliminar nota
Descripción	
Se deben poder eliminar notas de la pantalla. Para ello el usuario solo tiene que seleccionarla y pulsar en eliminar.	
Estimación	2
Prioridad	4
Dependencias	4.16.
Pruebas de aceptación	
<ul style="list-style-type: none"> - El usuario realiza la acción de eliminar una nota sin que haya ninguna y se le mostrará un mensaje indicando que antes seleccione una. - El usuario selecciona una nota, la elimina y desaparece de la lista y de la vista. 	

Cuadro 4.24: Historia de usuario - Eliminar nota

4.18.	Editar nota
Descripción	
El usuario debe ser capaz de editar la posición de una nota. Para ello selecciona y arrastra la flecha a donde apunta.	
Estimación	3
Prioridad	5
Dependencias	4.16.
Pruebas de aceptación	
<ul style="list-style-type: none"> - El usuario selecciona y arrastra el punto donde apunta una nota y se actualiza la posición de ésta. 	

Cuadro 4.25: Historia de usuario - Editar nota

4.19.	Ocultar nota
Descripción	
El usuario debe ser capaz de ocultar una nota de la vista sin eliminarla. Para ello la selecciona en la lista y pulsa en ocultar. Se mostrará de forma distinta en la lista de notas para que se tenga claro que se ha ocultado.	
Estimación	2
Prioridad	6
Dependencias	4.16.
Pruebas de aceptación	
<ul style="list-style-type: none"> - El usuario realiza la acción de ocultar una nota sin que haya ninguna y se le mostrará un mensaje indicando que antes seleccione una. - El usuario selecciona una nota, la oculta y desaparece de la vista, pero no de la lista. 	

Cuadro 4.26: Historia de usuario - Ocultar nota

4.20.	Mostrar nota
Descripción	
El usuario debe ser capaz de mostrar una nota que previamente ha ocultado. Para ello la selecciona en la lista y pulsa en mostrar. Se volverá a mostrar como antes en la lista de notas.	
Estimación	1
Prioridad	6
Dependencias	4.19.
Pruebas de aceptación	
<ul style="list-style-type: none"> - El usuario realiza la acción de ocultar una nota sin que haya ninguna y se le mostrará un mensaje indicando que antes seleccione una. - El usuario selecciona una nota, la oculta y desaparece de la vista, pero no de la lista. 	

Cuadro 4.27: Historia de usuario - Mostrar nota

Mejoras en código

5.1. Internacionalización	
Descripción	
El usuario tiene que tener la posibilidad de utilizar la aplicación en dos idiomas distintos, español e inglés. Para ello se utilizará el mecanismo de internacionalización que provee Qt y se generarán dos instaladores: uno en español y otro en inglés.	
Estimación	6
Prioridad	8
Dependencias	-
Pruebas de aceptación	
- El usuario ejecutará el programa en cualquiera de los idiomas y se mostrarán los textos con el idioma correspondiente.	

Cuadro 4.28: Historia de usuario - Internacionalización

Capítulo 5

Diseño

En este capítulo se mostrarán los diagramas arquitectónico, de paquetes y de clases de la aplicación.

Estos diagramas UML han sido generados con la aplicación StarUML [9] y están disponibles en la rama de diseño del repositorio del proyecto alojado en GitHub: <https://github.com/fblupi/3DCurator/tree/design> por si es necesario verlos a mayor tamaño.

5.1. Arquitectura del software

El software está íntegramente escrito en C++ a excepción de las interfaces gráficas que usan un formato específico basado en XML.

Se hace uso a nivel hardware tanto de la CPU como de la GPU usando OpenGL como librería gráfica de bajo nivel pese a no programar directamente con esta librería. En su lugar se utiliza la librería gráfica de alto nivel VTK que abstrae de la complejidad de OpenGL.

Para gestionar la interfaz de usuario se utiliza Qt que permite una buena integración con VTK.

Se usan también, como librerías adicionales, ITK para aplicar algoritmos de filtros a los volúmenes, OpenCV para los algoritmos de visión por computador y Boost para la gestión de ficheros XML.

Para generar el proyecto pre-compilando todas estas librerías citadas anteriormente, así como para hacerlo multiplataforma y poder compilarlo en cualquier sistema operativo se usa CMake.

Librerías de alto nivel	Boost	VTK	ITK	OpenCV	Qt
Librerías de bajo nivel	OpenGL				
Lenguaje de programación	C++				
Nivel Hardware	CPU		GPU		

Cuadro 5.1: Arquitectura del software

5.2. Diagrama de paquetes

En el siguiente diagrama (Figura 5.1) se muestran las dependencias entre los distintos paquetes cuyas clases se detallarán en el siguiente apartado.

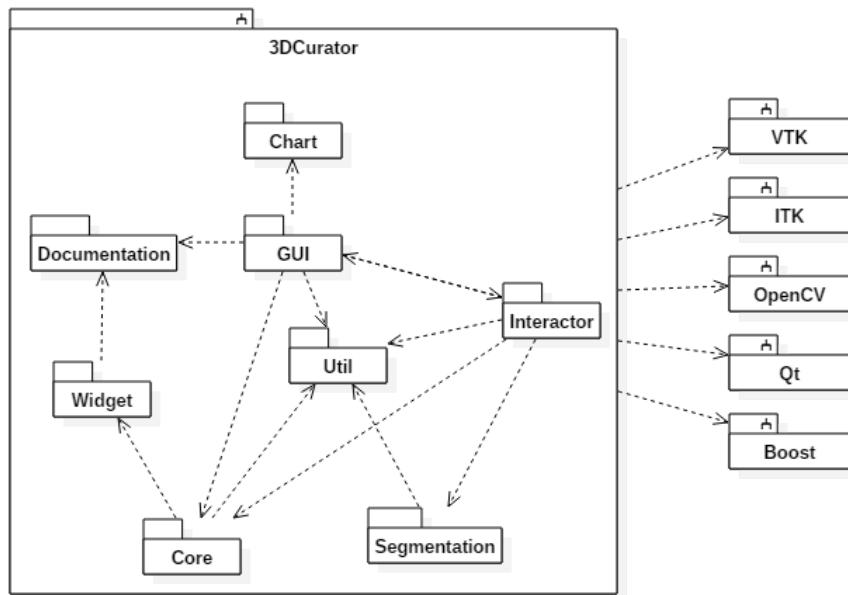


Figura 5.1: Diagrama de paquetes de 3DCurator

5.3. Diagramas de clases

Se presentan las distintas clases de los distintos módulos que contiene 3DCurator.

5.3.1. *Chart*

Este módulo contiene las clases auxiliares para controlar las gráficas utilizadas para visualizar y editar la función de transferencia:

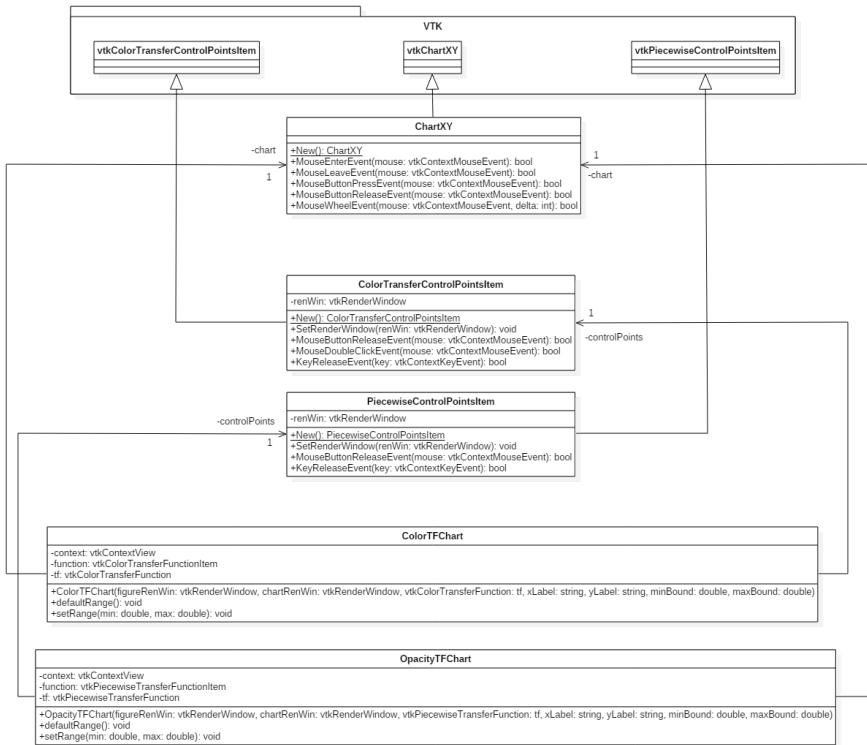
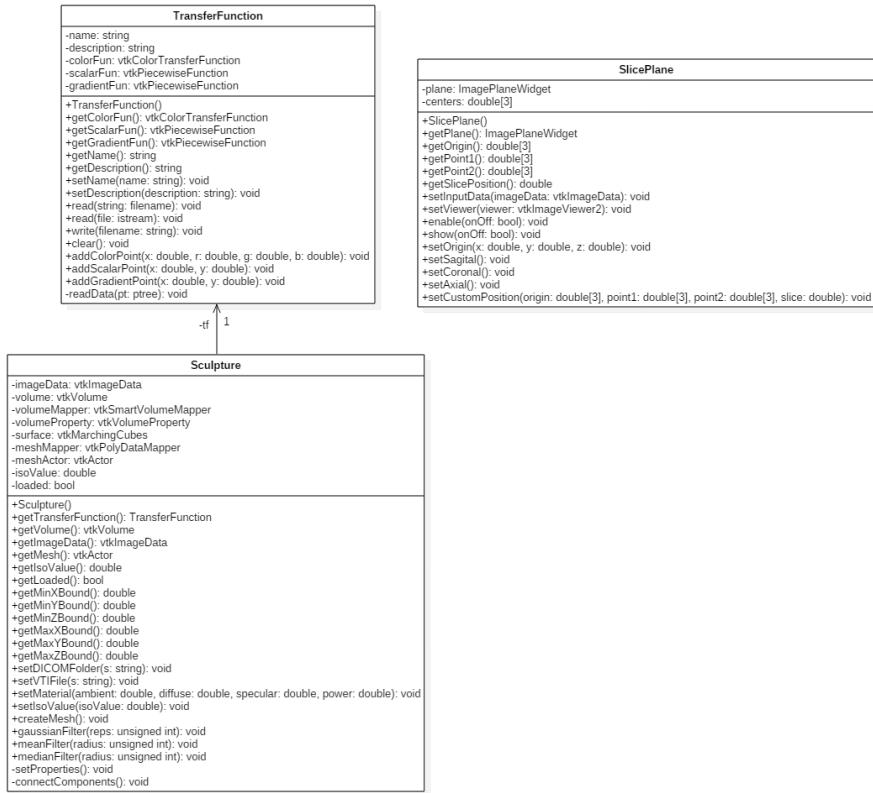


Figura 5.2: Diagrama de clases del paquete *Chart*

5.3.2. *Core*

Este es el módulo principal de 3DCurator. Contiene las clases que gestionan los datos del volumen (*Sculpture*), la función de transferencia (*Transfer Function*) y plano de corte (*SlicePlane*):

Figura 5.3: Diagrama de clases del paquete *Core*

5.3.3. Documentation

En este módulo se encuentra la clase que almacena los distintos elementos utilizados para la documentación:

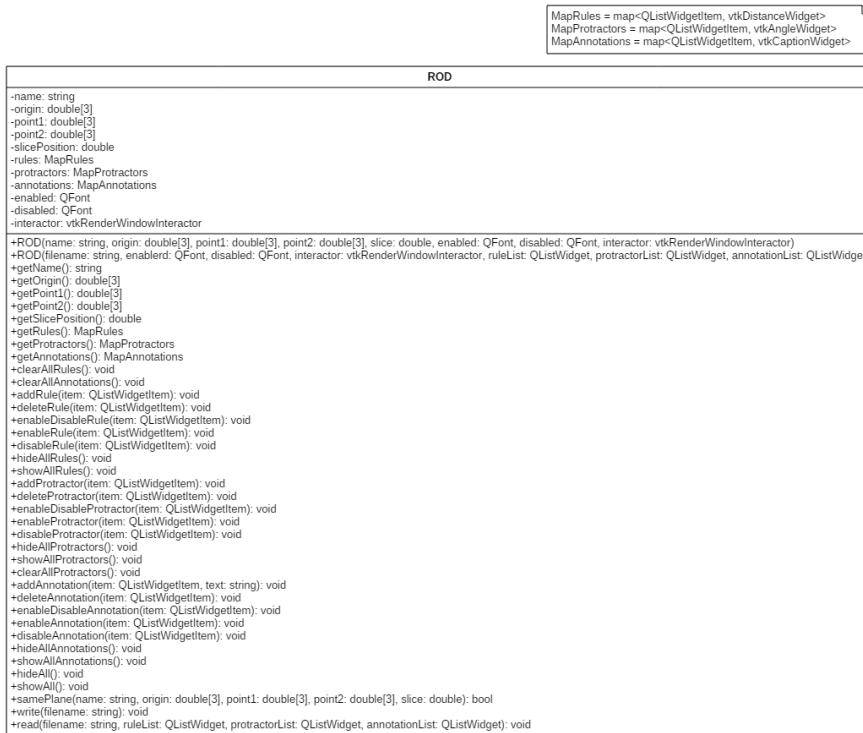


Figura 5.4: Diagrama de clases del paquete *Documentation*

5.3.4. GUI

Este módulo contiene las distintas clases que gestionan las ventanas de la GUI:

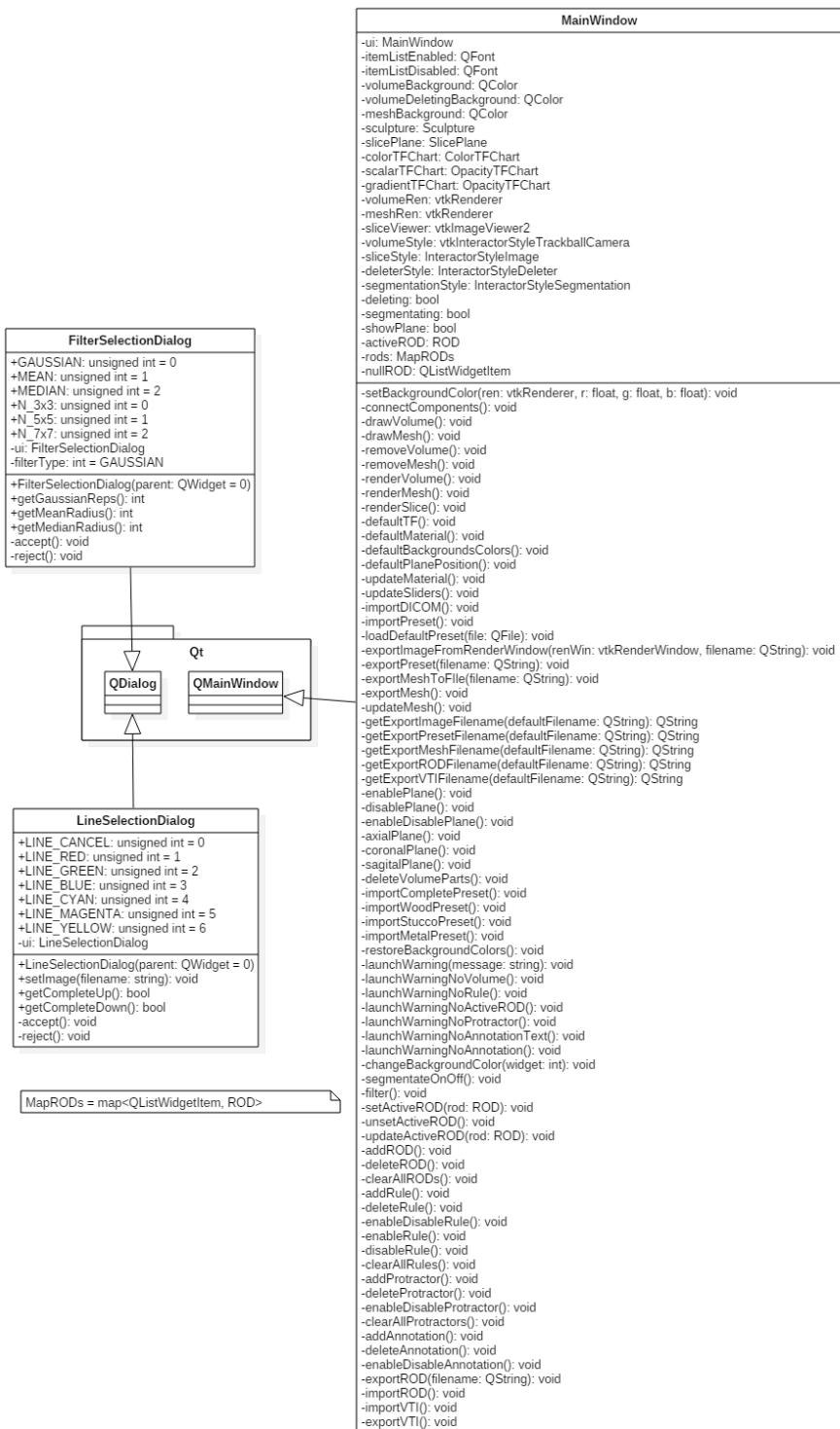


Figura 5.5: Diagrama de clases del paquete *GUI*

5.3.5. *Interactor*

Este módulo contiene los distintos interactuadores con las ventanas de visualización:

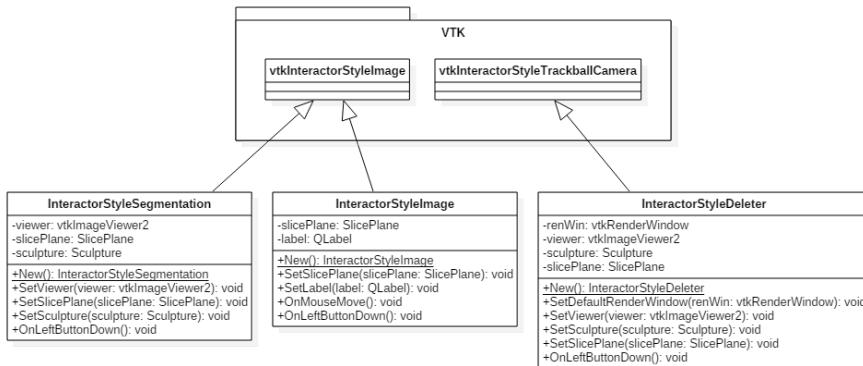


Figura 5.6: Diagrama de clases del paquete *Interactor*

5.3.6. *Segmentation*

En este módulo se encuentran las clases utilizadas para realizar segmentaciones en el volumen:

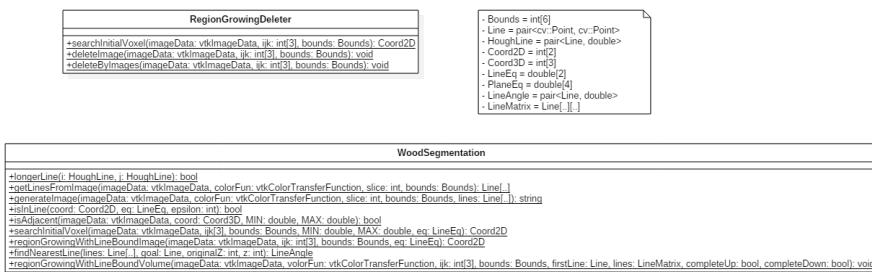
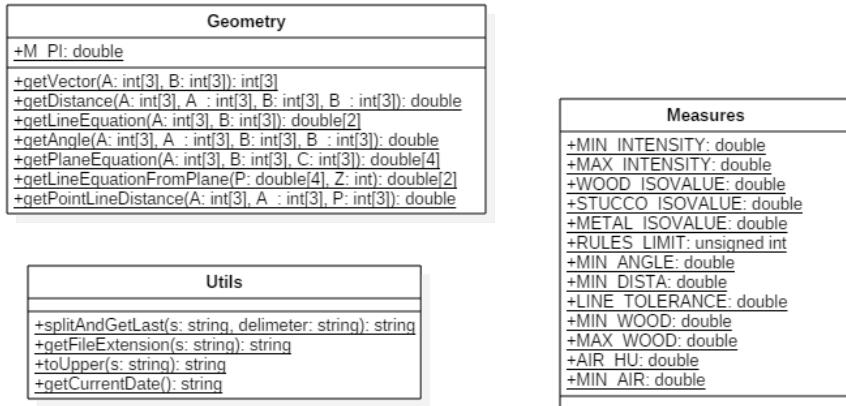


Figura 5.7: Diagrama de clases del paquete *Segmentation*

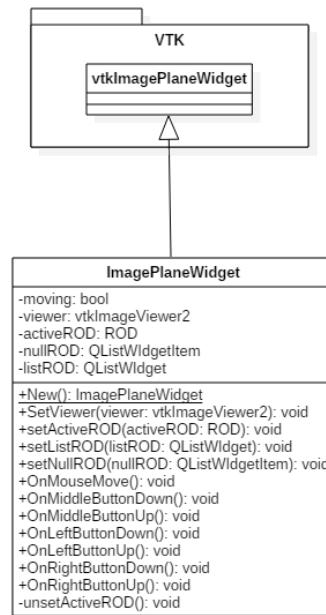
5.3.7. *Util*

Este módulo contiene funciones utilizadas frecuentemente por distintos módulos del programa:

Figura 5.8: Diagrama de clases del paquete *Util*

5.3.8. *Widget*

Este módulo contiene el *widget* personalizado de VTK que se utiliza:

Figura 5.9: Diagrama de clases del paquete *Widget*

Capítulo 6

Desarrollo del trabajo

En este capítulo se realizará un estudio detallado del estado del arte para posteriormente detallar las distintas fases de desarrollo y los problemas que han surgido con sus respectivas soluciones.

6.1. Desarrollo profundo del estado del arte

Este trabajo es la continuación de mi anterior TFG donde traté el tema de la renderización de conjuntos de datos volumétricos, este trabajo está dirigido al tratamiento de este conjunto de datos.

Un conjunto de datos volumétrico o campo escalar 3D está representado por una función $R^3 \rightarrow R$. En otras palabras, se trata de un conjunto de datos representado por una matriz tridimensional, donde cada elemento de esta matriz se puede denominar voxel y es importante tener claro que el valor de este no es un color sino un valor como tal que posteriormente se renderizará con un color y una transparencia según una función de transferencia. Es ahí donde erradica la separación conceptual entre modelado y visualización de un conjunto de datos volumétrico.

El flujo a la hora de representar un conjunto de datos volumétricos sería el siguiente:

- Obtención de imágenes
- Filtrado
- Segmentación
- Visualización

Durante el TFG traté el último de los pasos y en este TFM comentaré las distintas opciones del primero y trabajaré en técnicas del segundo y el tercero aplicadas a esculturas de madera policromadas.

6.1.1. Obtención de imágenes

Existen diversas técnicas de obtención de imágenes volumétricas. En esta sección se describirán las dos técnicas más usadas actualmente y se realizará una comparación entre ellas. No se han incluido técnicas como el PET, SPECT o ecografía pues necesitan de contrastes que no se podrían aplicar en una escultura si se quiere preservar su estado.

Tomografía Computarizada

La Tomografía Computarizada (TC o CT en inglés) fue la primera de las técnicas que surgió para la obtención de datos volumétricos pero ha ido evolucionando hasta el día de hoy como se detalló en la introducción.

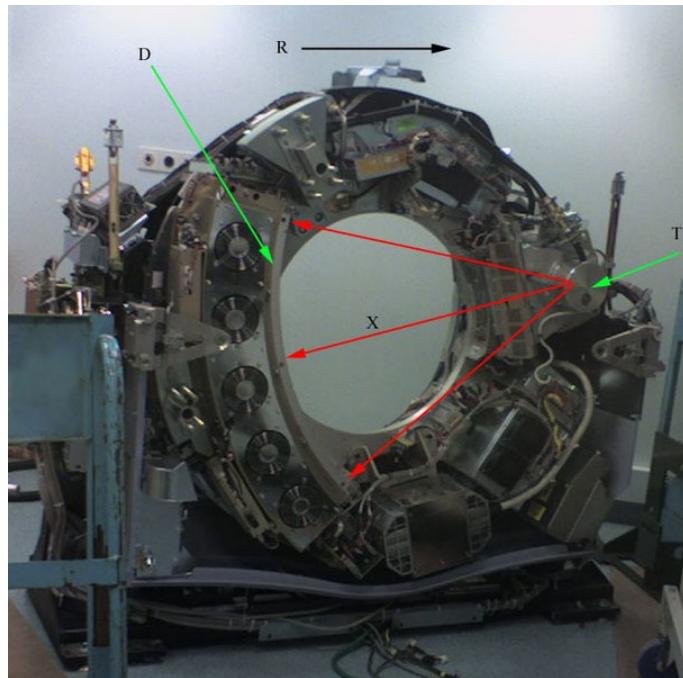


Figura 6.1: Escáner TC sin la carcasa, por lo que se puede ver sus componentes internos. T: Tubo rayos X, D: Detectores rayos X, X: Haces de rayos X y R: *Gantry*. Imagen extraída de: <https://en.wikipedia.org/wiki/File:Ct-internals.jpg>

Los parámetros utilizados en una TC son los siguientes:

- **Resolución espacial** (número de cortes, píxeles por corte y distancia entre véxoles): Si la resolución es más alta los datos serán más ruidosos si la dosis de radiación se mantiene.
- **Dosis de radiación**: Si la dosis de radiación es mayor se conseguirá mejor ratio señal-ruido y las imágenes podrán ser de mayor resolución sin que el ruido sea un problema.
- **Gantry tilt** (sistema de rotación emisor/receptor): Se puede adaptar el *gantry tilt* para una parte específica a examinar.

Los valores de intensidad de las imágenes extraídas se encuentran en unidades Hounsfield (HU). Que es una unidad de medida normalizada que hace que un tejido tenga ese valor sean cuales sean los parámetros del escáner.

Imagen por Resonancia Magnética

La Imagen por Resonancia Magnética (IRM o MRI en inglés) es una técnica en la que, a diferencia de la TC donde se usa radiación ionizante, se usan campos magnéticos para diferenciar los distintos materiales del objeto escaneado, específicamente la ocurrencia del núcleo de hidrógeno que son capaces de absorber y emitir radio frecuencia cuando se colocan en un campo magnético externo [23].

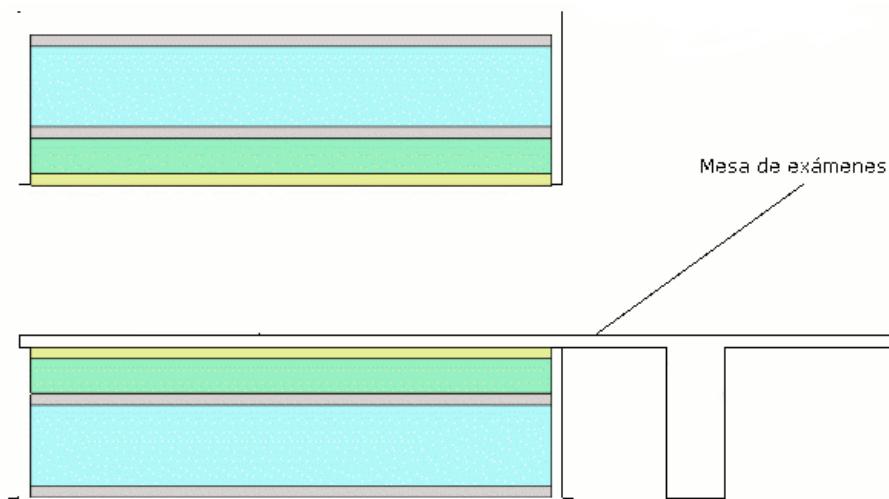


Figura 6.2: Esquema de un escáner IRM (Sección Longitudinal). Imagen extraída de: https://en.wikipedia.org/wiki/File:Mri_scanner_schematic_labelled.svg

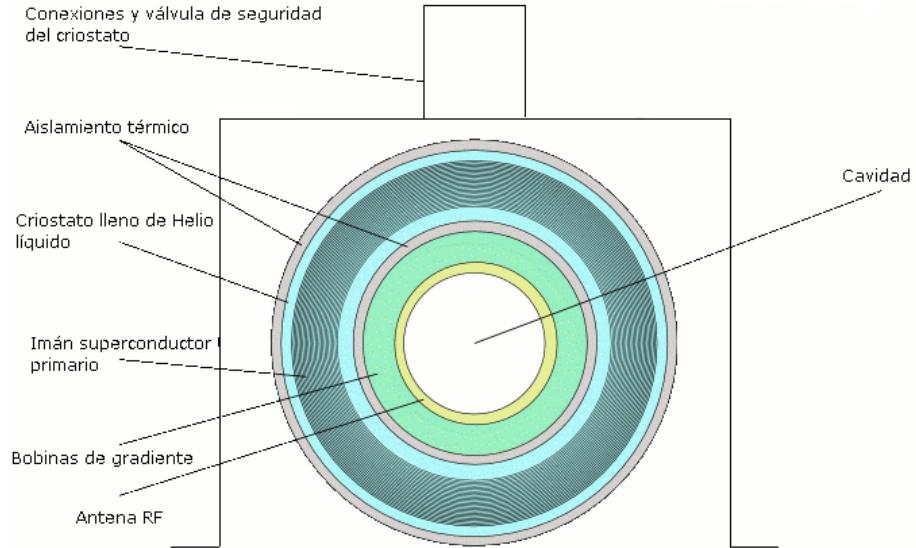


Figura 6.3: Esquema de un escáner IRM (Sección Axial). Imagen extraída de: https://en.wikipedia.org/wiki/File:Mri_scanner_schematic_labelled.svg

El campo magnético alinea los momentos magnéticos de los núcleos atómicos de hidrógeno en dirección paralela y anti-paralela.

A continuación se emite radiación electromagnética a un pulso de radiofrecuencia determinado. Algunos núcleos que se encuentran en dirección paralela pasarán a anti-paralela y al volver a su dirección original perderán energía en forma de fotones que podrán ser detectados.

Estos dos tiempos, T1 (*phase*) y T2 (*dephase*) son medidos. Para obtener T1 hay que ver el valor en la gráfica de relajación longitudinal al 63 % y para obtener T2 hay que hacer lo mismo en la gráfica de relajación transversal al 37 % [16].

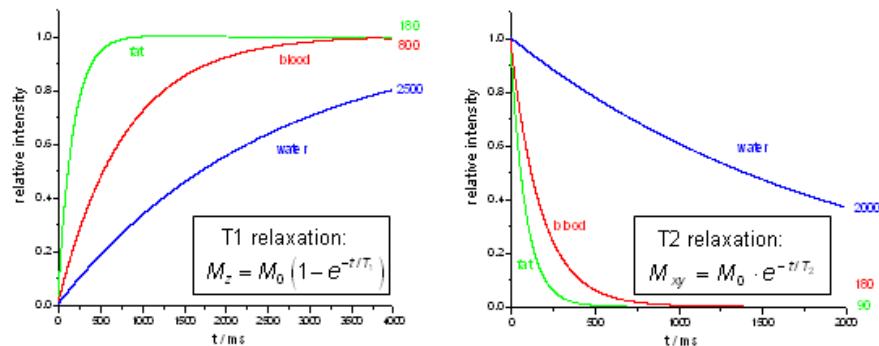


Figura 6.4: A la izquierda gráfica de la relajación longitudinal (crecimiento logarítmico) y a la derecha gráfica de la relajación transversal (crecimiento exponencial) [16]

Comparación entre TC e IRM

Las principales diferencias entre ambas técnicas son las siguientes:

- La IRM obtiene imágenes de menor resolución que la TC.
- La IRM proporciona un mayor contraste entre tejidos poco densos.
- Los datos obtenidos con una TC son más entendibles por médicos mientras que los obtenidos con una IRM por radiólogos.
- El tiempo y coste de escaneo de una IRM es mayor que el de una TC.
- Los datos obtenidos con una TC se encuentran en unidades normalizadas mientras que los obtenidos con una IRM variarán dependiendo de los parámetros del escáner.

Al necesitar unos *presets* con los que se pueda visualizar la escultura sin necesidad de que el usuario edite la función de transferencia, el último punto nos haría decantar por la TC. Además gracias a esta podremos obtener imágenes de mayor resolución. El único punto en contra en esta elección es que el contraste entre materiales de baja densidad (como la madera) no será tan distingible como con la IRM.

6.1.2. Filtrado

Los datos en crudo obtenidos con las técnicas anteriormente descritas muchas veces no son lo suficientemente buenas y tienen lo que se denomina

ruido. Hay muchos tipos de ruido y existen distintos filtros que aplicar a las imágenes para reducirlo.

Antes de describir los filtros más usados, se va a profundizar en ciertos aspectos teóricos necesarios para entender mejor cómo funcionan. Estos conceptos se van a describir para un espacio 2D, aunque pasar a un espacio 3D como el de los volúmenes es trivial pues tan solo haría falta utilizar una variable más para la profundidad.

Lo primero que hay que comprender es el concepto de vecindario. Que no es más que los píxeles que lo rodean a una distancia concreta. Por ejemplo un vecindario de tamaño 3x3 sobre el punto p es un conjunto de píxeles con tamaño 3x3 con centro en el píxel p (Figura 6.5).

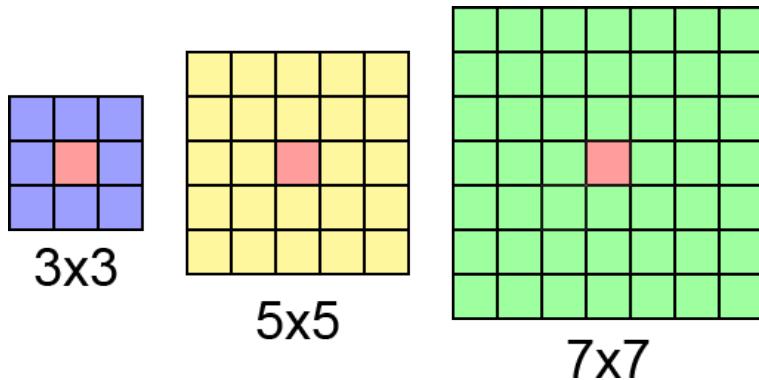


Figura 6.5: Vecindarios 3x3, 5x5 y 7x7 de un píxel (rojo).

Un dominio espacial se denota con la expresión:

$$g(x, y) = T[f(x, y)]$$

Donde:

- $f(x, y)$ es la imagen de entrada
- $g(x, y)$ es la imagen de salida
- T es un operador en f definido sobre el vecindario (x, y)

Los filtros espaciales consisten en aplicar el operador T a los píxeles del vecindario. Por ejemplo en un vecindario 3x3 y un operador T definido como la intensidad media del vecindario, el valor $g(x_i, y_j)$ será la suma del valor $f(x_i, y_j)$ de su vecindario dividido entre 9.

Filtro media

El filtro media es el definido anteriormente. Es decir, usa un *kernel* en el que todos los vecinos tienen el mismo peso.

Es un filtro utilizado para suavizar imágenes con mucho ruido.

Filtro mediana

El filtro media hace uso de la mediana para calcular el valor de salida del píxel. Por ejemplo, para la matriz:

$$\begin{bmatrix} 1 & 4 & 0 \\ 2 & 2 & 4 \\ 1 & 0 & 1 \end{bmatrix}$$

El valor para el punto p correspondiente a $M(1,1)$ con valor original 2, sería 1. Porque es la mediana de su vecindario $(0, 0, 1, 1, 1, 2, 2, 4, 4)$.

Este filtro es muy usado por ser el más efectivo para reducir el ruido de tipo *salt-and-pepper*



Figura 6.6: Ejemplo de ruido tipo *salt-and-pepper*, utilizando un filtro mediana. Imagen extraída de: <https://en.wikipedia.org/wiki/File:Medianfilterp.png>

Filtro gaussiano

El filtro *gaussiano* o binomial (Figura 6.7) hace uso de una versión discretizada de la función *gaussiana* y, por tanto, se basa en la convolución de una matriz que para un vecindario 5x5 sería:

$$\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Para normalizar los elementos del *kernel* haría falta que sumasen 1, por lo que se divide entre la suma de todos sus elementos (256 en el caso de 5x5).

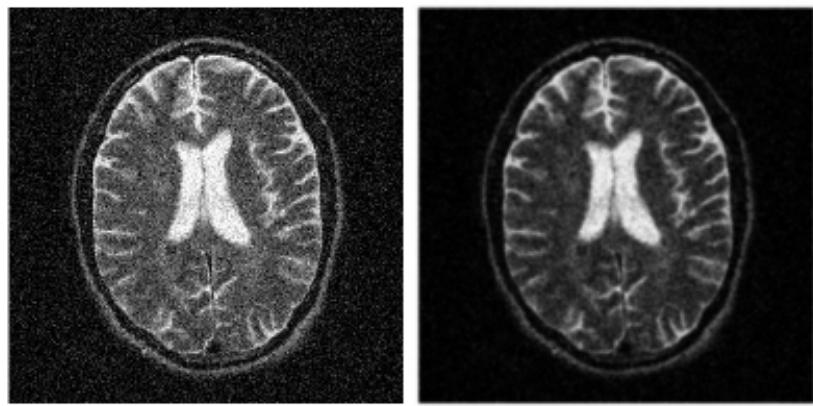


Figura 6.7: Ejemplo de suavizado usado un filtro *gaussiano*

Este filtro, al igual que el que hace uso de la media es, es un filtro paso baja utilizado para suavizar imágenes.

6.1.3. Segmentación

La segmentación es la separación de partes de un volumen para su estudio por separado. Antes de profundizar en algunas de las técnicas utilizadas en la actualidad, hay que tener claro una serie de conceptos [21].

- **Adyacencia:** Dos píxel son adyacentes si son vecinos y satisfacen un criterio de similitud, por ejemplo, que su valor de intensidad esté entre -700 y -300.
- **Camino:** Un camino entre dos píxeles es una secuencia de píxeles adyacentes que va desde un píxel hasta otro.
- **Conectividad:** Existe conectividad entre dos píxeles si se puede trazar al menos un camino entre ambos.

- **Componente conectado en un subconjunto de una imagen:** En un subconjunto S de una imagen, para todos los píxeles p en S , el conjunto de píxeles en S conectados a p se denominan componentes conectados en S .
- **Conjunto conectado:** Si el subconjunto de una imagen S solo tiene un componente conectado, entonces se convierte en un conjunto conectado.
- **Región:** Un subconjunto S de una imagen es una región si es un conjunto conectado.
- **Borde:** El borde de una región S es el conjunto de píxeles que tienen uno o más vecinos no pertenecientes a S .

La segmentación trata de buscar los vértices que pertenecen a una estructura. Existen distintas aproximaciones como se detallaron en la introducción. A continuación se detallarán algunos de los métodos que se citaron.

Segmentación basada en umbrales

Esta segmentación es muy básica y trata de diferenciar las regiones utilizando dos umbrales de valor de intensidad y realizando un filtro en el que se descartasen todos aquellos vértices que no se encuentran entre estos valores [25].

El uso de los histogramas de valores de la imagen pueden resultar útiles a la hora de seleccionar los umbrales.

Este método puede resultar útil para separar materiales con diferencias notables entre valores de densidad. En nuestro caso podría servir para separar madera de estuco, por ejemplo.

Segmentación basada en crecimiento usando umbrales

La segmentación basada en crecimiento usando umbrales es una extensión del método anterior, con la diferencia de que la segmentación basada en umbrales obtiene todas las regiones que se encuentran en la imagen y en la basada en crecimiento solo obtiene una región, obteniendo todos los puntos conectados a uno inicial que tienen el umbral como criterio de similitud [19] (Figura 6.8).

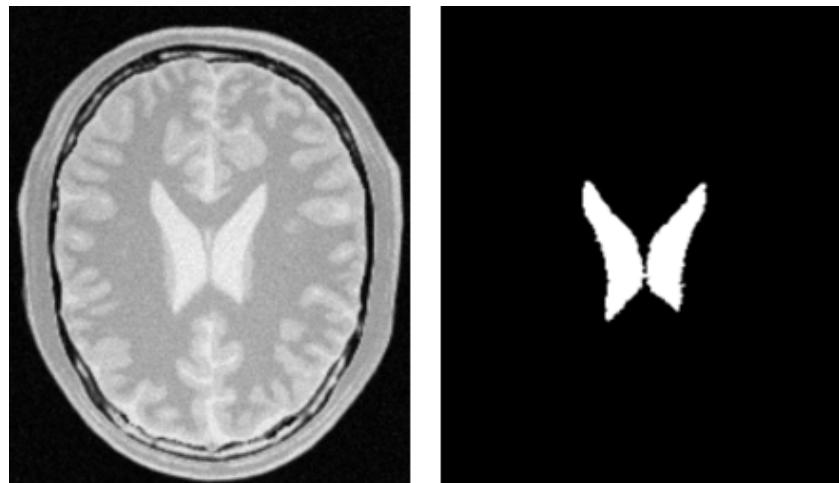


Figura 6.8: Segmentación de los plexos coirodeos de una imagen de un cerebro usando una segmentación basada en crecimiento usando un umbral entre 210 y 255

Una variante de este método sería usar umbrales dinámicos. Es decir, en lugar de utilizar un valor máximo y mínimo global para toda la imagen (por ejemplo, valores de densidad entre -700 y -300), usar un rango dependiendo del punto donde se encuentra (por ejemplo, si el rango es de 100 y estamos en un punto con valor -532, el umbral para sus vecinos estaría entre -632 y -432). Este es el método de segmentación que se implementó en el TFG para segmentar las piezas separadas de la escultura como los elementos de la camilla.

En este caso es muy importante elegir el punto inicial, pues si se escoge un punto que se encuentre cercano al borde, puede hacer que la segmentación se extienda a zonas no deseadas.

Watershed

La transformación divisoria, más conocida como *watershed* se basa en ver las imágenes como un relieve topográfico con crestas y cuencas. Las elevaciones del terreno estarían definidas por los valores de densidad de la imagen o por el gradiente [11] (Figura 6.9).

El resultado es la descomposición de la imagen en distintas cuencas hidrográficas para cada mínimo local. Pero el elevado número de mínimos locales da lugar a sobresegmentación por lo que hay que definir un método de mezclado. El método más utilizado es el de la inundación. Se marca una posición de inundación que podría mezclar varias cuencas.

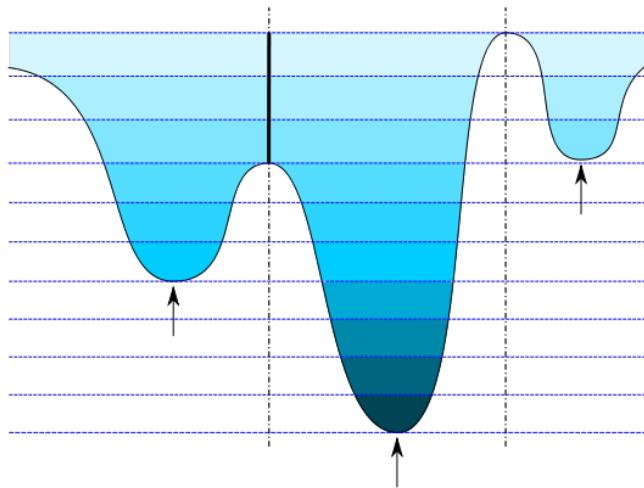


Figura 6.9: El terreno se indica con una línea continua negra, las distintas regiones están marcadas con líneas discontinuas negras, las flechas los mínimos locales, las líneas discontinuas azules los niveles de agua y las líneas discontinuas azules los distintos niveles de agua para realizar la inundación. Imagen extraída de https://en.wikipedia.org/wiki/File:Watershed_transform--flood_interpretation.svg

Livewire

El método de segmentación *livewire* (también conocido como tijeras inteligentes) es, a diferencia de los anteriores, un método basado en bordes.

Livewire es un método manual de segmentación donde hace falta la intervención del usuario en prácticamente todo momento pues tiene que ir seleccionando puntos del borde de la región a segmentar en cada corte.

Se hace uso del algoritmo de *Dijkstra* para calcular el camino de coste mínimo entre el punto seleccionado por el usuario y uno anterior. Hace uso de la componente gradiente para dar coste a los nodos del grafo [24].

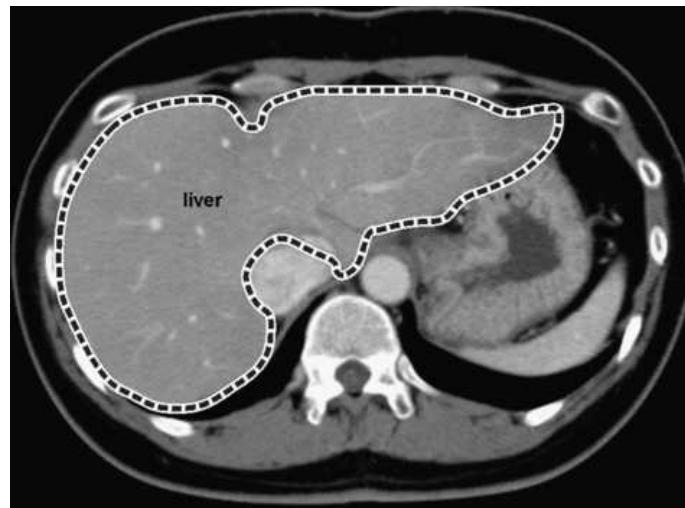


Figura 6.10: Ejemplo de hígado en un corte segmentado usando *livewire* [28]

Este método es quizás el más preciso, pero para obtener un resultado hace falta un usuario delante del ordenador recortando todos y cada uno de los cortes del conjunto de datos volumétrico.

6.2. Plataforma de desarrollo

El *software* se ha desarrollado usando CMake [2] por lo que se pueden crear *makefiles* para distintas plataformas.

En mi caso, lo he desarrollado con Windows 10 Pro y el compilador de Visual Studio de 2017.

Se ha compilado con las versiones más recientes de los entornos de desarrollo y librerías:

- Visual Studio Community 2017 15.7.3 (Junio de 2018)
- CMake 3.11.3 (Mayo de 2018)
- Qt5.11.0 (Mayo de 2018)
- VTK 8.1.1 (Mayo de 2018)
- ITK 4.13.0 (Diciembre de 2017)
- Boost 1.67.0 (Abril de 2018)
- OpenCV 3.4.1 (Febrero de 2018)

Ha sido un compromiso y esfuerzo propio utilizar las últimas versiones de estas librerías porque entre una versión y otra hay problemas de incompatibilidad y funciones que quedan *deprecated*, pero al mismo tiempo, tener un *software* que usa las últimas versiones estables de sus librerías ayuda a no estancarse en una versión antigua y que una futura migración no se convierta en un esfuerzo titánico.

6.3. Instalación y configuración

6.3.1. Entorno de desarrollo

Visual Studio Community 2017

- Descargar Visual Studio Community 2017 e instalar

Qt5.11.0

- Descargar e instalar Qt5.11.00 e instalar para el compilador *msvc2017 64-bit*
- Crear una nueva variable de entorno QTDIR con el valor C:\Qt\Qt5.11.0
- Añadir al *path*: C:\Qt\Qt5.11.0\5.11.0\msvc2017_64\bin

CMake 3.11.3

- Descargar CMake 3.11.3 e instalar

6.3.2. Generar librerías

VTK 8.1.1

- Descargar VTK 8.1.1 y extraer los ficheros
- Abrir CMake y completar los siguientes campos:
 - *src*: C:\VTK\8.1.1\src
 - *build*: C:\VTK\8.1.1\build\msvc2017_64
- Usar *Visual Studio 15 2017 Win 64* como generador
- Marcar las opciones avanzadas

- Una vez generado marcar las siguientes opciones:
 - Module_vtkDICOM
 - Module_vtkGUISupportQt
 - Module_vtkGUISupportQtOpenGL
 - Module_vtkRenderingQt
 - Module_vtkViewsQt
 - VTK_Group_Imaging
 - VTK_Group_Qt
- Configurar hasta que no aparezca ninguna opción en rojo
- Pulsar en generar y creará directorios y ficheros en:
C:\VTK\8.1.1\build\msvc2017_64

ITK 4.13.0

- item Descargar ITK 4.1.30 y extraer los ficheros
- Abrir CMake y completar los siguientes campos:
 - *src*: C:\ITK\4.13.0\src
 - *build*: C:\ITK\4.13.0\build\msvc2017_64
- Usar Visual Studio 15 2017 Win 64 como generador
- Marcar las opciones avanzadas
- Una vez generado marcar las siguientes opciones:
 - Module_ITKVtkGlue
- Configurar y aparecerá un error. Hay que darle a la variable VTK_DIR:
C:\VTK\ 8.1.1\build\msvc2017_64
- Configurar hasta que no aparezca ninguna opción en rojo
- Pulsar en generar y creará directorios y ficheros en:
C:\ITK\4.13.0\build\msvc2017_64

6.3.3. Compilar librerías

VTK 8.1.1

- Abrir VTK.sln
- Compilar ALL_BUILD en modo *Release* y esperar un tiempo hasta que termine
- Crear una nueva variable de entorno VTK_DIR con el valor:
C:\VTK\8.1.1\build\build\msvc2017_64
- Añadir al *path*: %VTK_DIR%\bin\Release

ITK 4.13.0

- Abrir ITK.sln
- Compilar ALL_BUILD en modo *Release* y esperar un tiempo hasta que termine
- Crear una nueva variable de entorno ITK_DIR con el valor:
C:\ITK\4.13.0\build\build\msvc2017_64
- Añadir al *path*: %ITK_DIR%\bin\Release

Boost 1.67.0

- Descargar Boost 1.67.0 y extraer los ficheros
- Abrir una *Development Command Prompt for VS2017* en la carpeta donde se han extraido
- Escribir bootstrap.bat y generar Boost.build
- Compilar con:
b2 toolset=msvc --build-type=complete --abbreviate-paths
address-model=64 install -j4
- Crear una nueva variable de entorno BOOST_ROOT con el valor:
C:\Boost

OpenCV 3.4.1

- Descargar OpenCV 3.4.1
- Ejecutar y extraer

- Mover la carpeta extraída a
C:\OpenCV\3.4.1
- Crear una nueva variable de entorno OPENCV_ROOT con el valor:
C:\OpenCV\3.4.1\opencv\build\x64\vc15
- Crear una nueva variable de entorno OPENCV_DIR con el valor:
%OPENCV_ROOT%\lib
- Añadir al path: %OPENCV_ROOT%\bin

6.3.4. Compilar proyecto

- Abrir CMake y completar los campos:
 - *src: %PROJECT_PATH%\src*
 - *build: %PROJECT_PATH%\build*
- Usar Visual Studio 15 2017 Win64 como generador
- Configurar hasta que no haya ninguna opción en rojo
- Presionar en generar y creará directorios y ficheros en:
%PROJECT_PATH%\build
- Abrir 3DCurator.sln
- Abrir *Project Properties* e ir a *Configuration Properties* → *Linker* → *System* y:
 - En *Subsystem* seleccionar la opción:
Windows (/SUBSYSTEM:WINDOWS)
 - En *Enable Large Addresses* seleccionar la opción:
Yes (/LARGEADDRESSAWARE)
- Establecer la solución 3DCurator como proyecto predeterminado
- Una vez compilado copiar a la carpeta build\Release los archivos DLL libEGL.dll, libGLESv2.dll, Qt5Core.dll, Qt5Gui.dll y Qt5Widgets.dll de la carpeta:
C:\Qt\Qt5.11.0\5.11.0\msvc2017_64\bin

6.4. Fases de desarrollo

Inicialmente se planteó un desarrollo en tres partes bien diferenciadas e independientes, aunque una de ellas podría ayudar a lograr mejores resultados en la siguiente.

Hablo de la de pre-procesamiento de datos (correspondiente a la etapa de filtrado del flujo de representación de datos volumétricos) en la que se marcó el objetivo de reducir el ruido, específicamente el que producían los objetos metálicos.

Gracias a esto sería más fácil detectar las distintas piezas de madera para la sección de subdivisión de piezas de madera (correspondiente a la etapa de segmentación).

Por último, ya con estas dos secciones terminadas, se pasaría a añadir herramientas de documentación que ayudarían al restaurador a realizar sus tareas de documentación en la propia aplicación sin necesidad de utilizar herramientas externas.

6.4.1. Pre-procesamiento de datos

El objetivo aquí era la reducción del ruido que se podría encontrar en las imágenes. Como ya se ha explicado anteriormente, hay filtros básicos que nos permiten reducirlo. El filtro media y *gaussiano* nos ayuda a suavizar y el filtro mediana a acabar con *outliers* y ruido de tipo *salt-and-pepper* en general. Aunque en los datos de prueba en los que trabajamos no hay ruido de tipo *salt-and-pepper* es importante proveer este filtro pues en otras imágenes podría haber y es el más efectivo a la hora de acabar con él.

Para no tener que crear las matrices de convolución manualmente *reinventando la rueda*, se hizo uso de una librería que nació precisamente para realizar todas las tareas previas al renderizado de volúmenes y que cuenta con una gama de filtros ya implementados. Hablo de ITK [5]. Una librería de código abierto de la misma compañía que VTK, *Kitware*.

Esta librería incluye los tres filtros citados con anterioridad:

- **Media:** Usando la clase `itkMeanImageFilter` pasando como parámetro el tamaño del vecindario.
- **Mediana:** Usando la clase `itkMedianImageFilter` pasando como parámetro el tamaño del vecindario.
- **Gaussiano:** Usando la clase `itkBinomialBlurImageFilter` pasando como parámetro el número de repeticiones a realizar.

Para poder hacer el filtrado hace falta transformar la imagen actual que se encuentra en el formato utilizado en VTK para ser renderizada al formato de ITK. Y una vez realizado el filtro hacer el paso opuesto. Para ello hay que utilizar las clases `itkVTKImageToImageFilter` e `itkImageToVTKImageFilter`.

A continuación se presenta un pequeño *script* con los pasos a seguir para realizar el filtrado en una imagen en VTK usando filtros de ITK (Código 6.1):

Listing 6.1: *script* para usar el filtro media de ITK en una imagen en VTK

```

1 // Definir tipo de imagen ITK usando signed short y 3 dimensiones
2 typedef signed short PixelType;
3 const unsigned int Dimension = 3;
4 typedef itk::Image<PixelType, Dimension> ImageType;
5
6 // Crear pipeline para pasar una imagen VTK a ITK
7 typedef itk::VTKImageToImageFilter<ImageType> VTKImageToImageType;
8 VTKImageToImageType::Pointer vtkImageToImage = VTKImageToImageType::
9     New();
10 vtkImageToImage->SetInput(imageData);
11 vtkImageToImage->Update();
12
13 // Filtrar la imagen ITK usando un filtro media
14 typedef itk::MeanImageFilter<ImageType, ImageType> MeanImageFilterType;
15 MeanImageFilterType::Pointer meanFilter = MeanImageFilterType::New();
16 meanFilter->SetInput(vtkImageToImage->GetOutput());
17 meanFilter->SetRadius(radius);
18 meanFilter->Update();
19
20 // Pasara de imagen ITK filtrada a VTK
21 typedef itk::ImageToVTKImageFilter<ImageType> ImageToVTKImageType;
22 ImageToVTKImageType::Pointer imageToVTKImage = ImageToVTKImageType::
23     New();
24 imageToVTKImage->SetInput(meanFilter->GetOutput());
25 imageToVTKImage->Update();
26
27 // Actualizar la instancia de la imagen VTK
28 imageData->DeepCopy(imageToVTKImage->GetOutput());
29 imageData->Modified();

```

Integración con la GUI

La integración con la interfaz es simple e intuitiva. Tan solo hay que pulsar en el botón de filtrado (Figura 6.11) y aparecerá un cuadro de diálogo donde se podrá elegir el tipo de filtro a aplicar y sus parámetros (Figura 6.12). Una vez seleccionado se pulsa en OK y empezará a filtrar. Es un proceso largo por lo que se coloca un cuadro de diálogo informando al usuario que tenga paciencia.

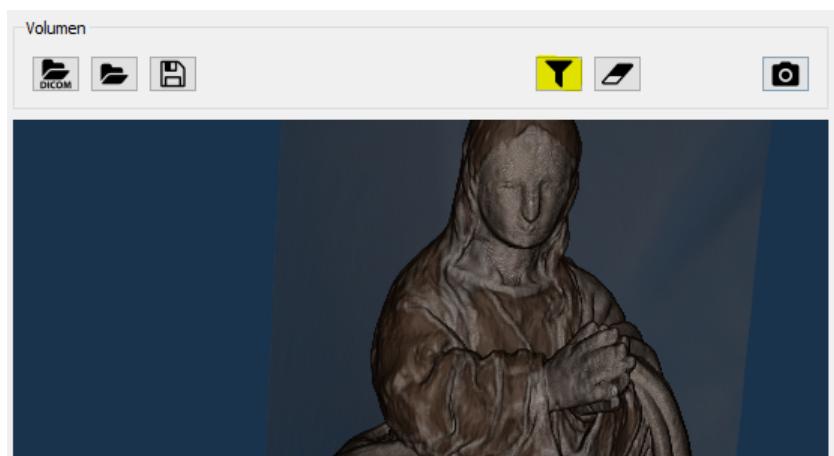


Figura 6.11: Botón que hay que pulsar para que aparezca el diálogo de filtrado

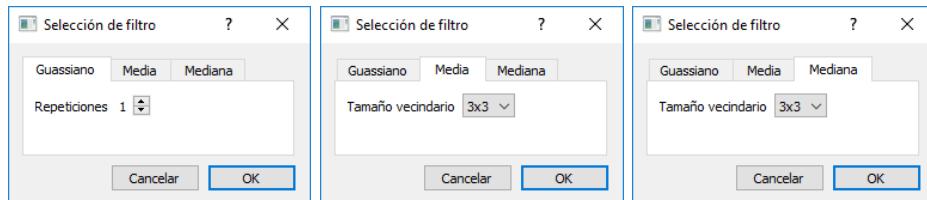


Figura 6.12: Cuadro de diálogo con los filtros posibles. Se muestran tres imágenes, una por cada pestaña abierta para mostrar el parámetro que hay que establecer para aplicar el filtro

Problema con el ruido de elementos metálicos

Ningún *pipeline* de los filtros citados anteriormente ha ayudado a eliminar el ruido producido por los elementos metálicos (Figura 6.13).

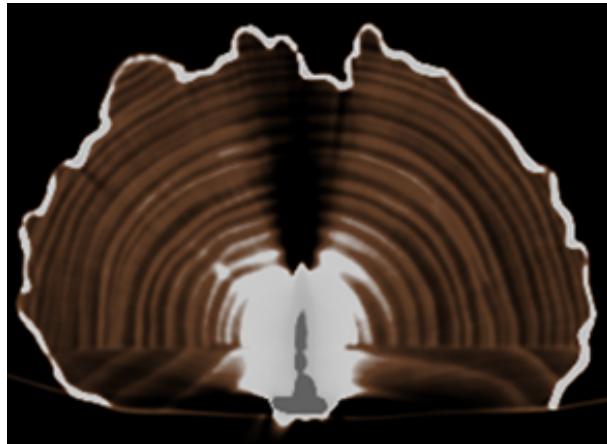


Figura 6.13: Ruido producido por los elementos metálicos

Y es que ningún filtro estudiado sería capaz de reducir este ruido. Pues el tamaño de vecindario que se debería aplicar sería demasiado grande y suavizaría tanto la imagen que se perderían tantos detalles que no parecería ni la imagen original.

Hay estudios que aseguran haber reducido este ruido en imágenes médicas con pequeños implantes metálicos [22] [30], pero no detallan el comportamiento de su algoritmo para reproducirlo y además, en las imágenes de prueba que utilizan el ruido es ínfimo comparado con el caso de nuestros datos.

Otro estudio más reciente [12] usando escáneres más modernos aseguran que retocando los parámetros de este podría eliminarse prácticamente por completo. Por lo que, resolver este problema pasa a ser responsabilidad de la primera fase del flujo del modelado de conjuntos volumétrico, la obtención de datos.

6.4.2. Subdivisión de piezas de madera

El objetivo de esta sección es poder dividir el modelo de una escultura en las distintas piezas de madera que la conforman.

Anteriormente hemos analizado las soluciones que se utilizan actualmente y hemos visto como no son efectivas para resolver nuestro problema. Por tanto, es el momento de proponer e implementar una solución efectiva capaz de separar las distintas piezas de las que está compuesta la escultura.

Para abordar esta situación se va a seguir una de las máximas más importantes a la hora de solucionar problemas, que no es más que dividir el problema en sub-problemas más pequeños. Por lo que, en primer lugar se

intentará segmentar una sola imagen y posteriormente aplicar esta solución en todos los cortes del conjunto de datos.

Segmentación 2D

Lo primero que tenemos que hacer es preguntarnos a nosotros mismos cómo detectar el cambio entre dos piezas de madera en una escultura. Lo primero que se nos puede ocurrir es que se puede ver un cambio de continuidad en la curvatura de los anillos, pero analizar todos los anillos y realizar un estudio de su continuidad en cada corte puede ser muy costoso y además hay ocasiones en las que el cambio de continuidad no es tan obvio (Figura 6.14).

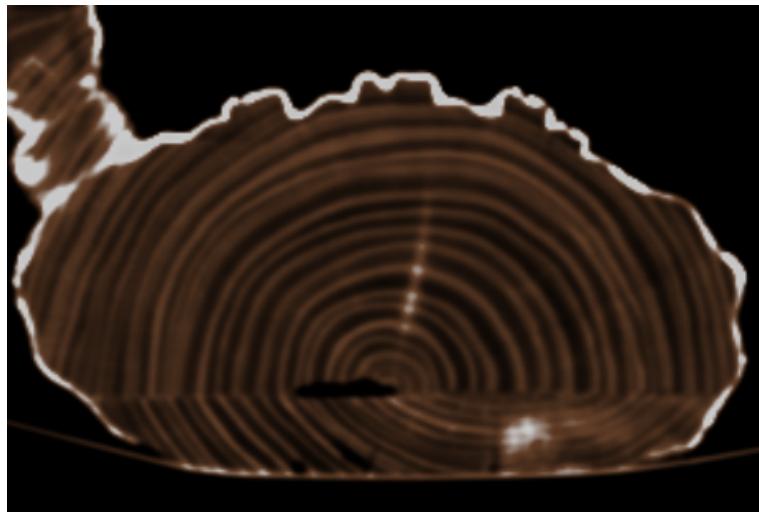


Figura 6.14: A la izquierda podemos ver continuidad en algunos anillos

Tenemos que buscar una solución computacionalmente menos costosa porque tendríamos que lanzar este algoritmo en todos los cortes del conjunto de datos volumétrico y, normalmente, trabajamos con datos de cientos de cortes.

Cuando los escultores ensamblan las piezas de madera que forman el embón de la escultura usan piezas cortadas previamente que encajan y se acoplan más fácilmente. Por tanto el límite entre dos piezas de madera distintas es una línea recta. la detección de este tipo de líneas por un ordenador se puede realizar rápidamente usando técnicas de visión por computador existentes como la transformada de *Hough* [15].

Pero aplicar este algoritmo directamente a una imagen de un corte no nos daría muy buenos resultados si previamente no aplicamos un algorit-

mo de detección de bordes. Para hacer esto, vamos a usar uno de los más utilizados, el algoritmo de detección de bordes de *Canny* [14]. Con este algoritmo y ajustando algunos parámetros podemos obtener con mayor o menor tolerancia los bordes de una imagen (Figura 6.15) que permitiría utilizar de forma más efectiva la transformada de *Hough* para detectar las líneas rectas de la imagen.



Figura 6.15: Corte al que hemos aplicado el algoritmo de detección de bordes de *Canny*. Podemos apreciar fácilmente la línea recta que separa las dos piezas de madera

Para evitar la generación de algunas líneas producidas por materiales que no son madera, vamos a aplicar previamente un filtro por umbral para quedarnos solo con los datos de la madera. A continuación aplicamos el algoritmo previamente citado de *Hough* para buscar las líneas.

Las líneas obtenidas las ordenamos de mayor a menor tamaño para des-
cartar líneas rectas que podrían haber anillos o en los bordes de la escultura
(Figura 6.16).

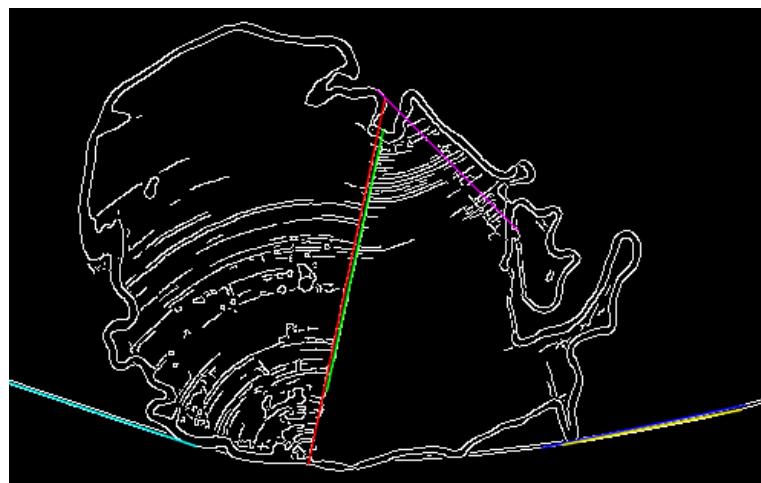


Figura 6.16: Corte al que hemos aplicado el algoritmo de detección de bordes de *Canny* para aplicar a continuación el algoritmo de *Hough* para detectar líneas rectas. Se muestran las cinco líneas más largas que se han encontrado

Una vez tenemos la línea que separa las dos piezas, la segmentación es sencilla porque solo tenemos que aplicar un algoritmo de crecimiento limitado no solo por el umbral, sino por la línea que hemos seleccionado.

Por tanto, para realizar la segmentación en una sola imagen nos hace falta una semilla (la usada en el algoritmo de crecimiento por umbral tradicional) y una línea que limite el crecimiento (Figura 6.17).

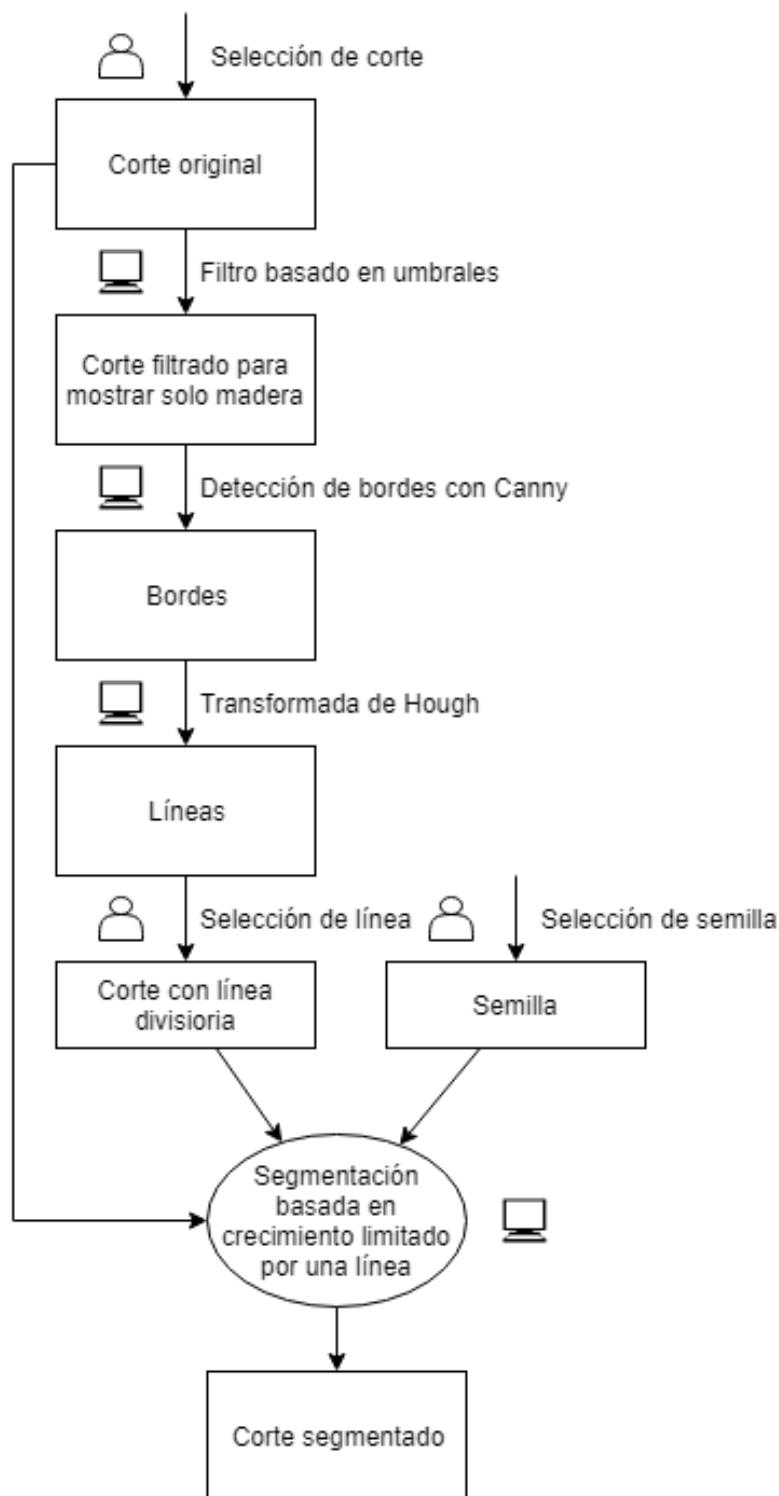


Figura 6.17: Proceso completo de la segmentación 2D donde se detalla cuándo interviene el ordenador y cuándo lo hace el usuario

Segmentación 3D

Para llevar a cabo la segmentación en la totalidad del conjunto de datos volumétrico se realiza la segmentación 2D corte a corte en el eje axial.

Pero como se ha explicado anteriormente, hacen falta tanto la línea que limite la zona de crecimiento como la semilla donde se comience a expandir que no serán los mismos en todos los cortes.

En el caso de la línea es muy difícil que coincida pues rara vez nos encontraremos con un montaje en el que el corte entre las dos piezas forme un ángulo exacto de 90° con respecto al eje axial. Sin embargo, la diferencia entre un corte y otro es milimétrica y la distancia será muy pequeña. Por ello, en lugar de pedir al usuario que elija la línea en cada corte, lo cual sería enormemente tedioso, se realizará una búsqueda de líneas y se escogerá la más cercana a la línea del corte anterior.

Para determinar el criterio de proximidad vamos a utilizar dos factores:

- **Ángulo entre dos líneas:** Dos líneas proyectadas en el mismo plano que forman un ángulo muy pequeño serán muy similares.

$$\cos(\alpha) = \frac{|u_0v_0 + u_1v_1|}{\sqrt{u_0^2 + u_1^2} \sqrt{v_0^2 + v_1^2}}$$

- **Distancia de un punto de una de ellas a la otra:** El ángulo solo no basta. Pues puede que tengan un ángulo muy parecido pero su punto de corte se encuentre muy lejos del espacio que se observa en el visor. Es por ello que hay que tener en cuenta este factor, pues los dos puntos que definen una línea se van a encontrar siempre en el espacio del visor pues es en este donde se lleva a cabo la operación de la búsqueda de líneas rectas y si la distancia mínima de uno de los dos puntos a la otra recta es pequeña significará que las dos líneas se cortan o en el propio espacio que vemos o cerca de éste.

$$d(P, r) = \frac{AP_0 + BP_1 + C}{\sqrt{A^2 + B^2}}$$

En determinados cortes puede ser que no se observe con suficiente nitidez el borde que separan las dos piezas de madera y por tanto, la transformada de *Hough* no encuentre aquí ninguna línea recta.

Para lidiar con este problema se omitirán este corte y todos los sucesivos donde no encuentre ninguna línea hasta que se de con uno en el que sí. Entonces se trazará un plano que pase por la línea del último corte donde se había encontrado hasta la línea de éste y se utilizará este plano para generar las líneas en todos los cortes intermedios.

Que la semilla de expansión varíe entre corte y corte pasa en menos ocasiones que con la línea pero, a no ser que se trate de una escultura muy regular y se escoja una semilla que sirva para todos los cortes, habrá ocasiones en las que habrá que cambiarla.

Un método que podría servir sería calcular el centroide del área inmediatamente anterior, pero el cálculo de éste es bastante costoso, por lo que, en vez de utilizar el centroide, se utiliza el punto medio:

$$M(x, y) = \left(\frac{x_{min} + x_{max}}{2}, \frac{y_{min} + y_{max}}{2} \right)$$

que puede ser calculado mientras se está realizando la segmentación.

Como habrá casos en los que este punto medio sea un píxel con un valor escalar fuera de los rangos de valores de madera, por ejemplo podría coincidir con una grieta en ésta, se realizará una expansión (respetando el límite de la línea previamente calculada) hasta encontrar el primer píxel cuyo valor corresponda a la madera para utilizarlo como semilla.

Como podemos ver en los ejemplos (Figura 6.18), se ha creado un método de segmentación semiautomática eficaz con el que poder segmentar dos piezas de madera en un conjunto de datos volumétrico adaptando la segmentación basada en el crecimiento de regiones tradicional para que se vea limitada por una línea que es detectada haciendo uso de técnicas de visión por computador.

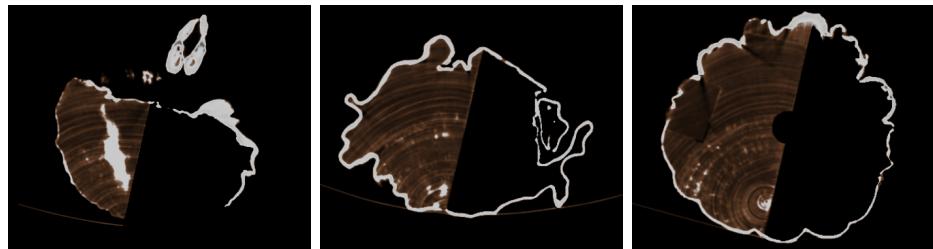


Figura 6.18: Resultado de segmentación en distintos cortes

Durante todo el proceso, la intervención del usuario solo se da en dos ocasiones: para elegir qué línea utilizar como separadora de piezas y para la selección de la semilla por donde comenzar la expansión.

Exportar e importar volumen

Para poder trabajar con el volumen resultado de la segmentación sin tener que segmentarlo todas las veces que se quiera trabajar con él es necesario poder exportarlo a un formato que luego pueda interpretar la aplicación para importarlo.

Se va a utilizar el formato VTI propio de VTK, que utiliza XML para guardar la información del volumen. Para ello hay que hacer uso de las clases `vtkXMLImageDataWriter` para exportar y `vtkXMLImageDataReader` para importar.

Además, gracias al uso de este tipo de archivo, se puede reducir bastante el tamaño de los archivos con datos volumétricos. Los datos en crudo en formato DICOM de una de las esculturas ocupan 236MB, mientras que esos mismos datos extraídos a formato VTI pasan a ocupar casi la mitad, 128MB.

Integración con la GUI

Para integrarlo con la interfaz gráfica se ha hecho algo parecido a lo que se hizo con la parte de borrar partes. Se puede activar un modo de segmentación que cambia el interactor del visor de cortes para que atienda al evento del click izquierdo del ratón para usar el píxel clickado como semilla (Figura 6.19).

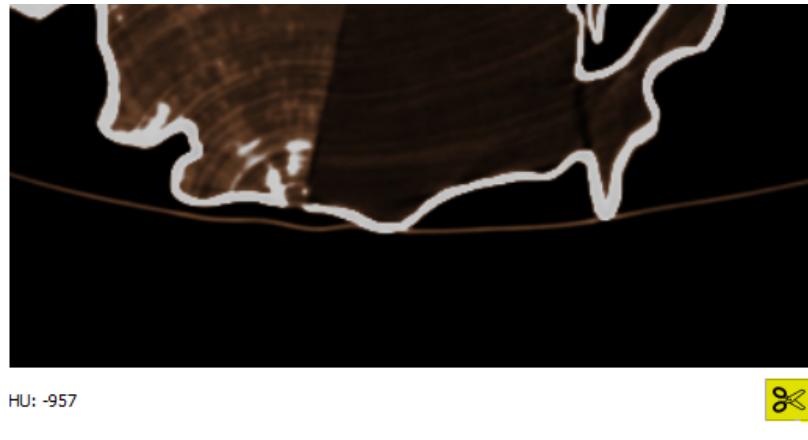


Figura 6.19: Botón que hay que pulsar para cambiar a modo de segmentación

A continuación se pasarán a detectar las líneas rectas de ese corte y se le mostrará una ventana al usuario donde elegirá la línea que quiere utilizar como semilla y si quiere que se fuerce hacia arriba y hacia abajo la segmentación (Figura 6.20). Si no se seleccionan estas opciones, si se deja de encontrar líneas coincidentes con la semilla en cortes superiores o inferiores no se forzará el crecimiento en estos cortes.

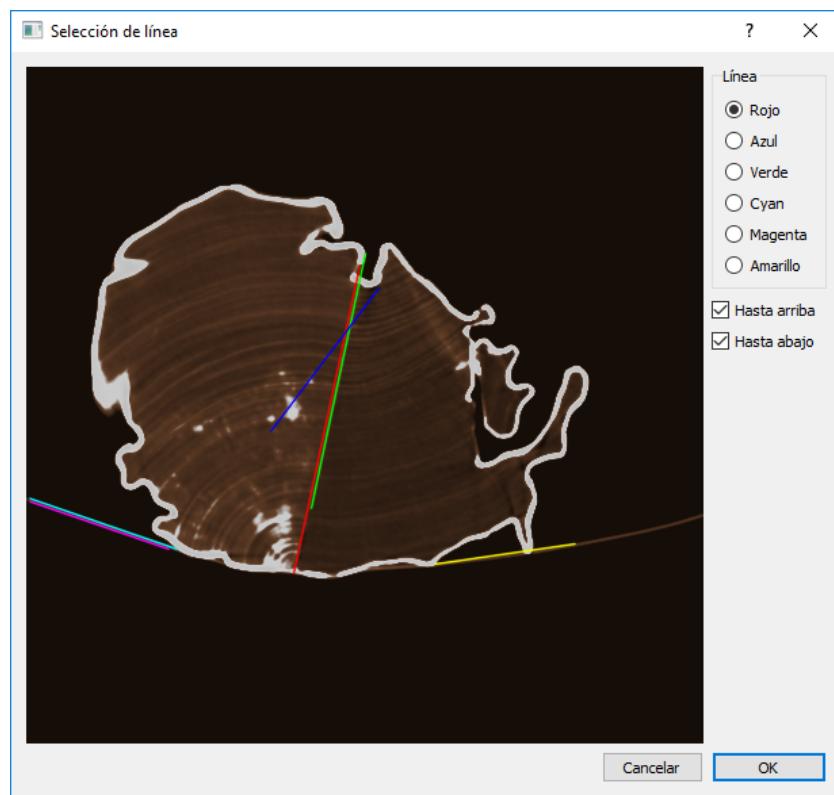


Figura 6.20: Líneas detectadas en el corte. El usuario tiene que elegir cuál quiere utilizar

Una vez seleccionados estos parámetros pasará a segmentar la pieza y al acabar mostrará otra ventana con el volumen resultado para preguntar al usuario si lo quiere o no exportar.

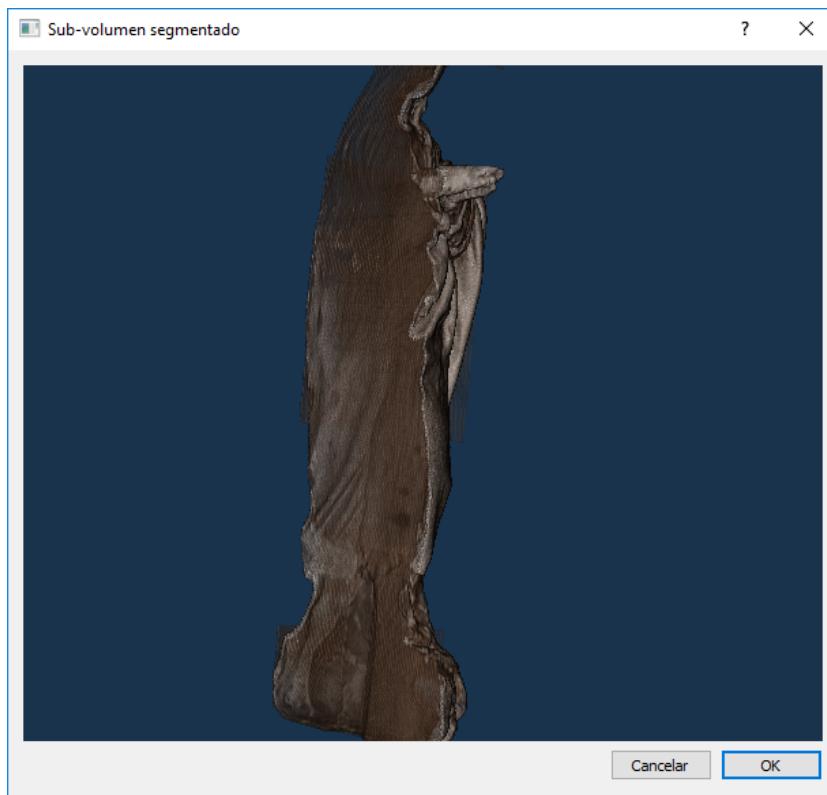


Figura 6.21: Resultado de la segmentación, si se pulsa en OK se pasará a guardar

6.4.3. Herramientas de documentación

Hasta ahora, el usuario ha podido analizar la escultura, pero todas las anotaciones las ha debido hacer a parte en una libreta o un documento digital. Lo complicado está en cuando ha tenido que colocar el plano en una posición extraña a la hora de encontrar un artefacto de interés. ¿Cómo lo vuelve a colocar así cuando abra de nuevo el programa? O más complicado todavía, ¿cómo lo pone un compañero que también esté trabajando en esa figura?

Es necesario poder proveer a la aplicación de funcionalidad para poder guardar anotaciones y dónde las ha encontrado para poder continuar con el trabajo en distintas sesiones.

ROD

Se ha creado un término, ROD (que viene de las siglas del inglés *Region of Documentation*), que no es más que la posición de un plano en una figura

donde realizar anotaciones.

Para crearlo, el usuario tan solo tiene que colocar el plano en la posición deseada y añadir una ROD (Figura 6.22).

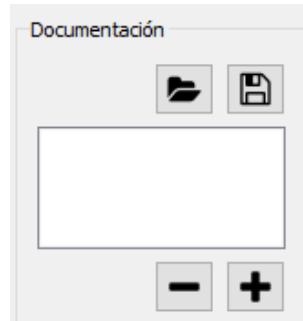


Figura 6.22: Listado de ROD y acciones para realizar con ellas

Al crear la ROD se le pedirá darle un nombre, y si no se proporciona ningún nombre se le pone uno por defecto.

Una vez esté la ROD creada, si el usuario mueve el plano de corte y quiere volver al plano de corte que había guardado, solo tiene que pulsar en el dentro de la lista de ROD.

Si se ha equivocado o ya no lo necesita lo puede borrar utilizando el botón específico para ello.

Reglas

La herramienta de las reglas ya se creó anteriormente en el TFG usando el *widget* de VTK *vtkDistanceWidget*. Sin embargo, al introducir este nuevo concepto de ROD, hay que adaptarlo para que, las reglas creadas no pertenezcan a un espacio global si no a una ROD específica. De esta forma, al mover el plano estas medidas, que ya dejan de tener sentido, desaparecen y puedes volver a verlas seleccionando de nuevo la ROD a la que pertenecen.

Las acciones a realizar son: añadir, eliminar y ocultar/mostrar (Figura 6.23).

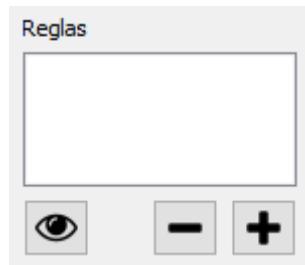


Figura 6.23: Listado de reglas y acciones para realizar con ellas

Al añadir una nueva regla habrá que seleccionar los puntos inicial y final que se quieren medir y se mostrará la medida real en milímetros. No hay límite de reglas, por lo que se pueden crear tantas como se quieran. Si se quiere editar tan solo hay que seleccionar y mover uno de los puntos.

Transportadores de ángulos

Otra funcionalidad útil parecida a la de medir distancias es medir ángulos. Además, VTK también proporciona un *widget* para realizar esta tarea: `vtkAngleWidget`. El funcionamiento con los ángulos es exactamente el mismo que el de las reglas. Pertenece a una ROD, se pueden añadir tantos como se quieran y se pueden eliminar y ocultar/mostrar (Figura 6.24).

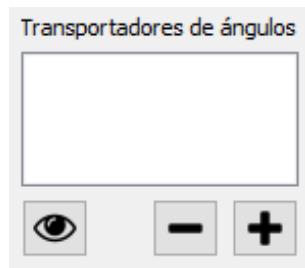


Figura 6.24: Listado de transportadores de ángulos y acciones para realizar con ellos

Para añadir un nuevo transportador de ángulos hay que seleccionar tres puntos en este orden: desde dónde, centro y hasta dónde. El resultado será el ángulo formado en grados. Para editarlo solo hay que seleccionar y mover cualquiera de sus puntos.

Notas

Además de medir distancias y ángulos, una de las tareas más utilizadas por un restaurador a la hora de examinar una escultura es realizar anotaciones de texto. Para realizar esta tarea se usará otro *widget* de VTK creado para ello, *vtkCaptionWidget* y se podrán realizar las mismas acciones que con reglas y ángulos (Figura 6.25).



Figura 6.25: Listado de notas y acciones para realizar con ellas

La diferencia con respecto a las anteriores es que antes de crearla hay que introducir en un cuadro de texto el contenido de ella. Esto creará el objeto y el usuario tendrá que mover la flecha a donde apunta.

Exportar e importar ROD

Todo lo que se ha desarrollado no tiene sentido si no hay posibilidad de guardar todas estas anotaciones para que otro usuario, o el mismo incluso, pueda cargarlos posteriormente.

Para esto se decidió usar XML almacenando esta información en un formato en el que el software pudiese exportar e importarla (Código 6.2).

Listing 6.2: Descripción de formato XML utilizado usando XML Schema

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3   <xs:element name="rod">
4     <xs:complexType>
5       <xs:attribute name="name" type="xs:string" use="required"/>
6       <xs:complexType>
7         <xs:sequence>
8           <xs:element name="pos">
9             <xs:complexType>
10               <xs:sequence>

```

```
11 <xs:element name="o">
12   <xs:complexType>
13     <xs:sequence>
14       <xs:element type="xs:float" name="x"/>
15       <xs:element type="xs:float" name="y"/>
16       <xs:element type="xs:float" name="z"/>
17     </xs:sequence>
18   </xs:complexType>
19 </xs:element>
20 <xs:element name="p1">
21   <xs:complexType>
22     <xs:sequence>
23       <xs:element type="xs:float" name="x"/>
24       <xs:element type="xs:float" name="y"/>
25       <xs:element type="xs:float" name="z"/>
26     </xs:sequence>
27   </xs:complexType>
28 </xs:element>
29 <xs:element name="p2">
30   <xs:complexType>
31     <xs:sequence>
32       <xs:element type="xs:float" name="x"/>
33       <xs:element type="xs:float" name="y"/>
34       <xs:element type="xs:float" name="z"/>
35     </xs:sequence>
36   </xs:complexType>
37 </xs:element>
38   <xs:element type="xs:float" name="slice"/>
39 </xs:sequence>
40 </xs:complexType>
41 </xs:element>
42 <xs:element name="rules">
43   <xs:complexType>
44     <xs:element name="rule" maxOccurs="unbounded" minOccurs="0">
45       <xs:complexType>
46         <xs:sequence>
47           <xs:element type="xs:string" name="name"/>
48           <xs:element name="p1">
49             <xs:complexType>
50               <xs:sequence>
51                 <xs:element type="xs:float" name="x"/>
52                 <xs:element type="xs:float" name="y"/>
53                 <xs:element type="xs:float" name="z"/>
54               </xs:sequence>
55             </xs:complexType>
56           </xs:element>
57           <xs:element name="p2">
58             <xs:complexType>
59               <xs:sequence>
60                 <xs:element type="xs:float" name="x"/>
61                 <xs:element type="xs:float" name="y"/>
62                 <xs:element type="xs:float" name="z"/>
63               </xs:sequence>
64             </xs:complexType>
65           </xs:element>
66         </xs:sequence>
67       </xs:complexType>
68     </xs:element>
69   </xs:complexType>
70 </xs:element>
71 <xs:element name="protractors">
72   <xs:complexType>
```

```

73  <xs:element name="protractor" maxOccurs="unbounded" minOccurs="0"
74      ">
75      <xs:complexType>
76          <xs:sequence>
77              <xs:element type="xs:string" name="name"/>
78              <xs:element name="p1">
79                  <xs:complexType>
80                      <xs:sequence>
81                          <xs:element type="xs:float" name="x"/>
82                          <xs:element type="xs:float" name="y"/>
83                          <xs:element type="xs:float" name="z"/>
84                      </xs:sequence>
85                  </xs:complexType>
86              </xs:element>
87              <xs:element name="p2">
88                  <xs:complexType>
89                      <xs:sequence>
90                          <xs:element type="xs:float" name="x"/>
91                          <xs:element type="xs:float" name="y"/>
92                          <xs:element type="xs:float" name="z"/>
93                      </xs:sequence>
94                  </xs:complexType>
95              </xs:element>
96              <xs:element name="o">
97                  <xs:complexType>
98                      <xs:sequence>
99                          <xs:element type="xs:float" name="x"/>
100                         <xs:element type="xs:float" name="y"/>
101                         <xs:element type="xs:float" name="z"/>
102                     </xs:sequence>
103                 </xs:complexType>
104             </xs:element>
105         </xs:complexType>
106     </xs:element>
107     </xs:complexType>
108 </xs:element>
109 <xs:element name="annotations">
110     <xs:complexType>
111         <xs:element name="annotation" maxOccurs="unbounded" minOccurs="0"
112             ">
113             <xs:complexType>
114                 <xs:sequence>
115                     <xs:element type="xs:string" name="name"/>
116                     <xs:element type="xs:string" name="text"/>
117                     <xs:element name="p">
118                         <xs:complexType>
119                             <xs:sequence>
120                                 <xs:element type="xs:float" name="x"/>
121                                 <xs:element type="xs:float" name="y"/>
122                                 <xs:element type="xs:float" name="z"/>
123                             </xs:sequence>
124                         </xs:complexType>
125                     </xs:element>
126                 </xs:sequence>
127             </xs:complexType>
128         </xs:element>
129     </xs:element>
130     </xs:sequence>
131 </xs:complexType>
132 </xs:schema>
```

Por ejemplo, una ROD colocada en el centro de la figura con dos reglas, un transportador de ángulos y dos anotaciones sería el siguiente (Código 6.3):

Listing 6.3: Ejemplo de ROD el formato XML descrito

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <rod name="Centro">
3   <pos>
4     <o>
5       <x>-0.2685546875</x>
6       <y>-0.2685546875</y>
7       <z>344.25</z>
8     </o>
9     <p1>
10    <x>274.7314453125</x>
11    <y>-0.2685546875</y>
12    <z>344.25</z>
13  </p1>
14  <p2>
15    <x>-0.2685546875</x>
16    <y>274.7314453125</y>
17    <z>344.25</z>
18  </p2>
19  <slice>344.25</slice>
20 </pos>
21 <rules>
22   <rule>
23     <name>Anchura pieza izquierda</name>
24     <p1>
25       <x>42.417439223592751</x>
26       <y>167.74032513989715</y>
27       <z>7.5979656197973364</z>
28     </p1>
29     <p2>
30       <x>124.99678864408101</x>
31       <y>148.54934957034706</y>
32       <z>7.5979656197973364</z>
33     </p2>
34   </rule>
35   <rule>
36     <name>Anchura pieza derecha</name>
37     <p1>
38       <x>122.67060978716583</x>
39       <y>140.98926828537276</y>
40       <z>7.5979656197973364</z>
41     </p1>
42     <p2>
43       <x>212.22849577839958</x>
44       <y>119.47211385890752</y>
45       <z>7.5979656197973364</z>
46     </p2>
47   </rule>
48 </rules>
49 <protractors>
50   <protractor>
51     <name>Angulo frontal</name>
52     <p1>
```

```

53   <x>128.48605692945375</x>
54   <y>168.32186985412591</y>
55   <z>7.5979656197973364</z>
56 </p1>
57 <p2>
58   <x>181.98817063850245</x>
59   <y>177.62658528178659</y>
60   <z>7.5979656197973364</z>
61 </p2>
62 <c>
63   <x>136.62768292865681</x>
64   <y>202.05146327939576</y>
65   <z>7.5979656197973364</z>
66 </c>
67 </protractor>
68 </protractors>
69 <annotations>
70 <annotation>
71   <name>Grieta</name>
72   <text>Posible grieta</text>
73   <p>
74     <x>180.27886141092529</x>
75     <y>95.373333133522266</y>
76     <z>-6.0396132539608516e-14</z>
77   </p>
78 </annotation>
79 <annotation>
80   <name>Ruido pan de oro</name>
81   <text>Ruido provocado por el pan de oro</text>
82   <p>
83     <x>104.67804856118269</x>
84     <y>188.42048741013227</y>
85     <z>-1.5898393712632242e-13</z>
86   </p>
87 </annotation>
88 </annotations>
89 </rod>

```

6.4.4. Internacionalización

Una vez terminadas el resto de secciones se pasó a realizar una tarea que acabó resultando tediosa por culpa de no haberla hecho desde un principio. Hablo de la internacionalización: preparar la aplicación para poder traducirla a distintos idiomas. Hasta ahora se tenían todos los textos *hard-coded*, se ha pasado a usar claves y dar los valores a esos claves en una archivo por idioma con el formato TS de Qt.

Una vez llenos estos archivos hay que compilarlos a formato QM con la herramienta `lrelease` que proporciona Qt y leer este archivo en el `main` usando la función `load()` de `QTranslator`. Para que los archivos sean accesibles desde la aplicación hay que añadirlos como recursos en un archivo QRC como ya se hace con las imágenes, los iconos y los *presets*.

Con estos cambios se puede generar un ejecutable en español e inglés. Con la posibilidad de añadir otros idiomas fácilmente traduciendo los títulos

al idioma que se quiere traducir en un nuevo archivo TS, que se compila a QM, se añade al archivo de recursos QRC y se lee con la función `load()` de `QTranslator`.

Capítulo 7

Resultados

En este capítulo se van a mostrar resultados obtenidos con las siguientes funcionalidades implementadas: Filtrado, segmentación y documentación.

Para realizar las pruebas se han utilizado la esculturas de Inmaculada Concepción y San Juan Evangelista (Figura 7.1), ambas patrimonio de la Universidad de Granada y cuyos datos DICOM han sido proporcionados por el proyecto de Portal Virtual de Patrimonio de las Universidades Andaluzas, coordinado por la Universidad de Granada.



Figura 7.1: Esculturas utilizadas para realizar las pruebas. Inmaculada Concepción (izquierda) y San Juan Evangelista (derecha)

7.1. Filtrado

Se implementaron los filtros de reducción de ruido *gaussiano*, media y mediana dando la posibilidad al usuario a elegir ciertos parámetros.

Para probar estos filtros se ha utilizado la escultura de San Juan Evangelista que es la que más ruido presentaba.

7.1.1. Filtro *gaussiano*

El filtro *gaussiano* es uno de los filtros de suavizado más utilizados. A continuación se va a mostrar los resultados obtenidos con éste para 1, 2 y 3 repeticiones (Figura 7.2).

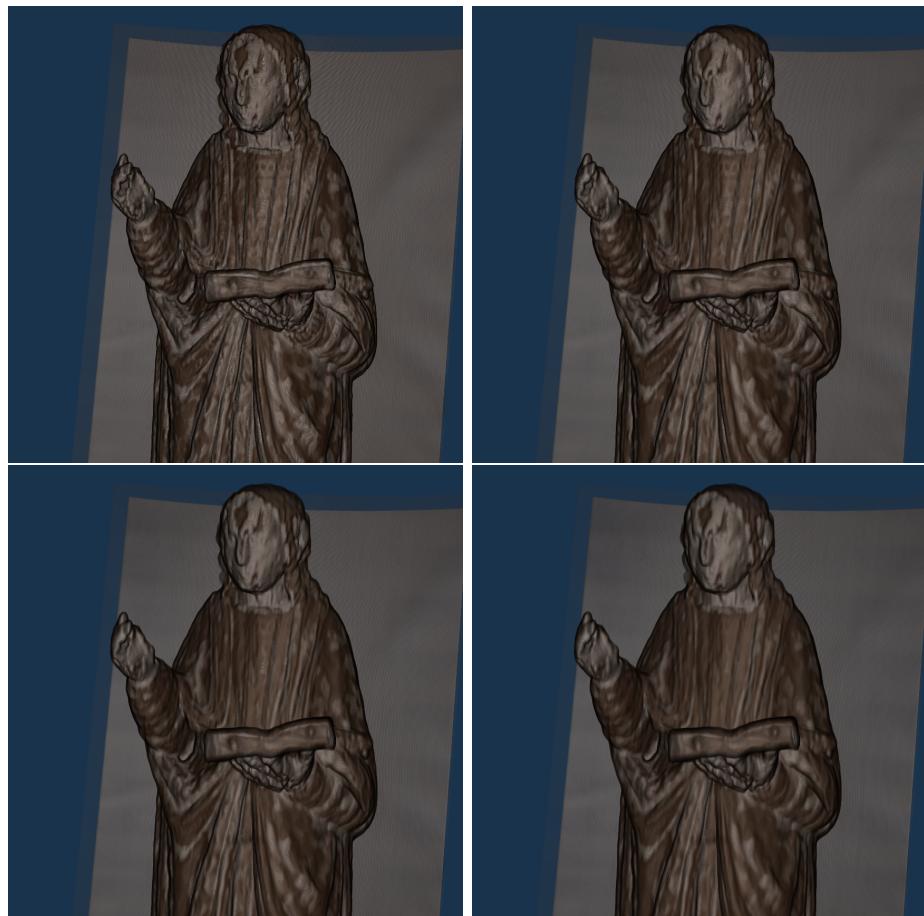


Figura 7.2: De izquierda a derecha y arriba a abajo: figura original y aplicando el filtro *gaussiano* con 1, 2 y 3 repeticiones

Se puede observar como apenas hay diferencia entre la original y la que se le ha aplicado el filtro con una sola repetición.

Con la que mejor resultado se obtiene es con la que se le aplica 2 repeticiones, porque con 3 ya empieza a suavizarse de más.

7.1.2. Filtro media

El filtro media es un filtro de suavizado bastante agresivo que usa una convolución y donde el único parámetro que se puede utilizar es el tamaño del vecindario para la convolución. A continuación se va a mostrar los resultados obtenidos con éste para vecindarios de 3x3, 5x5 y 7x7 (Figura 7.3).

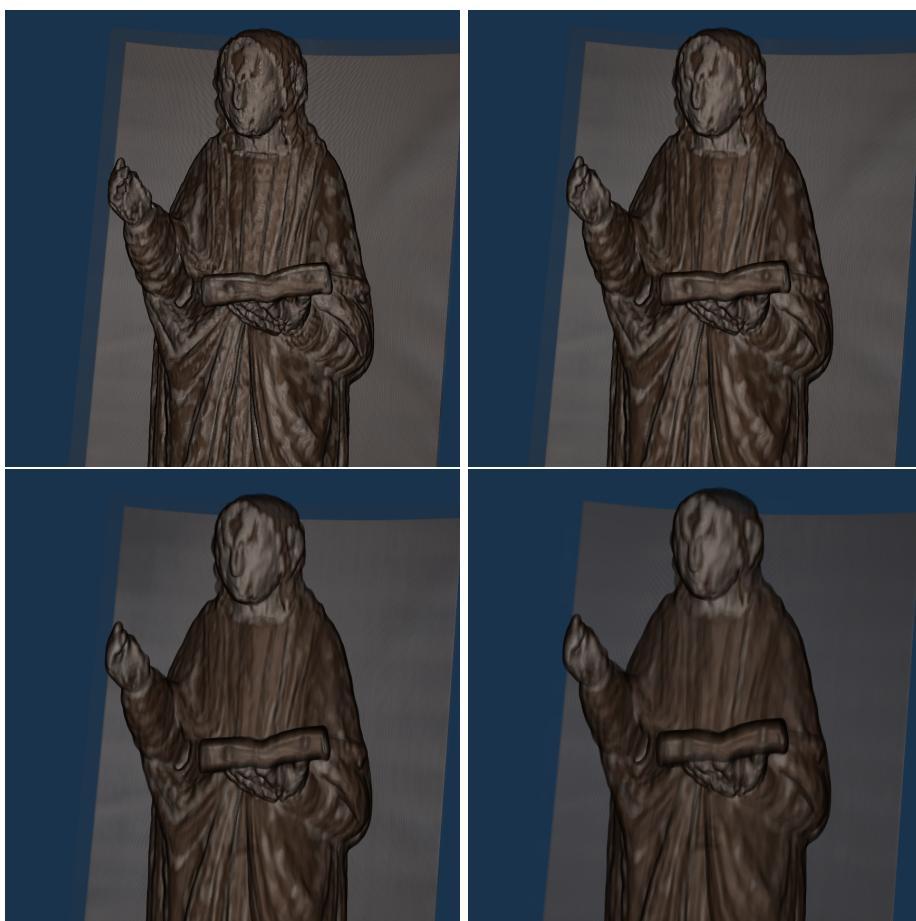


Figura 7.3: De izquierda a derecha y arriba a abajo: figura original y aplicando el filtro media con vecindarios de 3x3, 5x5 y 7x7

Se puede ver como este filtro es bastante más agresivo que el *gaussiano* y

se suaviza de más. Por tanto, habría que utilizarlo para imágenes con mucho más ruido.

7.1.3. Filtro mediana

El filtro mediana se utiliza para reducir el ruido de tipo *salt-and-pepper*. Nuestras imágenes no sufren de este tipo de ruido. No obstante se ha probado el filtro con vecindarios de 3x3, 5x5 y 7x7 (Figura 7.4).

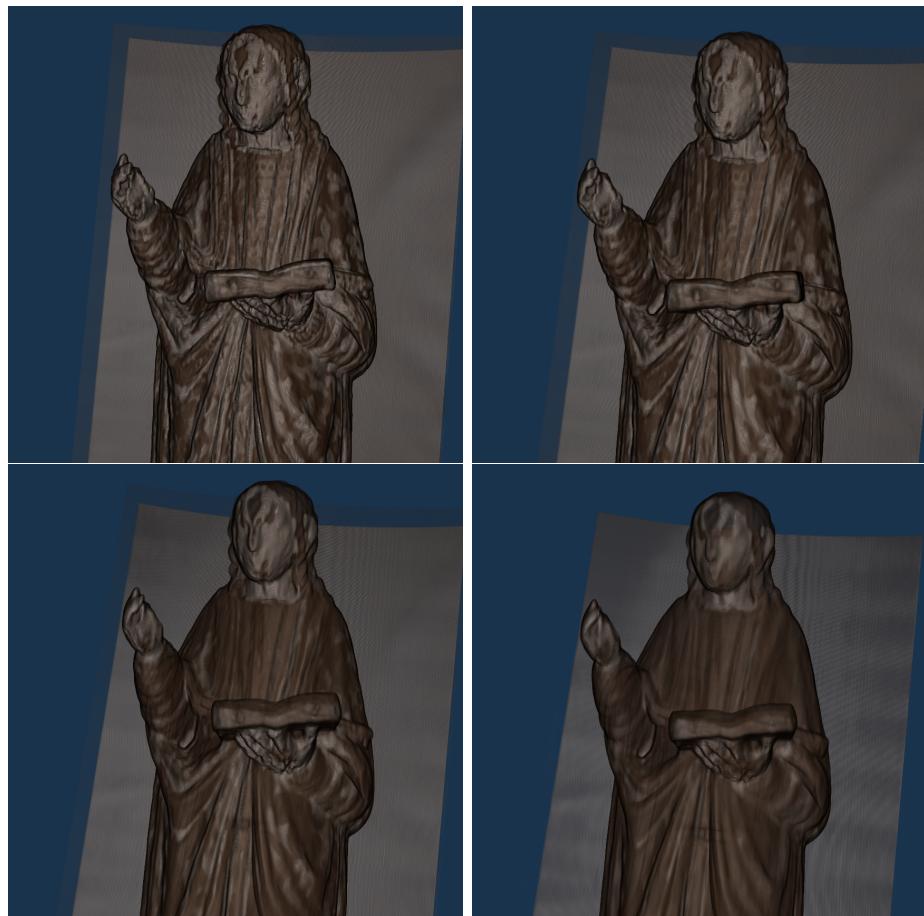


Figura 7.4: De izquierda a derecha y arriba a abajo: figura original y aplicando el filtro mediana con vecindarios de 3x3, 5x5 y 7x7

Los resultados con este filtro son incluso más agresivos que los obtenidos con el de media. Pero es lógico al ser un filtro creado para reducir un tipo de ruido que no presentan nuestras imágenes.

7.2. Segmentación

Se ha probado el método de segmentación propuesto para separar las piezas del embón de ambas esculturas.

7.2.1. Inmaculada Concepción

En la Inmaculada Concepción hay dos piezas principales en el embón a derecha e izquierda que se puede contemplar muy bien en cualquier corte axial desde la altura del pecho hasta abajo. Por tanto se podría usar como semilla cualquiera de estos cortes pues seguramente encuentre la línea que separa ambas partes entre las líneas rectas más grandes encontradas en este corte.

Se ha utilizado uno de estos cortes y se ha encontrado fácilmente esta línea (Figura 7.5).

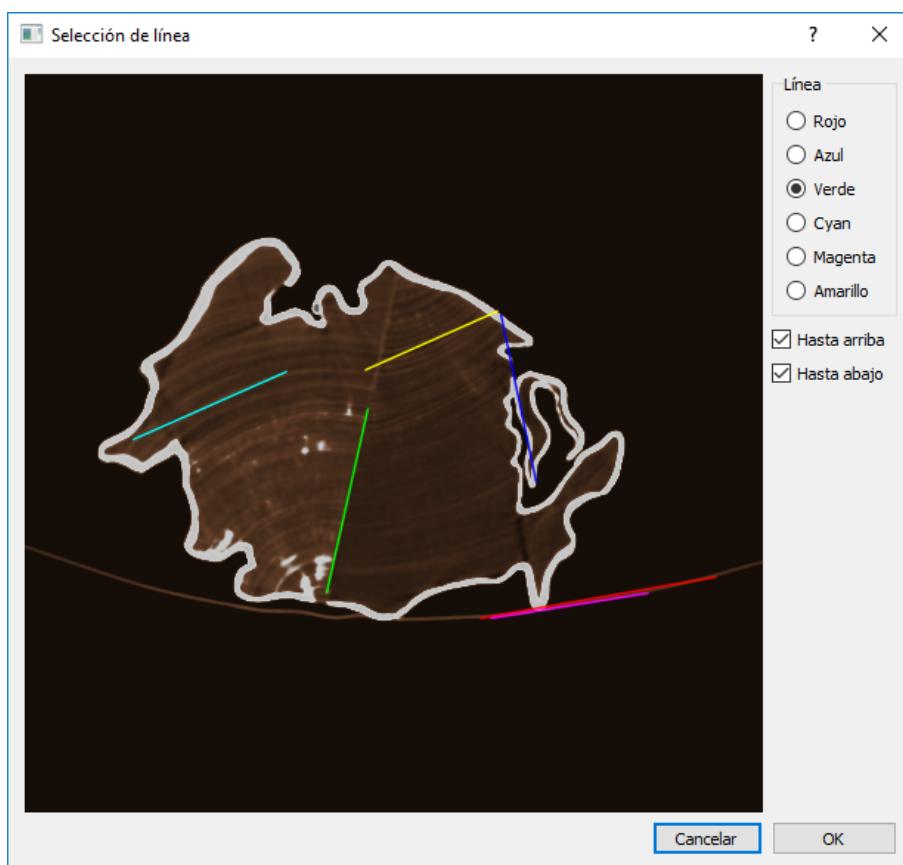


Figura 7.5: Líneas encontradas en un corte a la altura de la cintura donde se ve la línea (verde) que separa las dos piezas del embón

Esta línea divide perfectamente en dos la figura obteniendo los resultados de a continuación (Figuras 7.6 y 7.6).

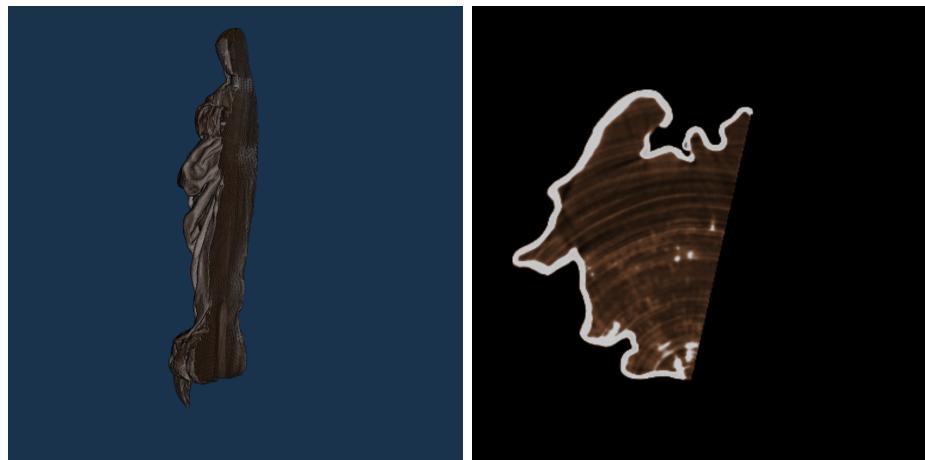


Figura 7.6: Volumen con la pieza de la derecha y un corte donde se ve dónde ha cortado

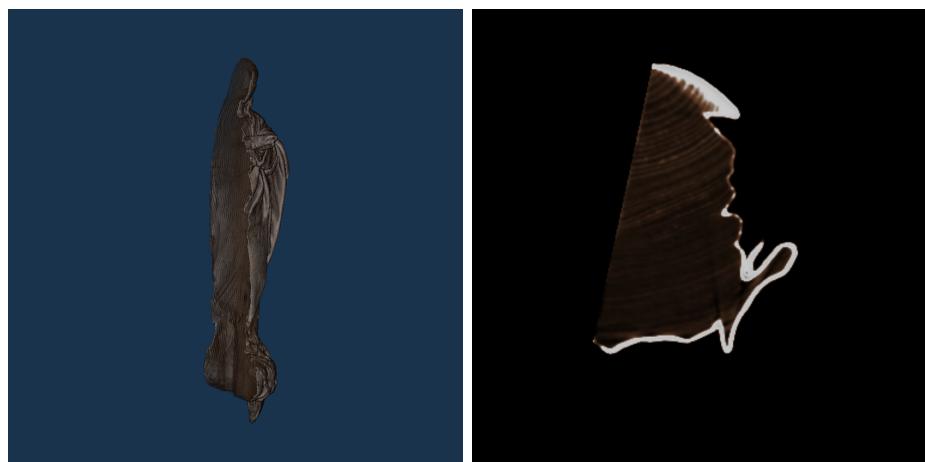


Figura 7.7: Volumen con la pieza de la izquierda y un corte donde se ve dónde ha cortado

7.2.2. San Juan Evangelista

En el San Juan Evangelista hay dos piezas principales en el embón una muy grande frontal y otra bastante más pequeña trasera. Esta pequeña no se puede ver por la zona de la cabeza y en la zona de la cintura hay un clavo que hace que no se diferencie muy bien la línea recta. Sin embargo, por la

zona del pecho se puede encontrar una buena semilla donde se detecte la línea que corta ambas piezas de madera.

Se ha utilizado un corte por esta zona y se ha encontrado esta línea (Figura 7.5).

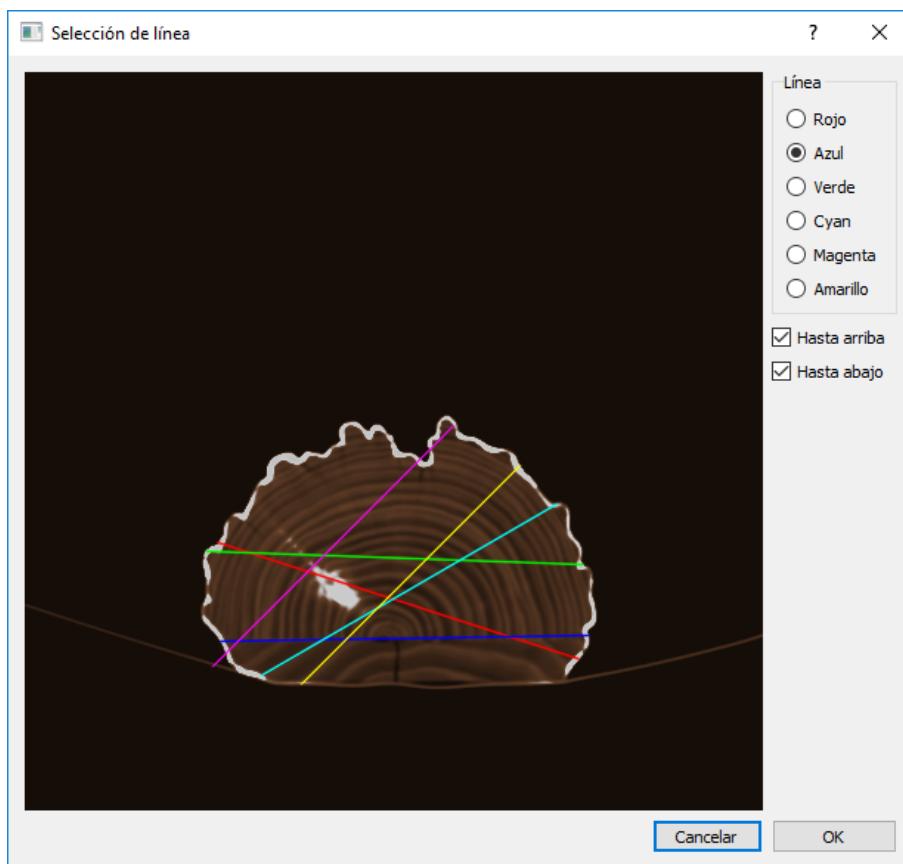


Figura 7.8: Líneas encontradas en un corte a la altura de la cintura donde se ve la línea (azul) que separa las dos piezas del embón

Esta línea divide perfectamente en dos la figura obteniendo los resultados de a continuación (Figuras 7.9 y 7.10).



Figura 7.9: Volumen con la pieza frontal y un corte donde se ve dónde ha cortado

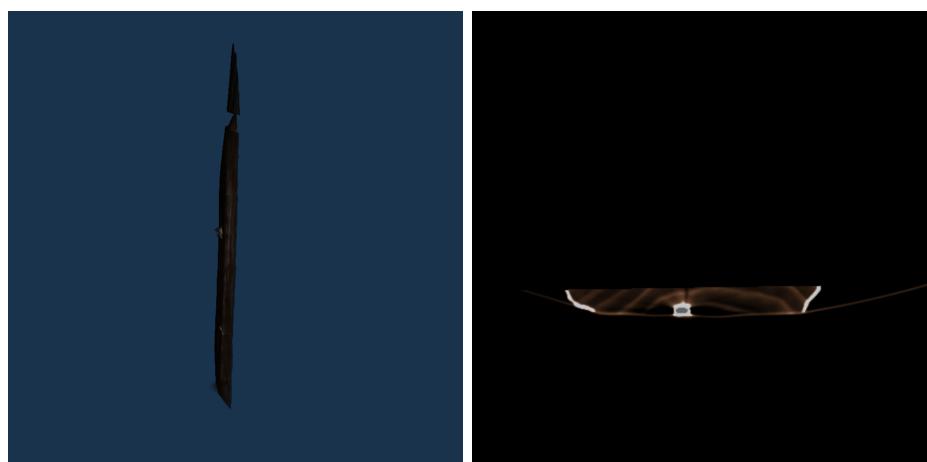


Figura 7.10: Volumen con la pieza trasera y un corte donde se ve dónde ha cortado

Capítulo 8

Conclusiones y trabajos futuros

TODO

Bibliografía

- [1] Boost. <http://www.boost.org/>.
- [2] Cmake. <https://cmake.org/>.
- [3] Especificación de requisitos según el estándar de iee 830. <https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf>.
- [4] Gitflow workflow. <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>.
- [5] Itk. <https://itk.org/>.
- [6] Opencv. <http://opencv.org/>.
- [7] Qt. <https://www.qt.io/>.
- [8] Radiant dicom viewer. <https://www.radiantviewer.com/>.
- [9] Staruml. <http://staruml.io/>.
- [10] Vtk. <http://www.vtk.org/>.
- [11] Serge Beucher and Christian Lantuéjoul. Use of watersheds in contour detection. 1979. https://www.researchgate.net/publication/230837989_Use_of_Watersheds_in_Contour_Detection.
- [12] F Edward Boas and Dominik Fleischmann. Ct artifacts: causes and reduction techniques. *Imaging Med*, 4(2):229–240, 2012. <http://www.edboas.com/science/CT/0012.pdf>.
- [13] Francisco Javier Bolívar and Francisco Javier Melero. 3dcurator: A 3d viewer for cts of polychromed wood sculptures. In Alejandro García-Alonso and Belen Masia, editors, *Spanish Computer Graphics Conference (CEIG)*. The Eurographics Association, 2016. <http://dx.doi.org/10.2312/ceig.20161311>.
- [14] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986. <http://dx.doi.org/10.1109/TPAMI.1986.4767851>.

- [15] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972. <http://dx.doi.org/0.1145/361237.361242>.
- [16] Universitätsklinikum Düsseldorf. Nuclear magnetic resonance: Relaxation. http://www.nmr.uni-duesseldorf.de/main/mtheorie_3_e.html.
- [17] 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>.
- [18] Robert Iván García Zavaleta. La historia y las generaciones de la tomografía computarizada. 2014. https://issuu.com/ivangz/docs/la_historia_de_la_tomografia_y_sus_.
- [19] Robert M Haralick and Linda G Shapiro. Image segmentation techniques. *Computer vision, graphics, and image processing*, 29(1):100–132, 1985. <http://dx.doi.org/10.1111/12.948400>.
- [20] 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.
- [21] Alejandro José León Salas. Técnicas de visualización gráfica: Introduction to image processing and analysis.
- [22] B. De Man, J. Nuyts, P. Dupont, G. Marchal, and P. Suetens. Metal streak artifacts in x-ray computed tomography: a simulation study. In *1998 IEEE Nuclear Science Symposium Conference Record. 1998 IEEE Nuclear Science Symposium and Medical Imaging Conference (Cat. No.98CH36255)*, volume 3, pages 1860–1865 vol.3, 1998. <https://doi.org/10.1109/NSSMIC.1998.773898>.
- [23] Donald W. McRobbie, Elizabeth A. Moore, Martin J. Graves, and Martin R Prince. *MRI from Picture to Proton*. 2010. <http://www.cambridge.org/ro/academic/subjects/medicine/medical-imaging/mri-picture-proton-2nd-edition?format=HB&isbn=9780521683845#051hhdYxoAdiMKXf.97>.

- [24] Eric N Mortensen and William A Barrett. Intelligent scissors for image composition. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 191–198. ACM, 1995. <http://dx.doi.org/10.1145/218380.218442>.
- [25] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979. <http://dx.doi.org/10.1109/TSMC.1979.4310076>.
- [26] 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>.
- [27] María Francisca Sarrió Martín. *Aplicación de la tomografía computarizada médica para el análisis y estudio en escultura policromada en madera*. November 2015. <http://hdl.handle.net/10251/61986>.
- [28] Klaus D. Toennies. *Segmentation: Principles and Basic Techniques*, pages 171–209. Springer London, 2012. https://doi.org/10.1007/978-1-4471-2751-2_6.
- [29] J. Gonzales Vasquez. *Manual Práctico de Tomografía*. 2011. https://www.academia.edu/10780497/MANUAL_PRACTICO_DE_TOMOGRAFIA.
- [30] Oliver Watzke and Willi A. Kalender. A pragmatic approach to metal artifact reduction in ct: merging of metal artifact reduced images. *European Radiology*, 14(5):849–856, May 2004. <https://doi.org/10.1007/s00330-004-2263-y>.