Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

Universidad de Jaén

Escuela Politécnica Superior
de Jaén

TRABAJO FIN DE GRADO

GENERACIÓN DE MODELOS 3D A PARTIR DE IMÁGENES MÉDICAS

Alumno

Rubén Martínez Parras

Tutor

Juan Roberto Jiménez Pérez
(Departamento de Informática)

Félix Paulano Godino
(HT Medica)

Noviembre, 2020

(Página intencionalmente en blanco)

Universidad de Jaén

Departamento de Informática

Don Juan Roberto Jiménez Pérez y Don Félix Paulano Godino, tutores del Trabajo Fin de grado titulado: '**Generación de modelos 3d a partir de imágenes médicas**', que presenta Don Rubén Martínez Parras, otorga el visto bueno para su entrega y defensa en la Escuela Politécnica Superior de Jaén.

Jaén, Noviembre de 2020

El alumno:

Rubén Martínez Parras

Los tutores:

JIMENEZ PEREZ, JUAN ROBERTO (FIRMA)
c=ES, serialNumber=44259364G, sn=JIMENEZ, givenName=JUAN ROBERTO, cn=JIMENEZ PEREZ, JUAN ROBERTO (FIRMA)
2020.11.17 08:34:05 +01'00'

PAULANO GODINO FELIX - 77361880T
Firmado digitalmente por PAULANO GODINO FELIX - 77361880T
Fecha: 2020.11.17 10:16:46 +01'00'

Juan Roberto Jiménez Pérez

Félix Paulano Godino

(Página intencionalmente en blanco)

Agradecimientos

Agradezco este trabajo de fin de grado a mis tutores Roberto y Félix, sin su ayuda no habría sido posible.

FICHA DEL TRABAJO FIN DE TÍTULO	
Titulación	Grado en Ingeniería Informática
Modalidad	Proyecto de Ingeniería
Especialidad (solo TFG)	Tecnologías de la Información
Mención (solo TFG)	Sistemas Gráficos
Idioma	Español
Tipo	Específico
TFT en equipo	No
Fecha de asignación	13/11/2019
Descripción corta	Los recientes avances en campos como la simulación médica o la impresión 3D, han hecho de la generación de modelos 3D a partir de imágenes médicas un proceso esencial. Uno de las áreas médicas que más ha aprovechado estos avances es la traumatología. En la actualidad, la generación de modelos 3D de estructuras óseas posibilita llevar a cabo una planificación virtual de la reducción de la fractura, imprimir guías quirúrgicas adaptadas al paciente que simplifiquen y hagan más precisa la cirugía, e incluso el diseño de prótesis personalizadas. Este Trabajo Fin de Grado plantea el desarrollo de un prototipo que obtenga un modelo 3D de un hueso a partir de imágenes de TAC (tomografías computarizadas). El prototipo contemplará tanto la segmentación de zonas óseas a partir de TAC, como la generación de modelos 3D de representación de frontera en las zonas segmentadas

NORMAS APLICADAS EN ESTE DOCUMENTO	
LOCALES	
TFT-UJA:2017	Normativa de Trabajos Fin de Grado, Fin de Máster y otros Trabajos Fin de Título de la Universidad de Jaén (Normativa marco UJA aprobada en Consejo de Gobierno)
TFT-EPSJ:2017	Normativa sobre Trabajos Fin de Grado y Fin de Máster en la Escuela Politécnica Superior de Jaén (Normativa EPSJ aprobada en Junta de Escuela)
TFT-EPSJ	Criterios de evaluación y normas de estilo para TFG y TFM de la Escuela Politécnica Superior de Jaén
NACIONALES	
UNE 157001:2014	Criterios generales para la elaboración formal de los documentos que constituyen un proyecto técnico
UNE 157801:2007	Criterios generales para la elaboración de proyectos de sistemas de información
INTERNACIONALES	
ISO 2145:1978	Documentación - Numeración de divisiones y subdivisiones en documentos escritos
APA 6ª edición	Estilo de referencias y citas de APA (American Psychological Association)

Contenido

Índice de ilustraciones

Índice de tablas

1 - ESPECIFICACIÓN DEL PROYECTO

En este capítulo se presenta la especificación del proyecto, con una estructura y contenidos que siguen los criterios y recomendaciones que establece la norma UNE 157801:2007 - "*Criterios Generales para la elaboración de proyectos de Sistemas de Información*".

1.1 - Índice general

Como índice de la documentación requerida por la norma UNE 157801 se utilizará en su lugar la tabla de contenido global del presente TFT, ya que todos los documentos indicados en dicha norma están contenidos en el mismo documento maestro.

1.2 - Memoria

A continuación, se presentan los contenidos de la especificación del proyecto definido en el presente TFT, y en el que se describen todos los elementos que deberán entregarse al final de la ejecución del proyecto, así como la especificación de los métodos, herramientas y métricas utilizadas para garantizar el éxito del mismo.

1.2.1 - Introducción

El proyecto consiste en una aplicación destinada a ayudar a los profesionales en medicina, generando modelos 3D de estructuras óseas los cuales sirven para llevar a cabo una planificación virtual de la reducción de la fractura, imprimir guías quirúrgicas adaptadas al paciente que simplifiquen y hagan más precisa la cirugía, e incluso el diseño de prótesis personalizadas.

Estos usuarios serán capaces, cargando un archivo DICOM (*Digital Imaging and Communication On Medicine*), de visualizar un TAC (tomografías computerizadas) segmentado mediante el método de umbralización y un modelo 3D generado automáticamente mediante estas imágenes. El modelo es guardado en un archivo de tipo STL (*Standard Triangle Language*) para un posible proceso de impresión 3D.

1.2.2 - Objeto del proyecto

Realizar un estudio de las bibliografías de técnicas de segmentación de tejido óseo a partir de imágenes tomográficas (TAC) y desarrollar un prototipo de aplicación que permita:

- Segmentar imágenes 3D.
- Generar modelos 3D

Además del estudio de uso de programas de edición de modelos 3D para hacer posible llevar a cabo una impresión 3D de dicho modelo.

1.2.3 - Antecedentes

Los recientes avances en campos como la simulación médica o la impresión 3D, han hecho de la generación de modelos 3D a partir de imágenes médicas un proceso esencial. Una de las áreas médicas que más ha aprovechado estos avances es la traumatología. En la actualidad, la generación de modelos 3D de estructuras óseas posibilita llevar a cabo una planificación virtual de la reducción de la fractura, imprimir guías quirúrgicas adaptadas al paciente que simplifican y hacen más precisa la cirugía, e incluso el diseño de prótesis personalizadas.

Mi devoción por la informática gráfica y mi respeto hacia la medicina hicieron que me decantara por este estudio. Más que la utilidad que pueda llegar a tener este prototipo en un futuro, buscaba la adquisición del conocimiento que pudiera aportarme este trabajo, pero sin duda, podría servir muy bien de ejemplo para los estudiantes que se decanten por la mención de Sistemas Gráficos.

1.2.4 - Descripción de la situación actual

En la actualidad los profesionales en medicina disponen de muchas herramientas complejas para la generación de modelos 3D a partir de imágenes tomográficas. La mayor parte de esta complejidad se haya en el proceso de segmentación. Para la generación del modelo existen algoritmos que permiten generarlo a partir de las imágenes, el más usado en medicina es el algoritmo “Marching Cubes”.

Como explicación rápida de este algoritmo, toma ocho localizaciones vecinas, creando un cubo imaginario y determina los polígonos necesarios para representar la

isosuperficie (superficie que representa puntos de valor constante dentro de un volumen de espacio). Los polígonos se van fusionando y creando la malla deseada.

El método de segmentación más utilizado en la medicina es el de umbralización, pero requiere un extenso post-procesamiento manual. Las investigaciones actuales y futuras se están centrando en el desarrollo de métodos de segmentación utilizando CNN (Convolutional Neural Network o Redes Neuronales Convolucionales) [1].

Desde el punto de vista de un usuario inexperto, el uso de estas herramientas complejas puede llegar a abrumar debido a que existen muchísimos métodos de segmentación, cada uno con sus ventajas y desventajas, por esto, tienen que ser elegidos por un usuario profesional. Cada situación específica deriva al uso de distintos métodos. Por este motivo, este prototipo busca ayudar a usuarios inexpertos que quieren estudiar esta área de conocimiento. estos podrán ver los resultados que se pueden obtener.

Desarrollar una aplicación extensible, hará investigadores que en un futuro quieran desarrollar herramientas de apoyo a la traumatología, puedan usarla para comparar distintas técnicas y perfeccionar su aplicación.

1.2.4.1 - Descripción del entorno actual

El entorno sobre el cual se va a enfocar mi proyecto es en la asignatura de Procesamiento de información visual (la segmentación tiene mucho que ver con esta asignatura) y en la mención de Sistemas Gráficos en general. Los alumnos podrían ver la gran utilidad que pueden tener las áreas de conocimiento mencionadas anteriormente.

1.2.4.2 - Resumen de las principales deficiencias identificadas

Las principales deficiencias encontradas y superadas con este proyecto son:

- Dificultad a la hora de estudiar y entender herramientas complejas del mismo tipo.
- Dificultad para poder experimentar con la generación de modelos 3D y la segmentación mediante umbralización de imágenes médicas.

- Dificultad a la hora de visionar rápidamente los distintos cambios que adquieren las imágenes segmentadas mediante umbralización cambiando sus parámetros.

1.2.5 - Normas y referencias

A continuación, se presentan las normas, reglamentos y referencias de cualquier tipo que han sido de aplicación en la elaboración del proyecto o en la ejecución del mismo. La norma UNE 157801 incluye aquí un apartado sobre bibliografía. Sin embargo, al integrarse el documento de especificación del proyecto en el documento global del TFG, la bibliografía de este capítulo se integra junto con la bibliografía del documento maestro.

Para el diseño de la interfaz se ha tenido en cuenta lo aprendido en la asignatura de IPO (Interacción Persona-Ordenador), manteniendo un mensaje sencillo y evitando el uso de colores si no son necesarios.

1.2.5.1 - Herramientas

Herramientas utilizadas:

- VTK: Sistema de software de código abierto y de libre acceso para gráficos en 3D, modelado, procesamiento de imágenes, representación de volúmenes, visualización científica y trazado en 2D.
- Visual Studio (VS): Entorno de desarrollo integrado para Windows compatible con VTK y con el lenguaje de programación utilizado, C++.
- Qt: Software utilizado para el desarrollo del diseño de la interfaz gráfica.
- CMake: Herramienta de software multiplataforma gratuita y de código abierto para administrar el proceso de compilación de software mediante un método independiente del compilador. Es importante entender que no es un sistema de compilación, concretamente es un generador de sistema de compilación.
- MeshMixer: Software de código abierto gratuito para edición y preparación de modelos 3D.
- GanttProject: Software de código abierto gratuito para planificaciones temporales y creación de diagramas de Gantt.

1.2.5.2 - Mecanismos de control de calidad aplicados durante la redacción del proyecto

El aseguramiento de la calidad en la redacción del proyecto implica la verificación de la completitud (falta de omisiones), la integridad de la documentación del proyecto, así como una redacción clara, concisa y entendible por todos los participantes e interesados en el proyecto. Además, deben establecerse mecanismos de verificación de la integridad y completitud de la documentación.

Al estar el proyecto desarrollado por un solo autor y verificado por dos tutores, no es necesario llevar un documento de control de edición y revisión de la documentación. De esta forma, el autor del proyecto ha utilizado mecanismos básicos para la verificación de la integridad y completitud de la documentación del proyecto, que incluyen un control de versiones a nivel de documentación y un control sencillo de la trazabilidad de requisitos y especificaciones del proyecto.

1.2.6 - Definiciones y abreviaturas

- AM: *Additive Manufacturing*, construcción de objetos 3D mediante la adición de capas ultrafinas, una sobre otra, de distintos tipos de materiales como plástico o metal.
- CAD: *Computer-aided Design*, uso de ordenadores para ayudar en la creación, modificación y análisis u optimización de un diseño.
- CNN: *Convolutional Neural Network* o redes neuronales convolucionales.
- DICOM: Estándar reconocido mundialmente para el intercambio de pruebas médicas, pensado para su manejo, visualización, almacenamiento, impresión y transmisión.
- ITK: *Insight Toolkit*, biblioteca de código abierto y multiplataforma que proporciona a los desarrolladores un amplio conjunto de herramientas de software para el análisis de imágenes.
- Nivel de detalle: Se refiere al uso de más o menos triángulos a la hora de creación del modelo 3D, recordemos que estos se van formando mediante la unión de triángulos. A más triángulos, más detalle.
- Qt: Software utilizado para el desarrollo del diseño de la interfaz gráfica.

- STL: *Standard Triangle Language*, formato de CAD que define geometría de objetos 3D.
- TAC, TC o CT: Tomografía computerizada, se refiere a la obtención de imágenes de cortes o secciones de algún objeto, en este caso, de huesos.
- Valores de umbralización: El valor mínimo hará que todos los niveles de grises menores al umbral especificado se conviertan en negro y el valor máximo hará que todos los niveles de grises mayores al umbral especificado se conviertan en blanco. Posible en las imágenes compuestas con escala de grises.
- VTK: *The Visualization Toolkit*, sistema de software de código abierto y de libre acceso para gráficos en 3D, modelado, procesamiento de imágenes, representación de volúmenes, visualización científica y trazado en 2D.

1.2.7 - Requisitos iniciales

- El usuario debe poder modificar los parámetros para poder obtener el resultado deseado, estos parámetros son la ruta del archivo DICOM, nombre deseado para el archivo STL generado, valores de umbralización, nivel de detalle de modelo, colores del modelo 3D generado y del fondo de la escena.
- El usuario debe poder moverse libremente entre todas las imágenes o *slices* del archivo DICOM especificado.
- El usuario debe poder ver la escena 3D del modelo generado e interactuar en ella con la cámara.
- El usuario debe obtener un archivo de formato STL que corresponda al modelo generado, para que este pueda aplicarle una limpieza con herramientas como MeshMixer y pueda obtener un modelo imprimible.

1.2.8 - Alcance

El alcance del proyecto queda desde la lectura de archivos DICOM a la generación del modelo 3D. Este proyecto está modulado para posibles cambios futuros, como el uso de un método de segmentación distinto a umbralización.

1.2.9 - Hipótesis y restricciones

El TFT se define como una asignatura de 12 créditos, lo que supone que la duración total del proyecto será de 300 horas, incluyendo todas las etapas del ciclo de vida, con la excepción del mantenimiento. Por consiguiente, la principal restricción aplicable es la limitación de la duración del trabajo.

1.2.10 - Estudio de alternativas y viabilidad

Respecto al estudio de alternativas, no se han podido contemplar alternativas debido al uso obligado del sistema de software VTK. Procederé a realizar una serie de explicaciones en los distintos ámbitos en los que VTK interviene.

En cuanto al IDE utilizado, Visual Studio, era la elección más razonable debido a que el ordenador donde ha sido desarrollado tiene instalado como sistema operativo Windows 10. Donde sí podemos indagar es sobre las distintas versiones de VS:

- Visual Studio 2019: Primera opción escogida debido a ser la última versión compatible con las demás herramientas utilizadas como pueden ser CMake, VTK y el lenguaje C++. Luego deshechada por la imposibilidad de vinculación con la biblioteca ITK, la cual sirve de ampliación para el sistema VTK. Por último retomada deibido a la decisión de no utilizar ITK, ya que, VTK nos proporcionaba todo lo necesario y el uso de ITK hacía los procesos de instalación y desarrollo bastante más complicados, a veces incompatibles con el ordenador utilizado.

- Visual Studio 2017: Tenido en cuenta debido a que era la versión compatible con la biblioteca ITK y la única que no daba problemas, a la hora de descartar ITK, se volvió al uso de VS 2019.

- Qt: Cuenta con su propio entorno de desarrollo, pero debido al uso de CMake, cuya configuración es más fácil de realizar con Visual Studio a la hora de la instalación de VTK y la generación de proyectos, se descartó de inmediato para este uso.

  En cuanto a la herramienta para el desarrollo de la interfaz, fue decantado rápidamente al uso de Qt, ya que, VTK cuenta con módulos que en principio hace posible la vinculación y el trabajo entre ambas herramientas. Debido a problemas técnicos la vinculación no pudo llevarse a cabo, pero como

sabemos, la implementación de una interfaz es crucial en una aplicación, por eso, se ha diseñado la interfaz para su posible implementación en un futuro. Para el diseño, si es verdad que pueden usarse muchas herramientas, pero para que fuera lo más cercano posible debido al vínculo con VTK, igualmente se usó Qt.

Pasando a la elección de lenguajes de programación, las alternativas más viables eran Python y C++. Este proyecto ha requerido un gran proceso de aprendizaje para el uso de un gran número de herramientas con las cuales no me sentía familiarizado, por ello, rápidamente decidí utilizar el lenguaje con el que más conocimiento contaba, C++ en este caso.

Por último, podemos pasar al uso de herramientas de reparación de archivos STL. Esta parte no era necesaria para este proyecto, pero decidimos incluirla para poder entregar un trabajo completo, con todas sus partes, para poder llegar a obtener un modelo imprimible. Procederé a enumerar las diferentes alternativas:

- MeshMixer: Programa de edición y reparación de archivos STL compatible con Windows y Mac. Fue el programa elegido debido a que cuenta con una dificultad intermedia y una gran comunidad, lo que hace posible encontrar tutoriales e información de cualquier tipo rápidamente.
- MeshLab: Programa de edición y reparación de archivos STL compatible con Windows, Mac y Linux. Tenido en cuenta debido a que cumplia con lo necesario, pero descartado por tener una alta dificultad de aprendizaje, ya que, es una herramienta muy profesional.
- MakePrintable: Programa de edición y reparación de archivos STL de navegador. Cuenta con una dificultad intermedia, pero debido a ser un programa de navegador, supuse que no contaría con lo necesario y que la computación sería lenta, así que lo descarté.

Poco se puede hablar en cuanto a alternativas sobre herramientas generadoras de sistema de compilación, CMake es la herramienta que cuenta con soporte oficial para VTK.

1.2.11 - Descripción de la solución propuesta

La solución propuesta trata sobre una aplicación de escritorio, simple e intuitiva que permite a usuarios tanto novatos como expertos experimentar y adquirir modelos 3D a partir de imágenes médicas de forma rápida y sencilla.

La aplicación cuenta con características como:

- Posibilidad de parametrizar distintas utilidades de la aplicación como pueden ser la ruta del archivo DICOM, el nombre del modelo que será generado automáticamente en la raíz del ejecutable, los niveles de umbralización para la segmentación de las imágenes, el nivel de detalle del modelo y los colores del modelo generado y el fondo de la escena de visualización 3D.
- Un visualizador de las imágenes segmentadas a partir del archivo DICOM y la interacción correspondiente.
- Un visualizador de la escena 3D del modelo generado y la interacción correspondiente.
- Un generador automático de archivos STL para un posible proceso de limpieza del modelo generado.

1.2.12 - Organización y gestión del proyecto

- Organigrama y Matriz de responsabilidades en el proyecto:
    - Tutores: Juan Roberto Jiménez Pérez y Félix Paulano Godino.
    - Desarrollador, investigador y jefe de proyecto: Rubén Martínez Parras.
- Directrices para el seguimiento del proyecto:

  Para un correcto seguimiento se han realizado varias videoconferencias y un intercambio de información constante vía GMail.
- Directrices a seguir para la aprobación de los entregables:

  Se debe entregar el código fuente del proyecto junto con el manual de instalación y de uso del mismo abordando todas las posibles dudas que

puedan surgirle al usuario, además de la documentación asociada al proyecto.

- Lugar donde se realizará el trabajo:

  El desarrollador realizará el trabajo desde su domicilio.

1.2.13 - Planificación temporal

Para la realización, se ha seguido la metodología de análisis, diseño, implementación y pruebas.

Las distintas tareas han sido:

- Estudio exhaustivo de las herramientas VTK e ITK.
- Instalación de las herramientas Visual Studio 2019, CMake y VTK.
- Realización de pruebas con las herramientas instaladas y comprobación de su correcto funcionamiento.
- Instalación de la biblioteca ITK.
- Realización de pruebas con la biblioteca ITK.
- Comienzo de desarrollo, implementación del lector de archivos DICOM.
- Implementación de segmentación de imágenes mediante umbralización.
- Implementación de generación de modelos 3D a partir de las imágenes segmentadas.
- Implementación de guardado automático de modelos 3D en archivos de formato STL.
- Pruebas de funcionamiento de la aplicación.
- Estudio sobre impresión de modelos 3D y posibles herramientas para la limpieza de los mismos.
- Pruebas con la herramienta elegida para la limpieza de los modelos generados.
- Instalación de Qt y realización del diseño de la interfaz gráfica.

Nombre	Fecha de inicio	Fecha de fin
1. Estudio exhaustivo de la...	4/02/20	16/03/20
2. Instalación de las herra...	17/03/20	30/03/20
3. Realización de pruebas ...	31/03/20	2/04/20
4. Instalación de la bibliote...	3/04/20	6/04/20
5. Realización de pruebas ...	7/04/20	8/04/20
6. Comienzo de desarrollo,...	9/04/20	15/04/20
7. Implementación de seg...	16/04/20	24/04/20
8. Implementación de gen...	27/04/20	5/05/20
9. Implementación de gua...	6/05/20	12/05/20
10. Pruebas de funcionami...	13/05/20	19/05/20
11. Estudio sobre impresió...	20/05/20	2/06/20
12. Pruebas con la herrami...	3/06/20	9/06/20
13. Instalación de Qt y real...	10/06/20	23/06/20

Ilustración 1.2.13 Planificación temporal

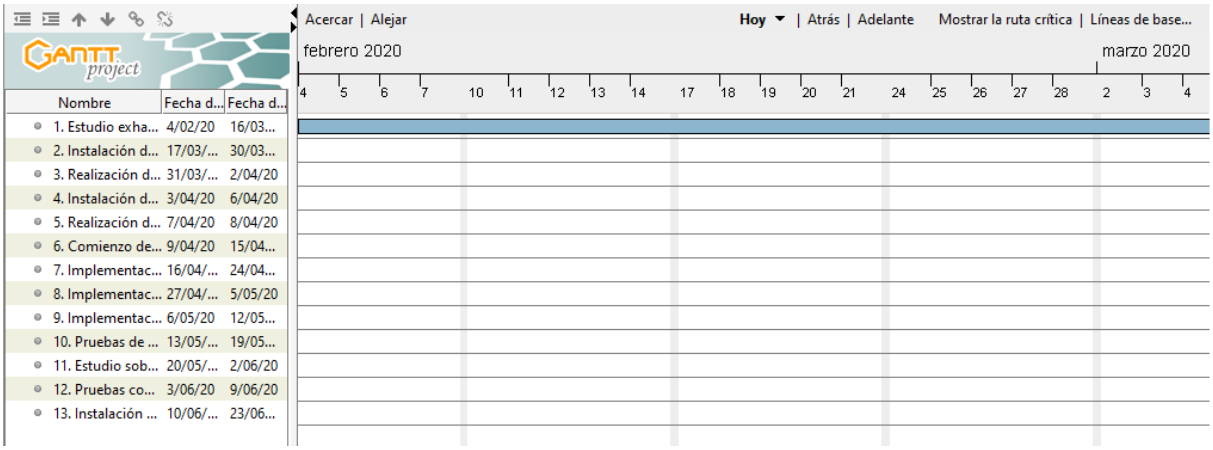


Ilustración 1.2.13 Planificación temporal: Diagrama de Gantt 1

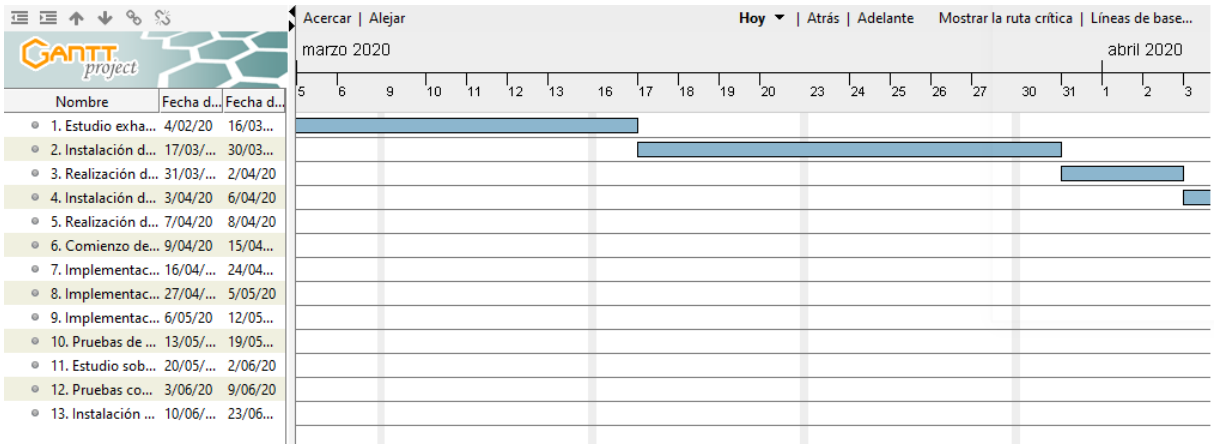


Ilustración 1.2.13 Planificación temporal: Diagrama de Gantt 2

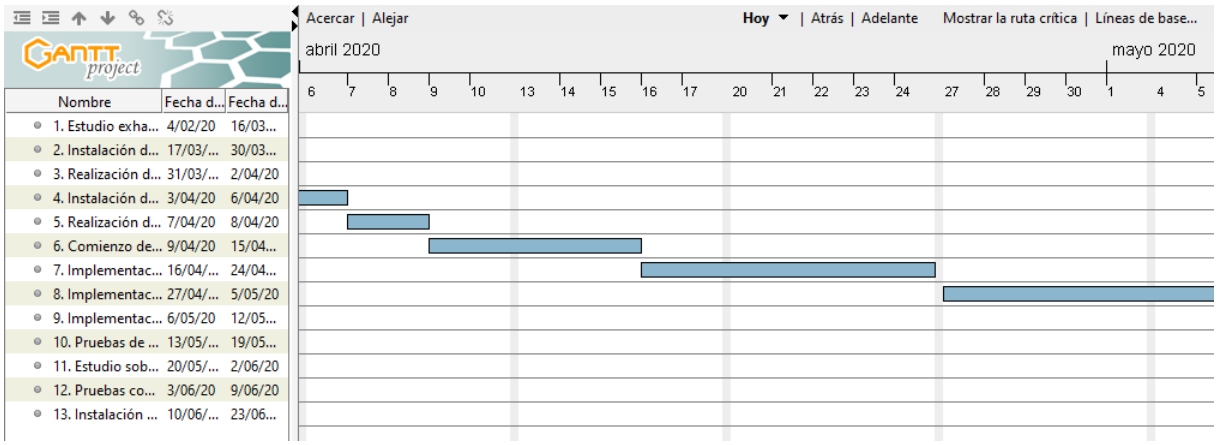


Ilustración 1.2.13 Planificación temporal: Diagrama de Gantt 3

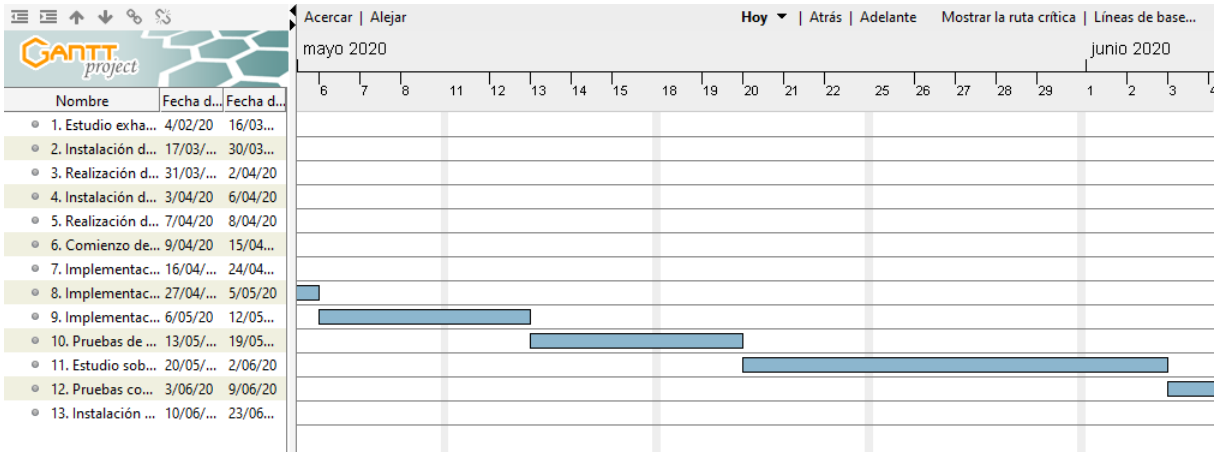


Ilustración 1.2.13 Planificación temporal: Diagrama de Gantt 4

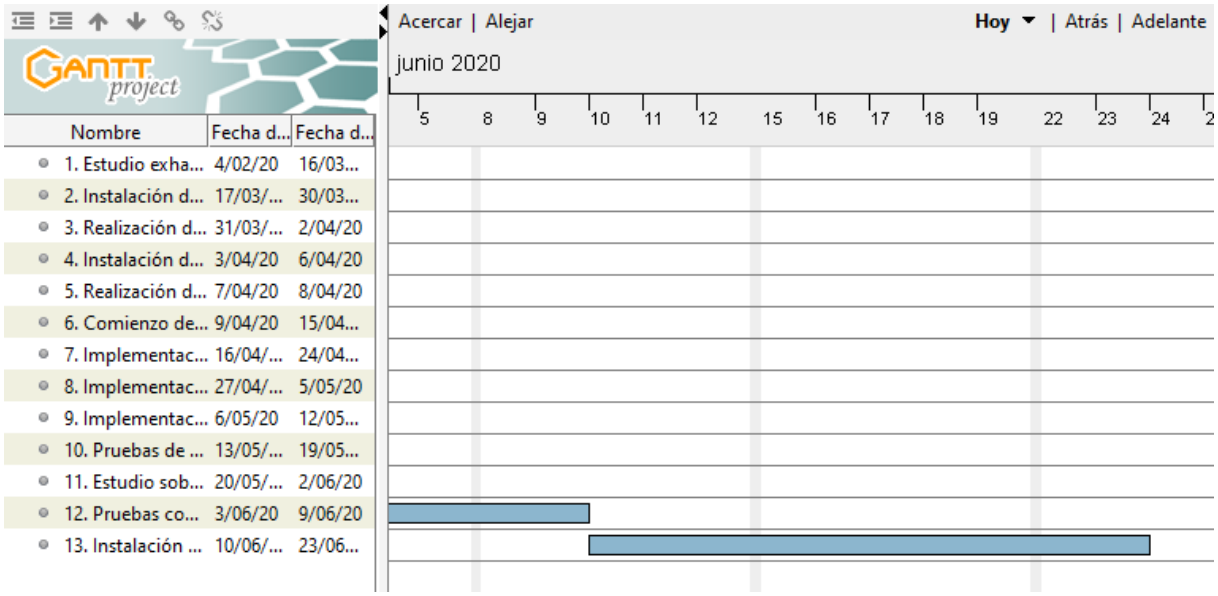


Ilustración 1.2.13 Planificación temporal: Diagrama de Gantt 5

1.2.14 - Resumen del presupuesto

Para la realización del proyecto es necesario:

- Un desarrollador informático con conocimientos sobre C++, VTK e impresión 3D. Coste aproximado: 7000€.
- Equipo informático con el hardware y el software necesario para la compatibilidad de las herramientas utilizadas. Coste aproximado 1000€.
- Facturas de servicios eléctricos e internet. Coste aproximado 200€.

Para el correcto desarrollo del proyecto será necesario disponer de un presupuesto aproximado de 8200€. Se verá con detalle en la sección 1.4 Presupuesto.

1.2.15 - Orden de prioridad de los documentos básicos del proyecto

El proyecto se presenta dentro de este capítulo del documento global con los contenidos indicados en la norma UNE 157801. El orden de prioridad utilizado es el siguiente:

1. Memoria del proyecto.
2. Especificaciones del sistema.
3. Presupuesto.
4. Estudios con entidad propia.
5. Anexos según la norma (numerados como A1...AN).

1.3 - Especificaciones del sistema

- Requisitos funcionales:
  - El usuario debe tener la posibilidad de poder modificar la ruta donde el programa buscará el archivo DICOM para el resto del proceso.
  - El usuario debe poder modificar el nombre del archivo STL que se generará automáticamente.

- El usuario debe poder modificar los valores mínimos y máximos de umbralización para poder obtener un resultado que se ajuste a los diferentes tipos de ejemplo utilizados.
- El usuario debe poder modificar el nivel de detalle del modelo 3D para poder obtener un resultado que se ajuste a los diferentes tipos de ejemplos utilizados.
- El usuario debe poder modificar el color del modelo 3D.
- El usuario debe poder modificar el color de fondo de la escena 3D.
- El usuario debe poder moverse libremente entre las imágenes o *slices* del archivo DICOM mediante teclado o ratón.
- El usuario debe poder girar el modelo con el ratón en la ventana de visualización 3D.
- El sistema debe de mostrar las distintas escenas de forma apropiada, cuando se cierre una, aparecerá la siguiente. Las distintas escenas son la interfaz de parametrización, el visor de imágenes o *slices* del archivo DICOM y el visor de la escena 3D del modelo.
- El sistema debe poder generar automáticamente un objeto 3D de formato STL correspondiente a los parámetros indicados por el usuario.

- Requisitos no funcionales:
  - Un usuario que llegue a utilizar este programa debe de ser capaz de usar la aplicación después de unos 20 minutos de entrenamiento y habiendo realizado antes un estudio sobre umbralización.

1.4 - Presupuesto

Componente	Meses	Horas/día	Cantidad	Costo unitario	Costo total
Mano de obra					
Desarrollador informático	3	8h	480	14,583€	7000€
Hardware					
Equipo informático	3	8h	480	2,083€	1000€
Software					
VTK	0	0	1	0	0
Visual Studio	0	0	1	0	0
CMake	0	0	1	0	0
MeshMixer	0	0	1	0	0
Qt	0	0	1	0	0
Servicios					
Factura electricidad	3	0	3	0,10€	48€
Factura internet	3	0	3	50€	150€
					Total
					8198€

Tabla 1.4 Presupuesto

Las cantidades del Desarrollador informático y el Equipo informático, se refieren al uso total en los 3 meses. Si cada mes tiene de media unos 20 días laborales, 3 meses son 60 días a 8 horas de trabajo, lo que hacen 480 horas. El coste unitario es el costo total entre la cantidad, por tanto, el desarrollador cobrará 14,583€ la hora y el uso del equipo por hora costará 2,083€.

La cantidad de las facturas se refiere a las veces que son cobradas, una vez al mes en ambos casos, por tanto, llegan 6 facturas en total. Se ha estimado que una buena conexión a internet puede llegar a costar unos 50€ al mes y que el costo actual del Kw/h puede salir por unos 0,10€ la hora.

1.5 - A1: Documentación de entrada

Como equivalente al pliego de condiciones se incluye la documentación de entrada especificada en el Apéndice Guía original del Trabajo Fin de Título.

1.6 - A2: Análisis y Diseño del sistema

1.6.1 - Análisis del sistema

Partiendo de los requisitos iniciales, el sistema será modelado de la siguiente manera:

La herramienta VTK, es la única de uso obligatorio, para un mayor entendimiento, vamos a explicar en qué consiste. VTK es un sistema de software de código abierto y de libre acceso para gráficos en 3D, modelado, procesamiento de imágenes, representación de volúmenes, visualización científica y trazado en 2D. Admite una amplia variedad de algoritmos de visualización y técnicas avanzadas de modelado, y aprovecha tanto el procesamiento paralelo de memoria enhebrada como el distribuido para lograr velocidad y escalabilidad, respectivamente.

VTK emplea el proceso de software de calidad de Kitware, que incluye CMake, CTest, CDash y CPack para construir, probar y empaquetar el sistema. Combinado con una fuerte comunidad de desarrolladores distribuidos, el resultado es un código robusto de muy alta calidad. La funcionalidad principal de VTK está escrita en C++ para maximizar la eficiencia. Esta funcionalidad está envuelta en otros vínculos de lenguaje para exponerla a una mayor audiencia. La interoperabilidad con Python está particularmente bien refinada.

Como software de código abierto, VTK es libre de usar para cualquier propósito. Técnicamente, VTK tiene una licencia de estilo BSD, que impone restricciones mínimas para las aplicaciones de código abierto y cerrado.

Se debería de tener en cuenta la herramienta ITK, es una plataforma open source que utiliza para mostrar sus resultados. ITK es una biblioteca de código abierto y multiplataforma que proporciona a los desarrolladores un amplio conjunto de herramientas de software para el análisis de imágenes. Desarrollado a través de metodologías de programación extrema, ITK se basa en una arquitectura probada y orientada al espacio para el procesamiento, la segmentación y el registro de imágenes científicas en dos, tres o más dimensiones. Los objetivos de ITK incluyen:

- Establecer una base para futuras investigaciones reproducibles.

- Crear un repositorio de algoritmos fundamentales.
- Desarrollar una plataforma para el desarrollo de productos avanzados.
- Apoyar la aplicación comercial de la tecnología.
- Crear convenciones para el trabajo futuro.
- Apoyar la educación en el análisis científico de imágenes.
- Hacer crecer una comunidad autosuficiente de usuarios y desarrolladores de software.

Utilizar VTK + ITK ofrece un gran y potente abanico de posibilidades. Para este proyecto, se investigó y se llegó a la conclusión de que lo ofrecido por VTK era suficiente para la realización del trabajo, pero es importante saber que, si en un futuro se desea ampliar la aplicación y hacerla más potente, será necesario el uso de ITK.

Ahora, vamos a repasar de forma teórica los pasos que da esta aplicación, desde la adquisición de la imagen hasta la manufacturación aditiva (impresión 3D). Previamente, debe observarse que precisión y repetibilidad general de los constructores médicos de manufacturación aditiva utilizados para la reconstrucción de huesos todavía necesita ser mejorada. En este contexto, una revisión sistemática reciente de Martelli y otros identificó 34 estudios (21,5%) que informaron sobre la precisión insatisfactoria de los constructores médicos de manufacturación aditiva [1].

El proceso médico de manufacturación aditiva usado para la reconstrucción del sistema músculo-esquelético puede ser dividido en cuatro pasos:

1. Adquisición de imagen médica (CT imaging).
2. Procesamiento de la imagen.
3. Diseño asistido por computador (CAD)
4. Manufacturación aditiva.

Cada uno de estos pasos puede introducir desviaciones geométricas que pueden causar distorsiones en las construcciones médicas resultantes. Sin embargo, estudios recientes sugieren que la mayoría de las inexactitudes se

introducen durante los pasos 1 y 2, más que durante la fabricación (paso 4), es decir, el proceso de impresión en 3D, que generalmente se considera preciso.

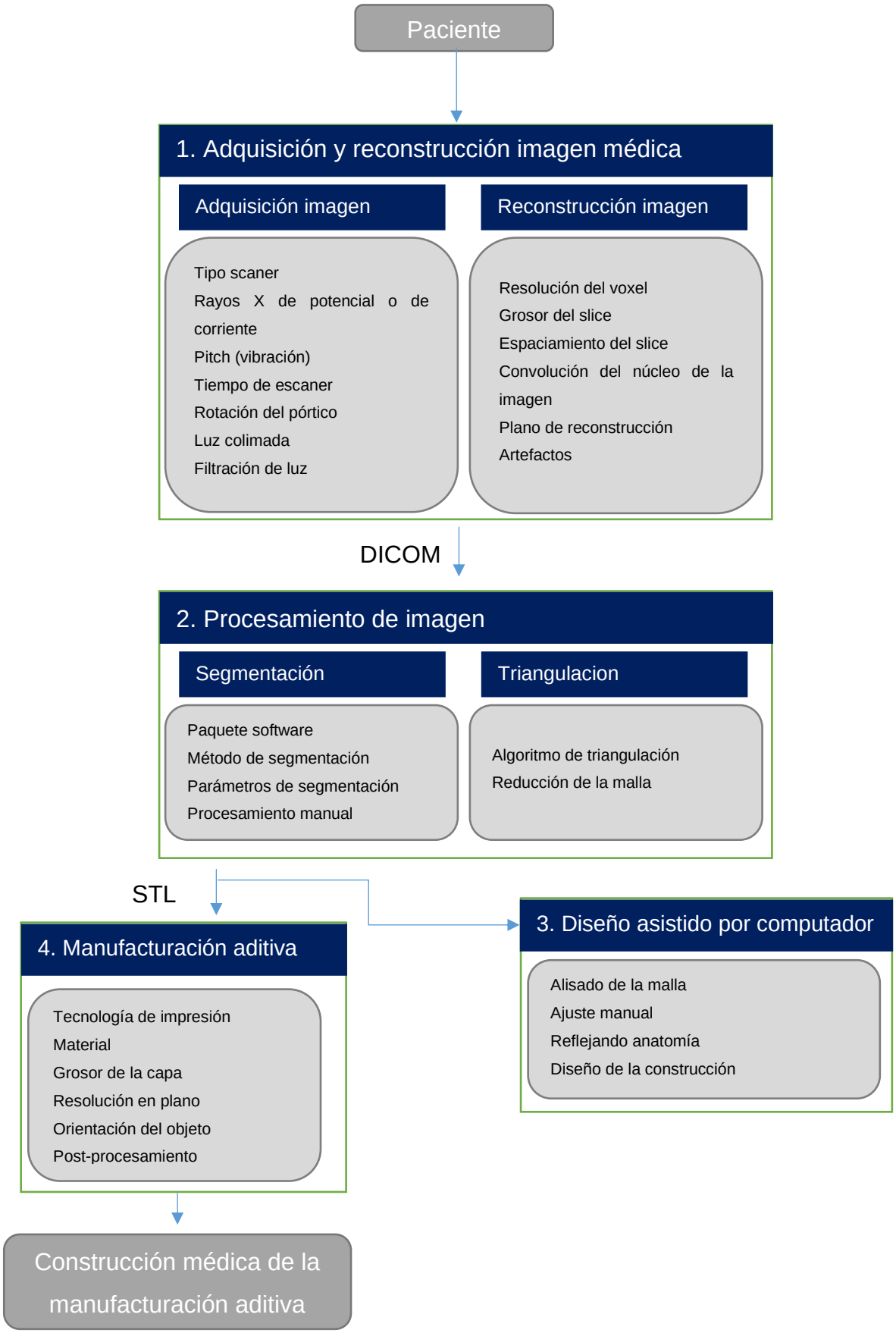


Ilustración 1.6 Análisis del sistema: Pasos del proceso médico de MA (traducción)

Se procederá a explicar más a fondo algunas de las bases teóricas como son la segmentación de imágenes, las modalidades de imagen y el estándar DICOM

1. Segmentación de imágenes.

   La segmentación se refiere a la división de las imágenes en regiones de interés (ROI) que corresponden a las estructuras anatómicas. En la actualidad, hay una gran cantidad de métodos diferentes de segmentación de imágenes disponibles para las estructuras óseas. Sin embargo, sigue sin estar claro qué método de segmentación ofrece los modelos de superficie 3D más precisos. Ahora, revisaremos los diferentes métodos de segmentación de imágenes por TAC que se utilizan actualmente para la segmentación ósea en la medicina Am y evaluaremos el impacto de los diferentes métodos de segmentación de imágenes en la precisión geométrica de los modelos de superficie en 3D.

   A. Basado en umbralización (thresholding)

      a. Gray-level Thresholding

      El trabajo de Beveridge et al. ofreció un ejemplo decente de un procedimiento que integra el umbral de nivel de gris, que es una técnica bajo la segmentación de imagen basada en el umbral [2]. En su trabajo, una imagen de entrada puede ser una imagen en escala de grises o en color. La imagen se divide entonces en sectores de tamaño y ubicación fijos. Se calcula un histograma de intensidad para cada sector (y en las imágenes en color, para cada canal de color), y se utiliza para producir una segmentación local. Para cada sector se utiliza la información de sus vecinos para detectar agrupaciones para las que puede no haber suficiente apoyo local debido a la partición artificialmente inducida de la imagen.

      1. Ventajas:

         Fácil de implementar y eficiente. No necesita información o cálculos previos.

      2. Desventajas:

Bordes con ruido y borrosos. Falta de sensibilidad y nitidez, difícil de definir, complejo para la multidimensionalidad.

b. Método Ostu

En 1979, la investigación de Ostu muestra que el método de umbral se basa en una idea muy simple, que consiste en calcular el umbral que minimiza la varianza ponderada dentro de la clase [2]. En el experimento, el método de Ostu dio como resultado un avance en la técnica basada en el umbral al convertir una imagen en escala de grises en una imagen binaria de forma automática.

1. Ventajas:

   Funciona bien y de forma estable.

   General: no se ha asumido ninguna forma específica de histograma. Capaz de extenderse a un umbral de varios niveles.

2. Desventajas:

   Pueden producirse falsos máximos durante la función de optimización. El método tiende a ampliar artificialmente las clases pequeñas para obtener una "mejor separación"; las clases pequeñas podrían fusionarse y perderse.

c. Método de mezcla gaussiana

El enfoque de mezcla gaussiana para la segmentación de imágenes es un método para estimar el número de componentes con sus medios y covarianza en forma secuencial sin requerir ninguna inicialización como lo proponen Nicole y Alexander en 2012 [2]. El procedimiento del experimento parte de un solo componente de la mezcla que abarca todo el conjunto de datos y se divide secuencialmente de manera incremental durante las etapas de maximización

de las expectativas. Este método de mezcla gaussiana muestra con éxito su eficacia después de muchos experimentos.

1. Ventajas:

   Un modelo de histograma relativamente general. Cuando el modelo es inválido, minimiza la probabilidad de error de clasificación. Aplicable a clases de tamaño pequeño.

2. Desventajas:

   Muchos histogramas no son gausianos. Las intensidades son finitas y no negativas. Difícil de detectar los modos cercano y plano. Requiere una simplificación significativa de la extensión de la forma del modelo a multithresholding.

B. Basado en región

   a. Región de crecimiento

   En 2001, Cootes y Taylor demostraron que el método de región de crecimiento puede lograrse añadiendo una compensación y escalando para transformar todos los valores grises en media cero y en varianza de unidades [2]. Como alternativa al trabajo con intensidades, Cootes y Taylor sugirieron un método que consiste en almacenar la dirección y la fuerza de los gradientes, esta última se traza de forma no lineal, este método demostró ser mejor que la intensidad normalizada. En 2002, Boasch et.al. describieron un procedimiento no lineal en su documento [2]. Este último método demostró ser especialmente útil en imágenes con una distribución fuertemente no gaussiana como la encontrada, por ejemplo, en las imágenes de ultrasonido. Los estudios continuados por Scott et al. en 2003, utilizaron la orientación del gradiente, la fuerza de las esquinas y los bordes para la

detección de vértebras en imágenes de rayos X de doble energía [2].

1. Ventajas:

   Funciona mejor en imágenes con ruido. Fácil de calcular.

2. Desventajas:

   El punto de partida debe ser especificado, es costoso y las imágenes pueden estar sub-segmentadas o sobre-segmentadas.

b. División y fusión de regiones

La técnica de división y fusión de regiones es bien conocida en el enfoque basado en la región. Esta técnica es una combinación de división y fusión de regiones. Manousakes et al. en 1998, aplicaron la técnica de división y fusión para tratar de superar las dificultades que se presentaban al utilizar las medidas de homogeneidad [2]. Este experimento verificó con éxito que los métodos de división y fusión pueden aplicarse en la IRM 2D y 3D con un rendimiento direccionalmente uniforme. Las mejoras que entrañan el recocido simulado y la eliminación de los límites también pueden aplicarse eficazmente en la resonancia magnética tridimensional o bidimensional. El proceso de ejecución requiere menos tiempo.

1. Ventajas:

   La imagen puede ser dividida progresivamente según la resolución demandada. Puede dividir la imagen por la media o la variación del valor de los píxeles del segmento. El criterio de fusión puede ser diferente al criterio de división.

2. Desventajas:

   Puede producir segmentos bloqueados.

C. Basado en bordes

a. Detección de bordes

En general, los métodos de detección de bordes son el proceso de identificación y localización de discontinuidades nítidas en una imagen. La detección de bordes es importante para el reconocimiento de los órganos humanos en las imágenes médicas. En los años 2006, Y.Q Zhang et al. introdujeron la teoría y operaciones morfológicas matemáticas básicas, la novedosa detección de bordes morfológicos matemáticos se propone para detectar el borde de la imagen de TC de los pulmones con sal y ruido de picos [2]. El resultado experimental muestra que el método propuesto es más eficiente tanto para la eliminación del ruido de la imagen médica como para la detección de bordes que el método de detección de bordes existente.

1. Ventajas:

   Delimita grandes áreas. Aplicable a imágenes con iluminación desigual.

2. Desventajas:

   Sólo aplicable al fondo simple. No se garantizan los contornos cerrados.

b. Detección de bordes de Prewitt

Según Prewitt y J.M., 1970, el detector de bordes de Prewitt es una forma apropiada de estimar la magnitud y la orientación de un borde [2]. Aunque la detección de bordes de gradientes diferenciales necesita un cálculo bastante largo para estimar la orientación a partir de las magnitudes en las direcciones x e y, la detección de bordes de brújula obtiene la orientación directamente del núcleo con la máxima respuesta. El operador de Prewitt se limita a ocho orientaciones posibles; sin embargo, los experimentos muestran que la mayoría de las

estimaciones de orientación directa no son tan precisas. Este detector de bordes basado en el gradiente se estima en la vecindad 3x3 para ocho direcciones. Las ocho máscaras de convección están calculadas. Luego se selecciona una máscara de convolución, la del módulo más grande. En este experimento, el detector de Prewitt es capaz de distinguir claramente los bordes.

1. Ventajas:

   Simple y capaz de detectar los bordes y sus orientaciones.

2. Desventajas:

   Sensible al ruido e inexactitud.

c. Laplaciana de Gauss

La Laplaciana de Gauss fue introducida por primera vez por Marr y Hildreth en 1980, quienes combinaron el filtrado Gauss con la técnica Laplaciana [2]. Este algoritmo no se utiliza a menudo en la visión artificial. Los investigadores que continuaron con este método fueron Berzins en 1984, Shah, Sood y Jain en 1986, Huertas y Medioni en 1986 [2].

1. Ventajas:

   Capaz de encontrar los lugares correctos de los bordes. Puede ser usado para probar un área más amplia alrededor del píxel.

2. Desventajas:

   Mal funcionamiento en las esquinas, curvas y donde la función de intensidad del nivel de gris varía. No se encuentra la orientación del borde debido al uso del filtro Laplaciano.

d. Watershed

Es un método de segmentación de imágenes introducido por S. Beucher y F. Meyer en 1990 que separa los objetos superpuestos [2]. Ellos propusieron usar la morfología matemática en la segmentación de imágenes. El objetivo del trabajo es evitar el problema de la sobre-segmentación. Este método involucra dos herramientas que son la transformación de la cuenca y la modificación de la homotopía (se utilizan en topología algebraica para clasificar los espacios topológicos).

1. Ventajas:

   Funciona mejor en imágenes con ruido. Rápida velocidad y resultados fiables.

2. Desventajas:

   El seed point debe ser especificado y fácilmente sobre-segmentado. Sólo se aplica en el gradiente y lleva mucho tiempo.

D. Basado en agrupación

   a. Método de agrupación del conjunto difuso (Fuzzy C-Mean)

   La teoría del conjunto difuso fue introducida por Zadeh, y aplicada con éxito en la segmentación de imágenes. El algoritmo difuso c-means fue propuesto por Bezdek en 1981 basado en la teoría difusa, es el algoritmo más estudiado y utilizado en la segmentación de imágenes por su simplicidad y la capacidad de obtener más información de las imágenes [2]. Los estudios fueron ampliados por muchos investigadores, uno de los últimos estudios de los investigadores fue realizado por Krinidis y Chatzis en 2010. Propusieron un c-medio de información local difusa (FLICM) para superar el problema de establecer parámetros en los métodos basados en el FCM [2]. Este algoritmo utiliza tanto la información local espacial como la de nivel gris, y está totalmente libre de ajuste de parámetros (excepto por el número de clusters).

1. Ventajas:

   Simple y fácil de entender.

2. Desventajas:

   Solución óptima indefinida, proceso de inicialización delicado y no compatible con datos ruidosos.

b. Agrupación K-Means

Según Kaus et al. en 2004, el método de agrupación de los K-Means es un proceso para clasificar un conjunto de datos determinado a través de un cierto número de agrupaciones K [2]. Después de agrupar todas las características en n clases con el algoritmo de la media k, cada punto de referencia se asigna a la agrupación que contiene el mayor número de características de formación a partir de ese punto. En la investigación, las características muestreadas en un determinado punto de referencia sólo necesitan ser comparadas con el grupo asignado. De este modo, se admiten automáticamente diferentes modelos de apariencia para diferentes segmentos límite.

1. Ventajas:

   El número rápido de agrupaciones es fijo, el algoritmo K-means es fácil de implementar y es más rápido que la agrupación jerárquica.

2. Desventajas:

   El usuario tiene que seleccionar el número de grupos de salida deseados antes de empezar a clasificar los datos. El resultado es sensible a la selección de los centroides aleatorios iniciales. No puede mostrar los detalles de la agrupación.

c. Agrupación jerárquica

Es uno de los métodos basados en los bordes desarrollados por D. Cordes en 2002 [2]. El autor había hecho una investigación utilizando la agrupación jerárquica para medir la conectividad en la resonancia magnética funcional. Este método es capaz de detectar similitudes de fluctuaciones de baja frecuencia, y los resultados indicaron que los patrones de conectividad funcional pueden obtenerse con la agrupación jerárquica que se asemeja a las conexiones neuronales conocidas.

1. Ventajas:

   Simple, salida fiable. Sólo hay que calcular las distancias entre cada patrón, en lugar de calcular el centroide de los clusters.

2. Desventajas:

   Los tiempos de cálculo son altos.

d. Desplazamiento medio

En el año 2002, Comaniciu y Meer describieron un método de segmentación basado en el algoritmo de desplazamiento medio [2]. El algoritmo de desplazamiento medio de Cheng en 1995 está diseñado para localizar puntos de máxima densidad local en el espacio de las características. Los vectores de los accidentes geográficos que contienen información de escala de grises o de color, así como las coordenadas de los píxeles, se calculan para cada píxel.

1. Ventajas:

   Una herramienta extremadamente versátil para el análisis de las características del espacio. Adecuado para espacios de características arbitrarias.

2. Desventajas:

El ancho de banda del núcleo es el único factor que puede controlar la salida. El tiempo de cálculo es bastante largo.

E. Otros métodos

a. Método de fijación de niveles

El método de fijación de niveles fue presentado por Osher y Sethian en 1988 y luego ampliado por Malladi et al. en 1995 [2]. Es ligeramente diferente a la investigación de Leventon y otros, por la que ampliaron la formulación energética original por un término adicional que deforma el contorno hacia una modelo de forma. Una crítica frecuente es que los mapas de distancia firmados que la forma en el que se basa el modelo, no forma un espacio lineal, lo que puede llevar a formas inválidas si las muestras de entrenamiento varían demasiado. En 2006, Pohl et al. presentaron un método de incrustando los mapas de distancia firmados en el espacio lineal de Log Odds, que podría resolver los problemas de modelado [2]. Para mantener esta revisión en una extensión razonable, tenían que ignorar la teoría y las técnicas de fijación de niveles: Las diferencias conceptuales entre la representación implícita y los modelos discretos en los que pretenden centrarse tendrían requería un tratamiento especial para todas las secciones siguientes.

1. Ventajas:

   Versátil, robusto, preciso y eficiente.

2. Desventajas:

   Requiere una considerable reflexión para construir las velocidades apropiadas para avanzar en la función del conjunto de niveles.

b. Red neuronal artificial

En 2003, Pal and Pal realizó una investigación sobre el método de las RNA utilizado para la segmentación [2]. Como Pal and Pal predijo, las RNA se aplican ampliamente en el procesamiento de imágenes. Últimamente la investigación de este método ha sido continuada por Indira, S.U., y Ramesh en los años de 2011. Los métodos 1074 L.K. Lee et al. Kohonen's Self-Organizing Maps (SOM) form ANN y (Genetic Algorithm) GA fueron combinados para identificar los principales rasgos presentes en la imagen [2]. SOM combinado con GA y algunas de las variantes de SOM como el SOM de estructura variable (VSSOM), SOM sin parámetros (PLSOM) se comparan y se evalúa su rendimiento. Se desarrolla un nuevo método no paramétrico no supervisado combinando las ventajas de VSSOM y PLSOM. Los experimentos realizados en la imagen de satélite muestran que el PLSOM modificado es eficiente y el tiempo de segmentación es menor en comparación con los otros métodos.

1. Ventajas:

   Aplicable a gran variedad de problemas. Fácil de implementar.

2. Desventajas:

   Falta de una base teórica profunda para las RNA. Problema en la elección de la mejor arquitectura. Problema de la black-box.

Según las modalidades revisadas, los métodos recomendados se muestran en la siguiente tabla [2].

Modalidades	Método recomendado
TAC	Umbralización y basado en región Región de crecimiento y watershed
Resonancia magnética tridimensional	Método de fijación de niveles Red neuronal artificial
Resonancia magnética	Basado en agrupación
Ultrasonido	Basado en umbralización
Rayos X	Basado en bordes y watershed

Tabla 1.6 Análisis del sistema: Métodos recomendados

Cada método de segmentación de imágenes tiene sus propias limitaciones. Por lo tanto, lo explicado puede utilizarse como referencia. La precisión de la segmentación sigue siendo el tema más preocupante a la hora de determinar casos críticos como la detección de tumores a través de imágenes médicas [3]. Las recomendaciones para futuros trabajos incluyen la mejora de la precisión y la velocidad de la segmentación de imágenes para la marca de agua digital en las imágenes médicas.

Aún tenemos que hablar sobre las últimas bases teóricas, en concreto sobre posible estado de los pacientes y el estándar DICOM.

A la hora de realizar la segmentación, hay que tener en cuenta si los huesos están sanos o tienen algún tipo de enfermedad que dificulte la realización de dicho proceso. La osteoporosis disminuye la densidad ósea, esto dificulta realizar una segmentación basada en el nivel de gris. Otras patologías como la osteoartritis o la artritis reumatoide conlleva un desgaste de los huesos en las zonas de las articulaciones, el desgaste afecta al hueso compacto que se encuentra en el exterior del hueso, lo que hace que no exista un borde pronunciado del hueso en la imagen.

2. Modalidades de imagen [3].

Tomografía Axial Computarizada (TAC) o CT es una técnica de imagen médica que utiliza la radiación X para obtener cortes o secciones de objetos anatómicos con el fin de obtener un diagnóstico. El TAC nos permite crear una imagen transversal del cuerpo humano.

En un dispositivo de imágenes TAC podemos diferenciar tres elementos principales: el tubo de rayos X, el computador encargado de realizar la reconstrucción, y los detectores encargados de transformar la radiación recibida en señales digitales que el computador pueda procesar.

El tubo emisor de rayos X gira alrededor del paciente. En el lado opuesto se encuentran los detectores que forman un perfil de la radiación que logra atravesar el cuerpo. La radiación que llega a los detectores depende de las densidades de los órganos atravesados, a mayor densidad mayor atenuación.

3. Estándar DICOM [3].

El estándar DICOM es el estándar reconocido mundialmente para el intercambio de pruebas médicas, pensado para su manejo, visualización, almacenamiento, impresión y transmisión. La cabecera de este formato, permite almacenar información sobre el paciente, las condiciones en las que se tomó la imagen y el formato interno de ésta, lo que permite que no se desvincule la imagen de la información.

DICOM permite la integración de escáneres, servidores, estaciones de trabajo, impresoras, y hardware de red de múltiples proveedores dentro de un sistema de almacenamiento y comunicación de imágenes, por lo que el estándar no es sólo un formato de fichero para imágenes médicas sino que pretende ser un estándar completo que cubra todas las necesidades de un PACS: almacenamiento, pero también transmisión, comunicaciones en general e impresión. Para lograr esto el estándar especifica:

- Conjunto de protocolos para las comunicaciones de red que deben ser seguidos por los dispositivos conformes al estándar.
- Sintaxis y semántica de los comandos y la información asociada que puede ser intercambiada usando estos protocolos.

- En cuanto a la comunicación de datos, se define un conjunto de servicios para el almacenamiento de datos que deben ser seguidos por los dispositivos conformes al estándar, así como un formato de fichero y una estructura de directorio médico para facilitar el acceso a las imágenes y a la información relacionada.

El formato de fichero DICOM es muy complejo, ya que en la cabecera se especifican una gran cantidad de campos. Además, existen varios tipos de cabecera y la imagen se puede grabar en multitud de formatos distintos.

El fichero DICOM se puede dividir en cuatro partes diferenciadas:

- Preámbulo y prefijo identificativo del fichero
- Meta-cabecera.
- Cabecera.
- Imagen (aunque desde el punto de vista del formato, la imagen es un elemento más de la cabecera).

El preámbulo tiene un tamaño fijo de 128 bytes, y está pensado para tener un uso definido por la implementación. El preámbulo puede contener información sobre el nombre de la aplicación usada para crear el fichero, por ejemplo. En cuanto al prefijo consiste en cuatro bytes que contienen la cadena de caracteres DICOM. Esta cadena debe estar codificada siempre con las letras en mayúscula y usando el repertorio de caracteres ISO 8859 G0. El propósito de este prefijo es permitir a las implementaciones diferenciar si un fichero es DICOM o no.

En cuanto a la cabecera y la meta-cabecera de un fichero DICOM, consisten en una serie de campos con toda la información necesaria sobre la imagen en cuestión, incluyendo la propia imagen.

Entre estos campos podemos encontrar datos sobre el paciente (nombre, sexo), y sobre el tipo de imagen entre otros. Al conjunto de toda la información codificada sobre un campo se le conoce con el nombre de Elemento de Datos (Data Element).

En un elemento de datos podemos diferenciar los siguientes campos:

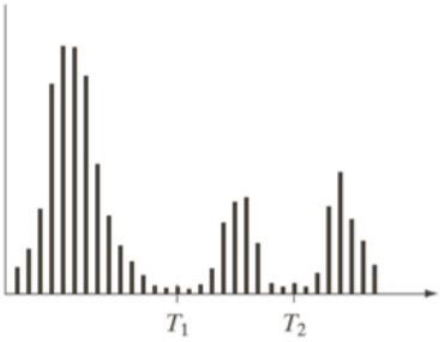
- Etiqueta del Elemento de Datos (Data Element Tag): sirve para identificar cada elemento de datos de forma unívoca. Esta etiqueta está constituida por un número de grupo y un número de elemento.
- Representación del Valor (Value Representation, abreviado VR): indica la forma en que se codifica el valor del elemento. Por ejemplo, el valor puede estar codificado como una cadena de caracteres o un entero sin signo.
- Longitud del Valor (Value Length): como su nombre indica, es la longitud del campo valor.
- Valor (Value): es el valor del elemento de datos, codificado según el campo VR y con la longitud que indica el campo longitud del valor.

Teniendo estas bases teóricas, puedo proceder a especificar cuál será la forma y los métodos con los que trabajaremos. Nuestra aplicación recibirá un archivo con el formato DICOM, los ejemplos que tenemos son tomografías computarizadas. La aplicación leerá y aplicará a las imágenes o slices un método de segmentación, en concreto, el de umbralización (recordemos que, para los TAC, este método es el más recomendable). A las imágenes segmentadas, se les aplicará un algoritmo de triangulación, específicamente el algoritmo de marching cubes [5], el más utilizado en el área de visualización médica. De esta forma obtendremos nuestro modelo 3D a partir de un TAC, el cual se guardará automáticamente en un archivo STL. El archivo quedará a disposición del usuario para que pueda realizar un proceso de limpieza del modelo para impresión 3D, la herramienta que se utilizarán en las pruebas será MeshMixer.

De esta forma, con estas herramientas y con las bases teóricas aprendidas, el usuario podrá manejar la interfaz diseñada. Podrá especificar cuál será el archivo DICOM que quiere que se procese, especificar el nivel de detalle del modelo, ajustar los niveles de umbralización, especificar el nombre del archivo STL que se guardará automáticamente y especificar los colores que desee que aparezcan en la escena del modelo 3D.

Lo más difícil de ajustar para el usuario serían los niveles de umbralización, el umbral mínimo y umbral máximo. El umbral mínimo hará que todos los niveles

de grises menores al umbral especificado se conviertan en negro y el umbral máximo hará que todos los niveles de grises mayores al umbral especificado se conviertan en blanco. Este método se utiliza para diferenciar un objeto del fondo de la imagen llamado binarización (negro y blanco). Utiliza dos umbrales debido a que en un TAC nos encontramos con dos objetos claros sobre un fondo oscuro.



$$g(x,y) = \begin{cases} a & \text{si } f(x,y) > T_2 \\ b & \text{si } T_1 < f(x,y) \leq T_2 \\ c & \text{si } f(x,y) \leq T_1 \end{cases}$$

Ilustración 1.6 Análisis del sistema: Umbralización

Una vez ajustado, el usuario pasará a interactuar con las imágenes ya segmentadas, desplazándose entre los distintos slices y haciendo zoom. Por último, podrá interactuar con el modelo 3D generando, rotando sobre dicho objeto y haciendo zoom.

Como feedback, si hay algún fallo en los parámetros o en los archivos DICOM se notificará al usuario mediante una ventana emergente.

1.6.2 - Diagramas de casos de uso

Caso de uso	Configurar parámetros
Actor Primario	Usuario
Sistema	Aplicación
Participantes	Usuario
Nivel	Objetivo Usuario
Condición previa	-
Operaciones básicas	
1	Configurar parámetros
2	Generar imágenes
3	Generar modelo
4	Guardar modelo
Alternativas	
1.A	¿Los parámetros son correctos?
1.A.1	Caso afirmativo, continuar.
1.A.2	Caso negativo, mostrar error y volver a paso 1

Tabla 1.6.2 Casos de uso: Configurar parámetros

Caso de uso	Visualizar imágenes
Actor Primario	Usuario
Sistema	Aplicación
Participantes	Usuario
Nivel	Objetivo Usuario
Condición previa	Configurar parámetros correctamente
Operaciones básicas	
1	Visualizar imágenes
2	Pasar imágenes
3	Incrementar/Decrementar zoom imágenes

Tabla 1.6.2 Casos de uso: Visualizar imágenes

Caso de uso	Visualizar modelo
Actor Primario	Usuario
Sistema	Aplicación
Participantes	Usuario
Nivel	Objetivo Usuario
Condición previa	Cerrar visualización imágenes
Operaciones básicas	
1	Visualizar modelo
2	Rotar cámara
3	Incrementar/Decrementar zoom

Tabla 1.6.2 Casos de uso: Visualizar modelo

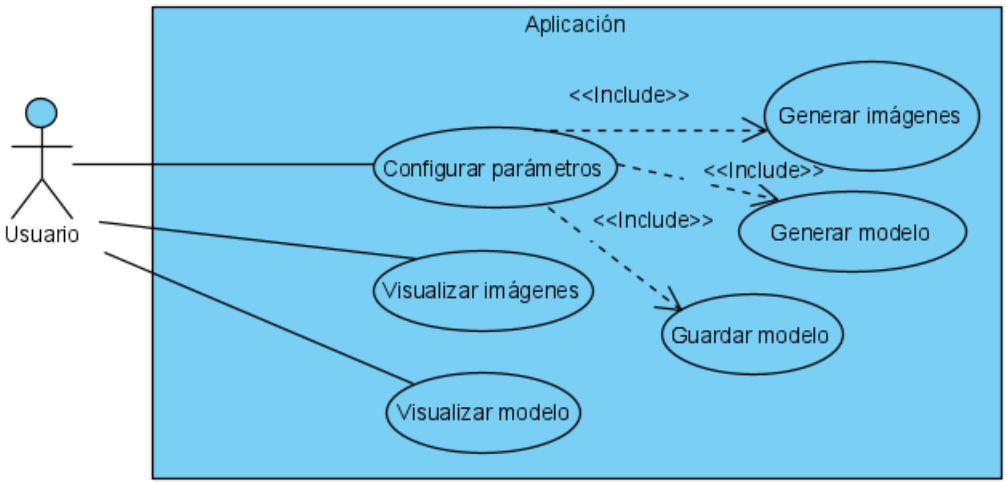


Ilustración 1.6.2 Casos de uso

1.6.3 - Diagramas de secuencia de sistema

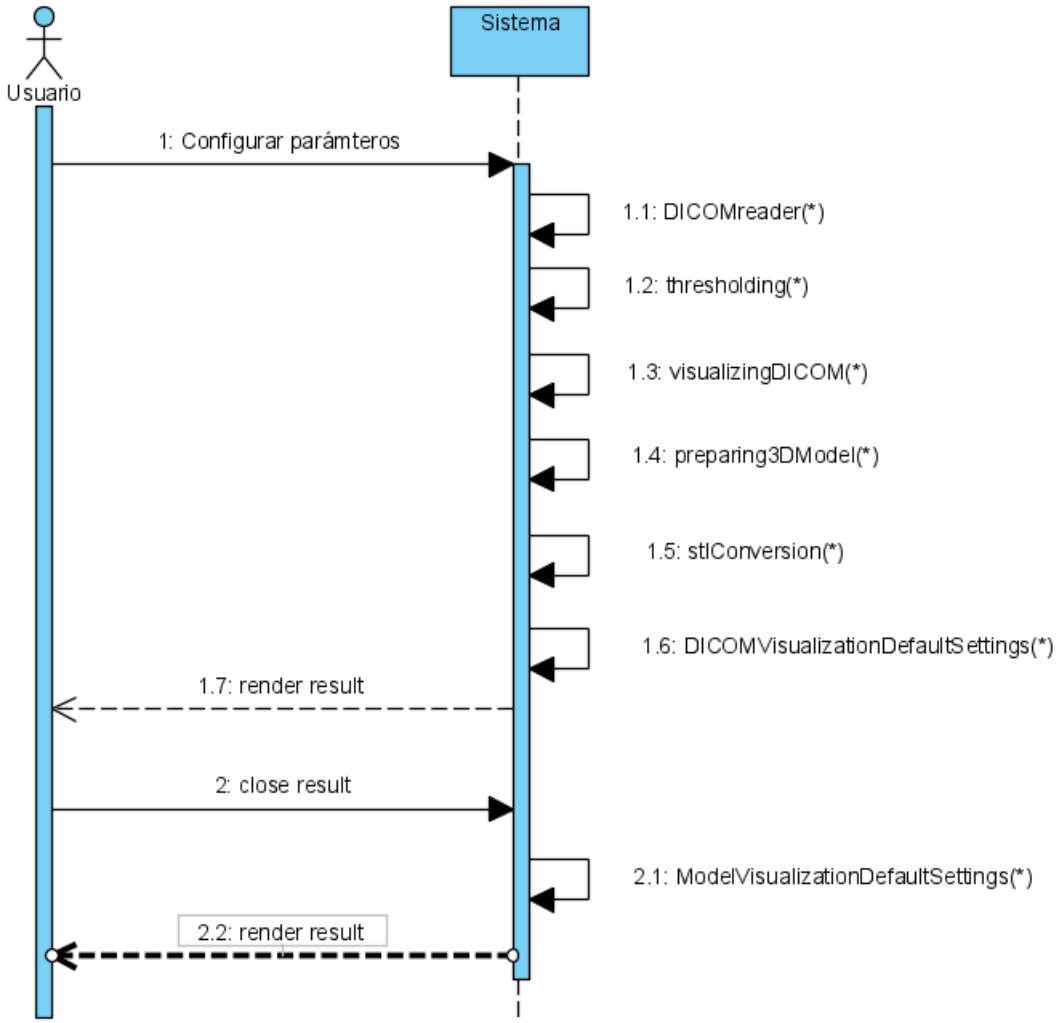


Ilustración 1.6.3 Diagramas de secuencia de sistema: Aplicación

Los parámetros de las funciones son los siguientes:

```
void DICOMReader(vtkSmartPointer<vtkDICOMImageReader> reader, std::string folder){ ... }

void thresholding(vtkSmartPointer<vtkDICOMImageReader> reader, vtkSmartPointer<vtkImageThreshold> imageThreshold,
                  double lower, double upper){ ... }

void visualizingDICOM(vtkSmartPointer<vtkImageThreshold> imageThreshold, vtkSmartPointer<vtkImageViewer2> imageViewer,
                      vtkSmartPointer<vtkTextProperty> sliceTextProp, vtkSmartPointer<vtkTextMapper> sliceTextMapper,
                      vtkSmartPointer<vtkActor2D> sliceTextActor, vtkSmartPointer<vtkTextProperty> usageTextProp,
                      vtkSmartPointer<vtkTextMapper> usageTextMapper, vtkSmartPointer<vtkActor2D> usageTextActor,
                      vtkSmartPointer<vtkRenderWindowInteractor> renderWindowInteractor,
                      vtkSmartPointer<myVtkInteractorStyleImage> myInteractorStyle){ ... }

void preparing3DModel(vtkSmartPointer<vtkImageThreshold> imageThreshold, double detailLevel, vtkSmartPointer<vtkNamedColors> colors,
                      std::array<unsigned char, 4> skinColor, std::array<unsigned char, 4> bkg,
                      vtkSmartPointer<vtkRenderer> aRenderer, vtkSmartPointer<vtkRenderWindow> renWin, vtkSmartPointer<vtkRenderWindowInteractor> iren,
                      vtkSmartPointer<vtkMarchingCubes> skinExtractor, vtkSmartPointer<vtkPolyDataMapper> skinMapper, vtkSmartPointer<vtkActor> skin,
                      vtkSmartPointer<vtkOutlineFilter> outlineData, vtkSmartPointer<vtkPolyDataMapper> mapOutline, vtkSmartPointer<vtkActor> outline){

void stlConversion(vtkSmartPointer<vtkMarchingCubes> skinExtractor, vtkSmartPointer<vtkSTLWriter> stlWriter, const char* modelName){ ... }
```

```
void DICOMVisualizationDefaultSettings(vtkSmartPointer<vtkImageViewer2> imageViewer, vtkSmartPointer<vtkRenderWindowInteractor> renderWindowInteractor){

void ModelVisualizationDefaultSettings(vtkSmartPointer<vtkCamera> aCamera, vtkSmartPointer<vtkRenderer> aRenderer,
                                       vtkSmartPointer<vtkActor> outline, vtkSmartPointer<vtkActor> skin, vtkSmartPointer<vtkNamedColors> colors,
                                       vtkSmartPointer<vtkRenderWindow> renWin, vtkSmartPointer<vtkRenderWindowInteractor> iren){ ... }
```

Ilustración 1.6.3 Diagramas de secuencia de sistema: Parámetros

1.6.4 - Diseño de la interfaz y storyboards

Durante el desarrollo de esta aplicación, no se ha podido implementar una interfaz por problemas de compatibilidad entre las distintas herramientas, pero si se ha realizado un storyboard, teniendo en cuenta las distintas necesidades que puede llegar a tener el usuario en el uso de la aplicación, sobre todo, el cambio de valores entre distintos parámetros.

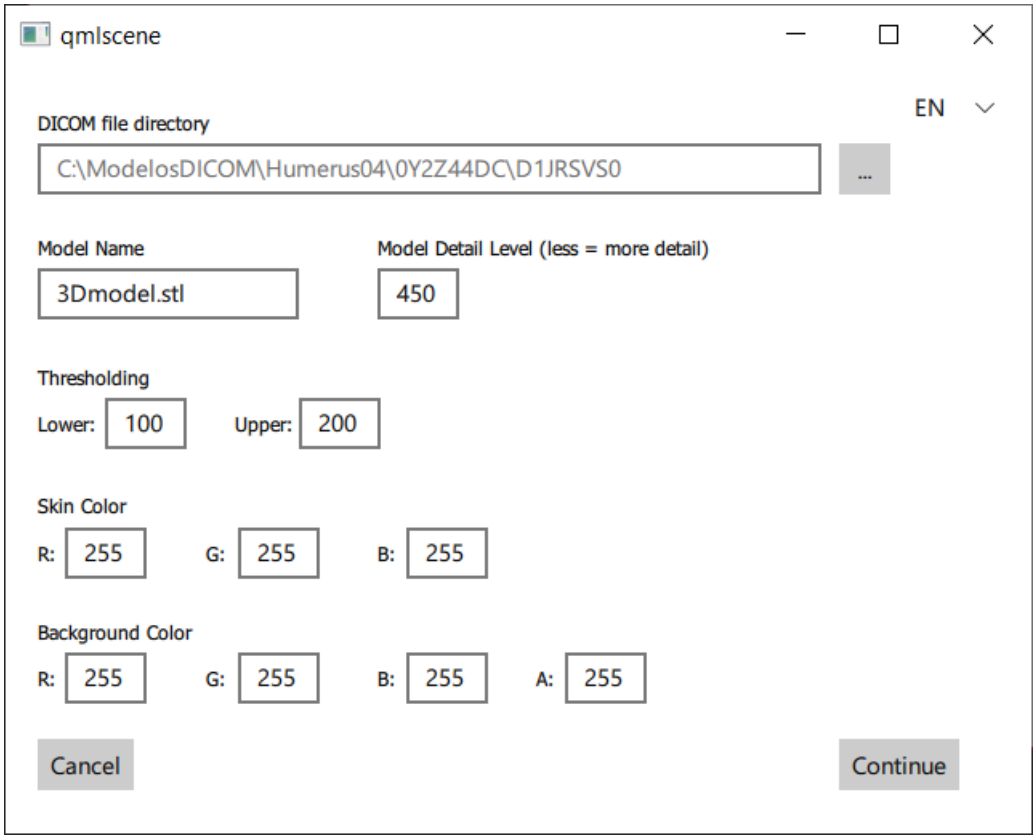


Ilustración 1.6.4 Storyboards: Configuración de parámetros

El método que se usará para configurar parámetros será por línea de comandos.

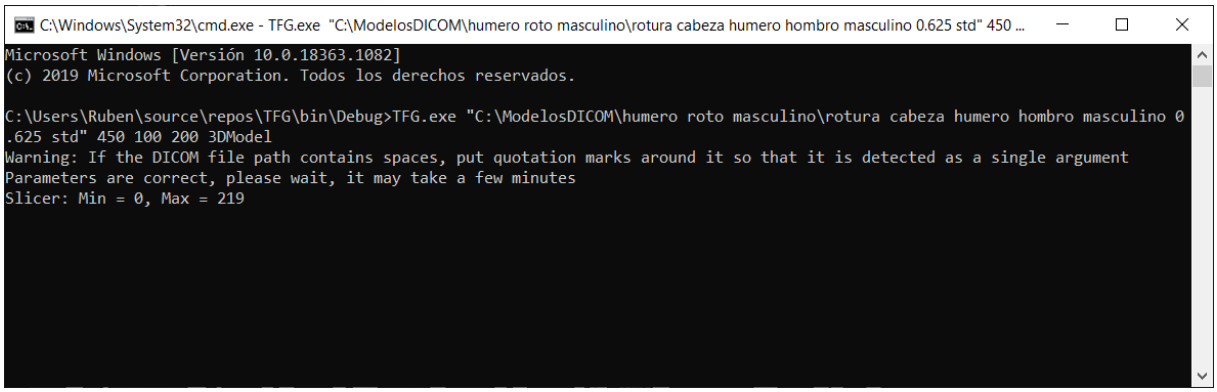


Ilustración 1.6.4 Diseño de la interfaz: Configuración de parámetros

La ventana de visualización de archivos DICOM se ha diseñado para proporcionar feedback al usuario, mostrando un pequeño diálogo de uso y el estado actual de la visualización (el número del slice que se está mostrando).

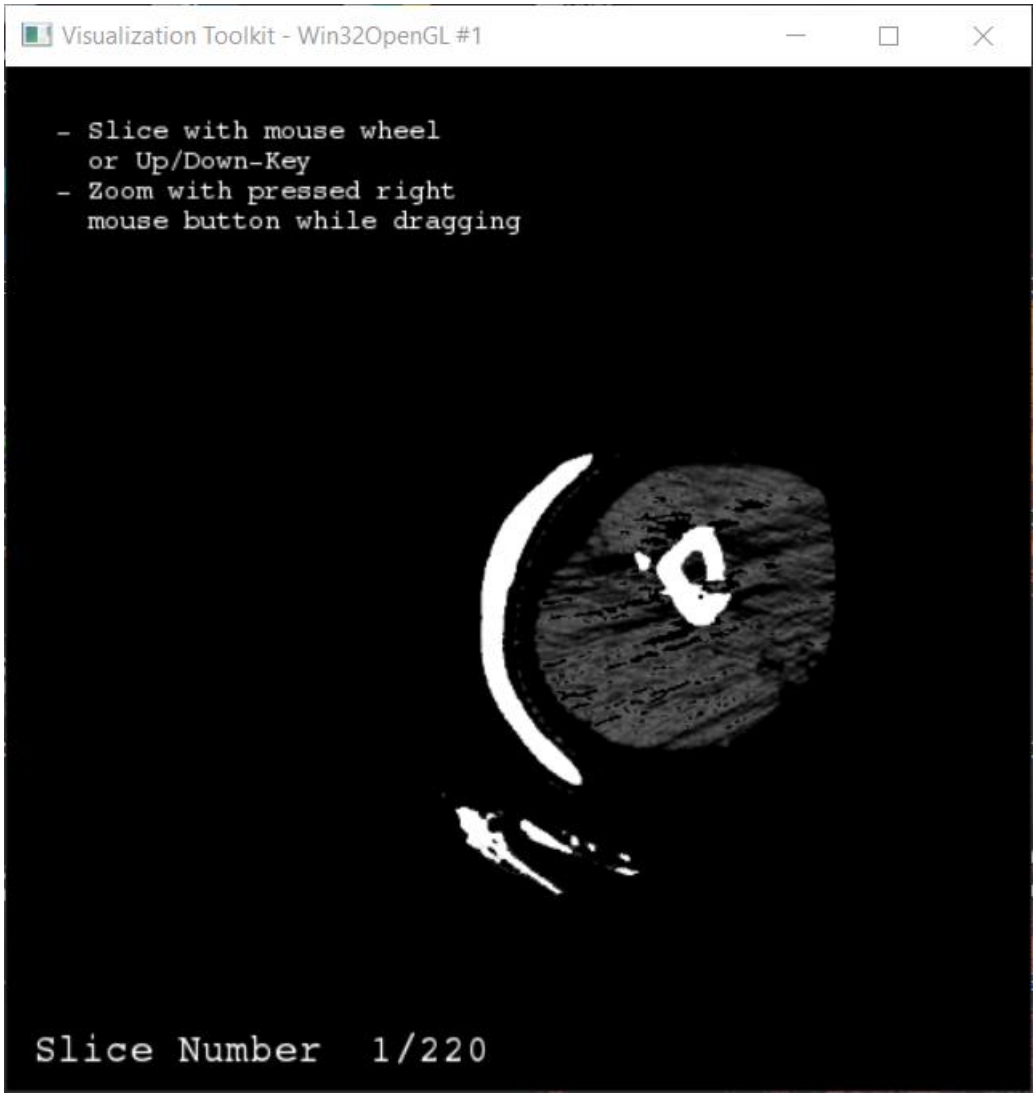


Ilustración 1.6.4 Diseño de la interfaz: Visualización de slices

La ventana de visualización del modelo es más limpia, para así poder visualizar el hueso con total plenitud.

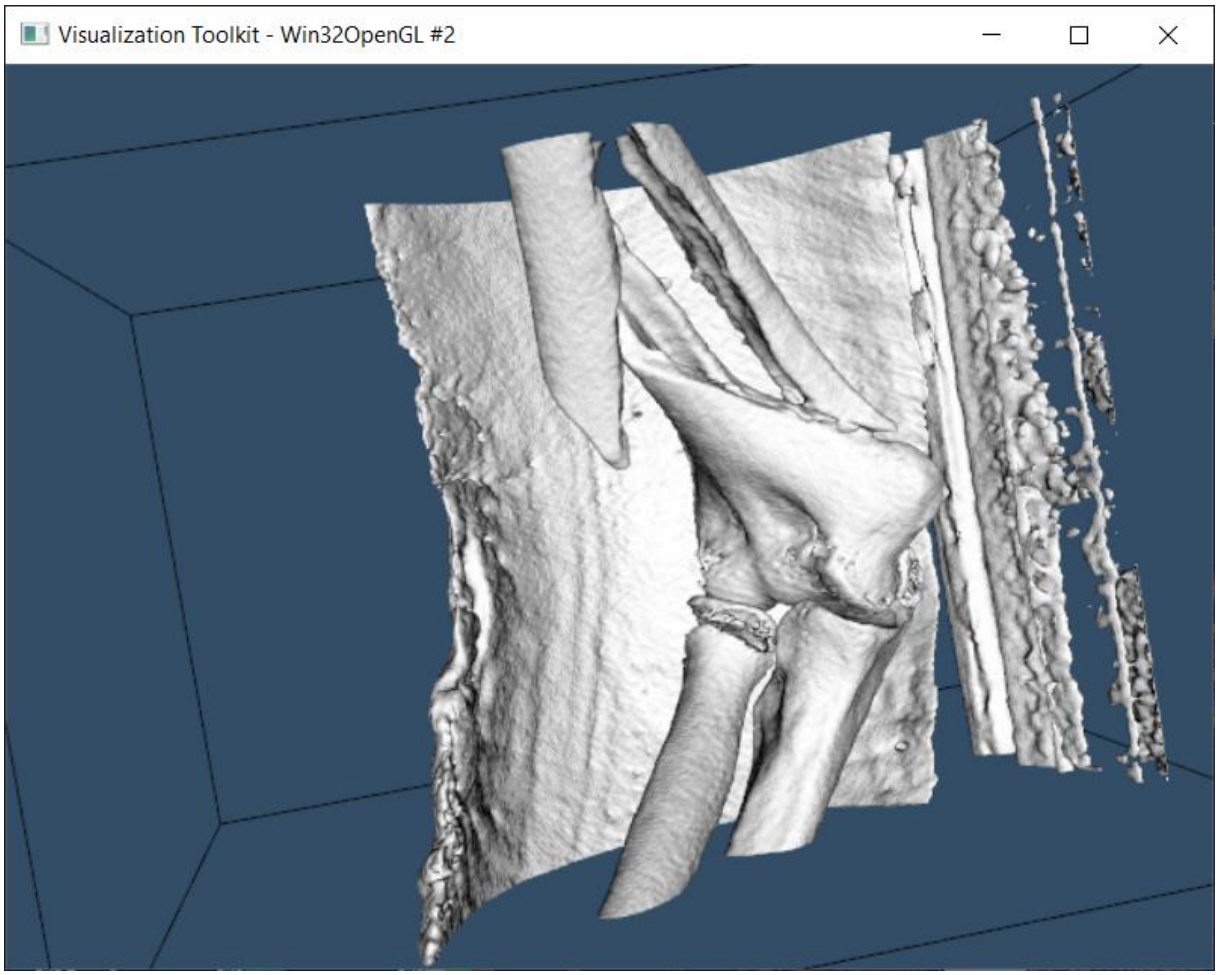


Ilustración 1.6.4 Diseño de la interfaz: Visualización de modelo

Por último, existe una ventana que proporciona feedback al usuario en el caso de que aparezca algún error en los distintos procesos de la aplicación.

Ilustración 1.6.4 Diseño de la interfaz: Ventana de errores

1.6.5 - Diseño del sistema

Ya realizado el análisis del sistema, procederé a explicar el diseño de las clases que lo formarán, todas ellas procedentes de la herramienta VTK.

Para que el usuario pueda interactuar y tener algo de feedback a la hora de manejarse en la visualización de los slices, se han creado dos clases en el mismo archivo de ejecución, ya que, los estilos de interacción en la visualización de varias imágenes no los proporciona VTK, pero son programables. Estas clases son “StatusMessage”, para indicar el estado actual de la escena (el slice que se está visualizando en le momento) y “myVtkInteractorStyleImage” para programar la forma con la que el usuario podrá interactuar.

Las variables que se han parametrizado para el usuario son “folder”, indicará la ruta del archivo DICOM, “detailLevel”, indicará el detalle del modelo 3D, “lower y “upper”, indicarán el umbral mínimo y máximo de la umbralización y por último “modelName”, que indicará el nombre del modelo que se generará automáticamente. El nivel de detalle solo podrá ser entre 200 y 500, menos, es más, es decir, cuanto menor sea el valor, mayor será el detalle. El umbral mínimo y máximo podrán estar entre 0 y 255 (esto es debido a que en las imágenes con escala de grises, cada píxel puede tener un valor desde la intensidad más clara, blanco, a la más oscura, negro, en las imágenes con profundidad de 8 bits, estos valores son del 0 al 255), el umbral mínimo siempre tendrá que ser menor al máximo y el máximo siempre tendrá que ser mayor al mínimo. Todo esto será ejecutado mediante línea de comandos debido a la imposibilidad de la instalación de Qt para realizar una interfaz para los parámetros.

Para la lectura del fichero DICOM, se utilizará la clase “vtkDICOMImageReader”, esta solo necesita de la ruta donde se encuentre el fichero para cargar las imágenes.

Para realizar el proceso de umbralización, se utilizará la clase “vtkImageThreshold”, esta necesita de las imágenes obtenidas por

"vtkDICOMImageReader" y de los valores de los umbrales mínimo y máximo aportados por el usuario.

El proceso de creación de la escena de visualización de los slices ya empieza a ser algo más complejo. Este proceso requiere de un visor de imágenes proporcionado por la clase "vtkImageViewer2". Para dar feedback al usuario mediante mensajes, tanto de consejos de uso como de estado actual, se utilizarán las clases "vtkTextProperty", "vtkTextMapper" y "vtkActor2D". Por último, necesitará de las clases que ayudarán a interactuar con la escena, estas clases son "vtkRenderWindowInteractor" y la clase que hemos creado manualmente y explicado anteriormente, "myVtkInteractorStyleImage", que especifica el tipo de interacción. El visor de imágenes "vtkImageViewer2" se conectará con la clase que realiza el proceso de umbralización, ya que, es la encargada de mostrar las imágenes. Seguidamente, las clases que se encargan de mostrar los mensajes de consejo de uso y estado actual, son parametrizadas con los valores deseados, es decir, posición, tamaño de fuente, justificación, mensaje deseado… Por último, a "myInteractorStyle" se le indica que "imageViewer2" va funcionar como un mapeado de texto para hacer visible el estado actual de la imagen que se está procesando y también se indica qué objeto de la clase de "vtkRenderWindowInteractor" va a ser el que interactúe con el objeto de "imageViewer2". De esta forma, ya podemos decirle a la ventana de renderizado cuál va a ser nuestro estilo de interacción y ya podemos renderizar los mensajes de texto.

Con esto, ya tenemos configurado lo necesario para ser capaces de mostrar los slices segmentados de forma satisfactoria. El siguiente paso es la preparación de la siguiente escena, la que nos muestro nuestro modelo 3D ya creado, otro proceso bastante complejo.

Primero, crearemos los objetos que necesitaremos para indicar los colores de la escena, es decir, los colores del modelo y del fondo. Para ello, se utilizará la clase "vtkNamedColors" y dos vectores de tamaño cuatro que representan los valores RGBA, un vector para el modelo y otro para el fondo.

Seguidamente, necesitaremos un renderizador para mostrar el modelo, la ventana para mostrar el render y el que se encarga de las diferentes

interacciones, estas clases son “vtkRenderer”, “vtkRendererWindow” y “vtkRedererWindowInteractor”.

A continuación, procederemos a crear lo necesario para generar el modelo 3D, para ello, necesitaremos la clase que utilizará el algoritmo de Marching Cubes, este se encarga de realizar la triangulación a través de la información que ya tenemos, los slices. También necesitaremos el mapeador de los datos del polígono y un actor que nos ayude a unirlo todo. Las clases son “vtkMarchingCubes”, “vtkPolyDataMapper” y “vtkActor”.

Por último, necesitaremos un esquema que proporcione contexto en torno a los datos, para ello necesitaremos un filtro, un mapeador y un actor. Las clases serían “vtkOutlineFilter” y las otras dos que ya hemos utilizado, “vtkPolyDataMapper” y “vtkActor”.

Con esto ya se podrá proceder, en principio, se indicará al objeto que contiene los colores (“vtkNamedColors”), los que deseamos, con ayuda de los dos vectores en formato RGBA ya creados. Seguidamente, se indicará a nuestra ventana de renderizado cuál será el render que mostrará y a nuestro objeto que define las interacciones, con qué ventana está vinculada.

A continuación, se generará el modelo a partir de las imágenes segmentadas, para ello, el objeto de la clase “vtkMarchingCubes” es conectado con el objeto que las imágenes segmentadas, el objeto ya actúa automáticamente y genera el modelo con el algoritomo. El objeto que se encarga de mapear es conectado con el que genera el modelo, ya que, este contiene la información necesaria para realizar el mapeado. Por último, para este proceso, se indica al actor cuál es el mapeador y se le indica el color deseado.

Para terminar, el último proceso que necesitamos es el que crea el esquema que proporciona contesto en torno a los datos. Al objeto de tipo “vtkOutlineFilter” se le proporciona la información de las imágenes segmentadas, al mapeador se le proporciona la información que contiene el filtro creado anteriormente y al actor se le proporciona el mapeador y el color deseado.

El siguiente proceso sería el de guardar el modelo generado en un archivo STL. Para ello solo necesitaremos un objeto de la clase “vtkSTLWriter”, al que le

proporcionaremos el nombre del archivo indicado por el usuario y el modelo generado por objeto que realizaba el algoritmo de Marching Cubes.

Con todo esto, ya solo quedaría inicializar nuestras escenas, en primer lugar, la que muestra los slices y en segundo lugar, la que muestra el modelo. Para esta última, lo único que nos queda es crear una cámara, este objeto sería de clase “vtkCamera”. Una vez configurada, para iniciar las escenas, se le indican a los renderer, cuáles son los actores, las resoluciones de las ventanas y se inician.

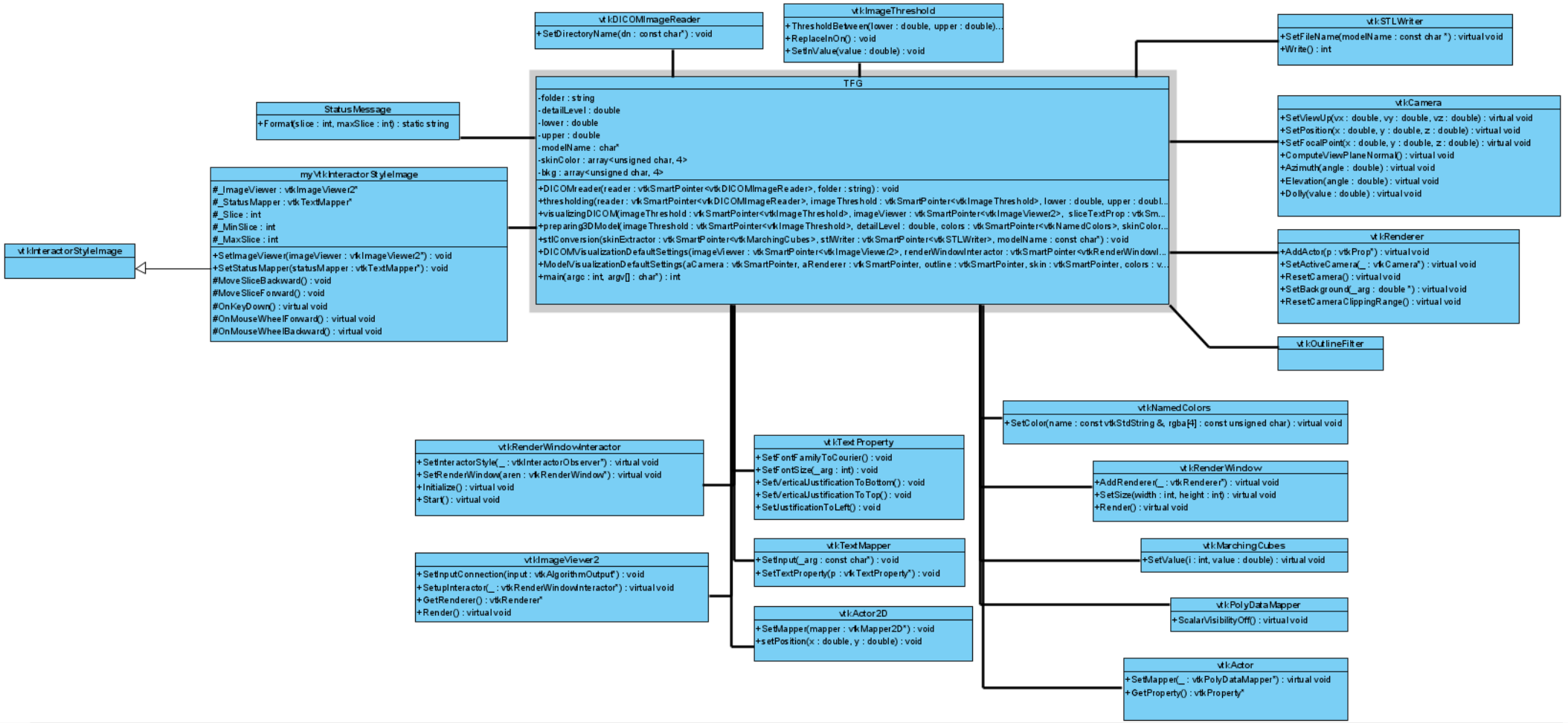


Ilustración 1.6.5 Desarrollo del sistema: UML

1.6.6 - Metodología de desarrollo

Se ha usado una metodología incremental, donde cada incremento ha pasado por diferentes etapas, análisis, diseño, implementación y pruebas.

Se ha decidido utilizar esta metodología debido a la gran investigación previa realizada, por ello, se ha podido tener los requisitos definidos desde un primer momento, consiguiendo de forma progresiva los objetivos de la aplicación

1.7 - A3: Estimación del tamaño y esfuerzo

Ya que el presente proyecto es un TFT, no existen restricciones de tipo económico, sino de tipo temporal (un número aproximado de horas). Por consiguiente, los cálculos de tamaño del proyecto están supeditados el tiempo disponible. En cuanto al esfuerzo, se dispone de tan un solo efectivo (el autor).

2 - DESARROLLO DEL PROYECTO

2.1 - Diseño

2.1.1 - Diseño arquitectónico del sistema

Para este TFG, es obligatorio el uso de VTK, ya que, se buscaba su investigación y aprendizaje en el ámbito de la medicina concretamente.

Este software es de código abierto y de libre acceso para procesamiento de imágenes, modelado, gráficos en 3D, representación de volúmenes, visualización científica y trazado en 2D. En resumen, es un software que nos administra todo lo necesario para la realización del proyecto. Solo este hace posible la lectura de archivos DICOM, segmentación de imágenes, modelado, representaciones gráficas tanto en 2D como en 3D y el guardado de modelos en archivos de diferentes formatos.

Como generador de sistema de compilación, es necesario el uso de CMake. Esta herramienta es capaz de generar nuestro proyecto con los módulos de VTK que se han necesitado para el desarrollo de la aplicación. CMake cuenta con soporte oficial para VTK, debido a ello, es la mejor herramienta para este cometido.

2.1.2 - Diagramas de clases

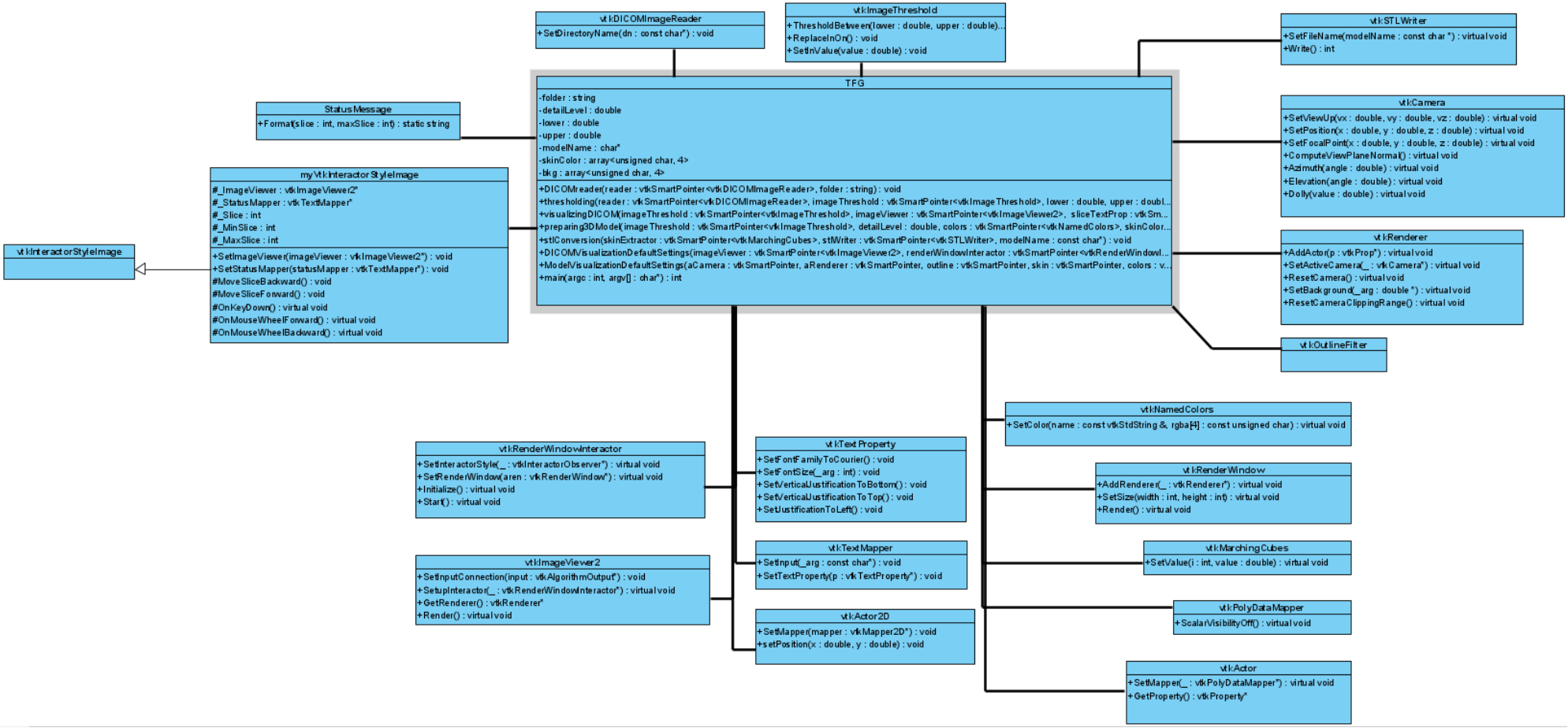


Ilustración 2.1.2 Diagramas de clase: UML

Debido al uso de numerosos parámetros en algunas funciones de la clase principal “TFG”, no se puede ver todo lo necesario, procederé a enseñarlo en una ventana a parte. También recomiendo descargar la imagen pulsando click derecho y “Guardar como imagen” para poder ampliarla.

La función “visualizingDICOM” devuelve void y sus parámetros son los siguientes:

General | Parameters

Name	Type
imageThreshold	vtkSmartPointer<vtkImageThreshold>
imageViewer	vtkSmartPointer<vtkImageViewer2>
sliceTextProp	vtkSmartPointer<vtkTextProperty>
sliceTextMapper	vtkSmartPointer<vtkTextMapper>
sliceTextActor	vtkSmartPointer<vtkActor2D>
usageTextProp	vtkSmartPointer<vtkTextProperty>
usageTextMapper	vtkSmartPointer<vtkTextMapper>
usageTextActor	vtkSmartPointer<vtkActor2D>
renderWindowInteractor	vtkSmartPointer<vtkRenderWindowInteractor>
myInteractorStyle	vtkSmartPointer<myVtkInteractorStyleImage>

Ilustración 2.1.2 Diagramas de clase: Función 1

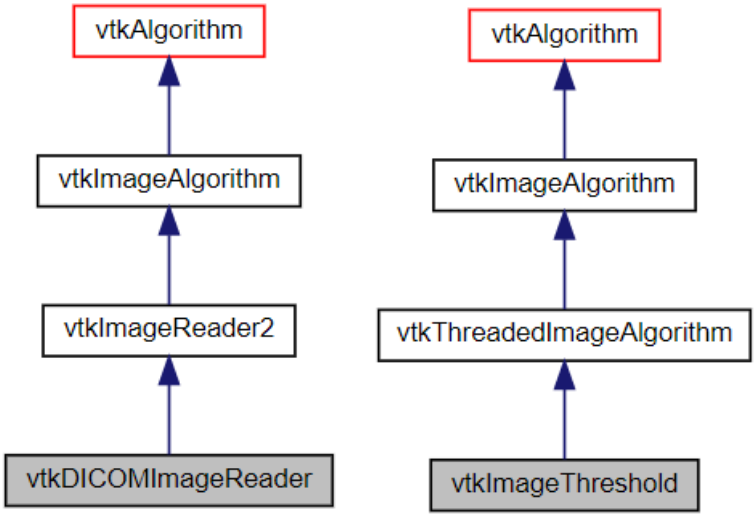
La función “preparing3DModel” devuelve void y sus parámetros son los siguientes:

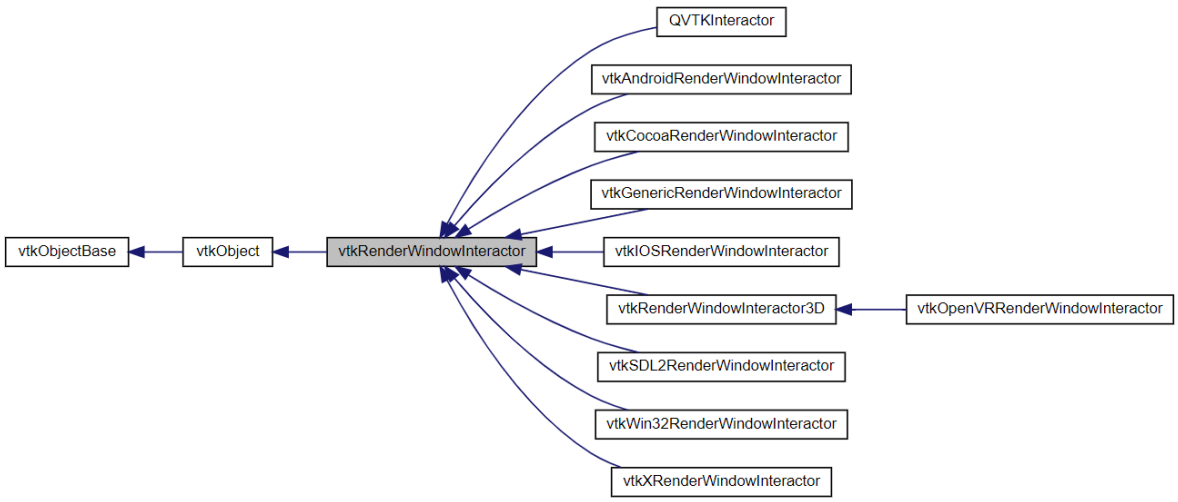
General | Parameters

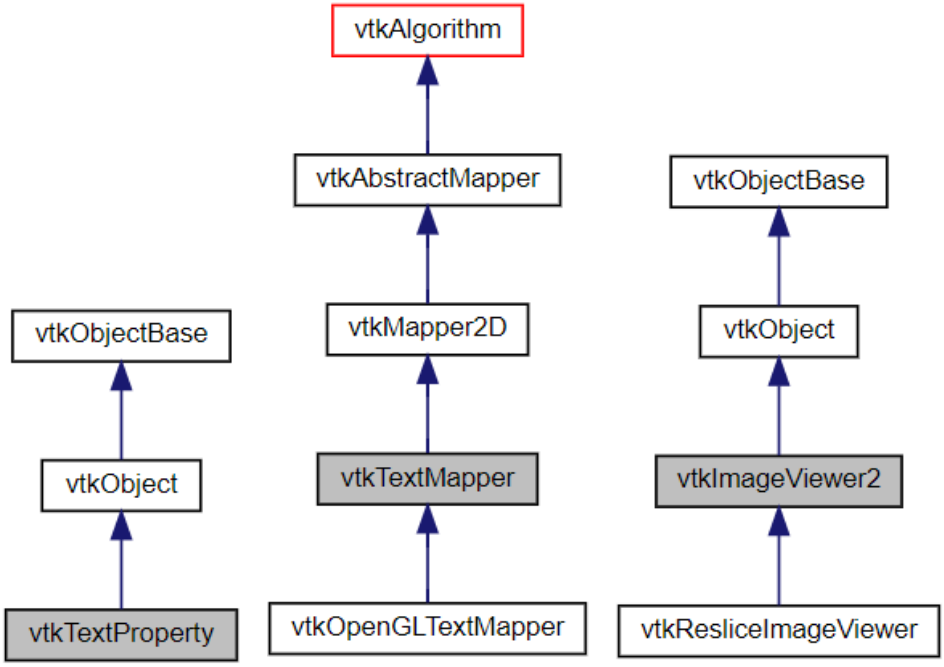
Name	Type	Defau...	Direct...
imageThreshold	vtkSmartPointer<vtkImageThreshold>		inout
detailLevel	double		inout
colors	vtkSmartPointer<vtkNamedColors>		inout
skinColor	array<unsigned char, 4>		inout
bkg	array<unsigned char, 4>		inout
aRenderer	vtkSmartPointer<vtkRenderer>		inout
renWin	vtkSmartPointer<vtkRenderWindow>		inout
iren	vtkSmartPointer<vtkRenderWindowInteractor>		inout
skinExtractor	vtkSmartPointer<vtkMarchingCubes>		inout
skinMapper	vtkSmartPointer<vtkPolyDataMapper>		inout
skin	vtkSmartPointer<vtkActor>		inout
outlineData	vtkSmartPointer<vtkOutlineFilter>		inout
mapOutline	vtkSmartPointer<vtkPolyDataMapper>		inout
outline	vtkSmartPointer<vtkActor>		inout

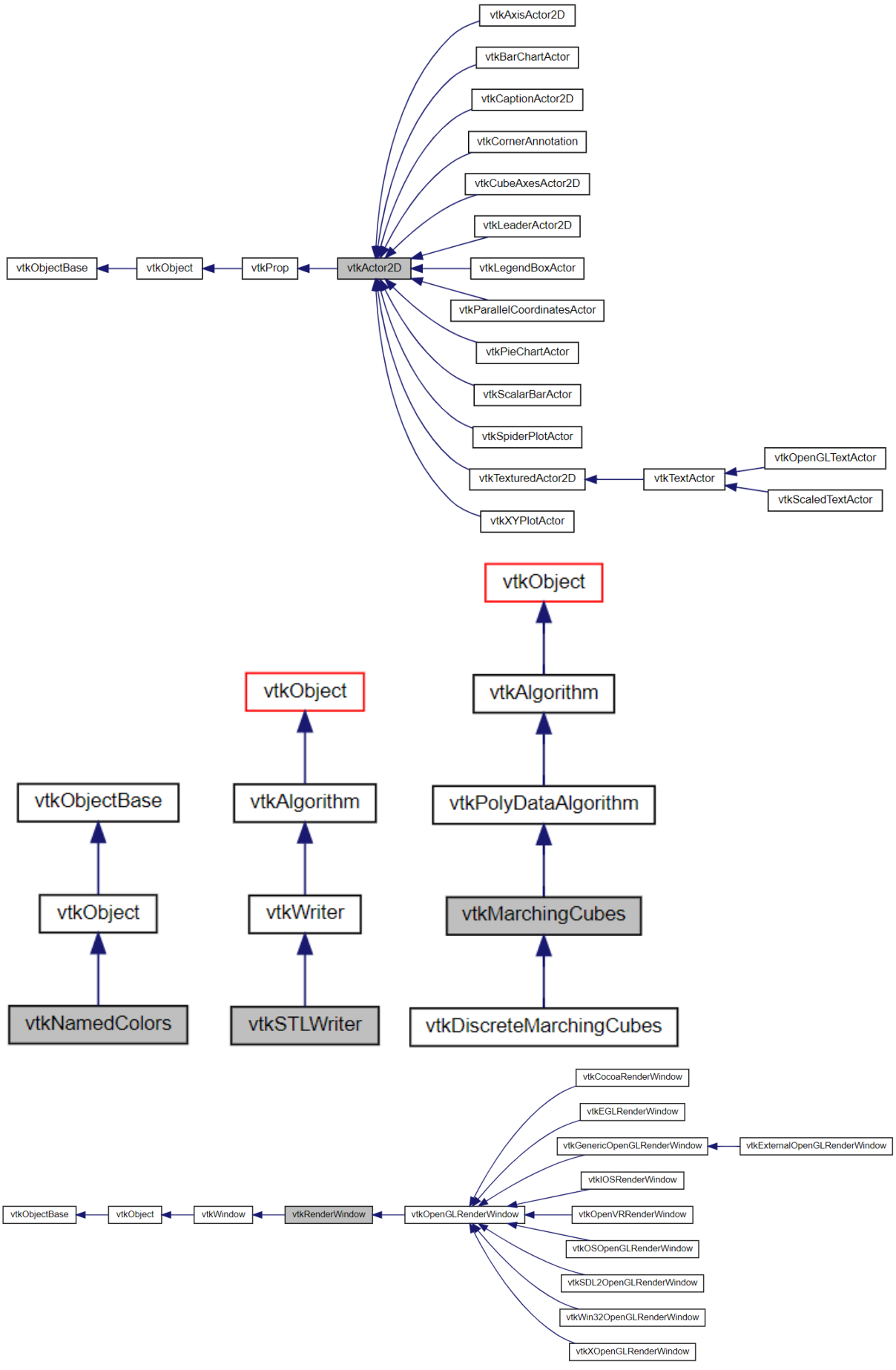
Ilustración 2.1.2 Diagramas de clase: Función 2

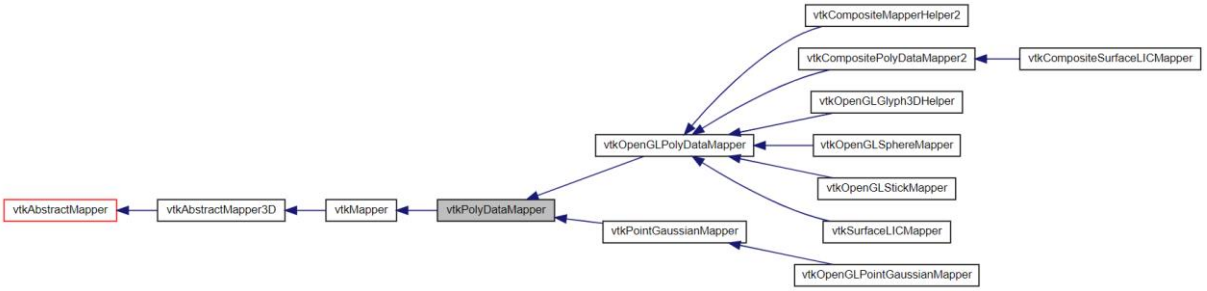
La mayoría de las clases son de VTK [4] y sus diagramas son los siguientes.

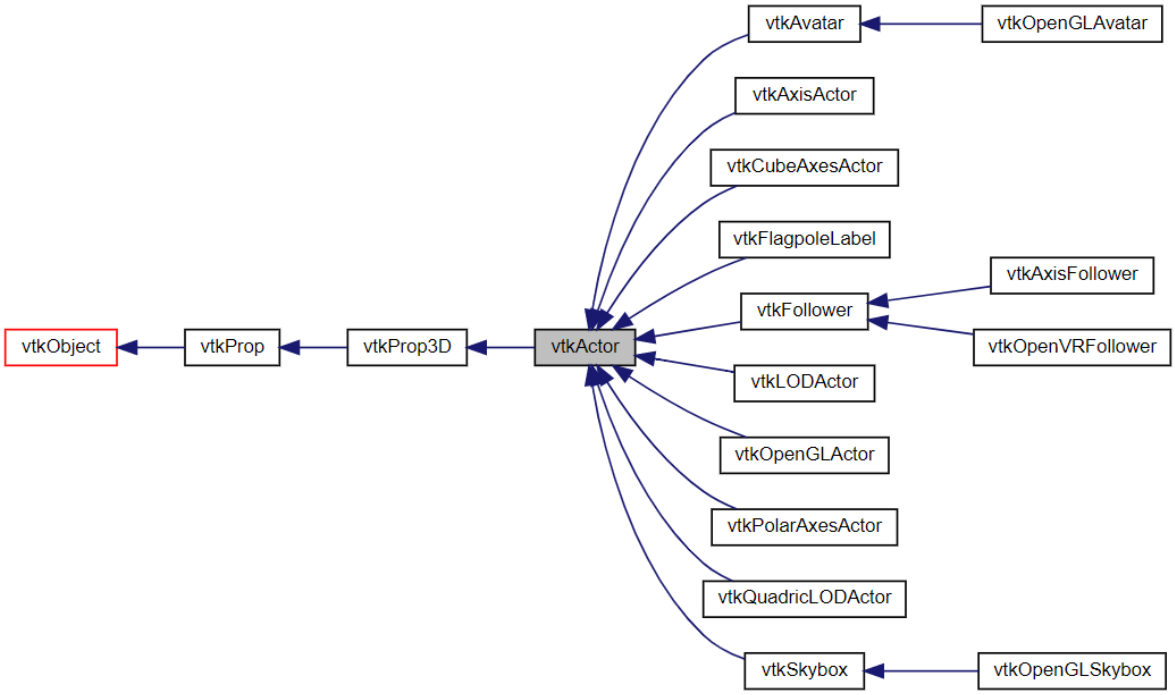



QVTKInteractor
vtkAndroidRenderWindowInteractor
vtkCocoaRenderWindowInteractor
vtkGenericRenderWindowInteractor
vtkObjectBase
vtkObject
vtkRenderWindowInteractor
vtkIOSRenderWindowInteractor
vtkRenderWindowInteractor3D
vtkOpenVRRenderWindowInteractor
vtkSDL2RenderWindowInteractor
vtkWin32RenderWindowInteractor
vtkXRenderWindowInteractor


vtkAlgorithm
vtkAbstractMapper
vtkObjectBase
vtkObjectBase
vtkMapper2D
vtkObject
vtkObject
vtkTextMapper
vtkImageViewer2
vtkTextProperty
vtkOpenGLTextMapper
vtkResliceImageViewer


vtkAxisActor2D
vtkBarChartActor
vtkCaptionActor2D
vtkCornerAnnotation
vtkCubeAxesActor2D
vtkLeaderActor2D
vtkObjectBase
vtkObject
vtkProp
vtkActor2D
vtkLegendBoxActor
vtkParallelCoordinatesActor
vtkPieChartActor
vtkScalarBarActor
vtkSpiderPlotActor
vtkTexturedActor2D
vtkTextActor
vtkOpenGLTextActor
vtkScaledTextActor
vtkXYPlotActor
vtkObject
vtkAlgorithm
vtkObjectBase
vtkAlgorithm
vtkPolyDataAlgorithm
vtkObject
vtkWriter
vtkMarchingCubes
vtkNamedColors
vtkSTLWriter
vtkDiscreteMarchingCubes
vtkCocoaRenderWindow
vtkEGLRenderWindow
vtkGenericOpenGLRenderWindow
vtkExternalOpenGLRenderWindow
vtkIOSRenderWindow
vtkObjectBase
vtkObject
vtkWindow
vtkRenderWindow
vtkOpenGLRenderWindow
vtkOpenVRRenderWindow
vtkOSOpenGLRenderWindow
vtkSDL2OpenGLRenderWindow
vtkWin32OpenGLRenderWindow
vtkXOpenGLRenderWindow


vtkCompositeMapperHelper2
vtkCompositePolyDataMapper2
vtkCompositeSurfaceLICMapper
vtkOpenGLGlyph3DHelper
vtkOpenGLPolyDataMapper
vtkOpenGLSphereMapper
vtkOpenGLStickMapper
vtkAbstractMapper
vtkAbstractMapper3D
vtkMapper
vtkPolyDataMapper
vtkPointGaussianMapper
vtkSurfaceLICMapper
vtkOpenGLPointGaussianMapper


vtkAvatar
vtkOpenGLAvatar
vtkAxisActor
vtkCubeAxesActor
vtkFlagpoleLabel
vtkAxisFollower
vtkFollower
vtkObject
vtkProp
vtkProp3D
vtkActor
vtkOpenVRFollower
vtkLODActor
vtkOpenGLActor
vtkPolarAxesActor
vtkQuadricLODActor
vtkSkybox
vtkOpenGLSkybox

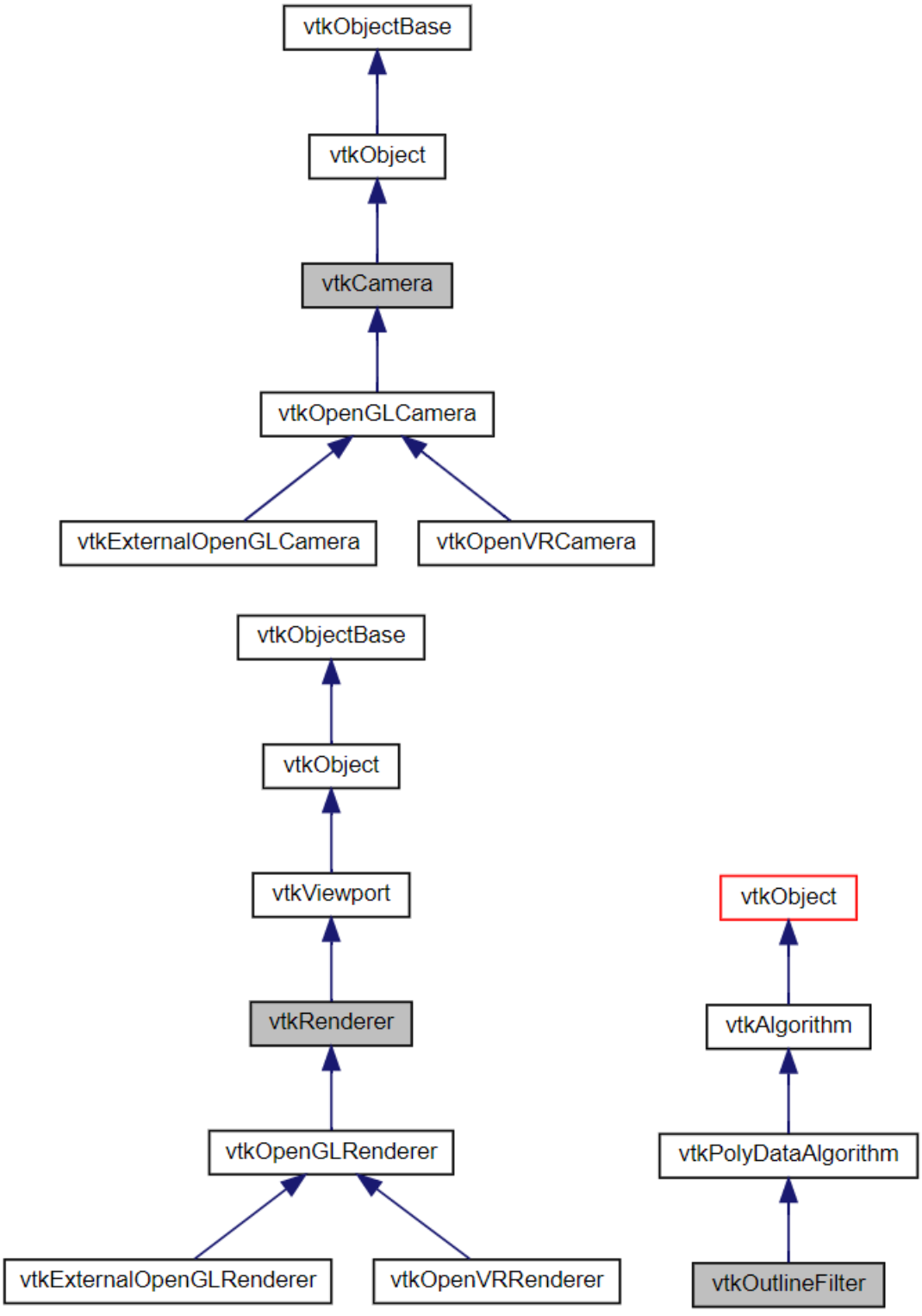


Ilustración 2.1.2 Diagramas de clase: Diagramas VTK

2.2 - Implementación

La implementación de este proyecto se ha llevado a cabo de la siguiente forma:

En primer lugar, se implementó una función que fuera capaz de leer archivos DICOM, para ello, fue necesario el módulo de VTK “vtkDICOMImageReader”.

```
void DICOMReader(vtkSmartPointer<vtkDICOMImageReader> reader, std::string folder) {
    reader->SetDirectoryName(folder.c_str());
    reader->Update();
}
```

Este módulo dispone de la función “SetDirectoryName”, si a esta le pasamos un directorio correcto, cargará nuestro objeto con el archivo DICOM. Hay varias cosas a tener en cuenta, las cuales, se repetirán a lo largo de todo el proyecto, una de ellas es que todas las variables de VTK se crean como punteros inteligentes (“vtkSmartPointer”), con esto nos evitamos las copias y hacemos el programa más eficiente. Lo último a tener en cuenta por ahora es que los cambios que se hagan en una variable de VTK tienen que ser actualizados para que se realice el proceso deseado, esa sería la explicación de la última llamada (“Update()”).

El siguiente paso fue la creación de la función que se encargara de umbralizar las imágenes que hemos leído con el con “DICOMReader”.

```
void thresholding(vtkSmartPointer<vtkDICOMImageReader> reader, vtkSmartPointer<vtkImageThreshold> imageThreshold,
                  double lower, double upper) {

    imageThreshold->SetInputConnection(reader->GetOutputPort());

    imageThreshold->ThresholdBetween(lower, upper);
    imageThreshold->ReplaceInOn();
    imageThreshold->SetInValue(0);
    imageThreshold->Update();
}
```

VTK trabaja conectando salidas y entradas, es decir, en este ejemplo, como entrada de nuestro objeto de tipo “vtkImageThreshold”, necesitamos imágenes, por tanto, cogemos la salida de nuestro objeto de tipo “vtkDICOMImageReader”. Teniendo las imágenes, ya solo nos hace falta parametrizar el proceso, con la función “ThresholdBetween()” especificamos el umbral mínimo y máximo de la umbralización.

La siguiente función “ReplaceInOn()” determina que se debe reemplazar el píxel que se encuentre en el rango de umbralización. “SetInValue(0)” determina el valor por el cual se reemplaza el píxel que se encuentre en el rango, negro en este caso. Por último, actualizamos con los parámetros deseados y realiza la umbralización. Esta función sería la que habría que reemplazar si se deseara cambiar o ampliar este proyecto con distintos métodos de segmentación.

El paso que vamos a ver a continuación es la creación de la función que nos permite visualizar los archivos DICOM. Para esta ha sido necesario crear una clase que indique los métodos que vamos a usar para la interacción.

```
// Define own interaction style
class myVtkInteractorStyleImage : public vtkInteractorStyleImage
{
public:
   static myVtkInteractorStyleImage* New();
   vtkTypeMacro(myVtkInteractorStyleImage, vtkInteractorStyleImage);

protected:
   vtkImageViewer2* _ImageViewer;
   vtkTextMapper* _StatusMapper;
   int _Slice;
   int _MinSlice;
   int _MaxSlice;

public:
   void SetImageViewer(vtkImageViewer2* imageViewer) {
      _ImageViewer = imageViewer;
      _MinSlice = imageViewer->GetSliceMin();
      _MaxSlice = imageViewer->GetSliceMax();
      _Slice = _MinSlice;
      cout << "Slicer: Min = " << _MinSlice << ", Max = " << _MaxSlice << std::endl;
   }
```

Esta hereda de una clase de VTK llamada “vtkInteractorStyleImage” y las variables que necesitan serían el propio visualizador de imágenes (“vtkImageViewer2”), el mapeador de estado (“vtkTextMapper”) y tres enteros que indican la imagen actual, el número del primer slice y el número del último slice.

Una de las funciones implementadas para su funcionamiento han sido “SetImageViewer()” con la que rellenamos la mayoría de nuestras variables, con el visualizador de imágenes ya podemos obtener el número de slices. El slice actual empezaría por el primero, por eso se iguala al mínimo.

```
void SetStatusMapper(vtkTextMapper* statusMapper) {
   _StatusMapper = statusMapper;
}
```

Se ha creado una función a parte para dar valor a nuestro mapeador de estado.

```
protected:
    void MoveSliceForward() {
        if(_Slice < _MaxSlice) {
            _Slice += 1;
            cout << "MoveSliceForward::Slice = " << _Slice << std::endl;
            _ImageViewer->SetSlice(_Slice);
            std::string msg = StatusMessage::Format(_Slice, _MaxSlice);
            _StatusMapper->SetInput(msg.c_str());
            _ImageViewer->Render();
        }
    }

    void MoveSliceBackward() {
        if(_Slice > _MinSlice) {
            _Slice -= 1;
            cout << "MoveSliceBackward::Slice = " << _Slice << std::endl;
            _ImageViewer->SetSlice(_Slice);
            std::string msg = StatusMessage::Format(_Slice, _MaxSlice);
            _StatusMapper->SetInput(msg.c_str());
            _ImageViewer->Render();
        }
    }
```

Después se crearon dos de las principales funciones, las encargadas de pasar entre las distintas imágenes. Primero comprobamos que no podamos pasarnos de los límites del número de imágenes, una vez seguros de esto, incrementamos o decrementamos el slice que queremos que se nos muestre y se lo indicamos al visualizador de imágenes para que cambie el slice que se nos está mostrando. Por último, aportamos feedback al usuario para que conozca el slice en el que se encuentra, para ello, se ha creado una clase para dar un formato al mensaje.

```
// helper class to format slice status message
class StatusMessage {
public:
    static std::string Format(int slice, int maxSlice) {
        std::stringstream tmp;
        tmp << "Slice Number  " << slice + 1 << "/" << maxSlice + 1;
        return tmp.str();
    }
};
```

Le pasamos al mapeador de estados el mensaje que queremos que muestre al usuario e indicamos al visualizador de imágenes que renderice la nueva imagen.

Para terminar con esta clase, se han implementado las funciones que indican las acciones a realizar con las entradas de la rueda del ratón y las flechas del teclado.

```
    virtual void OnKeyDown() {
        std::string key = this->GetInteractor()->GetKeySym();
        if(key.compare("Up") == 0) {
            //cout << "Up arrow key was pressed." << endl;
            MoveSliceForward();
        }
        else if(key.compare("Down") == 0) {
            //cout << "Down arrow key was pressed." << endl;
            MoveSliceBackward();
        }
        // forward event
        vtkInteractorStyleImage::OnKeyDown();
    }


    virtual void OnMouseWheelForward() {
        //std::cout << "Scrolled mouse wheel forward." << std::endl;
        MoveSliceForward();
        // don't forward events, otherwise the image will be zoomed
        // in case another interactorstyle is used (e.g. trackballstyle, ...)
        // vtkInteractorStyleImage::OnMouseWheelForward();
    }


    virtual void OnMouseWheelBackward() {
        //std::cout << "Scrolled mouse wheel backward." << std::endl;
        if(_Slice > _MinSlice) {
            MoveSliceBackward();
        }
        // don't forward events, otherwise the image will be zoomed
        // in case another interactorstyle is used (e.g. trackballstyle, ...)
        // vtkInteractorStyleImage::OnMouseWheelBackward();
    }
};
```

De esta forma, si pulsamos la flecha hacia arriba o movemos la rueda del ratón hacia arriba, nos moveremos entre los slices en el sentido positivo, pulsando la tecla de la flecha hacia abajo o movemos la rueda hacia abajo, nos moveremos entre los slices en el sentido negativo. Si no se implementan las acciones de la rueda del ratón, estás tienen de forma predeterminada hacer zoom en la imagen, lo mismo que si movemos el ratón mientras pulsamos click derecho. Como esta última función no ha sido sobrescrita, realizará el zoom.

Con esta clase explicada, ya podemos ver la función encargada de mostrar las imágenes DICOM. Recibe un gran número de parámetros, los iremos comentando a medida que los vayamos utilizando

```
void visualizingDICOM(vtkSmartPointer<vtkImageThreshold> imageThreshold, vtkSmartPointer<vtkImageViewer2> imageViewer,
                      vtkSmartPointer<vtkTextProperty> sliceTextProp, vtkSmartPointer<vtkTextMapper> sliceTextMapper,
                      vtkSmartPointer<vtkActor2D> sliceTextActor, vtkSmartPointer<vtkTextProperty> usageTextProp,
                      vtkSmartPointer<vtkTextMapper> usageTextMapper, vtkSmartPointer<vtkActor2D> usageTextActor,
                      vtkSmartPointer<vtkRenderWindowInteractor> renderWindowInteractor,
                      vtkSmartPointer<myVtkInteractorStyleImage> myInteractorStyle) {
    imageViewer->SetInputConnection(imageThreshold->GetOutputPort());

    // slice status message
    sliceTextProp->SetFontFamilyToCourier();
    sliceTextProp->SetFontSize(20);
    sliceTextProp->SetVerticalJustificationToBottom();
    sliceTextProp->SetJustificationToLeft();

    std::string msg = StatusMessage::Format(imageViewer->GetSliceMin(), imageViewer->GetSliceMax());
    sliceTextMapper->SetInput(msg.c_str());
    sliceTextMapper->SetTextProperty(sliceTextProp);

    sliceTextActor->SetMapper(sliceTextMapper);
    sliceTextActor->SetPosition(15, 10);
```

En primer lugar, realizamos la conexión entre el visualizador de imágenes y las propias imágenes que se van a mostrar, en este caso, las imágenes segmentadas.

En segundo lugar, configuramos los mensajes de estado de los slices que proporcionaran feedback al usuario. Necesitaremos tres objetos, uno de tipo “vtkTextProperty”, otro de tipo “vtkTextMapper” y por último uno de tipo “vtkActor2D”. Al principio, configuramos las propiedades del texto que vamos a mostrar, después, especificamos al mapeador el mensaje que va a mostrar y las propiedades que hemos configurado anteriormente y por último vinculamos al actor con el mapeador y especificamos la posición donde se encontrará el mensaje.

```
    // usage hint message
    usageTextProp->SetFontFamilyToCourier();
    usageTextProp->SetFontSize(14);
    usageTextProp->SetVerticalJustificationToTop();
    usageTextProp->SetJustificationToLeft();

    usageTextMapper->SetInput("- Slice with mouse wheel\n  or Up/Down-Key\n- Zoom with pressed right\n  mouse button while dragging");
    usageTextMapper->SetTextProperty(usageTextProp);

    usageTextActor->SetMapper(usageTextMapper);
    usageTextActor->GetPositionCoordinate()->SetCoordinateSystemToNormalizedDisplay();
    usageTextActor->GetPositionCoordinate()->SetValue(0.05, 0.95);
```

La siguiente parte de la función es la encargada de dar feedback al usuario sobre la forma de usar la interfaz, es muy parecida a la anterior y necesita los mismos tipos de objetos. Uno para las propiedades del texto, otro para mapear y un actor.

```
// make imageviewer2 and sliceTextMapper visible to our interactorstyle
// to enable slice status message updates when scrolling through the slices
myInteractorStyle->SetImageViewer(imageViewer);
myInteractorStyle->SetStatusMapper(sliceTextMapper);

imageViewer->SetupInteractor(renderWindowInteractor);
// make the interactor use our own interactorstyle
// cause SetupInteractor() is defining it's own default interatorstyle
// this must be called after SetupInteractor()
renderWindowInteractor->SetInteractorStyle(myInteractorStyle);
// add slice status message and usage hint message to the renderer
imageViewer->GetRenderer()->AddActor2D(sliceTextActor);
imageViewer->GetRenderer()->AddActor2D(usageTextActor);
```

Ya solo nos quedan los últimos pasos de esta función. La variable “myInteractorStyle” es del tipo “myVtkInteractorStyleImage”, recordemos que esta fue la clase que creamos anteriormente para programar las interacciones. Esta necesitaba el visualizador de imágenes y el mapeador que contiene el estado de los slices. La variable “renderWindowInteractor” es del tipo “vtkRenderWindowInteractor”, es la encargada de crear la ventana donde se visualizarán las imágenes, por ahora, solo es necesario indicarle el estilo de interacción. Por último, le ordenamos a nuestro visualizador de imágenes que renderice los mensajes encargados de proporcionar feedback.

En esta misma función, podríamos crear las ordenes necesarias para que empezara la visualización de las imágenes, pero decidí crear una función distinta para ello. Esto es porque prefiero que la visualización empiece cuando termine los procesos de creación de modelo y guardado automático del modelo. De esta forma, el usuario solo tendrá un tiempo de espera, que será el inicial. Una vez se realicen todos los procesos costosos, aparecerán las ventanas y el usuario no tendrá que esperar por ningún otro proceso.

La siguiente función es la encargada de crear y preparar el modelo 3D a partir de las imágenes segmentadas. Como la función anterior, contiene un alto número de parámetros, se irán explicando mientras aparecen.

```
void preparing3DModel(vtkSmartPointer<vtkImageThreshold> imageThreshold, double detailLevel, vtkSmartPointer<vtkNamedColors> colors,
                      std::array<unsigned char, 4> skinColor, std::array<unsigned char, 4> bkg,
                      vtkSmartPointer<vtkRenderer> aRenderer, vtkSmartPointer<vtkRenderWindow> renWin, vtkSmartPointer<vtkRenderWindowInteractor> iren,
                      vtkSmartPointer<vtkMarchingCubes> skinExtractor, vtkSmartPointer<vtkPolyDataMapper> skinMapper, vtkSmartPointer<vtkActor> skin,
                      vtkSmartPointer<vtkOutlineFilter> outlineData, vtkSmartPointer<vtkPolyDataMapper> mapOutline, vtkSmartPointer<vtkActor> outline) {
    colors->SetColor("SkinColor", skinColor.data());
    colors->SetColor("BkgColor", bkg.data());

    renWin->AddRenderer(aRenderer);
    iren->SetRenderWindow(renWin);

    skinExtractor->SetInputConnection(imageThreshold->GetOutputPort());
    skinExtractor->SetValue(0, detailLevel);

    skinMapper->SetInputConnection(skinExtractor->GetOutputPort());
    skinMapper->ScalarVisibilityOff();

    skin->SetMapper(skinMapper);
    skin->GetProperty()->SetDiffuseColor(colors->GetColor3d("SkinColor").GetData());

    outlineData->SetInputConnection(imageThreshold->GetOutputPort());

    mapOutline->SetInputConnection(outlineData->GetOutputPort());

    outline->SetMapper(mapOutline);
    outline->GetProperty()->SetColor(colors->GetColor3d("Black").GetData());
}
```

En primer lugar, actuamos con la variable que nos ayudará a indicar los distintos colores de la escena, la cual, es de tipo “vtkNamedColors”. En este objeto, podemos guardar los colores que queramos y ponerles un nombre, para ello, solo necesita un string que indica el nombre que le otorgamos a un color y el color en si en formato RGBA. Por ello, en las dos primeras líneas, los datos sobre el color que le otorgamos son un array de tamaño 4.

Las dos siguientes líneas, se encargan de indicar a la ventana que muestra el render (“vtkRenderWindow”), cual es el render (“vtkRenderer”) que mostrará y de indicar a nuestro interactuador de ventana de renderizado (“vtkRenderWindowInteractor”), la ventana de renderizado a la que va asociada. Las interacciones de esta ventana no hace falta programarlas en otra clase como las interacciones de la visualización de imágenes, las que vienen por defecto son suficientes.

Las seis líneas siguientes serán las encargadas de crear nuestro modelo a partir de las imágenes, los objetos encargados de esto son de tipo “vtkMarchingCubes”, “vtkPolyDataMapper” y “vtkActor”. El primero es el encargado de crear la malla mediante, este necesita las imágenes segmentadas y que especifiquemos el nivel de detalle del modelo. El siguiente necesita del modelo creado anteriormente para mapear los datos del polígono, seguidamente, indicamos que se cree una tabla de búsqueda por defecto. Por último, vinculamos al actor con el mapeador y añadimos el color con el que queremos que se muestre el hueso, el cual, hemos guardado al principio de la función.

Las últimas líneas son las encargadas de crear el contorno de nuestro modelo, necesita de tres objetos de tipo “vtkOutlineFilter”, “vtkPolyDataMapper” y “vtkActor”. El

primero es el encargado de crear el contorno del modelo con ayuda de las imágenes segmentadas, el segundo mapea los datos del polígono con ayuda del contorno que acabamos de crear y por último vinculamos el actor con este mapeador e indicamos el color del contorno, negro en este caso.

Ya hemos creado la visualización de nuestras imágenes y la visualización de nuestro modelo, ya solo queda guardar el modelo en formato STL y mandar que se vayan mostrando las ventanas para nuestro usuario.

La creación de un archivo STL es muy sencilla con la ayuda del módulo “vtkSTLWriter”.

```
void stlConversion(vtkSmartPointer<vtkMarchingCubes> skinExtractor, vtkSmartPointer<vtkSTLWriter> stlWriter, const char* modelName) {
    stlWriter->SetFileName(modelName);
    stlWriter->SetInputConnection(skinExtractor->GetOutputPort());
    stlWriter->Write();
}
```

En estas líneas, lo que hacemos es indicar cual será el nombre que tendrá el archivo, vincular con el modelo que queremos guardar y mandar la escritura del archivo. Por defecto, el archivo se guarda donde se ejecute la aplicación.

La penúltima función realiza las acciones necesarias para que se muestre la visualización de las imágenes al usuario.

```
/*Function to initialize and insert the default values of the DICOM file display*/
void DICOMVisualizationDefaultSettings(vtkSmartPointer<vtkImageViewer2> imageViewer, vtkSmartPointer<vtkRenderWindowInteractor> renderWindowInteractor) {
    // initialize rendering and interaction
    //imageViewer->GetRenderWindow()->SetSize(400, 300);
    //imageViewer->GetRenderer()->SetBackground(0.2, 0.3, 0.4);
    imageViewer->Render();
    imageViewer->GetRenderer()->ResetCamera();
    imageViewer->Render();
    renderWindowInteractor->Start();
}
```

Indicamos a nuestro visualizador que renderice la imagen inicial, reseteamos la cámara para asegurarnos de que se posiciona correctamente y volvemos a renderizar para guardar este último cambio. Por último, iniciamos la visualización.

La última función realiza las acciones necesarias para que se muestre la visualización del modelo 3D al usuario. Esta es algo más tediosa debido a que tenemos que configurar la cámara para que la visualización inicial quede como quiero.

```
/*Function to insert the default options of the camera, the background, the display window,
the actors to be displayed and the initialization of the display event*/
void ModelVisualizationDefaultSettings(vtkSmartPointer<vtkCamera> aCamera, vtkSmartPointer<vtkRenderer> aRenderer,
                                       vtkSmartPointer<vtkActor> outline, vtkSmartPointer<vtkActor> skin, vtkSmartPointer<vtkNamedColors> colors,
                                       vtkSmartPointer<vtkRenderWindow> renWin, vtkSmartPointer<vtkRenderWindowInteractor> iren) {

    // It is convenient to create an initial view of the data. The FocalPoint
    // and Position form a vector direction. Later on (ResetCamera() method)
    // this vector is used to position the camera to look at the data in
    // this direction.
    aCamera->SetViewUp(0, 0, -1);
    aCamera->SetPosition(0, -1, 0);
    aCamera->SetFocalPoint(0, 0, 0);
    aCamera->ComputeViewPlaneNormal();
    aCamera->Azimuth(30.0);
    aCamera->Elevation(30.0);

    // Actors are added to the renderer. An initial camera view is created.
   // The Dolly() method moves the camera towards the FocalPoint,
   // thereby enlarging the image.
    aRenderer->AddActor(outline);
    aRenderer->AddActor(skin);

    aRenderer->SetActiveCamera(aCamera);
    aRenderer->ResetCamera();
    aCamera->Dolly(1.5);

    // Set a background color for the renderer and set the size of the
    // render window (expressed in pixels).
    aRenderer->SetBackground(colors->GetColor3d("BkgColor").GetData());
    renWin->SetSize(640, 480);
```

La cámara es de tipo “vtkCamera” y las primeras líneas son de configuración, con esto, lo que hacemos es que la cámara se encuentre en la posición deseada y mire hacia el modelo 3D.

A continuación, añadimos los actores del modelo que hemos creado en la función “preparing3DModel” a nuestro render. Lo hago aquí y no antes porque es recomendable crear la cámara antes y para mantener un orden. Seguidamente indicamos la cámara que va a usar nuestro render y la reseteamos por motivos de seguridad. Con el método “Dolly()” movemos la cámara hacia el punto focal, de esta forma, ampliamos la imagen.

Después, indicamos a nuestro reder el color de fondo que queremos que tenga la escena (el cual guardamos anteriormente) e indicamos las dimensiones de la ventana.

```
    // Note that when camera movement occurs (as it does in the Dolly()
    // method), the clipping planes often need adjusting. Clipping planes
    // consist of two planes: near and far along the view direction. The
    // near plane clips out objects in front of the plane; the far plane
    // clips out objects behind the plane. This way only what is drawn
    // between the planes is actually rendered.
    aRenderer->ResetCameraClippingRange();


    // Initialize the event loop and then start it.
    renWin->Render();
    iren->Initialize();
    iren->Start();
```

Por último, reseteamos los planos de recorte para asegurarnos de que el modelo quede entre ellos y no sea recortado, indicamos a nuestra ventana que renderice y iniciamos la visualización para el usuario.

Con estas funciones y clases creadas, el sistema cuenta con lo necesario para iniciar la aplicación. Las siguientes imágenes son líneas de código que se encuentran en nuestro main, en estas, vamos creando los objetos que vamos necesitando y llamando a las funciones creadas anteriormente.

```
// Read all the DICOM files in the specified directory.
vtkSmartPointer<vtkDICOMImageReader> reader = vtkSmartPointer<vtkDICOMImageReader>::New();

DICOMReader(reader, folder);

//Performing the threshold to the DICOM files
vtkSmartPointer<vtkImageThreshold> imageThreshold = vtkSmartPointer<vtkImageThreshold>::New();

thresholding(reader, imageThreshold, lower, upper);


// Visualizing DICOM files with threshold
vtkSmartPointer<vtkImageViewer2> imageViewer =
   vtkSmartPointer<vtkImageViewer2>::New();

// slice status message
vtkSmartPointer<vtkTextProperty> sliceTextProp = vtkSmartPointer<vtkTextProperty>::New();
vtkSmartPointer<vtkTextMapper> sliceTextMapper = vtkSmartPointer<vtkTextMapper>::New();
vtkSmartPointer<vtkActor2D> sliceTextActor = vtkSmartPointer<vtkActor2D>::New();

// usage hint message
vtkSmartPointer<vtkTextProperty> usageTextProp = vtkSmartPointer<vtkTextProperty>::New();
vtkSmartPointer<vtkTextMapper> usageTextMapper = vtkSmartPointer<vtkTextMapper>::New();
vtkSmartPointer<vtkActor2D> usageTextActor = vtkSmartPointer<vtkActor2D>::New();

// create an interactor with our own style (inherit from vtkInteractorStyleImage)
// in order to catch mousewheel and key events
vtkSmartPointer<vtkRenderWindowInteractor> renderWindowInteractor =
   vtkSmartPointer<vtkRenderWindowInteractor>::New();
```

```
vtkSmartPointer<myVtkInteractorStyleImage> myInteractorStyle =
   vtkSmartPointer<myVtkInteractorStyleImage>::New();

visualizingDICOM(imageThreshold, imageViewer, sliceTextProp, sliceTextMapper, sliceTextActor, usageTextProp, usageTextMapper,
                 usageTextActor, renderWindowInteractor, myInteractorStyle);

//preparing the 3D model
vtkSmartPointer<vtkNamedColors> colors = vtkSmartPointer<vtkNamedColors>::New();
std::array<unsigned char, 4> skinColor{ {255, 255, 255} };
std::array<unsigned char, 4> bkg{ {51, 77, 102, 255} };

// Create the renderer, the render window, and the interactor. The renderer
// draws into the render window, the interactor enables mouse- and
// keyboard-based interaction with the data within the render window.
vtkSmartPointer<vtkRenderer> aRenderer = vtkSmartPointer<vtkRenderer>::New();
vtkSmartPointer<vtkRenderWindow> renWin = vtkSmartPointer<vtkRenderWindow>::New();
vtkSmartPointer<vtkRenderWindowInteractor> iren = vtkSmartPointer<vtkRenderWindowInteractor>::New();

// An isosurface, or contour value of 460 is known to correspond to the skin of the patient.
vtkSmartPointer<vtkMarchingCubes> skinExtractor = vtkSmartPointer<vtkMarchingCubes>::New();
vtkSmartPointer<vtkPolyDataMapper> skinMapper = vtkSmartPointer<vtkPolyDataMapper>::New();
vtkSmartPointer<vtkActor> skin = vtkSmartPointer<vtkActor>::New();

// An outline provides context around the data.
vtkSmartPointer<vtkOutlineFilter> outlineData = vtkSmartPointer<vtkOutlineFilter>::New();
vtkSmartPointer<vtkPolyDataMapper> mapOutline = vtkSmartPointer<vtkPolyDataMapper>::New();
vtkSmartPointer<vtkActor> outline = vtkSmartPointer<vtkActor>::New();

preparing3DModel(imageThreshold, detailLevel, colors, skinColor, bkg, aRenderer, renWin, iren, skinExtractor, skinMapper,
                 skin, outlineData, mapOutline, outline);


//Lets save the generated 3D model in an stl file
vtkSmartPointer<vtkSTLWriter> stlWriter = vtkSmartPointer<vtkSTLWriter>::New();
stlConversion(skinExtractor, stlWriter, modelName);

//Creation of the camera for the visualization of the 3d model
vtkSmartPointer<vtkCamera> aCamera = vtkSmartPointer<vtkCamera>::New();

//First we visualize the DICOM file and then the generated 3D model
DICOMVisualizationDefaultSettings(imageViewer, renderWindowInteractor);
ModelVisualizationDefaultSettings(aCamera, aRenderer, outline, skin, colors, renWin, iren);


return EXIT_SUCCESS;
```

Lo último que vamos a ver de código, será la programación de la recogida de argumentos por línea de comandos. Se realizó lo último, ya sabiendo los parámetros que serían necesarios.

```
int main(int argc, char* argv[])
{
    std::cout << "Warning: If the DICOM file path contains spaces, put quotation marks around it so that it is detected as a single argument" << std::endl;
    // Verify input arguments
    if (argc != 6)
    {
        std::cout << "Usage: " << argv[0]
            << " DICOMFolderPath"
            << " ModelDetailLevel"
            << " MinimumThresholdingValue"
            << " MaximumThresholdingValue"
            << " ModelName" << std::endl;
        return EXIT_FAILURE;
    }
    std::string folder = argv[1];
    double detailLevel = atof(argv[2]);
    double lower = atof(argv[3]);
    double upper = atof(argv[4]);

    char result[100];
    strcpy_s(result, argv[5]);
    strcat_s(result, ".stl");
    char* modelName = result;
```

Si el usuario intenta iniciar la aplicación sin el número de argumentos correcto, se le muestra un pequeño tutorial y seguidamente se van guardando los distintos parámetros en sus variables correspondientes. Recordemos que se necesita la ruta del archivo DICOM, el nivel de detalle del modelo, el valor mínimo y máximo de umbralización y el nombre del modelo. En este último, se añade automáticamente la extensión del archivo, de esta forma, no es necesario que el usuario lo haga.

```
if (!(detailLevel <= 500) || !(detailLevel > 200)) {
    std::cout << "ModelDetailValue should be a numeric value between 200 and 500" << std::endl;
    return EXIT_FAILURE;
}

if (!(lower >= 0) || !(lower < upper)) {
    std::cout << "MinimumThresholdingValue should be a numeric value between 0 and MaximumThresholdingValue" << std::endl;
    return EXIT_FAILURE;
}

if (!(upper > lower) || !(upper <= 255)) {
    std::cout << "MinimumThresholdingValue should be a numeric value between MinimumThresholdingValue and 255" << std::endl;
    return EXIT_FAILURE;
}

std::cout << "Parameters are correct, please wait, it may take a few minutes" << std::endl;
```

Por último, comprobamos distintas variables e informamos al usuario si no se especifican debidamente. El detalle del modelo tiene que estar entre 200 y 500, este rango es de elección personal, menos detalle de 500 haría que el modelo no fuera representativo y más detalle de 200 haría el proceso de creación del modelo demasiado lento y pensado, recordemos que mientras menor sea este valor, mayor es el detalle.

El valor de umbralización mínimo debe ser mayor de 0 y menor que el valor de umbralización máximo, de la misma manera, el valor de umbralización mínimo debe ser menor de 255 y mayor que el valor de umbralización mínimo.

Si el usuario intenta poner valores que no sean del mismo tipo que corresponde, por ejemplo, un valor no numérico al especificar el detalle del modelo, estas advertencias también aparecerán.

2.3 - Pruebas finales

2.3.1 - Pruebas de verificación del sistema

Prueba	Resultado
Comprobación de parámetros	Correcto
Lectura de archivo DICOM	Correcto
Segmentación de las imágenes mediante umbralización	Correcto

Generación de modelos	Correcto
Guardado automático de archivo STL	Correcto
Movimientos de cámara	Correcto
Desplazamiento entre slices	Correcto
Zoom en imágenes	Correcto
Zoom en modelos	Correcto
Ventana emergente de errores	Correcto
Cambio en los valores de umbralización	Correcto
Cambio en el detalle del modelo	Correcto

Tabla 2.3.1 Pruebas de verificación del sistema

2.3.2 - Pruebas de validación del sistema

Prueba	Resultado
¿La aplicación es ligera?	Sí
¿La aplicación es robusta?	Sí
¿La aplicación es eficiente?	Sí
¿Un usuario inexperto puede usar esta aplicación?	Sí
¿El usuario recibe feedback?	Sí
¿La aplicación cumple con los requisitos?	Sí
¿El usuario puede realizar las tareas para las que está destinada la aplicación?	Sí

Tabla 2.3.2 Pruebas de validación del sistema

3 - Resultados y Discusión

Las partes más importantes a discutir serían la segmentación de imágenes y la generación de modelos. En cuanto a la primera, como se comenta durante todo este documento, se ha utilizado el método de umbralización, el cual es capaz de obtener buenos resultados en muchos de los casos, pero como ya comentamos en el apartado 1.6 - A2: Análisis y Diseño del sistema, tendríamos un gran abanico de posibilidades. La generación de modelos se realiza con el algoritmo “Marching cubes” [5], siendo este el más utilizado para visualizaciones médicas y el que tiene implementado VTK.

Vamos a proceder a visualizar varios ejemplos, en primer lugar, veamos un húmero sano.

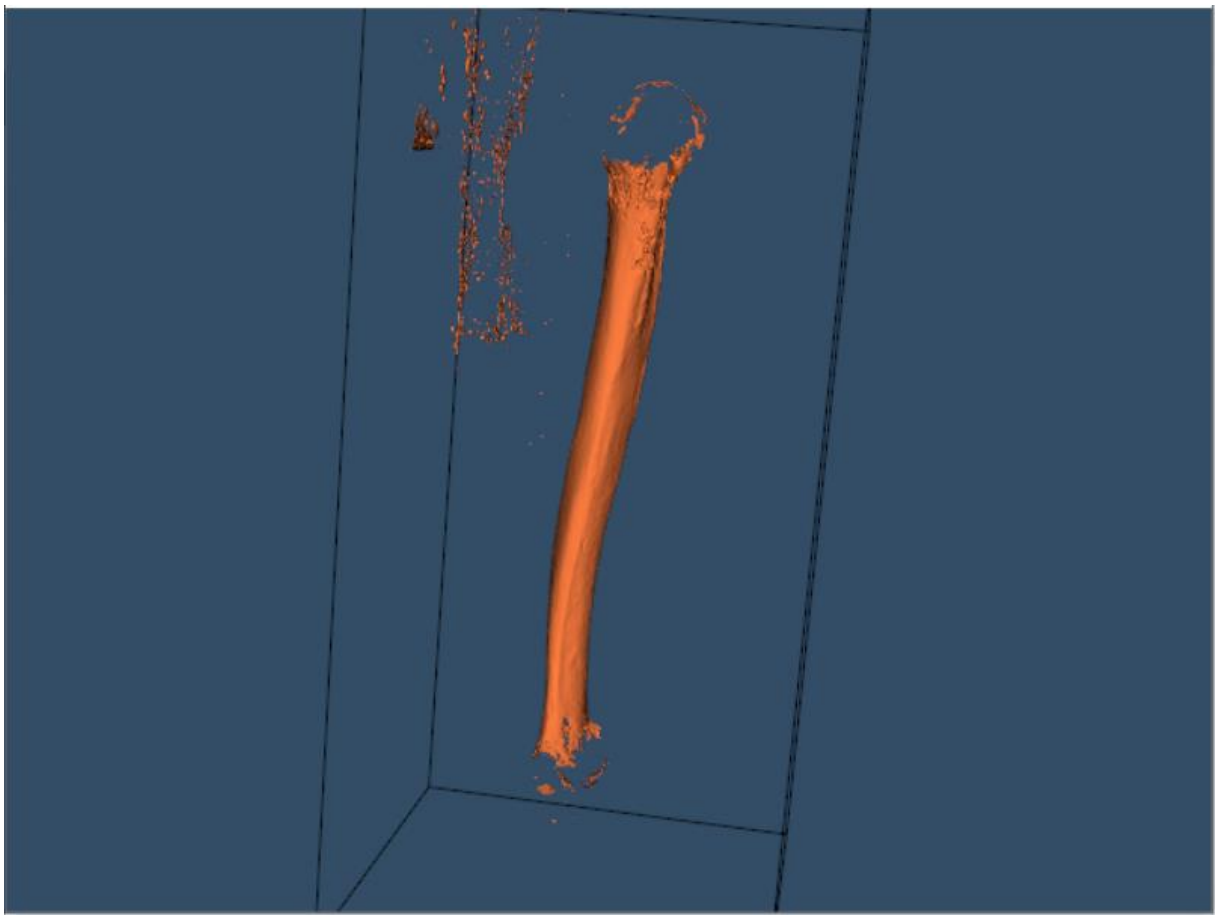


Ilustración 3 Resultados y Discusión: Ejemplo 1.1

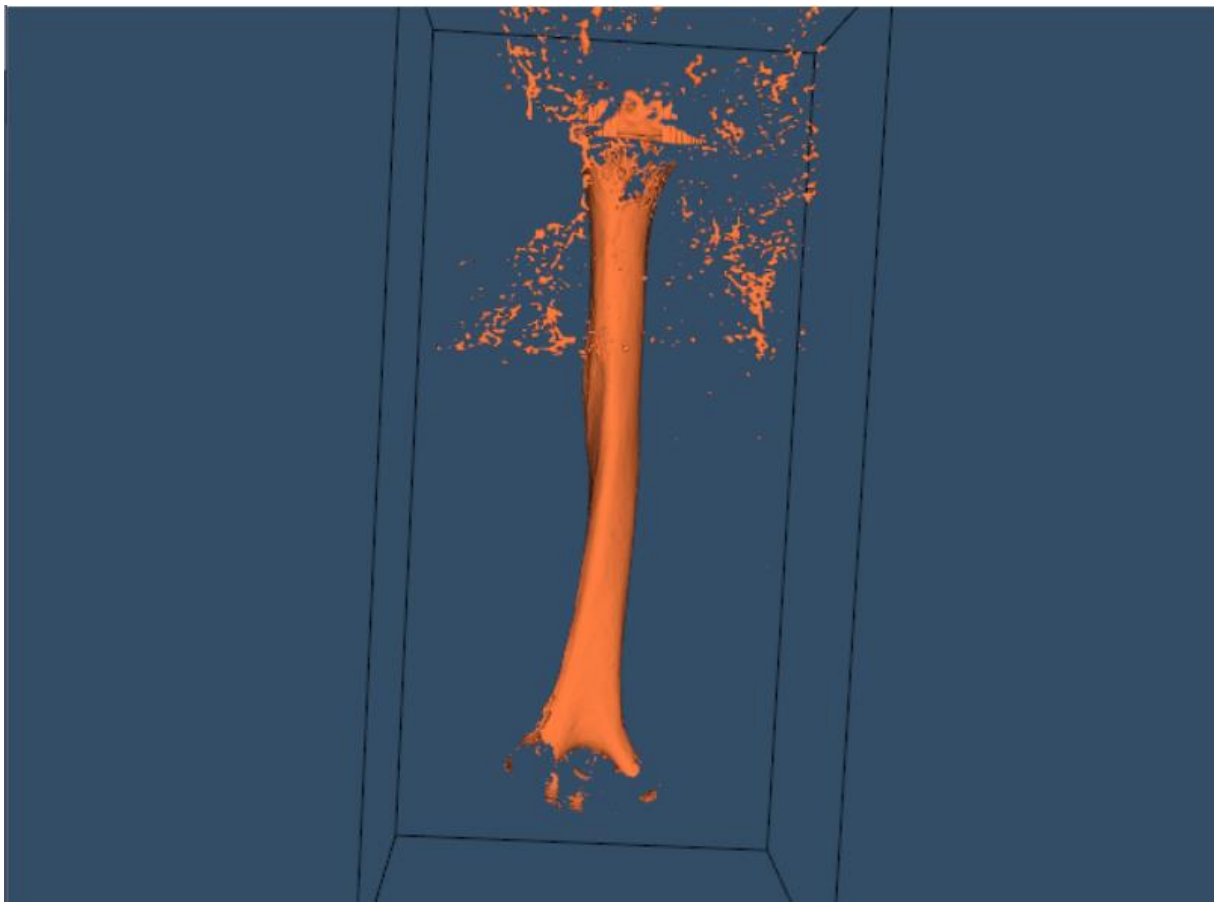


Ilustración 3 Resultados y Discusión: Ejemplo 1.2

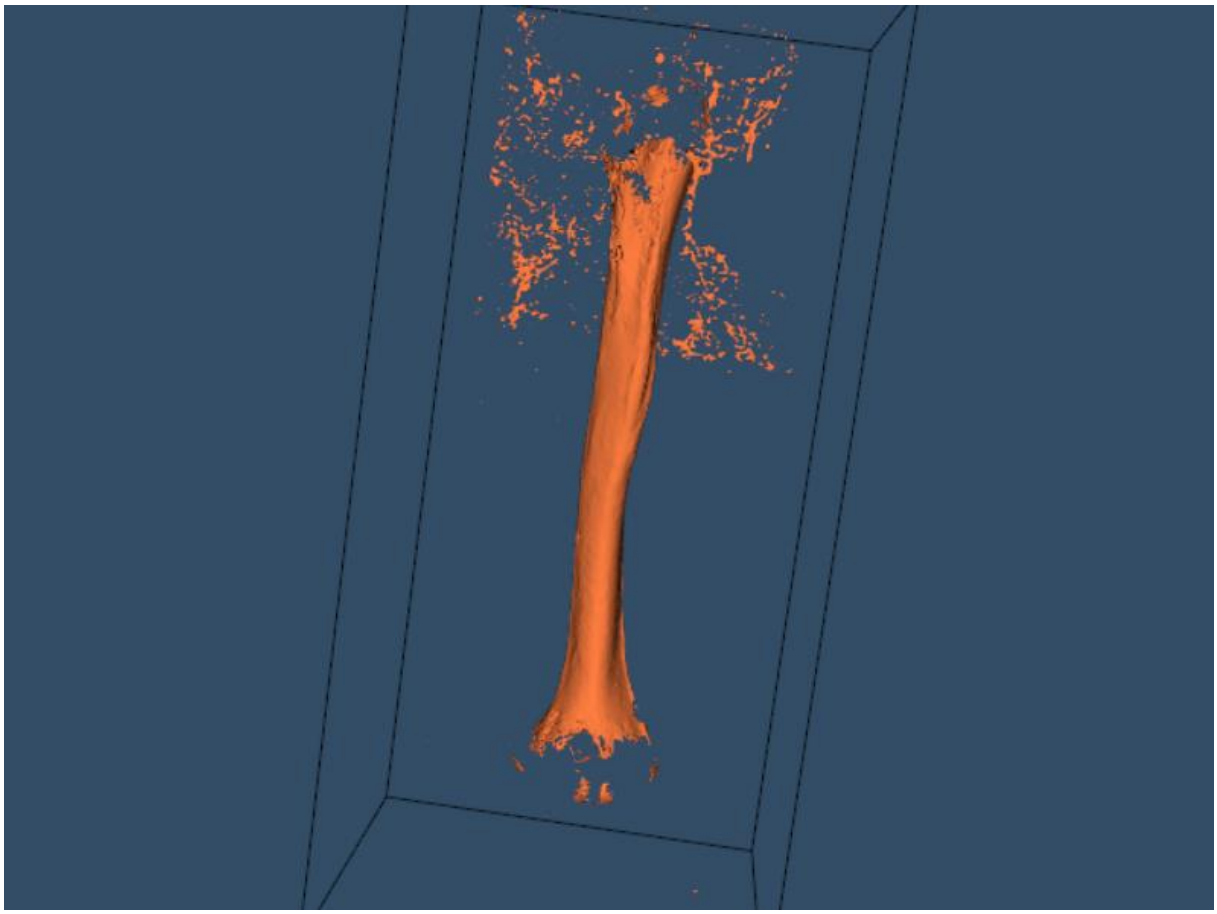


Ilustración 3 Resultados y Discusión: Ejemplo 1.3

Uno de los fallos que podemos detectar es que las partes cercanas a las articulaciones no están bien formadas y muestran mucho ruido, esto se debe a que el algoritmo de umbralización no es capaz de segmentar estas partes de forma correcta, si quisiéramos un mejor resultado en estas partes, tendríamos que probar a usar otro algoritmo. El ruido de fondo forma parte del TAC, el algoritmo de umbralización no es capaz de eliminarlo, pero es algo que ocurre frecuentemente con casi cualquier algoritmo, por eso, es necesario limpiar el modelo con otras herramientas.

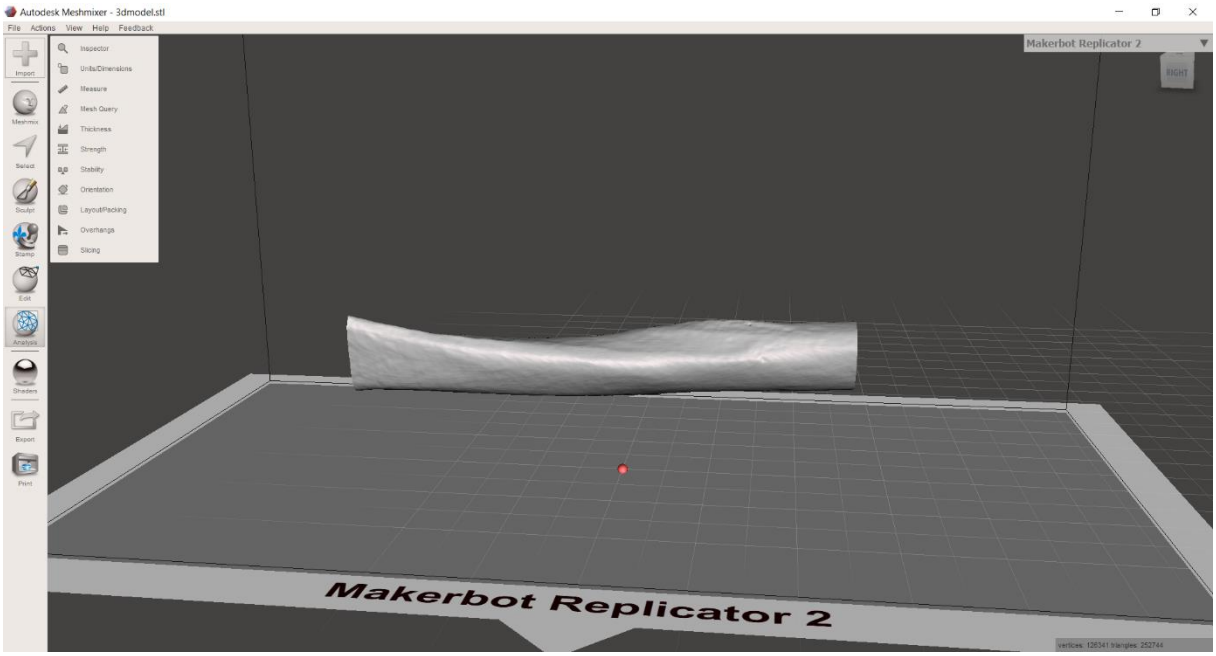


Ilustración 3 Resultados y Discusión: Ejemplo 1.4

Este ya sería un resultado limpio e imprimible.

Ahora veamos un ejemplo de un hueso blando, este tipo de hueso puede ser segmentado de forma más eficaz debido a que, en las imágenes, se ven más blancos, por tanto, se pueden diferenciar mejor las partes que son hueso y las que no.

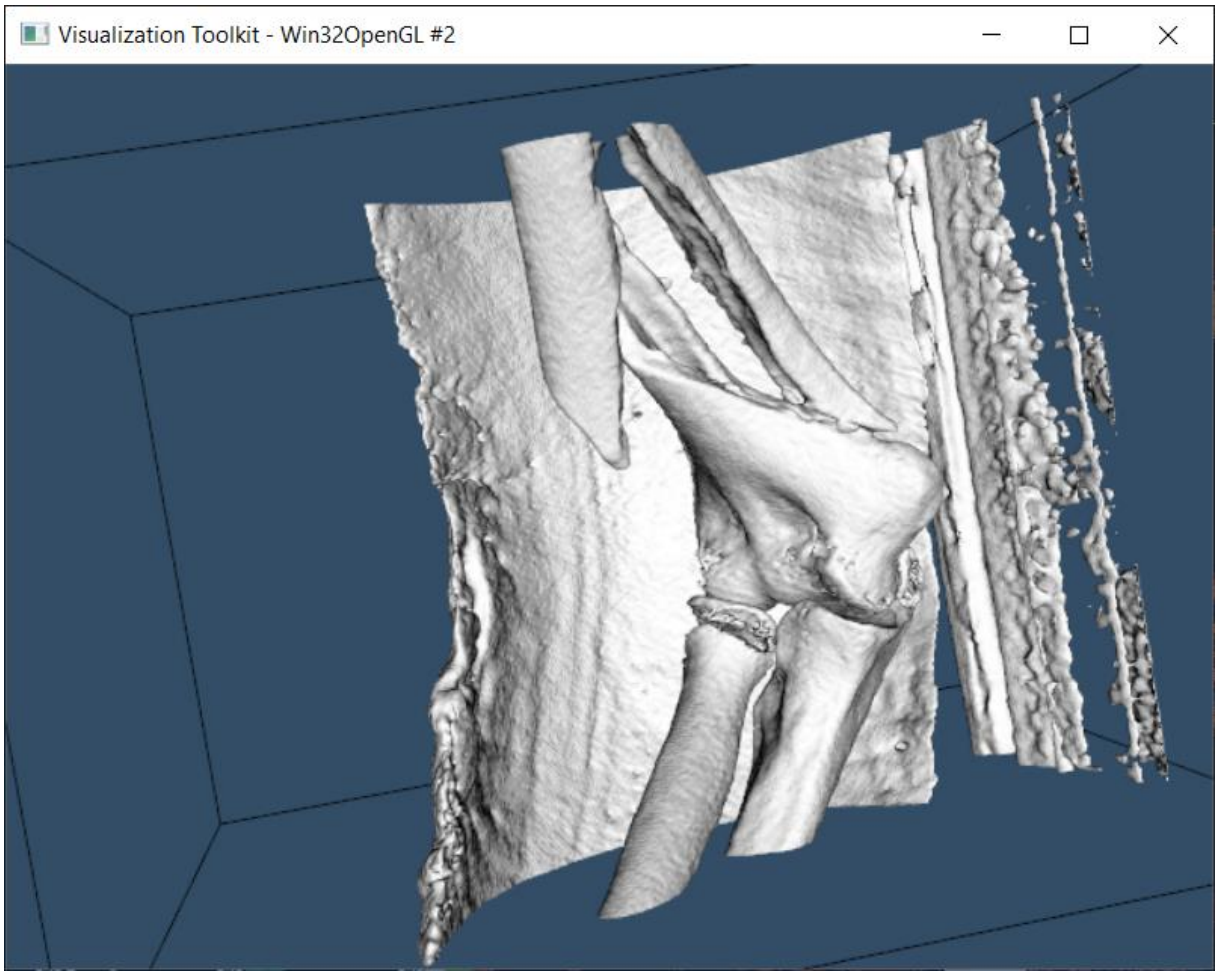


Ilustración 3 Resultados y Discusión: Ejemplo 2.1

Como podemos ver, se trata de un húmero roto y lo que es el hueso, se segmenta y se genera sin ruido. Por la parte de detrás del modelo se sigue generando ruido debido al TAC, si lo limpiamos queda de la siguiente manera.

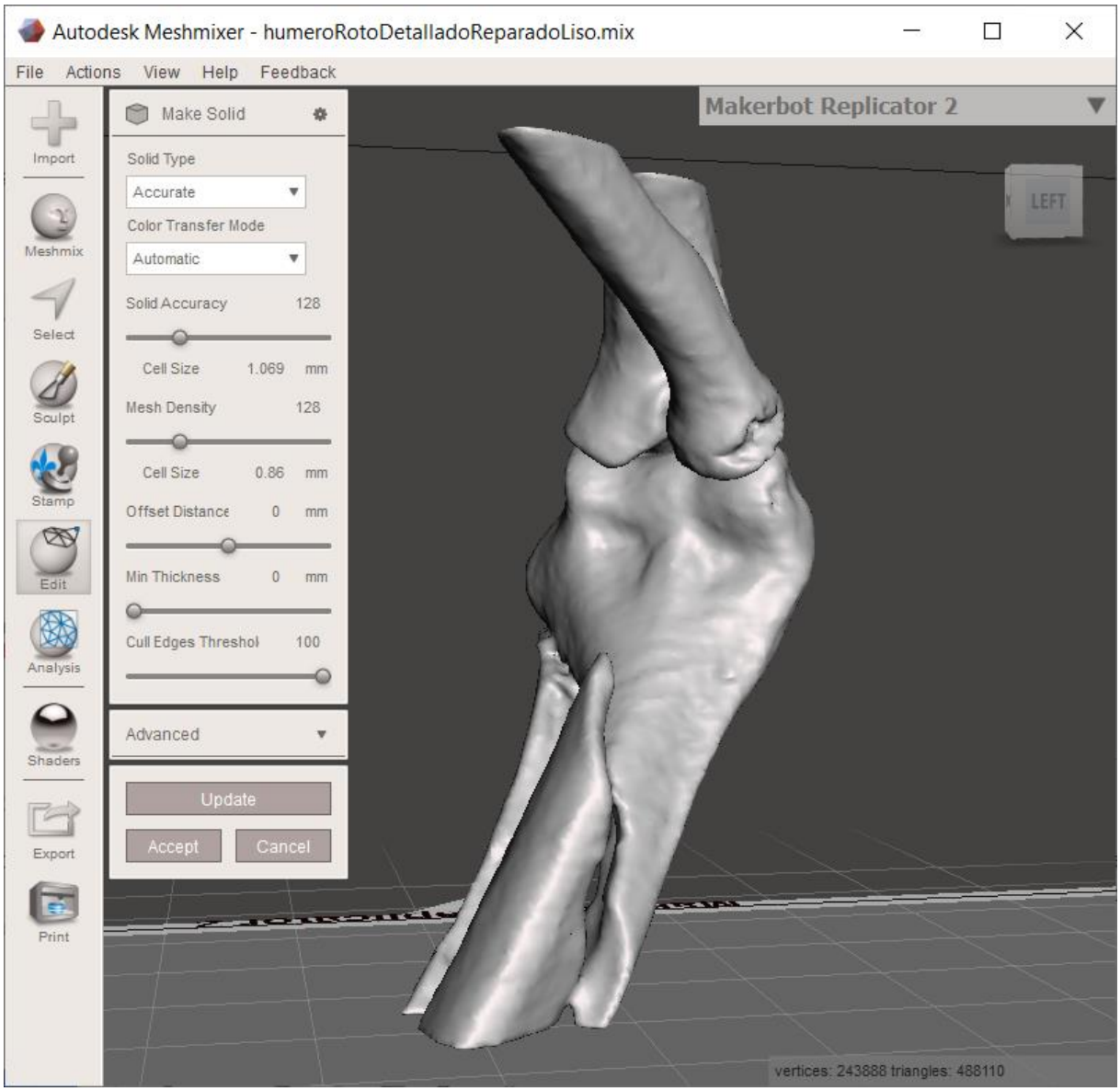


Ilustración 3 Resultados y Discusión: Ejemplo 2.2

Como podemos ver, el resultado es altamente satisfactorio.

Ya que el usuario también es capaz de ver los slices generados, podemos comprobar los resultados.

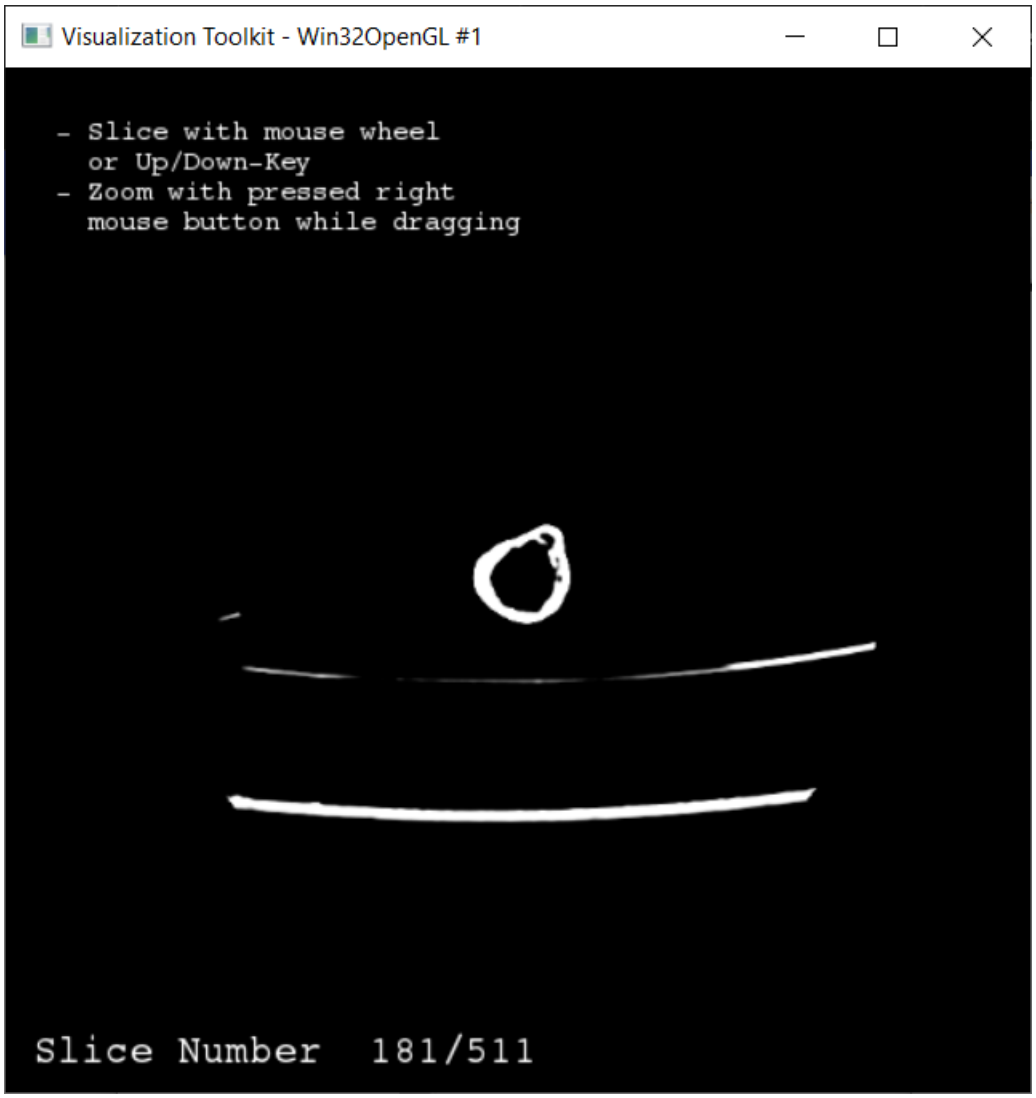


Ilustración 3 Resultados y Discusión: Ejemplo 4.1

Este slice corresponde al del primer húmero que hemos visto, el círculo que hay en medio es el que corresponde el hueso, las líneas de abajo puedo deducir que serán del material donde se ha hecho el TAC o algo referente al paciente como puede ser piel, músculo…

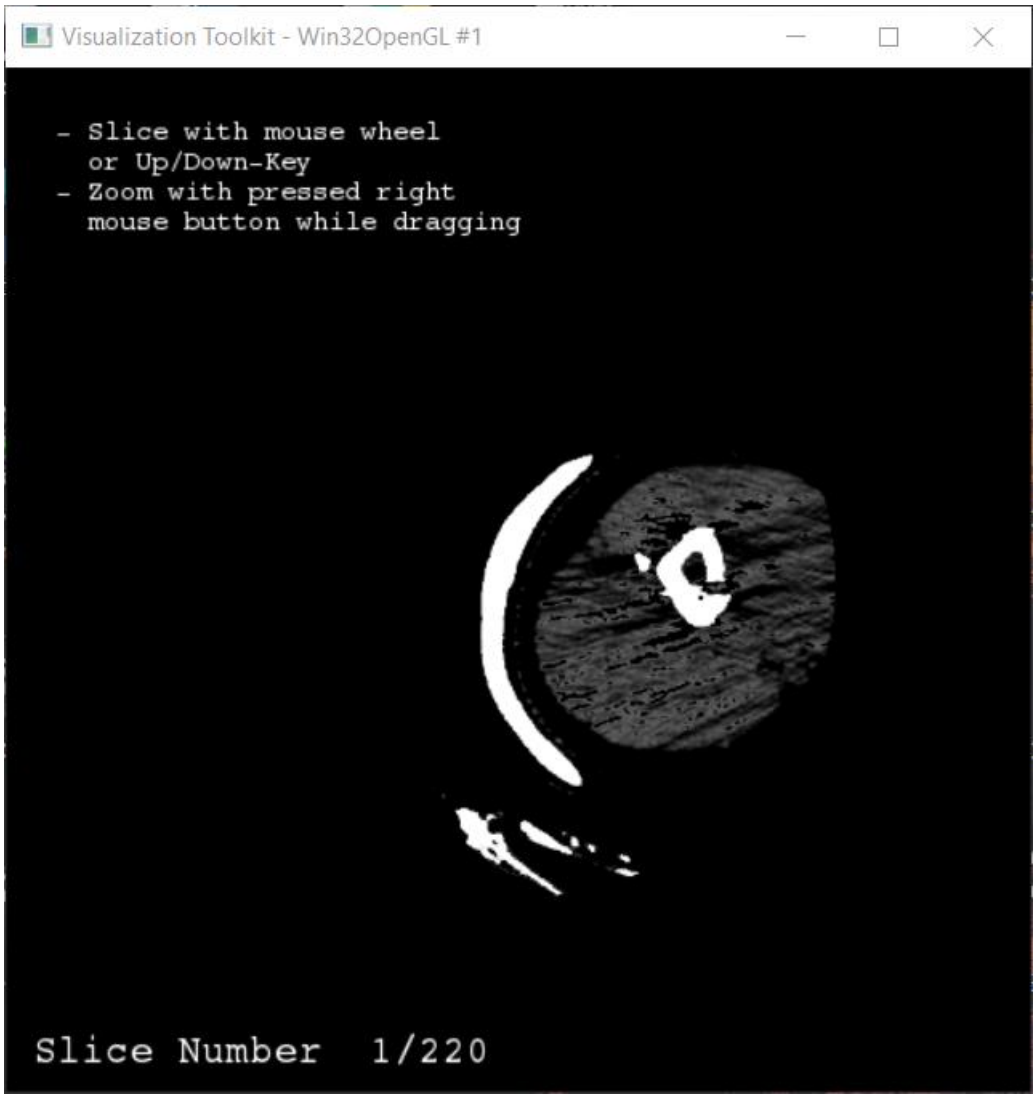


Ilustración 3 Resultados y Discusión: Ejemplo 5.1

Este slice corresponde al del húmero roto, el hueso sería el círculo que aparece entre lo gris. Segmentando de forma adecuada, podemos quitar este ruido en mayor o menor medida, lo recomendable es ir probando y ver el mejor resultado para cada ejemplo.

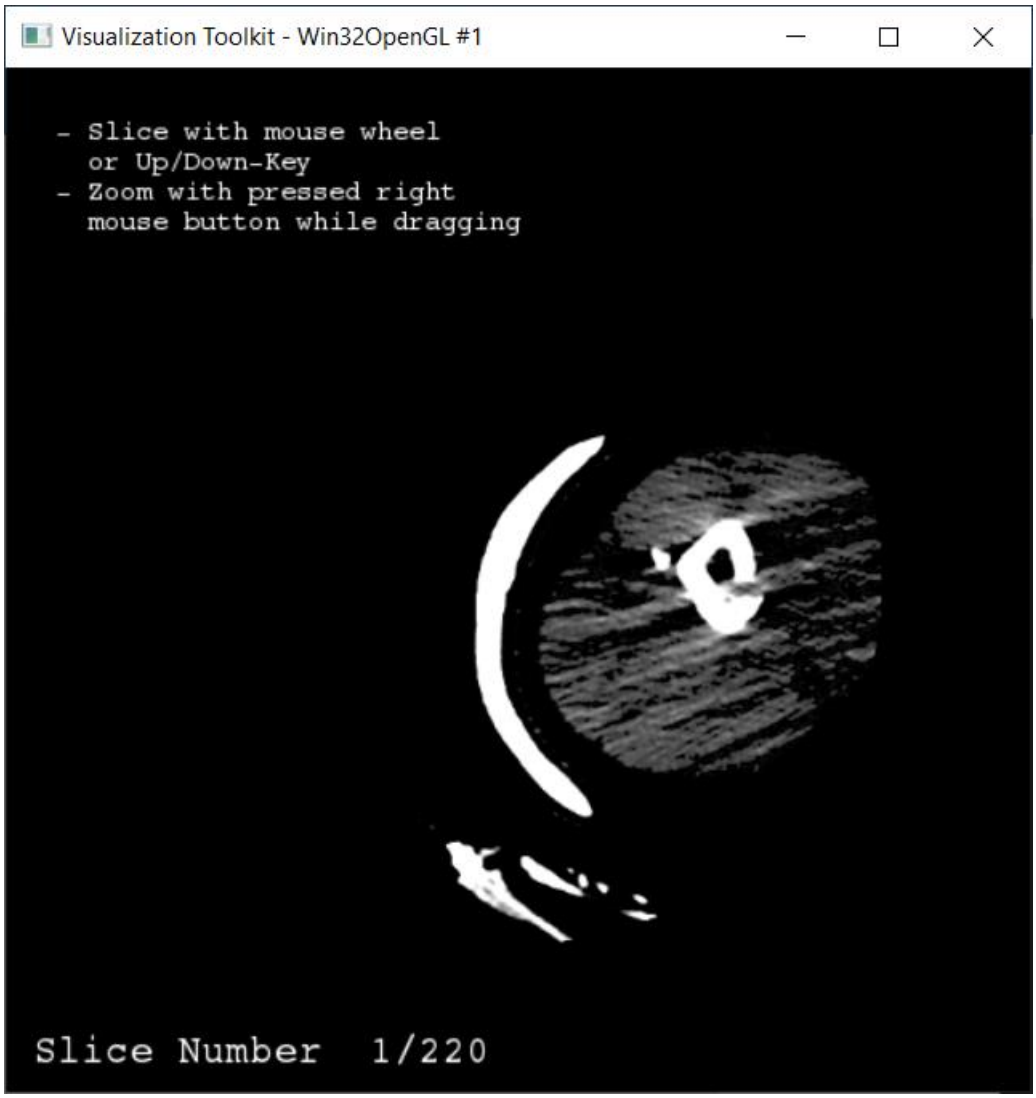


Ilustración 3 Resultados y Discusión: Ejemplo 5.2

Este es el mismo slice, pero con distinto rango de umbralización, si nos fijamos bien, podemos ver que cambian.

Vuelvo a comentar el tema anterior, es muy importante tener en cuenta que distintos ejemplos necesitan de distintos parámetros e incluso de distintos algoritmos de segmentación para poder obtener el resultado deseado, de ahí la importancia del análisis sobre los métodos realizado en el apartado 1.6 - A2: Análisis y Diseño del sistema.

En conclusión, para los ejemplos en los que he podido probar la aplicación, es recomendable su uso en huesos blandos y cuando las articulaciones no se encuentren en los extremos, ya que, no es capaz de segmentarlas de forma apropiada.

4 - CONCLUSIONES Y TRABAJOS FUTUROS

Este proyecto es indicado para enseñanza y para uso profesional en ejemplos muy concretos. Sin duda, una gran mejora de este, sería la implementación de más métodos de segmentación, ya que, es la parte más importante del proyecto en cuanto a obtener un resultado deseado. Esto haría que el uso de la aplicación pudiera llegar a ser usada de forma profesional sin ningún tipo de problema.

Como propuesta de trabajo futuro, propongo la ampliación de este proyecto hacia su uso en realidad virtual. No he podido realizar la impresión 3D de ninguno de los modelos mostrados debido a la pandemia que actualmente, año 2020, estamos viviendo, pero puedo imaginarme que, debido a la complejidad de estos, puede ser complicado limpiar y preparar un modelo para su impresión. Por esto, como idea, recomiendo el uso de realidad virtual para la visualización de los modelos, de esta forma, se podrá ofrecer al usuario una muy buena experiencia y a la vez aminorar el trabajo de limpieza de los modelos y eliminar el costo de impresión.

5 - APÉNDICES

5.1 - Instalación y configuración del sistema

4. Instalación de VTK:

   La instalación de VTK es necesaria para que el proyecto pueda recoger los módulos que necesita y usarlos. Previamente necesitaremos instalar CMake y Visual Studio, pero estos no necesitan de explicación ya que sus instalaciones son asistidas y casi automáticas.

   En primer lugar, accederemos a la página oficial de VTK para descargar el software.

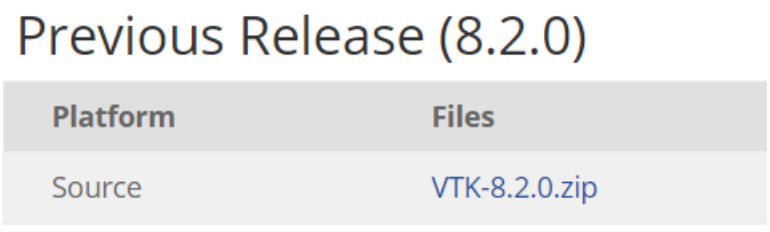


Ilustración 5.1 Instalación y configuración del sistema: Instalación VTK 1

Este será su contenido una vez descomprimido.

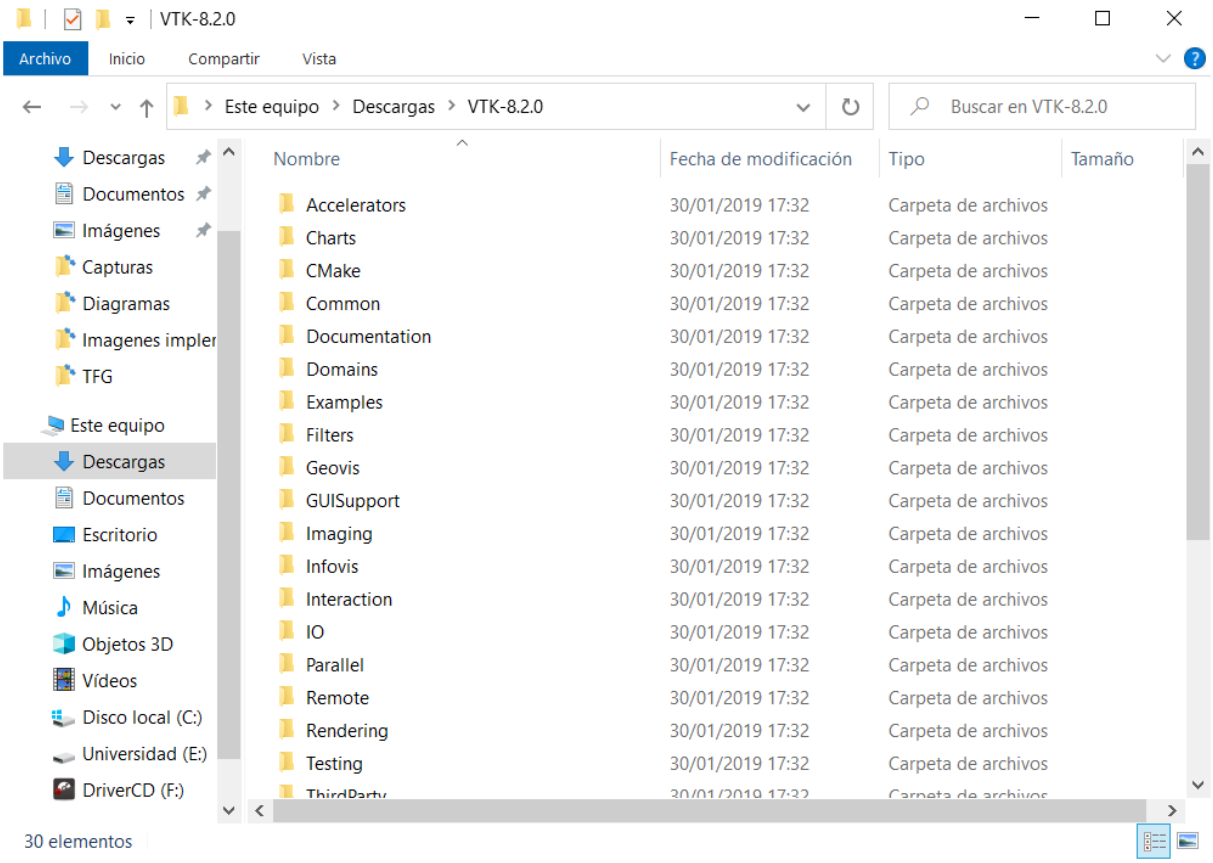


Ilustración 5.1 Instalación y configuración del sistema: Instalación VTK 2

Seguidamente, nos crearemos una carpeta con el siguiente contenido donde queramos, recomiendo crearlas en el disco C:.

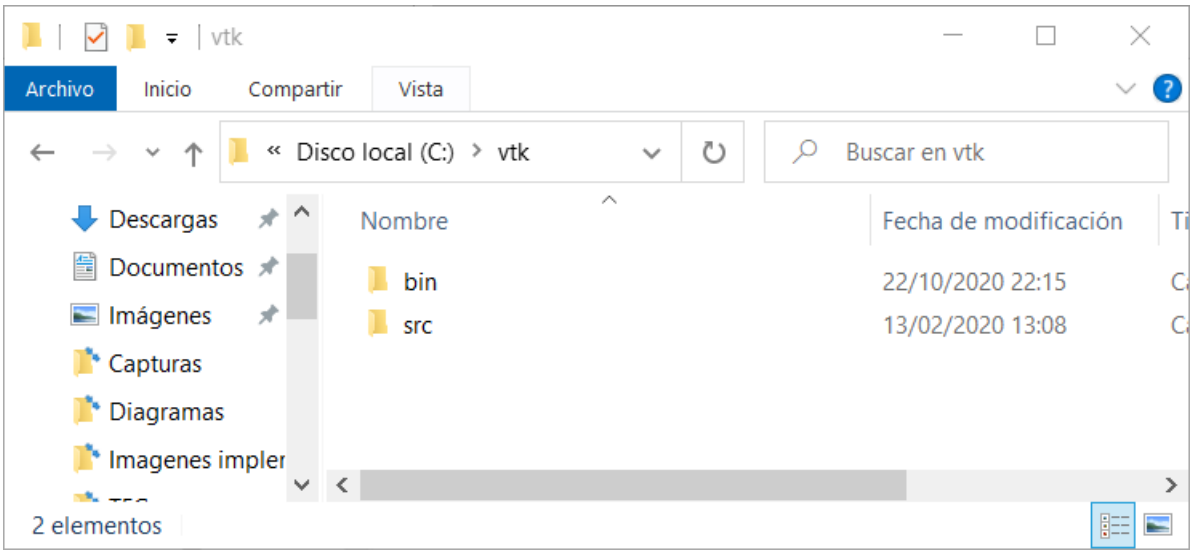


Ilustración 5.1 Instalación y configuración del sistema: Instalación VTK 3

Copiaremos el contenido del programa VTK en la carpeta llamada “src” que hemos creado.

Después, abriremos CMake y en la parte de arriba, indicaremos las rutas de las carpetas que hemos creado.

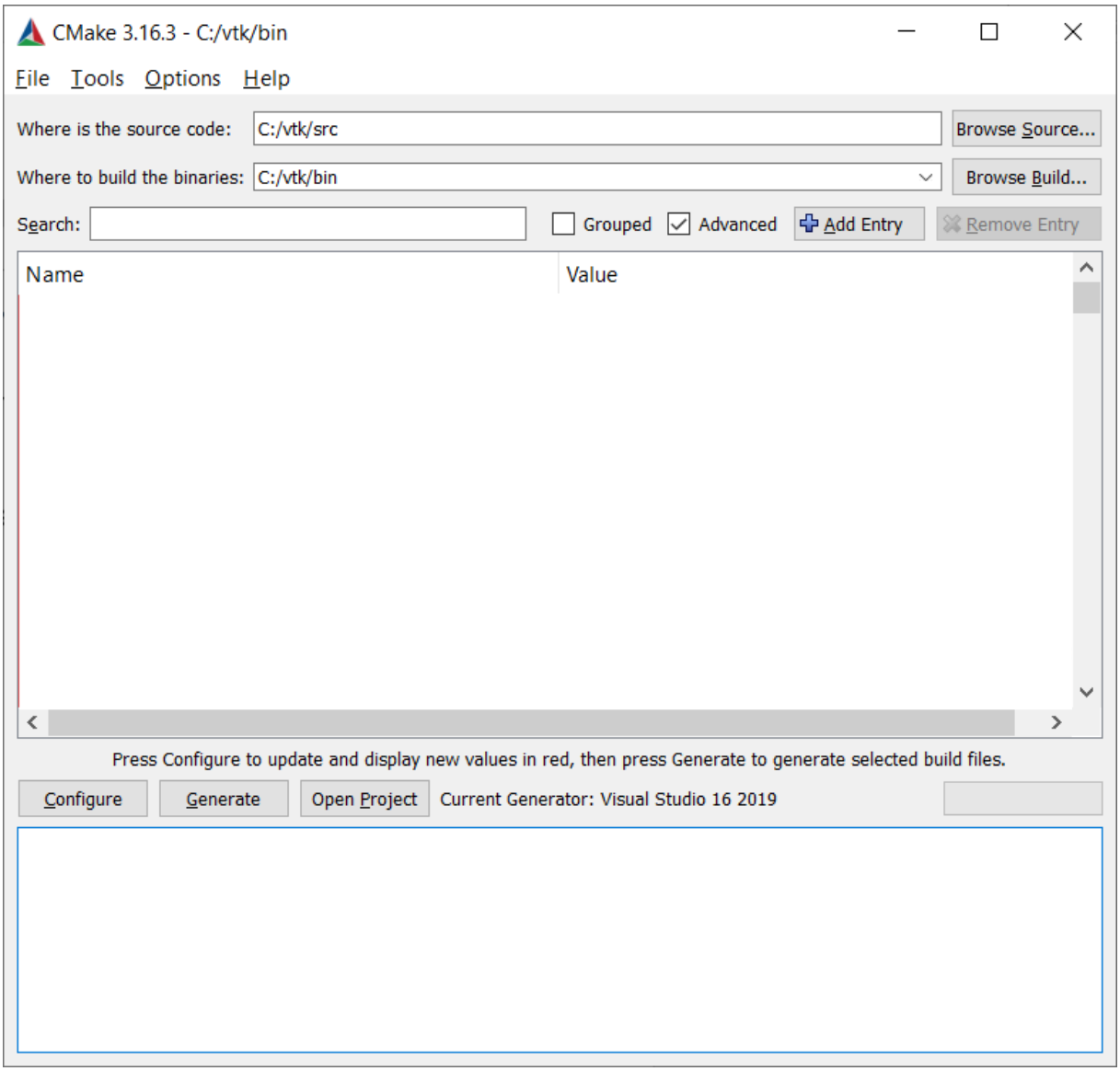


Ilustración 5.1 Instalación y configuración del sistema: Instalación VTK 4

Pulsaremos en “Configure”, tendremos que marcar el IDE al que va destinado y si nuestro sistema es de 64 bits, aceptamos y una vez que termine el proceso, nos aparecerá algo así.

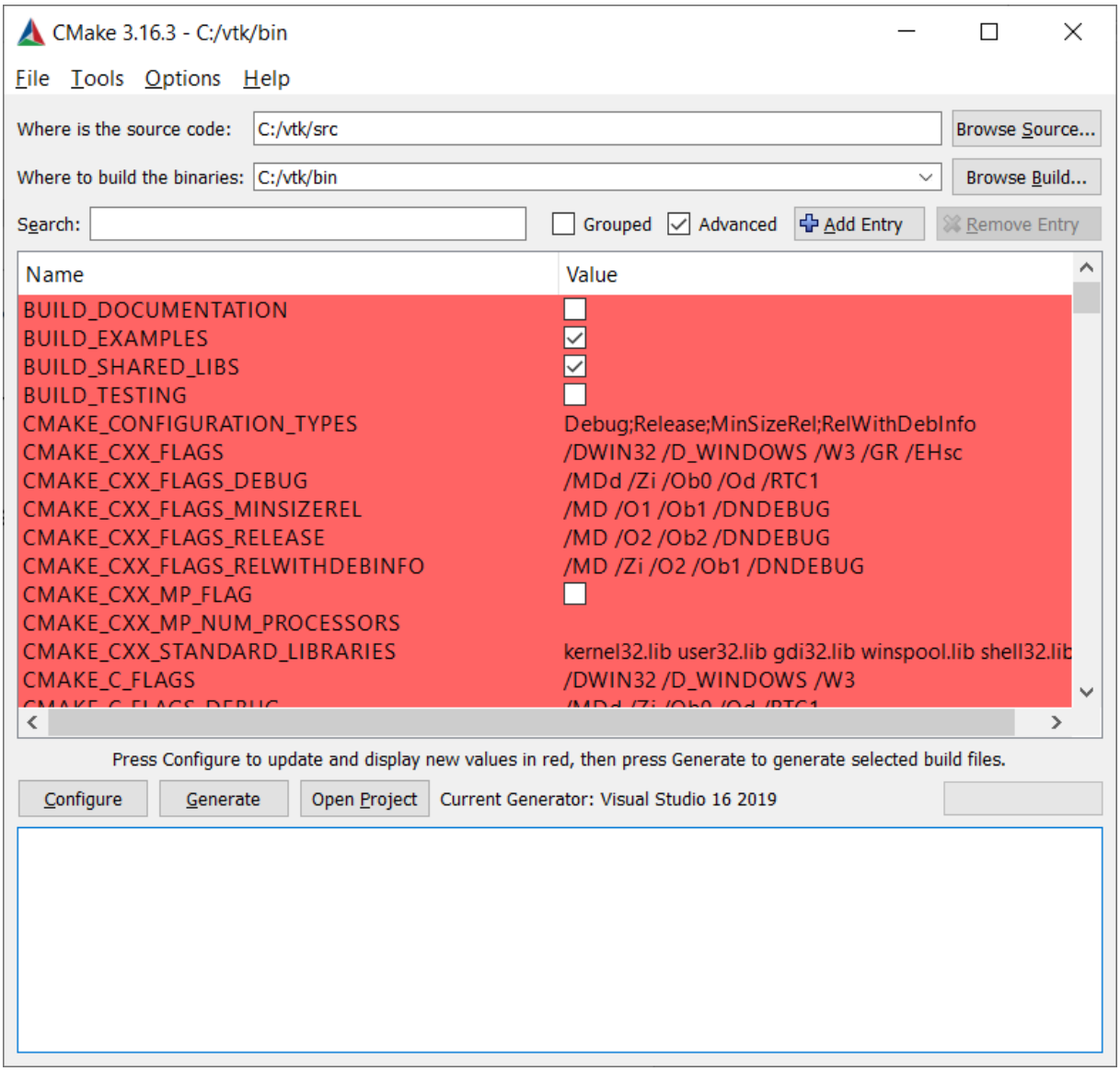


Ilustración 5.1 Instalación y configuración del sistema: Instalación VTK 5

Tenemos que asegurarnos de marcar las opciones “BUILD_EXAMPLES”, “Module_vtkTestingCore” y “Module_vtkTestingRendering”. Pulsaremos en “Configure” de nuevo y cuando termine el proceso pulsaremos en “Generate”.

Con esto, hemos conseguido que se nos genere código para abrir en Visual Studio. Abrimos la carpeta bin que hemos creado anteriormente y nos aparecerá algo así.

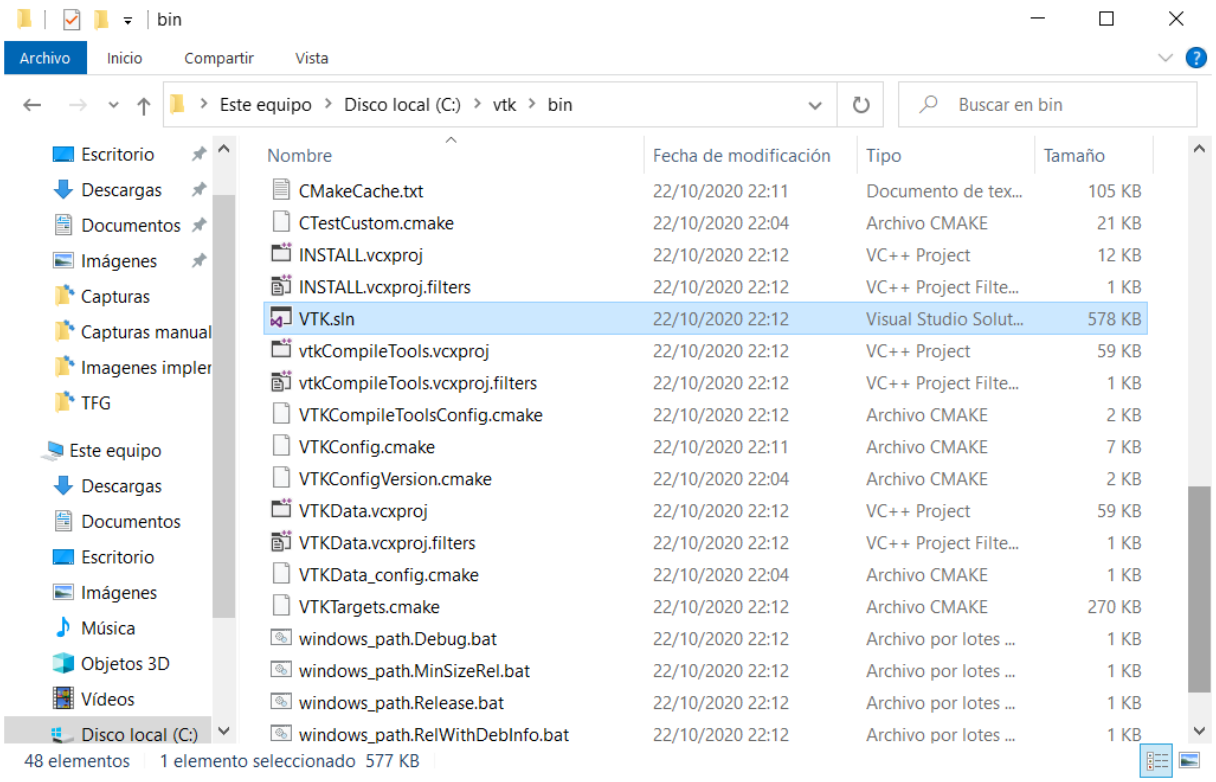


Ilustración 5.1 Instalación y configuración del sistema: Instalación VTK 6

Abriremos la solución que aparece marcada en la imagen con Visual Studio, compilamos donde pone “ALL_BUILD” y luego donde pone “INSTALL”.

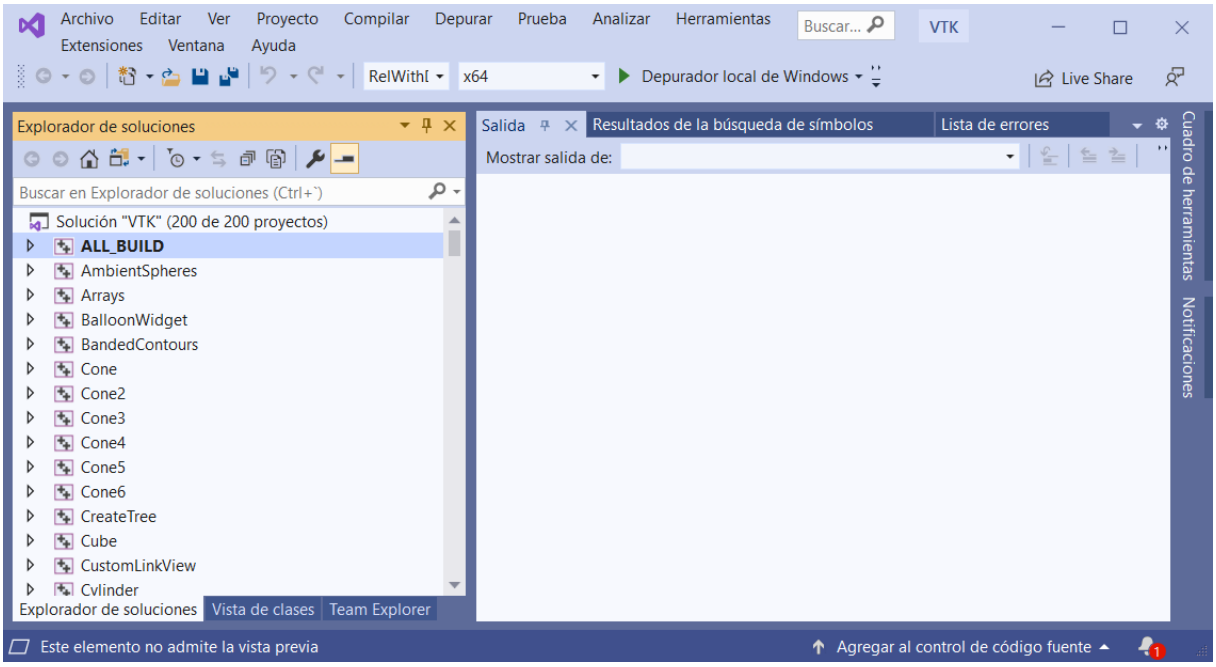


Ilustración 5.1 Instalación y configuración del sistema: Instalación VTK 7

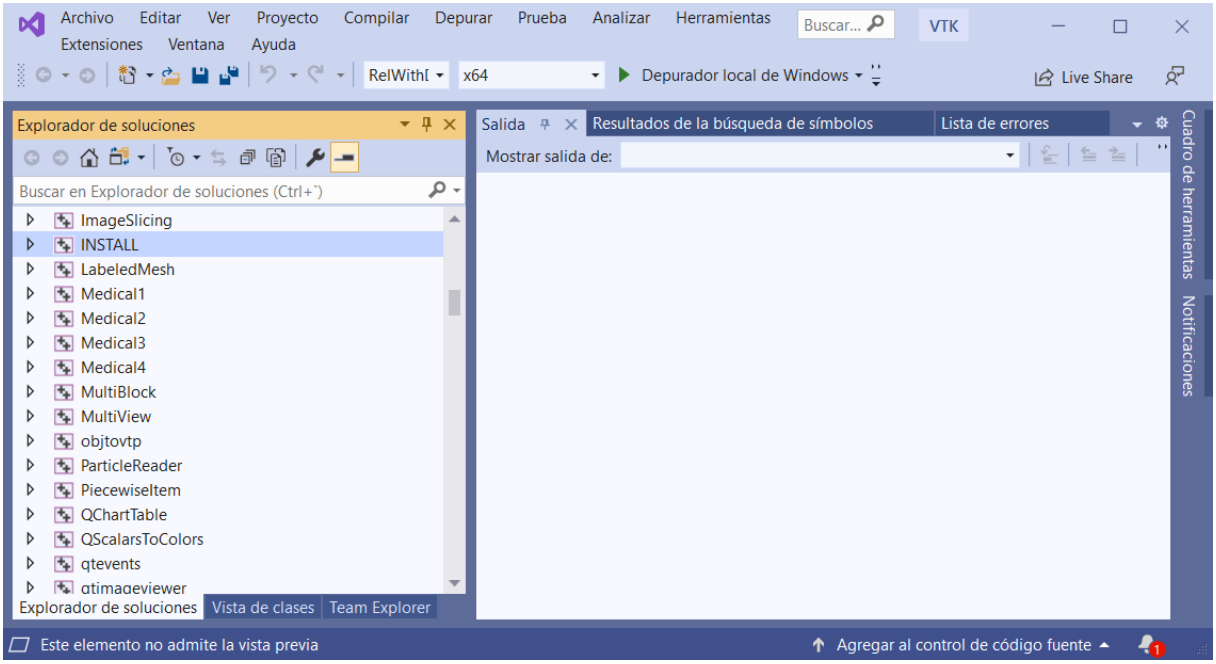


Ilustración 5.1 Instalación y configuración del sistema: Instalación VTK 8

Para comprobar que se ha instalado correctamente, accedemos a nuestro Disco loca > Archivos de programa (x86) y comprobamos que existe la carpeta VTK.

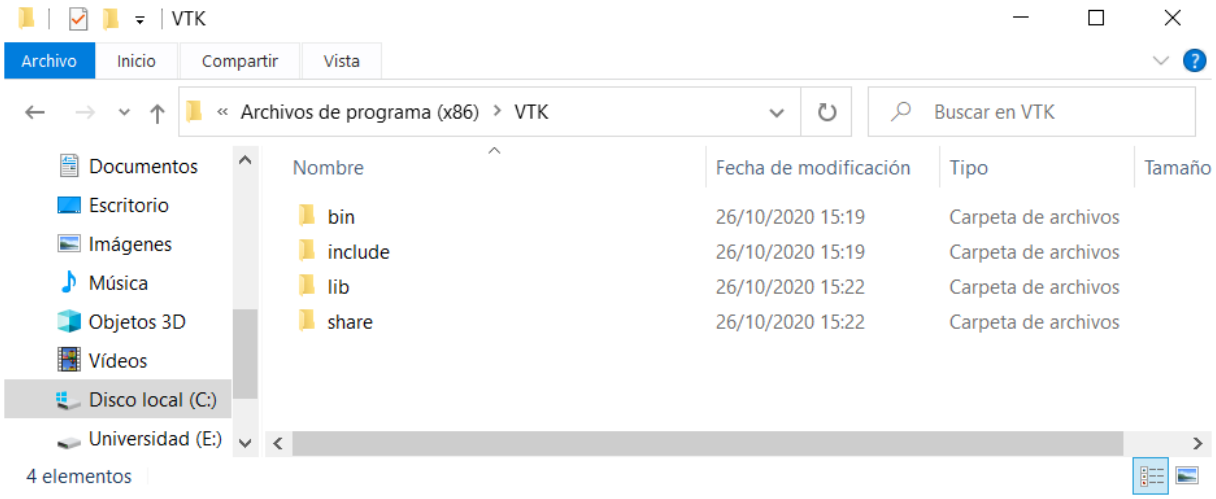


Ilustración 5.1 Instalación y configuración del sistema: Instalación VTK 9

Por último, para asegurarnos de que todo funcione correctamente, añadiremos el programa a las variables de entorno.

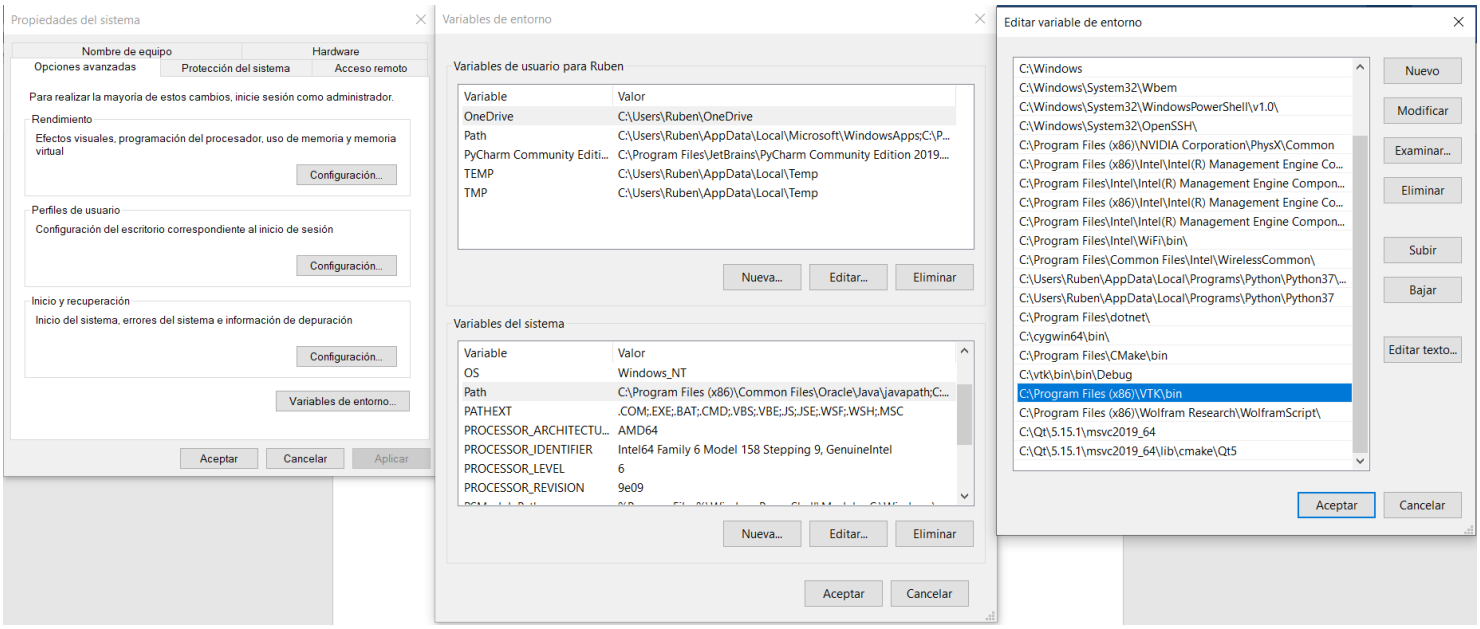


Ilustración 5.1 Instalación y configuración del sistema: Instalación VTK 10

5. Instalación de la aplicación:

La instalación de la aplicación no requiere de mucha explicación, solo hace falta descomprimir los archivos que están en la plataforma, la aplicación en cuestión es la carpeta llamada “TFG” y tiene el siguiente contenido.

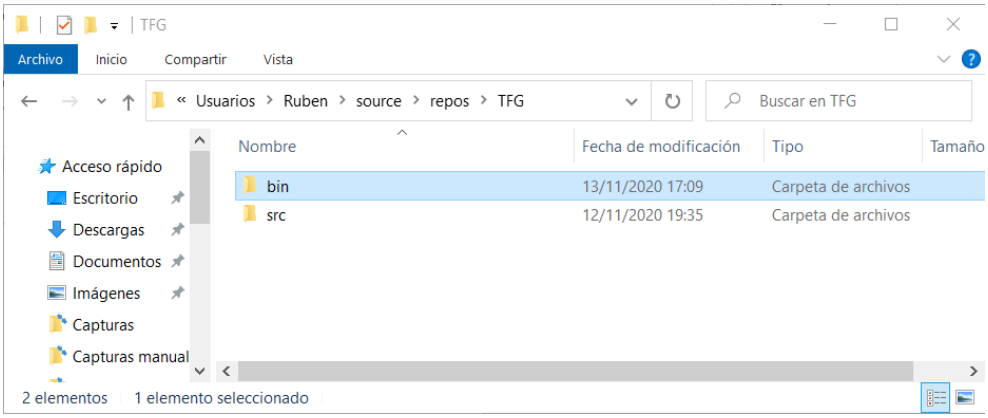


Ilustración 5.1 Instalación y configuración del sistema: Instalación de la aplicación 1

Si accedemos a las carpetas bin > Debug, encontraremos el ejecutable del programa “TFG.exe”.

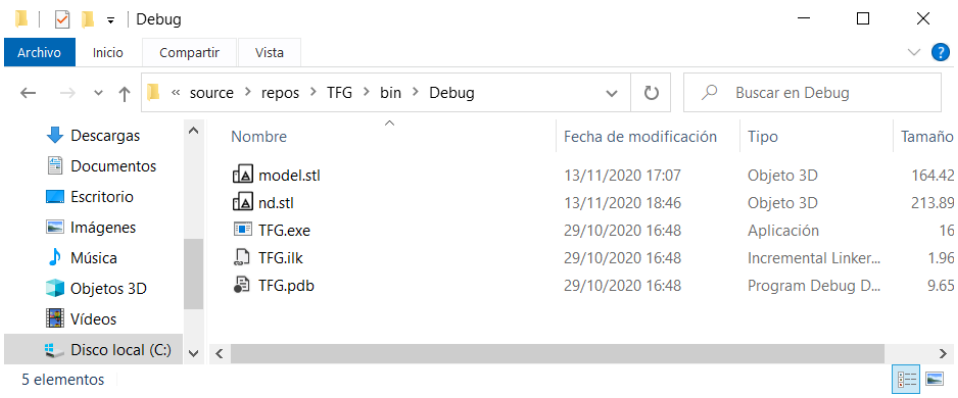


Ilustración 5.1 Instalación y configuración del sistema: Instalación de la aplicación 2

5.2 - Manuales de usuario

El uso de la aplicación es muy sencillo, solo hace falta abrir el CMD, acceder a la carpeta donde se encuentra el ejecutable y poner los parámetros de forma correcta.

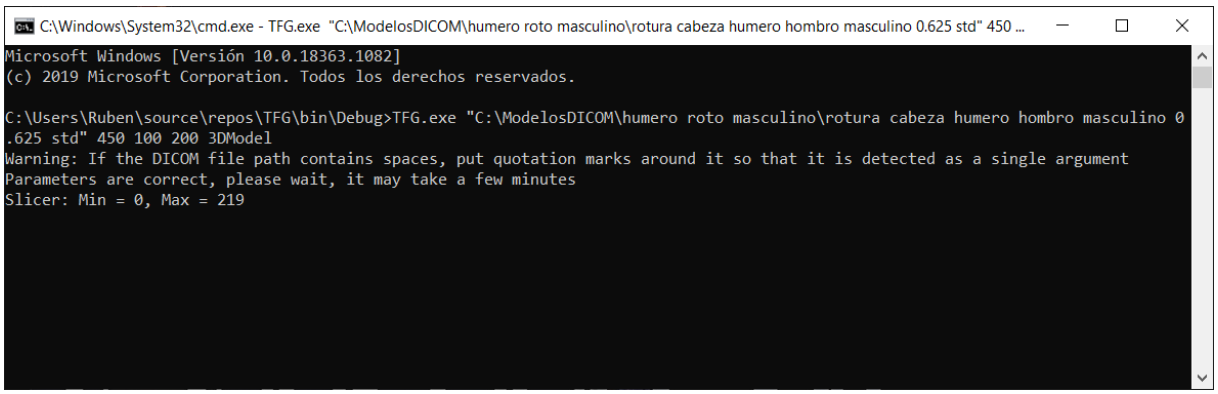


Ilustración 5.1 Instalación y configuración del sistema: Manual de usuario 1

Aquí tenemos un ejemplo, recordemos el significado de cada uno de los parámetros. El primero corresponde al ejecutable que se encuentra en la carpeta, el segundo a la ruta donde se encuentra el archivo DICOM, si contiene espacios, hay que poner la ruta entre comillas. El tercero corresponde al nivel de detalle del modelo, el cuarto y quinto a los valores de umbralización mínimo y máximo y el último al nombre que tomará el archivo STL que se creará automáticamente.

Recordemos que el nivel de detalle solo puede estar entre 200 y 500, personalmente recomiendo poner 450 para tener un modelo no tan pesado y con un nivel de detalle suficiente. Si se pusiera a 200 (nivel de detalle máximo), el programa tardaría mucho generar el modelo y este pesaría mucho, pero obtendríamos un modelo muy bien detallado. En cambio, si se pone en 500, el programa se ejecutaría

rápido y el archivo STL sería poco pesado, pero quizás el modelo no esté lo suficientemente detallado.

El rango de los valores de umbralización tienen que estar entre 0 y 255, mínimo no puede superar al máximo y el máximo no puede ser menor que el mínimo. Al nombre del modelo ya se le añade el formato automáticamente, por tanto, no hace falta poner ".stl" al final.

Una vez se pulse enter con los parámetros adecuados, nos aparecerá las lonchas o slices.

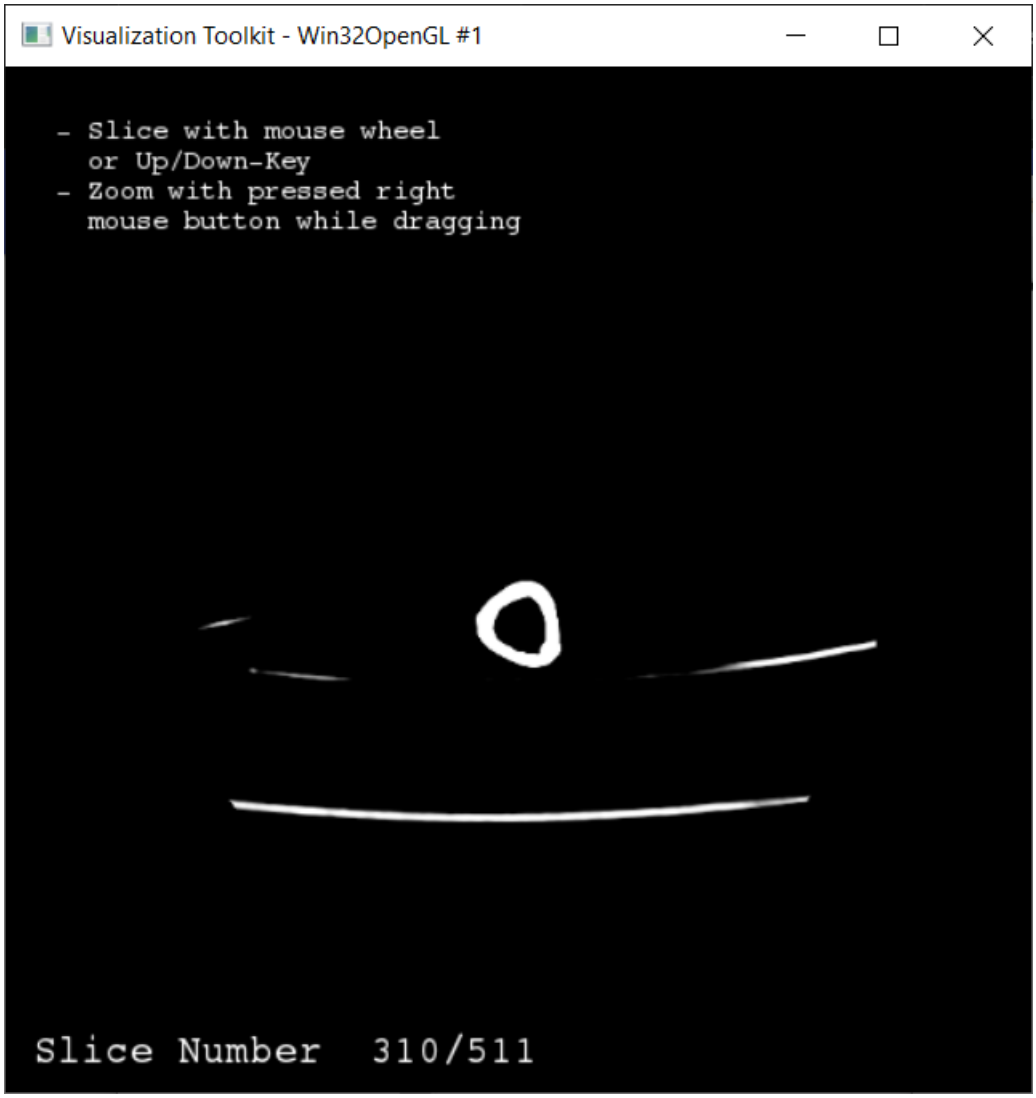


Ilustración 5.1 Instalación y configuración del sistema: Manual de usuario 2

Se podrá pasar de slice presionando las teclas arriba o abajo o bien moviendo la rueda del ratón hacia arriba o hacia abajo. Si se pulsa click derecho en la imagen y se mueve el ratón, se incrementará o decrementará el zoom de la imagen.

Cuando se cierre esta ventana, aparecerá automáticamente la ventana del modelo.

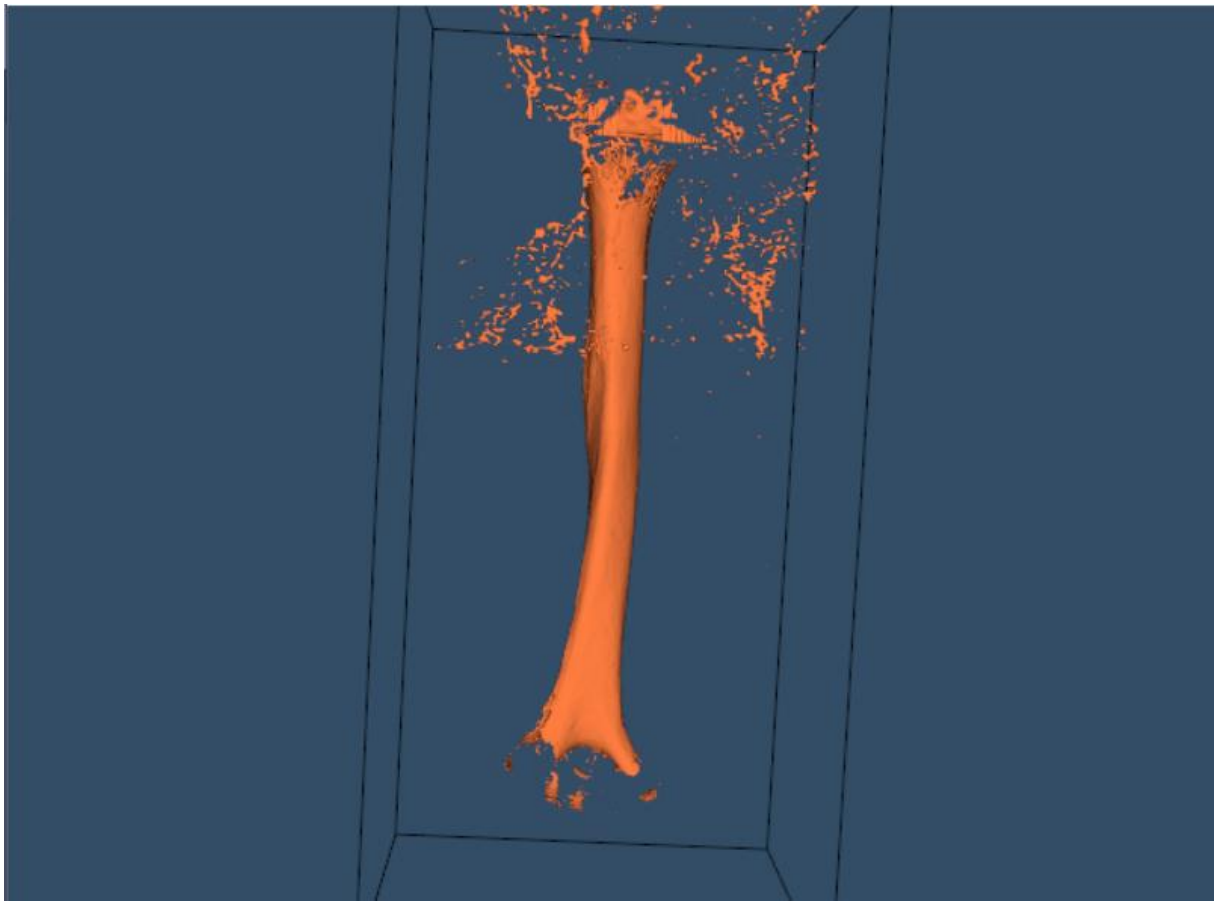


Ilustración 5.1 Instalación y configuración del sistema: Manual de usuario 3

Se podrá mover la cámara alrededor del modelo si pulsamos click izquierdo y movemos el ratón mientras se mantiene. También se podrá incrementar o decrementar el zoom moviendo la rueda del ratón.

Cuando esta última ventana sea cerrada, la ejecución del programa se detendrá y el archivo STL se habrá generado de forma automática en el mismo sitio donde se haya ejecutado la aplicación.

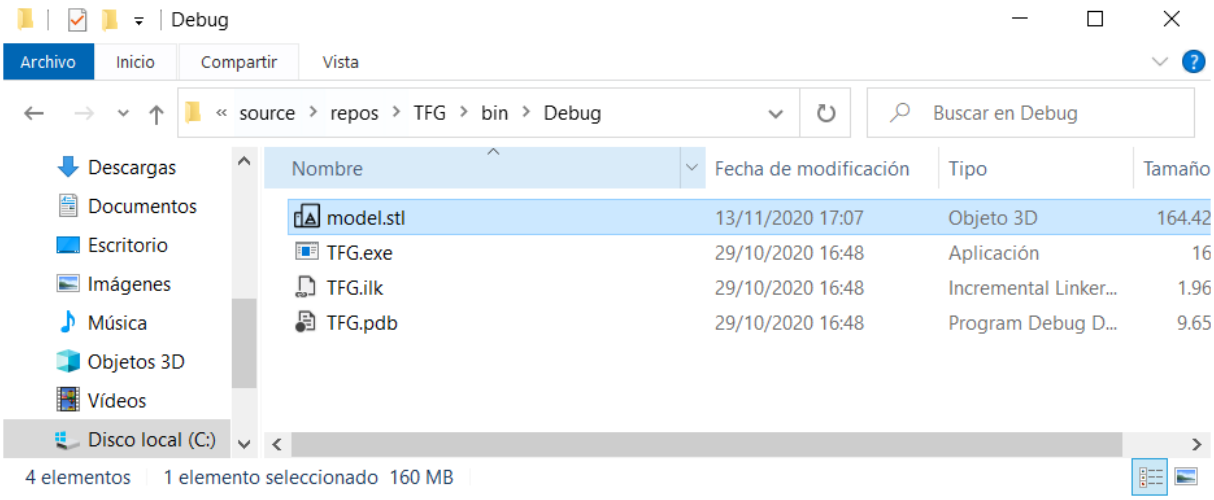


Ilustración 5.1 Instalación y configuración del sistema: Manual de usuario 4

5.3 - Guía original del Trabajo Fin de Título

Se puede acceder al documento pinchando en el siguiente enlace:

Guía original del Trabajo de Fin de Grado.

6 - BIBLIOGRAFÍA

[1] van Eijnatten, M., van Dijk, R., Dobbe, J., Streekstra, G., Koivisto, J., & Wolff, J. (2018). CT image segmentation methods for bone used in medical additive manufacturing. Medical Engineering and Physics, Vol. 51. https://doi.org/10.1016/j.medengphy.2017.10.008

[2] Lee, L. K., Liew, S. C., & Thong, W. J. (2015). A Review of Image Segmentation Methodologies in Medical Image. Advanced Computer and Communication Engineering Technology, 1069–1080. https://doi.org/10.1007/978-3-319-07674-4_99

[3] Francisco Javier Olías Sánchez, Jose Antonio Pérez Carrasco (2014). Segmentación en 3D de Huesos en Imágenes TAC. https://drive.google.com/file/d/0Bw_Bume5f9t5Y3ZOdFUwVm13TVNTTE9DdUxqbV9WNG1YOGNZ/view

[4] VTK https://vtk.org/doc/nightly/html/

[5] Marching Cubes http://www.cs.carleton.edu/cs_comps/0405/shape/marching_cubes.html#1