Universidad Central “Marta Abreu” de Las Villas
Facultad Matemática Física y Computación
Licenciatura en Ciencia de la Computación

TRABAJO DE DIPLOMA

Segmentación de la aorta a partir de Imágenes de Resonancia Magnética cardiovasculares 4D.

Autor: **Denis Deniz González**
Tutores: **Lic. Marcelino Rodríguez Cancio**
Lic. Gonzalo Ramón Nápoles Ruiz

“Año 54 de la Revolución”

Santa Clara
2012

DECLARACIÓN JURADA

El que suscribe, _____, hago constar que el trabajo titulado _____ fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de los estudios de la especialidad de _____, autorizando a que el mismo sea utilizado por la institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos ni publicado sin la autorización de la Universidad.

_____

Firma del autor

Los abajo firmantes, certificamos que el presente trabajo ha sido realizado según acuerdos de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del tutor Firma del jefe del Laboratorio

_____

Fecha

AGRADECIMIENTOS

Le estoy enteramente agradecido:

- Principalmente a toda mi familia; en particular a mis padres, pues debo a ellos mi vida y sencillamente todo lo que soy.
- A todos mis amigos, ya sean de siempre o de solo un instante, por haber estado allí para mí.
- A cada uno de los profesores que tuve el privilegio de tener en todos estos años de estudio, que siempre dejan en nosotros parte de su conocimiento y espíritu. Particularmente, a mis tutores por su comprensión y paciencia.
- En general, a todas las personas que he conocido; a las que me han dejado, un buen ejemplo a seguir, o incluso sus defectos para nunca agregarlos a los tantos con los que contamos.

A todos, simplemente, **gracias**.

DEDICATORIA

A mi “herma”, mi papá y, en espacial, a la memoria de mi mamá.

"Una cosa que he aprendido en una larga vida
es que toda nuestra ciencia, medida contra la realidad,
es primitiva e infantil y aun así es
la cosa más preciosa que tenemos."

Albert Einstein

RESUMEN

Este trabajo propone un método de segmentación automática de la aorta a partir de imágenes 4D (tres dimensiones espaciales unidas a la temporal) y un procedimiento para la extracción de la línea central de la misma. Con este propósito fundamental, se diseña e implementa una tubería de segmentación específica en la que se aplican varios filtros o procedimientos de manera secuencial. Con el objetivo de incorporar posteriormente una mayor información visual acerca del comportamiento hemodinámico, se analizan las diferentes formas de representación del flujo sanguíneo. El procedimiento empleado permite que el experto en medicina pudiera manejar características dinámicas del flujo sanguíneo en el diagnóstico de enfermedades relacionadas con dicha arteria, y no basar su criterio en el tradicional análisis de las dimensiones de la aorta, pues este parámetro no resulta determinante ni robusto.

ABSTRACT

This work propose a method of aorta automatic segmentation from 4D images (three spatial dimensions joined to the temporal) and a procedure for the centerline extraction of the same. For this fundamental object, is design and implement a specific segmentation pipeline in that several filters or procedures are applied of sequential way. With the target of subsequently incorporate major visual information about the hemodynamic behavior, are analyzed different methods of blood flow representation. The procedure used allows that medicine expert may handle dynamic characteristics of the blood flow in the diagnosis of diseases pertaining to this artery, and not base his criteria on the traditional analysis of the aorta dimensions, since this parameter is not determinant or robust.

TABLA DE CONTENIDO

INTRODUCCIÓN

Planteamiento del problema

Durante las últimas décadas las Imágenes de Resonancias Magnéticas (*MRI* por sus siglas en inglés) han contribuido sustancialmente al diagnóstico de diferentes enfermedades. El empleo de las *MRI* constituye una técnica adaptada al estudio del flujo de la sangre por la aorta; en particular, resultan de gran utilidad en la detección temprana de desórdenes congénitos en los tejidos conjuntivos. Tradicionalmente, el máximo diámetro de la aorta es el principal parámetro que permite la evaluación del riesgo de ruptura o disección de dicha arteria, pero sucede que este parámetro no resulta determinante ni robusto (Xavier, Lalande et al. 2007). Por lo que resulta necesario manejar otros criterios que permitan un análisis más exhaustivo y confiable.

Una *MRI* cardiovascular 4D (3 dimensiones espaciales unidas al tiempo) no es más que una secuencia de imágenes tridimensionales a lo largo de la cual el ciclo cardíaco cumple una importante función informativa acerca del movimiento aórtico. Se presenta entonces la oportunidad de conocer acerca de las variaciones del flujo de la sangre a través de la arteria.

Sin embargo, lograr una segmentación de la aorta a partir de una MRI cardiovascular 4D no es una tarea trivial. Si se intenta abordar esta tarea manualmente, obtener trazados reproducibles de la aorta en una imagen *n*-fase 4D requiere conocimiento experto. Además, resulta un gran desafío para el especialista, consume un tiempo considerable y es realmente una tarea tediosa. Por tanto, surge la necesidad de desarrollar un algoritmo de segmentación automático de la aorta.

Problema de investigación

Realizar una aplicación para la segmentación automática de la aorta y la extracción de la línea central a partir de *MRI* 4D.

Objetivo general

Desarrollar una aplicación para la segmentación de la aorta a partir de *MRI* cardiovasculares 4D y el cálculo de la línea central de la misma, de manera que se

obtenga automáticamente y que sea viable brindar información sobre las variaciones del flujo sanguíneo a través de la aorta, implementándola en un ambiente de trabajo que permita integrar las diferentes herramientas de software, y con aplicación en cualquier centro que realice estos estudios médicos o cuente con una base de casos conformada por esta clase de imágenes.

Objetivos Específicos

- Implementar un algoritmo que determine la línea central de la aorta.
- Desarrollar un algoritmo automático para la segmentación de la aorta a partir de imágenes *MRI*.
- Desarrollar una interfaz gráfica que brinde información al experto en medicina, a partir de la segmentación realizada, para el diagnóstico de enfermedades relacionadas a la aorta.

Preguntas de investigación

- ¿Cómo detectar la aorta dentro de una *MRI*?
- ¿Cómo lograr una segmentación efectiva de la aorta con la menor intervención posible del experto?
- ¿Cómo debe presentarse en la aplicación la información al experto?

Tareas de investigación

- Estudiar los métodos usados tradicionalmente para la segmentación de estructuras similares a la aorta.
- Realizar una búsqueda acerca del proceso de obtención de la línea central automáticamente.
- Analizar los algoritmos con mejores desempeños en la segmentación de esta arteria a partir de las *MRI*.

Justificación de la investigación

La evaluación de arterias coronarias constituye un paso importante cuando se diagnostica enfermedades relacionadas con las mismas. Existe una amplia gama de algoritmos especializados para la segmentación de esta clase de arterias, pero de manera general estos deben ser inicializados manualmente.

Se ha demostrado en diferentes trabajos realizados alrededor del tema de la segmentación en general, que la que se realiza manualmente por especialistas, unido al contratiempo mencionado anteriormente, presenta en muchas ocasiones imprecisiones asociadas al trazado de la misma por el experto. El empleo de una técnica de detección de la aorta, así como de un algoritmo automático para su segmentación, permitirá una identificación más precisa del contorno de esta arteria y, por tanto, una mayor certidumbre en el resultado del diagnóstico(Zhao, Zhang et al. 2006).

Viabilidad de la investigación

Para el desarrollo de la investigación se cuenta con gran cantidad de información científica alrededor del tema. Además, se utilizarán las herramientas *ITK* (*Insight Toolkit*, por sus siglas en inglés) para el procesamiento de las imágenes y *VTK* (*Vizualization Toolkit*, por sus siglas en inglés) para la visualización, que son software libre y multiplataforma. También, se emplea la biblioteca multiplataforma *Qt* para el desarrollo de la interfaz gráfica de usuario, el cual distribuido bajo los términos de *GNU Lesser General Public License* (y otras), es software libre y de código abierto. De manera general, la aplicación toma como entrada dos series de imágenes: las imágenes de magnitud que contienen la información morfológica y las imágenes de fase-contraste que manejan la información funcional (tres componentes vectoriales del flujo sanguíneo); de las cuales se cuenta con varios casos de prueba.

Finalmente, en el grupo CIMNE se cuenta con los medios técnicos y con el software necesario para poder realizar todo el trabajo de implementación.

Aportes de la investigación

El presente trabajo se adentra en un campo abierto de investigación en el mundo, el cual está lleno de nuevas posibilidades para nuestro país. El diagnóstico asistido por computadora sin lugar a dudas resulta una idea muy trabajada en la actualidad y que reporta innumerables aplicaciones en el campo de la medicina, por lo que una asimilación de las técnicas desarrolladas hasta el momento en la segmentación de la aorta posibilitaría adentrarse en esta área del conocimiento y a su vez desarrollar una herramienta de este género que podría apoyar el arduo trabajo de nuestros doctores, a la vez que permitiría extender estos conocimientos a otras estructuras de interés.

Estructura de la Tesis

Este Trabajo de Diploma cuenta con un resumen, *abstract*, introducción, tres capítulos, conclusiones, recomendaciones y anexos.

El Capítulo 1 es el marco teórico de este trabajo y trata fundamentalmente la teoría de la segmentación en el procesamiento de imágenes médicas, así como sus diferentes enfoques. También, aborda algunas características generales del estándar de imágenes tomado como entrada en la aplicación. Posteriormente, se valoran diferentes elementos de la representación dinámica del flujo de la sangre a través de la aorta, donde se habla sobre la representación y el procesamiento de estas representaciones. Finalmente, se explica la estructura fundamental del algoritmo de segmentación automática usado en la aplicación, en particular se habla de las tareas de pre-segmentación, extracción de la línea central y segmentación exacta.

El Capítulo 2 se adentra en las características propias del software. Primeramente, se hace una breve introducción a cada una de las herramientas de software empleadas (concretamente, *ITK*, *VTK* y *Qt*) en el proceso de implementación, destacando sus respectivos rasgos fundamentales; al tiempo que se exponen las particulares del entorno de trabajo disponible y sobre la gestión de la generación de código en la aplicación (ver Anexo I). A continuación, se introducen consideraciones generales de la implementación y se comienza explicando el mecanismo de lectura de las imágenes, así como las clases

encargadas del mismo. Luego, se exponen criterios relacionados con el proceso de visualización y la implementación de la interfaz de usuario de la aplicación. Seguidamente, se explica la estructura de la tubería de segmentación del software y se precisan pautas fundamentales de su desarrollo, y además, se exponen algunos criterios sobre los resultados obtenidos.

El Capítulo 3 constituye el manual de usuario de la aplicación desarrollada en el presente trabajo. En este se explican inicialmente algunos criterios relacionados con la estructura general del software y a continuación, se profundiza en las características de cada opción brindada por el mismo.

CAPÍTULO 1: PROBLEMÁTICA DE LA SEGMENTACIÓN DE LA AORTA.

1.1 Introducción

Este capítulo aborda fundamentalmente el tema de la segmentación como concepto básico en el procesamiento de imágenes médicas. Por otra parte, muestra características fundamentales del estándar *DICOM* (*Digital Imaging and COmmunications in Medicine*), que constituye el formato de imágenes con que se trabajará en la aplicación. Además, hace un recorrido por la representación dinámica del flujo sanguíneo en la aorta, deteniéndose en los patrones del mismo.

La segmentación de imágenes médicas constituye una tarea realmente desafiante. Un gran número de diferentes métodos han sido propuestos e implementados en los años recientes. A pesar de los enormes esfuerzos invertidos en este problema en concreto, no se ha encontrado una única estrategia que pueda de manera generalizada solucionar el problema de la segmentación para una amplia variedad de modalidades de imágenes existentes en la actualidad.

Los algoritmos de segmentación más efectivos son obtenidos mediante una cuidadosa combinación personalizada de los componentes. Los parámetros de esos componentes son puestos a punto según las características de la modalidad de la imagen usada como entrada y las características de la estructura anatómica a ser segmentada.

1.2 Propósito general de la segmentación.

La segmentación es el proceso de identificar y clasificar datos encontrados en una representación muestreada digitalmente (Ibáñez, Schroeder et al. 2005) . Típicamente, la representación muestreada es una imagen adquirida a partir de instrumentos médicos tales como una Tomografía Computarizada (*Computerized Tomography*) o como en nuestro caso, de un equipo de Resonancia Magnética (*Magnetic Resonance Imager*).

La segmentación de imágenes médicas es fundamental para muchas aplicaciones tales como la identificación de objetos y la compresión de imágenes basada en objetos. Para la interpretación de las imágenes y la descripción automática del contenido de la imagen, más de un elemento es considerado. Típicamente, esto resulta en una partición de la imagen dada. Las particiones son llamadas regiones y deberán representar objetos visuales importantes mostrados en la imagen. En el dominio de la medicina, los huesos, vasos, órganos y otros tejidos duros y blandos son considerados como objetos y se requieren diferentes niveles de detalles en dependencia de la tarea médica(Lehmann, Beier et al.).

De manera general, segmentación significa localización y delineación de estructuras relevantes. Mientras los humanos pueden usualmente realizar la parte de la localización relativa a la segmentación de imágenes de manera superior a los algoritmos computacionales, la parte referente a la delineación constituye el dominio típico del procesamiento y análisis de imágenes médicas. La segmentación basada en computadoras produce figuras detalladas y reproducibles o superficies de objetos relevantes. Consecuentemente, los algoritmos de segmentación automática están especializados en ciertas áreas de aplicación como en la segmentación del disco óptico en imágenes de la retina, o en el caso que se trata, en la segmentación de la aorta. Estos algoritmos resultan inaplicables a otros objetos de interés o incluso a otra modalidad de imágenes sin antes rediseñar el procedimiento, dado que es requerida una adecuada formulación del conocimiento a priori.

1.3 Diferentes enfoques de la segmentación.

La biblioteca ITK provee un conjunto poderoso de algoritmos que pueden ser usados para desarrollar y personalizar una aplicación completa de segmentación. Algunos de los componentes de segmentación más comúnmente usados en el procesamiento de imágenes médicas son abordados a continuación.

1.3.1 Segmentación basada en el crecimiento de la región.

Los algoritmos basados en el crecimiento de la región han probado ser un método efectivo en la segmentación de imágenes. La estrategia básica en un algoritmo de esta clase es comenzar en una región semilla, típicamente un conjunto de pixeles sobre los que se conoce a priori su pertenencia o no al interior del objeto a ser segmentado. Posteriormente, los pixeles en la vecindad de la semilla son evaluados para determinar si estos deben ser considerados como parte del objeto. Si es así, estos pixeles son agregados a la región y el proceso continúa mientras sea posible agregar nuevos elementos a la región. Los algoritmos basados en el crecimiento de la región varían en dependencia del criterio usado para determinar cuándo un determinado pixel debe ser incluido en la región o no, el tipo de conectividad usado para determinar las vecindades, y la estrategia usada para visitar los pixeles vecinos.

Dentro de esta clase de algoritmos se encuentra los que usan como criterio para incluir pixeles en la región la evaluación de los valores de intensidad dentro de un intervalo específico. Los valores del umbral inferior y superior de intensidad son provistos por el usuario, y el algoritmo incluye los pixeles cuyos valores de intensidad se encuentran dentro de dicho intervalo. El ruido presente en las imágenes puede reducir significativamente la capacidad de estos métodos para aumentar grandes regiones. Por tanto, cuando se encuentran imágenes con ruidos resulta conveniente pre-procesar la imagen por medio de la aplicación de algún algoritmo de reducción de ruido(Ibáñez, Schroeder et al. 2005). Como ejemplo de estos se tiene el implementado por la clase *CurvatureFlowImageFilter* de la biblioteca *ITK*.

Como la inicialización de estos algoritmos requiere que el usuario provea una semilla, es conveniente seleccionar este punto en una región típica dentro de la estructura anatómica a ser segmentada. En diferentes aplicaciones de este método se conoce que la región de interés no es completamente segmentada, lo cual ilustra la vulnerabilidad de los métodos de segmentación basados en el crecimiento de la región cuando la estructura anatómica a ser segmentada no cuenta con una distribución estadística homogénea sobre el espacio

de la imagen(Ibáñez, Schroeder et al. 2005). Entonces debe experimentarse con los valores de los umbrales para lograr obtener una región extendida.

1.3.2 Segmentación basada en vertientes.

La segmentación basada en vertientes clasifica los pixeles dentro de una clase usando el gradiente descendiente o conjugado (Luenberger and Ye 2008) de las características de la imagen y analizando los puntos débiles en las fronteras de la región. Si se imagina el agua cayendo sobre una topología en forma de paisaje y fluyendo según la gravedad hasta acumularse en las cuencas más bajas. El tamaño de dichas cuencas se incrementará si se incrementa la cantidad de precipitación hasta que ellos se derramen sobre otros, causando que las pequeñas cuencas se mezclen entre sí formando otras mayores. Las regiones (zonas de influencia) son formadas usando la estructura geométrica local para asociar puntos en el dominio de la imagen con extremo local en alguna característica medida tal como la magnitud de la curvatura o el gradiente. Esta técnica es menos sensitiva a los valores de umbral definidos por el usuario que los métodos clásicos basados en el crecimiento de la región y puede ser más conveniente para fusionar diferentes tipos de características de diferentes conjuntos de datos.

La técnica de vertientes es también más flexible debido a que no produce una única segmentación de la imagen, en cambio es posible extraer una jerarquía de segmentaciones de las cuales puede ser extraída a priori una única región o conjunto de regiones, usando un umbral o de manera interactiva con la ayuda de una interfaz de usuario.

La estrategia de la segmentación basada en vertientes es tomar una imagen f como una función de peso, esto es la superficie formada por f considerada como una función de sus parámetros independientes $x \in U (U \subset \mathbb{R})$. La imagen f no es en ocasiones el dato de entrada original, sino que es derivada de datos obtenidos a partir de algún filtro. La suposición es que grandes valores de f (o $-f$) indican la presencia de fronteras en los datos originales.

Las vertientes pueden por tanto ser consideradas como un paso final o intermediario en un algoritmo de segmentación híbrido, donde la segmentación inicial es la generación del mapa de aristas de características (*edge feature map*).

El gradiente descendiente asocia las regiones con valores de mínimo local de f (claramente puntos interiores) usando la vertiente del grafo de f. Esto es, la segmentación consiste en todos los puntos en U cuyos caminos de descenso pronunciado en el grafo de f terminan en el mismo mínimo en f. Así habrán tantos segmentos en la imagen como mínimos tenga f. En el caso 1D ($U \subset \mathbb{R}$), las fronteras de las vertientes son los máximos locales de f y los resultados de la segmentación de las mismas es trivial. Para dominios de imágenes de mayor dimensión, las fronteras de las vertientes no constituyen un simple fenómeno local, ellas dependen de la forma de la vertiente completa.

El inconveniente de la segmentación basada en vertientes es que produce una región para cada mínimo local (en la práctica son muchas regiones) y resultados demasiado segmentados. Para aliviar esto se puede establecer una profundidad mínima de la vertiente. La profundidad de una vertiente es la diferencia en altura entre la vertiente mínima y el punto más bajo en la frontera. En otras palabras, es la máxima profundidad de agua de una región que puede sostenerse sin fluir a algunos de sus vecinos. Así un algoritmo de segmentación basado en vertientes puede secuencialmente combinar las vertientes cuyas profundidades se encuentran por debajo del mínimo, hasta que todas las que tengan profundidad suficiente. Esta medición de la profundidad puede ser combinada con otras medidas de prominencia tales como el tamaño. El resultado es una segmentación que contiene regiones cuyas fronteras y tamaños resultan significativos. Debido a que el proceso de unión es secuencial, este produce una jerarquía de regiones. Diferentes trabajos previos (Ibáñez, Schroeder et al. (2005)) han demostrado que resulta beneficioso un procedimiento asistido por el usuario donde se muestre mediante una interfaz gráfica esta jerarquía, de manera que un especialista pueda de manera rápida moverse de regiones pequeñas que presentan un área de interés a la unión de regiones que correspondan a la estructura anatómica.

Existen dos algoritmos comúnmente usados a la hora de implementar el método de las vertientes: de arriba hacia abajo (*top-down*) y de abajo hacia arriba (*bottom-up*). En el primer caso la función *f* en cuestión tendrá valores de punto flotante. En la segunda estrategia, comienza con semillas en los puntos de mínimo local en la imagen e incrementa las regiones hacia afuera y hacia arriba en niveles de intensidad discretos (equivalente a una secuencia de operaciones morfológicas en ocasiones llamadas vertientes morfológicas). Esto limita la exactitud del mismo, ya que fuerza a solo valores discretos el conjunto de niveles de gris de la imagen.

1.3.3 Segmentación basada en conjunto-nivel.

El paradigma del conjunto-nivel es el que se basa en un método numérico para rastrear la evolución de los contornos y las superficies. En lugar de manipular el contorno directamente, este es embebido como el conjunto-nivel cero de una función de dimensión superior, llamada función de conjunto-nivel. Esta función es desarrollada bajo el control de una ecuación diferencial (Ho, Kim et al.). En cada momento el contorno desarrollado puede ser obtenido extrayendo el conjunto-nivel cero de la salida. La ventaja principal de usar la variante de conjunto-nivel es que pueden ser modeladas figuras arbitrariamente complejas y que son manejados de manera implícita cambios topológicos como la fusión y la división.

Los conjuntos-nivel pueden ser usados para la segmentación de imágenes mediante el empleo de características basadas en imágenes (*image-based*) tales como la intensidad media, el gradiente y las aristas en la ecuación diferencial que lo gobierna. En un método típico, un contorno es inicializado por un usuario y entonces es evolucionado hasta que llene la forma de la estructura anatómica en la imagen. Diferentes implementaciones y variantes del concepto básico son publicadas en la literatura.

1.4 El estándar DICOM.

Con la introducción de la Tomografía Computarizada (*CT*, por sus siglas en inglés) seguida de otras modalidades de diagnóstico de imágenes digitales en los años 70 y el crecimiento del uso de las computadoras en aplicaciones clínicas, el *American College*

of Radiology (ACR) y la *National Electrical Manufacturers Association (NEMA)* reconocieron la necesidad emergente de un método estándar para la transferencia de imágenes y la información asociada a las mismas entre dispositivos fabricados por diferentes proveedores. Estos dispositivos producen una gran variedad de formatos de imágenes diferentes.

El estándar *DICOM (Digital Imaging and Communications in Medicine)* se diseñó con el objetivo de (Association 2009):

- Promover la comunicación de información asociada a imágenes médicas, independientemente del fabricante del dispositivo.
- Facilitar el desarrollo y expansión de imágenes archivadas y sistemas de comunicación que puedan servir de interfaz con otros sistemas de información hospitalaria.
- Permitir la creación de bases de datos de información de diagnóstico que puedan ser interrogadas por una amplia variedad de dispositivos distribuidos geográficamente.

Este formato contiene información concerniente a una amplia variedad de aspectos de los datos de las imágenes médicas. Estos son organizados en una jerarquía con los pacientes en el inicio de la misma. Para cada paciente está disponible uno o varios estudios, cada estudio por su parte puede contener varias series de imágenes y finalmente cada serie consiste en varias imágenes. Para los datos de volúmenes, una imagen representa un corte bidimensional.

A continuación se menciona algunas partes importantes de los datos DICOM. El término oficial para estas partes es grupo; cada grupo consiste de información básica de las etiquetas (*tags*) *DICOM*.

- **Identificación.** Esta sección contiene información esencial para localizar una serie de datos de imágenes en una base de datos. Esto incluye el tiempo y la fecha de la adquisición de la imagen. Mucha de esta información resulta esencial para el uso de los resultados; por ejemplo, el nombre del físico referido.

- **Parámetros de adquisición.** Esta sección contiene información concerniente a la modalidad de las imágenes (*CT*, *MRI*, etc.), los parámetros específicos del explorador (*scanning*) (nombre de la secuencia, variantes) y la administración del agente contrastante. También, son registrados el nombre del fabricante y su producto. Finalmente, se indica la posición y orientación del paciente. Cierta información resulta específica para ciertas modalidades (y restringida a esa modalidad), tal como la fuerza del campo magnético y el tiempo de repetición para datos *MRI*.
- **Datos del paciente.** Constituyen parte de esta sección el nombre del paciente, el identificador único del paciente, fecha de nacimiento, sexo, entre otros. El peso y la estatura del paciente son etiquetas opcionales que resultan útiles, por ejemplo, para relacionar ciertos hallazgos con el peso del paciente.
- **Datos de la imagen.** Para la localización y evaluación de las imágenes, una variedad de valores resultan esenciales. Las etiquetas correspondientes son parte de dicha sección. Constituyen ejemplos de las mismas, el identificador del estudio, el número de la serie, el grupo de la imagen y el número de adquisición.
- **Presentación de la imagen.** En esta sección se encuentra la distancia entre cortes y el espaciado (*spacing*) entre pixeles. También son representados en esta sección el rango de valores y el número de bits asignados. Finalmente, puede ser especificada una configuración por defecto para la visualización de los datos (*WindowWidth*, *WindowCenter*).

Muchos de estos datos tienen una influencia inmediata en la interpretación de la imagen, es por tanto una práctica común en las estaciones de trabajo radiológicas incluir información en una leyenda cuando los datos radiológicos son mostrados. En particular, es mostrado usualmente información del paciente así como algunos datos de interpretación de la imagen.

Para la visualización médica, la parte 14 del estándar DICOM resulta particularmente importante. Esta describe una función de visualización estándar de escala de grises (*Grayscale Standard Display Function*). Esta describe la transformación de valores

medidos a valores observables de luminosidad en dependencia del dispositivo. El propósito de definir esta escala es permitir a las aplicaciones conocer a priori como se realiza dicha transformación.

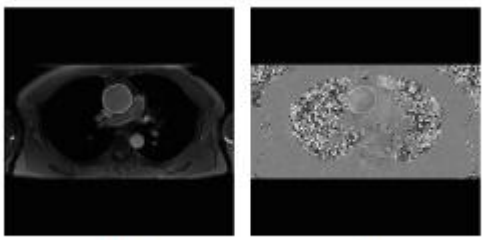
1.5 Representación dinámica del flujo de la sangre en la aorta.

En el caso de las enfermedades de la aorta, las decisiones en su tratamiento deberán evaluar el riesgo de complicaciones causadas por la propia enfermedad contra el riesgo de complicaciones causadas por una operación. El máximo diámetro de la aorta es el principal parámetro que permite la evaluación del riesgo de ruptura o disección. Sin embargo, como se ha mencionado antes, sucede que este parámetro no es ni confiable ni robusto.

Por consiguiente, se hace necesario evaluar otros criterios evolucionados para considerar los riesgos de ruptura dado que el único tratamiento disponible es la cirugía. El análisis de los patrones del flujo de la sangre al nivel de la enfermedad de la aorta parce ser una vía prometedora.

1.5.1 Procesamiento de imágenes de flujo.

Durante el ciclo cardiaco, con una resolución media temporal de 35 milisegundos (ms), son generadas una imagen de magnitud que provee información morfológica y 3 imágenes de fase-contraste (imágenes PC por el término inglés *phase-contrast*) también conocidas como imágenes de velocidad codificada (*velocity-encoded images*). Las tres imágenes de PC corresponden a la codificación de la velocidad en las siguientes 3 direcciones: a través del plano, izquierda-derecha y anterior-posterior.



(a) (b)

Figura 1.1: Imágenes de flujo. (a) Imagen de magnitud. (b) Imagen de fase-contraste (a través del plano de velocidad codificada) al nivel de la raíz de la aorta.

Resulta poco práctico lograr una detección de la pared de la aorta directamente de las imágenes PC dado que estas imágenes contienen poca información sobre la morfología de la aorta. Por esta razón, se lleva cabo la extracción de información morfológica de las imágenes de magnitud y se combina con la información funcional (velocidad del flujo sanguíneo) de imágenes PC.

- **Detección automática de aristas**. La detección de la pared de la aorta es realizada por un método de procesamiento automático de imágenes sin tomar en cuenta la localización en la aorta.

- **Detección del lumen de la aorta.** La detección automática de la pared de la aorta define una región de interés restringida al lumen de la aorta en toda la serie de imágenes de magnitud. En cada imagen PC, restringida a su región de interés, el conocimiento de la velocidad instantánea en cada dirección permite la construcción de una representación 3D del flujo sanguíneo de la aorta.

- **Supresión de ruido.** Un filtro de media 3x3 (*mean filter*) es aplicado para reducir el ruido en las imágenes PC. Este es en parte debido al movimiento respiratorio, por lo que estas señales no pueden ser suavizadas. El ruido causa dificultades para distinguir entre un tejido y otro (en las imágenes de magnitud) y para evaluar las velocidades del flujo con exactitud. El filtro *mean* es exclusivamente aplicado en píxeles localizados en el lumen de la aorta de las imágenes PC, esto permite evaluar señales del flujo sanguíneo de la aorta. Por supuesto, la detección automática de la pared de la aorta en las imágenes de magnitud es robusto frente al ruido.

- **Decodificación de la velocidad del flujo sanguíneo.** La velocidad del flujo sanguíneo localizada en un pixel ($P_{i,j}$) depende de la velocidad de adquisición ($V_{adquisition}$) y de la dinámica de la imagen ($Dynamic$) (4096 niveles de gris en

imágenes codificadas DICOM) de acuerdo a la siguiente fórmula:

$$Velocity_{P_{i,j}} = \frac{2\ P_{i,j}\ V_{adquisition}}{Dynamic} - V_{adquisition} \quad (\mathrm{cm/s})$$

$$Velocity_{P_{i,j}} \in [-V_{adquisition}, \quad V_{adquisition}]$$

En el ejemplo, $V_{adquisition} = 500\ \mathrm{cm/s}$ para imágenes PC a través del plano de dirección y $P_{i,j} \in [0, Dynamic = 4096]$. Si $P_{i,j} = 1024$ entonces $Velocity_{P_{i,j}} = \frac{1024 \times 500 \times 2}{4096} - 500 = -250\ \mathrm{cm/s}$ (Xavier, Lalande et al. 2007).

1.5.2 Análisis del flujo de la sangre por la aorta.

Las imágenes PC brindan la oportunidad de valorar las características 3D del flujo sanguíneo. Un análisis del flujo sanguíneo es también necesario para extraer información cuantitativa y cualitativa.

1. **Parámetros de velocidad.** El primer parámetro extraído es el componente máximo de las velocidades instantáneas en cada una de las 3 direcciones espaciales, dando información acerca de la dinámica de la velocidad. Una dinámica importante, como la variación amplia de velocidad, en las direcciones R-L (*Right-Left*) y A-P (*Anterior-Posterior*) parece indicar un flujo turbulento. Y una dinámica extensiva en la dirección del plano T-P (*Through-Plane*) indica presencia de un flujo pulsátil. La máxima velocidad del flujo sanguíneo 3D es la máxima norma de los vectores de velocidad 3D de todas las imágenes que cubren el ciclo cardiaco completo. Una alta velocidad del flujo sanguíneo 3D podría indicar un problema en la válvula de la aorta al nivel de la raíz o una pérdida de elasticidad al nivel de la pared de la aorta.

2. **Flujo sanguíneo instantáneo.** Evaluar el flujo sanguíneo de la aorta durante el ciclo cardiaco puede ayudar en el diagnóstico de algunas patologías en la aorta. Por ejemplo, las enfermedades en las válvulas de la aorta (especialmente en casos de regurgitación o vómitos) generan importantes flujos de regreso y consecuentemente una circulación sanguínea menos eficiente. El flujo sanguíneo instantáneo es calculado a partir de la velocidad del flujo sanguíneo media y del

área del lumen de la aorta.
La velocidad media instantánea ($Velocity_{Mean}(t)$) de cada imagen es calculada de acuerdo a la ecuación:

$$Velocity_{Mean}(t) = \sum_{i=0}^{PixelNb} \frac{Velocity_i(t)}{PixelNb} \quad (cm/s)$$

El área del lumen ($Area$) es calculada del número del pixel en el lumen de la aorta y del tamaño del pixel en la imagen ($PixelSurface$) en mm^2 y

$$Area = PixelSurface \times 0.01 \times PixelNb \quad (cm^2)$$

El flujo sanguíneo instantáneo ($Flow(t)$) es el flujo en el tiempo t del ciclo cardiaco y se define como:

$$Flow(t) = \frac{Area \times Velocity_{Mean}(t) \times 60}{1000} \quad (L/min)$$

3. **Presión hemodinámica y velocidad radial.** La presión hemodinámica ($P_{Blood_i}(t)$) es la presión resultante de una contracción del corazón suministrando un flujo arterial pulsátil. El movimiento de la sangre crea alguna presión sobre la pared de la aorta que depende de la velocidad y densidad de la sangre ($\rho_{Blood} = 1050\ kg.m^{-3}$).

$$P_{Blood_i}(t) = \frac{1}{2} \times \rho_{Blood} \times V_{Radial_i}(t)^2 \quad (Pa)$$

Para valorar la presión hemodinámica aplicada sobre la pared de la aorta, el componente radial ($\overrightarrow{V_{Radial}}$) del vector de velocidad instantánea ($\overrightarrow{V(x,y)}$) (cerca de la pared de la aorta) debe ser determinado. Velocidades radiales importantes pueden estar vinculadas a estrés parietal.
Con idea de obtener el componente radial, el centro de gravedad del lumen de la aorta (G) es calculado a partir del contorno del lumen de la aorta. Entonces, ($\overrightarrow{V_{Radial}}$) se define como la proyección ortogonal de $\overrightarrow{V(x,y)}$ sobre la línea que une el origen del vector de velocidad (B) y (G).

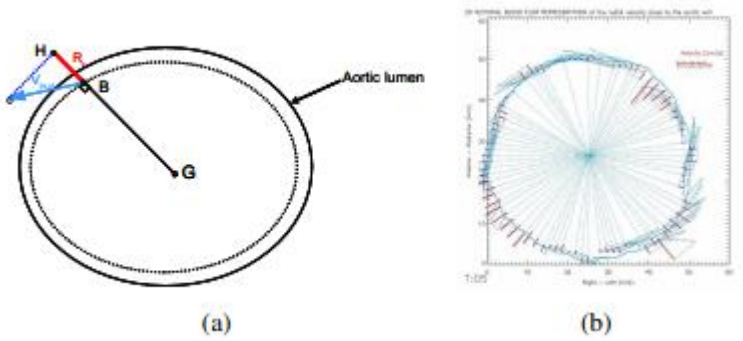


Figura 1.2: (a) Componente radial del vector de velocidad. (b) Ejemplo de visualización de la velocidad radial.

4. **Flujo hacia atrás.** El flujo hacia atrás está definido como el volumen de sangre moviéndose contrario hacia el corazón comparado con el volumen de sangre que se mueve hacia fuera desde él. Este es calculado a partir de una curva *spline* interpolando los valores de flujo ($Flow(t)$). Este es la razón de la integral de la curva por encima y por debajo del eje X. Este parámetro es en ocasiones evaluado con el objetivo de determinar la fracción de regurgitación (náusea o vómito)

1.5.3 Representaciones del flujo sanguíneo de la aorta.

1. **Diagrama vectorial 2D del flujo de la sangre.** El diagrama vectorial 2D del flujo de la sangre con codificación de colores (de acuerdo a la orientación de la velocidad) provee una visualización comprensiva y adaptable del movimiento del flujo en el plano.

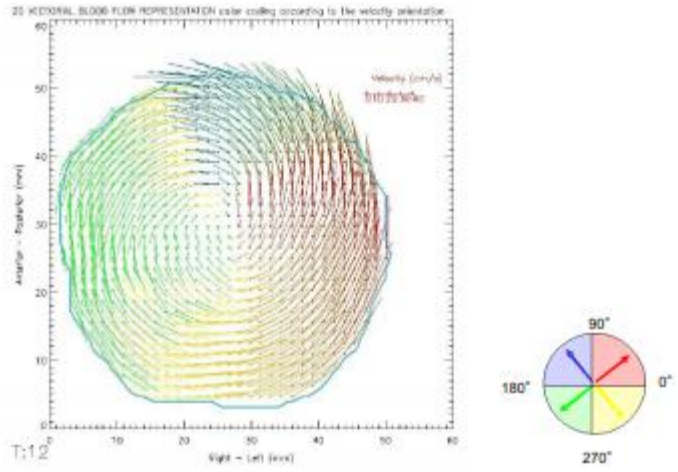


Figura 1.3: Representación vectorial 2D del flujo sanguíneo con codificación de colores de acuerdo a la orientación de la velocidad en el tiempo t = 540 ms. El flujo sanguíneo es impulsado por un movimiento circular.

2. **Representación de la velocidad radial.** En cada imagen la presión hemodinámica es evaluada a partir de las velocidades radiales medidas cerca de la pared de la aorta.

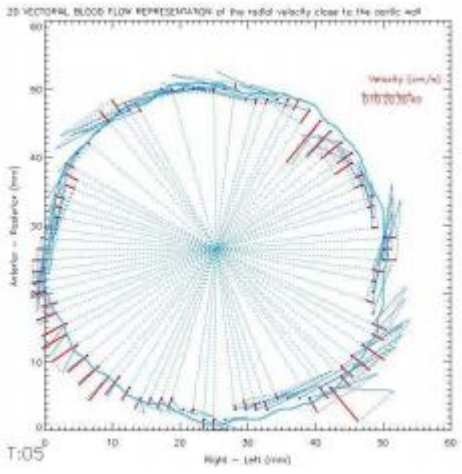


Figura 1.4: Representación vectorial 2D del flujo sanguíneo de la velocidad radial cercana a la pared de la aorta en el tiempo t = 540 ms.

3. **Visualizaciones planimétricas.** Las visualizaciones planimétricas son preferidas para el vector de amplitud de la velocidad instantánea 3D.

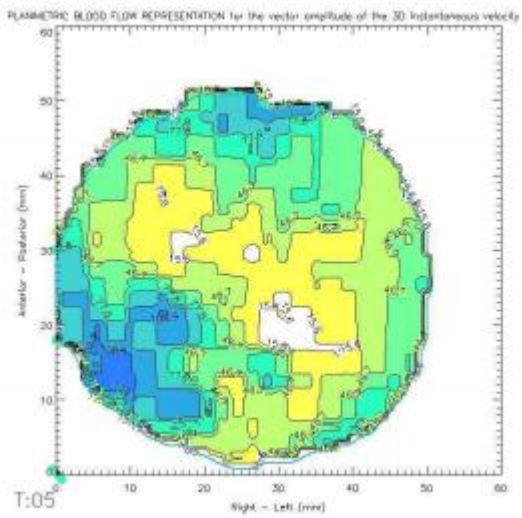


Figura 1.5: Representación planimétrica de la velocidad instantánea 3D en el tiempo t = 340 ms.

4. **Diagrama vectorial 3D del flujo de la sangre.** Un diagrama vectorial 3D del flujo de la sangre con codificación de colores proveen una visualización comprensiva y adaptable de las características del flujo, especialmente la forma del flujo y el flujo hacia atrás. La máxima velocidad 3D es deducida de los resultados obtenidos de la serie de imágenes completas que cubren un ciclo cardiaco.

1.6 Algoritmo de segmentación de la aorta.

Luego de abordar características de la segmentación y otros temas de interés, es momento de introducir el algoritmo de segmentación específico que se empleó en la aplicación.

Una imagen *n*-fase 4D (espacial-temporal) I puede ser vista como un conjunto discreto de n imágenes volumétricas definidas en n instantes de tiempo diferentes. La superficie aórtica 4D puede ser tomada como una secuencia de superficies. El algoritmo de segmentación 4D consiste en:

i. Pre-segmentación de la superficie de la aorta: un método marchando rápido de conjunto-nivel (*fast marching level set*) 4D simultáneamente produce superficies aórticas 4D aproximadas.

ii. Extracción de la línea central: las líneas centrales de la aorta son determinadas por cada superficie aproximada mediante la esqueletonización (del término inglés no oficial, *skeletonization*) (Lee, Kashyap et al. 1994).

iii. Segmentación de la superficie de la aorta: la superficie de la aorta es obtenida a partir del estudio del comportamiento sanguíneo. Inicialmente, se aplica método umbral binario (*binary threshold*) con el objetivo de obtener una imagen binaria a partir de la original.

1.6.1 Pre-segmentación de la superficie de la aorta.

Con el objetivo de pre-segmentar la imagen 4D de la aorta, un método marchando rápido (*fast marching*) 4D es usado. Comenzando con un punto de semilla interactivamente identificado dentro de la imagen 4D de la aorta, la superficie inicial lo propaga en dirección hacia afuera con una velocidad F. Sea $T_q(i,j,k)$ el tiempo de arribo en el cual la superficie conjunto-nivel (*level set*) pasa por el punto (i,j,k,q) en la imagen 4D. El gradiente de este tiempo de arribo será inversamente proporcional a la función de velocidad F.

$$|\nabla T|F = 1$$

La principal idea detrás del método marchando rápido es trazar la superficie de acuerdo a la función solución $T_q(i,j,k)$ resuelta según la ecuación anterior. El método marchando rápido consiste de los siguientes pasos:

1) Agregar el punto de semilla al conjunto *Known*, agregar sus vecinos al conjunto *Trial* y los restantes puntos al conjunto *Far*.

2) Comenzar el ciclo: Encontrar el punto P con el menor valor T en *Trial*, se elimina de *Trial* y se agrega al conjunto *Known*.

3) Chequear todos los puntos vecinos al punto P; si el vecino pertenece a *Far*, se elimina del conjunto *Far* y se adiciona a *Trial*.

4) Recalcular el valor T para todos los puntos vecinos de P de acuerdo a la ecuación anterior.

5) Retornar al inicio del ciclo.

Para mejorar el desempeño de la segmentación, se aplica primeramente un filtro de reforzamiento de vasos (*vessel enhancement*) 3D a la imagen original con el objetivo de suprimir el ruido y la imagen de fondo a objetos tubulares.

1.6.2 Extracción de la línea central.

Un algoritmo de esqueletonización (*binary thinning*) (Palágyi, Sorantin et al. 2001) es empleado con el objetivo de encontrar la línea central (esqueleto) de la imagen de entrada, en este caso de la aorta. La idea general del mismo es corroer la superficie de la aorta iterativamente hasta que solo reste el esqueleto. La corrosión se realiza simétricamente con el objetivo de garantizar que las líneas que definen el esqueleto de la aorta queden en una posición media y que a la vez la conexidad del objeto sea preservada.

El método empleado con este fin inicialmente transforma la imagen de entrada en una imagen binaria transformando el valor de todos los pixeles diferentes de cero en uno. Posteriormente, realiza una serie de pruebas para verificar si un determinado pixel puede ser eliminado del objeto. Esta idea es aplicada a todos los pixeles de la imagen y repetida mientras no ocurran más cambios en dicha imagen. De manera general, el pixel actual puede ser eliminado si:

- Es un pixel de la superficie. Esta prueba solo considera una de las seis posibles direcciones en el espacio tridimensional (considerando cada una de los tres ejes en sentido positivo y negativo) con el objetivo que la esqueletonización sea simétrico. Esto garantiza que la línea central no salga por ningún borde del objeto.
- No es punto final de una línea, porque la línea central es precisamente una poligonal.
- Al borrarlo no se producen huecos.
- Es un punto simple, esto significa que al eliminarlo no se altera el número de objetos conectados. Esta prueba se realiza al final dado que computacionalmente es la más compleja.

Este algoritmo descrito en ha ofrecido buenos resultados encontrando la línea central de diferentes estructuras en general. En el caso de la aorta, en múltiples ocasiones se ha detectado que el algoritmo agrega interconexiones y ramificaciones indeseadas a lo que representaría la línea central En la figura 1.6 se muestra un ejemplo en el que la salida obtenida por este algoritmo presenta estas características.

Con el objetivo de lograr obtener la línea central deseada debe aplicarse un post-procesado a la obtenida por el algoritmo explicado con anterioridad. Para ello se construye un grafo sin dirección donde se considera a cada pixel tridimensional de la línea central como un vértice del mismo. Adicionalmente, un vértice se dice que es adyacente a otro si el máximo entre los módulos de las diferencias de sus respectivas coordenadas es menor o igual a 1. Este criterio pude ser visualizado de la siguiente manera: en el espacio 3D de coordenadas enteras se toma uno de los dos puntos como centro de un cubo de dimensiones 3x3x3, entonces el otro punto considerado será adyacente a él si es uno de los restantes 26 puntos (diferentes al centro) dentro del cubo considerado.

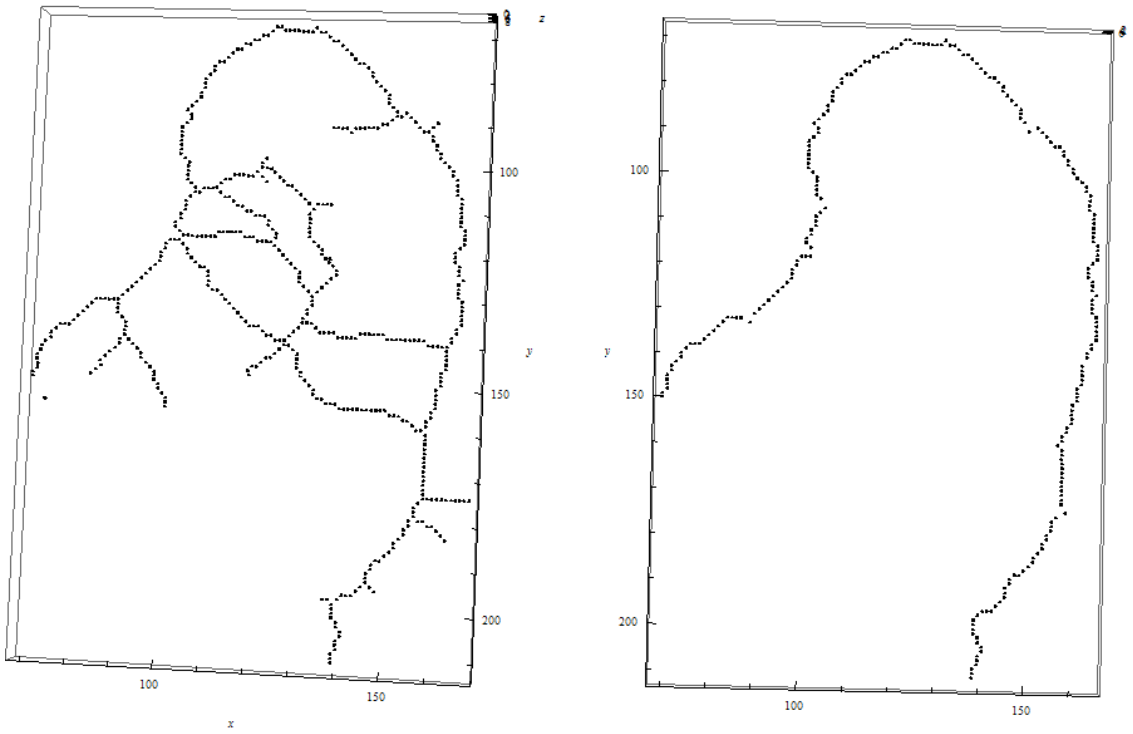


Figura 1.6: Visualizaciones del esqueleto de la aorta en el software *Wolfram Mathematica*. A la izquierda la esqueletonización obtenida con el algoritmo de esqueletonización explicado arriba. A la derecha la obtención definitiva de la línea central luego de pos procesar la esqueletonización inicial.

Inicialmente, se ordenan los puntos de entrada según el valor de su coordenada 'y', lo cual evidentemente no representa un orden total de los mismos, pero permite obtener el punto de menor coordenada 'y' que será tomado como punto inicial y además mejorará el proceso de construcción del grafo inicial, ya que los puntos cercanos a un punto en particular en este orden, tienen una alta probabilidad de representar un vértice adyacente al mismo en el grafo en cuestión. Posteriormente, este grafo es reducido a otro donde se consideran como vértices del mismo los vértices del grafo original que tengan más de dos vértices adyacentes o solo cuente con uno. El camino entre ambos vértices del grafo original se guarda como información asociada a la arista del grafo reducido. A partir de este momento, se aplican diferentes criterios para evitar ciclos y eliminar aristas dobles. A continuación se usa un método para encontrar el camino máximo partiendo del punto

inicial (que siempre se considera como un vértice del grafo), permitiendo obtener la salida deseada libre de bifurcaciones e interconexiones.

En el Capítulo 2 del presente trabajo se aborda más detalladamente el procedimiento descrito, así como también se ofrecen pautas importantes en su implementación.

1.6.3 Segmentación de la superficie de la aorta.

Gracias a que las imágenes con escala de grises pueden ser transformadas a imágenes binarias mediante la selección de un valor umbral, el algoritmo de detección de la superficie de la aorta inicialmente obtiene una representación binaria a partir de la imagen original mediante la aplicación de un método *binary threshold*. Seguidamente, se reajustan diferentes parámetros como los niveles de grises y la velocidad de adquisición. Luego, se aplica un algoritmo que obtiene una imagen 3D que representa la componente con mayor desviación estándar en la imagen 4D de entrada.

A continuación, se aplica una operación de corrosión binaria (*binary erode*) morfológica mediante la selección de un valor umbral (de manera similar a *binary threshold*) y con el propósito de eliminar pixeles de la frontera en la imagen original (Ibáñez, Schroeder et al. 2005).

Posteriormente, un método de componente conectado (*connected-component*) etiqueta los objetos dentro de la imagen binaria obtenida en el paso anterior con el objetivo de encontrar la máxima componente conectada. A cada objeto distinto le es asignada una etiqueta única, siendo las etiquetas finales consecutivas y comenzando en uno.

A partir de la imagen con componentes etiquetados obtenida, se emplea una transformación en la que se vuelven a etiquetar dichos componentes. Este asegura que las etiquetas finales sean números consecutivos sin huecos intermedios. Además, ordena las etiquetas basadas en el tamaño del objeto al que pertenecen: el mayor objeto tendrá etiqueta uno, el segundo mayor será etiquetado como dos, y así sucesivamente (la etiqueta cero se emplea para representar el fondo).

En el próximo capítulo se detalla acerca de cada uno de los métodos mencionados y su implementación.

1.6 Conclusiones parciales.

Conocer los diferentes enfoques de la segmentación de imágenes médicas permitirá valorar de manera correcta los algoritmos empleados en la aplicación, así como pensar en posibles mejoras a ser realizadas en un futuro. Además, resulta importante conocer las características del formato de imágenes *DICOM* para tener acceso a todos sus metadatos, lo cual en el presente Trabajo de Diploma resulta imprescindible en el proceso de lectura y composición de las mismas. Igualmente, estudiar algunas de las características dinámicas del flujo sanguíneo en la aorta, hará posible seleccionar que valores resultan de interés al experto en medicina y deberán ser mostrados en una aplicación afín.

CAPÍTULO 2: ARQUITECTURA DE LA APLICACIÓN DE SEGMENTACIÓN DE LA AORTA.

2.1 Introducción.

En el presente capítulo se abordó la forma en que se cumplieron los pasos propuestos en Algoritmo de segmentación de la aorta. Adicionalmente, se describirán los aspectos relacionados con la arquitectura del software (denominado *Simple Aorta Flow Application*), cada una de las características que posee el mismo, además de las principales clases programadas que estén más estrechamente vinculadas con el funcionamiento general del mismo.

2.2 Tecnología usada.

Para la construcción de *SAFA* (*Simple Aorta Flow Application*) se decidió utilizar la tecnología *Qt* para el desarrollo de la interfaz de usuario, aprovechando su capacidad de integración con las restantes herramientas empleadas. Por otra parte, para el procesamiento de imágenes médicas se usó la biblioteca multiplataforma *ITK* (*Insight Toolkit*), lo que permitió contar con gran soporte base a nuestra aplicación en lo que respecta al tratamiento de las imágenes. Adicionalmente, se empleó el sistema de software *VTK* (*Visualization Toolkit*) para manejar la visualización de las imágenes y los gráficos asociados a las mismas. A continuación se expondrán algunas consideraciones sobre estos aspectos.

2.2.1 Características de *Qt*.

Qt es una biblioteca multiplataforma ampliamente usada para desarrollar aplicaciones con una interfaz gráfica de usuario así como también para el desarrollo de programas sin interfaz gráfica como herramientas para la línea de comandos y consolas para servidores. También es usada en sistemas informáticos empotrados para automoción, aeronavegación y aparatos domésticos como frigoríficos.

Funciona en todas las principales plataformas, y tiene un amplio apoyo. La *API* de la biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como

uso de *XML*, gestión de hilos, soporte de red, una *API* multiplataforma unificada para la manipulación de archivos y una multitud de otros para el manejo de ficheros, además de estructuras de datos tradicionales.

Existen tres ediciones de *Qt* disponibles en cada una de las plataformas, llamadas:

- *GUI Framework*: edición con nivel reducido de *GUI (Graphical User Interface)*, orientado a redes y bases de datos.
- *Full Framework* : edición comercial completa.
- *Open Source*: edición código abierto completa.

En el desarrollo del software se empleó la edición *Open Source* de *Qt* (versión 4.7.0). Además, se dispuso de C++ como lenguaje de desarrollo, el cual es el que soporta de manera estándar, aunque adicionalmente puede ser utilizado en varios otros lenguajes de programación (como *Python*, *Java*, *Pascal*, *C#*, y otros más) a través de *bindings*.

Qt cuenta con los siguientes módulos:

- *QtCore*: módulo que contiene el núcleo de las clases que no pertenecen al conjunto de clases *GUI* incluyendo el manejador de eventos y el mecanismo de señales y aberturas (*signals and slots*). Además, incluye abstracciones para hilos, mapeo de archivos, memoria compartida, expresiones regulares, entre otros.
- *QtGui*: módulo que contiene la gran mayoría de las clases *GUI*. Esta incluye clases de tablas, árboles y listas basadas en el patrón de diseño modelo-vista-controlador. También provee un sofisticado *widget canvas 2D* capaz de almacenar miles de elementos incluyendo *widgets* ordinarios.
- *QtMultimedia*: módulo que implementa funcionalidad multimedia de bajo nivel.
- *QtNetwork*: módulo que contiene clases para escribir clientes y servidores *UDP (User Datagram Protocol)* y *TCP (Transmission Control Protocol)*, respectivamente. Entre otras funcionalidades.
- *QtOpenGL*: módulo que contiene clases que permiten el uso de *OpenGL* en el dibujo de gráficos 3D.

- *QtOpenVG*: módulo que constituye un *plugin* que provee soporte para *OpenVG*.
- *QtScript*: módulo que constituye un procesador de scripts basados en *ECMAScript*.
- *QtScriptTools*: módulo que provee componentes adicionales para aplicaciones que usen *QtScript*.
- *QtSql*: módulo que contiene clases que se integran con bases de datos *SQL*.
- *QtSvg*: módulo que presenta clases para la visualización del contenido de archivos *svg*.
- *QtWebKit*: módulo que provee clases para la interacción con contenido web.
- *QtXml*: módulo que implementa interfaces entre *SAX* y *DOM* con el analizador (*parser*) de *XML* de *Qt*.
- *QtXmlPatterns*: módulo que ofrece soporte para los esquemas de validación *XPath*, *XQuery*, *XSLT* y *XML*.
- *Phonon*: *API* que provee un control simple de multimedia.
- *Qt3Support*: módulo que contiene clases cuya función es dar soporte a *Qt3*.
- *QtDeclarative*: módulo que constituye una plataforma declarativa para construir interfaces de usuario de manera fluida en *QML*.

A continuación se muestra un código C++ simple de ejemplo.

```
// Código ejemplo 1.
// Muestra la estructura de una aplicación Qt simple.
#include <QtGui>
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QLabel label("Hello, world!");
    label.show();
    return app.exec();
}
```

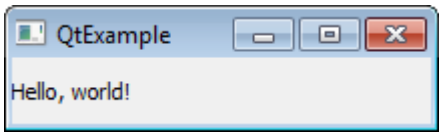


Figura 2.1: Ventana mostrada según el código anterior.

Como característica fundamental de esta plataforma se encuentra el uso del compilador *moc* (*metaobject compiler*), el cual es una herramienta empleada para interpretar ciertas macros del código fuente C++ en forma de anotaciones y generar código adicional a partir de estas con meta información acerca de las clases usadas en la aplicación. Esta meta información es usada por *Qt* para proveer características del lenguaje no presentes de manera nativa en C++, tales como el sistema de señales-aberturas, introspección y llamadas asíncronas de funciones.

2.2.2 Características de *ITK*.

ITK es una plataforma de desarrollo de código abierto (*open source*) y multiplataforma (*cross-platform*) ampliamente usada para el desarrollo de aplicaciones de segmentación de imágenes, en particular de imágenes médicas. La plataforma provee algoritmos de segmentación de alto rendimiento para una, dos, tres y más dimensiones.

Por otro lado, usa el ambiente de trabajo *CMake* para administrar el proceso de configuración. De manera general, la herramienta de *software ITK* está desarrollada en lenguaje C++, siguiendo un estilo de programación genérica basado en el uso de plantillas. Esto significa que el código es altamente eficiente y que varios de los problemas del software son descubiertos en tiempo de compilación y no durante la ejecución del programa. Además, puede ser usada en otros lenguajes *scripts* como *Python* y *Tcl* mediante un proceso de empaquetamiento conocido como *wrapping* (empleando la herramienta *CableSwig*).

Esta plataforma está organizada siguiendo una arquitectura basada en el flujo de datos (*data-flow*). Esto significa que los datos son representados como objetos de dato que son procesados en turnos por objetos de proceso (estos son los llamados filtros). Los objetos

de dato y los objetos de proceso son interconectados mediante tuberías. Las tuberías son capaces de procesar los datos por partes de acuerdo al límite de memoria especificado por el usuario en la misma. Se emplean fábricas de objetos para instanciar los mismos, las cuales permiten una extensión en tiempo de ejecución del sistema. Además, en *ITK* se emplea el patrón comando-observador (*command-observer*) para el procesamiento del sistema de eventos.

Los filtros son un componente esencial de esta herramienta y tienen como objetivo fundamental el procesamiento de imágenes. Estos aceptan una o varias imágenes como dato de entrada y pueden producir a su vez múltiples imágenes de salida. De esta forma, parte de la salida de un determinado filtro A es comúnmente parte de los datos de entrada de otro filtro B, y así sucesivamente conforman las mencionadas tuberías.

A continuación un ejemplo sencillo de una aplicación de *ITK* (lo más simple posible) provista como parte de su código fuente.

```
// Código ejemplo 2.
// Se encuentra disponible en ITK/Examples/Installation/HelloWorld.cxx
#include "itkImage.h"
#include <iostream>

int main()
{
  typedef itk::Image< unsigned short, 3 > ImageType;

  ImageType::Pointer image = ImageType::New();

  std::cout << "ITK Hello World !" << std::endl;

  return 0;
}
```

2.2.3 Características de *VTK*.

VTK (Visualization Toolkit) es un sistema de *software* libre, libremente disponible para la realización de gráficos 3D por computadora, procesamiento de imagen y visualización. *VTK* consiste en una amplia biblioteca de clases implementada en

lenguaje *C++* y varias capas de interfaz de lenguajes interpretados como *Tcl/Tk*, *Java*, y *Python*. Además, cuenta con un amplio marco de visualización de la información, provee un conjunto de *widgets* de interacción 3D, soporta el procesamiento en paralelo y se integra con diversas bases de datos de herramientas *GUI* como *Qt* y *Tk*. *VTK* es multiplataforma y se ejecuta en plataformas *Linux*, *Windows*, *Mac* y *Unix*. También, incluye soporte auxiliar de *widgets* de interacción 3D, anotación bidimensional y tridimensional, y computación paralela.

VTK es mundialmente utilizado en aplicaciones comerciales, investigación y desarrollo, y es la base de muchas aplicaciones de visualización avanzadas tales como: *ParaView*, *3DSlicer*, *MayaVi2* y *OsiriX.3*.

A continuación se muestra la estructura de directorios de *VTK*.

- *VTK/Common*: Contiene las clases del núcleo.
- *VTK/Filtering*: Contiene las clases relacionadas con el procesamiento de datos en la tubería de visualización.
- *VTK/Rendering*: Contiene las clases usadas en la visualización.
- *VTK/VoluineRendering*: Contiene las clases empleadas en la visualización volumétrica.
- *VTK/Imaging*: Contiene los filtros encargados del procesamiento de imágenes.
- *VTK/Graphics*: Contiene los filtros que se especializan en el procesamiento de datos 3D.
- *VTK/IO*: Contiene los filtros relacionados con la lectura y escritura de datos.
- *VTK/Hybrid*: Contiene clases complejas de las que dependen las encontradas en *Imaging* y *Graphics*.
- *VTK/Widgets*: Contiene una serie de clases que representan *widgets* 3D.
- *VTK/Parallel*: Contiene clases que garantizan el soporte al procesamiento paralelo.
- *VTK/GenericFiltering*: Contiene las clases involucradas en soportar la interfaz entre *VTK* y paquetes de simulación externos.

- *VTK/Examples*: Contiene ejemplos bien documentados sobre el correcto uso de las diferentes componentes de *VTK*.
- *VTK/Utilities*: Contiene código que brinda soporte a *png*, *jpeg*, *tiff*, y entre otros. El directorio *Doxygen* contiene scripts y la configuración de programas para generar la documentación de tipo *Doxygen*.
- *VTK/Wrapping*: Contiene código que ofrece soporte al empaquetamiento (*wrapping*) para *Tcl*, *Python* y *Java*.
- *VTK/GUISupport*: Contiene las clases que permiten usar *VTK* con los paquetes de interfaz de usuario *MFC* y *Qt*. En particular, define diferentes clases que permiten la comunicación entre los sistemas de eventos de *Qt* y *VTK* (*vtkEventQtSlotConnect*), así como un *widget* que ofrece integración entre ambas plataformas (*QVTKWidget*).

A continuación se expone un prototipo simple del uso de *VTK*.

```
// Código ejemplo 3.
// Ejemplo que se encuentra en VTK/Infovis/Cxx/HelloWorld.cxx
#include "vtkGraphLayoutView.h"
#include "vtkRandomGraphSource.h"
#include "vtkRenderWindow.h"
#include "vtkRenderWindowInteractor.h"

int main(int, char*[])
{
  vtkRandomGraphSource* source = vtkRandomGraphSource::New();
  vtkGraphLayoutView* view = vtkGraphLayoutView::New();
  view->SetRepresentationFromInputConnection( source->GetOutputPort() );
  vtkRenderWindow* window = vtkRenderWindow::New();
  view->SetupRenderWindow(window);
  window->GetInteractor()->Start();

  source->Delete();
  view->Delete();
  window->Delete();
```

```
  return 0;
}
```

2.2.4 Entorno de trabajo y gestión de la generación de código.

Para la implementación de la aplicación se empleó la plataforma de desarrollo integrado *Eclipse Platform* versión 3.5.2, con el *CDT 6.0* (*C/C++ Development Toolkit*), y un compilador *MinGW* (*gcc* 4.4.0). También, se contó con el plugin *Qt Eclipse Integration* 1.5.2 que integra varias herramientas de desarrollo de Qt a la plataforma Eclipse.

Además, se utilizó el sistema de código abierto y multiplataforma *CMake* para administrar el proceso de generación de código de la aplicación, el cual emplea un método independiente del compilador usado. Esta herramienta cuenta con dependencias mínimas, pues simplemente necesita la existencia en el sistema de un compilador *C++*.

Este entorno de generación fue usado inicialmente para compilar las bibliotecas *ITK* y *VTK* a partir de sus respectivos códigos fuentes y posteriormente fue empleado para compilar la propia aplicación *SAFA*.

Vale señalar que como todas las herramientas usadas para la implementación del software son multiplataforma (aunque necesitan que se genere el código para cada plataforma específica), esta puede ser compilada para todas ellas a partir de la misma estructura del código fuente original.

2.3 Consideraciones generales de la implementación.

La clase *itk::Image* sigue el paradigma de la programación genérica, donde el tipo de dato es separado del comportamiento algorítmico de la clase. Además, soporta imágenes con cualquier tipo de pixel y cualquier dimensión espacial. En el código ejemplo 2 se muestra como se declara una imagen *ITK*, esta representa el típico caso de una imagen escalar (el tipo de dato es un tipo primitivo). Sin embargo, para algunas tareas de procesamiento el tipo de dato requerido no es escalar, un ejemplo típico lo constituye una imagen de vectores. Este último es el tipo de imagen empleado para representar el gradiente de una imagen escalar.

El software *SAFA* toma como datos de entrada, como se ha mencionado anteriormente, imágenes *MRI* 4D cardiovasculares en formato *DICOM*. En particular, se necesitan cargar dos series de imágenes diferentes: las imágenes de magnitud que se corresponden con imágenes escalares y las imágenes de fase-contraste o de velocidad codificada que a su vez constituyen imágenes vectoriales. El tipo de dato de ambas se encuentra declarado en el archivo *Common/itkImageTypes.h*. Seguidamente, algunas declaraciones de los diferentes tipos de imágenes empleadas en la aplicación.

```
// Dimensiones
const unsigned int  Dim3D = 3;
const unsigned int  Dim4D = 4;

// Tipo de pixeles escalares
typedef float                                   PixelType;
typedef unsigned char                           MaskPixelType;

// Tipos de pixeles vectoriales
typedef itk::VariableLengthVector< PixelType > VPixelType;
typedef itk::Vector< PixelType, 3 >            VPixel3Type;


// Tipos de las imagines empleadas
typedef itk::Image< int, Dim3D >             Integer3DImageType;
typedef itk::Image< PixelType, Dim3D >       Scalar3DImageType;
typedef itk::Image< PixelType, Dim4D >       Scalar4DImageType;
typedef itk::Image< VPixel3Type, Dim3D >     Vector3DImageType;
typedef itk::Image< VPixel3Type, Dim4D >     Vector4DImageType;
typedef itk::Image< MaskPixelType, Dim3D >   MaskScalar3DImageType;
```

De acuerdo a las declaraciones anteriores, en la aplicación las imágenes de magnitud son de tipo `Scalar4DImageType`, mientras que las imágenes de flujo son de tipo `Vector4DImageType`.

2.4 Lectura de las imágenes.

Para la lectura de los diferentes tipos de imágenes mencionadas anteriormente fueron implementadas las siguientes clases.

- *DICOMScalar4DImageLoader*
- *DICOMVectorial4DImageLoader*
- *DICOMVectorial3DImageLoader*

Las imágenes de entrada se encuentran compuestas por series de cortes bidimensionales como estructura básica. Una imagen 3D es vista como la composición de una de estas series 2D. De manera similar, una imagen 4D no es más que una serie de imágenes 3D en un intervalo de tiempo. En cada una de ellas fue necesario acceder al diccionario de metadatos asociado a las mismas para obtener características como el componente (x, y o z) de un determinado corte de imagen o el tiempo asociado a una imagen en particular. Se cuenta además con la clase *DICOMHeaderReader* para la obtención de los valores de diferentes tags de dicho diccionario. Los archivos fuentes fundamentales de *ITK* que brindan soporte básico para la lectura de imágenes en formato *DICOM* son: *itkGDCMImageIO.h* y *itkGDCMSeriesFileNames.h*.

Los métodos principales de las clases encargadas de la lectura de imágenes son:

- `static Pointer New(void)`
- `void SetDirectory( std::string const &dirName )`
- `void Update ()`
- `Pointer GetOutput()`

El método *New* es una rutina empleada por *ITK* para crear un objeto de una determinada clase a través de una factoría de objetos (*factory*). Esto forma parte del estilo de programación seguido por esta herramienta donde tanto el constructor como el destructor de la clase no son públicos, en tanto debe emplearse el método *New* para inicializar un determinado objeto y el método *Delete* para liberar la memoria del mismo.

Por otra parte, la función *SetDirectory* es empleada para asignar el directorio donde se encuentra la serie de imágenes a leer y es donde a su vez se realiza la lectura de la misma. Luego, el método *Update* actualiza el flujo de datos entre los diferentes objetos vinculados a la lectura y finalmente, *GetOutput* devuelve un puntero (en realidad un puntero inteligente) a la imagen cargada en memoria.

Como parte del código fuente de la aplicación, dentro del directorio *Test* se encuentran disponibles pruebas funcionales para cada una de las clases mencionadas.

2.5 Visualización e interfaz de usuario.

Anteriormente, se ha mencionado la estructura de las imágenes en *ITK*. Sucede que para pasar a trabajar con *VTK* estas imágenes necesitan ser exportadas, pues el formato interno que usan ambas herramientas para almacenar las imágenes no es exactamente igual.

En ocasiones como estas en que es necesario conectar tuberías *ITK* y *VTK* con el objetivo de manipular datos de imágenes, resulta útil considerar una clase que maneje la importación y exportación de datos entre ambas bibliotecas, pues mantener estos dos procesos por separado es desventajoso. Para ello se emplea la clase *vtkKWImage*, la cual tiene presente los elementos anteriores y además permite pasar de un formato a otro sin tener que manipular de forma directa los diferentes tipos de pixeles (Enquobahrie and Ibanez 2007).

Un objeto *vtkRenderer* controla el proceso de dibujado de los diferentes objetos. Este proceso de dibujado es el encargado de convertir la geometría del objeto, la especificación de los colores y la vista de la cámara (Schroeder, Martin et al. 2006), en una imagen concreta.

En el proceso de visualización una de las clases que resulta de vital importancia es *vtkRenderWindow*. Esta es una clase abstracta que especifica el comportamiento de una ventana de dibujo. Una ventana de dibujo es una ventana en la interfaz de usuario donde los objetos de tipo *vtkRenderer* dibujan sus imágenes. Además, provee varios métodos para la sincronización del proceso de dibujado, controlar el tamaño de las ventanas, entre otros.

Por otra parte, la clase *QVTKWidget* permite visualizar datos de *VTK* en un *widget* de *Qt*, lo cual constituye una vía de comunicación entre ambas bibliotecas. Específicamente, la comunicación se logra inicializando el atributo *vtkRenderWindow* perteneciente a la clase *QVTKWidget* con el objeto del mismo tipo creado en el código correspondiente a *VTK* (ver código ejemplo 3). Adicionalmente, *vtkEventQtSlotConnect* hace posible la interacción entre los eventos propios de *VTK* y el sistema de eventos de *Qt*.

La clase principal en la aplicación es *AortaFlowApp* la cual hereda de la clase *QMainWindow*. Esta última es una clase de *Qt* que provee una plataforma para construir aplicaciones de interfaz de usuario. *QMainWindow* cuenta con un diseño propio al que se le pueden añadir diferentes *widgets*: *QToolBars*, *QDockWidgets*, un *QMenuBar*, y un *QStatusBar* (todos presentes en *SAFA*). Este diseño cuenta con un área central que puede ser ocupada con cualesquiera otros *widgets*. A continuación una imagen que representa este diseño.

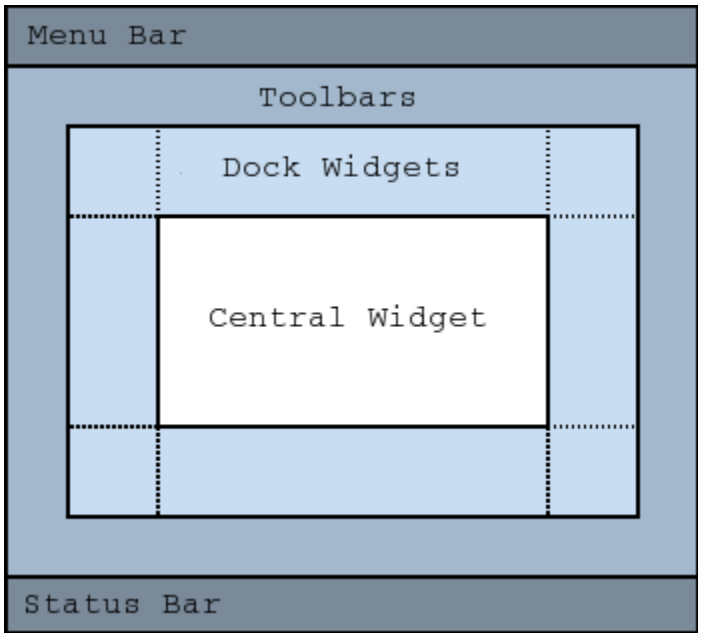


Figura 2.2: Diseño de QMainWindow.

En el presente caso el widget central es ocupado por seis *widgets* personalizados de la clase *Viewer*. El resultado obtenido a partir de esta clase se muestra en la figura 2.3. De manera concreta cuenta con un *QVTKWidget* que se encarga de la visualización de la imagen en cuestión, tres etiquetas para mostrar diferentes informaciones (orientación del corte de la imagen, corte actualmente mostrado, etc.) y una barra deslizante (*slider*) para el desplazamiento entre las diferentes imágenes bidimensionales.

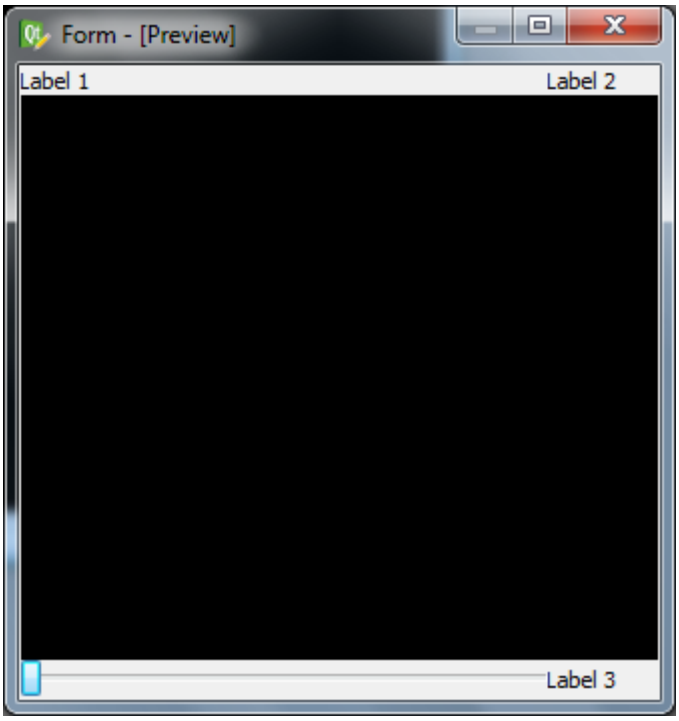


Figura 2.3: Vista previa de un objeto *Viewer*.

Cada uno de estos seis objetos tiene asociado un menú flotante que cuenta con diferentes opciones como: tomar una instantánea del corte que muestra (*Take snapshot*), activar o desactivar las anotaciones en los extremos (*Cornner annotation*), activar o desactivar la escala con leyenda (*Leyend scaler)* o rotar 180 grados el corte actualmente visualizado (*Flip image*).

De esta manera el *widget* central quedaría conformado por dos paneles, uno superior y uno inferior, cada uno de los cuales cuenta con tres visores para la visualización de los cortes clásicos: sagital, axial y coronal. En la figura 2.4 se muestra un ejemplo de una vista por defecto del *widget* central de la aplicación en el que se aprecian cada uno de estos cortes de una imagen en particular.

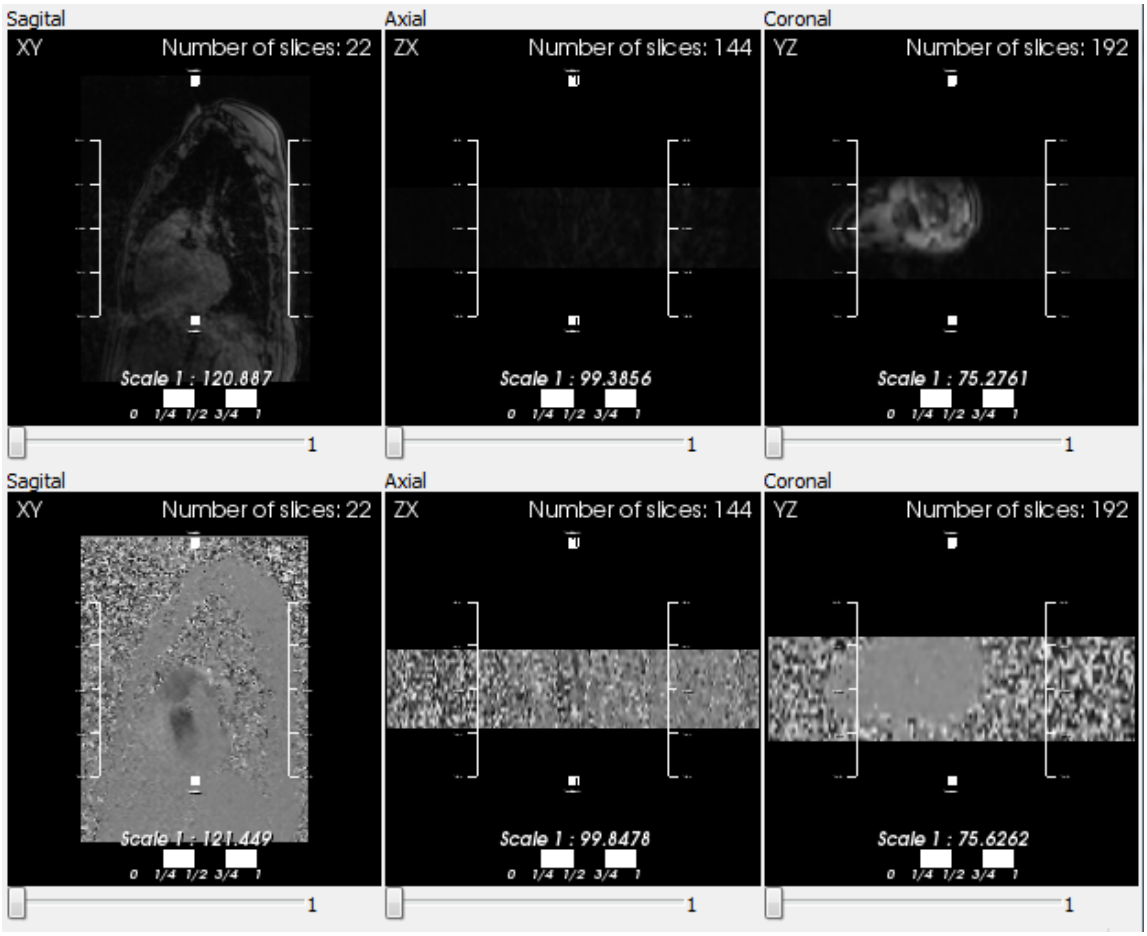


Figura 2.4: Vista por defecto del *widget* central de la aplicación *SAFA*. En el panel superior se muestran las imágenes de magnitud y en el inferior, las imágenes de flujo.

Adicionalmente, existe una clase denominada *Form3D* que tiene como función proveer una vista volumétrica de una imagen tridimensional. Esta permite acercar o alejar la imagen e incluso rotarla en cualquier sentido, para así lograr tener diferentes puntos de vista de una misma imagen. En la figura 3.8 correspondiente al Capítulo 3 del presente Trabajo de Diploma se presenta una vista volumétrica donde se aprecia la aorta. Esta imagen es la misma expuesta en el panel superior de la figura 2.4.

En el capítulo 3 del presente trabajo se confecciona un manual de usuario de la aplicación y se explican en su totalidad las opciones que brinda la aplicación *SAFA*.

2.6 Implementación del algoritmo de segmentación.

2.6.1 Pre-segmentación y extracción de la línea central.

En la implementación del algoritmo de segmentación de la aorta inicialmente se aplica un procesamiento previo o pre-procesado con la intención de obtener mejores resultados finales. Se comienza con la aplicación del filtro implementado en ITK: `MultiScaleHessianSmoothed3DToVesselnessMeasureImageFilter`, cuya declaración es la siguiente.

```
typedef MultiScaleHessianSmoothed3DToVesselnessMeasureImageFilter<
      Scalar3DImageType ,Scalar3DImageType>  MultiScaleVesselnessFilterType;
```

Seguidamente, se toma la salida del filtro anterior como entrada de otro filtro: `FastMarchingImageFilter`, que presenta esta estructura.

```
typedef FastMarchingImageFilter< Scalar3DImageType ,Scalar3DImageType >
      FastMarchingFilterType;
```

Posteriormente, se realiza el cálculo de la línea central. Como se ha sugerido en el capítulo anterior de este Trabajo de Diploma, inicialmente se aplica un algoritmo de esqueletonización (*skeletonization*), en particular se hace uso de la clase `BinaryThinningImageFilter3D` (Homann 2007).

```
typedef BinaryThinningImageFilter3D<Scalar3DImageType, Scalar3DImageType>
      FilterThinningType;
```

Una vez que se obtiene una aproximación de la línea central (*centerline*), en el archivo AF4DCenterLine.h perteneciente al directorio *Segmentation*, se comienza el proceso de construcción del grafo correspondiente en el método:

```
void CenterLine::BuildGraph();
```

Primeramente, se realiza un orden parcial de los puntos de entrada tomando como referencia el valor de la coordenada 'y' y se construye el grafo siguiendo el criterio expuesto en el capítulo anterior (para mayor información ver 1.6.2 Extracción de la línea central.).

El procedimiento principal a partir del punto anterior es el método recursivo:

```
vector<int> CenterLine::AnalyzePaths(vector<int> v, int index);
```

Este recorre todos los caminos que comienzan en el primer punto, guarda los que tengan una mayor longitud en la una variable y mantiene además el menor costo de los caminos guardados. Para una mejor comprensión del algoritmo en general, se dispone con excelente documentación en el código fuente del mismo.

De manera general, la extracción de la línea central de la aorta permite:

- Conocer el trayecto exacto de la aorta.
- Mejorar la segmentación obtenida previamente.
- Encontrar semillas a lo largo de la aorta, lo cual resulta necesario para poder aplicar algoritmos de visualización.
- Encontrar cortes a lo largo de la aorta de manera automática.

2.6.2 Diseño e implementación de la tubería de segmentación.

La implementación de la tubería de segmentación de la aplicación se encuentra encapsulada dentro de la clase *AortaSegmentationPipeline* declarada dentro del directorio *Segmentation*. En ella se hace emplean diferentes filtros propios de *ITK* y otros que son parte del código fuente de la aplicación *SAFA*, presentes también en el propio directorio.

Con el objetivo de implementar el primer paso de la tubería de segmentación de la aorta comentada en el Capítulo I del presente trabajo, se aplica inicialmente el filtro `MinimumMaximumImageCalculator` con el propósito de encontrar el valor mínimo y máximo de intensidad de la imagen original. Este filtro es una clase plantilla (*template*) sobre el tipo de esta imagen, que este caso se instancia de la siguiente manera:

```
typedef MinimumMaximumImageCalculator<Scalar3DImageType>
      MinMaxCalculatorType;
```

Seguidamente, empleando los valores encontrados anteriormente, se usa el filtro:

```
typedef BinaryThresholdImageFilter<Scalar3DImageType, Scalar3DImageType>
      BinaryThresholdFilterType,
```

Tomando como valor mínimo de umbral `min = (max - min) * _percent + min` y valor máximo `max` (donde `min` y `max` representan los valores de intensidad mínimo y máximo respectivamente, y `percent = 0.025`).

Para una mayor comprensión de la estructura completa de la tubería de segmentación, se muestra el diagrama de flujo de datos entre los filtros y procedimientos empleados en la misma.

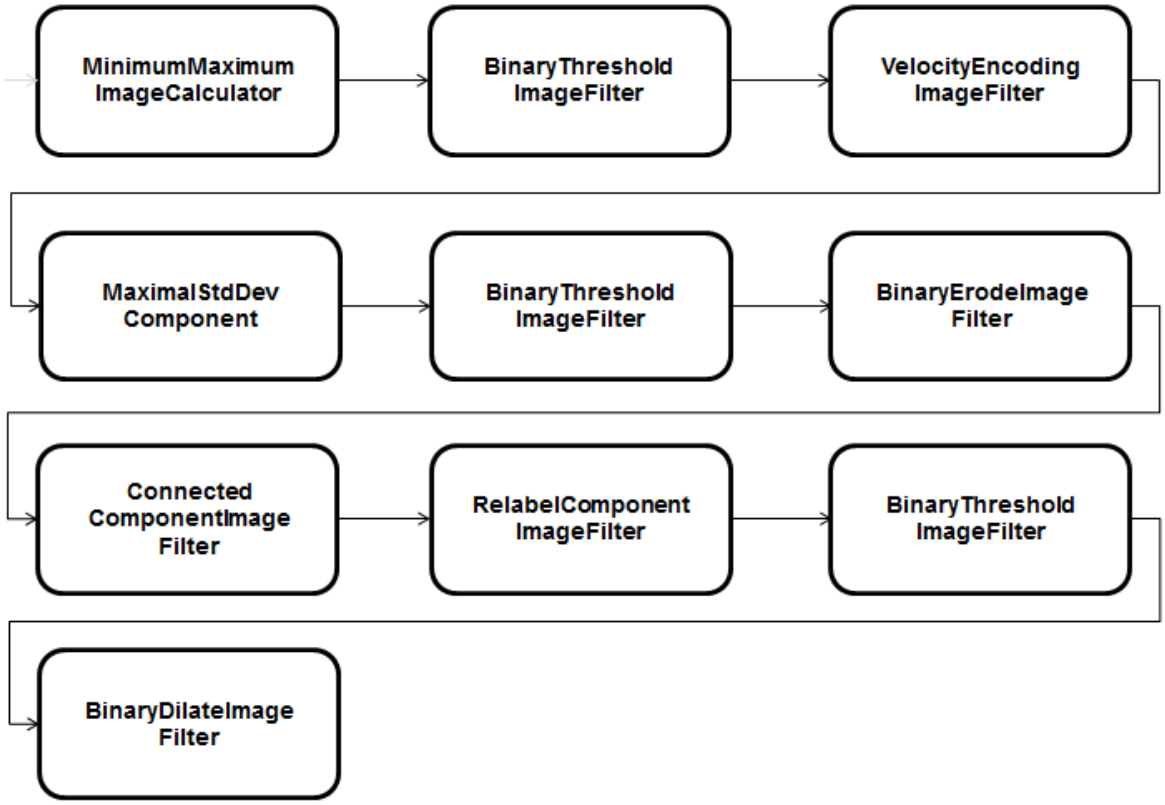


Figura 2.6: Diagrama del flujo de datos a través de la tubería de segmentación.

A continuación se ofrece la declaración de los restantes filtros y funciones mostrados en la figura 2.6.

```
typedef VelocityEncodingImageFilter< Vector4DImageType, Vector4DImageType >
            VelocityEncodingImageFilterType;

Scalar3DImageType::Pointer MaximalComponentStdDev
            (Scalar3DImageType::Pointer noiseMask,
             Vector4DImageType::Pointer input );

typedef BinaryBallStructuringElement< PixelType, 3 >
            StructuringElementType;
```

```
typedef BinaryErodeImageFilter<Scalar3DImageType, Scalar3DImageType,
            StructuringElementType> BinaryErodeImageFilterType;

typedef ConnectedComponentImageFilter<Scalar3DImageType,
            Integer3DImageType>  ConnectedComponentImageFilterType;

typedef RelabelComponentImageFilter<Integer3DImageType, Scalar3DImageType>
            RelabelComponentImageFilterType;
```

El flujo de datos comienza en el primer elemento de la figura 2.6 (`MinimumMaximumImageCalculator`) y a partir de este, la entrada correspondiente a un elemento se establece mediante el método `SetInput()`, a la vez que la salida se obtiene a través de la invocación de `GetOutput()`, por lo que resulta común el empleo de la sentencia `element2->SetInput(element1->GetOutput())`.

2.7 Análisis de los resultados.

El análisis cuantitativo de la calidad de una segmentación resulta una tarea pendiente en el campo del procesamiento digital de imágenes, pues no existe un estándar de oro o verdad absoluta en el tema (Lehmann, Beier et al.). De manera particular, se evalúa la delineación de los objetos manipulados. Otro de los desafíos que se presenta es la no reproducibilidad de las delineaciones realizadas por expertos humanos, por lo que un análisis de calidad requeriría de una base de casos de segmentaciones manuales realizadas por especialistas médicos.

Por el momento se puede decir que los resultados obtenidos son satisfactorios. Los obtenidos en estudios previos (Zhao, Zhang et al. 2006) al presente Trabajo de Diploma ofrecen garantías, aunque ninguno resulta concluyente. Además, se cuenta con la bibliografía requerida para en un futuro cercano abordar de manera eficiente dicha tarea.

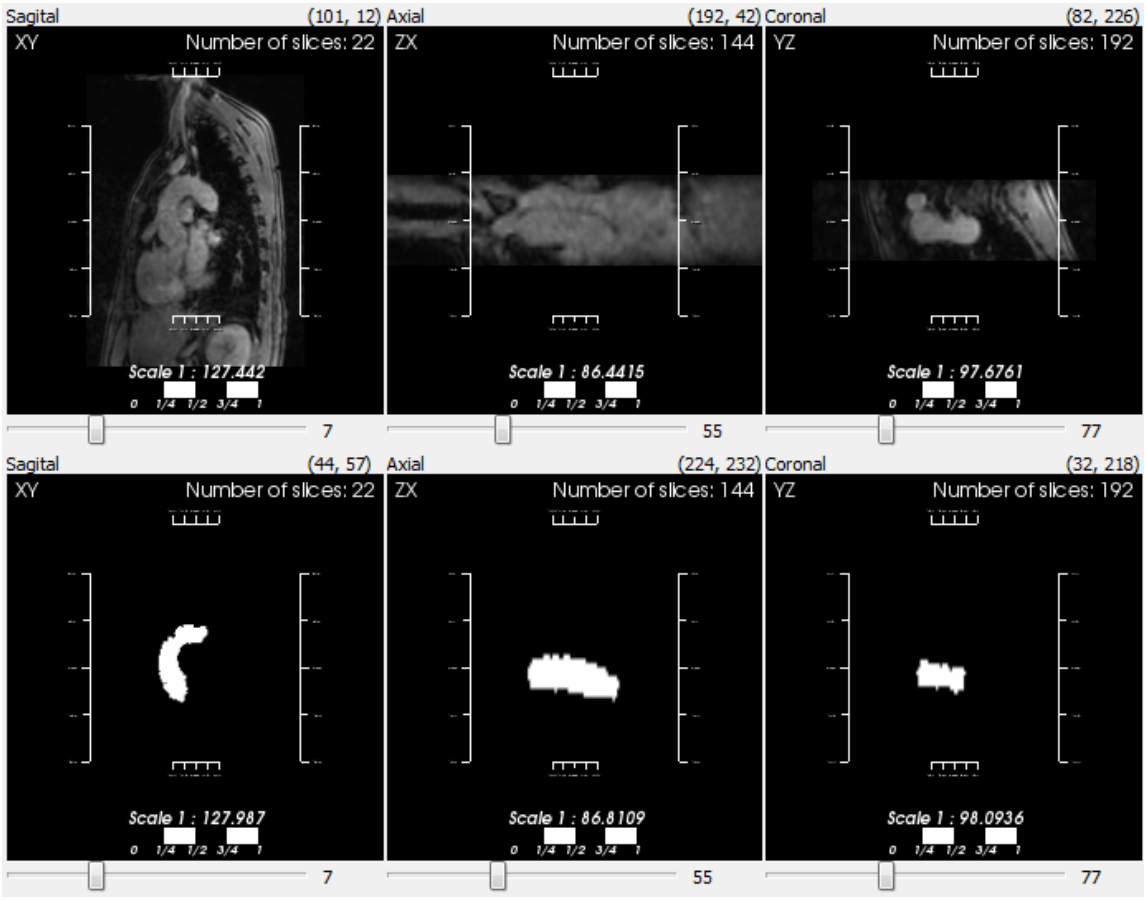


Figura 2.7: Visualización mostrada por la aplicación *SAFA* luego de la segmentación donde se observan diferentes cortes.

2.8 Conclusiones parciales.

En el capítulo que concluye se realizó una descripción general de las clases fundamentales que conforman la aplicación *SAFA*, aunque no en su totalidad debido a que son varias y el objetivo trazado fue destacar los principales componentes de la arquitectura del software (como es el caso de las clases *DICOMScalar4DImageLoader*, *AortaFlowApp*, etc.). Si se precisó detalles del proceso general de segmentación, en particular se especificó sobre la composición de la tubería (diagrama de flujo de datos, figura 2.6) y acerca de los filtros y procedimientos vinculados a la misma. Por otra parte, las diferentes herramientas empleadas durante la implementación de la aplicación fueron también descritas.

Como se explicó en el apartado anterior, se puede afirmar que los resultados obtenidos por la aplicación *SAFA* son los deseados (teniendo presente los demás puntos señalados), pues en definitiva se logran los objetivos fundamentales de segmentar la aorta y obtener la línea central de la misma con la menor intervención posible del usuario.

CAPÍTULO 3: MANUAL DE USUARIO DEL SOFTWARE *SAFA*.

3.1 Generalidades.

La aplicación *SAFA* se realizó con el objetivo fundamental de realizar una segmentación automática de la aorta. La misma puede ser compilada para diferentes entornos de trabajo (sistemas operativos) como *Windows*, *Mac* y *Linux*, gracias a que las herramientas de software empleadas en su implementación son todas multiplataforma.

El software fue completamente desarrollado en lenguaje *C++* y cuenta con la siguiente estructura de directorios:

- *Common*: se encuentran definidas clases y rutinas de propósito general, así como la definición de diferentes tipos de datos empleados en la aplicación.
- *GUI*: se encuentran detalladas las clases de la interfaz de usuario (en particular, *AortaFlowApp*).
- *IO*: se hallan especificadas las clases y funciones vinculadas al mecanismo de entrada y salida del software.
- *Segmentation*: se localizan las clases, filtros y rutinas en general relacionadas con el proceso de segmentación (por ejemplo, *AortaSegmentationPipeline*, *BinaryThinningImageFilter3D*, entre otras).
- *System*: se encuentra el archivo *AortaFlowMain.cxx* que constituye el punto de entrada a la aplicación.
- *Temp*: se hallan disponibles archivos que estarán en este lugar de manera temporal, ya sea porque serán eliminados completamente del código fuente del software (código en desuso) o porque aún no se han terminado de implementar (código sin comprobar).

Nota: También dentro del directorio del código fuente de la aplicación *SAFA* se encuentra un archivo nombrado *CMakeLists.txt*, este contiene la configuración requerida por el programa *CMake* a la hora de generar el proyecto o solución a partir del código fuente, en particular se encuentran definidas las diferentes dependencias del software (ver Anexo I). No se debe eliminar este archivo.

La aplicación se ejecuta en el archivo *SAFA.exe* y no presenta ninguna dependencia externa, a excepción de las bibliotecas de enlace dinámico (*.*dll*) que son parte del mismo directorio. Las bibliotecas de las cuales depende el software pertenecen a:

- *ITK* (http://www.itk.org)
- *VTK* (http://www.vtk.org)
- *Qt* (http://www.qt-project.org)

3.2 Ventana principal.

Al iniciar la aplicación esta muestra una ventana principal conformada por un menú, una barra de herramientas y un espacio central vacío, similar a la que muestra debajo en la figura 3.1.

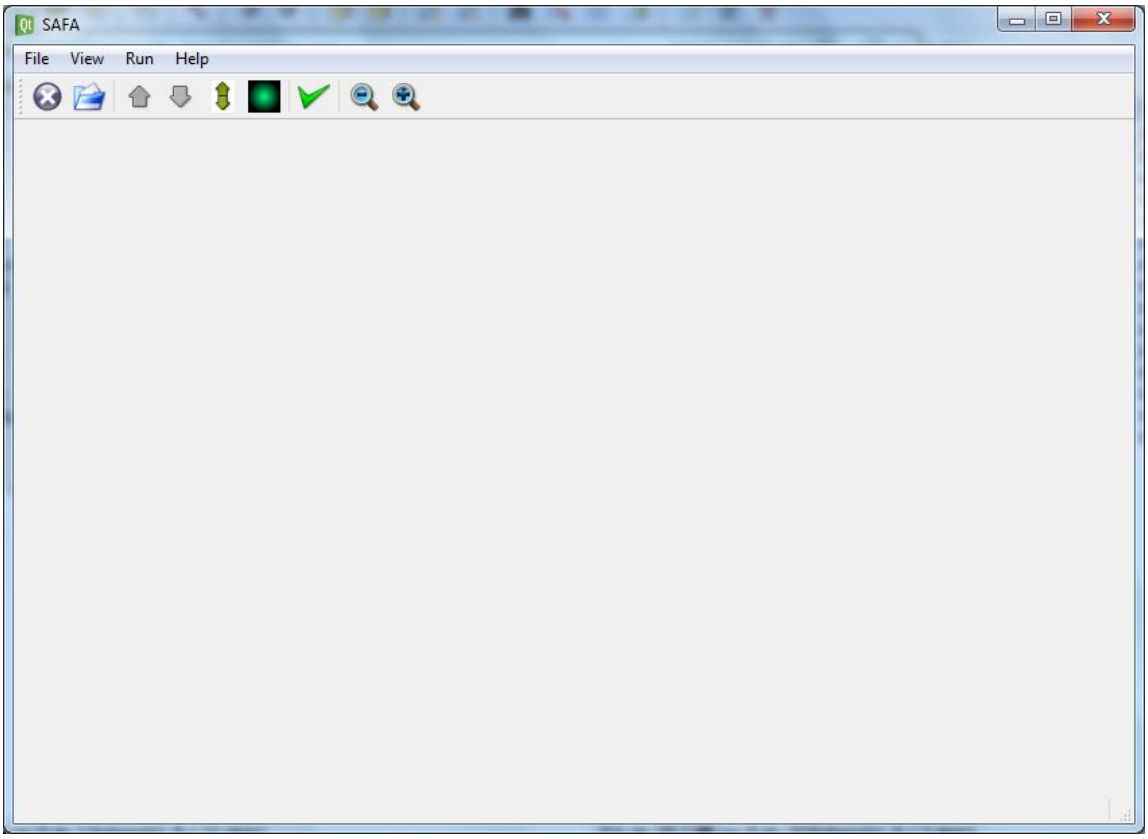


Figura 3.1: Vista de la ventana principal al iniciar la aplicación *SAFA*.

Como se puede observar el software *SAFA* cuenta con cuatro menús principales: *File*, *View*, *Run* y *Help*. Seguidamente, se ofrece información acerca de la función de cada uno de ellos.

3.3 El menú *File*.

El menú *File* contiene acciones generales de la aplicación como: *Load DICOM series*, *Close current serie* y *Quit* según se expone a continuación en la figura 3.2.

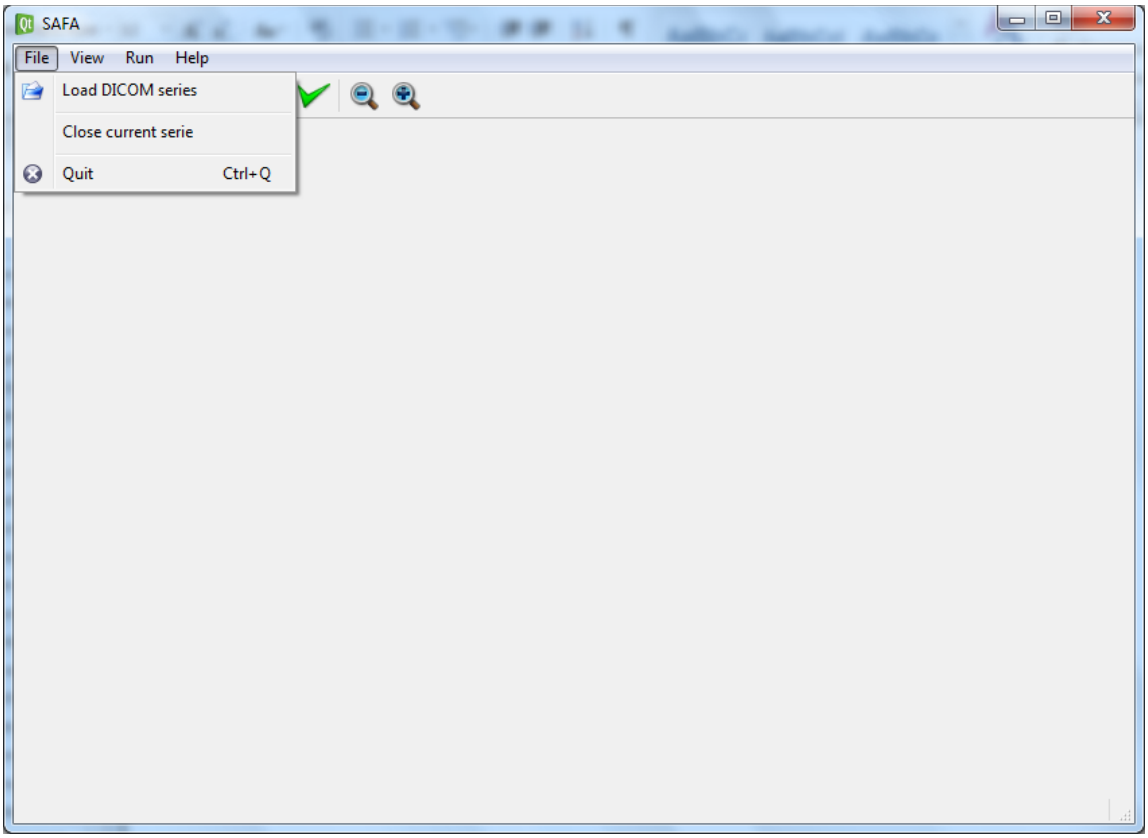


Figura 3.2: Menú *File* desplegado en la ventana principal.

La opción *Load DICOM series* se usa para especificar el directorio en el cual se encuentran las imágenes a ser cargadas en *SAFA*. Actualmente emplea el diálogo nativo del sistema operativo usado para abrir una determinada carpeta. En la figura 3.3 se muestra esta opción. Por defecto este diálogo se inicia en el subdirectorio *Data* de la aplicación, en el que a su vez se encuentran dos subdirectorios: *flow* y *mag*; estos contienen las imágenes *DICOM* de flujo (velocidad codificada) y magnitud (información

morfológica) respectivamente. En caso de que el directorio especificado por el usuario no tenga esta estructura se origina el mensaje de error ejemplificado en la figura 3.4.

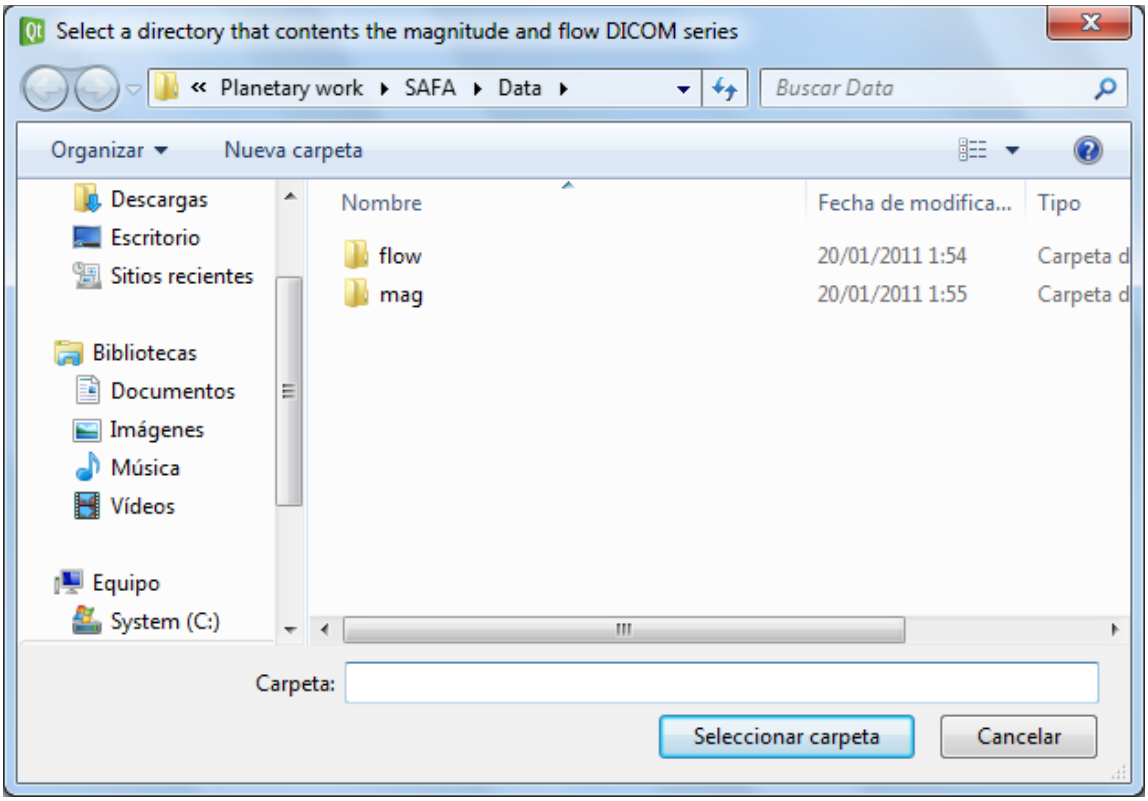


Figura 3.3: Menú *File* desplegado en la ventana principal.

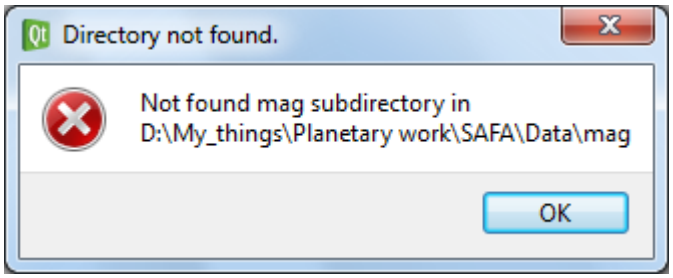


Figura 3.4: Mensaje de error mostrado en caso de que el directorio seleccionado por el usuario no contenga alguno de los subdirectorios *mag* o *flow*.

Si efectivamente, el directorio elegido por el usuario presenta la estructura mencionada anteriormente, entonces la aplicación comienza la el proceso de lectura y extracción de las imágenes. En la barra de estado (parte inferior) se va mostrando el progreso de estas operaciones como se presenta en la figura 3.5. Una vez terminado, se puede observar el

mensaje "*Ready*" en la parte derecha de la barra de progreso. A partir de este momento se muestran dos paneles con tres visores cada uno en los que se muestran los cortes sagital, axial y coronal de ambos tipos de imágenes (magnitud y flujo). Además, también aparece un *widget* a la izquierda de la ventana principal (conocido como *DockWidget*) en el que se muestra diferente información (ver figura 3.6). Este último tiene la particularidad de que puede ser arrastrado de esta posición y colgado en otra región de la ventana, o simplemente dejado de manera flotante sobre la misma. En caso de que alguno sea cerrado, se puede volver a mostrar haciendo clic secundario sobre la barra de herramientas y activándolo en el menú mostrado.

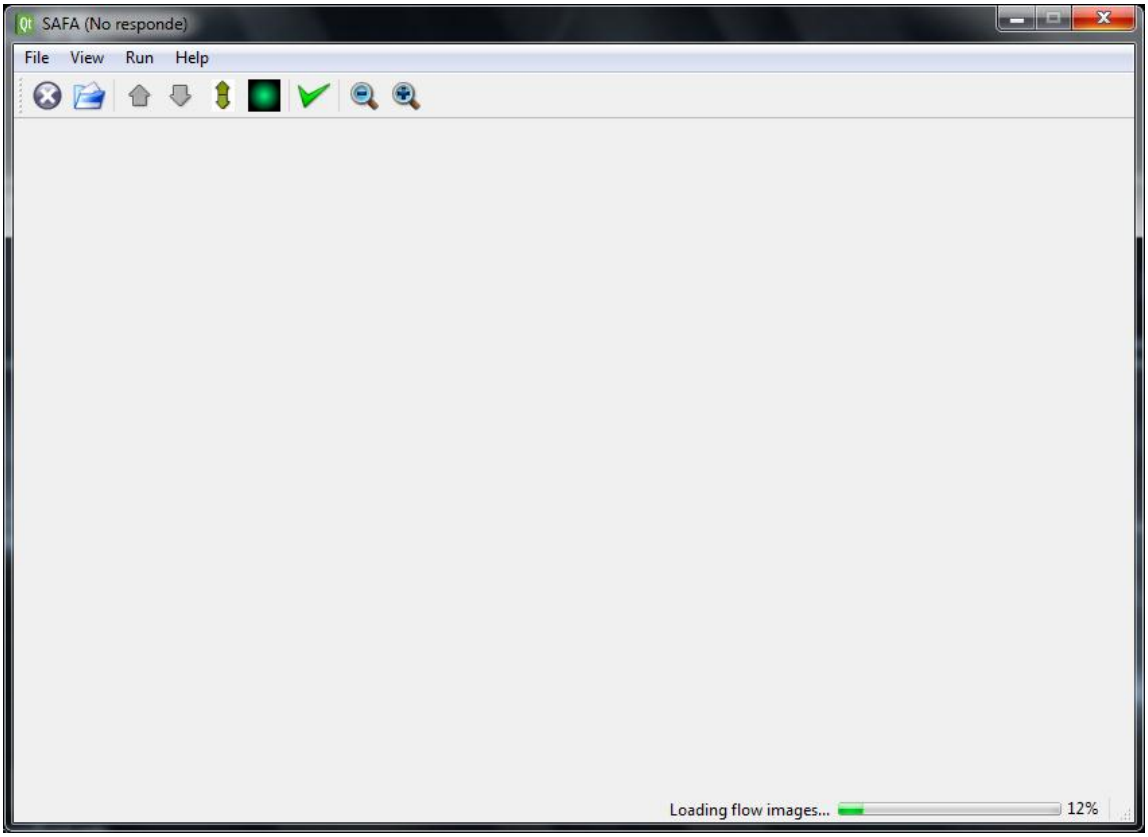


Figura 3.5: Ventana principal mostrando el progreso del proceso de lectura y extracción de las imágenes.

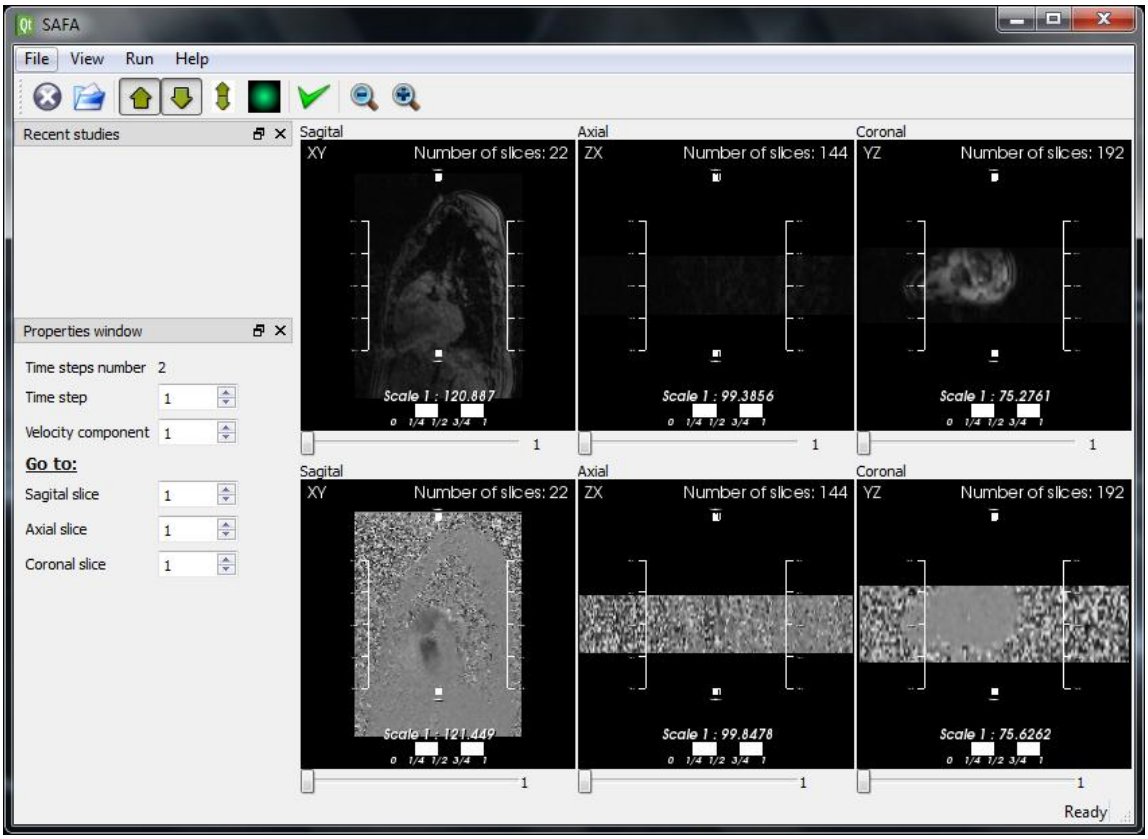


Figura 3.6: Vista de la ventana principal luego del progreso del proceso de lectura y extracción de las imágenes.

Otra de las opciones presentes en el menú *File* es *Close current serie*, esta simplemente cierra todas las visualizaciones y retorna la aplicación al estado en que se inicia, lista para la carga de otra posible serie de imágenes.

Finalmente, se encuentra la opción *Close* que como es de imaginar cierra el software *SAFA*, liberando previamente los recursos asociados al mismo.

En la barra de herramientas se presentan accesos directos a la primera y última de estas opciones (constituyen los dos primeros).

3.4 El menú *View*.

El menú *View* forma también parte del menú principal. Las opciones que este ofrece están relacionadas con operaciones de visualización y personalización de la interfaz de usuario. En concreto, se encuentran las siguientes opciones: *Magnitude frame (superior)*,

Flow frame (inferior), *3D View*, *Syncronize both frames* y el submenú *Window* que a su vez contiene las opciones *Full screen* y *Normal* (ver figura 3.7).

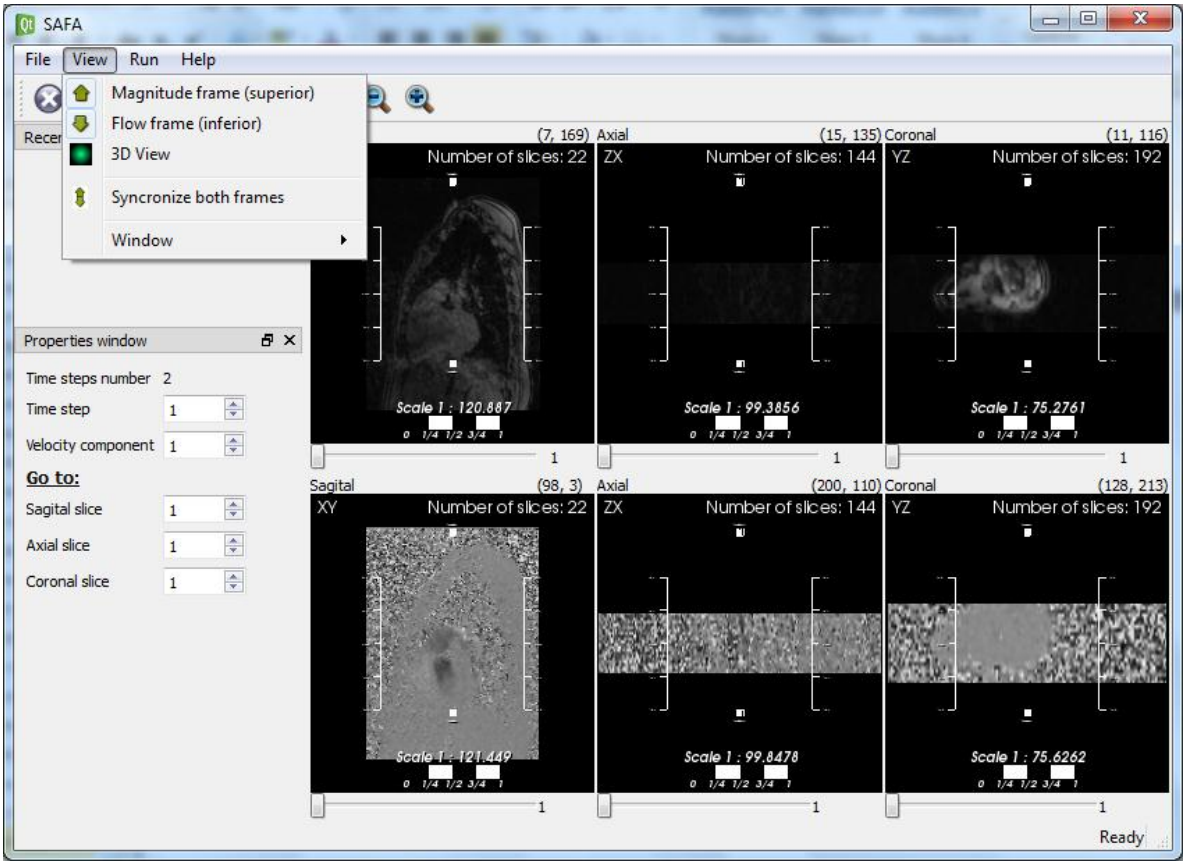


Figura 3.6: Vista de la ventana principal mostrando el menú *View* desplegado.

Como se hace evidente las dos primeras opciones se encuentran por defecto marcadas, lo cual se corresponde con el hecho de que tanto el marco (panel) superior como el inferior están visibles. Con estas dos opciones se controla la visibilidad de ambos paneles, lo que posibilita llegar a obtener visualizaciones más cercanas como la ejemplificada con la figura 3.7, donde se ha cerrado el panel inferior y ambos *dockwidgets*.

Seguidamente, se presenta la opción *3D View*, la que permite obtener una vista volumétrica de una determinada imagen de magnitud (ver figura 3.8). Esta ventana permite operaciones de acercamiento y alejamiento (*zoom in and zoom out*) de la visualización mediante el empleo de la rueda (*wheel*) del ratón (*mouse*). Para restaurar el acercamiento provisto por defecto se debe presionar la tecla "r".

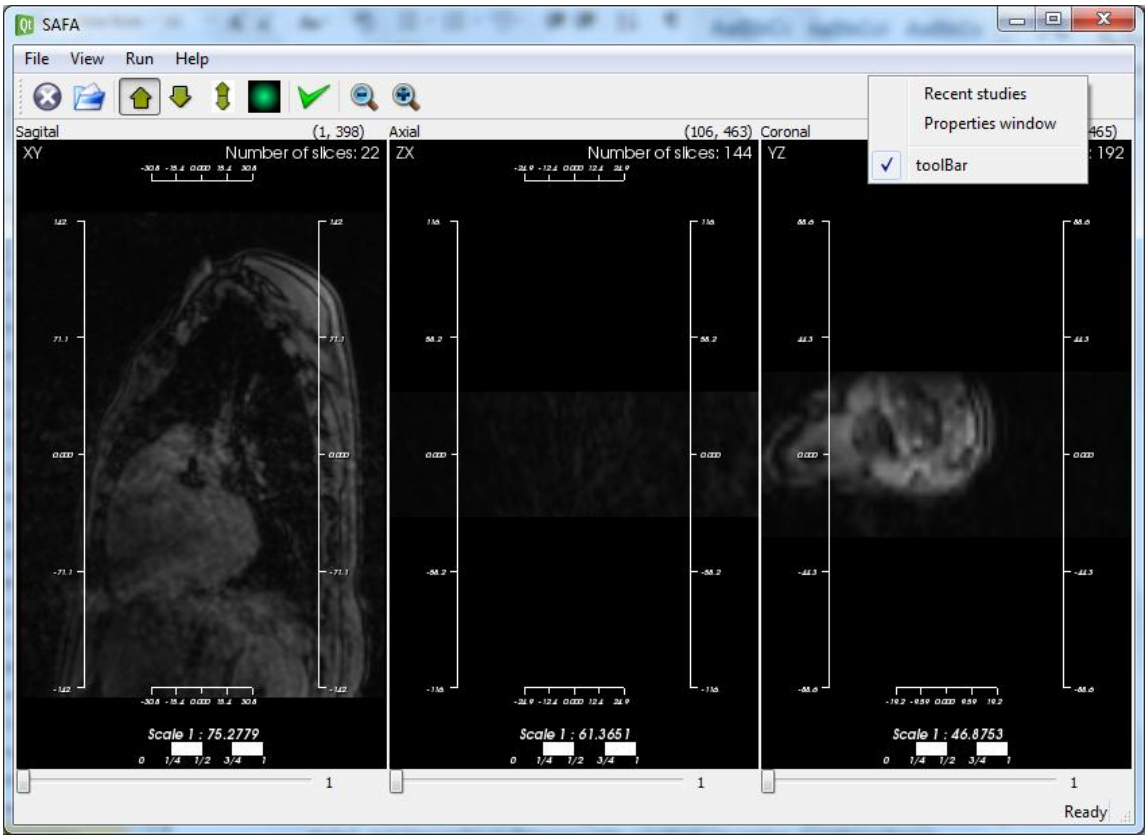


Figura 3.7: Vista de la ventana principal donde solo se expone el marco de las imágenes de magnitud. Además, se muestra desplegado el menú flotante que permite controlar la visibilidad de los *dockwidgets* y la barra de herramientas.

También, admite operaciones de rotación 3D de la imagen en cualquiera de los 360 grados. Esta última manipulación se consigue manteniendo presionado el botón izquierdo del ratón y seguidamente, moviendo el puntero en la dirección deseada.
A continuación, se encuentra la acción *Syncronize both frames*, que por defecto está desactivada. Con la activación de esta opción se sincronizan cada uno de los visores superiores con su respectivo visor perteneciente al panel inferior. Esto quiere decir, que al mover la barra deslizante (*slider*) correspondiente a uno de ellos, se moverá también la otra barra deslizante correspondiente; lo que hace posible que se pueda ir desplazándose por los diferentes cortes (*slices*) en ambas vistas.

Finalmente, las acciones *Full screen* y *Normal* como es de esperar, hacen que la ventana se muestre a pantalla completa y en estado normal, respectivamente.

Figura 3.8: Vista volumétrica de una imagen de magnitud donde se aprecia la aorta.

3.5 El menú *Run*.

Dentro del menú *Run* solo se encuentra la opción *Segment*, cuya función es realizar la segmentación de la imagen original. Al activar esta acción se puede ver la barra de progreso en la parte derecha inferior, donde se muestra además información relativa al filtro que se aplica en ese instante de tiempo. Una vez terminado este proceso, se visualiza en el panel inferior el resultado de la segmentación (ver figura 3.9).

3.5 El menú *Help*.

En el menú *Help* se hallan las acciones *SAFA Help*, *About…* y *About Qt…*, donde cada uno de estos muestra un cuadro de mensaje. El primero le informa al usuario que la ayuda se encuentra disponible dentro del directorio *Doc*, el segundo brinda una síntesis de lo que es *SAFA* y el último brinda información acerca de la versión de *Qt* empleada en el desarrollo del software.

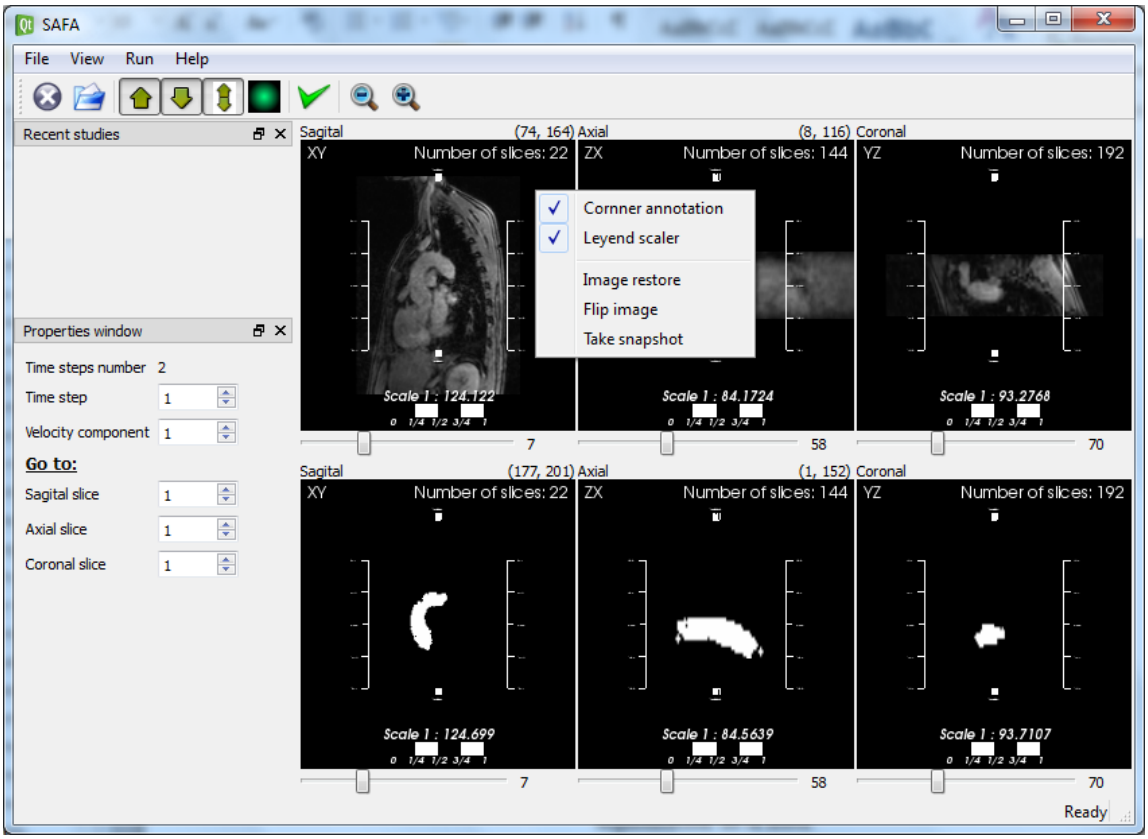


Figura 3.9: Vista de la aplicación donde se observa cortes del resultado de la segmentación de la aorta. Además, se muestra el menú flotante de los visores desplegado.

3.6 El menú flotante de los visores.

En la figura 3.9 se puede observar también desplegado el menú flotante asociado a los visores de los cortes. Por defecto, las acciones *Cornner annotation* y *Leyend scaler* se encuentran activadas, estas se corresponden con las anotaciones de las esquinas de cada visor (por ejemplo, en el sagital superior, serían *XY* y *Number of slices: 22*) y sus escalas, respectivamente.

Las opción *Image restore* se emplea con el propósito de restaurar la posición de la imagen en el visor y *Flip image*, para rotarla 180 grados en caso de que se quiera invertir. Finalmente, *Take snapshot* toma una instantánea del corte visualizado y lo guarda en un archivo con formato *PNG* (Portable Network Graphics) dentro del directorio de la aplicación (precisamente, con nombre snapshot.png).

CONCLUSIONES

Se desarrolló una aplicación (*SAFA*) que permite realizar una segmentación automática de la aorta y la extracción de la línea central usando fundamentalmente las herramientas de software *ITK*, *VTK* y *Qt*. Dicha aplicación fue implementada usando la plataforma de desarrollo *Eclipse*; tomando como lenguaje de desarrollo a *C++*, aprovechando el hecho de que las bibliotecas mencionadas anteriormente están todas implementadas usando el mismo, por lo que se contó con la integración requerida.

Fue diseñada e implementada la tubería de segmentación automática de la aorta siguiendo el paradigma de flujo de datos empleado por *ITK* donde la salida de un filtro se transforma en la entrada del próximo en la tubería, evitando así posibles copias intermedias de datos que podrían provocar una disminución considerable del rendimiento general del software.

Se propusieron diferentes características 3D del flujo sanguíneo a través de la aorta que permitirían a los expertos en medicina manejar una información más integral acerca del comportamiento general del mismo; a la vez que se valoraron posibles representaciones del flujo de la sangre, algunas de las cuales podrían ser incorporadas a la aplicación.

RECOMENDACIONES

- Validar la delineación obtenida por la aplicación a través de su comparación con otras realizadas por expertos humanos.
- Estudiar las posibilidades de mejorar la segmentación lograda de la aorta mediante el empleo de la información provista por la línea central de la misma.
- Incorporar representaciones en forma de histogramas al software con el propósito de incrementar la información visual en la interfaz.

REFERENCIAS BIBLIOGRÁFICAS

Association, N. E. M. (2009). Digital Imaging and Communications in Medicine (DICOM), National Electrical Manufacturers Association.

Bærentzen, J. A. (2000) On the implementation of fast marching methods for 3D lattices.

Eckel, B. (2000). Thinking in C++, Prentice Hall. **Volume 1-2**.

Enquobahrie, A. and L. Ibanez (2007) Seamless VTK-ITK pipeline connection for image data handling.

Gasser, M. Transfer functions for Volume Rendering applications and implementation results with the VTK.

Ho, S., Y. Kim, et al. Medical image segmentation by 3-D level set evolution.

Homann, H. (2007) Implementation of a 3D thinning algorithm.

Ibáñez, L., W. Schroeder, et al. (2005). The ITK Software Guide.

Kitware, I. (2006). The VTK User's Guide, Kitware, Inc.

Kolluri, R., J. R. Shewchuk, et al. (2004) Spectral Surface Reconstruction from Noisy Point Clouds.

Lee, T.-C., R. L. Kashyap, et al. (1994). "Building skeleton models via 3D medial surface/axis thinning algorithms."

Lehmann, T. M., D. Beier, et al. Segmentation of medical images combining local, regional, global, and hierarchical distances into a bottom-up region merging scheme.

Luenberger, D. G. and Y. Ye (2008). Linear and Nonlinear Programming, Springer.

Markl, M. (2006) Velocity Encoding and Flow Imaging.

Palágyi, K., E. Sorantin, et al. (2001) A Sequential 3D Thinning Algorithm and Its Medical Applications.

Preim, B. and D. Bartz (2007). "Visualization in medicine. Theory, algorithms, and applications."

Schroeder, W., K. Martin, et al. (2006). The Visualization Toolkit, Prentice-Hall, Inc.

Stroustrup, B. (1997). The C++ Programming Language. AT&T Labs, Murray Hill, New Jersey.

Xavier, M., A. Lalande, et al. (2007) Dynamic 4D Blood Flow Representation in the Aorta and Analysis from Cine-MRI in Patients.

Yang, F., W. Zuo, et al. (2008) 3D Cardiac MRI Data Visualization Based on Volume Data Preprocessing and Transfer Function Design

Zhao, F., H. Zhang, et al. (2006) Automated 4D Segmentation of Aortic Magnetic Resonance Images.

ANEXO I

Configuración del archivo CMakeLists.txt.

```
cmake_minimum_required(VERSION 2.8)

PROJECT(AortaFlowApp)
FIND_PACKAGE(ITK)
IF(NOT ITK_DIR)
  MESSAGE(FATAL_ERROR "Please set ITK_DIR.")
ENDIF(NOT ITK_DIR)
FIND_PACKAGE(VTK)
IF(NOT VTK_DIR)
  MESSAGE(FATAL_ERROR "Please set VTK_DIR.")
ENDIF(NOT VTK_DIR)

INCLUDE(${ITK_USE_FILE})
INCLUDE(${VTK_USE_FILE})

# use what QVTK built with
SET(QT_MOC_EXECUTABLE ${VTK_QT_MOC_EXECUTABLE} CACHE FILEPATH "")
SET(QT_UIC_EXECUTABLE ${VTK_QT_UIC_EXECUTABLE} CACHE FILEPATH "")
SET(QT_QMAKE_EXECUTABLE ${VTK_QT_QMAKE_EXECUTABLE} CACHE FILEPATH "")
SET(DESIRED_QT_VERSION ${VTK_DESIRED_QT_VERSION} CACHE FILEPATH "")

FIND_PACKAGE(Qt4 REQUIRED)
INCLUDE(${QT_USE_FILE})

# Use the include path and library for Qt that is used by VTK.
INCLUDE_DIRECTORIES(
  ${QT_INCLUDE_DIR}
  ${CMAKE_CURRENT_BINARY_DIR}
  ${CMAKE_CURRENT_SOURCE_DIR}
)
#############################################################################
SET (MainSrc
    System/AortaFlowMain.cxx
    )

SET (AortaFlowAppHeaders
    Common/AortaUtility.h
    Common/vtkKWImage.h
    Common/vtkKWImageIO.h
    Common/itkDICOMTags.h
    Common/itkImageTypes.h
    Common/itkReaderEvents.h
    Common/HeuristicBloodBehaviorDetection.h
    Common/ImageCursor.h

    Segmentation/AF4DCenterLine.h
    Segmentation/itkBinaryThinningImageFilter3D.h
    Segmentation/WriteImageRoutines.h
```

```
    Segmentation/AortaSegmentationPipeline.h
    Segmentation/itkVelocityEncodingImageFilter.h
    Segmentation/itkVectorMagnitudeMinimumMaximumImageCalculator.cxx

    IO/DICOMScalar4DImageLoader.h
    IO/DICOMVectorial3DImageLoader.h
    IO/DICOMVectorial4DImageLoader.h
    IO/AortaSeriesImporter.h
    IO/DICOMHeaderReader.h
    IO/ObserverReader.h
    IO/itkDICOM4DVPixelReader.h

    GUI/ViewerSynchronize.h
    GUI/AortaFlowApp.h
    GUI/Form3D.h
    GUI/Viewer.h
    )

SET (AortaFlowAppSrcs
    Common/AortaUtility.cxx
    Common/vtkKWImage.cxx
    Common/vtkKWImageIO.cxx
    Common/HeuristicBloodBehaviorDetection.cpp
    Common/ImageCursor.cxx

    Segmentation/AF4DCenterLine.cxx
    Segmentation/itkBinaryThinningImageFilter3D.txx
    Segmentation/WriteImageRoutines.cxx
    Segmentation/AortaSegmentationPipeline.cxx
    Segmentation/itkVelocityEncodingImageFilter.cxx
    Segmentation/itkVectorMagnitudeMinimumMaximumImageCalculator.cxx

    IO/DICOMScalar4DImageLoader.cxx
    IO/DICOMVectorial3DImageLoader.cxx
    IO/DICOMVectorial4DImageLoader.cxx
    IO/AortaSeriesImporter.cxx
    IO/DICOMHeaderReader.cxx
    IO/itkDICOM4DVPixelReader.cxx

    GUI/ViewerSynchronize.cxx
    GUI/AortaFlowApp.cxx
    GUI/Form3D.cxx
    GUI/Viewer.cxx
    )

SET (UIS
    GUI/AortaFlowApp.ui
    GUI/Form3D.ui
    GUI/Viewer.ui
    )

SET (AortaFlowAppResources
    GUI/ResourcesFile.qrc
    )
```

```
SET (TestAll
    Test/TestAll.cxx
    Test/TestLoaders.cxx

    Test/TestDICOMVectorial3DImageLoader.h
    Test/TestDICOMScalar4DImageLoader.h
    Test/TestDICOMVectorial4DImageLoader.h
    Test/TestAF4DCenterLine.h

    )

#############################################################################

QT4_WRAP_UI(UIHeaders ${UIS})
QT4_WRAP_CPP(MOCSrcs ${AortaFlowAppHeaders} )
QT4_ADD_RESOURCES(ResourceSrcs ${AortaFlowAppResources})

ADD_DEFINITIONS(-DQT_GUI_LIBS -DQT_CORE_LIB -DQT3_SUPPORT)
SET_SOURCE_FILES_PROPERTIES(${AortaFlowAppSrcs} PROPERTIES  OBJECT_DEPENDS
"${UIHeaders}")


# It's nice to have the ui in the windows project file...just double click on
it
# and designer comes up on that ui file :)
IF (${CMAKE_BUILD_TOOL} MATCHES "msdev")
  SET (AortaFlowAppSrcs ${AortaFlowAppSrcs} ${UIS})
ENDIF (${CMAKE_BUILD_TOOL} MATCHES "msdev")
IF (${CMAKE_BUILD_TOOL} MATCHES "devenv")
  SET (AortaFlowAppSrcs ${AortaFlowAppSrcs} ${UIS})
ENDIF (${CMAKE_BUILD_TOOL} MATCHES "devenv")

ADD_EXECUTABLE( AortaFlowApp MACOSX_BUNDLE ${AortaFlowAppSrcs} ${UISrcs}
${MOCSrcs} ${ResourceSrcs} ${MainSrc})

TARGET_LINK_LIBRARIES( AortaFlowApp
    QVTK
    ${QT_LIBRARIES}
    vtkRendering
    vtkVolumeRendering
    vtkGraphics
    vtkIO
    vtkCommon
    ITKBasicFilters
    ITKCommon
    ITKIO
)
```

ANEXO II

Ejemplo de segmentaciones obtenidas con el algoritmo presentado en este Trabajo de Diploma.

Ejemplo 1

Ejemplo 2