

UNIVERSIDAD POLITECNICA DE VALENCIA
ESCUELA POLITECNICA SUPERIOR DE GANDIA
I.T. TELECOMUNICACIÓN (SISTEMAS DE TELECOMUNICACIÓN)



UNIVERSIDAD
POLITECNICA
DE VALENCIA



ESCUELA POLITECNICA
SUPERIOR DE GANDIA

**“Herramienta software para la
segmentación y visualización 3D de
hígados reales a partir de imágenes de
escaneos 3D no invasivos”**

TRABAJO FINAL DE CARRERA

Autor/es:

M^a Teresa Martínez Latorre

Director/es:

Valery Naranjo Omedo

M^a José Rupérez Moreno

GANDIA, 2010

Índice general

Índice general	2
Índice de figuras	6
1. Introducción	11
1.1. Objetivos	11
1.2. Entorno	12
1.3. Planteamiento	14
1.4. Estado del arte	16
1.4.1. Anatomía del hígado	16
1.4.2. Métodos de adquisición de imágenes	18
1.4.2.1. Métodos de adquisición de imágenes no invasivos	18
1.4.3. Formato DICOM	20
1.4.4. Segmentación	22
1.4.4.1. Definición de segmentación	22
1.4.4.2. Técnicas de segmentación	22
1.4.4.2.1. Técnicas de segmentación basadas en píxeles	23
1.4.4.2.2. Técnicas de segmentación basadas en regiones	23

1.5. Organización	25
2. Herramientas	26
2.1. Visual Studio 2008	26
2.1.1. Instalación de Visual Studio 2008	27
2.2. Qt	28
2.2.1. Instalación de Qt	28
2.2.2. Integración Qt con Visual Studio 2008	29
2.3. CMake	30
2.3.1. Instalación de CMake	30
2.4. ITK (Insight Segmentation and Registration Toolkit)	31
2.4.1. Instalación de ITK	32
2.4.2. Compilación de ITK	32
2.4.3. Integración de ITK en Visual Studio 2008	34
2.4.3.1. Crear variables de entorno	38
2.5. VTK (Visualization Toolkit)	39
2.5.1. Instalación de VTK	41
2.5.2. Compilación de VTK	41
2.5.3. Integración de VTK en Visual Studio 2008	43
3. Diseño e implementación de la interfaz gráfica de usuario	47
3.1. Programación orientada a objetos	47
3.1.1. Conceptos básicos de programación en C++	47
3.2. Cómo crear un proyecto nuevo en Visual Studio 2008	49

3.3.	Estructura interna de la aplicación	52
3.4.	Cómo crear un cuadro de diálogo	55
3.4.1.	Widgets	55
3.4.2.	Eventos, signals y slots	56
3.4.3.	Crear un botón	57
3.4.3.1.	Ejemplo de botón	58
3.4.4.	Crear una etiqueta	59
3.4.4.1.	Ejemplo de etiqueta	59
3.4.5.	Crear una barra deslizadora	60
3.4.5.1.	Ejemplo de una barra deslizadora	61
3.4.6.	Crear un <i>spinbox</i>	62
3.4.6.1.	Ejemplo de un <i>spinbox</i>	63
3.4.7.	Crear un <i>combobox</i>	63
3.4.7.1.	Ejemplo de un <i>combobox</i>	64
3.4.8.	Crear un <i>frame</i>	65
3.4.8.1.	Ejemplo de un <i>frame</i>	66
3.5.	Clases de ITK utilizadas	68
3.6.	Clases de VTK utilizadas	71
4.	Resultados	74
4.1.	Interfaz Gráfica de Usuario	74
4.2.	Guía de utilización para usuarios	74
4.2.1.	Primera ventana	75
4.2.2.	Segunda ventana	77

4.3. Evaluación de la herramienta	79
5. Conclusiones y líneas futuras	83
Bibliografía	85

Índice de figuras

1.1. Ubicación del hígado en la anatomía humana.	14
1.2. Cadena de procesos que intervienen en la herramienta desarrollada.	14
1.3. Herramientas utilizadas e integradas en la aplicación.	15
1.4. Estructura básica del hígado.	16
1.5. División anatómica del hígado en segmentos según la nomenclatura de Couinaud.	17
1.6. Realización de una CT e imagen resultante.	18
1.7. Realización de una MR e imagen resultante.	19
1.8. Logotipo del estándar DICOM.	20
1.9. Ejemplo de estándar DICOM.	20
2.1. Logotipo de Visual Studio 2008.	26
2.2. Pasos a seguir en el proceso de instalación de Visual Studio 2008	27
2.3. Logotipo de Qt.	28
2.4. Pasos a seguir en el proceso de instalación de Qt.	29
2.5. Pasos a seguir en el proceso de la integración de Qt en Visual Studio 2008.	29
2.6. Menú de Qt en Visual Studio 2008.	30
2.7. Logotipo de CMake.	30

2.8. Proceso de instalación de CMake.	31
2.9. Logotipo de ITK.	31
2.10. Vista de la aplicación CMake.	32
2.11. Selección de la plataforma.	33
2.12. Compilación de ITK: Configure.	33
2.13. “Itk.sln” - Visual Studio 2008.	34
2.14. Propiedades de un proyecto en Visual Studio 2008.	34
2.15. directorios de inclusión y bibliotecas adicionales.	35
2.16. Panel de control: Sistema.	38
2.17. Pasos para crear nuevas variables de entorno.	38
2.18. Logotipo de VTK.	39
2.19. Tubería de VTK.	40
2.20. Ejemplos de objetos VTK.	40
2.21. Compilación de VTK: “VTK_USE_GUISUPPORT”.	41
2.22. Compilación de VTK: “VTK_USE_QVTK”.	42
2.23. Compilación de VTK: “DESIRED_QT_VERSION”.	42
2.24. Compilación de VTK: “Configure”.	42
2.25. “Vtk.sln” - Visual Studio 2008.	43
3.1. Primeros pasos para crear un proyecto nuevo.	49
3.2. Asistente para nuevos proyectos Qt.	50
3.3. Archivos generados en una aplicación Qt.	50
3.4. Archivo “main” generado por defecto al crear un proyecto.	51
3.5. Archivos “.h” y “.cpp” generados por defecto al crear un proyecto.	51

3.6. Archivos “.ui” generado por defecto al crear un proyecto.	52
3.7. Diagrama de la estructura interna de la aplicación.	52
3.8. Archivo “main.cpp” de la aplicación.	53
3.9. Clase QWidget	55
3.10. Clase QWidget	56
3.11. Cómo conectar <i>signals</i> y <i>slots</i>	57
3.12. Clase QPushButton	57
3.13. Ejemplo de botón.	58
3.14. Clase QLabel	59
3.15. Ejemplo de etiqueta.	59
3.16. Clase QSlider.	60
3.17. Ejemplo de barra deslizador.	61
3.18. Clase QSpinBox.	62
3.19. Ejemplo de <i>spinbox</i>	63
3.20. QComboBox.	64
3.21. Ejemplo de <i>combobox</i>	64
3.22. Clase QFrame.	65
3.23. Estilos y líneas de QFrame.	66
3.24. Ejemplo de segmentación de un hígado visualizada en un <i>frame</i>	66
4.1. Visualización de la primera ventana.	76
4.2. Visualización de la segunda ventana.	78
4.3. Aspecto de la IGU al apretar el botón “OPEN”.	79
4.4. Aspecto de la IGU al desplazarse por la barra deslizador.	79

4.5. Aspecto de la IGU al apretar el botón “SEGMENTATION” y realizar un método de segmentación.	80
4.6. Aspecto de la IGU al guardar las imágenes.	80
4.7. Aspecto de la IGU al abrir un directorio de imágenes DICOM segmentadas previamente.	81
4.8. Aspecto de la IGU al visualizar las imágenes segmentadas y el volumen 3D de las mismas.	81
4.9. Aspecto de la IGU al rotar y hacer zoom en el volumen 3D.	82

Agradecimientos

Quiero dar las gracias a mis dos tutoras de proyecto, Valery y M^aJosé, por haber confiado en mi y haber podido contar con ellas en todo momento.

A Fernando, porque no sólo ha sido un muy buen amigo, sino que también ha sido como un tutor para mi; siempre dispuesto a ayudarme y sin perder jamás la paciencia conmigo.

También quiero mencionar a Carlos y a los demás miembros del proyecto NaRALAp, que junto con mis compañeros de LabHuman han hecho que estos meses de trabajo hayan sido más fáciles.

Especial mención merecen mis padres, por estar junto a mi dándome su cariño y su apoyo incondicional, día tras día, todos los días de mi vida. No imagino mejores padres que los que tengo. Gracias por enseñarme a ser feliz a pesar de las dificultades. Gracias por nuestra pequeña familia.

A Christian, mi novio y además mi mejor amigo, porque desde que nos conocimos no me ha dejado ni un solo momento. Hace mucho tiempo que no somos uno y uno, sino dos, y juntos afrontamos los retos que se nos presentan. Gracias por hacerme tan feliz.

Por último, quiero darles las gracias a todas y cada una de las personas que siempre han creído en mi, mis amigos y mi familia.

Capítulo 1

Introducción

1.1. Objetivos

El principal objetivo de este proyecto es el desarrollo de una herramienta software que permita la segmentación y visualización tridimensional (3D) de los diferentes tejidos de hígados reales a partir de un volumen de imágenes procedentes de dispositivos clínicos de escaneo 3D no invasivos. Estos instrumentos de exploración no invasiva pueden ser por ejemplo, la resonancia magnética (MR: *Magnetic Resonance*) o la tomografía computerizada (CT: *Computer Tomography*).

Este objetivo principal se descompone en el siguiente conjunto de objetivos específicos:

- Estudio, análisis y evaluación de los diferentes algoritmos de imágenes médicas aplicados al hígado.
- Implementación de aquellos algoritmos que mejor se ajustan a la segmentación de tejidos buscada en este proyecto.
- Desarrollo de una interfaz gráfica de usuario (IGU) que permita la visualización 3D de los tejidos del hígado segmentados. Los tejidos segmentados podrán visualizarse desde diferentes puntos de vista, esto permitirá la selección de aquellos tejidos sobre los que se podrán aplicar diferentes niveles de transparencia.

Este último objetivo permitirá analizar la bondad de los diferentes algoritmos de segmentación implementados.

El proyecto final de carrera se engloba dentro del proyecto NaRALAp, un proyecto nacional del plan Avanza. NaRALAp tiene como objetivo el diseño e implementación de un sistema de Realidad Aumentada para la asistencia en cirugía abdominal así como el desarrollo de un modelo biomecánico de hígado que, junto con el sistema de Realidad Aumentada permitirá la navegación intra-operatoria a los cirujanos durante la cirugía laparoscópica de dicho órgano. NaRALAp tiene como objetivo avanzar en la creación de tecnología para el desarrollo de un sistema de navegación laparoscópica fundamental para la realización de intervenciones laparoscópicas más seguras y acordes con la planificación preoperatoria.

1.2. Entorno

Hoy en día, el entorno hospitalario está equipado con aparatos médicos que habitualmente proporcionan valiosa información en forma de imagen médica como soporte al diagnóstico o al tratamiento de un paciente en particular. La tomografía computarizada (CT), la resonancia magnética (MR), la ecografía y la tomografía por emisión de positrones (PET) son ejemplos de modalidades de imágenes que se utilizan con frecuencia.

Los algoritmos para la extracción de información a partir de imágenes son conocidos como algoritmos de segmentación, y se utilizan en numerosas aplicaciones biomédicas de tratamiento de imagen. La segmentación de las imágenes permite extraer y clasificar una región específica o un volumen de interés, aunque para ello a menudo es necesario dividir los datos de la imagen en sus componentes. La segmentación de imágenes proporciona información cuantitativa acerca de la anatomía pertinente, sirve por ejemplo para determinar el tamaño o volumen. También permite la visualización tridimensional de una estructura particular, con la triangulación de superficie, isosuperficies (*isosurfaces*) o *volume rendering*. No existe un único enfoque que pueda resolver el problema de la segmentación, y los diferentes métodos serán más eficaces en función de la modalidad de imagen que esté siendo procesada [1].

El procesamiento y análisis de imágenes biomédicas es una herramienta importante en la mejora de la calidad y cantidad de información dada por métodos no invasivos, es por ello, que juega un papel muy importante para el diagnóstico, tratamiento y seguimiento

de patologías. La identificación de los diferentes tejidos es importante para la planificación de los tratamientos a seguir, principalmente en los casos de pacientes con tumores.

Los principales objetivos del análisis de imágenes biomédicas son según Rangaraj et al.[2]:

- Recopilación de información: A través de la medición de un fenómeno para poder interpretar un órgano, un proceso o un sistema.
- Diagnóstico: Detección o confirmación de un mal funcionamiento, patología o anomalía.
- Seguimiento: Obtención de información periódica acerca de un sistema.
- Terapia y control: Modificación de la conducta de un sistema basándose en los resultados de las actividades mencionadas anteriormente para garantizar un resultado específico.
- Evaluación: Análisis de los objetivos para determinar la capacidad para satisfacer los requisitos funcionales, rendimiento, control de calidad o cuantificar el efecto del tratamiento.

Es en este entorno donde un sistema de visualización 3D que nos muestre la segmentación de los diferentes tejidos o regiones de un órgano puede ser de gran utilidad, tanto para la recopilación de información, cómo para el diagnóstico, tratamiento de una enfermedad, o planificación de una cirugía.

En esta aplicación se utiliza el lenguaje de programación Visual C++ como código de unión y las siguientes herramientas(Figura 1.3):

- Visual Studio 2008 y Qt para el entorno gráfico.
- Las librerías ITK para la segmentación de las imágenes biomédicas del hígado.
- Las librerías VTK para el renderizado y visualización de las estructuras del hígado en 3D.

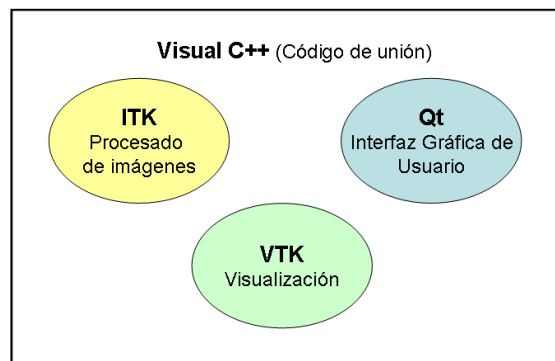


Figura 1.3: Herramientas utilizadas e integradas en la aplicación.

1.4. Estado del arte

1.4.1. Anatomía del hígado

El hígado humano es un tejido blando, no lineal y viscoelástico. Además, sus estructuras internas son complejas y difíciles de identificar en los diferentes modos de exploración no invasiva.

El hígado es el órgano más grande del cuerpo humano. Pesa en cadáver alrededor de 1500 gramos y en vivo este peso aumenta 400 gramos por la sangre contenida en el órgano. En los humanos consiste en una masa continua de células, dividida en forma incompleta por separaciones de tejido conectivo. Dentro de esta masa de células continua, las subdivisiones de los conductos biliares y de los vasos hepáticos tienen numerosas conexiones.

El hígado está situado en la parte superior del abdomen, debajo del diafragma y está mantenido en su posición por:

- Vena cava inferior, a la cual está unido a través de las venas supra-hepáticas.
- Ligamento redondo del hígado, que reemplaza en el adulto la vena umbilical del feto.
- Repliegues peritoneales.

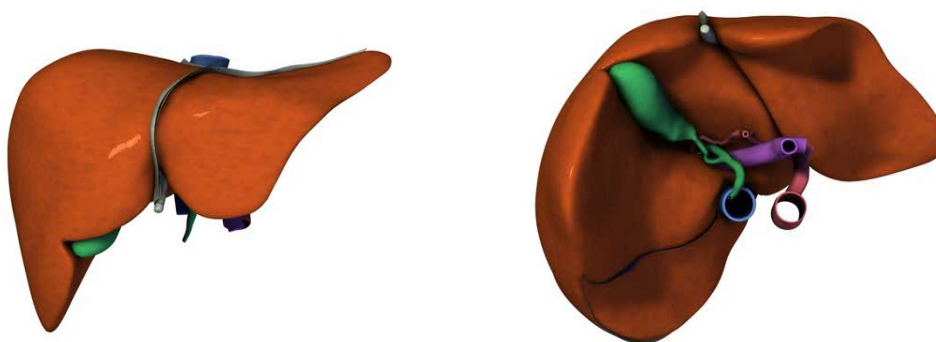
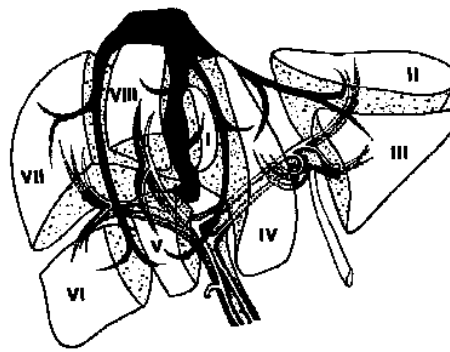


Figura 1.4: Estructura básica del hígado.

A pesar de la estructura monolítica del hígado arbitrariamente se lo considera compuesto por lóbulos. Consta de dos lóbulos principales, el derecho y el izquierdo que están divididos por un ligamento llamado falciforme, comprendiendo el lóbulo derecho cinco

sextos y el lóbulo izquierdo un sexto de la masa hepática. Un tercer lóbulo más pequeño llamado cuadrado y el lóbulo de Spiegel (lóbulo caudado) (Figura 1.4).

Cuando el cirujano planea una cirugía, hace una división del hígado en segmentos (Figura 1.5). Los segmentos del hígado se definen como las regiones servidas por una subdivisión de la vena porta, de la arteria hepática y del conducto hepático común, que viajan juntos a través de toda la masa hepática. Esta segmentación es una segmentación anatómica (clasificación de Couinaud) que divide al hígado en ocho segmentos funcionales independientes. Cada segmento es independiente porque tiene su propio flujo de entrada vascular, su flujo de salida vascular y su flujo de salida biliar.



División funcional del hígado y los segmentos de acuerdo a la nomenclatura de Couinaud (Reproducción de Bismuth H. *Surgical Anatomy and Anatomical Surgery of the Liver*. *World J. Surg.* 6: 6, 1982)

Figura 1.5: División anatómica del hígado en segmentos según la nomenclatura de Couinaud.

Hay muchos trabajos relacionados con esta segmentación [3]-[10] entre otros, ya que es la segmentación utilizada habitualmente en cirugía. De hecho, una herramienta de visualización 3D que permitiese visualizar los diferentes tejidos del hígado sería de gran ayuda a la hora de planificar una intervención que requiera de esta segmentación.

Se han estudiado algunos de los métodos de segmentación del hígado por tejidos más recientes ([11]-[15]) y se ha llegado al siguiente conjunto de conclusiones:

1. El número de tejidos que se diferencian en el hígado escaneado son el portal hepático, las venas hepáticas, las arterias hepáticas y el sistema biliar.
2. El sistema de escaneo más utilizado es la CT.

3. Para la segmentación se utilizan contrastes y como la toma de imágenes se realiza en diferentes instantes de tiempo para cada contraste, hay que hacer un registro de las imágenes para situarlas en un sistema de coordenadas común.

Todas estas conclusiones serán tenidas en cuenta a la hora de hacer la segmentación.

1.4.2. Métodos de adquisición de imágenes

Los procedimientos de adquisición de imágenes se pueden catalogar en procedimientos invasivos y no invasivos. Los procedimientos invasivos implican la colocación de dispositivos o materiales dentro del cuerpo, como la inserción una aguja, una sonda o un endoscopio, mientras que en los procedimientos no invasivos no se introduce instrumental médico dentro del cuerpo del paciente. Los procedimientos no invasivos son deseables a fin de minimizar el riesgo para el paciente [2].

1.4.2.1. Métodos de adquisición de imágenes no invasivos

Entre los procedimientos no invasivos habitualmente usados se encuentran la tomografía computarizada y la resonancia magnética.

- **Tomografía computarizada**

La tomografía computarizada (CT) consiste en la obtención de imágenes detalladas de los órganos y estructuras internas del cuerpo, por medio de una fuente emisora de radiación ionizante (rayos X) (Figura 1.6).



Figura 1.6: Realización de una CT e imagen resultante.

La fuente emisora de radiación emite un haz de rayos X que incide sobre el objeto a estudiar. Parte de dicha radiación es absorbida por el objeto y el resto es recogida por un receptor. Tanto emisor como receptor giran en círculo alrededor del objeto sobre un eje que cambia su inclinación. Una vez completados los 360° se habrán obtenido las proyecciones de un corte del objeto. Se producen múltiples imágenes como consecuencia de dicha rotación. Un ordenador combina todas esas imágenes en una imagen final que representa un corte. A continuación, se mueve una pequeña distancia y se repite el proceso para obtener un nuevo corte.

Las tomografías computarizadas muestran más detalles que las radiografías estándar. Mientras que con las radiografías estándar se obtiene una única imagen, la tomografía permite obtener varias imágenes en distintos planos de la zona anatómica a estudio y por tanto, tener muchas vistas diferentes del mismo órgano o de la misma estructura proporcionando así, mayor detalle. La información de los rayos X es enviada a un ordenador que interpreta los datos de los rayos X y los presenta en forma bidimensional en un monitor.

■ Resonancia magnética

La resonancia magnética (MR) consiste en la obtención de las imágenes de los órganos y estructuras internos del cuerpo mediante el empleo de un campo electromagnético (imán), un emisor/receptor de ondas de radio (escáner) y un ordenador (Figura 1.7).

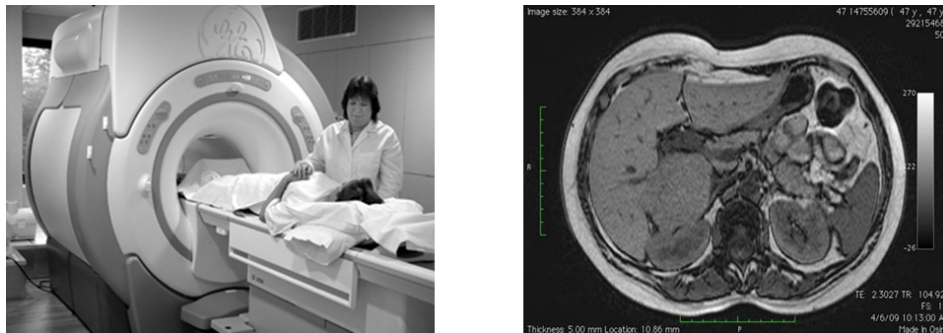


Figura 1.7: Realización de una MR e imagen resultante.

La resonancia magnética permite obtener imágenes del organismo de forma no invasiva sin emitir radiación ionizante (rayos X) y en cualquier plano del espacio. Las ondas de radio y el campo electromagnético excitan a los protones (núcleos de los átomos de hidrógeno) que se encuentran en los tejidos que van a ser estudiados,

provocando que se alineen unos con otros. Cuando la radiación electromagnética deja de emitirse los protones se liberan y regresan a su posición inicial liberando energía en forma de ondas de radio que son recogidas por el escáner y enviadas a un ordenador para su procesamiento en forma de imagen.

1.4.3. Formato DICOM

La información de entrada al sistema será un conjunto de imágenes MR y/o CT correspondientes al hígado. Ambos tipos de exploraciones se almacenan en los equipos de exploración clínicos en formato DICOM.



Figura 1.8: Logotipo del estándar DICOM.

DICOM (*Digital Imaging and Communication in Medicine*) es un estándar para el intercambio de imágenes médicas, pensado para el manejo, almacenamiento, impresión y transmisión de imágenes médicas (Figura 1.8). El estándar DICOM especifica la forma en que se debe guardar la información relativa a las exploraciones en archivos digitales y el contenido que estos archivos pueden tener. Además, ofrece la oportunidad de interconectar sistemas informáticos de diferentes fabricantes y hace posible que se comuniquen entre sí.

Los ficheros DICOM tienen una extensión *.dcm* y consisten en dos partes diferenciadas (Figura 1.9):

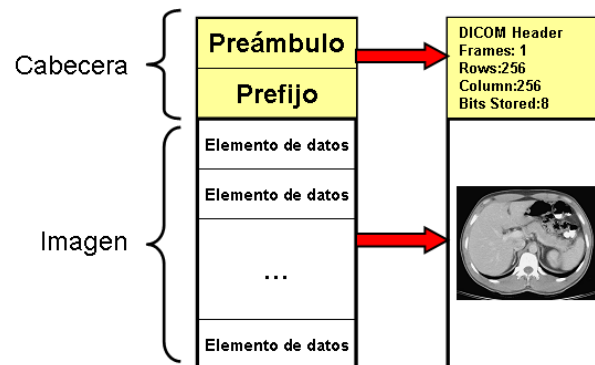


Figura 1.9: Ejemplo de estándar DICOM.

- Cabecera

En la cabecera se especifican datos administrativos (datos del paciente, tipo de exploración, equipo de adquisición, condiciones en que se tomó la imagen, hospital etc.) y datos sobre la imagen (tamaño etc.).

La cabecera está compuesta por:

- Preámbulo

El estándar DICOM especifica que un fichero en formato DICOM debe de comenzar con un preámbulo. Este preámbulo tiene un tamaño fijo de 128 bytes y está pensado para tener un uso definido por la implementación. Puede contener información sobre el nombre de la aplicación usada para crear el fichero o información que permita a aplicaciones acceder directamente a los datos de la imagen almacenada en el fichero.

- Prefijo

Este prefijo consiste en cuatro bytes que contienen la cadena de caracteres "DICM". El propósito de este prefijo es permitir a las implementaciones diferenciar si un fichero es DICOM o no.

- Imagen

Una imagen se define como una función bidimensional compuesta por sus coordenadas espaciales (x,y) y su intensidad o nivel de gris. La imagen consiste en un mapa de densidades. Esta información puede interpretarse como una imagen en escala de grises si se hace una superposición de la escala de grises sobre la escala de densidades (unidades Hounsfield). La escala de densidad o intensidad va de -1000 a 1000 o de -1000 a 4000. Por otra parte, sólo contamos con 255 niveles de grises, donde 0 es blanco y 255 negro, por lo que se hace una transformación de unidades de Hounsfield a la escala de grises. La forma en que se hace esta transformación de escalas está definida por el estándar DICOM [16].

1.4.4. Segmentación

1.4.4.1. Definición de segmentación

La segmentación de imágenes es el proceso mediante el cual asignamos una etiqueta a cada uno de los píxeles de una imagen, de manera que píxeles con características comunes son agrupados como pertenecientes a una misma entidad.

El objetivo de la segmentación de una imagen consiste en separar los distintos objetos que aparecen en la misma del fondo y diferenciarlos entre ellos. Tras la segmentación, cada píxel de la imagen quedará identificado como perteneciente a uno de los objetos que la forman o al fondo.

Las propiedades deseables de una región serían la homogeneidad del color o intensidad dentro de la misma, la homogeneidad de su textura y la existencia de bordes claros con las regiones adyacentes.

La segmentación no tiene porque limitarse a la imagen bidimensional, sino que el mismo concepto puede ser extendido a volúmenes tridimensionales.

1.4.4.2. Técnicas de segmentación

Las técnicas de segmentación se pueden dividir en técnicas basadas en píxeles o técnicas basadas en regiones de la imagen. Las primeras asignan la misma etiqueta a píxeles con intensidades similares, independientemente de si los píxeles etiquetados forman un conjunto espacialmente compacto. Las técnicas basadas en regiones asignan la misma etiqueta sólo a píxeles que forman una región compacta.

A continuación se describen las técnicas de segmentación que se van a aplicar en este proyecto: una basada en píxel, la umbralización, y otra basada en regiones, el crecimiento de regiones y sus variantes.

1.4.4.2.1. Técnicas de segmentación basadas en píxeles

- Umbralización

La umbralización (*thresholding*) es un método de segmentación de imágenes que actúa según el nivel de intensidad de las mismas. La umbralización trata de determinar un valor de intensidad, denominado umbral (*threshold*), que separa las diferentes partes de la imagen.

La segmentación se logra agrupando todos los píxeles con mayor intensidad cercanos a los diferentes umbrales. En el caso de la umbralización binaria (*binary thresholding*) la segmentación resultante será una imagen con los píxeles agrupados a nivel 1 y el resto (fondo de la imagen) a valor 0.

La umbralización es una técnica efectiva para obtener la segmentación de imágenes donde estructuras diferentes tienen características diferenciables. Existen métodos interactivos, basados en la apreciación visual del usuario, pero también existen métodos automáticos.

Generalmente, la umbralización es el paso inicial de una secuencia de operaciones de procesamiento de imágenes. Su principal limitación es que es sensible al ruido e inhomogeneidades de intensidad, las cuales pueden ocurrir en imágenes de resonancia magnética. Estos factores corrompen el histograma de la imagen, haciendo la separación más difícil.

1.4.4.2.2. Técnicas de segmentación basadas en regiones

- Crecimiento de regiones

El crecimiento de regiones (*region growing*) es una técnica para extraer regiones de la imagen que están conectadas según cierto criterio predefinido. Este criterio puede estar basado en información de intensidades y/o bordes de la imagen.

Este método requiere un punto semilla (*seed point*) que es seleccionado manualmente por el usuario, y extrae todos los píxeles conectados a la semilla, que tengan el mismo valor de intensidad. Una región homogénea crece alrededor de la semilla y los píxeles vecinos con intensidades similares se van añadiendo iterativamente a la región.

Su desventaja principal es que requiere interacción manual para obtener el punto semilla. La región creciente también puede ser sensible al ruido, causando que las regiones extraídas tengan agujeros e inclusive que se desconecten.

Los algoritmos de crecimiento de regiones varían según el criterio utilizado para decidir si un píxel debe ser incluido en una región o no, la conectividad usada para determinar los vecinos y la estrategia utilizada para visitar los píxeles vecinos [17].

Algunos de esos algoritmos son:

- *Connected Threshold*

Un criterio simple para la inclusión de píxeles en una región es evaluar el valor de intensidad en el interior de un intervalo específico.

Los valores de límite inferior y superior deben ser proporcionados por el usuario. El algoritmo incluye todos aquellos píxeles cuya intensidad se encuentran dentro del intervalo:

$$I(x) \in [inferior, superior]$$

- *Confidence connected*

El algoritmo comienza calculando la media m y la desviación estándar σ para todos los píxeles que figuran actualmente en la región donde hemos situado la semilla. El usuario proporciona el factor que se utiliza para multiplicar la desviación estándar y definir un rango alrededor de la media.

Los píxeles vecinos cuyos valores de intensidad se hallan dentro del rango son aceptados e incluidos en la región. Cuando no hay más píxeles vecinos que cumplan el criterio, se considera que el algoritmo ha terminado su primera iteración. En ese punto, se vuelven a calcular la media y desviación estándar de los niveles de intensidad, utilizando todos los píxeles que figuran actualmente en la región. Esta media y desviación estándar definen un nuevo rango de intensidad que se utiliza para visitar a los vecinos de la región actual y evaluar si su intensidad está dentro del rango. Este proceso iterativo se repite hasta que no haya más píxeles que añadir o se alcanza el número máximo de iteraciones.

$$I(x) \in [m - f\sigma, m + f\sigma]$$

1.5. Organización

Durante los siguientes capítulos se pretende mostrar con detalle cada una de las tareas que se han realizado para la consecución de los objetivos del proyecto.

En el capítulo 2 se explicará detalladamente las distintas herramientas utilizadas para llevar a cabo la aplicación implementada en este proyecto y los pasos previos para la instalación de tales herramientas.

En el capítulo 3 se mostrarán los conceptos básicos de la programación orientada a objetos, los pasos para la creación de un proyecto nuevo y la estructura interna que posee la aplicación. Además se detallarán los elementos básicos para crear la interfaz gráfica de usuario.

En el capítulo 4 se evaluarán las funcionalidades de la aplicación realizada con distintas imágenes.

Por último, en el capítulo 5 se expondrán las conclusiones obtenidas junto con las líneas futuras.

Capítulo 2

Herramientas

En la aplicación desarrollada en el presente proyecto se han utilizado el siguiente conjunto de herramientas y librerías de programación que serán descritas en este capítulo:

- Visual Studio 2008
- Qt
- CMake
- ITK
- VTK

2.1. Visual Studio 2008

Visual Studio 2008 (Figura 2.1) es un entorno integrado de desarrollo que permite la programación orientada a objetos. Entre los lenguajes que soporta se encuentran: Visual Basic, Visual C# y Visual C++, siendo éste último el elegido para la programación de esta herramienta.



Figura 2.1: Logotipo de Visual Studio 2008.

El entorno de desarrollo integrado de Visual Studio ofrece un conjunto de herramientas destinadas a ayudar al programador a escribir y modificar código, así como a detectar y corregir errores.

Los tres pilares básicos de Visual Studio 2008 son: el desarrollo rápido de aplicaciones, la colaboración eficiente entre equipos y la innovación en experiencias de usuario.

2.1.1. Instalación de Visual Studio 2008

Visual Studio 2008 se instalará en el ordenador con las opciones por defecto, tal y como se muestra en la Figura 2.2.



Figura 2.2: Pasos a seguir en el proceso de instalación de Visual Studio 2008

2.2. Qt

Qt (Figura 2.3) es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario, programación web, bases de datos y también para el desarrollo de programas sin interfaz gráfica como herramientas de la consola y servidores. Qt utiliza el lenguaje de programación orientado a objetos C++, aunque adicionalmente puede ser utilizado con otros lenguajes de programación.

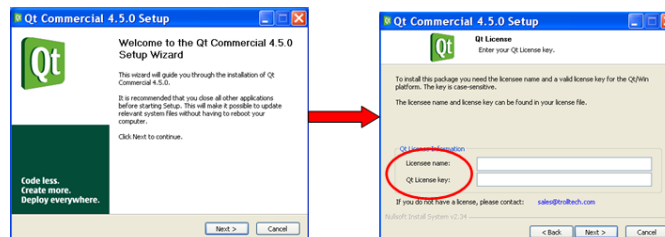


Figura 2.3: Logotipo de Qt.

Permite realizar una programación visual y dirigida por eventos. En el caso de la programación visual, el programador centra su atención en diseñar el aspecto gráfico de la aplicación, la distribución de los elementos visuales llamados *widgets*, la interacción entre los mismos, los distintos tipos de ventanas existentes, etc. En tanto a lo concerniente a la programación dirigida por eventos, el programador escribe el código que se ejecutará en respuesta a determinados eventos. No existe la idea de un control de flujo secuencial en el programa, sino que el programador toma el control cuando se dispara un evento. La labor del programador, es por tanto, asociar a cada evento el comportamiento adecuado.

2.2.1. Instalación de Qt

En el presente proyecto se ha utilizado la versión 4.5.0 de Qt. Para su instalación se ejecutará el fichero con extensión “.exe”, de la versión comercial o de la versión de prueba proporcionado en la página web <http://qt.nokia.com/downloads> y se instalarán las opciones por defecto, de acuerdo con la Figura 2.4.



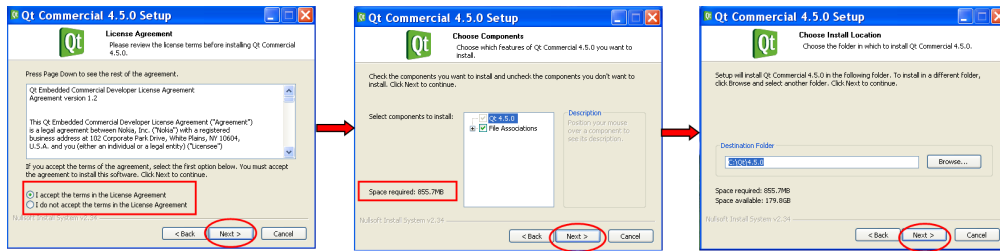


Figura 2.4: Pasos a seguir en el proceso de instalación de Qt.

2.2.2. Integración Qt con Visual Studio 2008

Se ha elegido utilizar Qt con Visual Studio 2008 aunque funciona en todas las principales plataformas. Para la integración de Qt con Visual Studio 2008 hay que ejecutar el fichero “*qt-vs-addin-1.0.2.exe*” (o cualquier versión posterior) que proporciona Qt en la página web <http://qt.nokia.com/downloads/visual-studio-add-in> y se instalarán las opciones por defecto, tal y como se muestra en la Figura 2.5.

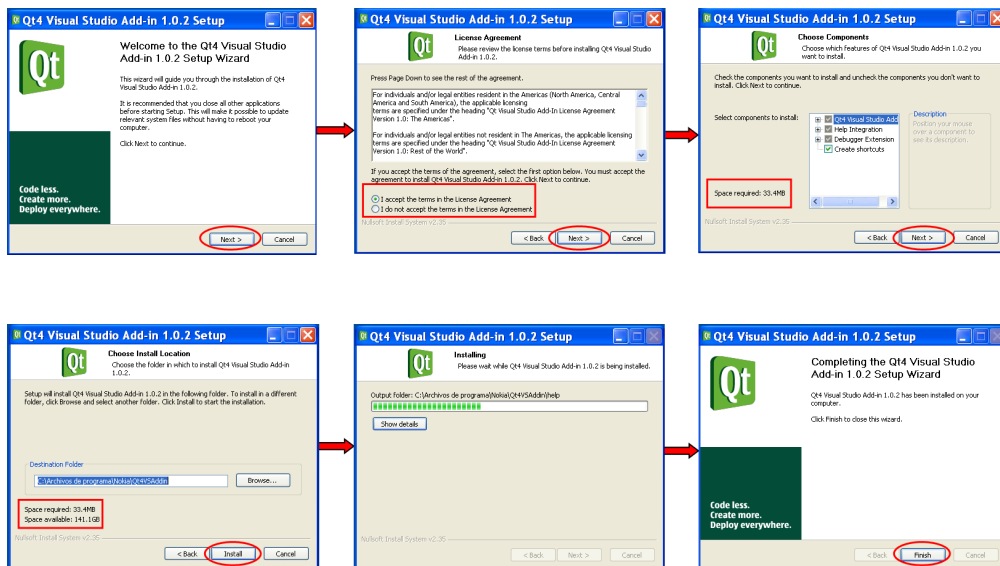


Figura 2.5: Pasos a seguir en el proceso de la integración de Qt en Visual Studio 2008.

Para comprobar que se ha instalado correctamente entramos en Visual Studio 2008 y vemos como aparece un menú de Qt (Figura 2.6).

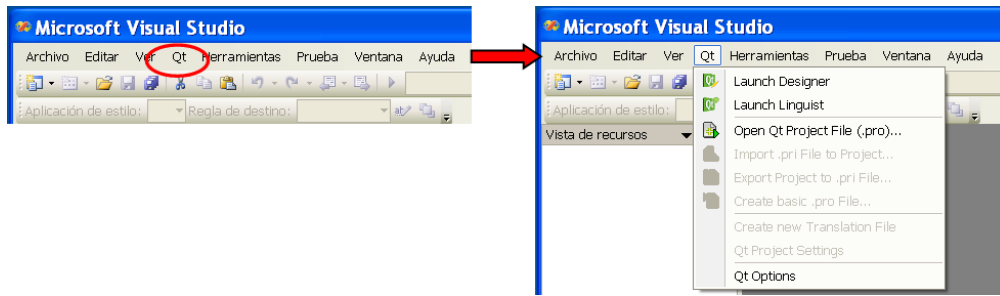


Figura 2.6: Menú de Qt en Visual Studio 2008.

2.3. CMake

CMake (Figura 2.7) es una multiplataforma basada en un sistema de código abierto. Se utiliza para controlar el proceso de compilación. La compilación es un proceso de traducción de las instrucciones escritas en un determinado lenguaje de programación a lenguaje máquina, el cuál es interpretado por la computadora.



Figura 2.7: Logotipo de CMake.

2.3.1. Instalación de CMake

Como se trata de un compilador libre se puede bajar el instalable “.exe” de la página web: www.cmake.org, en concreto, en la página web: <http://www.cmake.org/cmake/resources/software.html>. Una vez se haya bajado, se instalará en el ordenador con las opciones por defecto, tal y como se muestra en la Figura 2.8

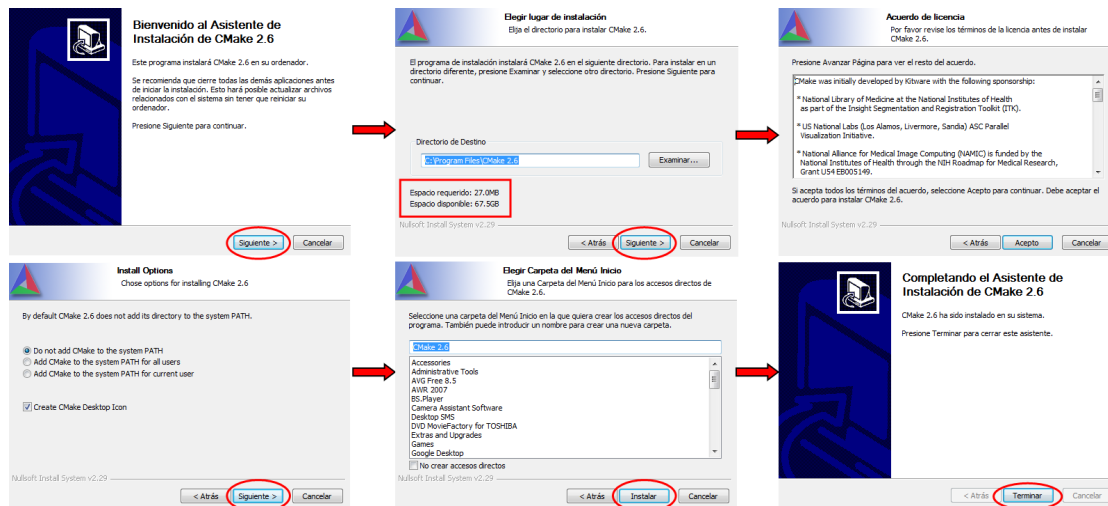


Figura 2.8: Proceso de instalación de CMake.

2.4. ITK (Insight Segmentation and Registration Toolkit)

ITK (Figura 2.9) es un sistema multiplataforma que proporciona herramientas de registro y segmentación para el análisis de imágenes biomédicas. ITK está implementado en C++, y utiliza un entorno de desarrollo conocido como CMake para gestionar el proceso de compilación de una manera independiente de la plataforma. ITK consiste en librerías de código libre que pueden integrarse en herramientas como Qt.



Figura 2.9: Logotipo de ITK.

ITK está organizado en torno a una arquitectura de flujo de datos, es decir, los datos se representan mediante objetos que a su vez son procesados por filtros, conectados entre sí mediante tuberías (*pipelines*). Los filtros se pueden combinar entre sí.

2.4.1. Instalación de ITK

Como ITK son librerías libres se puede bajar el instalable “.exe” de la página web: www.itk.org, en concreto, en la página web: <http://www.itk.org/ITK/resources/software.html>. Se bajarán las *InsightToolkit* y las *InsightApplications*. Son dos archivos comprimidos con extensión “.zip”, por lo que habrá que descomprimir ambos.

2.4.2. Compilación de ITK

Para compilar las librerías ITK se necesitan conocer y disponer de tres datos: el compilador que va a ser empleado, la dirección del código fuente y la dirección donde se generarán el código objeto, las librerías y los archivos binarios producidos en la compilación.

Al ejecutar la aplicación CMake (Figura 2.10), donde indica *Where is the source code*: habrá que poner el directorio donde estén las ITK descomprimidas del apartado anterior (*InsigthToolkit*). Mientras que en *Where to build the binaries* habrá que poner una carpeta creada por el usuario donde estarán las ITK compiladas que se van a crear.

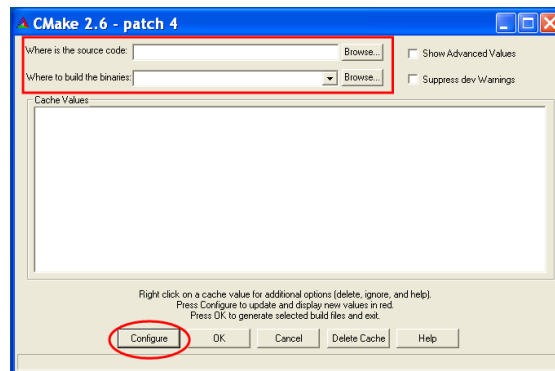


Figura 2.10: Vista de la aplicación CMake.

Una vez hecho esto, se hará *click* en *Configure*. El siguiente paso pedirá la plataforma en la que se va a hacer la compilación de las librerías (Figura 2.11). Se pondrá la versión Visual Studio 9 (que corresponde al Visual Studio 2008).

A continuación se hará de nuevo *click* en *Configure*. Saldrán una serie de variables en rojo en la pantalla que pueden ser modificadas o hacer *click* en *Configure*, una vez más.



Figura 2.11: Selección de la plataforma.

Las variables pasarán de estar en rojo a estar en gris y estará disponible la opción “ok”. Se pulsará y el programa se cerrará por si solo al finalizar (Figura 2.12).

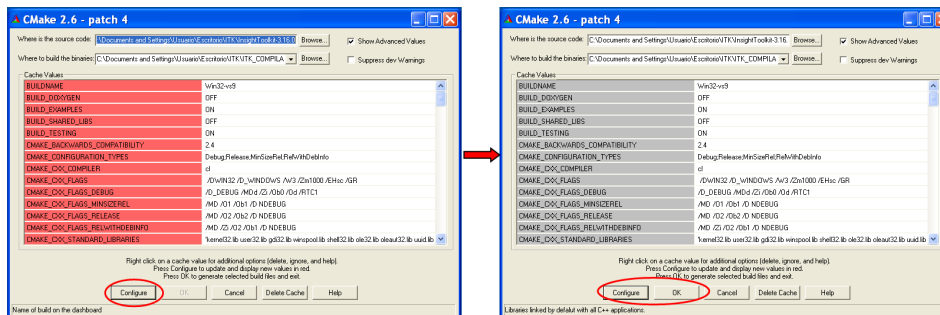


Figura 2.12: Compilación de ITK: Configure.

Cmake ha convertido los archivos binarios en C++ a Visual C++. Al abrir la carpeta creada por el usuario donde estarán las ITK compiladas (el directorio que se introdujo en *Where to build the binaries*) aparece un proyecto llamado “**Itk.sln**” y que se ejecutará en Visual Studio 2008.

En el proyecto “**Itk.sln**” al situarse encima de “**ALL_BUILD**”, se pulsará el botón derecho y se dará a “**Generar**” (Figura 2.13). Este proceso tardará alrededor de una hora y lo que se van a crear ahora son las librerías que posteriormente se usarán en los programas. Se puede ver simultáneamente cómo se van creando estas librerías en la carpeta “*bin/debug*”, que se habrá creado en el directorio *Where to build the binaries* (la carpeta donde están las ITK compiladas). Una vez hecho esto, se podrán incluir las librerías ITK en cualquier proyecto (Figura 2.13).

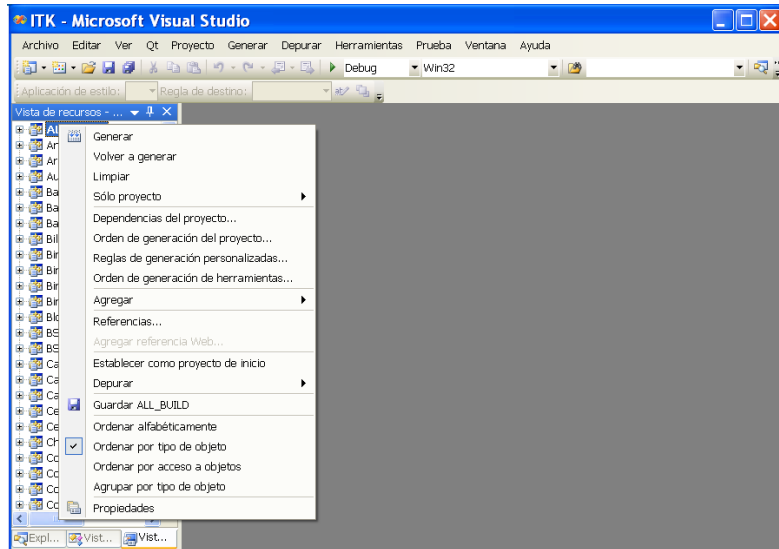


Figura 2.13: “Itk.sln” - Visual Studio 2008.

2.4.3. Integración de ITK en Visual Studio 2008

Llegado a este punto, sólo quedará añadir las librerías ITK al proyecto de Visual Studio 2008 en el que vayamos a trabajar. Se hace *click* con el botón derecho del ratón encima del proyecto de Visual Studio 2008, tal y como aparece en la Figura 2.14 y seleccionamos la opción “Propiedades” que se muestra en el menú desplegable.

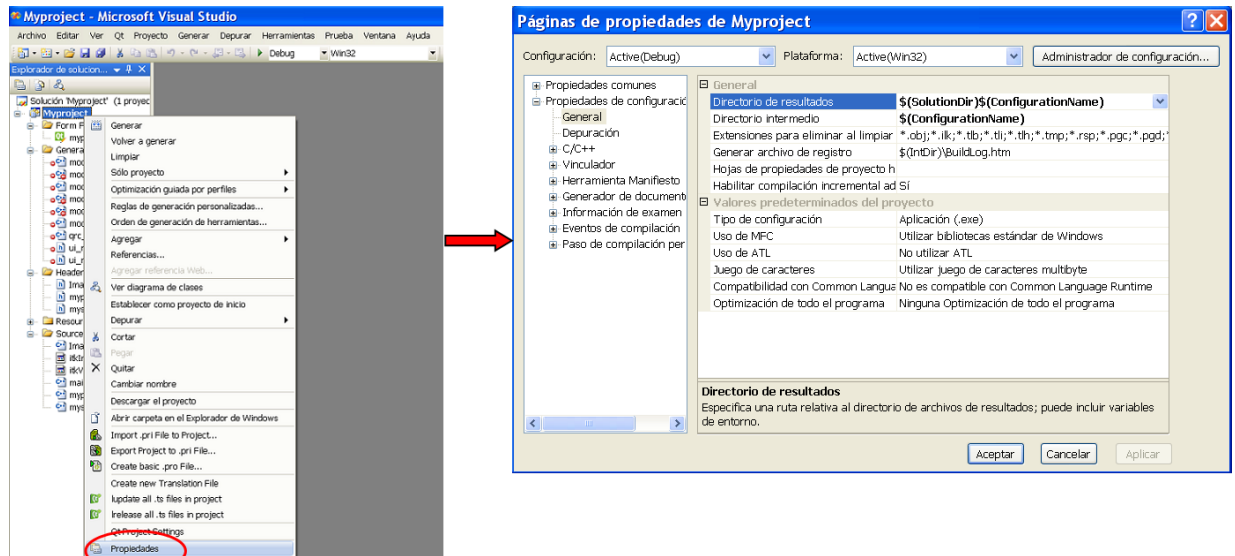


Figura 2.14: Propiedades de un proyecto en Visual Studio 2008.

Las librerías se añadirán en dos pasos (Figura 2.15):

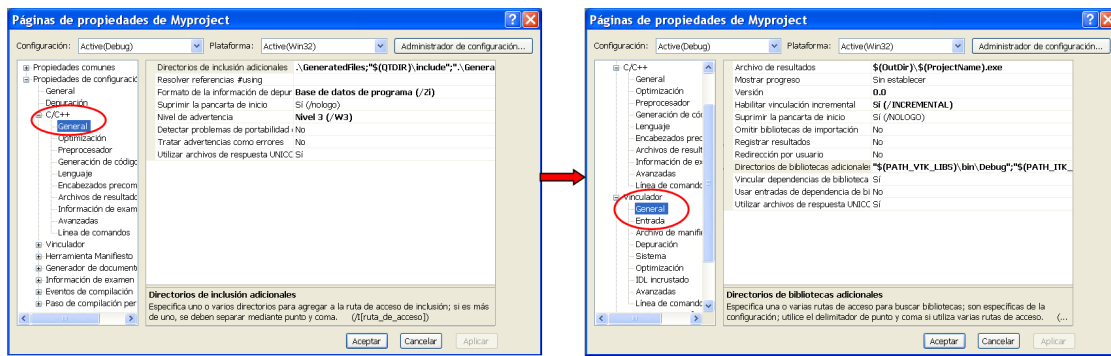


Figura 2.15: directorios de inclusión y bibliotecas adicionales.

1. Añadir los directorios de inclusión adicionales:

- $$(PATH_ITK_INCLUDES)\Code\Algorithms$
- $$(PATH_ITK_INCLUDES)\Code\BasicFilters$
- $$(PATH_ITK_INCLUDES)\Code\Common$
- $$(PATH_ITK_INCLUDES)\Code\IO$
- $$(PATH_ITK_INCLUDES)\Code\Numerics$
- $$(PATH_ITK_INCLUDES)\Code\Numerics\FEM$
- $$(PATH_ITK_INCLUDES)\Code\Numerics\NeuralNetworks$
- $$(PATH_ITK_INCLUDES)\Code\Numerics\Statistics$
- $$(PATH_ITK_INCLUDES)\Code\SpatialObject$
- $$(PATH_ITK_INCLUDES)\nifti\znzlib$
- $$(PATH_ITK_INCLUDES)\nifti\niftilib$
- $$(PATH_ITK_INCLUDES)\Utilities\vxl\core$
- $$(PATH_ITK_INCLUDES)\Utilities\vxl\vcl$
- $$(PATH_ITK_INCLUDES)\Utilities\vxl\v3p\netlib$
- $$(PATH_ITK_INCLUDES)\Utilities$
- $$(PATH_ITK_INCLUDES)\Utilities\DICOMParser$
- $$(PATH_ITK_INCLUDES)\Utilities\expat$
- $$(PATH_ITK_INCLUDES)\Utilities\gdcmsrc$

- `$(PATH_ITK_INCLUDES)\Utilities\itkExtHdrs`
- `$(PATH_ITK_INCLUDES)\Utilities\NrrdIO`
- `$(PATH_ITK_INCLUDES)\Utilities\MetaIO`
- `$(PATH_ITK_LIBS)\Utilities`
- `$(PATH_ITK_LIBS)\Utilities\DICOMParser`
- `$(PATH_ITK_LIBS)\Utilities\expat`
- `$(PATH_ITK_LIBS)\Utilities\gdcm`
- `$(PATH_ITK_LIBS)\Utilities\NrrdIO`
- `$(PATH_ITK_LIBS)\Utilities\vxI\core`
- `$(PATH_ITK_LIBS)\Utilities\vxI\vcl`
- `$(PATH_ITK_LIBS)\Utilities\vxI\v3p\netlib`

Las variables `$(PATH_ITK_INCLUDES)` y `$(PATH_ITK_LIBS)` hacen referencia a la ruta donde se encuentra el código fuente de las librerías ITK y las librerías compiladas, respectivamente. En el apartado siguiente se explicará como crear dichas variables.

2. Añadir los directorios de bibliotecas adicionales

- En “General”: `$(PATH_ITK_LIBS)\bin\Debug` y `$(PATH_ITK_LIBS)\bin\Release`
- En “Entrada”:
 - `ITKStatistics.lib`
 - `ITKFEM.lib`
 - `ITKAlgorithms.lib`
 - `itkNetlibSlatec.lib`
 - `ITKNumerics.lib`
 - `ITKBasicFilters.lib`
 - `ITKIO.lib`
 - `ITKNrrdIO.lib`
 - `itkgdcm.lib`
 - `itkjpeg12.lib`
 - `itkjpeg16.lib`

- itkopenjpeg.lib
- itkpng.lib
- itktiff.lib
- itkjpeg8.lib
- ITKSpatialObject.lib
- ITKCommon.lib
- itkvnl_inst.lib
- itkvnl_algo.lib
- itkv3p_netlib.lib
- itkvnl.lib
- itkvcl.lib
- ITKMetaIO.lib
- itksys.lib
- ITKDICOMParser.lib
- ITKEXPAT.lib
- ITKniftio.lib
- ITKznc.lib
- itkzlib.lib

2.4.3.1. Crear variables de entorno

Una variable de entorno es un valor almacenado en memoria que simplifica el valor al que corresponde y puede ser utilizado en varios procesos que funcionan simultáneamente. Así, para evitar especificar en Visual Studio 2008 todas las rutas donde se encuentran las librerías ITK podemos crear nuevas variables de entorno que simplifiquen este paso. Para ello, se debe ir al “Panel de control”, a través del menú Inicio de Windows y hacer *click* en “Sistema” (Figura 2.16).

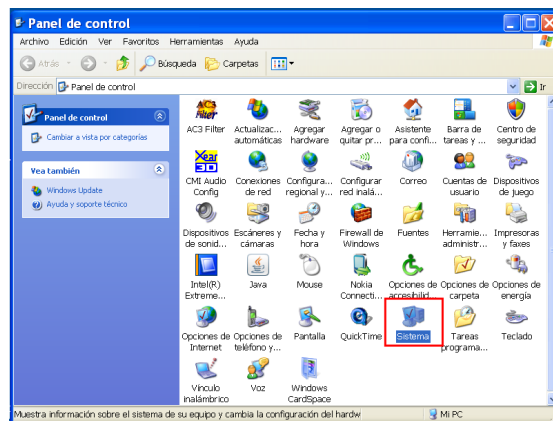


Figura 2.16: Panel de control: Sistema.

A continuación, hacer *click* en “Opciones avanzadas” y después acceder a la pestaña llamada “Variables de entorno”. Por último hacer *click* en el botón “Nueva” de “Variables de usuario” (Figura 2.17).

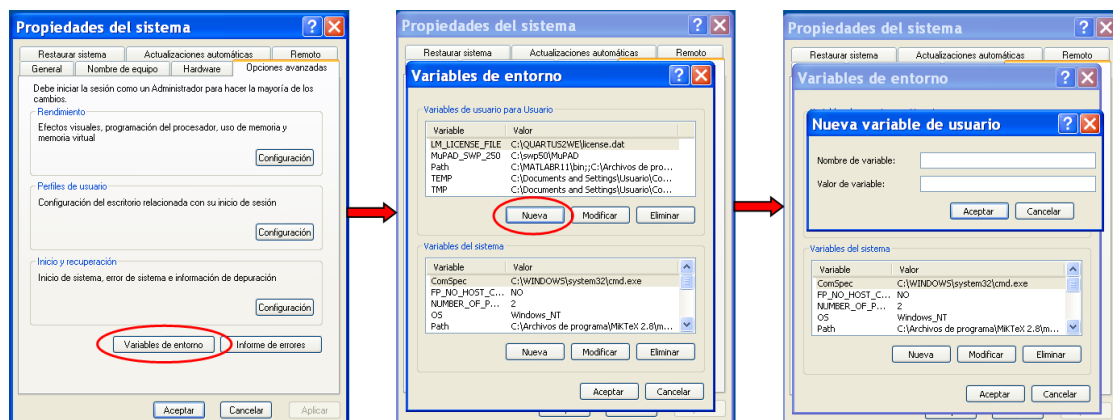


Figura 2.17: Pasos para crear nuevas variables de entorno.

2.5. VTK (Visualization Toolkit)

VTK (Figura 2.18) es un conjunto de librerías de código abierto orientada a objetos y disponible para visualización y tratamiento de la imagen. Al igual que ocurre con ITK, VTK está implementado en C++, utiliza la aplicación CMake para el proceso de compilación y además puede integrarse en herramientas como Qt.

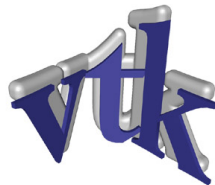


Figura 2.18: Logotipo de VTK.

VTK es una de las herramientas estándar más importante en la programación para el análisis de imágenes ya que cuenta con una amplia funcionalidad tanto para la imagen, como para tratamiento de superficies y visualización. No sólo nos permite visualizar figuras geométricas, sino que además soporta una amplia variedad de algoritmos de visualización y otras modernas técnicas de modelado en 2D y 3D. Debido a su gran potencia, se hacen necesarios amplios recursos de memoria en el ordenador para poder aprovechar la totalidad de sus funcionalidades.

Las claves del éxito utilizando VTK son:

- La comprensión de la estructura de la jerarquía orientada a objetos.

Para generar una imagen 3D se necesitan unas entidades llamadas actores, luces y cámaras. Los actores hacen referencia a los objetos que aparecen en la imagen y poseen diversas propiedades que permiten cambiar su apariencia. La imagen 3D está iluminada por un conjunto de luces y el usuario mira la imagen como si lo hiciera a través de una cámara virtual que enfoca dicha imagen tridimensional. Se puede cambiar la posición y orientación de las luces y las cámaras.

- La comprensión de su arquitectura basada en tuberías.

El procesamiento de los datos se realiza a través de una arquitectura basada en *pipelines* o tuberías tal y como se muestra en las Figuras 2.19 y 2.20 y se compone de:

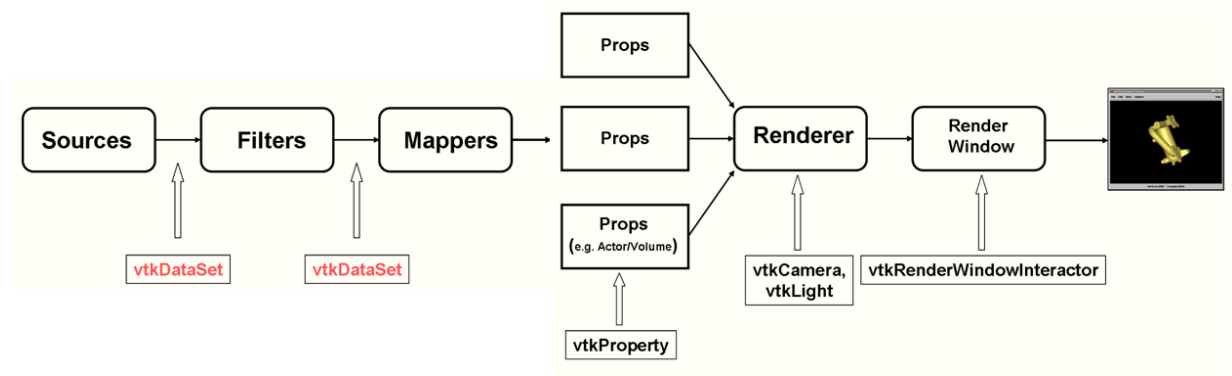


Figura 2.19: Tubería de VTK.

1. Fuentes: Producen datos.
2. Filtros: Operan en algunos datos para producir una versión modificada.
3. *Mappers*: Definen la interfaz entre los datos (por ejemplo, imágenes) y las primitivas gráficas o software de representación.
4. Actores/Volúmenes: Generan la representación visible de los datos.
5. *Renderers*: Generan una imagen 2D a partir de una escena 3D. Un *renderer* es un objeto que controla el proceso de convertir la geometría, una especificación para las luces y una vista de cámara en una imagen.
6. *Render Window*: Hace referencia a la ventana de presentación, es decir, es el espacio en pantalla en el que la imagen de la cámara virtual se muestra.

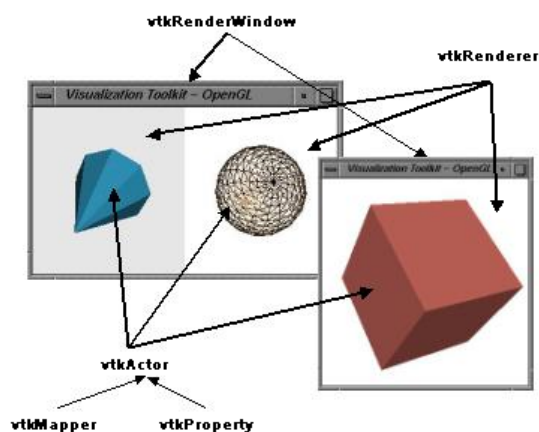


Figura 2.20: Ejemplos de objetos VTK.

2.5.1. Instalación de VTK

Al igual que ocurría con las ITK, las VTK son librerías libres así que se puede bajar el instalable “.exe” de la página web: www.vtk.org, en concreto, en la página web: <http://www.vtk.org/VTK/resources/software.html>. Se bajará el archivo “vtk-5.4.2.zip”. Al ser un archivo comprimido (“.zip”) habrá que descomprimirlo.

2.5.2. Compilación de VTK

Al igual que ocurría en las librerías ITK, la instalación completa del código fuente requiere la compilación del código fuente de VTK, por lo tanto lo compilaremos con la ayuda de la aplicación CMake.

Al ejecutar la aplicación CMake, donde indica *Where is the source code* habrá que poner el directorio donde estén las VTK descomprimidas del apartado anterior (“vtk-5.4.2”). Mientras que en *Where to build the binaries* habrá que poner una carpeta creada por el usuario donde estarán las VTK compiladas que se van a crear.

Después se hará *click* en *Configure* y el siguiente paso pedirá la plataforma en la que se va a hacer la precompilación de las librerías (Figura 2.11). Se pondrá la versión Visual Studio 9 (que corresponde al Visual Studio 2008). A continuación se hará de nuevo *click* en *Configure*. Saldrán una serie de variables en rojo en la pantalla, de las cuales se cambiará “VTK_USE_GUISUPPORT” de “OFF” a “ON” (Figura 2.21).

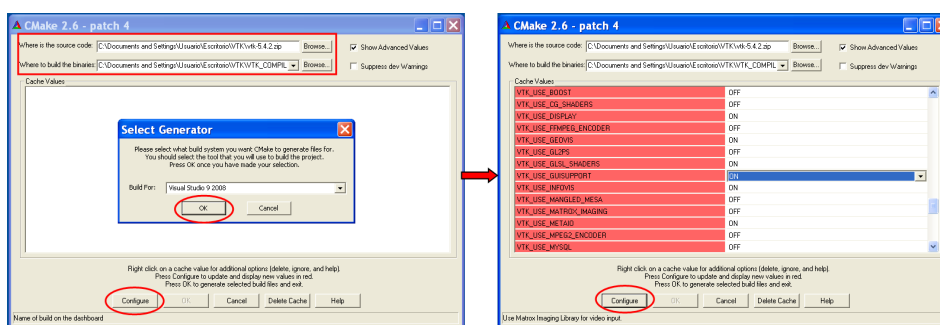


Figura 2.21: Compilación de VTK: “VTK_USE_GUISUPPORT”.

Se repetirá el proceso, haciendo *click* en *Configure*. Esta vez se cambiará “VTK_USE_QVTK” de “OFF” a “ON” (Figura 2.22).

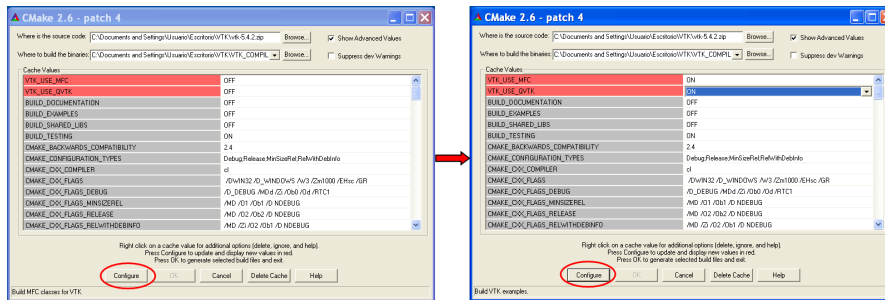


Figura 2.22: Compilación de VTK: “VTK_USE_QVTK”.

Una vez más se repetirá el proceso, haciendo *click* en “*Configure*”. Esta vez se introducirá en “DESIRED_QT_VERSION” la versión utilizada de Qt (en el proyecto actual correspondería con la versión 4, ya que se trata de la versión) y se cambiará “VTK_USE_QVTK_QTOPENGL” de “OFF” a “ON” (Figura 2.23).

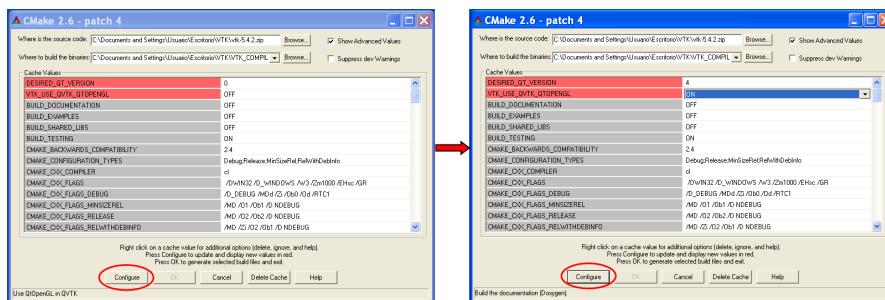


Figura 2.23: Compilación de VTK: “DESIRED_QT_VERSION”.

Por último se vuelve a hacer *click* en “*Configure*” y cuando las variables aparezcan el gris se hace *click* en “*Ok*” (Figura 2.24).

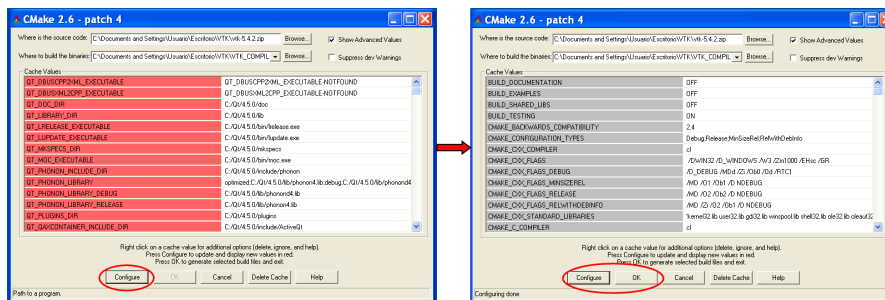


Figura 2.24: Compilación de VTK: “Configure”.

Cmake ha convertido los archivos binarios en C++ a Visual C++. Al abrir la carpeta creada por el usuario donde estarán las VTK compiladas (el directorio que se introdujo en *Where to build the binaries*) aparece un proyecto llamado “Vtk.sln” y que se ejecutará en Visual Studio 2008.

Al situarse encima de “ALL_BUILD”, se pulsará el botón derecho y se dará a “Generar” (Figura 2.25). Este proceso tardará más de una hora y lo que se van a crear ahora son las librerías que posteriormente se podrán usar en los programas. Se puede ver cómo se van creando simultáneamente estas librerías en la carpeta “bin/debug”, que se habrá creado en el directorio *Where to build the binaries* (la carpeta donde están las VTK compiladas). Una vez hecho esto, se podrán incluir las librerías VTK en cualquier proyecto.

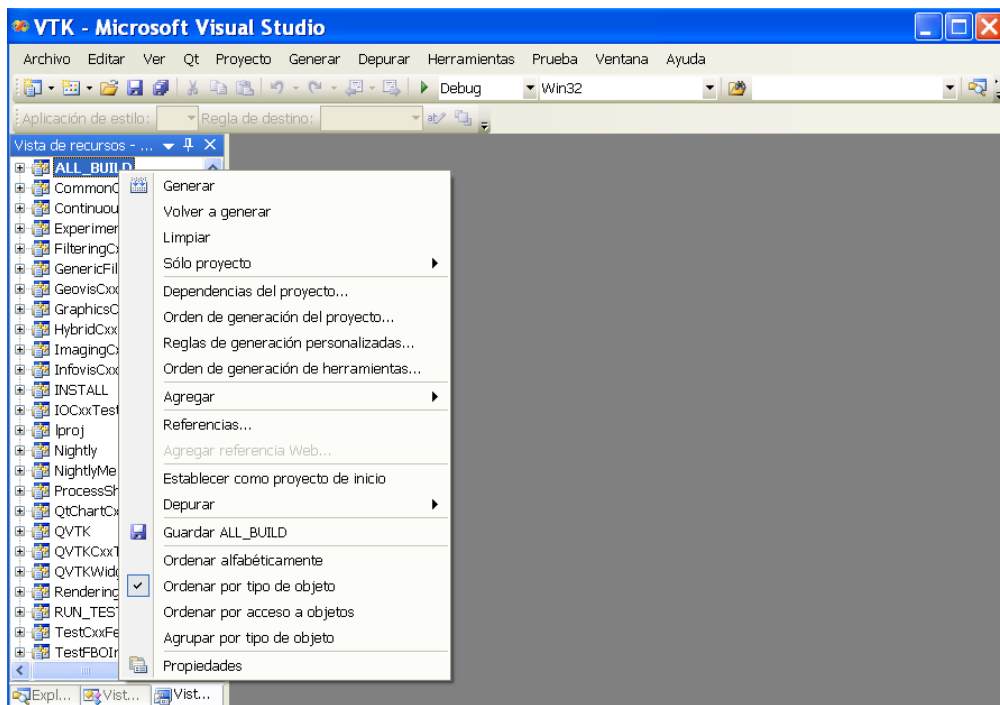


Figura 2.25: “Vtk.sln” - Visual Studio 2008.

2.5.3. Integración de VTK en Visual Studio 2008

El proceso para integrar las librerías VTK en Visual Studio es idéntico al explicado en el apartado 2.4.3 para las librerías ITK. Por tanto, las librerías se añadirán en dos pasos:

1. Añadir los directorios de inclusión adicionales:

- `$(PATH_VTK_INCLUDES)\Auxiliary\vtk"`
- `$(PATH_VTK_INCLUDES)\Common`
- `$(PATH_VTK_INCLUDES)\Common\Testing\Cxx`
- `$(PATH_VTK_INCLUDES)\Filtering`
- `$(PATH_VTK_INCLUDES)\GenericFiltering`
- `$(PATH_VTK_INCLUDES)\Geovis`
- `$(PATH_VTK_INCLUDES)\Graphics`
- `$(PATH_VTK_INCLUDES)\GUISupport\Qt`
- `$(PATH_VTK_INCLUDES)\IO`
- `$(PATH_VTK_INCLUDES)\Imaging`
- `$(PATH_VTK_INCLUDES)\Hybrid`
- `$(PATH_VTK_INCLUDES)\Infovis`
- `$(PATH_VTK_INCLUDES)\Rendering`
- `$(PATH_VTK_INCLUDES)\Rendering\Testing\Cxx`
- `$(PATH_VTK_INCLUDES)\Utilities`
- `$(PATH_VTK_INCLUDES)\Utilities\DICOMParser`
- `$(PATH_VTK_INCLUDES)\Utilities\MaterialLibrary`
- `$(PATH_VTK_INCLUDES)\Utilities\vtkalglib`
- `$(PATH_VTK_INCLUDES)\Utilities\verdict`
- `$(PATH_VTK_INCLUDES)\Utilities\vtkexodus2\include`
- `$(PATH_VTK_INCLUDES)\Utilities\vtkfreetype\include`
- `$(PATH_VTK_INCLUDES)\Utilities\vtklibproj4`
- `$(PATH_VTK_INCLUDES)\Utilities\vtknetcdf`
- `$(PATH_VTK_INCLUDES)\Views`
- `$(PATH_VTK_INCLUDES)\VolumeRendering`
- `$(PATH_VTK_INCLUDES)\Widgets`
- `$(PATH_VTK_LIBS)\Common`
- `$(PATH_VTK_LIBS)\GUISupport\Qt`

- `$(PATH_VTK_LIBS)\Rendering`
- `$(PATH_VTK_LIBS)\Utilities`
- `$(PATH_VTK_LIBS)\Utilities\DICOMParser`
- `$(PATH_VTK_LIBS)\Utilities\MaterialLibrary`
- `$(PATH_VTK_LIBS)\Utilities\verdict`
- `$(PATH_VTK_LIBS)\Utilities\vtkauglib`
- `$(PATH_VTK_LIBS)\Utilities\vtkexodus2\include`
- `$(PATH_VTK_LIBS)\Utilities\vtkfreetype\include`
- `$(PATH_VTK_LIBS)\Utilities\vtklibproj4`
- `$(PATH_VTK_LIBS)\Utilities\vtknetcdf`
- `$(PATH_VTK_LIBS)\VolumeRendering`

Las variables `$(PATH_VTK_INCLUDES)` y `$(PATH_VTK_LIBS)` hacen referencia a la ruta donde se encuentra el código fuente de las librerías VTK y las librerías compiladas, respectivamente, cómo crear dichas variables se ha explicado en el apartado 2.4.3.1

2. Añadir los directorios de bibliotecas adicionales

- En “General”: `$(PATH_VTK_LIBS)\bin\Debug` y `$(PATH_VTK_LIBS)\bin\Release`
- En “Entrada”:
 - `vtkRendering.lib`
 - `vtkGraphics.lib`
 - `vtkHybrid.lib`
 - `vtkImaging.lib`
 - `vtkIO.lib`
 - `vtkFiltering.lib`
 - `vtkCommon.lib`
 - `vtkftgl.lib`
 - `vtkfreetype.lib`
 - `vtkDICOMParser.lib`
 - `vtkpng.lib`

- `vtktiff.lib`
- `vtkzlib.lib`
- `vtkjpeg.lib`
- `vtkexpat.lib`
- `vtksys.lib`
- `vtkexoIIc.lib`
- `vtkNetCDF.lib`
- `vw32.lib`
- `vtkverdict.lib`
- `vtkmetaio.lib`
- `vtksqlite.lib`
- `QVTK.lib`
- `QVTKWidgetPlugin.lib`
- `vtkQtChart.lib`

Capítulo 3

Diseño e implementación de la interfaz gráfica de usuario

Este capítulo pretende ser una guía sobre nociones básicas de la programación orientada a objetos (POO) en C++ y en Qt, para explicar cómo crear un nuevo proyecto y el funcionamiento y estructura interna de la herramienta implementada, en función de dichos conceptos. Además, en este capítulo se explicará detalladamente cómo diseñar e implementar la interfaz de usuario. Para ello, es necesario crear un cuadro de diálogo mediante los diferentes objetos de Qt y posteriormente, dar funcionalidad a dichos objetos mediante las librerías ITK y VTK.

3.1. Programación orientada a objetos

3.1.1. Conceptos básicos de programación en C++

A las personas que tienen conocimientos básicos de programación orientada a objetos en C++, les resultará sencillo entender el método de desarrollo que se sigue en Qt en asociación con Visual Studio 2008, ya que se utilizan conceptos tales como:

- **Clases**

Una clase es una agrupación de atributos (datos) y métodos que operan sobre esos atributos.

- **Objetos**

Un objeto es la instancia de una clase concreta.

- **Herencia**

La herencia es la propiedad que permite contruir clases a partir de otras. Se utiliza para crear clases (hijas) que incorporen propiedades y métodos de otras clases (padres). Así, podremos construir clases a partir de otras sin tener que reescribirlo todo. Una clase nueva creada a partir de una clase existente será una subclase de esta clase ya existente.

En resumen, las clases forman una estructura jerárquica de clasificación, ya que los objetos heredan las propiedades y atributos de todas las clases a las que pertenecen.

- **Polimorfismo**

El polimorfismo es la capacidad que tienen los objetos de una clase para responder al mismo mensaje o evento en función de los parámetros utilizados durante su invocación.

A continuación se describirán los pasos básicos para crear una clase y un objeto perteneciente a dicha clase:

1. La forma básica para definir una clase es:

```
class nombre_clase
{
//Atributos de la clase
tipo_atributo nombre_atributo;
//Métodos de la clase
tipo_método nombre_método (parámetros_método);
}
```

Siendo “class” una palabra reservada para declarar una clase y las dos barras inclinadas “//” indicadores de que lo que sigue es un comentario.

2. Una vez definida la clase, se puede crear un objeto de la clase que hemos declarado de la siguiente forma:

```
class nombre_clase nombre_objeto;
```


3. La declaración de un método se realiza:

```
tipo_método nombre_clase::nombre_método((párametros_método)
{
...
}
```

Como veremos después, las definiciones de clases, así como las declaraciones de sus respectivos métodos, y la creación de objetos de clases se programarán en distintas partes del proyecto.

3.2. Cómo crear un proyecto nuevo en Visual Studio 2008

Para crear un proyecto en Visual Studio 2008 se abrirá la aplicación y se seleccionará el menú “Archivo”. Se hará *click* en “Nuevo” y después en “Proyecto”. A continuación, se mostrará una pantalla que nos indica los diferentes tipos de proyectos que podemos crear (Figura 3.1).

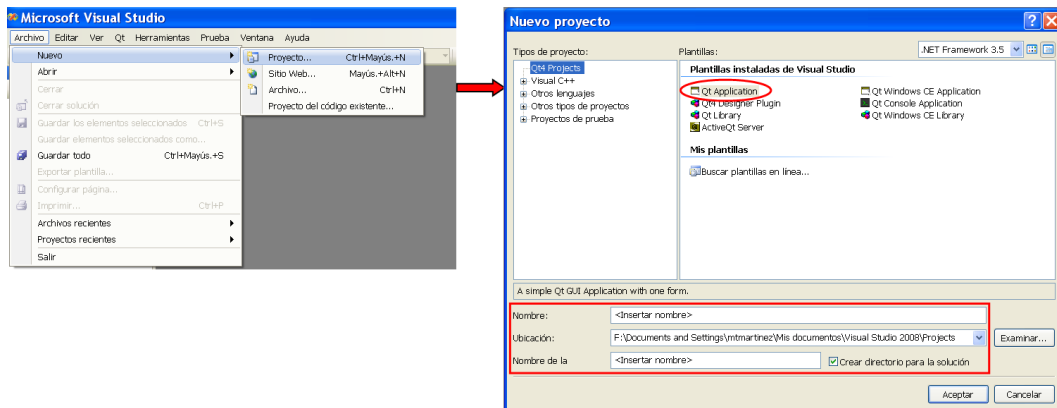


Figura 3.1: Primeros pasos para crear un proyecto nuevo.

Como tipo de proyecto se elegirá un proyecto basado en Qt: “Qt4 Projects” y después “Qt Application”. Habrá que indicar el nombre y la ubicación del proyecto (en el caso que se muestra, se ha elegido como nombre “nuevoproyecto”). Una vez hecho esto, se abrirá un asistente para la creación de un proyecto Qt.

Mediante el asistente se podrán incluir distintos módulos de librerías (SQL, OpenGL, XML etc), pero se recomienda utilizar las opciones por defecto. El último paso del asistente permite modificar los nombres de los archivos que se generarán automáticamente y la clase base a partir de la cual se generará el proyecto (Figura 3.2).

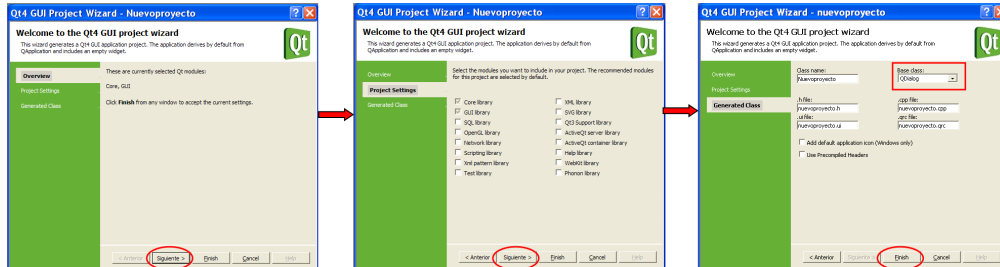


Figura 3.2: Asistente para nuevos proyectos Qt.

Al finalizar el asistente, en el menú llamado “Explorador de soluciones” se mostrarán todos los archivos generados automáticamente y que están incluidos dentro del nuevo proyecto (Figura 3.3).

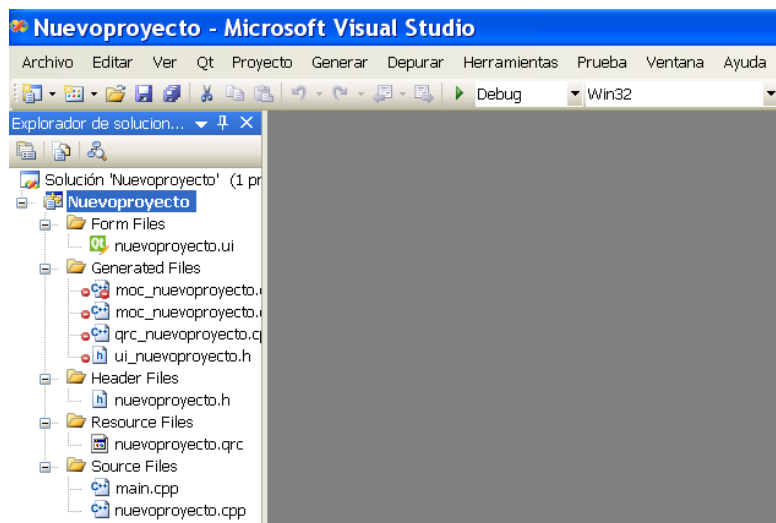


Figura 3.3: Archivos generados en una aplicación Qt.

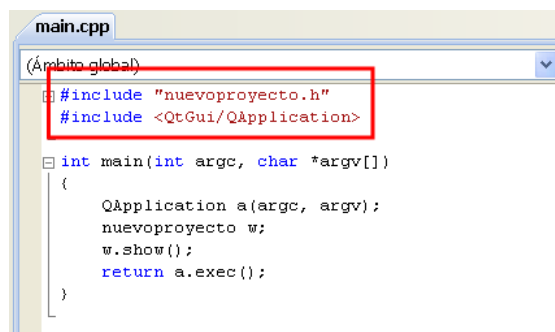
Los archivos generados (Figura 3.3) corresponden a uno de los siguientes tipos:

- **.h**: Los archivos .h sirven para declarar clases, las cuales se definen como hemos visto en el apartado anterior.
- **.cpp**: Los archivos .cpp sirven para implementar los métodos, tal y como hemos visto

anteriormente, asociados a las clases declaradas en los archivos .h.

- **.ui**: Los archivos .ui sirven para implementar una interfaz de usuario a través de la aplicación “Qt Creator”, sin embargo en el presente proyecto la implementación se realizará directamente por código mediante los archivos .cpp. Para diferenciar unos archivos .cpp de otros, utilizaremos el atributo “ui” (*user interface*) para hacer referencia a aquellos dónde se implementan los objetos de la interfaz de usuario.
- **.pro** y **.moc**: Los archivos .pro y .moc sirven para compilar el programa y se generan automáticamente, por lo que el usuario será ajeno a ellos.

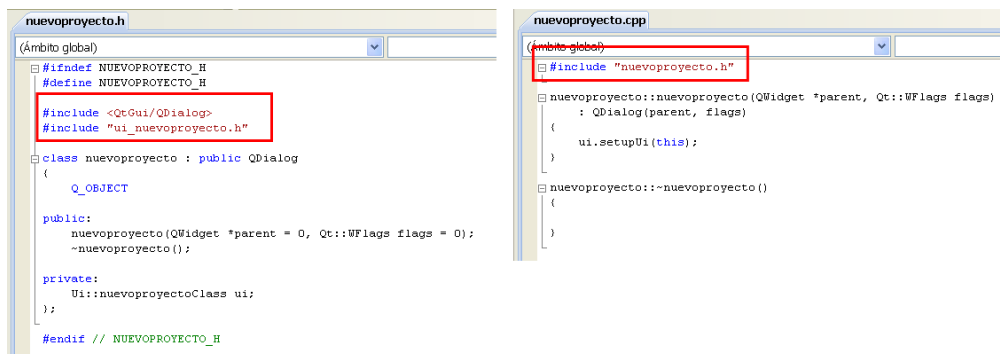
Al abrir los archivos generados podemos observar (Figuras 3.4, 3.5 y 3.6) que éstos han sido creados por defecto con una serie de instrucciones. Además, se han destacado las líneas que se deben añadir para incluir las librerías, ya que es conveniente tener en cuenta que se debe realizar una inclusión de los diferentes archivos que se generan y de las librerías que se utilicen en la aplicación. Para ello, bastará añadir “# include “archivo.extensión”” al principio de cada archivo.



```
main.cpp
(Ámbito global)
#include "nuevoproyecto.h"
#include <QtGui/QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    nuevoproyecto w;
    w.show();
    return a.exec();
}
```

Figura 3.4: Archivo “main” generado por defecto al crear un proyecto.



```
nuevoproyecto.h
(Ámbito global)
#ifndef NUEVOPROYECTO_H
#define NUEVOPROYECTO_H

#include <QtGui/QDialog>
#include "ui_nuevoproyecto.h"

class nuevoproyecto : public QDialog
{
    Q_OBJECT

public:
    nuevoproyecto(QWidget *parent = 0, Qt::WFlags flags = 0);
    ~nuevoproyecto();

private:
    Ui::nuevoproyectoClass ui;
};

#endif // NUEVOPROYECTO_H

nuevoproyecto.cpp
(Ámbito global)
#include "nuevoproyecto.h"

nuevoproyecto::nuevoproyecto(QWidget *parent, Qt::WFlags flags)
: QDialog(parent, flags)
{
    ui.setupUi(this);
}

nuevoproyecto::~nuevoproyecto()
{
}
```

Figura 3.5: Archivos “.h” y “.cpp” generados por defecto al crear un proyecto.

```

ui_nuevoproyecto.h
(Ambito global)
#ifndef UI_NUEVOPROYECTO_H
#define UI_NUEVOPROYECTO_H

#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QDialog>
#include <QtGui/QHeaderView>

QT_BEGIN_NAMESPACE
class Ui_nuevoproyectoClass
{
public:
    void setupUi(QDialog *nuevoproyectoClass)
    {
        if (nuevoproyectoClass->objectName().isEmpty())
            nuevoproyectoClass->setObjectName(QString::fromUtf8("nuevoproyectoClass"));
        nuevoproyectoClass->resize(600, 400);

        retranslateUi(nuevoproyectoClass);

        QObject::connectSlotsByName(nuevoproyectoClass);
    } // setupUi
    void retranslateUi(QDialog *nuevoproyectoClass)
    {
        nuevoproyectoClass->setWindowTitle(QApplication::translate("nuevoproyectoClass", "nuevoproyecto",
            0, QApplication::UnicodeUTF8));
        Q_UNUSED(nuevoproyectoClass);
    } // retranslateUi
};
namespace Ui {
class nuevoproyectoClass: public Ui_nuevoproyectoClass {}
} // namespace Ui
QT_END_NAMESPACE
#endif // UI_NUEVOPROYECTO_H

```

Figura 3.6: Archivos “.ui” generado por defecto al crear un proyecto.

3.3. Estructura interna de la aplicación

Una vez visto cómo se crea desde cero un proyecto nuevo, estamos en disposición para entender la estructura interna del proyecto objeto de este trabajo (Figura 3.7):

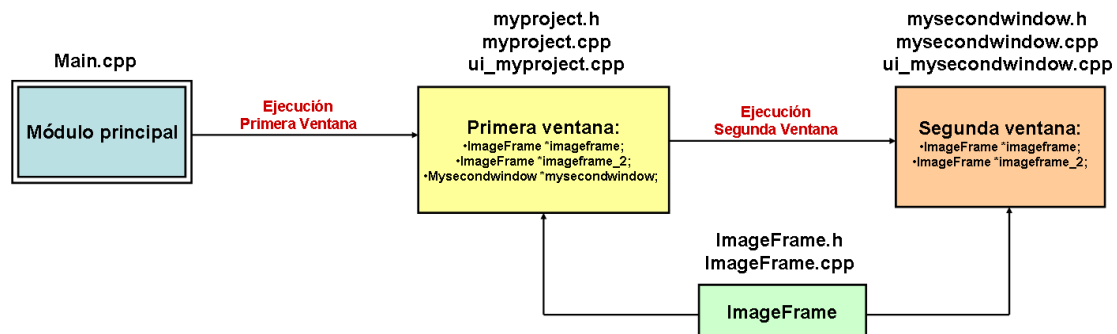


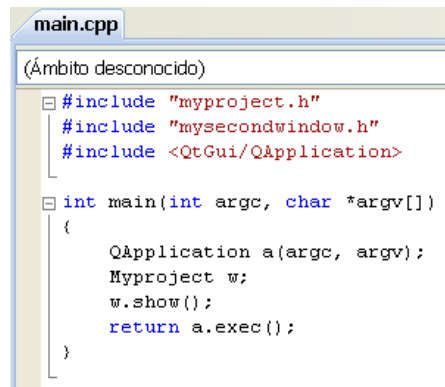
Figura 3.7: Diagrama de la estructura interna de la aplicación.

- **Módulo principal**

El programa contiene un módulo principal donde se encuentra la función *main*. El archivo *main.cpp* contiene el código para inicializar la aplicación y los archivos de cabecera que contienen la lógica de la aplicación personalizada y los componentes de la IGU. Se realiza sólo una instancia por aplicación y debe crearse antes de cualquier objeto relacionado con la IGU.

En el archivo *main.cpp* de la aplicación implementada en el presente proyecto se incluyen los archivos de cabecera para los componentes de aplicación. Posteriormente, se crea un objeto de la clase *QApplication* llamado “a”. La clase *QApplication* administra el flujo de control de la aplicación de la IGU. Contiene el bucle de eventos principal, donde todos los eventos del sistema de ventanas y otras fuentes son procesadas y enviadas. También se ocupa de la inicialización de la aplicación y finalización.

A continuación, se define una ventana de diálogo llamada “w”, se muestra la ventana “w” y se ejecuta la aplicación “a”. Por último, se hace una llamada al método *exec()* de la clase *QApplication* para iniciar el ciclo de eventos de Qt y de este modo el control pasa a Qt. El archivo *main.cpp* de la aplicación se define entonces, como en la Figura 3.8.



```
main.cpp
(Ámbito desconocido)
#include "myproject.h"
#include "mysecondwindow.h"
#include <QtGui/QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Myproject w;
    w.show();
    return a.exec();
}
```

Figura 3.8: Archivo “main.cpp” de la aplicación.

■ ImageFrame

La clase *ImageFrame* se ha creado con la finalidad de visualizar las imágenes DICOM. Se utilizará tanto en la primera ventana (*Myproject*) de la aplicación como en la segunda (*Mysecondwindow*). *Myproject* y *Mysecondwindow* son dos clases asociadas a las dos ventanas sobre las que está implementada la herramienta. En la clase *ImageFrame* se han definido todos los objetos, variables y funciones relacionadas con las imágenes que se van a procesar y visualizar en ambas ventanas.

ImageFrame hereda de la clase *QFrame*. Utilizamos esta clase para crear un marco o *frame* en que visualizar imágenes. En el capítulo 5 se explicará con mayor detalle en que consiste dicha clase *QFrame*.

Para visualizar una imagen DICOM se deben seguir los siguientes pasos:

1. Leer la imagen DICOM mediante las librerías ITK:

Para leer la imágenes se recorrerán todos los píxeles que la constituyen y sus valores se almacenará en un *buffer*, al cuál tendremos acceso mediante un puntero creado en la clase *ImageFrame*. Este paso está ampliamente explicado en [17].

2. Convertir la imagen de ITK a Qt:

A partir del puntero que tiene acceso a la imagen, se realiza una conversión de ITK a un objeto llamado “image” de la clase *QImage*. Esta clase es la utilizada en Qt para dibujar imágenes.

3. Dibujar y visualizar la imagen en la interfaz gráfica de usuario:

Para dibujar los objetos de tipo QImage se ha asociado un evento llamado “paintEvent” a la clase *ImageFrame* y que automáticamente dibuja una imagen dentro de unos objetos llamados *frames* que actúan a modo de marco para contener en ellos la imagen.

Además de visualizar las imágenes DICOM también se han implementado los métodos de segmentación de dichas imágenes mediante las librerías ITK [17] y los métodos para visualizar el volumen 3D a partir de dichas imágenes, gracias a las librerías VTK [18].

■ **Primera ventana**

Para implementar la primera ventana se ha definido la clase *Myproject*. En esta clase se definen todos los widgets y funciones que aparecen y se ejecutan al lanzar la aplicación.

■ **Segunda ventana**

Al igual que ocurre en la primera ventana, se ha definido una clase para la segunda ventana llamada *Mysecondwindow*. En esta clase se definen todos los widgets y funciones que aparecen y se ejecutan al lanzar la aplicación .

Se ha creado una variable de la clase *Mysecondwindow* en la clase *Myproject*, para tener acceso a todos los atributos y métodos de la segunda ventana. Mediante esta variable se lanzará la segunda ventana apretando un botón en la primera.

3.4. Cómo crear un cuadro de diálogo

QObject es la base del sistema de objetos de Qt. Todos los objetos y la mayoría de clases de Qt heredan de él. Por su parte, *QWidget* es la base de todos los objetos de cualquier interfaz de usuario. Los más comunes son *QMainWindow* y *QDialog* y sus respectivas subclases (Figura 3.9). En esta aplicación utilizaremos la clase *QDialog*.

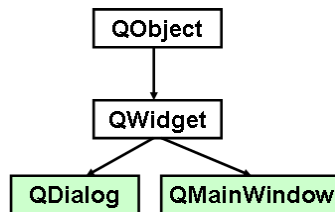


Figura 3.9: Clase QWidget

QDialog es la clase base de cualquier ventana de diálogo. Una ventana de diálogo es una ventana utilizada para tareas y comunicaciones con el usuario. Para poder crear una ventana de diálogo, se debe incluir en el archivo *main.cpp* la siguiente instrucción:

`#include <QtGui>`, donde *QtGui* es la colección básica de componentes gráficos para crear cualquier interfaz.

3.4.1. Widgets

Los *widgets* son los bloques de construcción básicos de la interfaz gráfica de usuario, es decir, un *widget* es todo aquel elemento visual dentro de una interfaz de usuario. Los botones y etiquetas son ejemplos de *widgets*. Cada componente de la interfaz gráfica de usuario puede ser colocado dentro de una interfaz existente o mostrarse como una ventana independiente.

Cada tipo de componente es proporcionada por una subclase particular de *QWidget*, que es en sí misma una subclase de *QObject*, siendo *QObject* a su vez, la clase base del sistema de objetos de Qt (Figura 3.10). Todo *widget* y la mayoría de las otras clases de Qt heredan de él.

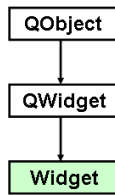


Figura 3.10: Clase QWidget

3.4.2. Eventos, signals y slots

Las acciones del usuario sobre los *widgets* generan eventos (también llamados mensajes) que el programa puede responder normalmente ejecutando una o más funciones. Por ejemplo, cuando un usuario aprieta o hace *click* en un *widget*, un evento *mouse press* y un *mouse release* se generan.

Los *widgets* emiten señales llamadas *signals*, para indicar que se ha producido una acción del usuario o ha ocurrido un cambio de estado. Por ejemplo, cuando el usuario aprieta un botón, que es un objeto perteneciente a la clase *QPushButton*, éste emite una señal llamada *clicked()*. Serían también ejemplos de *signals* elegir una opción del menú, abrir o cerrar una ventana, etc.

Una señal se puede conectar a una función llamada *slot*. Así pues, cuando la señal es emitida, el slot es ejecutado automáticamente. Normalmente, la manera de hacerlo sería:

connect(widget, SIGNAL(), widget, SLOT())

Donde *signal* y *slot* corresponderían a la señal que emite el widget y al *slot* que se produce cuando la señal ha sido emitida. También cabe la posibilidad de poder conectar una o varias señales a un único *slot* y también una o varias señales a varios *slots*, como se muestra en la Figura 3.11.

A continuación se van a analizar los widgets utilizados en la aplicación para diseñar la interfaz gráfica de usuario.

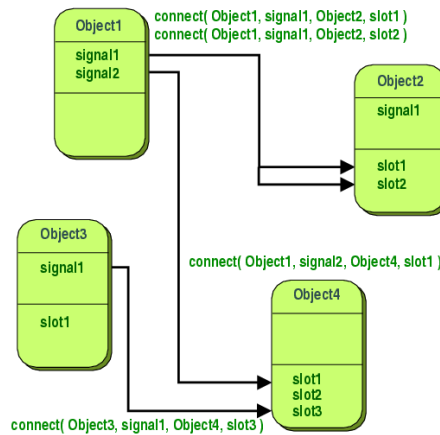


Figura 3.11: Cómo conectar *signals* y *slots*.

3.4.3. Crear un botón

Para crear un botón se creará un objeto de la clase *QPushButton*. Esta clase pertenece a su vez a la clase *QAbstractButton* que es utilizada para crear diferentes tipos de botones (Figura 3.12).

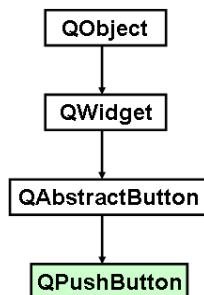


Figura 3.12: Clase *QPushButton*

Un *pushButton* es un botón de comandos. Quizás sea el *widget* más utilizado en cualquier interfaz de usuario, ya que es sencillo hacer que al activar dicho botón se pueda ordenar al ordenador que realice alguna acción o responda a alguna pregunta.

Un *pushbutton* emite la señal *clicked()* cuando es activado mediante el ratón o el teclado. Por tanto, conectaremos esta señal con la acción que deseemos asociar al botón.

Un *pushbutton* es rectangular y normalmente muestra una etiqueta de texto describiendo su acción y opcionalmente un icono. Para ello utilizaríamos las propiedades *setText()* y *setIcon()*.

3.4.3.1. Ejemplo de botón

Para crear un botón como el de la Figura 3.13 se deben incluir las siguientes instrucciones en el programa:



Figura 3.13: Ejemplo de botón.

1. `QPushButton *openButton;`
2. `openButton = new QPushButton(MyprojectClass);`
3. `openButton->setObjectName(QString::fromUtf8("openButton"));`
4. `openButton->setGeometry(QRect(40, 630, 100, 25));`
5. `openButton->setCheckable(true);`
6. `openButton->setText(QApplication::translate("MyprojectClass", "OPEN SEGMENTED IMAGES", 0, QApplication::UnicodeUTF8));`
7. `QFont font;`
`font.setBold(true);`
`openButton->setFont(font);`

Mediante las instrucciones (1) y (2) se crea el botón llamado "openbutton" de la clase *QPushButton* y que a su vez pertenece a la clase *MyprojectClass*. *MyprojectClass* es la clase principal de la aplicación de este proyecto, asociada a la primera pantalla de la herramienta. La estructura de la programación de la aplicación se explicará detalladamente en el capítulo 4.

En las instrucciones (3), (4), (5), (6) y (7) se establecen algunas propiedades del botón: ponerle el nombre al objeto, definir su dimensión, habilitar el botón, poner el texto que llevará el botón y por último, poner en negrita el texto.

Para todos los ejemplos sobre *widgets* que aparecen en este capítulo conviene remarcar que existen más propiedades que se podrían aplicar y para ello, se recomienda consultar el asistente de Qt que proporciona el software, junto con la referencia [19].

Para conectar el botón con una función que se ejecute al apretarlo:

```
QObject::connect ( openButton, SIGNAL (clicked ()), MyprojectClass, SLOT (click-  
openButton ())) ;
```

En este caso, al apretar el botón `openButton` se emitirá la señal `clicked()` y se ejecutará la función `clickopenButton()` de la clase `MyprojectClass`.

3.4.4. Crear una etiqueta

Para crear una etiqueta se creará un objeto de la clase `QLabel` (Figura 3.14). La clase `QLabel` se utiliza para mostrar un texto o una imagen. El aspecto visual de la etiqueta se puede configurar de varias maneras. Es importante remarcar que cuando el contenido se cambia cualquier contenido anterior se borra.

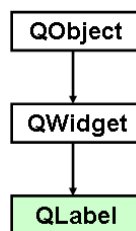


Figura 3.14: Clase QLabel

3.4.4.1. Ejemplo de etiqueta

Para crear una etiqueta como la de la Figura 3.15 se deben incluir las siguientes instrucciones en el programa:

IMAGE SLICE INDEX

Figura 3.15: Ejemplo de etiqueta.

1. `QLabel *label;`
2. `label = new QLabel(MyprojectClass);`
3. `label->setObjectName(QString::fromUtf8("label"));`
4. `label->setGeometry(QRect(40, 600, 120, 20));`
5. `label->setText(QApplication::translate("MyprojectClass", "IMAGE SLICE INDEX", 0, QApplication::UnicodeUTF8));`

Mediante las instrucciones (1) y (2) se crea la etiqueta llamada "label" de la clase *QLabel* y que a su vez pertenece a la clase *MyprojectClass*. Mediante (3) y (4) definimos el nombre del objeto, su tamaño y el texto que aparecerá en la etiqueta. En las etiquetas no podemos utilizar la función *connect* ya que no se puede interactuar con ninguna ellas.

3.4.5. Crear una barra deslizador

Para crear una barra deslizador se creará un objeto de la clase *QSlider*, que a su vez pertenece a la clase *QAbstractSlider* (Figura 3.16). La clase *QAbstractSlider* proporciona un valor entero dentro de un rango mientras que *QSlider*, más específicamente, permite crear una barra deslizador vertical u horizontal. Se utiliza para el control de un valor limitado. Permite al usuario mover este control deslizante vertical u horizontalmente, y la posición seleccionada se traduce en un valor. *QSlider* sólo proporciona un rango de valores enteros. Cuando el valor ha cambiado se emite la señal *valueChanged()*.

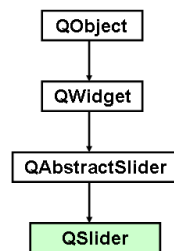


Figura 3.16: Clase *QSlider*.

Las funciones más útiles para definir una barra deslizador son:

- `setValue()` para establecer el deslizador directamente a un cierto valor.
- `setSingleStep()` y `setPageStep()` para establecer los pasos.
- `setMinimum()` y `setMaximum()` para definir el rango de la barra de desplazamiento.

3.4.5.1. Ejemplo de una barra deslizador

Para crear una barra deslizador como la de la Figura 3.17 de deben incluir las siguientes instrucciones en el programa:

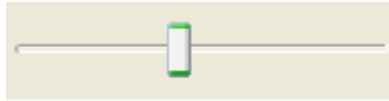


Figura 3.17: Ejemplo de barra deslizador.

1. `QSlider *horizontalSlider;`
2. `horizontalSlider = new QSlider(MyprojectClass);`
3. `horizontalSlider->setObjectName(QString::fromUtf8("horizontalSlider"));`
4. `horizontalSlider->setGeometry(QRect(160, 600, 170, 25));`
5. `horizontalSlider->setSliderPosition(0);`
6. `horizontalSlider->setOrientation(Qt::Horizontal);`
7. `horizontalSlider->setPageStep(1);`

Mediante las instrucciones (1) y (2) se crea la barra deslizador llamada “horizontalSlider” de la clase *QSlider* y que a su vez pertenece a la clase *MyprojectClass*.

En las instrucciones (3), (4), (5), (6) y (7) se establecen algunas propiedades: ponerle el nombre al objeto, definir su dimensión, indicar la posición inicial, definir la orientación (horizontal, en este ejemplo) y el salto que dará al desplazarse.

Para conectar la barra deslizador con una función que se ejecute al desplazarse sobre ella:

```
QObject::connect( horizontalSlider, SIGNAL (valueChanged(int)), MyprojectClass,  
SLOT (slider(int)));
```

En este caso, al desplazar la barra se emitirá la señal *valueChanged(int)*, indicando que el valor ha cambiado y se ejecutará la función *slider(int)* de la clase *MyprojectClass*.

3.4.6. Crear un *spinbox*

Para crear un *spinbox* se creará un objeto de la clase *QSpinBox*, que a su vez pertenece a la clase *QAbstractSpinBox*. Esta clase proporciona un cuadro y una línea de edición para mostrar los valores (Figura 3.18), mientras que la clase *QSpinBox* está diseñada para manejar enteros y conjuntos de valores discretos, por tanto proporciona un cuadro para introducir un número.

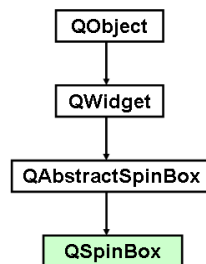


Figura 3.18: Clase *QSpinBox*.

QSpinBox permite al usuario elegir un valor haciendo *click* en los botones arriba/abajo o pulsando la tecla de arriba/abajo en el teclado para aumentar o disminuir el valor que se muestra actualmente. El usuario puede también escribir el valor en forma manual.

Cada vez que cambia el valor de *QSpinBox* emite la señal *valueChanged()*. El valor actual se puede obtener con el *value()* y establecer con *setValue()*.

Al igual que ocurría en *QSlider*, podemos utilizar las funciones *setSingleStep()* y *setPageStep()* para establecer los pasos y *setMinimum()* y *setMaximum()* para definir el rango.

3.4.6.1. Ejemplo de un *spinbox*

Para crear un *spinbox* como la de la Figura 3.19 se deben incluir las siguientes instrucciones en el programa:



Figura 3.19: Ejemplo de *spinbox*.

1. `QSpinBox *spinBox;`
2. `spinBox = new QSpinBox(MyprojectClass);`
3. `spinBox->setObjectName(QString::fromUtf8("spinBox"));`
4. `spinBox->setGeometry(QRect(335, 600, 45, 25));`
5. `spinBox->setValue(0);`

De igual modo que ocurría en la barra deslizador, mediante las instrucciones (1) y (2) se crea el *spinbox* llamado “spinBox” de la clase *QspinBox* y que a su vez pertenece a la clase *MyprojectClass*.

En las instrucciones (3), (4) y (5) se establecen algunas propiedades: ponerle el nombre al objeto, definir su dimensión e indicar el valor inicial.

Para conectar el *spinbox* con una función que se ejecute al cambiar su valor:

```
QObject::connect( spinBox, SIGNAL( valueChanged(int)), horizontalSlider, SLOT(
setValue(int)));
```

3.4.7. Crear un *combobox*

Un *widget* de la clase *QComboBox* (Figura 3.20) permite crear un botón combinado y una lista emergente. La clase *QComboBox* proporciona un medio de presentar una lista de opciones para el usuario de manera que tenga la cantidad mínima de espacio en la pantalla.

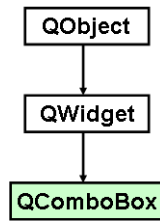


Figura 3.20: QComboBox.

Un *combobox* es un widget de selección que muestra el elemento actual, y puede mostrar una lista de elementos seleccionables. Puede ser modificable, permitiendo al usuario modificar cada elemento de la lista. Puede contener mapas de píxeles, así como cadenas. La inserción se realiza mediante *insertItem()*.

Si se produce un cambio en el elemento actual del *combobox*, se emiten dos señales *currentIndexChanged()* y *activated()*:

- *currentIndexChanged()* se emite siempre, independientemente si el cambio se hizo mediante programación o por la interacción del usuario.
- *activated()* sólo se emite cuando el cambio es causado por la interacción del usuario.

3.4.7.1. Ejemplo de un *combobox*

Para crear un *combobox* como la de la Figura 3.21 se deben incluir las siguientes instrucciones en el programa:

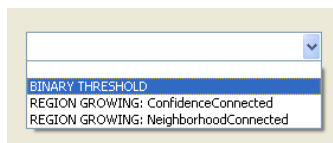


Figura 3.21: Ejemplo de *combobox*.

1. `QComboBox *comboBox;`
2. `comboBox = new QComboBox(MysecondwindowClass);`
3. `comboBox->setObjectName(QString::fromUtf8("comboBox"));`

4. `comboBox->setGeometry(QRect(170, 670, 250, 25));`

5. `comboBox->hide();`

Mediante las instrucciones (1) y (2) se crea el *comboBox* llamado “comboBox” de la clase *QcomboBox* y que a su vez pertenece a la clase *MysecondwindowClass*. *MysecondwindowClass* es la clase asociada a la segunda ventana.

En las instrucciones (3), (4) y (5) se establecen algunas propiedades: ponerle el nombre al objeto, definir su dimensión y ocultar el objeto hasta que se produzca algún evento (como por ejemplo, apretar otro botón para que se haga visible el *comboBox*).

Para conectar un *item* del *comboBox* con una función que realizara un método de segmentación:

```
QObject::connect( comboBox, SIGNAL( highlighted(int)),MysecondwindowClass, SLOT(
segmentation(int)));
```

3.4.8. Crear un *frame*

Tal y como se vió en apartados anteriores se ha utilizado la clase *QFrame* como clase padre para definir la clase *ImageFrame*. La clase *QFrame* (Figura 3.22) es la clase base de los *widgets* que pueden tener un marco. Un marco tiene tres atributos que describen el grosor del borde:

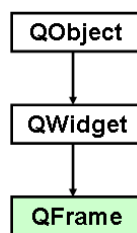


Figura 3.22: Clase *QFrame*.

- Anchura de línea: es el ancho del borde del marco. Se puede modificar para personalizar el aspecto del cuadro.

- *midLineWidth*: es la mitad de ancho de la línea. Especifica el ancho de una línea adicional en el centro del marco, que utiliza un tercer color para obtener un efecto 3D especial.
- *frameWidth*: es el ancho del marco, que viene determinado por el estilo de marco.

La Figura 3.23 muestra algunas de las combinaciones de estilos y anchuras de línea:

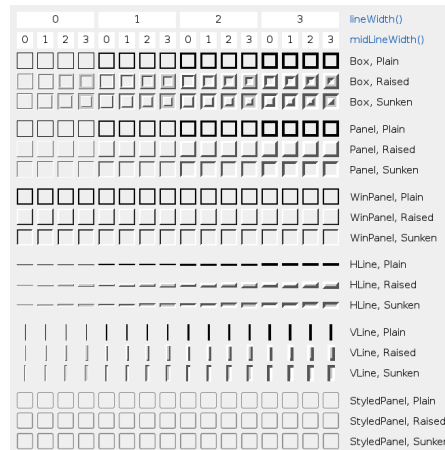


Figura 3.23: Estilos y líneas de QFrame.

3.4.8.1. Ejemplo de un *frame*

Para crear un *frame* como la de la Figura 3.24 deberemos incluir las siguientes instrucciones en el programa:

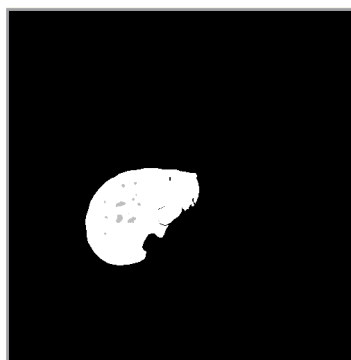


Figura 3.24: Ejemplo de segmentación de un hígado visualizada en un *frame*.

1. `ImageFrame *imageframe;`
2. `imageframe = new ImageFrame(MyprojectClass);`
3. `imageframe->setObjectName(QString::fromUtf8("imageframe"));`
4. `imageframe->setGeometry(QRect(40, 40, 512, 512));`
5. `imageframe->setVisible(true);`

Mediante las instrucciones (1) y (2) se crea el *frame* llamado “imageframe” de la clase *ImageFrame*. La clase *ImageFrame* es una clase definida para poder visualizar las imágenes DICOM tanto en la primera como en la segunda ventana.

En las instrucciones (3) y (4) se establecen algunas propiedades: ponerle el nombre al objeto, definir su dimensión y que sea visible.

3.5. Clases de ITK utilizadas

En esta sección, se destacan las clases de ITK empleadas en el proyecto por orden alfabético:

1. `itkAddImageFilter`

Esta clase implementa un operador de píxeles para poder sumar dos imágenes. Esta clase está parametrizada sobre los tipos de las dos imágenes de entrada y el tipo de la imagen de salida. El tipo resultante de la suma, se convierte en el tipo de píxel de la imagen de salida.

El tipo de píxel de la imagen de la entrada “1” debe tener una definición válida para poder sumarlo al tipo de píxel de la imagen “2”. Esta condición es necesaria porque el interior de este filtro se realice la operación:

$$\text{píxel de salida} = \text{píxel de la imagen "1"} + \text{píxel de la imagen "2"}$$

Por ejemplo, este filtro puede ser utilizado directamente para añadir imágenes cuyos píxeles son los vectores de la misma dimensión, y para almacenar el vector resultante en una imagen de salida de los píxeles de vectores.

Las imágenes que se añaden se establecen utilizando los métodos: `SetInput1 (image1);` y `SetInput2 (imagen2);`

2. `itkBinaryThresholdImageFilter`

Este filtro se utiliza para transformar una imagen en una imagen binaria cambiando los valores de los píxeles de acuerdo con dos umbrales definidos por el usuario que se definen empleando los métodos `SetUpperThreshold()` y `SetLowerThreshold()` y dos intensidades que se definen mediante `SetInsideValue()` y `SetOutsideValue()`. El valor de cada píxel de la imagen de entrada es comparado con el umbral alto y bajo. Si el valor de píxel esta en el interior del rango definido por $[Lower, Upper]$ la salida de píxel se le asigna a *Inside Value*. En caso contrario la salida del píxel se le asigna a *Outside Value*.

Este filtro espera que tanto las imágenes de entrada y de salida tenga el mismo número de dimensiones.

3. **itkCastImageFilter**

Esta clase se utiliza para hacer conversión de píxeles, para después poder operar con el segundo tipo de píxel.

4. **itkConfidenceConnectedImageFilter**

Esta clase segmenta los píxeles utilizando el método “Confidence Connected” de crecimiento de regiones.

Este filtro extrae un conjunto conectado de píxeles cuya intensidad de píxel coincide con la de los píxeles de un punto semilla. Píxeles conectados a este punto, cuyos valores se encuentran dentro del intervalo de confianza están agrupados. La anchura del intervalo de confianza es controlada por la variable “multiplicador” (el intervalo de confianza es la media más o menos el “multiplicador” veces la desviación estándar).

Después de esta segmentación inicial se recalcula la media y la varianza. Todos los píxeles de la segmentación anterior se utilizan para calcular la media de la desviación estándar (en lugar de utilizar los píxeles de la vecindad del punto de lanzamiento). La segmentación se vuelve a calcular utilizando estas estimaciones refinadas para la media y la varianza de los valores de píxel. Este proceso se repite para el número especificado de iteraciones.

El límite inferior y superior se limita a estar dentro de los límites de validez numérica de los datos de entrada de tipo píxel. Además, los límites pueden ser ajustados para contener la intensidad del punto de semilla.

5. **itkCurvatureFlowImageFilter**

Esta clase elimina el ruido de una imagen utilizando *Curvature Flow*.

6. **itkGDCMImageIO**

Esta clase utiliza la clase *ImageIO* para la lectura y escritura de imágenes DICOM.

7. **itkGDCMSeriesFileNames**

Esta clase genera una secuencia de ficheros cuyos nombres provienen de un fichero de imágenes DICOM. Sigue el siguiente procedimiento: lee todas las imágenes contenidas en un directorio (asumiendo que sólo hay una serie) y extrae la orientación y la posición, a partir de las cuales extrae el valor de la coordenada 3D del corte.

8. **itkImage**

Esta clase representa una imagen con una dimensión y un tipo de píxel. Las imágenes son modeladas como arrays y por tanto, se definen con un índice inicial y un tamaño. Conviene destacar que en ITK, una región de la imagen puede ser procesada por otras clases en el sistema.

9. **itkImageFileReader**

Esta clase lee los datos de una imagen desde un único fichero, que puede tener distintos formatos.

10. **itkImageSeriesReader**

Esta clase lee los datos de una imagen y construye imágenes n-dimensionales a partir de series de ficheros. Estos ficheros almacenados en un vector de caracteres se leen usando la clase `itkImageFileReader`. El formato de los ficheros puede variar entre ellos, pero los datos de las imágenes deben tener el mismo tamaño en todas sus dimensiones.

11. **itkImageFileWriter**

Esta clase escribe los datos que recibe como entrada en un único fichero de salida.

12. **itkImageSeriesWriter**

Esta clase escribe los datos de una imagen de entrada a una serie de archivos de salida. Por lo general, el tipo de imagen de salida tendrá menos dimensiones que el tipo de imagen de entrada. Cada archivo tiene un nombre creado mediante *SeriesFormat*. Esta cadena se utiliza como argumento para construir un nombre de archivo.

13. **itkRescaleIntensityImageFilter**

Esta clase aplica una transformación lineal a los niveles de intensidad de la imagen de entrada. La transformación lineal se define por el usuario en términos de valores máximos y mínimos que la imagen de salida debe tener.

3.6. Clases de VTK utilizadas

En esta sección, se destacan las clases de VTK empleadas en el proyecto por orden alfabético:

1. **vtkActor**

Esta clase representa un objeto mediante su geometría y propiedades en un escena.

2. **vtkDecimatePro**

vtkDecimatePro es un filtro utilizado para reducir el número de triángulos en una malla triangular, formando una buena aproximación a la geometría original. La entrada a vtkDecimatePro es un objeto vtkPolyData y se tratan sólo los triángulos.

3. **vtkImageCast**

vtkImageCast es un filtro que convierte el tipo de datos de entrada con el tipo de salida en la tubería de procesamiento de una imagen. El filtro no hace nada si la entrada ya tiene el tipo correcto.

4. **vtkImageGaussianSmooth**

Esta clase implementa una convolución gaussiana de la imagen de entrada.

5. **vtkImageMarchingCubes**

Esta clase genera isosuperficies de volúmenes e imágenes. vtkImageMarchingCubes es un filtro que toma como ejemplo las imágenes de entrada y genera a la salida una o más isosuperficies. Uno o más valores de contorno deben ser especificados para generar las isosuperficies. Alternativamente, se puede especificar un rango escalar mínimo y máximo y el número de curvas de nivel para generar una serie de valores de contorno espaciados.

Conviene remarcar que este filtro está especializado en volúmenes. Para el contorno de otros tipos de datos, se utilizará vtkContourFilter.

6. **vtkImageThreshold**

Esta clase puede hacer un umbral binario o continuo para superior, inferior o un rango de datos. El tipo de salida de datos puede ser diferente de la entrada, pero por defecto del mismo tipo.

7. **vtkPolyData**

Esta clase es un conjunto de datos concretos que representa una estructura geométrica compuesta de vértices, líneas, polígonos, y/o tiras de triángulo. Puntos, valores escalares y vectores también se representan.

8. **vtkPolyDataNormals**

`vtkPolyDataNormals` es un filtro que calcula las rectas normales para una malla poligonal. El filtro puede reordenar polígonos para asegurar la orientación consistente a través de los polígonos vecinos.

El algoritmo funciona mediante la determinación de las rectas normales a cada polígono y luego un promedio de ellos en los puntos compartidos. Cuando los bordes afilados están presentes, los bordes se dividen y se generan nuevos puntos para evitar bordes borrosos.

9. **vtkProperty**

Esta clase representa la iluminación y otras propiedades de la superficie de un objeto geométrico. Las características principales que se pueden establecer son los colores (en general, ambiente, difuso, especular, y el color de borde), la opacidad del objeto y la representación del objeto (puntos o superficies). Además, algunas características especiales se pueden ajustar y manipular con este objeto.

10. **vtkRenderer**

Esta clase proporciona una especificación abstracta para “renderers”. Un “renderer” es un objeto que controla el proceso de “rendering” de los objetos. De representación es el proceso de conversión de la geometría, una especificación de las luces, y una vista de cámara en una imagen. `vtkRenderer` también lleva a cabo la transformación de coordenadas entre las coordenadas del mundo, ver coordenadas (el equipo de procesamiento de gráficos sistema de coordenadas), y visualizar las coordenadas (la pantalla de coordenadas reales en el dispositivo de visualización). Algunas características avanzadas de renderizado como dos caras de iluminación también puede ser controlado.

11. **vtkRenderWindow**

Esta clase crea una ventana para “renderers” en la que dibujar dentro.

12. **vtkRenderWindowInteractor**

`vtkRenderWindowInteractor` proporciona una plataforma de mecanismo de interacción independiente para el ratón, las teclas y los eventos temporales.

Capítulo 4

Resultados

4.1. Interfaz Gráfica de Usuario

Las principales funcionalidades de la interfaz gráfica de usuario son:

- Leer y visualizar imágenes médicas.
- Segmentar imágenes médicas mediante:
 - Umbralización binaria (Binary threshold)
 - Crecimiento de regiones (Region Growing)
- Visualizar volúmenes 3D a partir de imágenes médicas previamente segmentadas.
- Guardar imágenes segmentadas o volúmenes 3D de dichas imágenes segmentadas.

4.2. Guía de utilización para usuarios

La aplicación desarrollada consta de dos ventanas, una dedicada a la reconstrucción tridimensional del hígado a partir de imágenes DICOM segmentadas y otra dedicada a la segmentación de imágenes DICOM mediante diferentes métodos. A continuación se detallará el funcionamiento de cada una de dichas ventanas.

4.2.1. Primera ventana

La primera ventana se denomina “**IMAGE VISUALIZER – 3D**” y en ella el usuario puede elegir entre las siguientes funcionalidades:

1. OPEN MENU

El botón “OPEN MENU” hace visible un menú con los siguientes botones:

a) OPEN SEGMENTED IMAGES:

El botón “OPEN SEGMENTED IMAGES” abre un directorio de imágenes segmentadas manualmente o mediante distintos métodos de segmentación de la aplicación y las muestra en el primer frame denominado “INPUT DICOM”. Los métodos de segmentación mediante los cuales se pueden segmentar en esta aplicación se explicarán en el siguiente punto.

b) OPEN NON-SEGMENTED IMAGES:

El botón “OPEN SEGMENTED IMAGES” abre la segunda pantalla donde se segmentan las imágenes.

2. « SEGMENTED DICOM »

En este *frame* se visualizan las imágenes segmentadas abiertas con el botón “OPEN SEGMENTED IMAGES”.

3. IMAGE SLICE INDEX

La barra deslizador y el *spinbox* “IMAGE SLICE INDEX” recorren las distintas imágenes pertenecientes al directorio de imágenes que se ha abierto en el primer *frame* “SEGMENTED DICOM”.

4. 3D

El botón “3D” realiza el volumen 3D a partir de las imágenes segmentadas que se representan en el *frame* “SEGMENTED DICOM”.

5. « 3D »

En este *frame* se visualiza el volumen 3D creado a partir de las imágenes segmentadas que se representan en el *frame* “SEGMENTED DICOM”.

Se ha implementado un código de colores para visualizar las diferentes partes del hígado:

- **Amarillo:** corresponde con la superficie del hígado y se verá semitransparente.
- **Rojo:** corresponde con el portal hepático y se verá opaco.
- **Verde:** corresponde con la vesícula y se verá opaca.

6. SAVE

El botón “SAVE” guarda las capturas de pantalla pertenecientes a los diferentes tejidos representados tridimensionalmente en el segundo *frame*: hígado, portal hepático y vesícula. El usuario tan sólo deberá seleccionar la carpeta donde desee que se guarden. Dentro de dicha carpeta seleccionada por el usuario se encontrarán tres ficheros con extensión *.STL* denominados: “HIGADO.STL”, “PORTALHEP-ATICO.STL” y “VESICULA.STL”, correspondientes a los tejidos que su propio nombre indica.

La extensión *.STL* se podrá abrir con cualquier programa de diseño gráfico o aplicaciones CAD, como 3Dmax, CATIA, SAT, etc.

7. EXIT

El botón “EXIT” permite salir de la aplicación.

8. Barra de progreso

La barra de progreso sirve para indicar al usuario que las imágenes se han cargado al 100 % en el *frame* “SEGMENTED DICOM”.

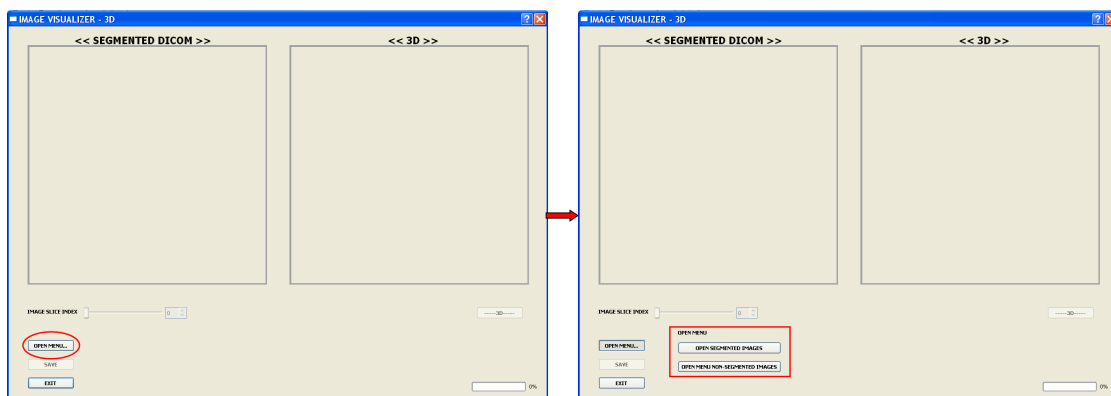


Figura 4.1: Visualización de la primera ventana.

4.2.2. Segunda ventana

La segunda ventana se denomina “**IMAGE VISUALIZER – SEGMENTATION**” y en ella el usuario puede elegir entre las siguientes funcionalidades:

1. OPEN

El botón “OPEN” abre un directorio de imágenes no segmentadas y las muestra en el primer *frame* denominado “INPUT DICOM”.

2. « INPUT DICOM »

En este *frame* se visualizan las imágenes a las que se les aplicarán los métodos de segmentación.

3. IMAGE SLICE INDEX

La barra deslizadora y el *spinbox* “IMAGE SLICE INDEX” recorren las distintas imágenes pertenecientes al directorio de imágenes que se ha abierto en el primer *frame* “INPUT DICOM”.

4. SEGMENTATION

El botón “SEGMENTATION” abre un *combobox* que permite seleccionar entre los métodos de segmentación implementados y realizar dicha segmentación sobre las imágenes mostradas en el *textitframe* “INPUT DICOM” y a continuación, mostrarlas en el segundo *frame* denominado “SEGMENTED DICOM”.

- Umbralización binaria (Binary threshold)
- Crecimiento de regiones (Region Growing)

Conviene resaltar que según el método elegido se requieren ciertas acciones adicionales. En el caso de la umbralización binaria habrá que seleccionar los umbrales máximos y mínimos de los tejidos del hígado, mientras que para el crecimiento de regiones habrá que situarse encima de la imagen y hacer *click* para proporcionarle al método un punto semilla.

5. « SEGMENTED DICOM »

En este *frame* se visualizan las imágenes segmentadas.

6. LIVER MAXIMUM Y MINIMUM

Estas barras deslizadoras y *spinboxes* permiten elegir los valores máximos y mínimos para realizar la umbralización binaria en la superficie del hígado.

7. HEPATIC HILUM MAXIMUM Y MINIMUM

Estas barras deslizadoras y *spinboxes* permiten elegir los valores máximos y mínimos para realizar la umbralización binaria en hilio hepático.

8. GALLBLADDER MAXIMUM Y MINIMUM

Estas barras deslizadoras y *spinboxes* permiten elegir los valores máximos y mínimos para realizar la umbralización binaria en la vesícula.

9. SAVE

El botón “SAVE” guarda las imágenes segmentadas en un único fichero con extensión *.dcm*. Es importante remarcar que el usuario tendrá que introducir el nombre del fichero junto a la extensión *.dcm*.

10. EXIT

El botón “EXIT” permite salir de la aplicación.

11. Barra de progreso

La barra de progreso sirve para indicar al usuario que las imágenes se han cargado al 100 % en el *frame* “INPUT DICOM”.

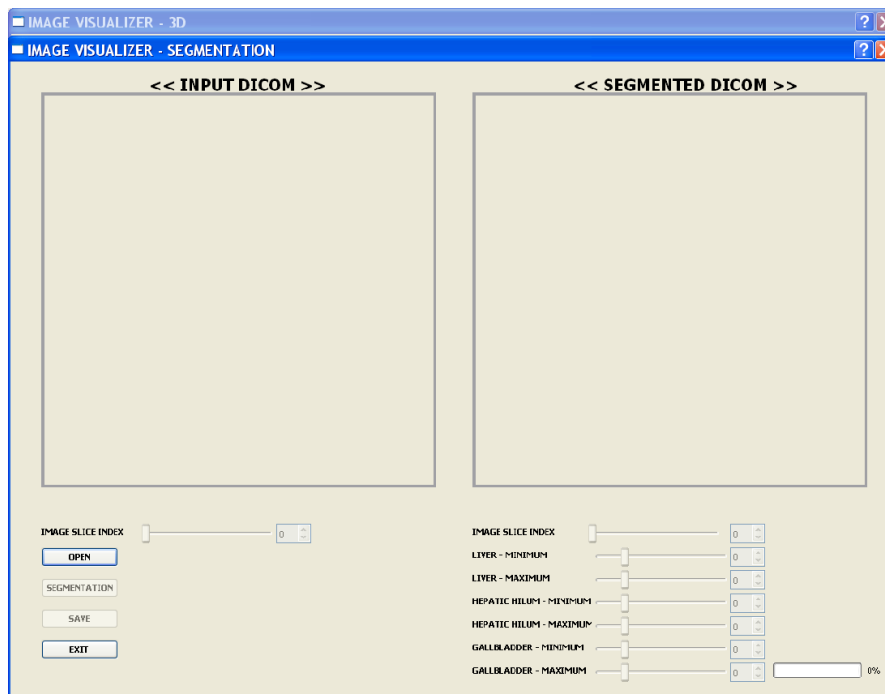


Figura 4.2: Visualización de la segunda ventana.

4.3. Evaluación de la herramienta

En esta sección se va a evaluar la herramienta paso a paso, de acuerdo con las opciones que proporciona:

1. Se ejecuta la aplicación y se aprieta el botón “OPEN MENU” y a continuación se aprieta el botón “OPEN NON-SEGMENTED IMAGES” para tener acceso a la segunda ventana (Figura 4.1).
2. Se aprieta el botón “OPEN” y se abre un directorio de imágenes DICOM no segmentadas pertenecientes al mismo paciente (Figura 4.3).

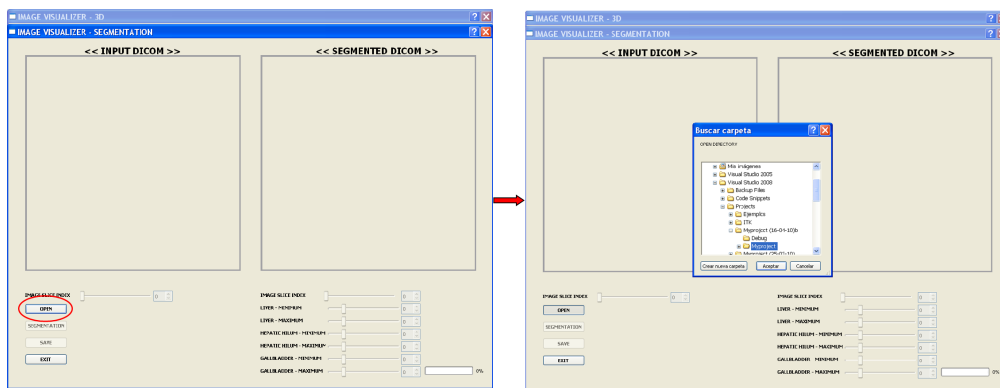


Figura 4.3: Aspecto de la IGU al apretar el botón “OPEN”.

3. Se desplaza la barra deslizador “IMAGE SLICE INDEX” por todas las imágenes DICOM no segmentadas pertenecientes al mismo directorio y que permite visualizar cada uno de los cortes transversales correspondientes a la CT o MR (Figura 4.4).

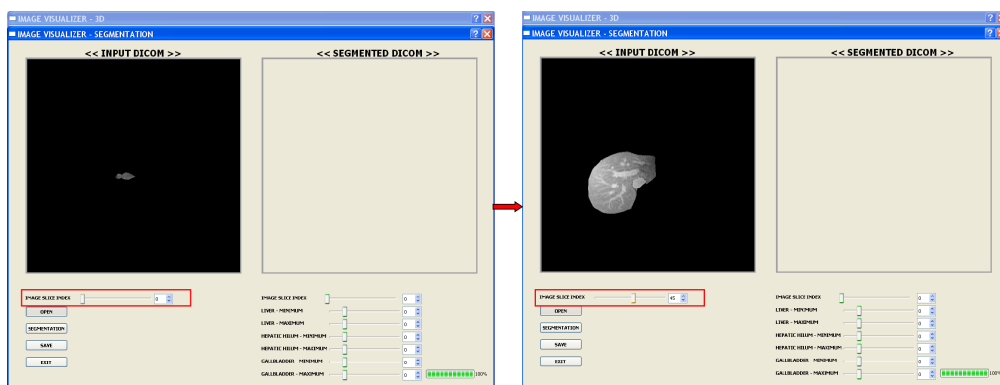


Figura 4.4: Aspecto de la IGU al desplazarse por la barra deslizador.

- Se aprieta el botón “SEGMENTATION” y selecciona un método de segmentación del *combobox* que aparece y se visualiza el resultado de la segmentación realizada. En el caso de elegir la umbralización binaria se elegirán también los valores máximos y mínimos de cada uno de los tejidos hepáticos (Figura 4.5). Mientras que si se elige el método de crecimiento de regiones será necesario hacer *click* en un punto del *frame* “INPUT DICOM” para proporcionarle al método un punto semilla.

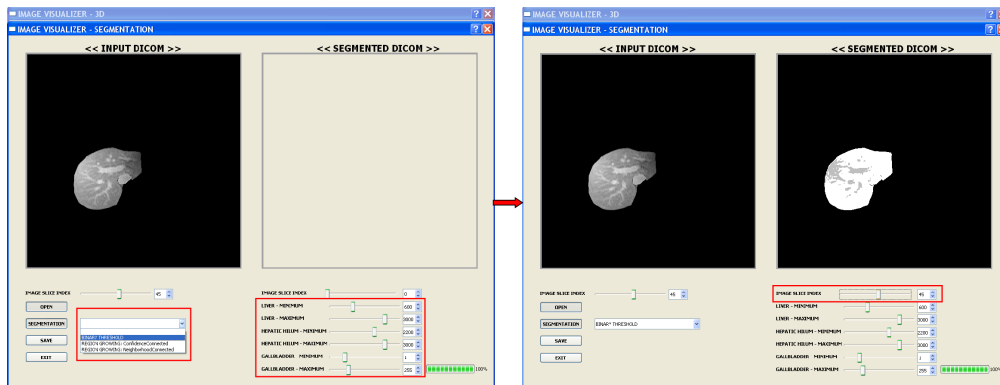


Figura 4.5: Aspecto de la IGU al apretar el botón “SEGMENTATION” y realizar un método de segmentación.

- Se guardan las imágenes DICOM segmentadas. Las imágenes se guardarán en un único archivo con extensión “.dcm”, que junto con el nombre del archivo deberá ser proporcionada por el usuario (Figura 4.6).

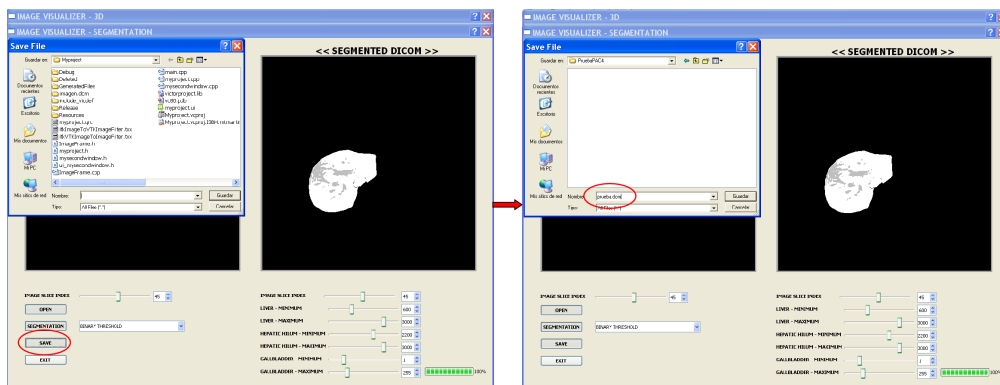


Figura 4.6: Aspecto de la IGU al guardar las imágenes.

- Se cierra la segunda ventana, para poder tener acceso a la primera.

- Se aprieta el botón “OPEN SEGMENTED IMAGES” y se abre el directorio de las imágenes DICOM segmentadas previamente. También podría ocurrir que el usuario tuviera ya un archivo con imágenes DICOM segmentadas por lo que no hubiera sido necesario acceder a la segunda ventana, sino que directamente se hubiera partido de este punto (Figura 4.7).

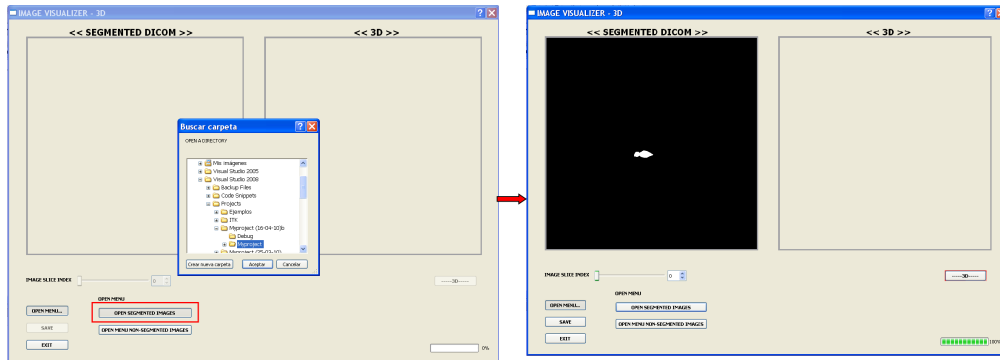


Figura 4.7: Aspecto de la IGU al abrir un directorio de imágenes DICOM segmentadas previamente.

- Se aprieta el botón “3D” y se visualiza el volumen 3D a partir de las imágenes DICOM segmentadas (Figura 4.8).

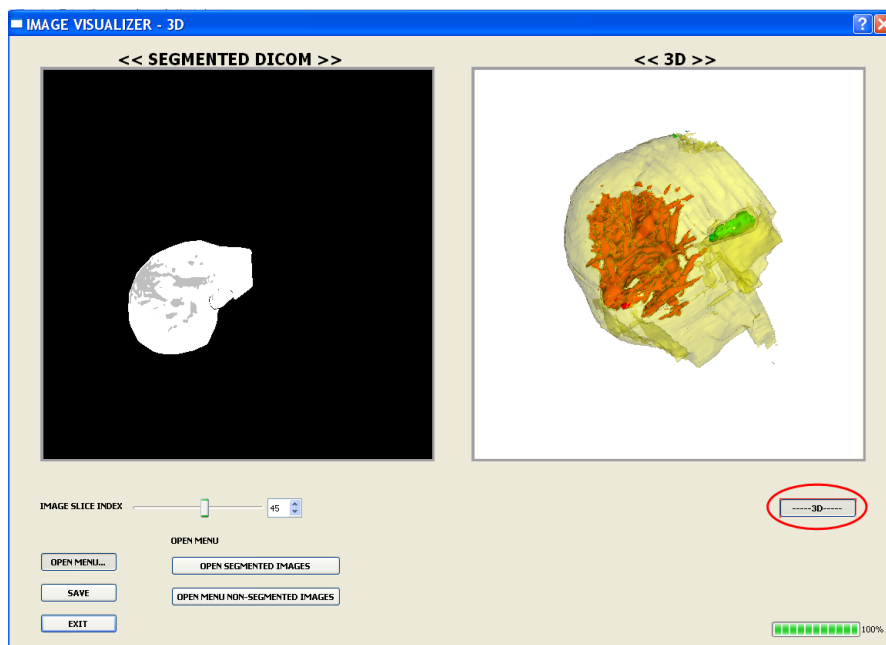


Figura 4.8: Aspecto de la IGU al visualizar las imágenes segmentadas y el volumen 3D de las mismas.

9. Rotación y zoom mediante el ratón del volumen 3D (Figura 4.9).

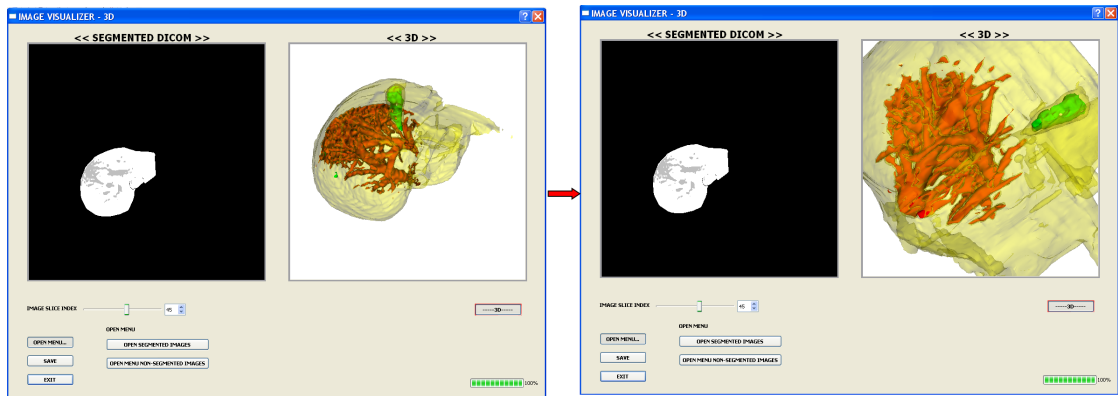


Figura 4.9: Aspecto de la IGU al rotar y hacer zoom en el volumen 3D.

Capítulo 5

Conclusiones y líneas futuras

En este proyecto se ha desarrollado una herramienta software para la segmentación y visualización 3D de hígados reales a partir de imágenes de escaneos 3D no invasivos. Esta herramienta se ha creado con el fin de ayudar a los médicos y cirujanos en el diagnóstico y la planificación de una cirugía hepática ya que, al proporcionar una imagen 3D del mismo, y permitir que los tejidos se hagan transparentes proporciona una visualización con mayor detalle las estructuras internas del hígado.

En el caso particular de la extirpación de una parte del hígado, los cirujanos suelen emplear la segmentación de Couinaud para cortar ya que estos segmentos hepáticos mantienen todos su flujo de entrada y de salida. Esta herramienta sería de gran utilidad para realizar esta segmentación tan complicada para los cirujanos ya que al hacer el hígado transparente y poder observar el hilio hepático dentro podrían hacer una planificación mucho más exhaustiva.

Los principales inconvenientes no han sido encontrados en el desarrollo de la herramienta ya sino que han dependido en gran medida de las imágenes utilizadas. La utilización de la herramienta Qt para la implementación de la interfaz gráfica ha permitido simplificar el diseño y el desarrollo. Es más, podemos afirmar que es una herramienta robusta ya que está preparada para añadir nuevos métodos de segmentación de imágenes y nuevas funcionalidades con facilidad. Las imágenes biomédicas poseen gran cantidad de ruido y una enorme variabilidad en sus propiedades. En especial las imágenes de resonancia magnética que han sido las utilizadas en este proyecto. Una solución sería homogeneizar la luminosidad durante el procesamiento de la imagen, por ejemplo, realizando

una corrección de sombras que nos permita homogeneizar la luminosidad e incrementar la nitidez de los bordes de la imagen. Además del problema de la luminosidad, pueden aparecer otros problemas como imágenes borrosas, pero al igual que la luminosidad, se pueden reducir tratando la imagen antes de segmentar.

En cuanto a los métodos de segmentación, su semi-automatización hace difícil que se obtengan resultados perfectos, ya que por ejemplo, en el caso de la segmentación por umbralización, es difícil encontrar los valores umbrales que den los resultados exactos, por ello, en líneas futuras se buscará implementar nuevos métodos de segmentación que proporcionen mejores resultados.

En trabajos futuros sería interesante:

- Incorporar nuevos métodos de segmentación para mejorar la precisión en los resultados obtenidos.
- Añadir un menú de ayuda que asista en los primeros pasos de la utilización de la herramienta.

Bibliografía

- [1] Neri E., Caramella D., Bartolozzi C. Image Processing in Radiology. Medical Radiology. Diagnostic Imaging. Baert A.L., Knauth M., Sartor K. Editors, Springer, 2008.
- [2] Rangaraj M. Rangayyan Biomedical Image Analysis, The Biomedical Engineering Series, CRC Press, 2005.
- [3] Fasel JH, Selle D, Evertsz CJ, Terrier F, Peitgen HO, Gailloud P. Segmental anatomy of the liver: poor correlation with CT. Radiology Jan;206(1):151-156, 1998.
- [4] Gao L, Heath DG, Kuszyk BS, Fishman EK. Automatic liver segmentation technique for three-dimensional visualization of CT data. Radiology 1996 Nov;201(2):359-364.
- [5] Jung G, Krahe T, Krug B, Hahn U, Raab M. Delineation of segmental liver anatomy. Comparison of ultrasonography, spiral CT and MR imaging for preoperative localization of focal liver lesions to specific hepatic segments. Acta Radiol 1996 Sep;37(5):691-695.
- [6] Soyer P, Heath D, Bluemke DA, Choti MA, Kuhlman JE, Reichle R, Fishman EK. Three-dimensional helical CT of intrahepatic venous structures: comparison of three rendering techniques. J Comput Assist Tomogr 1996 Jan;20(1):122-127.
- [7] Gazelle, G. et. al. Cholangiographic Segmental Anatomy of the Liver. Radiographics. 14(5):1005-1013. 1994 Sep. Abstract
- [8] Soyer, P. et. al. Surgical Segmental Anatomy of the Liver: Demonstration with Spiral CT During Arterial Portography and Multiplanar Reconstruction. AJR. 163:99-103, 1994.

- [9] Bae KT, Giger ML, Chen CT, Kahn CE Jr. Automatic segmentation of liver structure in CT images. *Med Phys* 1993 Jan;20(1):71-78.
- [10] Bismuth, H. *Surgical Anatomy and Anatomical Surgery of the Liver*. World J. Surg. 38-39, 1982.
- [11] Meinzer H.P., Schemmer P., Schöbinger M., Nolden M., Heimann T., Yalcin B., Richter G.M., Kraus T., Büchler M.W., Thorn M. Computer-Based surgery planning for living liver donation. In *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 34, 2004.
- [12] Nava A., Mazza E., Furrer M., Villiger P., Reinhart W.H. In vivo mechanical characterization of human liver. *Medical Image Analysis*, 12, 203-216, 2007.
- [13] Reitinger B. *Virtual Liver Surgery Planning: Simulation of Resections using Virtual Reality Techniques*. PhD Thesis Graz University of Technology, Institute for Computer Graphics and Vision, Australia, 2005.
- [14] Hongjian S., Farag A.A., Fahmi R., Chen D. Validation of Finite Element Models of Liver Tissue using Micro-CT. *IEEE Transactions on Biomedical Engineering*, 55 (3) 978-984, 2008.
- [15] P. Campadelli, E. Casiraghi, A. Esposito Liver segmentation from computed tomography scans: A survey and a new algorithm *Artificial Intelligence in Medicine* 45, 185-196, 2009.
- [16] Rodríguez Muro, M., *Sistema para el diagnóstico de pancreatitis utilizando RNA*. Tesis Licenciatura. Ingeniería en Sistemas Computacionales, 2003.
- [17] Ibáñez L, et al. *The ITK software guide*, 2003.
- [18] William J. Schroeder, et al. *The VTK user's guide*, 2001
- [19] J. Blanchette, M. Summerfield *C++ GUI Programming with Qt 4 (2nd Edition)* Prentice Hall Open Source Software Development Series, 2006.