ENTORNO GRÁFICO DE UN SIMULADOR QUIRÚRGICO 3D

Helber Mayorca Torres
Adriana Lucía Torres Silva

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Electrónica, Instrumentación y Control
Ingeniería en Automática Industrial
Popayán, Mayo de 2011

ENTORNO GRÁFICO DE UN SIMULADOR QUIRÚRGICO 3D

Monografía presentada como requisito parcial para optar por el título de Ingeniero en Automática Industrial

Helber Mayorca Torres
Adriana Lucía Torres Silva

Director: Ing. Elena Muñoz España

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Electrónica, Instrumentación y Control
Ingeniería en Automática Industrial
Popayán, Mayo de 2011

Nota de aceptación: _____

_____

_____

_____

_____

_____

_____

Firma del Presidente del Jurado

_____

Firma del Jurado

_____

Firma del Jurado

Popayán, Mayo de 2011

Agradecimientos

Principalmente queremos agradecerle a Dios por estar con nosotros en cada paso que damos, por fortalecer nuestro corazón e iluminar nuestra mente y por haber puesto en nuestro camino a aquellas personas que han sido nuestro soporte y compañía durante todo el periodo de estudio.

Agradecer hoy y siempre a nuestras familias porque a pesar de no estar presentes físicamente, sabemos que procuraron nuestro bienestar desde lejos, y está claro que si no fuese por el esfuerzo realizado por ellos, no hubiesen sido posible concluir nuestros estudios.

Agradecer a todos nuestros compañeros de estudio que nos ayudaron con sus oportunos consejos y su eterno apoyo.

Agradecemos a la ingeniera Elena Muñoz por su interés y acompañamiento.

A los evaluadores de este proyecto que con su visión incrementarán el aporte científico del mismo.

Y por último agradecemos a la Universidad del Cauca por formarnos como profesionales.

Resumen

El presente proyecto presenta el desarrollo de un entorno gráfico para un simulador quirúrgico 3D, constituido por imágenes biomédicas captadas por un aparato de diagnóstico TAC (Tomografía Axial Computarizada) a partir de las cuales se genera el cuerpo 3D del órgano deseado, en este caso se seleccionó el hígado por ser un órgano representativo del cuerpo humano que cumple diferentes funciones vitales. Mediante este prototipo computacional el usuario será capaz de interactuar con el objeto 3D y diferentes herramientas quirúrgicas creadas.

La metodología definida para la construcción del entorno comienza desde la segmentación de las imágenes médicas abdominales donde se aplican distintos algoritmos para la segmentación, continuando con la reconstrucción y visualización tridimensional de las estructuras de interés, terminando con la integración a un sistema de detección de colisiones entre el órgano y herramientas quirúrgicas virtuales, tales como tijeras, bisturí y lápiz.

Para llevar a cabo dicho desarrollo se recurrió al uso de librerías de *software* libre orientadas a la creación y visualización de objetos tridimensionales tales como VTK, además de ciertas bibliotecas especializadas para la segmentación de imágenes médicas ITK, QT para la creación de interfaces gráficas de usuario y *V-Collide* junto con *Rapid* para implementar el sistema de detección de colisiones que servirá para aumentar el realismo de la aplicación y así aproximarse a eventos muchos más reales que suceden en una sala de cirugía.

ABSTRACT

This work presents the development of a graphical environment for 3D surgical simulator, consisting of biomedical images captured by a diagnostic tool CT (Computed Tomography) from of which generates the desired 3D body organ, in this case selected the liver to be a representative body of the human body that fulfills several vital functions. With this prototype Computer users will be able to interact with the 3D object and different surgical tools created.

The methodology defined for the built environment begins segmentation of medical images which are applied abdominal algorithms for segmentation, continuing the Three-dimensional reconstruction and visualization of structures interest, ending with the integration of a detection system collisions between the body and virtual surgical tools, such scissors, scalpel and pencil.

To carry out this development resorted to using libraries free software aimed at the creation and display of objects dimensional such as VTK, in addition to certain libraries specialized for ITK medical image segmentation, QT for creating graphical user interfaces and V-Collide with Rapid to implement the collision detection system that will help increase the realism of the application and so many events approach more real that happen in an operating room.

Contenido

LISTA DE FIGURAS

LISTA DE TABLAS

1. INTRODUCCIÓN

La cirugía es una disciplina médica que data desde el antiguo Egipto, sobre el año 1.550 a.c. con métodos que continúan vigentes. Desde entonces hasta la actualidad, el conocimiento de las técnicas quirúrgicas se transmite mediante el entrenamiento de aprendices supervisados continuamente por un maestro [1]. El entrenamiento de los cirujanos en las técnicas quirúrgicas también se ha venido realizando mediante la utilización de cadáveres y animales vivos [2].

La utilización de los distintos métodos de entrenamiento quirúrgico presenta problemas debido a su elevado costo de mantenimiento, y ciertos problemas éticos por la utilización de animales vivos en experimentación. Por último, el aprendizaje de las diferentes técnicas a través de intervenciones quirúrgicas reales bajo la supervisión de un experto presenta problemas en cuanto al reducido número de veces que el cirujano puede practicar una determinada técnica [2].

Las ventajas de los simuladores quirúrgicos frente a los métodos tradicionales son potencialmente amplias ya que éstos permiten: reducir los costes asociados con la utilización de cadáveres y animales vivos en el entrenamiento en cirugía [2]; proveer de experiencia al médico con una mayor variedad de patologías y complicaciones; repetir los procedimientos quirúrgicos tantas veces como sea necesario hasta su correcto aprendizaje; permitir re-visualizar los procedimientos realizados con el objetivo de mejorarlos mediante la utilización de técnicas diferentes a las empleadas; practicar sobre la anatomía de un paciente específico previamente a su intervención quirúrgica a través de la utilización de imágenes médicas [3].

Considerando el alto grado de responsabilidad del cirujano sobre el paciente y la gravedad de las fallas de origen humano que pueden afectar el procedimiento, por este motivo y por el afán de nuevos conocimientos y nuevas alternativas para el adiestramiento quirúrgico de cirujanos en nuestro medio, se decide realizar este proyecto con el objetivo de construir un entorno gráfico para un simulador quirúrgico. Los distintos objetivos en este proyecto conforman la creación de un entorno donde se pueda manipular mediante herramientas quirúrgicas virtuales un objeto, en este caso el hígado, reconstruido en tres dimensiones a partir imágenes médicas. La metodología definida en el proyecto comienza con la segmentación de las imágenes médicas abdominales, continuando con la reconstrucción y visualización tridimensional de las estructuras de interés, y terminando con la integración a un sistema de detección de colisiones órgano – herramienta. Para llevar a cabo los objetivos del proyecto se recurrió al uso de librerías de *software* libre orientadas a la creación y visualización de objetos tridimensionales tales como VTK, además de ciertas librerías especializadas para la segmentación de imágenes médicas, ITK*;* QT

para la creación de interfaces gráficas, y *V-Collide* para implementar el sistema de colisiones.

En el capítulo 2, se abordan los conceptos sobre simuladores quirúrgicos, sus funciones y componentes, conceptos básicos sobre los métodos de obtención de imágenes médicas y de características del órgano a reconstruir, en este caso el hígado. En el capítulo 3 se describen cada herramienta *software* utilizada para creación del entorno. En el capítulo 4, se realiza una descripción sobre los métodos utilizados en la segmentación de las imágenes médicas y la visualización 3D de la anatomía del hígado. En el capítulo 5 se describe la estructura interna de la aplicación final, además se expone la funcionalidad del prototipo computacional, con cada uno de los componentes, pruebas y diferentes resultados. Para culminar con las conclusiones y anexos que contienen los manuales de instalación y manejo de la aplicación final.

2. CONCEPTOS PRELIMINARES

Este capítulo presenta los conceptos generales sobre simuladores quirúrgicos y algunos simuladores comerciales. Además se expone los distintos métodos de obtención de imágenes médicas las cuales son necesarias para la visualización del órgano en 3D.

2.1. SIMULADORES QUIRÚRGICOS

Un simulador quirúrgico es la herramienta *hardware* y *software* que muestra la escena de entrenamiento y permite a los cirujanos la interacción con dicha escena a través de interfaces hápticas[1] con realimentación de fuerza [3].

Los simuladores quirúrgicos constituyen entornos virtuales que representan una alternativa de formación, ya que permitirán la creación de ambientes de simulación interactivos en tres dimensiones, donde el cirujano tiene las mismas percepciones visuales y táctiles que durante la operación a un paciente real [1]. En la actualidad estos simuladores se están llevando a varias ramas de la cirugía como la Cardiocirugía, la Neurocirugía, Ortopedia y otras con resultados muy satisfactorios [4].

2.1.1. Simuladores comerciales

Varios países han fomentado proyectos de simulación quirúrgica, se resaltan, Simbionix ANGIE Mentor™, I-SIM y InsightArthoV entre otros, líderes globales en este tipo de tecnología.

El simulador **Simbionix ANGIO Mentor™** [4] que se observa Figura 1, es un producto multidisciplinario que ofrece ejercicios prácticos en un entorno de simulación completo para procedimientos endovasculares. Cardiólogos, radiólogos y cirujanos vasculares puede realizar procedimientos completos, así como la adquisición de competencias básicas.

El **I-SIM** [5] que se observa en la Figura 2, es un simulador avanzado. Ofrece una imagen de calidad, que lo hace ideal para el desarrollo de habilidades laparoscópicas básicas y avanzadas. Su diseño compacto hace de la portabilidad su mayor ventaja. El i-Sim puede ser conectado a una computadora para capturar y grabar clips de vídeo durante el entrenamiento, esto permite al aprendiz supervisar el progreso, y también crear una evaluación objetiva de habilidades quirúrgicas por el instructor.

[1] **Interfaces Háptica**: Aludimos a aquellos dispositivos que permiten al usuario tocar, sentir o manipular objetos simulados en entornos virtuales y sistemas tele operados.

Figura 1: Simbionix ANGIO Mentor™

Fuente: Simbionix ANGIO Mentor™ [4]

Figura 2: I-SIM

Fuente: ISURGICALS [5]

InsightArthroVR [6] es una herramienta multipropósito que se puede adaptar a diferentes técnicas artroscópicas. *InsightArthroVR* es un simulador de realidad virtual que integra un sistema de realimentación de fuerzas basado en el dispositivo háptico PHANTOM ® como se observa en la Figura 3, el cual proporciona las sensaciones y los sonidos que refuerzan el realismo de la simulación. Además *insightArthroVR* proporciona herramientas para evaluar las habilidades adquiridas con ejercicios prácticos.

Figura 3: InsightArthroV

Fuente: INSIGHTMIST [6]

2.1.2. Estructura de los simuladores quirúrgicos

Un sistema de simulación quirúrgico, estructuralmente, se puede descomponer en dos subsistemas. El subsistema de interfaz de cirujano y el subsistema de sensores.

El subsistema de interfaz del cirujano: tiene por objetivo el proveer al usuario de una visualización realista de la reconstrucción por computador de los órganos de interés para la simulación [5]. Este subsistema combina técnicas avanzadas de gráficos por computador y técnicas avanzadas de visualización, además posee sistemas de modelado avanzado de geometrías complejas que son útiles para la visualización realista de los órganos internos 3D. Estos sistemas permiten la reconstrucción 3D de los órganos de interés de un paciente estándar al que se le añaden patologías con cuya intervención quirúrgica se desea practicar [5] - [6]. Esta reconstrucción 3D se puede realizar mediante técnicas volumétricas o mediante técnicas de reconstrucción por superficies, el objetivo de estos sistemas es permitir mediante modelos deformables que los órganos se comporten de forma realista ante la aplicación de fuerzas y la realización de cortes o toques sobre su superficie [5] - [6].

En cuanto a las técnicas avanzadas de visualización, se utilizan técnicas de realidad virtual para la navegación 3D dentro del entorno quirúrgico virtual generado, y

técnicas de visualización 3D mediante sistemas estereoscópicos que provean al usuario de una visión 3D real del procedimiento quirúrgico [3].

El subsistema de sensores: está compuesto por un conjunto de dispositivos conocidos como "*Haptics*" que poseen realimentación de fuerzas. Estos dispositivos imitan las fuerzas experimentadas en la vida real durante la intervención quirúrgica de un paciente. Dichos dispositivos son extremadamente útiles en los Simuladores Quirúrgicos en los que el tacto provee de enormes sensaciones táctiles durante la intervención quirúrgica [3] - [6].

En la Figura 4, se observa el sistema completo con cada uno de los subsistemas. El subsistema de sensores(a) y el subsistema de visualización (b).

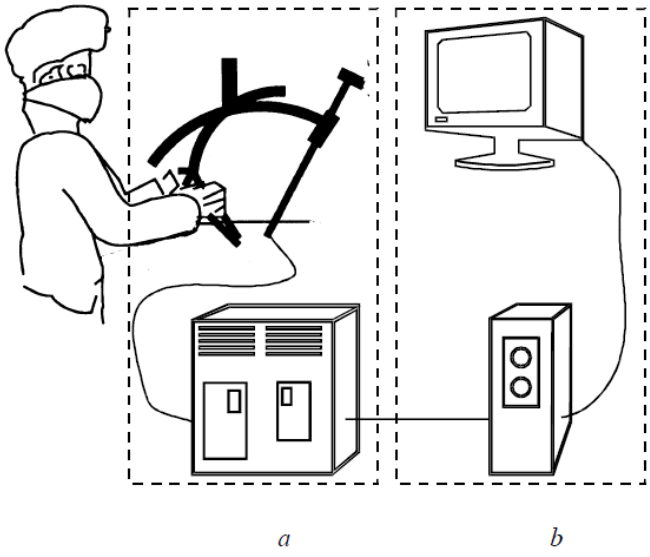


Figura 4: Sistema de Simulador Quirúrgico:(a) Subsistema sensores. (b) Subsistema de visualización

Fuente: Carlos Monserrat Aranda. Simulador para el entrenamiento en cirugías avanzadas [2]

2.2. REALIDAD VIRTUAL EN CIRUGÍA

La realidad virtual en cirugía abarca la aplicación de tecnologías computacionales interactivas útiles para planificar, simular y llevar a cabo procedimientos quirúrgicos. La realidad virtual se usa para presentar al cirujano visiones interactivas tridimensionales de áreas que se encuentran dentro del paciente [7].

Áreas en las que la realidad virtual está siendo aplicada entre otras:

1. Cirugía guiada por imágenes
2. Simuladores de entrenamiento
3. Planificación pre operativa

2.2.1. Cirugía guiada por imágenes

Al aplicar imágenes dentro de la cirugía éstas deben estar disponibles intra-operativamente. Actualmente, la realidad virtual se usa más para la planificación previa a las operaciones que para guiar a la cirugía actual. Cuando la realidad virtual se usa de forma intra-operativa, tiende a ser implementada como una manera de realidad aumentada[2]. La cirugía guiada por imágenes es también un prerrequisito para la telemedicina remota.

La realidad virtual resulta una fuente única de educación en lo referente a las estructuras anatómicas. Con la realidad virtual, el aprendiz puede explorar repetidamente estructuras de interés, separarlas, juntarlas y verlas desde casi cualquier perspectiva, sin embargo, uno de los mayores problemas en la cirugía guiada por imágenes es el de aportar una sensación de realidad en la interrelación de estructuras anatómicas en un espacio tridimensional [7].

2.2.2. Simuladores de entrenamiento

El desarrollo de simuladores de entrenamiento basados en realidad virtual permite al cirujano practicar procedimientos difíciles bajo el control del computador. Las ventajas de los simuladores de entrenamiento son obvias. El entrenamiento puede llevarse a cabo en cualquier momento y en cualquier lugar que disponga del adecuado equipamiento. Hacen posible la reducción de riesgos asociados al uso de nuevas técnicas, disminuyendo la mortandad quirúrgica [3] - [7].

Aun así, el auténtico retó consiste en simular con la suficiente confianza, de forma que las habilidades entrenadas con la simulación puedan ser llevadas íntegramente al quirófano. Los temas más importantes sobre los simuladores de entrenamiento incluyen el uso de realimentación de fuerza, aumento de la precisión en el modelado de tejidos blandos y el papel de la realimentación auditiva.

La Figura 5, muestra un entorno de simulación de cirugía endoscópica, donde es posible apreciar procedimientos realistas capaces de presentar formas anotómicas útiles para un sistema de educación médica.

[2]**Realidad Aumentada**: Visión directa o indirecta de un entorno físico del mundo real, combinando elementos reales y virtuales, en tiempo real [3].

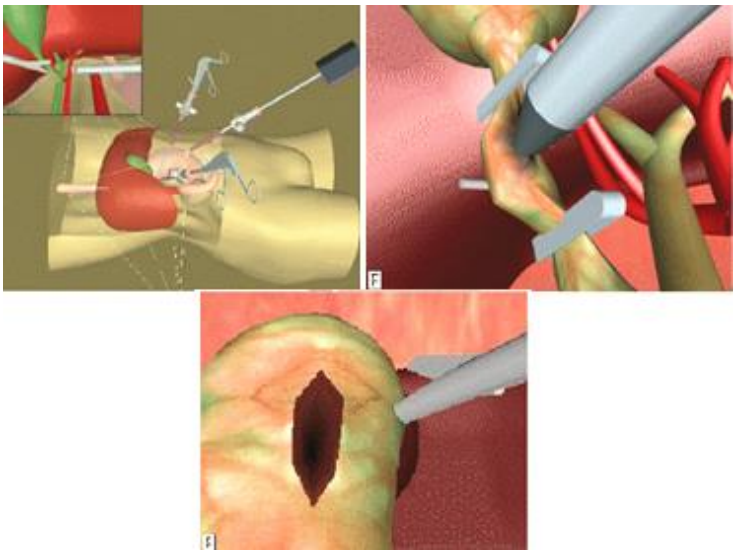


Figura 5: Entrenador de cirugía endoscópica

Fuente: Joel Orozco. Aplicaciones médicas de las técnicas inversivas de realidad virtual [7]

El área de los simuladores de entrenamiento que presenta más retos tecnológicos se refiere a aspectos altamente especializados de operaciones, como la cirugía cerebral [7]. El simulador de cirugía de *Johns Hopkins/KRDL*, basado en un cráneo, para entrenarse en recortes de aneurismas es un ejemplo presentado en la Figura 6.

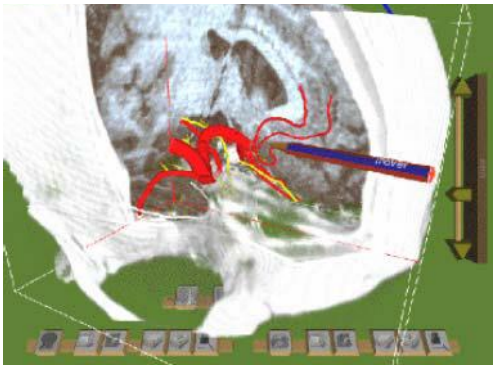


Figura 6: Simulador quirúrgico craneal (*Johns Hopkins/KRDL*)

Fuente: Joel Orozco. Aplicaciones médicas de las técnicas inversivas de realidad virtual [7]

2.2.3. Planificación previa a la operación

El simulador quirúrgico de tipo craneal [7] se aproxima a sistemas de planificación pre-operativo. Este tipo de sistemas se mezcla a veces con la realidad aumentada, puesto que la planificación se lleva a cabo en un paciente particular y actual, por lo que la realidad física y la realidad virtual pueden convivir de forma natural en la planificación. El objetivo de estos sistemas es el estudio de los datos del paciente antes de realizar la operación, y así planificar la mejor forma de llevar a cabo la misma.

En la planificación pre-operativa el principal objetivo es explorar los datos del paciente de la forma más completa posible, y evaluar posibles procedimientos de intervención con los datos, no reproducir la operación actual. Dichas simulaciones ayudan en el entrenamiento, y en las comunicaciones entre médicos y pacientes.

En la Figura 7, se presenta un sistema de planificación "puro" El objetivo es permitir a los cirujanos el examen de los datos del paciente de la forma más completa posible, y evaluar posibles vías de intervención. El sistema ofrece las coordenadas usadas de forma estándar para guiar el camino en la cirugía cerebral.

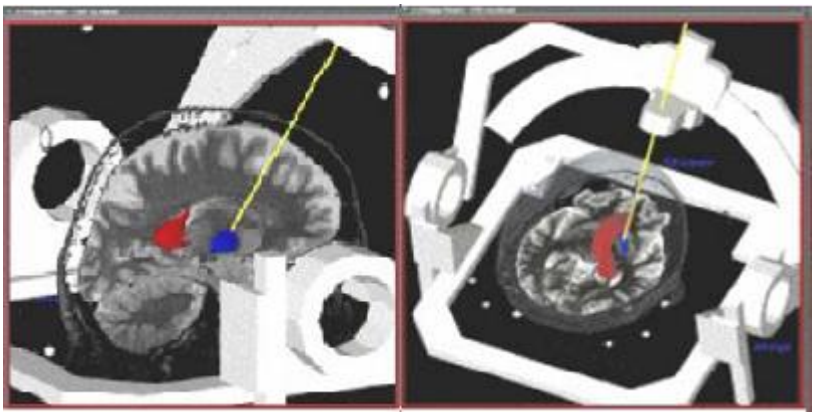


Figura 7: *Radionic's Stereoplan*

Fuente: Joel Orozco. Aplicaciones médicas de las técnicas inversivas de realidad virtual [7]

2.3. IMÁGENES MÉDICAS

En la actualidad existen diversas técnicas para el diagnóstico médico, que permiten observar en una pantalla alguna parte específica del cuerpo humano con alta resolución. Los sistemas actuales de diagnóstico computarizado pertenecen a la rama conocida como la digitalización de imágenes [8].

2.3.1. Digitalización de imágenes

La digitalización de imágenes tiene que ver con la generación o despliegue de imágenes mediante el uso de datos digitales. Para tal propósito la matriz se subdivide en una matriz de puntos o en un arreglo bidimensional de pequeños cuadritos que parecen puntos en la pantalla, donde cada uno de estos puntos tiene un valor que está codificado digitalmente según la información que proporcione. Estos pequeños puntos se conocen como píxeles [8]-[10].

La digitalización de las imágenes conlleva cuatro funciones separadamente.

1. Adquisición de la imagen digital
2. Procesamiento y despliegue de la imagen digital
3. Comunicación de la imagen digital
4. Almacenamiento y acceso a la imagen digital

Por ahora se describe la primera de las cuatro funciones y en el capítulo 4 se tratará sobre el procesamiento de las imágenes y almacenamiento de las mismas.

El sistema de adquisición de la imagen digital está compuesto de una fuente de energía, un sistema de detección, que convierte la energía en datos y un método digital para la construcción de la imagen [8]. En muchos casos la fuente de energía son los rayos X, el sistema de detección puede estar compuesto de pequeños cristales y el método digital es un programa o serie de instrucciones de computador que generan la imagen deseada.

Existen en la actualidad diversas técnicas que utilizan el concepto de digitalización de imágenes en el campo de la medicina [8], lo cual demuestra el significado que tiene la reconstrucción de imágenes. Algunas técnicas son:

1. Angiografía de Substracción Digital
2. Resonancia Magnética Nuclear
3. Tomografía Computarizada Axial

A continuación se explica brevemente cada una de estas técnicas y se profundizará en la tercera, puesto que fue el tipo de imágenes médicas utilizadas en este proyecto.

- **Angiografía de substracción digital**: es un sistema de procesamiento digital de imágenes que se realiza en tiempo real. Es utilizado para realizar exploración vascular y estudio de las funciones cardíacas. La fuente de energía para estos sistemas de digitalización de imágenes son los rayos X.

Este método radioscópico computarizado [11] proporciona buena visualización de los vasos arteriales. Se inyecta un medio de contraste a través de un catéter y se toman placas radiográficas. Un sistema de vídeo con intensificación de imagen muestra los vasos en un monitor de televisión, mientras que un computador elimina las imágenes no necesarias, de forma que se obtiene una imagen final intensificada del área estudiada.

- **Resonancia magnética nuclear:** es un sistema basado en una fuente que no es de rayos X, sino magnética, es decir que toda la energía parte de imanes colocados dentro del equipo [8].Este sistema de digitalización de imágenes no produce radiación por no utilizar rayos X como fuente. La energía es electromagnética la cual se utiliza para la construcción de la imagen estudiada, la resonancia magnética tiene la propiedad de poder estudiar casi cualquier parte del cuerpo humano: cráneo, abdomen, tórax, hígado, riñones etc. [8].

La resonancia magnética nuclear [8] permite producir imágenes de cualquier orientación es decir que sin necesidad de trasladar al paciente, resulta posible explorar no solo estratos axiales sino también vistas laterales. Además permite representar los tejidos de las partes blandas con un contraste muy elevado y por consiguiente mostrar alteraciones fisiológicas.

- **Tomografía Computarizada:** es una modalidad de digitalización de imágenes que usa tecnología digital tanto para la reconstrucción de imágenes como para el despliegue. La Tomografía Computarizada es un proceso en el cual mediante múltiples proyecciones de rayos X, a través de una pequeña sección transversal del cuerpo se logra reconstruir una imagen aproximada que se presenta en forma clara en una pantalla [9].

La tomografía utiliza una fuente de rayos X, además recorre el cuerpo desde varios ángulos y utiliza detectores de cristal como el Xenón, para absorber los rayos. Esta técnica, reconstruye imágenes de una sección transversal del cuerpo humano utilizando programas por computador para la reconstrucción de las imágenes. Esta modalidad se dio a conocer en 1969 por el físico inglés *GodFrey N Hounsfield*. En septiembre de 1971 la primer Tomografía Computarizada fue instalada en el hospital *Atkinson Morley*en Wimbledon Inglaterra. Este primer *Scanner* se restringía únicamente al estudio del cerebro, y no de otras partes del cuerpo.

La tomografía computarizada es considerada hoy en día, parte esencial y de gran aceptación en el área de diagnóstico médico, se cuenta actualmente con *Scanner* para el cuerpo entero que genera una imagen instantánea, esto gracias al avance tecnológico en computación y al diseño de equipos mucho más avanzados [9].

En la Figura 8, se observa la representación de un cuerpo ante una radiación de rayos X. El procedimiento se repite sucesivamente al rotar el tubo de rayos X y los detectores. Entre menor sea el ángulo mejor será la calidad de la imagen resultante [9].

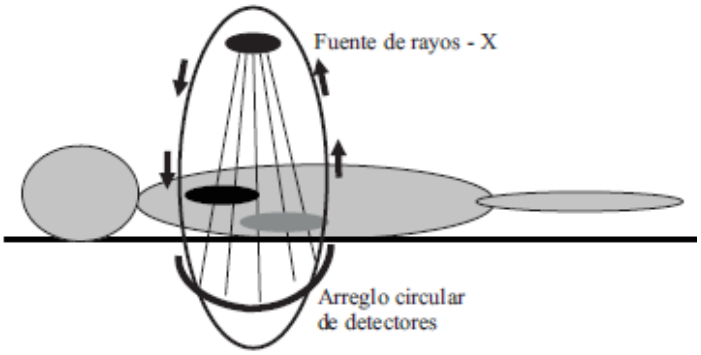


Figura 8: Obtención de Imágenes de Tomografía Computarizada

Fuente: Modulo de Navegación para un Sistema de Entrenamiento Virtual Aplicado a Cirugía de la Base del Cráneo. [9]

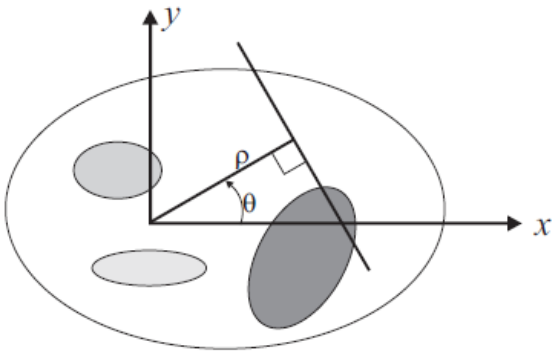


Figura 9: Trayectoria del rayo atravesando diferentes órganos

Fuente: Modulo de Navegación para un Sistema de Entrenamiento Virtual Aplicado a Cirugía de la Base del Cráneo [9]

El concepto matemático fundamental de la Tomografía Computariza es la transformada inversa de Radón, la cual se aplica a las señales obtenidas en los detectores como se indica en la Figura 9. Se asume que los rayos X viajan en líneas rectas y el haz presenta una atenuación μ(x;y) dependiente de la posición a lo largo de esa línea [9].

2.4. HÍGADO

El hígado es el segundo órgano más grande del cuerpo humano después de la piel, y es el mayor órgano interno. El hígado se ve por primera vez en el embrión en desarrollo, durante la cuarta semana de embarazo. El suministro de sangre del hígado es exclusivo, proviene tanto del corazón como del tracto digestivo en forma directa a través de un gran vaso sanguíneo llamado la vena porta [13].

El hígado se encarga de cerca de 500 funciones orgánicas desempeña un papel en la digestión, el metabolismo del azúcar y las grasas, e incluso en el sistema inmunitario. Procesa prácticamente todo lo que comemos, respiramos o absorbemos a través de la piel. Alrededor del 90% de los nutrientes del organismo procedentes de los intestinos pasan por el hígado. El hígado convierte los alimentos en energía, almacena nutrientes y produce proteínas sanguíneas. Además, actúa como filtro para eliminar patógenos y toxinas de la sangre [16].

2.4.1. Funcionamiento del hígado

Cuando se ingieren los alimentos, los nutrientes viajan por la garganta y llegan al estómago para seguir luego a los intestinos. El alimento se descompone en pequeños trocitos que son absorbidos por el torrente sanguíneo. La mayoría de estas pequeñas partículas viajan desde los intestinos hasta el hígado, el cual filtra y convierte el alimento en nutrientes que el torrente sanguíneo lleva a las células que lo necesitan. El hígado almacena estos nutrientes y los libera durante el día, a medida que el organismo va necesitándolos [13].

Las células hepáticas producen la bilis, un líquido amarillo verdoso que facilita la digestión y absorción de nutrientes liposolubles. La bilis llega al intestino delgado a través de las vías biliares; cuando no hay alimentos que digerir, la bilis sobrante se almacena en un pequeño órgano, denominada vesícula biliar, situado por debajo del hígado. Los derivados resultantes de la descomposición de los fármacos y las sustancias tóxicas procesadas por el hígado se transportan en la bilis y se excretan fuera del cuerpo. Las personas con daños hepáticos pueden experimentar alteraciones en la producción y el flujo de bilis [16].

Las células hepáticas también convierten el hemo (un componente de la hemoglobina que se libera cuando se descomponen los glóbulos rojos) en bilirrubina. Cuando el hígado está dañado, puede acumularse bilirrubina en la sangre, provocando ictericia (que se manifiesta con un color amarillento en la piel y el blanco de los ojos) [16].

2.4.2. División morfológica del hígado

Anatómicamente el hígado, puede ser considerado desde el punto de vista morfológico o funcional. La anatomía morfológica, considera la división clásica del hígado. En la Figura 10 se aprecia el lóbulo derecho y lóbulo izquierdo, separados por el ligamento falciforme.

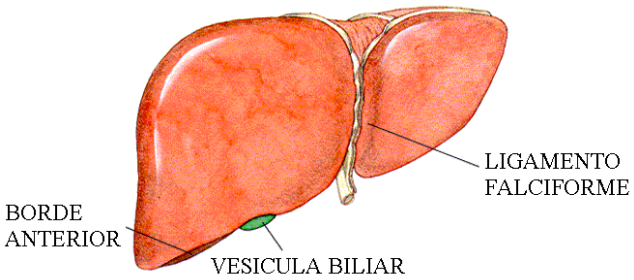


Figura 10: División Clásica del hígado

Fuente: Facultad de Medicina – UNAM [17]

El hígado presenta dos caras. La cara antero superior y la cara posteroinferior. Un borde anterior y un borde posterior, con dos extremidades una derecha y una izquierda.

En su cara posteroinferior presenta:

- El lóbulo cuadrado por delante.
- El lóbulo caudado o lóbulo de *Spiegel*, por detrás.

La cara anterosuperior del hígado presenta la *Línea de Cantlie*[3], que se extiende desde el borde derecho de la vesícula biliar, hasta el borde izquierdo de la vena cava inferior. Cada uno de los dos lóbulos principales contiene unidades más pequeñas llamadas lobulillos. La mayoría de los hígados tienen de 50.000 a 100.000 lobulillos que constan de una vena rodeada por minúsculas células hepáticas llamadas hepatocitos. Estas células purifican la sangre, eliminan los desechos y toxinas y almacenan nutrientes saludables para que el cuerpo los utilice cuando sea necesario [13].

[3]**Línea de cantlie:**Línea imaginaria que permite la división funcional del hígado en dos lóbulos

3. DESCRIPCIÓN DE HERRAMIENTAS SOFTWARE

La constante necesidad de reconstruir lo desconocido y la capacidad de crear escenarios donde existan objetos e instrumentos que realicen ciertas funciones, ha sido la base para implementar técnicas avanzadas de programación computacional, además teniendo en cuenta la importancia de la medicina dentro de la sociedad y su repercusión en la misma, se ve la necesidad de acoplar los avances tecnológicos, para así crear herramientas capaces de mejorar las habilidades humanas.

El problema planteado para este trabajo de grado comienza con la idea de desarrollar una aplicación software que se aproxime a un escenario quirúrgico en tres dimensiones, que utilice además herramientas software de código abierto que permita tocar órganos humanos reconstruidos a partir de imágenes médicas con herramientas quirúrgicas virtuales, por lo tanto en este y los capítulos posteriores se presentará el desarrollo de la misma, comenzando con la descripción de las herramientas software utilizadas para el desarrollo de esta aplicación.

3.1. SELECCIÓN DE HERRAMIENTAS SOFTWARE

En general el prototipo diseñado agrupa varias herramientas de alto desempeño, las cuales se escogieron principalmente por ser de código abierto cumpliendo de esta manera los objetivos del proyecto, y por sus variadas funcionalidades. Las bibliotecas seleccionadas son: ITK, para el procesamiento de imágenes, VTK biblioteca de visualización 3D, QT entorno de trabajo para el diseño de interfaces de usuario, *CMake* herramienta multiplataforma generadora de código, *V-Collide* y *RAPID* bibliotecas de detección de colisiones.

A continuación se explica brevemente la funcionalidad y estructura general de las bibliotecas anteriormente mencionadas y ejemplos para llevar a cabo una actividad determinada propia de cada una de éstas. Se recomienda para realizar las actividades revisar el anexo A de instalación de las bibliotecas.

3.1.1. Introducción a CMAKE

Cmake (*Cross Platform Makefile Generator*). *Cmake* es un sistema *open source* generador de proyectos que funciona en diferentes plataformas. Su función es la de configurar y gestionar el proceso de construcción de un proyecto en un archivo de texto simple, (*CmakeLists.txt*) donde se describe el proceso mediante código. Posee gran variedad de comandos que permiten básicamente vincular bibliotecas, crear ejecutables, cargar archivos al proyecto y la generación de instaladores.

- **Actividades ejecutadas en *CMake***

CMake es un programa que básicamente crea los proyectos basado en diferentes IDE como lo son *Visual Studio* y *Code Blocks* en diferentes plataformas, teniendo el código de la aplicación y un archivo donde se configura el proyecto a crear. En esta pequeña actividad se mostrará un ejemplo explicando el uso de esta herramienta.

Para la instalación del programa remítase al anexo A Instalación de Bibliotecas. Una vez instalado, se procede a la ejecución de "cmake-gui" que al abrir deberá mostrar una pantalla (ver Figura 11) como la que se muestra a continuación.

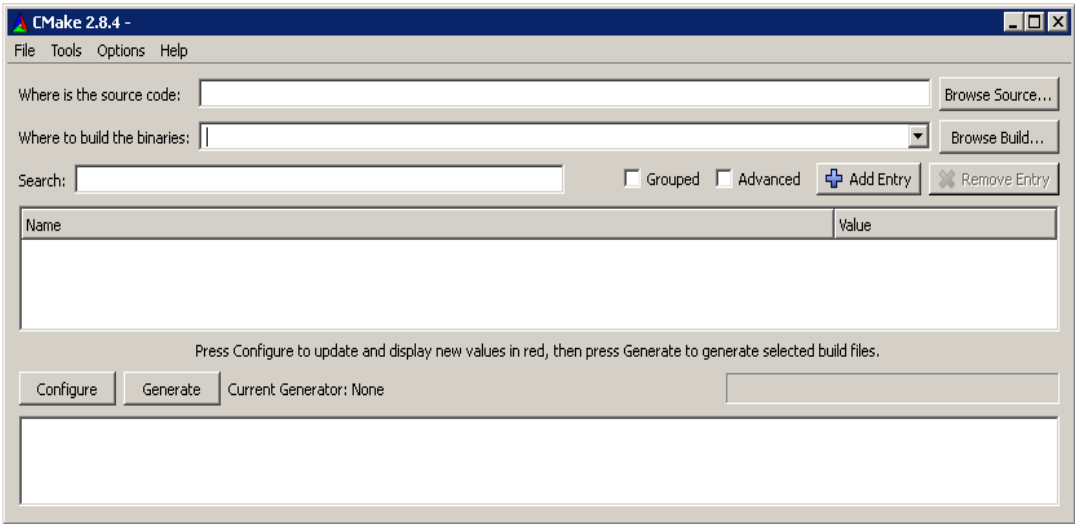


Figura 11: CMake

Cree una carpeta en cualquier lugar del disco en el computador y póngale el nombre "Código", allí serán almacenados los archivos que contienen el código de la aplicación. En esta carpeta cree un archivo de nombre "Circulo.h", donde se crearán las definiciones de la clase, ábralo en algún editor de texto y escriba lo siguiente:

```
#ifndef CIRCULO
#define CIRCULO

class circulo
{
private:
float radio;
public:
        circulo();
float area();
};
#endif
```

Cree otro archivo de nombre "Circulo.cpp", donde se implementarán los métodos de la clase anteriormente mencionada, ábralo con algún editor de texto y escriba lo siguiente:

```
#include"circulo.h"

circulo::circulo():radio(10)
{}
float circulo::area()
{
return (radio*radio*3.141516);
}
```

Cree otro archivo de nombre “main.cpp”, donde se implementará un cliente de la clase círculo, ábralo con algún editor y escriba lo siguiente:

```
#include<iostream>
usingnamespace std;

int main(void)
{
    circulo circ;
    cout<<"el área del circulo es:"<<endl;
    cout<< circ.area()<<endl;
    system("Pause");
return 0;
}
```

Por último, se crea el archivo que va organizar el proyecto y debe tener por nombre “CMakeLists.txt”, ábralo con algún editor y escriba lo siguiente:

```
PROJECT(PRUEBA1)

cmake_minimum_required(VERSION 2.6)

SET(Header Circulo.h)
SET(Source Circulo.cpp main.cpp)

INCLUDE_DIRECTORIES(
  ${CMAKE_CURRENT_BINARY_DIR}
  ${CMAKE_CURRENT_SOURCE_DIR})


ADD_EXECUTABLE(PRUEBA1 ${Header} ${Source})
```

A continuación se explicarán algunos comandos que se deben tener en cuenta para la generación de un proyecto:

${NOMBRE_VARIABLE}: Sirve para referirse al valor de una variable.

PROJECT(<PROJECTNAME>): Este comando sirve para asignarle un nombre al proyecto a crear, que lo recibe como un parámetro. El lenguaje debe ser especificado en seguida del nombre pero por defecto están habilitados C y C++.

SET(<VARIABLE><VALUE>): Este comando sirve para asignar uno o varios valores a una variable de CMake.

INCLUDE_DIRECTORIES(DIR1 DIR2 … DIRN): Incluye en el proyecto la ruta de las carpetas especificadas por las variables DIR1, DIR2,…,DIRN.

ADD_EXECUTABLE(<NAME> SOURCE1 SOURCE2 ... SOURCEN): Adiciona un ejecutable al proyecto con el nombre y archivos de código dadas en las variables SOURCE1, SOURCE2,...,SOURCEN.

TARGET_LINK_LIBRARIES(<TARGET> ITEM1 ITEM2 ... ITEMN): Adiciona los vínculos con las librerías especificadas por ITEM1,ITEM2,...,ITEMN.

Una vez realizados los anteriores pasos se procede a llenar los campos en la interfaz del CMake, donde aparecen los siguientes:

Where is the source code: en este campo se escribe la ruta de la carpeta donde se encuentra el código, pero si se quiere navegar por las carpetas del equipo se puede dar clic en el botón *Browse Build* y seleccionarla, recuerde que por recomendación se sugirió el nombre de “Código” pero este no es obligatorio.

Where to build the binaries: en este campo se escribe la ruta de la carpeta donde se desea generar el proyecto, pero si se quiere navegar por las carpetas del equipo se puede dar clic en el botón *Browse Build* y seleccionarla, de no existir la carpeta se debe crear sugiriendo que la ruta esté al mismo nivel de la carpeta que contiene el código y que el nombre sea “Generado”.

Una vez llenados los campos se procede a presionar el botón *Configure* en la parte inferior izquierda de la pantalla, en donde aparecerá un cuadro de diálogo preguntando el entorno de desarrollo para el proyecto. Una vez configurado el proyecto y de no tener errores se habilitará el botón *Generate* contiguo al anterior, después de unos pocos segundos se habrá generado el proyecto en la carpeta seleccionada anteriormente, Para el caso particular de *Visual Studio 2005* se generará un archivo de extensión *sln*.

Para mayor información remítase a la ayuda en línea del programa que está en la siguiente dirección:
http://www.cmake.org/cmake/help/documentation.html

3.1.2. Introducción a ITK

La biblioteca ITK (*Insight Toolkit*) ha sido desarrollada por seis principales organizaciones, pudiéndose encontrar entre ellas tanto organizaciones comerciales como académicas. La financiación del proyecto fue por parte de la *National Library of Medicine EE.UU.* Se compone de una serie de métodos de *open source* para el

registro y la segmentación de imágenes. De antemano se entiende que la segmentación es el proceso por el cual se identifican y se clasifican los datos encontrados en una representación digital muestreada, típicamente el resultado de un TAC (tomografía axial computarizada) o una resonancia magnética [26].

La biblioteca ITK se encuentra implementada en C++ y puede ser empleada en cualquier plataforma, y en múltiples lenguajes de programación para lo que requieren de CMake para configurar sus opciones de compilación. La implementación en C++ de ITK se ha realizado lo más genéricamente posible usando clases plantillas. Este uso de clases plantilla permite que el código sea muy eficiente y que la mayor parte de los errores sean descubiertos en tiempo de compilación en vez de en tiempo de ejecución.

Como ITK es un proyecto de código abierto, los desarrolladores de todo el mundo pueden usarlo, modificarlo y extender sus aplicaciones [26].

Para mayor información sobre ITK diríjase al documento"*The ITK software guide second edition update for ITK*" [26].

- **Actividad básica implementada en ITK:**

ITK es una biblioteca multiplataforma para el procesamiento de imágenes médicas. En esta actividad básica se mostrará un ejemplo del uso de ésta usando como lenguaje de programación C++, CMake para la vinculación del proyecto y como entorno de desarrollo Visual Studio 2005.

Este ejemplo comienza con la segmentación de regiones de igual nivel de gris dentro de la imagen, siguiendo con un filtrado y finalizando con la extracción del contorno de la misma. Para el uso de esta biblioteca es necesario tener conocimientos de procesamiento de imágenes, que no se explicarán en este ejemplo.

Las estrategias de solución que se implementan en ITK se denominan *pipeline* por ser un conducto de procesos uno enseguida del otro que se aplican a una imagen o datos, terminando con la salida deseada por el usuario. La estrategia para este proceso es la siguiente: (ver Figura 12: Pipeline ITK)

Figura 12: Pipeline ITK

Ahora se procede con la implementación en código de dicha estrategia con lo cual se crea un archivo con el nombre main.cpp y contiene lo siguiente:

```
#include"itkImageFileReader.h"
#include"itkImageFileWriter.h"
#include"itkBinaryContourImageFilter.h"
#include"itkBinaryThresholdImageFilter.h"
#include"itkBinaryBallStructuringElement.h"
#include"itkBinaryMorphologicalOpeningImageFilter.h"
#include"itkBinaryMorphologicalClosingImageFilter.h"

typedef itk::Image<unsignedchar,2> ImagenChar;
typedef itk::ImageFileReader<ImagenChar> CargarImagen;
typedef itk::ImageFileWriter<ImagenChar> GuardarImagen;
typedef itk::BinaryContourImageFilter<ImagenChar,ImagenChar>
ContornoImagen;
typedef itk::BinaryThresholdImageFilter<ImagenChar,ImagenChar>
                                    BinarizadoImagen;
typedef itk::BinaryBallStructuringElement<unsignedchar>
                                    ElementoEstructurante;
typedef itk::BinaryMorphologicalClosingImageFilter <ImagenChar,
                ImagenChar,ElementoEstructurante> CierreImagen;
typedef itk::BinaryMorphologicalOpeningImageFilter<ImagenChar,
                ImagenChar,ElementoEstructurante> AperturaImagen;

int main(int argc, char** argv)
{
     CargarImagen::Pointer imagen1 = CargarImagen::New();
constchar * nombreArchivo = "imagen1E.jpg";
     imagen1->SetFileName(nombreArchivo);
     imagen1->Update();

      BinarizadoImagen::Pointer binarizado1 =
                         BinarizadoImagen::New();
     binarizado1->SetInput(imagen1->GetOutput());
     binarizado1->SetOutsideValue(0);
     binarizado1->SetInsideValue(255);
     binarizado1->SetLowerThreshold(162);
     binarizado1->SetUpperThreshold(218);
     binarizado1->Update();

     ElementoEstructurante bola;
     ElementoEstructurante::SizeType tamañoBola;
     tamañoBola.Fill(10);
     bola.SetRadius(tamañoBola);
     bola.CreateStructuringElement();

     AperturaImagen::Pointer apertura1 = AperturaImagen::New();
     apertura1->SetInput(binarizado1->GetOutput());
     apertura1->SetKernel(bola);
     apertura1->Update();

     CierreImagen::Pointer cierre1 = CierreImagen::New();
     cierre1->SetInput(apertura1->GetOutput());
     cierre1->SetKernel(bola);
     cierre1->Update();

     ContornoImagen::Pointer contorno1 = ContornoImagen::New();
```

```
    contorno1->SetInput(cierre1->GetOutput());
    contorno1->Update();

    GuardarImagen::Pointer guardar1 = GuardarImagen::New();
    guardar1->SetInput(contorno1->GetOutput());
    nombreArchivo = "imagen1S.bmp";
    guardar1->SetFileName(nombreArchivo);
    guardar1->Update();

    return 0;
}
```

En forma general todo *pipeline* debe hacer lo siguiente:

Los tipos de datos debe ser ingresados por medio de una plantilla al igual que la definición de las clases, las diferentes instancias deben ser creadas siguiendo el paradigma de orientación a objetos el cual establece primero la llamada al constructor, se ingresan una serie de parámetros requeridos por cada *filter* o procedimiento sobre la imagen y el método *Update()* se encarga de realizarlo. La entrada y salida es asignada y llamada respectivamente por el comando *SetInput* y *GetOutput* la cual realiza la secuencia que dará como resultado la imagen deseada.

Por último, cree el archivo de configuración del proyecto CMakeLists.txt y escriba lo siguiente:

```
PROJECT(PruebaITK)
cmake_minimum_required(VERSION 2.6)

FIND_PACKAGE(ITK REQUIRED)
INCLUDE(${ITK_USE_FILE})

ADD_EXECUTABLE(Ejemplo main.cpp)
TARGET_LINK_LIBRARIES(Ejemplo ${ITK_LIBRARIES})
```

Genere el proyecto a partir de las rutas de las carpetas del código y adónde se va a generar, todo esto a través de CMake.

Una vez generado el proyecto se abre el archivo *PruebaITK.sln (r*ecuerde que para poder generar el proyecto de *Visual Studio 2005* debe tenerlo instalado). Compilamos el proyecto, una vez compilado podemos acceder al ejecutable en las carpetas *Debug* o *Release* según sea el modo de compilación.

En la carpeta *Debug* o *Release* debe existir una imagen de formato jpg con el nombre imagen1E.jpg (ver Figura 13)

Figura 13: Imagen a segmentar

La salida generada por la estrategia seguida se muestra a continuación (ver Figura 14)

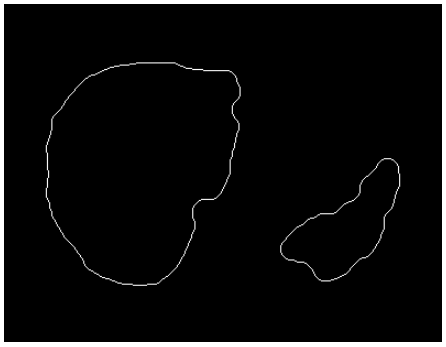


Figura 14: Imagen segmentada

Para más información puede visitar la página web http://www.itk.org/Doxygen320/html/classes.html donde podrá profundizar sobre los métodos y las clases utilizadas.

3.1.3. Introducción a VTK.

VTK (*Visualzation Toolkit*) es un conjunto de bibliotecas desarrolladas por Kitware [22] contienen un sistema que permite la visualización de geometrías 3D, soportando una amplia variedad de algoritmos de visualización y modelado. Es una herramienta *open source* que extiende su aplicación a prácticamente todos los campos en los que se emplean objetos 3D entre los que se destacan la medicina y las herramientas industriales.

Las funciones que integran VTK se basan en los principios de orientación a objetos conformando un completo sistema jerárquico de filtros y fuentes que dotan de gran flexibilidad a todo el sistema [22]. VTK no consiste únicamente en un sistema de visualización sino que aporta una amplia gama de algoritmos para tratar las imágenes y objetos creados, permitiendo entre otras cosas el añadir texturas, aplicar tensores, reducir el número de polígonos, además de proporcionar una gran

variedad de clases de datos para trabajar datos poligonales, imágenes, volúmenes y mallas.

VTK posee un modelo basado en el paradigma del flujo de datos adoptado por la mayoría de los sistemas comerciales. Según este paradigma, los diferentes módulos son conectados formando una red. La ejecución de la red de visualización es controlada en respuesta a las demandas de datos (conducción por demanda) o en respuesta a una entrada del usuario (conducción por eventos) [22]. La gran ventaja de este sistema es la flexibilidad, y puede ser rápidamente adaptado a diferentes tipos de datos o a nuevas implementaciones algorítmicas. La parte de visualización es la encargada de generar los datos que serán presentados por el modelo gráfico. VTK utiliza un flujo de datos aproximado para transformar información en datos gráficos.

- **Actividad básica implementada en VTK*:***

VTK se compone de una serie de clases que contienen métodos para la visualización de geometrías 3D, para esta actividad se procede a generar un volumen en tres dimensiones escogiendo para ello una esfera que podrá ser rotada por medio del *mouse*, además se le incluye una textura con el logo de la aplicación *software* desarrollada en el presente trabajo de grado.

Para el desarrollo de esta actividad se escogió como lenguaje de programación C++, CMake para la vinculación del proyecto y como entorno de desarrollo Visual Studio 2005. Al igual que ITK, en VTK se diseñan *pipeline* (Ver Figura 15) como estrategias de solución demandando datos en forma secuencial hasta llegar a la salida deseada por el usuario. La estrategia para este proceso es la siguiente:

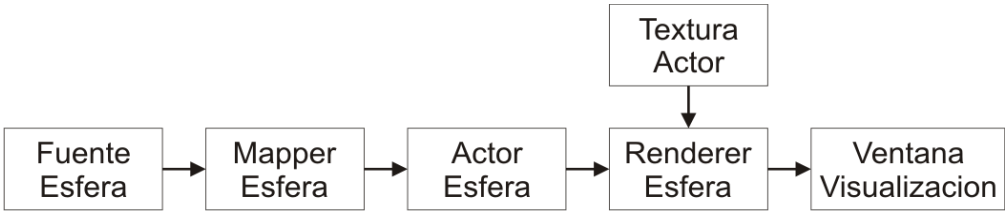


Figura 15: Pipeline para VTK

Ahora se procede con la implementación en código de dicha estrategia con lo cual se crea un archivo con el nombre main.cpp y contiene lo siguiente:

```
#include"vtkTexturedSphereSource.h"
#include"vtkPolyDataMapper.h"
#include"vtkRenderWindow.h"
#include"vtkActor.h"
#include"vtkRenderer.h"
#include"vtkRenderWindowInteractor.h"
#include"vtkInteractorStyleTrackballCamera.h"
#include"vtkPNGReader.h"
#include"vtkTexture.h"

int main(int argc, char** argv)
{
      vtkTexturedSphereSource *esferaSource =
                                    vtkTexturedSphereSource::New();
     esferaSource->SetRadius(10);
     esferaSource->SetThetaResolution(100);
     esferaSource->Update();

     vtkPolyDataMapper *esferaMapper = vtkPolyDataMapper::New();
     esferaMapper->SetInputConnection(esferaSource->
                                            GetOutputPort( ));
     esferaMapper->Update();

     vtkPNGReader *imagen = vtkPNGReader::New();
     imagen->SetFileName("logoP.png");
     imagen->Update();

     vtkTexture *textura = vtkTexture::New();
     textura->SetInputConnection(imagen->GetOutputPort());
     textura->InterpolateOn();

     vtkActor *esferaActor = vtkActor::New();
     esferaActor->SetMapper(esferaMapper);
     esferaActor->SetTexture(textura);

     vtkRenderer *esferaRenderer = vtkRenderer::New();
     esferaRenderer->AddActor(esferaActor);
     esferaRenderer->SetBackground(1,1,1);

     vtkRenderWindow *ventanaVisualizacion = vtkRenderWindow::New();
     ventanaVisualizacion->AddRenderer(esferaRenderer);
     ventanaVisualizacion->SetSize(300,300);

      vtkRenderWindowInteractor *interactor =
                  vtkRenderWindowInteractor::New();
     interactor->SetRenderWindow(ventanaVisualizacion);

      vtkInteractorStyleTrackballCamera *estilo =
                          vtkInteractorStyleTrackballCamera::New();
     interactor->SetInteractorStyle(estilo);
     interactor->Initialize();
     interactor->Start();

     esferaMapper->Delete();
     esferaSource->Delete();
     esferaActor->Delete();
     interactor->Delete();
     estilo->Delete();
     ventanaVisualizacion->Delete();
```

```
    esferaRenderer->Delete();
    imagen->Delete();
    textura->Delete();

    return 0;
}
```

En forma breve se describirán las clases más relevantes utilizadas en este ejemplo:

- **vtkTexturedSphereSource:** Genera los datos de una esfera a la cual se le podrá añadir una textura posteriormente.

- **vtkPolyDataMapper**: Genera los gráficos correspondientes a los datos ingresados.

- **vtkPNGReader:** Carga los datos de una imagen en formato png desde una ruta especificada.

- **vtkTexture:** Maneja una textura a partir de datos que pueden ser imágenes, materiales, etc.

- **vtkActor**: Representa un objeto en una escena.

Por último, se crea el archivo de configuración del proyecto CMakeLists.txt escribiendo lo siguiente:

```
PROJECT(PruebaVTK)
cmake_minimum_required(VERSION 2.6)

FIND_PACKAGE(VTK REQUIRED)
INCLUDE(${VTK_USE_FILE})

ADD_EXECUTABLE(Ejemplo main.cpp)
TARGET_LINK_LIBRARIES(Ejemplo vtkRendering)
```

Genere el proyecto a partir de las rutas de las carpetas del código y adónde se va a generar. Todo esto a través de CMake.

Una vez generado el proyecto se procede a abrir el archivo *PruebaVTK.sln* (recuerde que para poder generar el proyecto de Visual Studio 2005 debe tenerlo instalado). Compilamos el proyecto, una vez compilado podemos acceder al ejecutable en las carpetas *Debug* o *Release* según sea el modo de compilación.

En la carpeta *Debug* o *Release* debe existir una imagen de formato png con el nombre logoP.png (ver Figura 16) que representa la textura a cargar en la esfera.

Figura 16: Logo para cargar a VTK

El resultado de la actividad en VTK es el siguiente: (ver Figura 17)

Figura 17: Logo con efecto 3D

Para más información puede visitar la página web:
http://www.vtk.org/doc/nightly/html/classes.html donde podrá profundizar sobre los métodos de cada clase utilizada.

3.1.3. Introducción a QT

QT es un entorno de trabajo basado en C++ para el desarrollo de software en diferentes plataformas. QT incluye las herramientas necesarias para diseñar las aplicaciones de una forma muy rápida y sencilla. Esta biblioteca incluyen un conjunto de *widgets* (“controles” en la terminología de Windows) encargados de proporcionar las funciones estándar de una interfaz o GUI. QT presenta una alternativa para la comunicación inter-objeto denominada “*signals y slots*” que remplaza al antiguo sistema de *callbacks* utilizado por otros sistemas, además de proporcionar un modelo de eventos convencional para la captura de las pulsaciones del ratón, teclas u otras entradas del usuario haciendo posible la creación de aplicaciones multiplataforma mediante funciones estándar, incluyendo drivers para Linux, Windows, e incluso drivers nativos para bases de datos como Oracle ®, Microsoft® SQL Server, IBM DB2®, Borland.

Principios de QT (*Signals and Slots*) :(Señales y ranuras) es un mecanismo desarrollado por *TrollTech*[4] para la comunicación inte-objeto con el fin de reemplazar a los conocidos *callbacks*[5]. Este sistema es flexible, totalmente orientado a objetos he implementado en C++.

Una señal es emitida por un objeto cuando cambia su estado de alguna forma. Sólo aquellas clases que definan una señal y las subclases que deriven de ésta pueden emitir la señal. Cuando se emite una señal, el slot conectado a ella es ejecutado de forma inmediata como si se tratara de una función de llamada normal. Si varios slots están conectados a una misma señal todos ellos serán ejecutados uno después de otro pero en un orden arbitrario.

Los *slots* son funciones en C++ y por lo tanto también podrán ser llamadas de forma normal. Su única característica especial es que las señales pueden ser conectadas a una señal. En comparación con los *callbacks*, los *signals* y *slots* son mucho más sencillos y rápidos de utilizar ya que la diferencia con las aplicaciones reales es insignificante. La emisión de una señal es de cuatro a diez veces más lenta que trabajar directamente con los *callbacks* debido al tiempo requerido para localizar el objeto, pero su velocidad relativa aumenta al necesitar muchas menos operaciones de tipo *new* y *delete*. Además QT trabaja con un conjunto de *widget* que poseen funciones típicas para el diseño de interfaces de usuario, los *widget* son capaces de recibir los eventos del teclado o ratón junto a otros eventos del sistema.

Qt Designer: es un diseñador de interfaces gráficas para aplicaciones basados en las bibliotecas QT. Es posible escribir aplicaciones completamente en código fuente, o utilizando *Qt Designer* para un desarrollo más rápido. La arquitectura basada en componentes hace posible a los desarrolladores ampliar *Qt Designer* con *widgets* personalizados y extensiones. Este presenta el inconveniente de no poderse programar directamente sobre él y tener que recurrir a entornos de desarrollo para poder conectar los eventos del computador con otras aplicaciones.

Diseñar un archivo con *Qt Designer* es un proceso simple, consiste en arrastrar los ítems desde un menú a un formulario, utilizando herramientas de edición para seleccionar, cortar, pegar y cambiar el tamaño de los diferentes componentes. A la salida de este programa se obtiene un fichero con la extensión .ui que contiene una descripción de la interfaz indicando sus características, componentes y forma de comportamiento.

- **Actividad básica en *QT Designer***

En este ejemplo se va a diseñar una pequeña aplicación que consiste en un *widget* que contenga varios elementos de la *Widget Box* y conectarlos con eventos.

[4]TrollTech: Compañía de *software* de computadoras de Noruega
[5]Callbacks: Función que recibe como argumento la dirección o puntero de otra función.

Lo primero que se debe hacer es abrir *Qt Designer* y posteriormente se da clic en *File->New* con lo cual se abrirá un cuadro de dialogo preguntando qué es lo que se desea crear, en la parte izquierda aparecen diferentes opciones de formularios que por defecto están en la aplicación, en el menú *templates\forms* se selecciona *widget* y por último se da clic en el botón *Create*, con lo cual se despliega un formulario de trabajo vacío y listo para comenzar. (Ver Figura)

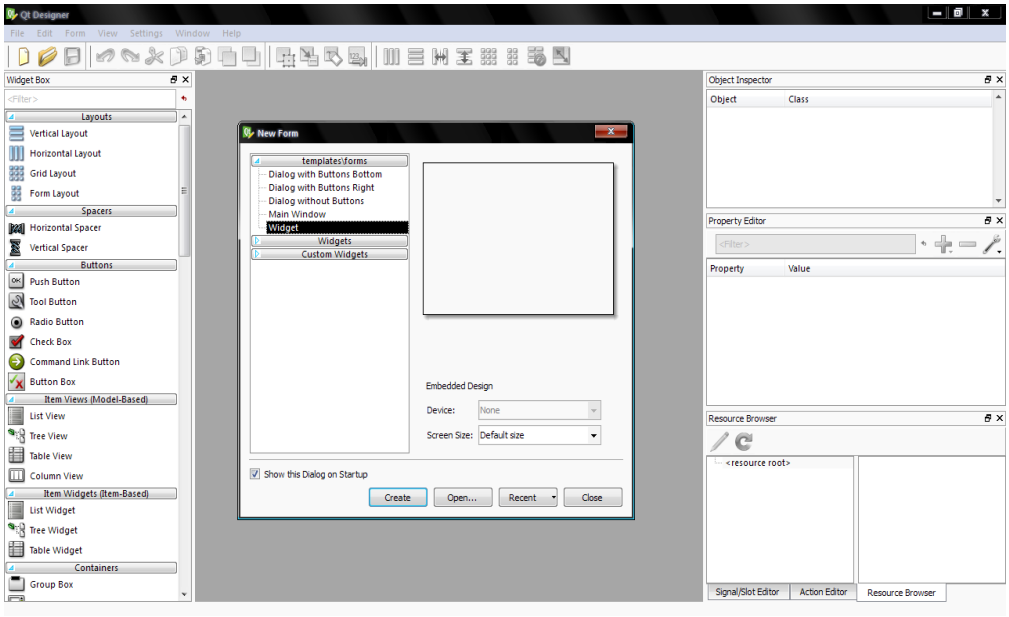


Figura 18: *QT Designer*

En la parte de la izquierda se encuentra una caja que contiene una serie de *widget* para el diseño de una interfaz de usuario. Inicialmente se busca el *push button* y se arrastra al formulario, se puede observar que ahora el botón hace parte de él (ver Figura 19). En la parte de la derecha una vez seleccionado el botón se observa el editor de propiedades, donde se pueden cambiar atributos del botón como el nombre, la posición, el tamaño, la fuente del texto, entre otras. Para este ejemplo solo cambiaremos el texto que aparece en el botón y su correspondiente nombre, para ello en el editor de propiedades vaya a los campos *text* y *objectName,* y escriba boton1 en los dos.

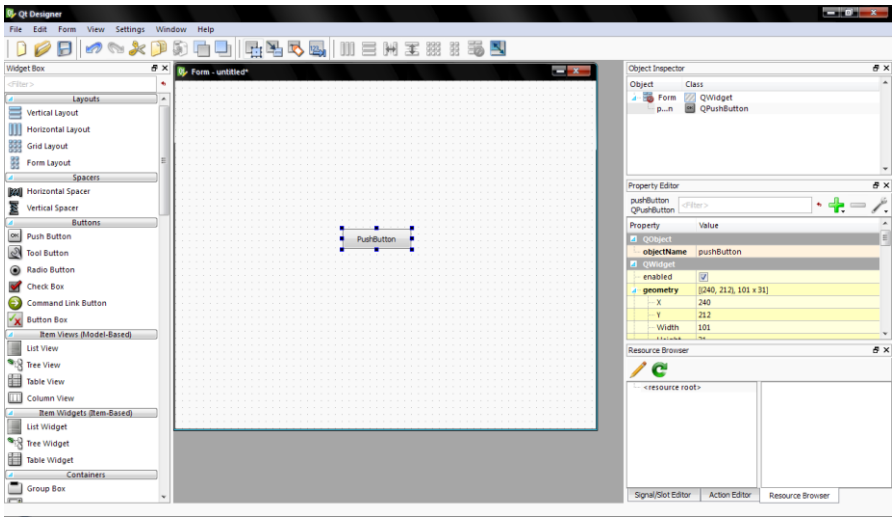


Figura 19: Diseño en *QT Designer*

Cree dos botones más con los textos y nombre boton2 y boton3 respectivamente, además cree un *Horizontal Slider*, un *Spin Box* y un *Text Browser* que también se encuentran en la caja de *widget*. El formulario debe verse como se muestra en la Figura 20. Recuerde que los *widgets* pueden ser movidos por el formulario solo con arrastrarlos y soltarlos con el *mouse* en la posición deseada.

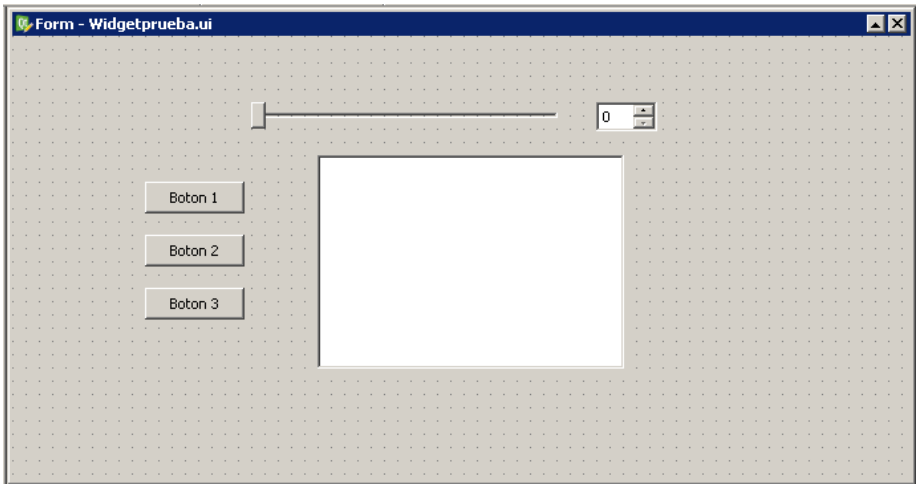


Figura 20: *Form en QT Designer*

Una vez finalizada la creación se procede a guardarla en una carpeta que como el ejemplo anterior va a contener solo el código de la aplicación. Puede colocarle cualquier nombre pero a modo de sugerencia puede ser “WidgetPrueba.ui”.

La conexión de los eventos de los diferentes *widgets* dentro del formulario se puede hacer de dos maneras a través de la herramienta *Qt Designer* o a través de código, la primera es menos potente puesto que solo se puede conectar o enviar

información limitadas por la aplicación. Para conectar señales se da clic en el botón *Edit Signals/Slots* de la barra de herramientas ubicada en la parte superior de la aplicación como se observa en la Figura 21.

Figura 21: Edición de *Signals y Slots*

Una vez allí la conexión es relativamente sencilla, solo con arrastrar el *mouse* desde el objeto que envía la señal hasta el objeto receptor creando una línea como se muestra en la Figura 22, automáticamente se despliega por la pantalla un cuadro de dialogo que muestra las señales del objeto emisor y las funciones que las reciben en este caso el objeto receptor. La señal escogida para el *Horizontal Slider* es *valueChanged (int)* que se enviará siempre y cuando éste cambie su valor, el *Slot* o función del *Spin Box* que recibe esta señal es *setValue (int)* con lo cual cada vez que se mueve el *Horizontal Slider* este valor se actualiza en el *Spin Box.* Una vez finalizada esta edición se puede volver a la forma inicial de edición de *widgets* presionando clic izquierdo en el botón *Edit Widget* en la barra de herramientas.

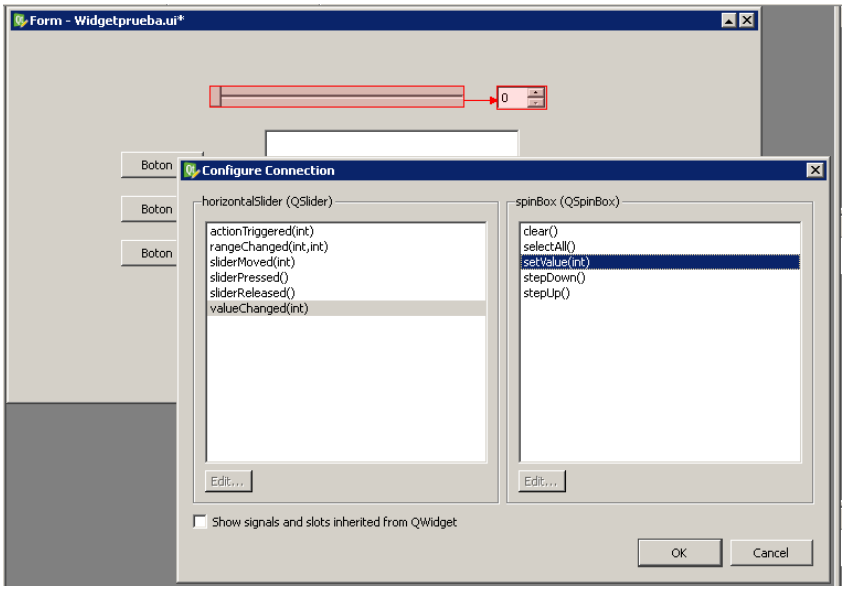


Figura 22: Conexión de *Signals y Slots*

Para probar el funcionamiento de la conexión realizada se puede escoger en el menú *Form->Preview* o presionar las teclas Ctrl+R para visualizar la interfaz de usuario. En ésta se puede observar que cuando se cambia la posición del *Horizontal Slider* este valor automáticamente se cambia el valor en el *SpinBox*, como se observa en Figura 23.

Figura 23: Prueba de funcionamiento del *Slider*

La otra forma de usar el diseño es a través de código para lo cual se necesitaría un IDE de desarrollo que para este ejemplo será *Visual Studio 2005* y, para crear y vincular el proyecto se usa CMake. Primero se debe crear el código de la aplicación para ello cree en la carpeta donde se guardó el *.ui* de la interfaz los siguientes archivos con un editor de texto.

Cree un archivo con el nombre Interfaz.h y guárdelo en la carpeta donde está guardando todo lo correspondiente al código y escriba lo siguiente:

```
#ifndef INTERFAZ
#define INTERFAZ

#include"ui_Widgetprueba.h"

#include<QtGui>
#include<QApplication>

class Interfaz: public QWidget, private Ui_Form
{
     Q_OBJECT
public:
     Interfaz(QWidget *parent = 0);
public slots:
     void accion_Boton1();
     void accion_Boton2();
     void accion_Boton3();
};
```

Cree un archivo con el nombre Interfaz.cpp y guárdelo en la carpeta donde está guardando todo lo correspondiente al código y escriba lo siguiente:

```
#include"Interfaz.h"

Interfaz::Interfaz(QWidget* parent): QWidget(parent)
{
    setupUi(this);
    connect(boton1,SIGNAL(clicked()),this,SLOT(accion_Boton1()));
    connect(boton2,SIGNAL(clicked()),this,SLOT(accion_Boton2()));
    connect(boton3,SIGNAL(clicked()),this,SLOT(accion_Boton3()));
}
void Interfaz::accion_Boton1()
{
    textBrowser->append("Boton1 presionado");
}
```

```
void Interfaz::accion_Boton2()
{
    textBrowser->append("Boton2 presionado");
}
void Interfaz::accion_Boton3()
{
    textBrowser->append("Boton3 presionado");
}
```

Cree un archivo con el nombre main.cpp y guárdelo en la carpeta donde está guardando todo lo correspondiente al código y escriba lo siguiente:

```
#include<qapplication.h>
#include"Interfaz.h"

int main( int argc, char** argv )
{
    QApplication app( argc, argv );
    Interfaz mainWidget;
    mainWidget.show();
return app.exec();
}
```

Por último cree el archivo de configuración del proyecto CMakeLists.txt y escriba lo siguiente:

```
PROJECT(PruebaWidget)
cmake_minimum_required(VERSION 2.6)

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

SET(Header Interfaz.h)
SET(Source Interfaz.cpp main.cpp)
SET(UisWidgetprueba.ui)

QT4_WRAP_UI(Ui_Header ${Uis})
QT4_WRAP_CPP(Moc_Source ${Header})

ADD_EXECUTABLE(Ejemplo ${Header}${Source} ${Uis} ${Moc_Source}
                ${Ui_Header})
TARGET_LINK_LIBRARIES(${QT_LIBRARIES})
```

Para el uso por código es necesario crear la aplicación como una clase heredada de la generada por el *Qt Designer* y utilizar de esta manera el método setupUI() para inicializarla. CMake es la que se encarga de configurar el proyecto para que genere el archivo cabecera de extensión .h que contiene la definición del .ui a través del comando QT4_WRAP_UI y enlazar los diferentes *signals y slots* en la aplicación generando un archivo de código utilizando el comando QT4_WRAP_CPP.

Genere el proyecto a partir de las rutas de las carpetas del código y a donde se va a generar, todo esto a través de CMake. Una vez generado el proyecto abrimos el archivo *PruebaWidget.sln* (recuerde que para poder generar el proyecto de *Visual Studio 2005* debe tenerlo instalado). Se compila el proyecto, una vez compilado

podemos acceder al ejecutable en las carpetas *Debug* o *Release* según sea el modo de compilación.

En la Figura 24 se muestra el resultado de una interfaz sencilla que indica las principales funciones en *Qt Designer*.

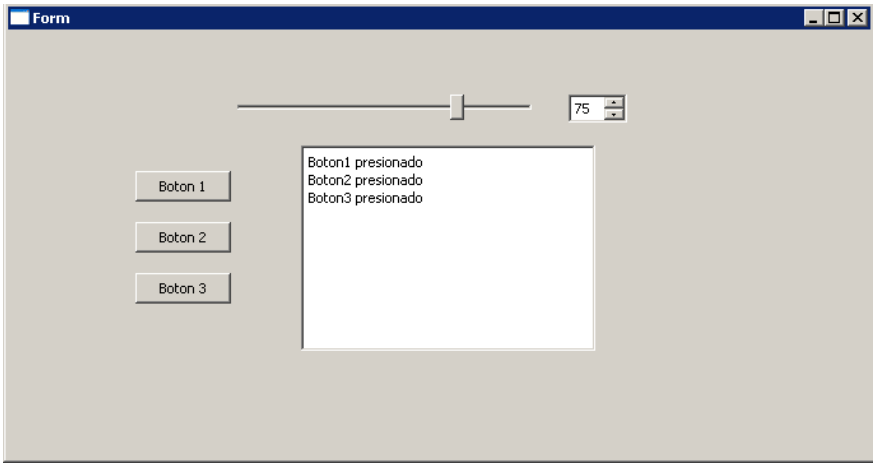


Figura 24: Interfaz en *QT Designer*

3.1.5. *V-COLLIDE*

V-Collide es una biblioteca de detección de colisiones completamente gratuita para uso no comercial desarrollada por el grupo GAMMA (*Geometric algorithm for modeling, motion and animation*) de la Universidad de Carolina del Norte (*The University of North Carolina at Chapel Hill*). Esta biblioteca realiza una detección rápida y exacta de colisiones entre modelos poligonales triangulados, por lo que puede ser usada para realidad virtual y en una gran variedad de aplicaciones de simulación. La biblioteca está diseñada para operar en ambientes que contienen un gran número de objetos geométricos formados con mallas de triángulos [24].

Los pasos básicos para el uso de esta biblioteca son: crear los objetos, añadir conjuntos de triángulos a estos objetos, elegir qué pares de objetos para ser evaluados, cambiar la posición de estos objetos actualizando la matriz, ejecutar la evaluación de colisiones y conseguir los resultados de la evaluación.

Para mayor información remítase a los ejemplos del programa que está en la carpeta: VCOLLIDE\demos. Claro está después de haber instalado la biblioteca correctamente en su computador.

Para mayor información sobre las bibliotecas explicadas diríjase al documento de J.Reolind, “Generación de órganos sólidos a partir de imágenes médicas” [22].

4. SISTEMA PROPUESTO PARA LLEVAR ACABO LA SEGMENTACIÓN Y RECONSTRUCCIÓN DE UN ÓRGANO HUMANO.

La idea básica planteada en este trabajo de grado es diseñar una aplicación software para dar solución al problema anteriormente expuesto en el capítulo 3, por lo tanto después de conocer generalmente las herramientas software para diseñar el programa se procede a dividir la solución del problema y de la misma manera proponer un sistema adecuado para cumplir con cada uno de los objetivos propuestos.

Específicamente este capítulo tiene como objetivo principal explicar y aclarar las bases teóricas de segmentación y visualización en tres dimensiones utilizadas, y a la vez ir explicando cómo se dispuso en la aplicación las cuatro partes esenciales que componen el sistema propuesto: pre-procesamiento, segmentación, reconstrucción y exportación del órgano al escenario quirúrgico.

4.1. PRE-PROCESAMIENTO

Una imagen es una colección de medidas o valores en el espacio bidimensional (2D) o tridimensional (3D). En imágenes discretas 2D, la posición de cada punto se conoce como píxel y en imágenes 3D, se le llama voxel [18] - [19]. Las imágenes pueden ser adquiridas en el dominio continuo como las películas de Rayos X, o en el discreto como en MRI (Imágenes de Resonancia Magnética).

Más concretamente una imagen medica es un conjunto de procesos y técnicas que permiten visualizar una zona del cuerpo humano, esencialmente las imágenes utilizadas para este proyecto, fueron las TAC (tomografías axiales computarizadas), las cuales permitieron realizar una reconstrucción optima, puesto que los niveles de gris están mejor definidos y presentan diferencias visuales entre ellos, no obstante dentro de la aplicación software desarrollada se puede realizar segmentación y reconstrucción con diferentes imágenes médicas tales como resonancia magnéticas y tomografía por emisión de positrones.

El pre-procesamiento inicia con la carga de las imágenes las cuales se obtuvieron de una base de imágenes gratuita en internet[6] que contiene diferentes imágenes médicas de algunas zonas del cuerpo humano en formato DICOM (*Digital Imaging and Communication in Medicine*) el cual es un estándar médico para la intercomunicación de imágenes médicas. Para realizar la función de carga de imágenes se dispuso de un asistente o *wizard* que se explicara en detalle en el capítulo 5 donde se presenta los resultados, desempeño y funcionalidad de la aplicación.

[6] http://pubimage.hcuge.ch:8080/

Las técnicas de procesamiento de imágenes permiten realzar y restaurar el contenido de una imagen y facilitar la extracción de información para su posterior interpretación. Estas técnicas se pueden emplear para disminuir o eliminar posibles artefactos o imperfecciones en las Tomografías Computarizadas que se deben principalmente a [9]:

- Ruido cuántico debido a fluctuaciones en los rayos X.
- Dispersión de los rayos X en el paciente.
- Movimientos del paciente o de órganos.
- Objetos metálicos.
- Fallas en la calibración del escáner de TC.
- Movimiento no uniforme del escáner.

La reducción de estos problemas y en especial los relacionados con ruido en la imagen, se pueden modificar utilizando técnicas de filtrado en el dominio de la frecuencia y en el espacio. El filtrado espacial es una operación "local" en procesamiento de imagen en el sentido de que modifica el valor de cada píxel de acuerdo con los valores de los píxeles que lo rodean; se trata de transformar los valores de grises originales de tal forma que se parezcan o se diferencien más de los correspondientes a los píxeles cercanos [9].

Existen diferentes tipos de filtros tales como los pasa bajos que asemejan los valores de grises de cada píxel al valor de grises de los píxeles vecinos, reduciendo la variabilidad espacial de la imagen. Ello produce pérdida en los bordes y en la nitidez visual de la imagen, pero ganando en homogeneidad. Para ello se dispone de una máscara de convolución generalmente representada por matrices estándar concretas, para cada filtro existe una matriz como la presentada a continuación.

$$\begin{vmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{vmatrix}$$

Los elementos de cada matriz expuesta tanto en el documento como en la aplicación fueron extraídas de libro de Arturo Escalera titulado ‘Visión Computacional fundamentos y métodos’ [14].

La ecuación (4.1) que se encuentra deducida en la página 24 del documento de Pérez Byron “*Modulo de navegación para un sistema de entrenamiento virtual aplicado a cirugía de la base del cráneo*” [9], representa un *kernel* de convolución, caracterizado por la desviación estándar σ, de un filtro Gaussiano que se utiliza para reducir el ruido de las imágenes.

$$H(u,v) = \frac{1}{2 * \pi * \sigma^2} e^{-(u^2+v^2)/(2*\sigma^2)} \quad (4.1)$$

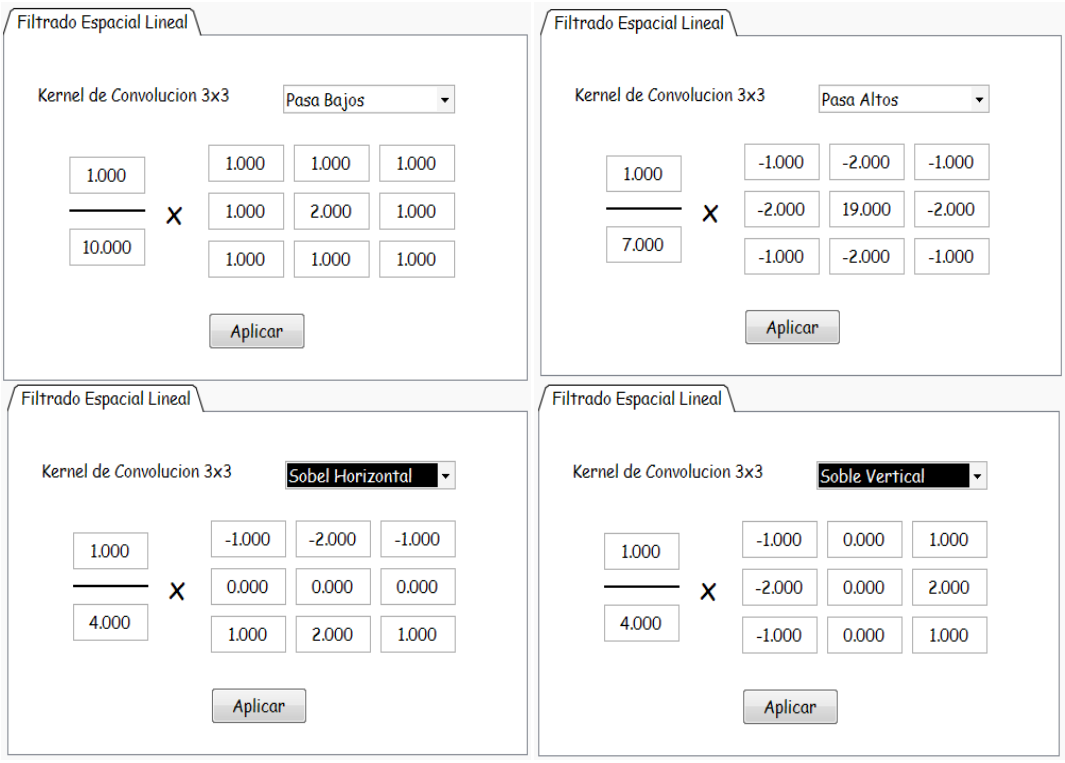
En la aplicación desarrollada se situaron los filtros: pasa bajos, pasa altos y filtros detectores de borde tales como *Sobel* y *Prewitt* tanto vertical y horizontal, junto con la matriz correspondiente a cada *kernel* perteneciente a cada uno de los filtros como se observa en la Figura 25.

Los distintos filtros dispuestos para el pre-procesamiento de las imágenes principalmente reducen el ruido y permite realzar las características que el usuario desee, por lo tanto la aplicación es flexible para cualquier imagen. Los filtros que mejores resultados generaron fueron los filtros de detección de bordes en este caso el filtro pasa altos, más adelante explicado, puesto que delinearon los órganos colocando límites entre ellos así poder lograr una mejor segmentación.

El filtro pasa altos enfatiza las altas frecuencias, para mejorar o afilar las características lineales como carreteras, fallas, o límites en general. Realizan por tanto el efecto contrario a los filtros pasa bajos, eliminando éstos las bajas frecuencias.

La Máscara de convolución o (kernel) utilizado para un filtro pasa-alto es:

$$\begin{vmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{vmatrix}$$



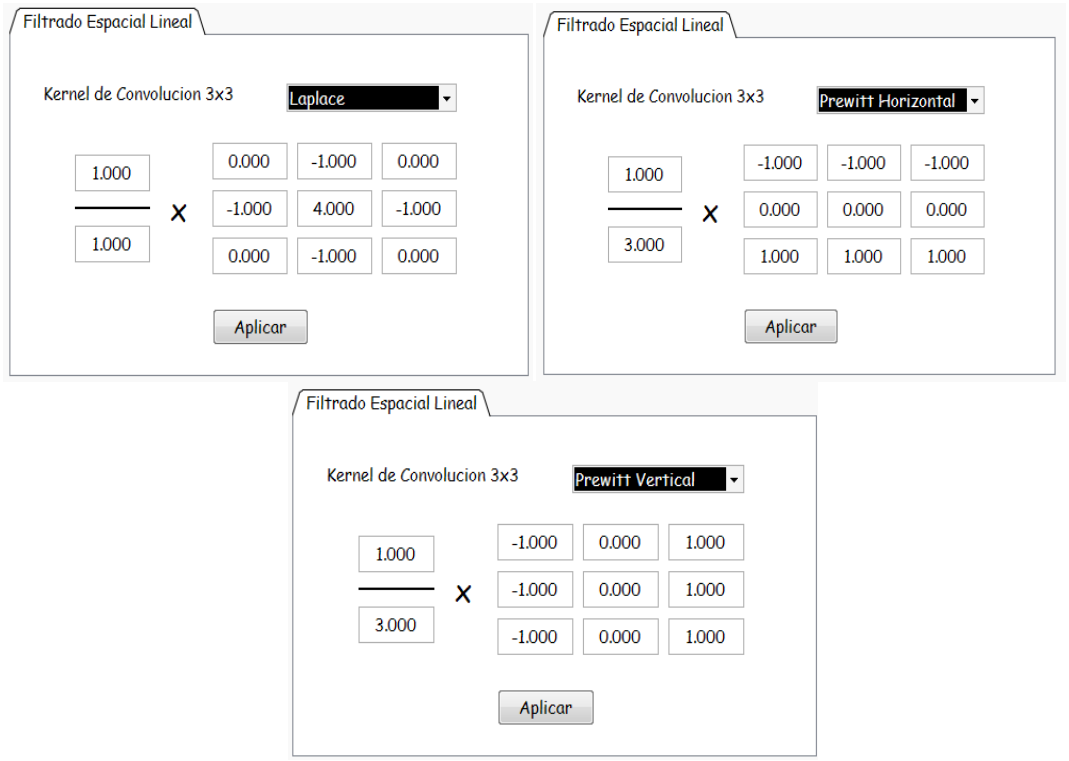


Figura 25: Opciones de filtros para el pre-procesamiento

En el capítulo 5 se encuentran los resultados con cada uno de los filtros dispuesto en el desarrollo de la aplicación.

Otra parte sustancial del pre-procesamiento consiste en eliminar zonas inoficiosas o realizar un trasformación geométrica sobre la imagen, para esta parte se dispuso una área dentro la aplicación la cual recorta la imagen eliminando así píxeles que no contienen información de la región a segmentar aumentando la velocidad de procesamiento a la hora de reconstruir, esta parte será explicada en el capítulo 5 con más detalle.

4.2. SEGMENTACIÓN

Generalmente la segmentación de imágenes se define como la partición de una imagen en regiones constituyentes no solapadas, las cuales son homogéneas con respecto a una característica como una intensidad o una textura [18]. La palabra 'segmentación' [19] tiene un significado que depende del uso y del contexto en el cual se utilice. El objetivo básico de la segmentación es definir una partición del espacio.

La segmentación consiste en hallar la estructura interna del conjunto de descripciones de los objetos (píxeles) en el espacio de representación (imagen). Esta estructura interna evidentemente depende, en una primera instancia, de la

selección de los píxeles y de la forma en que éstos se comprarán, es decir, del concepto de similitud que se utilice y de la forma en que éste se emplee [18] - [20].

4.2.1. Métodos de Segmentación

Los métodos para llevar a cabo las segmentaciones varían dependiendo de la aplicación, tipo de imagen, y otros factores. Por ejemplo, la segmentación del cerebro tiene diferentes requerimientos que la segmentación de un hígado [19]. Actualmente no existe un método de segmentación que alcance resultados aceptables para todo tipo de imagen médica, no existen pasos que sean generales y que puedan ser aplicados a cualquier variedad de datos, por esto a continuación se expone algunos de los métodos que se pueden explorar dentro de la aplicación *software* desarrollada correspondiente al fragmento denominado 'segmentación' y de esta forma seguir los pasos propuestos para dar solución al problema general.

4.2.1.1. Threshold

Es un método que busca segmentar imágenes escalares creando una partición binaria de las intensidades de las imágenes.*Threshold* trata de determinar un valor de intensidad, llamado umbral, que separa las zonas deseadas [18]. Una forma de segmentar órganos en imágenes de TC (tomografía computarizada) consiste en la separación de píxeles usando sus valores de intensidad de acuerdo con la Tabla 1 o aplicando funciones de opacidad sobre toda la imagen [9].

En esta parte del proceso la imagen se binariza con el fin de reducir la información de la misma, donde persisten valores que pueden ser representados por 0 y 1, o más frecuentemente por los colores negro y blanco, la finalidad de este análisis es separar la región u órgano de interés dentro de la tomografía. Los valores de los píxeles en las imágenes resultantes de TC (tomografía computarizada) se miden en unidades de *Hounsfield* (HU) y se obtienen a partir de una transformación lineal sobre la atenuación medida (ver ecuación (4.2)). Se define como cero el valor del coeficiente de atenuación para el agua destilada y de -1000 para el aire [9].

$$\frac{\mu x - \mu_{H_2O}}{\mu_{H_2O} - \mu_{air}} * 1000 \quad (4.2)$$

Donde μ_{H2O} y μ_{air} son los coeficientes de atenuación lineal del agua y del aire respectivamente en condiciones normales y presión de 1bar.

Tabla 1: Valores en unidades Hounsfield de distintas sustancias [9].

Unidades de Hounsfield	
Aire	-1000
Tejido Graso	-120 -150
Agua	0
Riñones	30
Musculo	10 – 40
Sangre	40
Hígado	10 – 40
Hueso	400

En el capítulo 5 se mostrarán los resultados con este método que arrogo dentro de la aplicación *software* desarrollada.

4.2.1.2. Crecimiento de regiones

Es el proceso básico para la fusión de regiones, se combinan basándose en el criterio de homogeneidad, hasta alcanzar una decisión final. El proceso puede ser inicializado definiendo un sistema de etiquetado que cubre parcialmente o completamente la partición, donde las regiones pueden estar formadas únicamente por un píxel. El análisis de todas estas posibles combinaciones puede resultar costoso para criterios de homogeneidad complejos [19]. A continuación se explican algunos métodos basados en las premisas de crecimiento de regiones.

- ***Método level sets*:** el método de *level sets*, desarrollado por *Osher* y *Sethian*, [9] es una técnica numérica para la deformación de una hipersuperficie implícita, hace parte del método del crecimiento de regiones. Permite evolucionar en el tiempo un contorno cerrado r (curva en el caso 2D, superficie en el caso 3D o hipersuperficie en el caso n-dimensional) definido dentro de la imagen y detenerlo de acuerdo con una función de velocidad F.

Las siguientes ecuaciones que hacen parte del método *level set* y que se encuentran deducidas en el documento de Pérez B [9]. La función de *level set* Φ se define como una función implícita y se representa como una transformada de la distancia euclidiana (ver ecuación (4.3)).

$$\Phi\left(x, t_{=0}\right) = \pm d \quad (4.3)$$

Donde x es un punto dentro de la imagen, t es el tiempo y ± d es la distancia de x hasta r (t = 0) donde el signo indica si la función está evaluada dentro o fuera del contorno. Para evolucionar el contorno, es decir para que exista una expansión o

contracción, se aplica una función de velocidad que determina como se mueve cada punto. La Figura 26 muestra la evolución en el tiempo t de un contorno Γ definido en el plano XY [9].

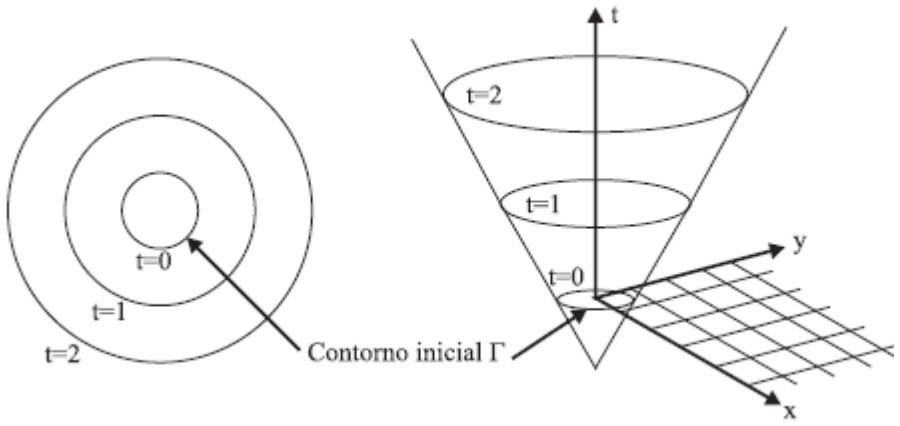


Figura 26: Evolución de un contorno en un tiempo t

Fuente: Modulo de Navegación para un Sistema de Entrenamiento Virtual Aplicado a Cirugía de la Base del Cráneo [9]

Teniendo en cuenta que el contorno Γ es el *level set* cero de la función Φ (x; t) en R^n, se define Γ como:

$$\Gamma(t) = \{x \in R^n | \phi(x,t) = 0\} \quad (4.4)$$

La formulación dinámica de la función Φ(x; t) se puede escribir como:

$$\frac{\partial \phi}{\partial t} = -\nabla\phi * \frac{dx}{dt} = -\nabla * F\,(x, D\phi, D^2\,\phi, \dots) \quad (4.5)$$

Donde D^n es el conjunto de las derivadas de orden-n de Φ evaluadas en x.

La función de velocidad F se escoge de tal forma que se pueda recuperar la forma de un objeto, utilizando el gradiente de la imagen y la curvatura del contorno Γ. Para el caso de imágenes médicas el contorno inicial se ubica dentro de la cavidad anatómica que se quiere segmentar y se evoluciona hasta que se recupere la forma deseada. La ecuación (4.6) muestra la función de velocidad [9].

$$F = P(I)(1 - \varepsilon k) \quad (4.6)$$

Donde P (I) es el término dependiente del gradiente de la imagen I, 0 <ε< 1 es una constante y *k* es la curvatura del contorno [9].La ecuación (4.7) hace que la velocidad tienda a cero cuando el gradiente de la imagen es alto, es decir, que el contorno que está evolucionando se detenga cuando se presente un cambio alto en la imagen.

$$P(I) = e^{-|\nabla(G\sigma * I)|} \quad (4.7)$$

La curvatura está dada por la ecuación (4.8) y se obtiene de la divergencia del gradiente del vector normal al contorno [9].

$$\mathrm{k} = \frac{\Phi_{xy}\ \Phi_{y^2} - 2\Phi_x\Phi_y\Phi_{xy}\ - \Phi_{yy}\ \Phi_{x^2}{}^{3/2}}{(\Phi_{y^2} + \Phi_{yx})} \quad (4.8)$$

Para solucionar la ecuación (4.5) de *level set* se utiliza un esquema iterativo de diferencias finitas, que se resuelve en forma computacionalmente eficientemente únicamente dentro de una banda angosta (*narrow band*) alrededor del *level set* cero. Se realiza un ciclo resolviendo la ecuación dentro de la banda y reconstruyendo la en el borde de la anterior. Este enfoque hace que no tenga que resolverse la ecuación dentro de toda la imagen, sino únicamente dentro de la banda que se va desplazando hacia afuera de acuerdo con la función de velocidad [9].

Asumiendo que la velocidad F no cambia de signo para que el contorno siempre se expanda (F >0), se define una retícula sobre el contorno y se determina el tiempo T empleado para pasar por cada punto (x;y). El contorno inicial evoluciona hasta que la función de velocidad es igual a cero. Este método, llamado *fast marching*, se define en la ecuación (4.9).

$$|\nabla T|F = 1 \quad (4.9)$$

El algoritmo utiliza un ciclo iterativo intercalando el método de *fast marching* para construir la función de distancia con signo y el método de *level sets* para evolucionar el contorno dentro de la *narrow band*. Este ciclo se realiza hasta que el contorno se aproxime a la estructura anatómica que se quiere segmentar, es decir, cuando la velocidad de evolución de cada punto del contorno se acerque a cero.

Al igual que los otros métodos, por lo general no se utiliza la región creciente solamente en una imagen, sino que se utiliza como parte de todo un conjunto de operaciones de procesamiento de imágenes, particularmente en la delineación de pequeñas y simples estructuras de órganos carentes de forma como es el caso del hígado. Su desventaja principal es que requiere interacción manual para obtener los

puntos semilla que son píxeles necesarios para iniciar los diferentes métodos de crecimiento de regiones, a partir de los cuales se generan contornos o regiones determinadas que segmentan las áreas de interés, además este método demanda mucho tiempo para la obtención de resultados.

En la aplicación desarrollada se dispuso la parte de segmentación como se observa en la Figura 27, donde se despliega la pestaña que muestra la posición en x, y, z en píxeles y la posición exacta del punto escogido, En la parte de la segmentación se puede escoger los diferentes métodos (ver Figura 28) junto con los parámetros definidos para cada uno de ellos, algunos de estos son escogidos a prueba y error y otros a partir de umbrales ya determinados dependiendo el órgano.

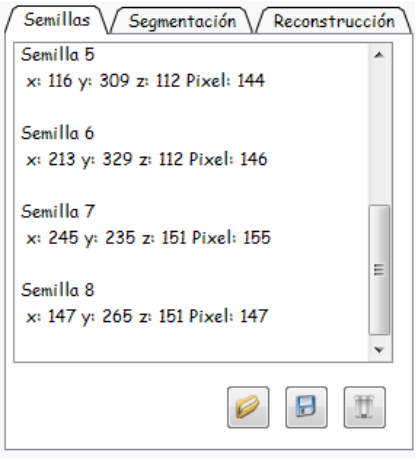


Figura 28

Figura 27: Escogencia punto semilla en la aplicación final

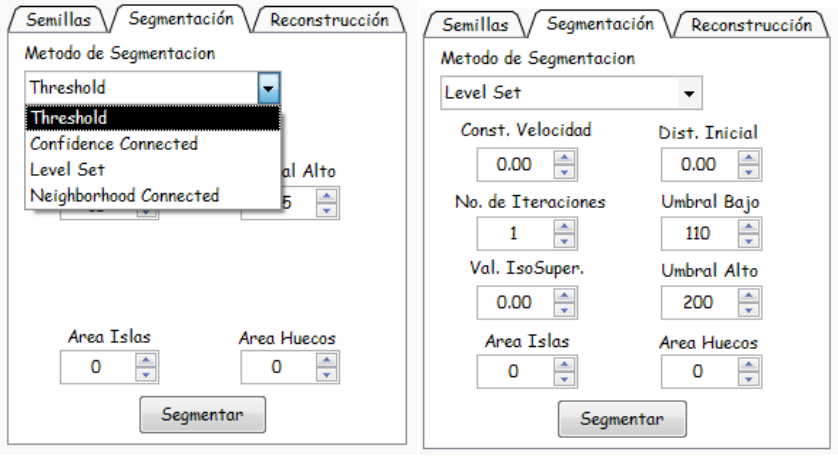


Figura 28: Parte de segmentación

Los métodos de segmentación que se pueden realizar son variados y arrojan distintos resultados unos mejores que otros claro está dependiendo de la imagen que se desee segmentar. Algunos métodos utilizados en la aplicación desarrollada se explican brevemente en el apartado 5.1 del capítulo 5.

En la aplicación desarrollada se debe ingresar parámetros que sirven para iniciar el método *level set*. A continuación en la Tabla 2 se explicarán brevemente estos parámetros.

Tabla 2: Parámetros método *level set*

Parámetro	Significado
Umbral Alto y Bajo	Limita los píxeles en un rango específico según un histograma.
Valor Dentro y Fuera	Valor de las imágenes binarizadas para clasificar entre negro, para fuera de la estructura anatómica segmentada, y blanco órgano segmentado.
Constante Velocidad	Esta constante pertenece a la función de velocidad, se escoge de tal forma que se pueda recuperar la forma del objeto a segmentar, utilizando el gradiente de la imagen y la curvatura del contorno. Para el caso de imágenes médicas el contorno inicial se ubica dentro de la cavidad anatómica que se quiere segmentar y se evoluciona hasta que se recupere la forma deseada [9] [26].
Nº de Iteraciones	Número de veces que la función de *level sets* se repite. Este valor se determina por el valor de la distancia que se debe recorrer dentro de la superficie a segmentar.
Distancia Inicial	Es la distancia necesaria para recorrer la superficie en su totalidad.

- ***Método confidence connected:*** [26] El criterio utilizado por el *confidence connected* se basa en simples estadísticas de la región actual. En primer lugar, el algoritmo calcula la media y la desviación estándar de valores de intensidad para todos lo píxeles incluidos actualmente en la región.

El usuario ingresa siempre el factor que se usa para multiplicar la desviación estándar y definir un rango alrededor de la media. Cuando no hay más píxeles que satisfacen el criterio, se considera que el algoritmo ha terminado su primera iteración. En ese momento, la media y desviación estándar de los niveles de intensidad se vuelven a calcular utilizando todos los píxeles incluidos actualmente en la región.

Este proceso iterativo se repite hasta que no se añaden más píxeles o el número máximo de iteraciones es alcanzado. La siguiente ecuación (4.11) indica el criterio de este filtro.

$$I(X) = \in [m - \ f\sigma, m + \ f\sigma] \qquad (4.11)$$

Donde m y σ son la media y desviación estándar de las intensidades de la región, f es un factor definido por el usuario, I es la imagen y X es la posición de los píxeles vecinos que se consideraron para su introducción en la región.

En la aplicación desarrollada es debe ingresar parámetros que sirven para llevar a cabo cada una de las ecuaciones dispuesta por este método. A continuación en la Tabla 3 se explicarán brevemente estos parámetros.

Tabla 3: Parámetros método *confidence connected*

Parámetro	Significado
Factor	Proporcionado por el usuario, se utiliza para multiplicar la desviación estándar y definir un rango alrededor de la media [26].
Nº de Iteraciones	Cuando no se encuentra un vecino que satisface el criterio, el algoritmo se considera que ha terminado su primera iteración. En ese momento, la media y desviación estándar de los niveles de intensidad se vuelven a calcular [26].
Radio Vecinos	Valor donde se encuentran píxeles cuya intensidad está dentro de una gama, entonces son aceptados e incluidos en la región.

Para mayor información acerca de métodos de segmentación diríjase "*The ITK software guide second edition update for ITK*" [26].

En el capítulo 5 se encuentran los diferentes resultados obtenidos a partir de los métodos dispuestos dentro de la aplicación software.

4.3. RECONSTRUCCIÓN Y VISUALIZACIÓN 3D

Existen dos métodos fundamentales para la reconstrucción de escenas 3D a partir de imágenes bidimensionales. La primera es conocida como estereoscopía o imagen estereoscópica. Con el fin de determinar la profundidad se combina la información de dos imágenes muy próximas entre sí. La segunda técnica es más avanzada y precisa. Consiste en la creación de objetos 3D a partir de imágenes transversales del objeto [22].

4.3.1. Creación de modelos 3D

Existen variedad de métodos para la obtención de objetos 3D a partir de imágenes. Cada método está basado en una técnica diferente, orientada a un tipo de imágenes específicas o a un determinado sistema, pero la gran mayoría siguen una serie de pasos comunes:

1. Pre-procesamiento y segmentación
2. Creación de modelos 3D (*Algoritmo Marching Cubes)*
3. Suavizado del mallado tridimensional.

La reconstrucción 3D es un proceso secuencial donde por cada etapa que la información trascurre, alcanza una salida y con ello un resultado. En función de la técnica cada etapa se encontrará más o menos personalizada pero en esencia serán comunes en todos los sistemas.

El primer paso, el procesado bidimensional o segmentación permite obtener las imágenes de los cortes transversales del abdomen debidamente segmentados como se explicó con anterioridad. Con el pre-procesado y la segmentación se obtuvo una máscara binaria, conforme con la cual son creados los objetos tridimensionales en el espacio 3D.

A continuación se analiza el algoritmo para la creación de modelos en 3D (*Algoritmo Marching Cubes*) como también se explican algunos filtros disponibles en VTK utilizados para suavizar la malla resultante de la reconstrucción.

4.3.2. *Algoritmo Marching* Cubes

Este algoritmo, diseñado por *William E. Lorensen y Harvey E. Cline* en 1987 [9], permite una representación implícita de superficies con densidad constante en una imagen 3D utilizando polígonos [9]. El mallado se genera teniendo en cuenta la pertenencia o no de cada uno de los 8 vecinos que forman un cubo con ancho de un voxel (ver Figura 29) al isocontorno que aproxima la superficie interna. Los 256 casos totales se obtienen a partir de reflexiones y rotaciones simétricas de los 15 casos mostrados en la Figura 30.

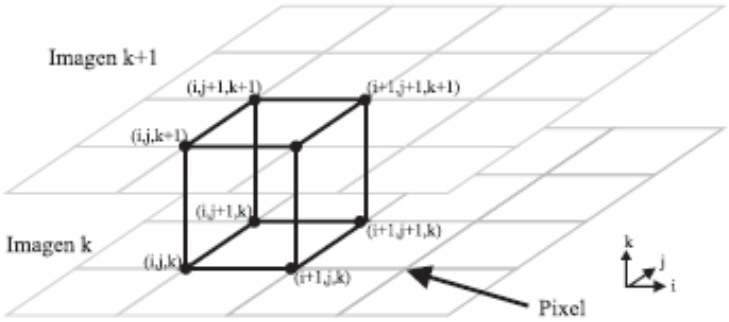


Figura 29: *Algoritmo Marching Cubes*

Fuente: Modulo de Navegación para un Sistema de Entrenamiento Virtual Aplicado a Cirugía de la Base del Cráneo [9]

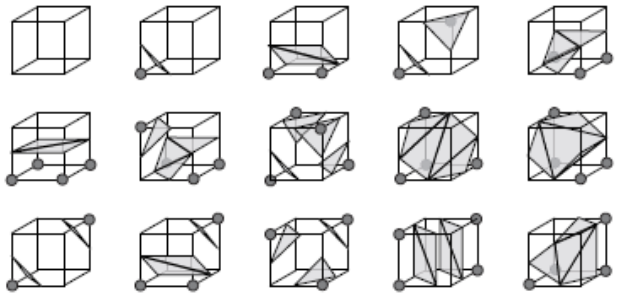


Figura 30: Casos del *Algoritmo Marching Cubes*

Fuente: Modulo de Navegación para un Sistema de Entrenamiento Virtual Aplicado a Cirugía de la Base del Cráneo [9]

Marching Cubes es un algoritmo para la representación de isosuperficies[7]para datos volumétricos. La idea básica es que se puede definir un voxel de los valores de píxeles en las ocho esquinas del cubo. Si uno o más píxeles de un cubo tienen valores inferiores al nivel gris especificado por el usuario, y uno o más tienen valores superiores a este valor, se sabe que el voxel debe contribuir algún componente de la malla superficial. Al determinar qué bordes del cubo son cortados por la malla, se pueden crear triángulos dentro del cubo que dividen éste entre las regiones de la superficie y regiones fuera. Al conectar los triangulares de todos los cubos en el límite de la malla, se obtiene una representación de la superficie [9].

El objetivo de esta técnica es generar el mallado triangular de la superficie externa del objeto 3D contenido en el volumen que se desea renderizar. El mallado se genera localmente teniendo en cuenta la pertenencia o no de los voxels (cubos) más próximos a uno situado en el límite del objeto. Dicha condición de pertenencia

[7]**Isosuperficies**: Es una superficie en tres dimensiones que representan un conjunto de valores de una función continua cuyo dominio es el espacio 3D.

permite la generación de un código de 8 bits a través del cual se puede determinar el mallado a aplicar a dicha zona [26].

En la biblioteca VTK desarrollada por el grupo *Kitware*[8] se encuentra *vtkMarchingCubes* el cual es un método que toma por entrada un volumen y se encarga de generar una salida de una o más mallas. Es necesario especificar uno o más valores de contorno para generar mallas superficiales, aunque alternativamente, es posible especificar un rango mínimo y máximo escalar y el número de contornos para generar una serie de valores de contorno equi-espaciados [26].

Marching Cubes permite la visualización de modelos complejos, este algoritmo permitió la reconstrucción del hígado utilizando la idea de calcular la malla superficial de la imagen tridimensional tomando grupos de ocho puntos vecinos formando cubos y después determinando los triángulos que representan la parte de la malla que pasa a través de dicho cubo [23].

4.3.3. Filtrado del mallado tridimensional

Durante la reconstrucción de órganos a partir de una imagen médica aparecen una serie de efectos perjudiciales que distorsionan o disminuyen la calidad del objeto resultante. Es por esto que es necesario usar algoritmos de suavizado para las diferentes superficies que permitan obtener modelos más realistas y mejor definidos.

Las operaciones de suavizado o filtrado reducen la información de alta frecuencia en la geometría de la malla. En caso de aplicar un suavizado excesivo es posible que se pierdan detalles importantes. El tipo de órgano (hígado) a reconstruir suele sufrir de zonas que pueden deformar el objeto original debido a la aparición de ejes erróneos o artificiales. Si se observan dos imágenes adyacentes, es muy probable que el mismo objeto difiera de una imagen a otra, en estas regiones es donde se debe suavizar el objeto con el fin de minimizar distorsiones de tipo escalera. A continuación se nombran algunos filtros con los cuales se hicieron pruebas para lograr un órgano menos “rugoso”.

Algoritmos de suavizado de Laplace: El filtro de Laplace es el algoritmo de suavizado de mallas más sencillo que existe. Mediante un análisis topológico de la malla se conforma un vecindario que tras unas pocas iteraciones del filtro de suavizado obtendrá resultados óptimos con pocas iteraciones. El inconveniente encontrado fue que repetidas veces encogía el objeto por lo que se vio la necesidad de introducir un factor de tamaño, con la finalidad de que este factor regulara el efecto de los vecinos directos al posicionarse dentro del vértice analizado, toda nueva posición depende de su anterior posición y de la posición de sus vecinos[22].

[8]**Kitware:** http://www.vtk.org/

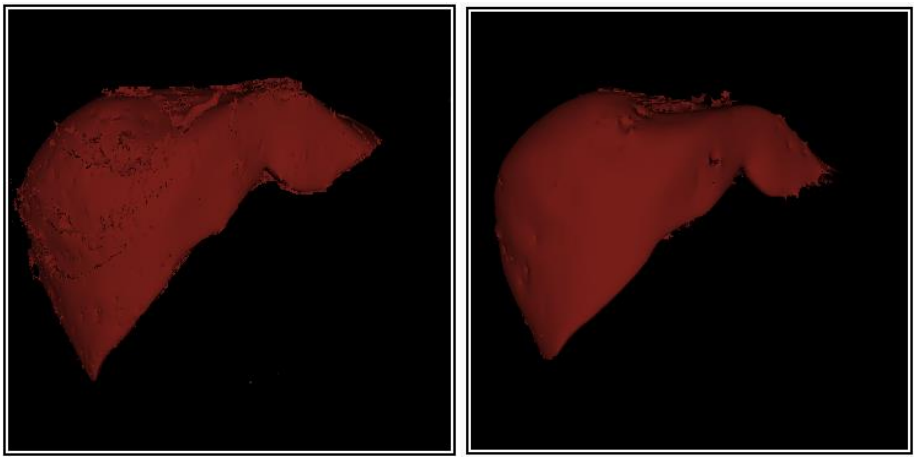
Algoritmos de suavizado basados del filtro pasa bajo: Los filtros paso bajo utilizados son la combinación de dos filtros similares a los Laplacianos, uno con un factor de expansión positivo y otro con un factor negativo ejecutado alternativamente. La diferencia encontrada con los filtros Laplacianos, es que este filtro incorpora un parámetro adicional [22].

En la biblioteca VTK, el equivalente al filtro paso bajo es *vtkWindowedSincPolyDataFilter*. Se encarga de ajustar las coordenadas de un punto mediante el uso de un *kernel* o máscara de convolución, utilizando parámetros tales como el factor de relajación que se encuentra entre los valores de 0 - 2 y el ángulo que corresponde a la disposición de los triángulos dispuestos en los cubos de la malla del objeto 3D, todos estos parámetros pueden ser definidos en la pestaña dispuesta para reconstrucción del órgano en la aplicación. La explicación de los parámetros se encuentra en la Tabla 7

Tabla 4: Parámetros filtros mallado tridimensional

Parámetro	Significado
Factor Relajación	El filtro iterativo de *Laplace* analiza todos los vértices del objeto y los desplaza hacia el centro geométrico de su vecindario topológico. Este procedimiento tiene el inconveniente de que encoge el objeto por lo que en ocasiones es común introducir un factor de tamaño. El objetivo de este factor es el de regular el efecto de los vecinos directos a la nueva posición del vértice analizado [22].
Angulo	Este término depende del factor de relajación puesto que éste define la nueva posición del vértice que depende de la anterior posición.

El resultado de este tipo de filtros es una relajación de la malla, consiguiendo unas celdas mejor definidas y los vértices más distribuidos. Este filtro opera con líneas, polígonos y grupos de triángulos mediante instancias a *vtkPolyData*. Los vértices de una célula no son nunca modificados. En la Figura 31, se observa la diferencia entre las dos imágenes en las cuales se aprecia la reconstrucción 3D del hígado, una sin filtrado del mallado y otra después del filtrado.



(a) (b)

Figura 31: Reconstrucción 3Da) La reconstrucción 3D Hígado sin filtros b) La reconstrucción 3D del hígado junto con el filtrado del mallado tridimensional.

4.4. ESCENARIO QUIRÚRGICO.

Para culminar con la última etapa del sistema propuesto se exporto el órgano debidamente reconstruido al escenario quirúrgico, el cual está compuesto de ciertas piezas construidas en su totalidad en *Solide Edges* exportadas a VTK por medio de *Blender* (ver Anexo B). En la Figura 32 se puede observar el escenario que fue diseñado con características básicas de una sala de cirugía real.

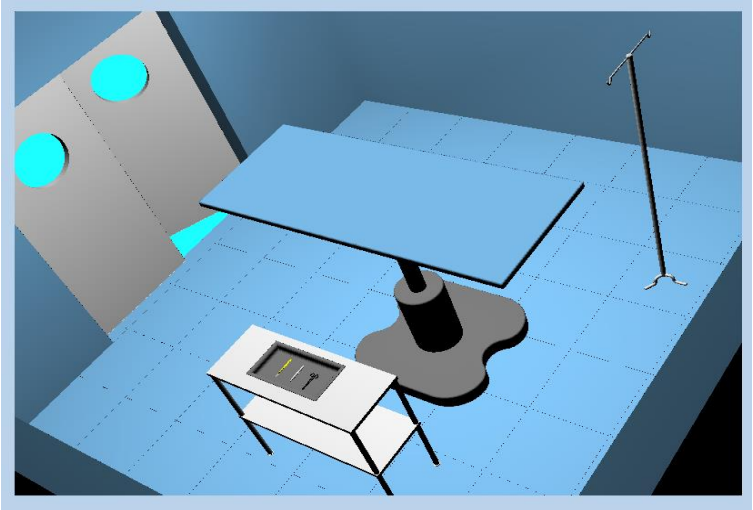


Figura 32: Escenario Quirúgico diseñado

Para aumentar el realismo y así aproximarse a eventos mucho más reales se procedió a implementar un sistema de colisiones adecuado para el tipo de escenario resultante. A continuación se encuentra la descripción de este sistema junto con el algoritmo para llevarlo a cabo esta actividad.

4.4.1. Sistema de colisiones

La detección de colisiones es un tema importante en la comunidad gráfica, tratar de detectar si dos o más objetos intersecan entre sí, en realidad es un problema algo complejo, puesto que además de saber si colisionan, es interesante conocer en qué punto entran en contacto los dos objetos. Existen aplicaciones, que requieren una respuesta lo más fiable físicamente posible. En el caso de los entornos de realidad virtual, y en concreto aquellos con interacción táctil del usuario con el mundo virtual, además de fiabilidad física es necesario que la simulación se realice en tiempo real.

El diseño de un sistema de detección de colisiones debe tener en cuenta diversos factores que influyen en su rendimiento. En primer lugar se debe considerar la representación del dominio, esto es, el formato en que los objetos gráficos están representados. Lo más común es el uso de una representación basada en triángulos, con la consecuente carencia de información explicita acerca del interior y el exterior del modelo [25]. La colisión entre dos objetos es muy costosa a nivel computacional, sobre todo cuando las estructuras a colisionar son complejas. Para minimizar este coste, se comprueba previamente la colisión existente, de tal manera que se realice una evaluación previa antes de determinar la colisión detallada entre objetos.

Para la aplicación desarrollada en este proyecto es necesario conocer la colisión entre objetos, más precisamente la colisión presentada entre órgano y herramienta quirúrgica virtual, que permita obtener un mayor realismo y posteriormente realizar proyectos que utilicen deformación de órganos.

A continuación se explicará el método adecuado para observar el efecto de la colisión entre objetos y sus diferentes tipos.

- **Volúmenes envolventes**: una primera estrategia para acelerar este proceso es usar volúmenes envolventes, descartando de forma rápida los casos de no-colisión. La idea es simple: tener como volumen envolvente una zona del espacio lo suficientemente simple como para realizar evaluación de intersección de una forma muy rápida. Estas zonas del espacio van desde esferas hasta las envolventes más complicadas. Cuanto más ajustada es la envolvente menos falsos positivos se producen, pero es mayor el tiempo y el espacio de almacenamiento requerido para realizar la consulta [25].

La idea consiste en utilizar una evaluación de colisión entre el volumen envolvente, menos costoso que la propia evaluación de colisión entre objetos. Esta técnica establece que el número de intersecciones entre volúmenes envolventes, con una relación de aspecto y factor de escala constantes, es casi el mismo que el número de intersecciones entre objetos [24]. Los volúmenes envolventes más representativos son las esferas envolventes, las cajas envolventes alineadas con los ejes, las cajas envolventes orientadas y los k-DOPs. A continuación se describen los volúmenes envolventes más representativos.

En la
Figura 33 se observa los distintos tipos de volúmenes envolventes existentes

Figura 33: Tipos de volúmenes envolventes.

Fuente: Modulo de Navegación para un Sistema de Entrenamiento Virtual Aplicado a Cirugía de la Base del Cráneo [9]

- **Esfera envolvente:** requieren la consulta más sencilla de todos los volúmenes envolventes, ya que para dos objetos, sus esferas envolventes, intersecan si la distancia entre sus centros es menor que la suma de sus radios [25]. Además, tiene la ventaja de que es invariante a la rotación. El inconveniente de este tipo de volumen envolvente consiste en que no se ajusta a determinados objetos, siendo difícil obtener la esfera que mejor se ajuste a un objeto dado [24].

- **Caja envolvente orientada:** también conocida como OBB (*Oriented Bounding Box*), este tipo de volumen envolvente es representado por una caja rectangular con una orientación arbitraria. Al contrario que las AABB, éste utiliza el teorema del eje separador, que mejora el ajuste a los objetos y su rápida actualización en las rotaciones. Alguno de sus inconvenientes es la utilización de mayor espacio para guardar mucha más información para representar este volumen, normalmente la forma del centro de la caja, tres longitudes y una matriz de rotación [25].

Existen diferentes bibliotecas que implementan métodos de detección de colisiones entre ellas se encuentran, *I-Collide*, biblioteca de detección de colisión para grandes entornos, utiliza un método de dos fases con cajas envolventes y determina la colisión exacta entre pares de objetos poliédricos convexos [24];*Rapid*, biblioteca para la detección de colisiones entre polígonos no estructurados. Y por último *V-Collide* biblioteca de detección de colisiones desarrollada por el grupo GAMMA en la Universidad de Carolina del Norte explicada con anterioridad en el capítulo 3.

En la aplicación diseñada se pueden determinar los pares de objetos que se encuentran potencialmente en contacto. Estos objetos son utilizados por la librería *Rapid* para la detección de la colisión. *V-Collide*, puesto que está diseñada para operar en ambientes que contienen objetos en el escenario formados por mallas de triángulos, realizando la detección de colisiones entre parejas de objetos a través de dos etapas.

1. La primera etapa consiste en la construcción OBB(Caja Envolvente Orientada) para cada objeto, con el fin de encontrar parejas de triángulos que posiblemente intercepten los OBBs.

2. La segunda etapa consiste en verificar si las parejas de triángulos detectadas en la etapa 1 realmente se interceptan.

5. ESTRUCTURA DE LA APLICACIÓN DESARROLLADA JUNTO CON RESULTADOS, PRUEBAS E INTEGRACIÓN DE SUS COMPONENTES.

Este capítulo describe el diseño de la aplicación desarrollada a nivel interno y su progreso basado en los requerimientos funcionales junto con los resultados y pruebas ejecutadas. Además se explicará brevemente las clases junto con los métodos para el desarrollo del programa, para concluir con diagramas UML que permitan al lector entender con mayor facilidad la estructura de la aplicación presentada.

5.1. CLASES CREADAS EN EL PROYECTO

En este apartado se describirán las principales clases creadas dentro del proyecto de forma general y algunos de las librerías utilizadas. Para más información sobre cada clase se recomienda utilizar el propio código fuente.

5.1.1. Clase cargar imagen

La función principal de esta clase es la de cargar imágenes médicas desde un espacio de memoria. La clase cargar imagen maneja diferentes objetos, útiles en la segmentación de imágenes médicas. Toda esta parte fue trabajada con la ayuda de la biblioteca ITK. A continuación en la Tabla 8 se mencionan algunas librerías utilizadas junto con su descripción.

Tabla 5: Métodos clase cargar imágenes

LIBRERÍAS	DESCRIPCIÓN
- **itkImageSeriesReader**	Librería encargada de leer datos de una serie de imágenes que se encuentran en el disco. Esta librería se utilizó para construir una imagen de n-dimensiones a partir de múltiples archivos n-1 dimensiones, para así ser guardados en un vector que se lee con el método **ImageFileReader**.
- **itkGDCMSeriesFileNames**	Esta librería se incluyó puesto que las imágenes médicas utilizadas se encontraban en formato DICOM en una secuencia de archivos. El proceso comienza cuando se lee todas las imágenes o una serie desde un punto en el directorio, luego se verifica su

	orientación, identificando las coordenadas exactas para poder construir el tramo o la serie en 3D, al final los archivos quedarán ordenados por orden lexicográfico[9]
- **itkGDCMImageIO**	Esta librería se utiliza para leer y escribir DICOM V3.0 y ACR / NEMA. Esta librería sirve como adaptador a la biblioteca GDCM anteriormente explicada.

5.1.2. Clase reconstructor

Esta clase se encarga de Reconstruir un órgano a partir de imágenes médicas cargadas. La clase reconstructor trabajada con la ayuda de las bibliotecas ITK y VTK. A continuación en la Tabla 6 se mencionan algunas librerías de ITK utilizados junto con su descripción.

Tabla 6: Métodos de ITK para la clase reconstructor

LIBRERÍAS	DESCRIPCIÓN
- **itkVTKImageExport**	Conecta objetos de ITK con VTK. **VTKImageExport** se utilizó para exportar imágenes de VTK a ITK. A través de solicitudes de actualización que vienen de VTK y se propagan automáticamente a ITK.
- **itkSigmoidImageFilter**	Librería utilizada como parte de los filtros aplicados a las imágenes.
- **itkCurvatureFlowImageFilter**	Esta librería fue utilizada como filtro para eliminar ruido de las imágenes como brillos o

[9] Lexicográfico: Relación de orden que se utiliza para ordenar el producto cartesiano de conjuntos ordenados

	contornos innecesarios.
- **itkConnectedThresholdImageFilter**	Librería utilizada para el control de los puntos semillas tanto de la posición inicial de éste, como su conexión con los demás píxeles.

A continuación en la Tabla 7, se mencionan algunas librerías de VTK utilizados junto con su descripción.

Tabla 7: Librerías de VTK para la clase reconstructor

LIBRERÍAS	DESCRIPCIÓN
- **vtkImageImport**	Esta librería fue utilizada para importar datos, **vtkImageImport** proporciona los métodos necesarios para importar datos de la imagen de una fuente independiente de VTK, tales como una matriz.
- **vtkImageViewer2**	Útil para trabajar con datos 2D posibilitando utilizar un visor de imágenes específico para la visualización.
- **vtkInteractorStyleUser**	Esta librería permite conectar el mouse u otro hardware para el cambio de eventos dentro de la aplicación.
- **vtkRenderWindowInteractor**	Método utilizado para poder interactuar con los datos de la escena. La clase **vtkRenderWindowInteractor** es capaz de soportar diferentes estilos de interacción. Los diferentes estilos observarán diferentes eventos y actuarán en consecuencia ejecutando la acción correspondiente. Siempre que se llama a la función **vtkRenderWindowInteractor** se instancia de forma automática a un inter actor apropiado al sistema para el que se esté trabajando.

- **vtkRenderWindow**	La **RenderWindow** es la ventana del ordenador donde el render crea el objeto. Es utilizada para dibujar las figuras 3D, de esta forma instanciar de forma automática las diferentes subclases mostradas en la plataforma de trabajo.
- **vtkRenderer.**	Es utilizada junto con las **vtkRenderWindow** para dirigir la interfaz entre la “máquina gráfica” y el sistema de ventanas del computador.
- **vtkCamera**	Capta la representación de la geometría 3D en una imagen 2D. Las cámaras están estrechamente relacionadas con el proceso controlando la posición y orientación.
- **vtkInteractorStyleImage**	La librería **vtkInteractorStyleImage** permite manipular interactivamente la cámara del escenario (girar, desplazar, zoom, etc.)
- **vtkEventQtSlotConnect.**	Librería utilizada para gestionar las conexiones entre eventos de VTK y los *slots* de QT.
- **vtkProperty**	Librería utilizada para la representación y modificación de propiedades de la superficie de un objeto geométrico, tales como iluminación, color, etc.
- **vtkOBJExporter**	Librería utilizada para exportar archivos de tipo.obj. Las herramientas quirúrgicas creadas para la aplicación se construyeron en software de tipo CAD. (*Solid Edge*), las cuales se exportan al programa multiplataforma de moldeamiento y animaciones 3D (*Blender*) el cual permitió finalmente exportar

	archivos compatibles de VTK.
- **vtkImageData**	Mediante este objeto **vtkImageData** es posible representar imágenes en 1, 2 o 3 dimensiones. **VtkImageData** es una subclase de **vtkDataSet** lo que permitió representar mediante un actor con un **vtkDataSetMapper**.

5.1.3. Clase interfaz usuario

Esta clase es la encargada de gestionar todos los recursos que posee el programa y brinda las herramientas para interactuar con la aplicación. Esta clase incluye las clases: cargar imágenes, barra de estado, reconstructor y también clases de QT. Otras como clase tijera, clase lápiz, clase sistema de colisión y clase objeto actor.

A continuación en la Tabla 8 se mencionan algunas de las librerías que hacen parte de la clase.

Tabla 8: Librerías de la clase interfaz de usuario

LIBRERÍAS	DESCRIPCIÓN
- **QtGui**	Componentes de la interfaz gráfica de usuario
- **QApplication**	La clase QApplication administra el flujo de la aplicación con la interfaz gráfica de control y ajustes principales. Contiene los eventos principales, donde el sistema de ventanas y otras fuentes son procesados y enviados, además Se ocupa de la inicialización aplicación y finalización, y proporciona la administración de sesiones. También se ocupa de configuración de la mayoría de todo el sistema y toda la aplicación.
- **vtkInteractorStyle**	Proporciona la interfaz orientada a cada uno de los eventos que pueden existir en la ventana (Define el modo

	TrackBall). Esta librería mantiene una relación de herencia con **vtkInteractorObserver**.
- **vtkInteractorStyleUser**	Esta librería mantiene una relación de herencia con **vtkInteractorStyle.** Permite personalizar las interacciones del usuario con la aplicación, es útil a la hora de conectar los movimientos del *mouse* u otro *hardware* para el cambio de eventos dentro de la aplicación.
- **vtkEventQtSlotConnect**	Gestiona las conexiones entre eventos VTK y los *Slots* de QT. Esta librería mantiene una relación de herencia con **vtkObject.**
- **vtkInteractorStyleTrackballCamera**	Esta librería mantiene una relación de herencia con**vtkInteractorStyle.** Se encarga de la manipulación interactiva de la cámara.

5.1.4. Clase barra estado

Esta es una clase amiga que hace parte de cada una de las demás clases. Los métodos definidos en esta despliegan información para las demás clases. También contiene QtGui (componentes de la interfaz gráfica de usuario).

5.1.5. Clase objeto actor

Esta clase carga figuras en formato .obj en un actor que hará parte de una escena. A continuación en la Tabla 9 se mencionan algunas librerías de VTK utilizadas junto con su descripción.

Tabla 9: Librerías de la clase objeto actor

LIBRERÍAS	DESCRIPCIÓN
- **vtkPolyDataMapper**	Representa un conjunto de datos como líneas, polígonos y triángulos, específicamente figuras primitivas.

- **vtkActor**	Representa un objeto de una escena renderizada. Hereda las funciones relacionadas con la posición de los actores y la orientación de **vtkProp**.
- **vtkOBJReader**	Sirve para leer archivos .Obj
- **vtkAlgorithmOutput**	Sirve para conectar puertos de entrada y salida.

5.1.6. Clase objeto 3D

Esta clase representa todas las herramientas 3D visualizadas dentro del entorno y sus acciones bajo éste. Además incluye a la clase sistema de colisiones y a la clase objeto actor. A continuación en la Tabla 10 se mencionan algunas librerías utilizadas junto con su descripción.

Tabla 10: Librerías clase herramienta 3D

LIBRERÍAS	DESCRIPCIÓN
- **vtkAssembly**	Hereda las funciones relacionadas **vtkprod3D**.
- **vtkCellArray**	Es un objeto de apoyo que explícitamente representa la conectividad de una estructura de datos, útil a la hora de guardar datos compactamente y de forma sencilla. Estas estructuras generalmente están formadas por números enteros los cuales representan los puntos de una celda.
- **vtkTransform**	Describe transformaciones lineales a través de una matriz 4x4. Lleva a cabo todas las operaciones de un sistema de coordenadas para la rotación de los diferentes objetos.
- **vtktMatrix4x4**	Representa y manipula las matrices 4x4 de transformación. En concreto trabaja con matrices que se encuentran en 3D usando coordenadas homogéneas [XYZW]

5.1.7. Clase tijera, lápiz y bisturí

Estas clases contienen herramientas y heredan todas las características de la clase herramienta 3D.

5.1.8. Clase sistema de colisiones

Clase encargada de detectar las colisiones presentes entre cada una de las herramientas quirúrgicas y el órgano 3D, a través de librerías como *V-Collide*. A continuación en la Tabla 11 se mencionan algunas librerías utilizadas junto con su descripción.

Tabla 11: Librerías clase sistema de colisiones 3D

LIBRERÍAS	DESCRIPCIÓN
- **vtkPolyData**	Representa un conjunto de datos como líneas, polígonos y triángulos.
- **vtkTriangleFilter**	Genera triángulos sobre un objeto 3D, este filtro también pasa por cada vértice y línea de ser necesario.

5.2. DIAGRAMAS UML

En esta parte se construyeron algunas vistas de diagramas UML con el fin de entender más a fondo la estructura de la aplicación creada. UML es una especificación de notación orientada a objetos [27]. Es un lenguaje estándar diseñado para visualizar, especificar, construir y documentar software orientado a objetos [28]. UML estandariza nueve tipos de diagramas para representar gráficamente un sistema desde distintos puntos de vista.

Los puntos de vista expuestos en este apartado serán:

Diagrama De Casos De Uso: Describe las interacciones del sistema con su entorno, identificando los actores que representan los diferentes roles de los usuarios del sistema y los casos de uso que corresponde a las diferentes funcionalidades que el sistema ofrece. Se puede identificar dentro del diagrama de caso de uso los siguientes componentes [29].

- **Actor**: Es el rol de un objeto al exterior del sistema, que interactúan directamente con él como parte de una unidad coherente de trabajo. Caracteriza el rol desempeñado por un objeto externo [29].

- **Caso de Uso:** Es una unidad coherente de funcionalidad suministrada por un sistema o una clase, se manifiesta a través de una secuencia de mensajes intercambiados entre el sistema y un actor o actores externos, junto con las acciones realizadas por el sistema [29].

A continuación se observan diferentes figuras las cuales hacen parte de los diagramas de uso de las principales clases de la aplicación. En la Figura 34 se indica el diagrama de usos para realizar el proceso de cargar las imágenes, esta función se realiza mediante un asistente o *wizard* que se explicará más adelante.

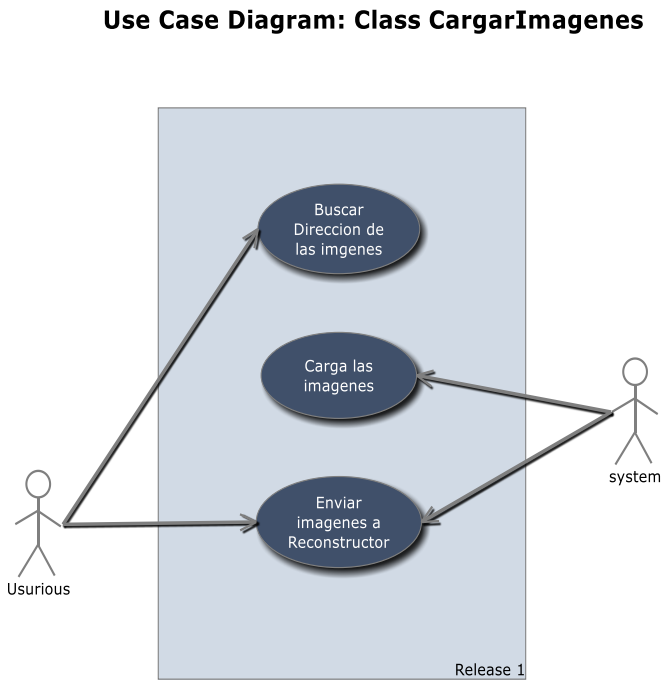


Figura 34: Diagrama Caso de uso – Clase Cargar Imágenes

En la Figura 35 se presenta el diagrama de usos para la clase reconstructor la cual inicia con la definición del método de segmentación y posteriormente con la señalización de los puntos semillas para concluir con la reconstrucción en 3D del órgano (hígado).

En la Figura 36 se indica el diagrama de usos dispuesto para el escenario quirúrgico el cual muestra las actividades que se puede realizar dentro del mismo.

En la Figura 37 presenta el sistema de colisiones, el cual comienza cuando se escoge una de las herramientas quirúrgicas y posteriormente son desplazadas hacia el objeto que deseamos colisionar.

Use Case Diagram: Class Reconstructor

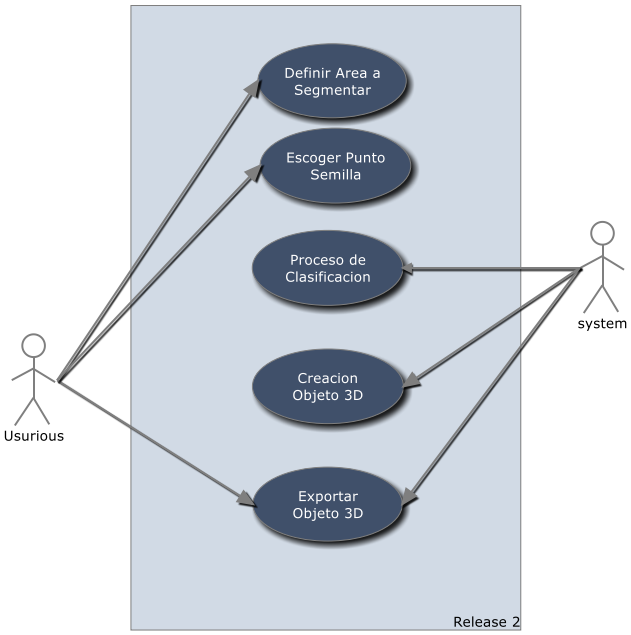


Figura 35: Diagrama Caso de uso – Clase Reconstructor

Use Case Diagram: Class Simulador

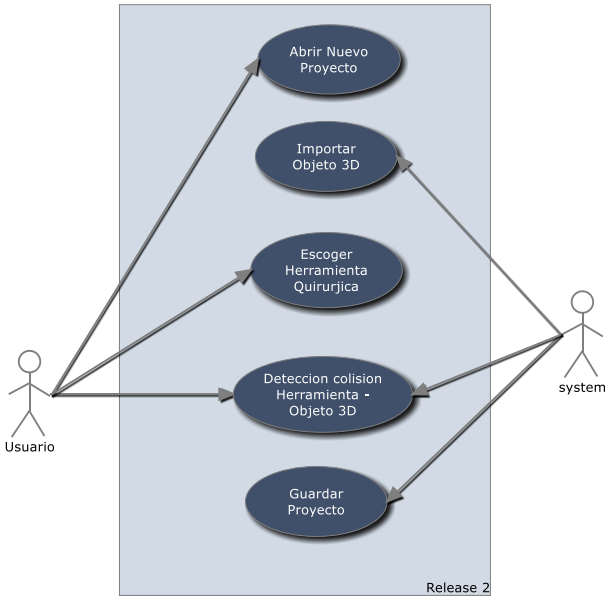


Figura 36: Diagrama Caso de uso – Clase Simulador

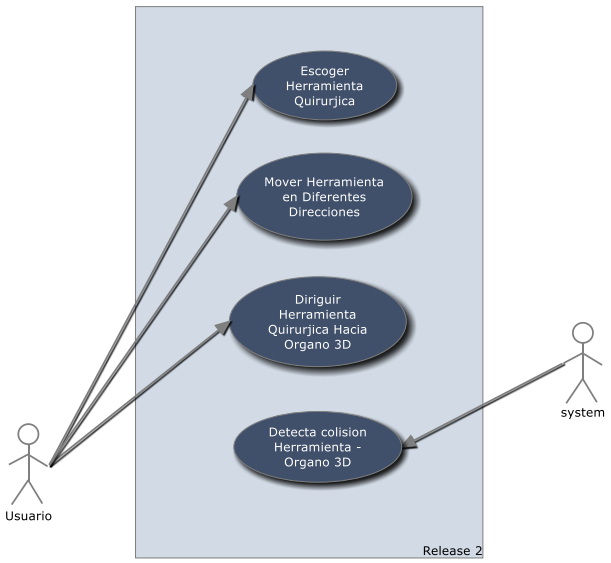


Figura 37: Diagrama Caso de uso – Clase Sistema Colisiones

Diagrama de clases y de relaciones: este diagrama sirve para visualizar las relaciones entre las clases que involucran un sistema. Representan un conjunto de elementos que son estáticos, como las clases, su contenido, y las relaciones que se establecen entre ellos [29].

Una clase está representada por un rectángulo que dispone de tres apartados, el primero para indicar el nombre, el segundo para los atributos y el tercero para los métodos. Cada clase debe tener un nombre único, que las diferencie de las otras. Se puede observar a continuación en el diagrama de clases de la Figura 38 donde se definen algunas características de cada una de las clases y sus relaciones de dependencia y generalización.

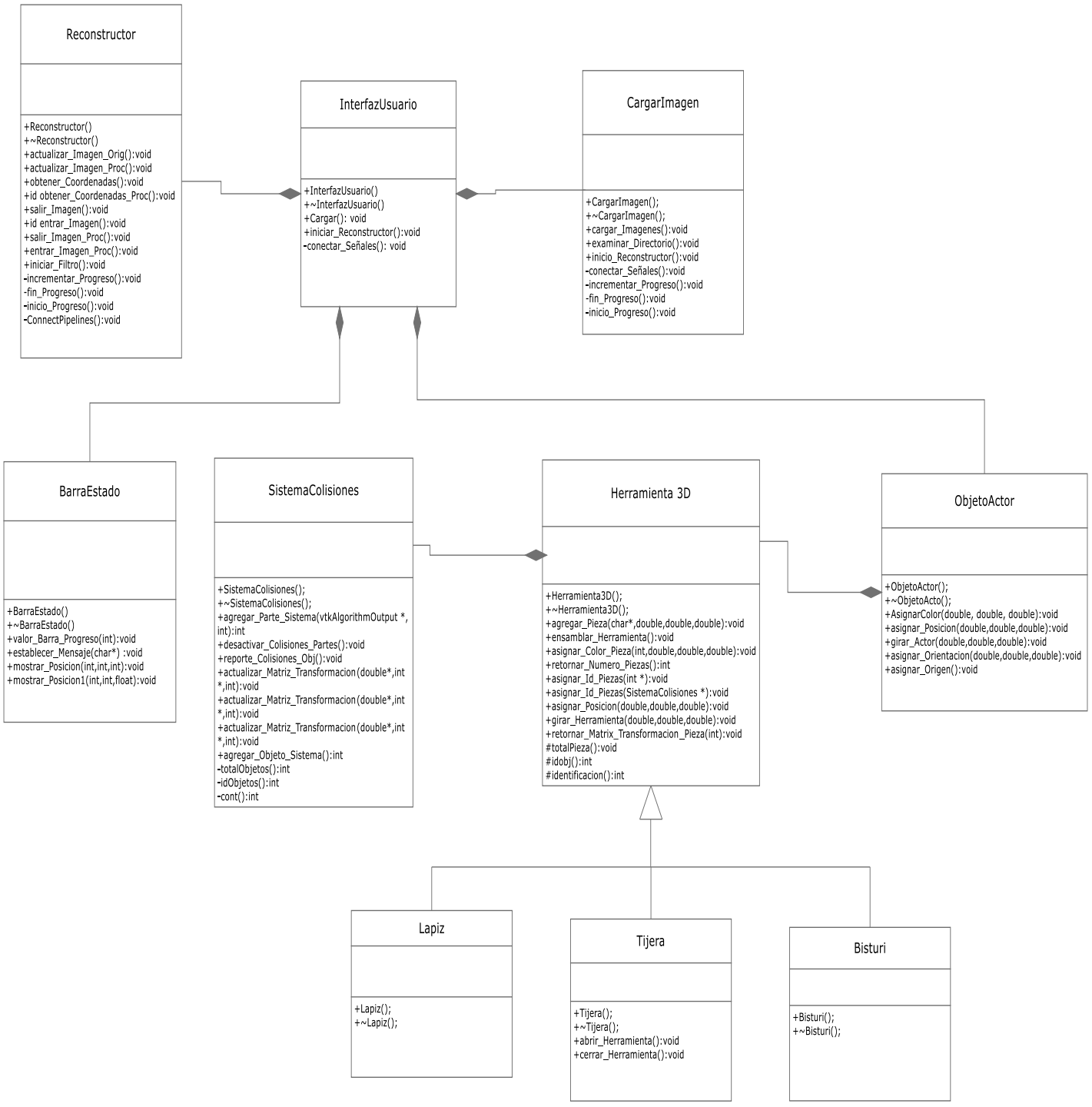


Figura 38: Diagrama de clases y de relaciones

5.3. RESULTADOS Y PRUEBAS

En este apartado se presentará la funcionalidad y resultados de las aplicaciones desarrolladas paso a paso tal como se concibió la solución al problema planteado en anteriores capítulos (sistema propuesto). Para información sobre las diferentes funciones de la aplicación diríjase al Anexo C: Manual de Usuario.

5.3.1. Pre-procesamiento

Como ya se anotó con anterioridad el pre-procesamiento de las imágenes médicas comienza con la carga de éstas a la aplicación desarrollada, estas imágenes deben estar alojadas en algún punto del computador, como se anotó en capítulos anteriores. Estas imágenes se obtuvieron de una base de imágenes gratuita[10] de internet que cuanta con distintas imágenes de zonas del cuerpo.

Para la función de carga se creó un *wizard* o asistente diseñado en *Qt Designer* que ayude al usuario a realizar este proceso. Para iniciar con el asistente es necesario dar clic en el ícono de nuevo que aparece en la Figura 39 o dando clic sobre la barra de menú archivo nuevo de la aplicación desarrollada.

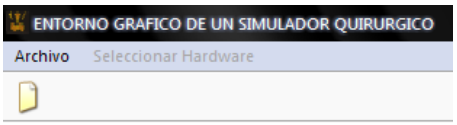


Figura 39: Barra de Iconos

Tras dar clic en el ícono de Nuevo se desplegará la imagen de la Figura 40, la cual muestra el inicio junto con el mensaje de bienvenida del asistente o w*izard* para cargar imágenes médicas desde cualquier ruta en el computador. Un *wizard* en inglés, es una aplicación al servicio del usuario que generalmente abrevia los pasos a seguir para realizar una tarea. Los asistentes hacen más sencillas las tareas de instalar dispositivos, programas o realizar alguna actividad.

Figura 40: Inicio asistente cargar imágenes.

[10] http://pubimage.hcuge.ch:8080/

El asistente guiará al usuario dentro de la primera actividad importante de la aplicación final y de esta manera cumplir con la primera tarea del pre-procesamiento. En la sección de carga de archivos, se encuentran los diferentes *combo box* diseñados en *Qt Designer* (ver Figura 41). En el primer *combo box* titulado 'Qué desea cargar', se encuentran dos opciones como se aprecia en la Figura 42. En éste se pide elegir entre cargar las imágenes médicas o cargar un órgano con anterioridad guardado con extensión .vtk

Figura 41: Sección de carga de archivos

Figura 42: *Combo box* 1

Si dentro de este *combo box* se seleccionó un Órgano Reconstruido los demás *combo box* se deshabilitarán quedando solamente el espacio donde se escribe la ruta del archivo u órgano a seleccionar. Después de escoger la ubicación del archivo en una carpeta del computador en este caso de un archivo con extensión .vtk puesto que corresponde a un objeto reconstruido, se procede a dar clic en la palabra siguiente, es cuando la aplicación empezará a cargar el órgano reconstruido con anterioridad. Tras dar término a este proceso se desplegará una ventana donde se apreciará el órgano, como se observa en la siguiente Figura 43.

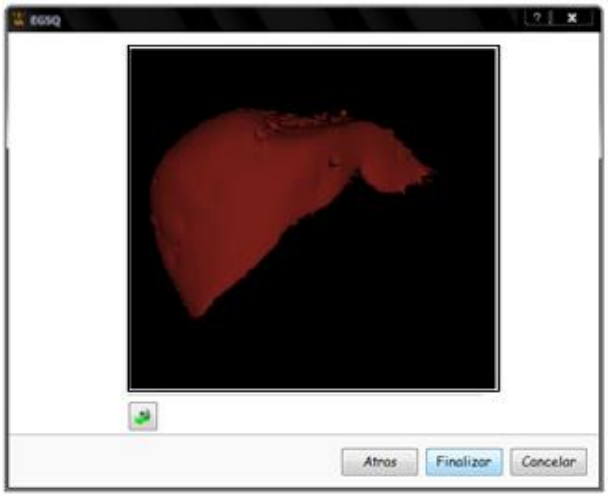


Figura 43: Órgano reconstruido

Si la opción seleccionada es Imágenes Médicas el siguiente *combo box* por elegir será el mostrado en la Figura 44, el cual contiene dos opciones Archivo Único y Serie de Imágenes, si la opción elegida es archivo único quiere decir que las imágenes se encuentran en un archivo comprimido que generalmente es de extensión (.mha), si la opción es una serie de imágenes quiere decir que las imágenes no se encuentran comprimidas y son varias y se pueden encontrar en diferentes formatos de visualización de imágenes comunes tales como: .jpg - .bmp - .png y formatos DICOM como .dcm

Figura 44: *Combo box 2*

Después de seleccionar la opción deseada en cada uno de los *combo box*. Se procede a dar clic en 'Examinar' para seleccionar la ubicación del archivo que se desea cargar tal como se muestra en la Figura 45.

Figura 45: Barra de proceso de carga de archivos

Luego que la aplicación termina este proceso de carga se desplegará la ventana mostrada en la Figura 46 donde observamos cada una de las imágenes cargadas desplazando la barra deslizante que se encuentra debajo de las imágenes. En la parte derecha se encuentra la información de los archivos cargados tales como resolución de la imagen en píxeles, número total de imágenes cargadas, tamaño real de las imágenes en mm, y el espaciado de cada una de las imágenes. En esta parte es posible ajustar el espaciado (mm) entre las imágenes en las posiciones X, Y, Z valor que se actualiza tras dar clic en 'Actualizar Espaciado'. Este valor es necesario para la reconstrucción final del órgano ya que se debe saber la posición de cada corte en el espacio.

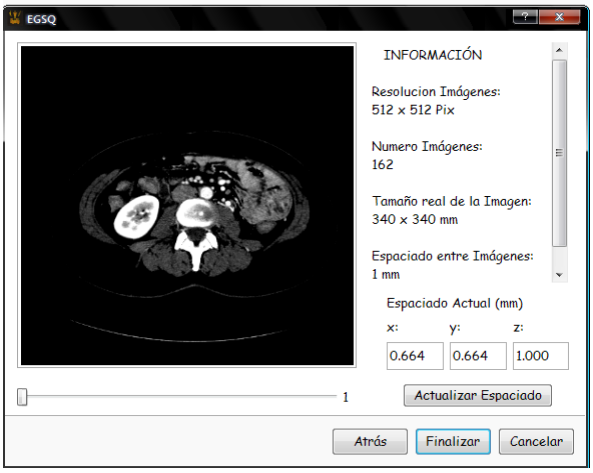


Figura 46: Información y visualización de las imágenes

Para mayor información acerca del funcionamiento de la aplicación diríjase al Anexo C Manual de Usuario.

Después de cargar las imágenes y continuando con el pre-procesamiento como ya se planteó con anterioridad, se pueden aplicar diferentes tipos de filtros espaciales y además transformaciones geométricas sobre la imagen (ver Figura 47).

Figura 47: Pre-procesamiento

En el lado derecho de la pantalla principal de pre-procesamiento (ver Figura 47) se observan las imágenes cargadas, en esta parte la aplicación es capaz de realizar una transformación geométrica (recortar) sobre la imagen señalando la misma al tamaño deseado y posteriormente dando clic en el botón de recortar. Para la visualización de cada imagen se desplaza de arriba abajo la barra que aparece en la parte izquierda, se observan también, dos botones uno de recortar y otro rehacer la imagen original. La otra parte de la ventana consta de una pestaña que contiene los distintos filtros espaciales que se pueden aplicar. En esta parte el usuario puede cargar diferentes filtros y automáticamente se cargará la máscara de convolución o (*kernel*). El usuario también puede ingresar la máscara de convolución o (*kernel*) como lo requiera.

Como ya es de suponer los resultados de la parte del pre-procesamiento es variado puesto que el usuario es libre de escoger el filtro espacial al acomodo de sus necesidades. A continuación se muestra la respuesta de la aplicación a algunos de ellos. Los filtros dispuestos en la aplicación se observan en la Figura 48.

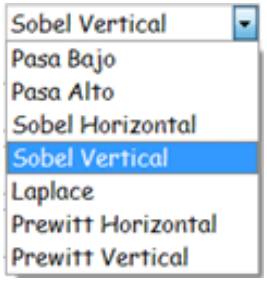
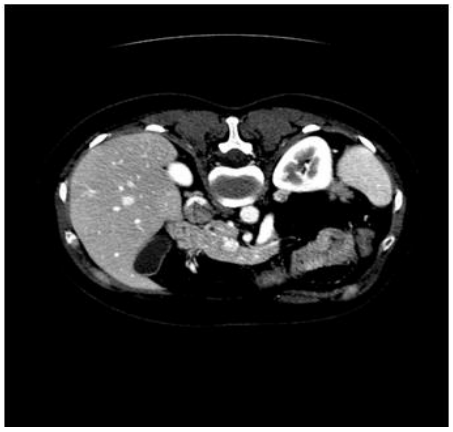
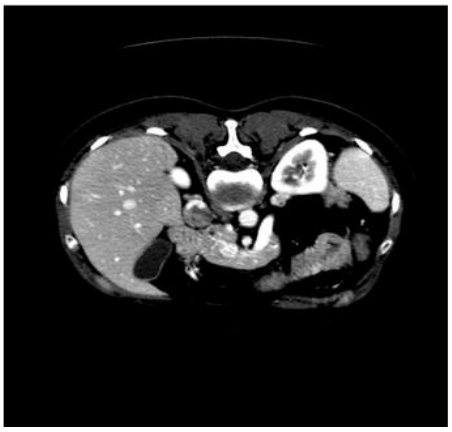


Figura 48: Tipos de filtros

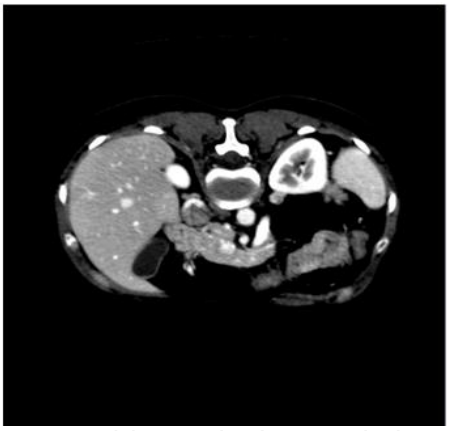
En la Figura 49, se observan distintas imágenes donde se encuentran los resultados de algunos filtros espaciales aplicados: a) Imagen original, b) Resultado pasa alto, c) Resultado pasa bajo, d) *Sobel* horizontal, e) *Laplace*, f) *Prewitt* vertical.



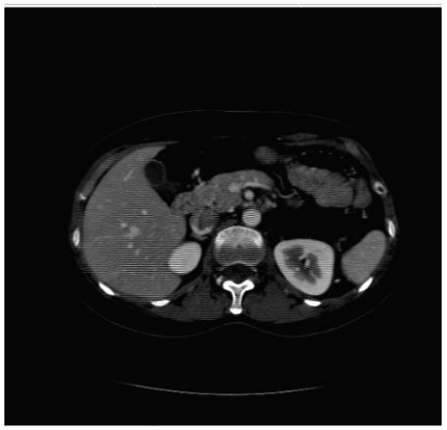
(a) Imagen original



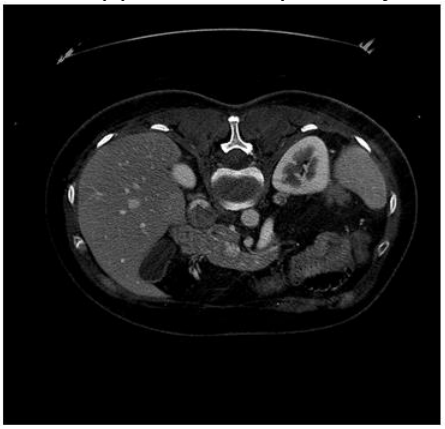
(b) Resultado paso alto



(c) Resultado paso bajo



(d) *Sobel* horizontal



(e) *Laplace*

(f) *Prewitt* vertical

Figura 49: Resultados de filtros espaciales

Los anteriores resultados son variados y su utilidad depende del órgano que se desee reconstruir posteriormente, en este caso como el órgano reconstruido fue el hígado el mejor resultado visualmente es la aplicación de un filtro pasa bajo, el cual realzó ciertas características ideales para seguir con el proceso de segmentación. Además observamos que los filtros como *Laplace* y *Prewitt*re resaltan características tales como el color blanco de las imágenes, este tipo de filtros sería ideal para reconstruir estructuras óseas.

5.3.2. Segmentación

Luego de realizar el pre-procesamiento de las imágenes se pueden aplicar los distintos métodos de segmentación siguiendo la secuencia como se dispuso en la aplicación desarrollada. La ventana de segmentación y reconstrucción se diseñó de tal manera que el usuario observe en cada sub-ventana como cada proceso va ocurriendo como se observa en la Figura 50. En el lado izquierdo se encuentran las imágenes pre-procesadas, en el centro de la ventana se dispuso como una pestaña los distintos métodos de segmentación y de reconstrucción. En el cuadro inferior central se observarán las imágenes segmentadas y en el recuadro de la parte derecha se encuentra el órgano reconstruido.

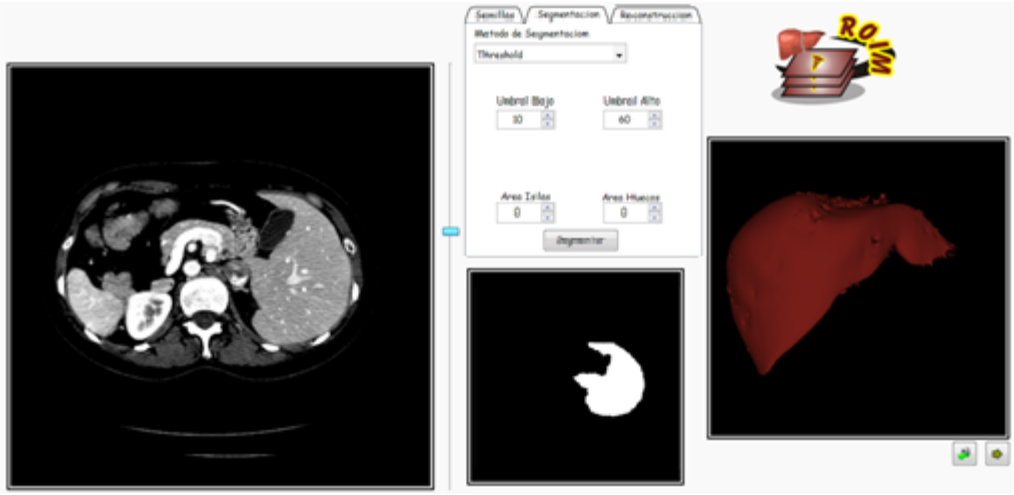


Figura 50: Visualización de métodos de segmentación

Los métodos de segmentación que se pueden realizar son variados y arrojan distintos resultados y dependen de los parámetros ingresados. Siempre es necesario señalar los puntos semilla dentro de la imagen para que los distintos métodos funcionen, algunos de los cuales se presentan a continuación.

Método *Threshold*: los parámetros ingresados con los cuales se obtuvieron mejores resultados se muestran en la Figura 51. Estos resultados demuestran una segmentación muy básica y no muy óptima ya que los niveles de gris del hígado se confunden con el de otros órganos.

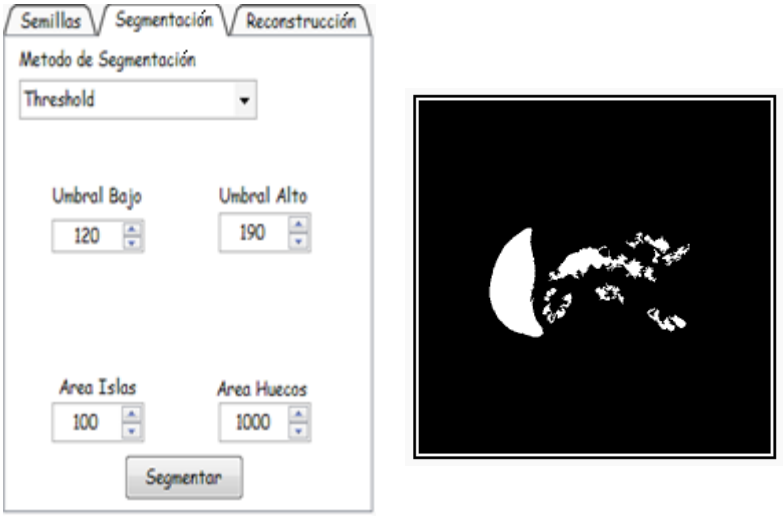


Figura 51: Parámetros método *threshold*

- **Método *confidence connected*:** el método se basa en simples estadísticas de la región actual. En primer lugar, el algoritmo calcula la media y desviación estándar de valores de intensidad para todos los píxeles incluidos actualmente en la región.

Los parámetros ingresados que arrojan mejores resultados se observan en la

Figura 52.

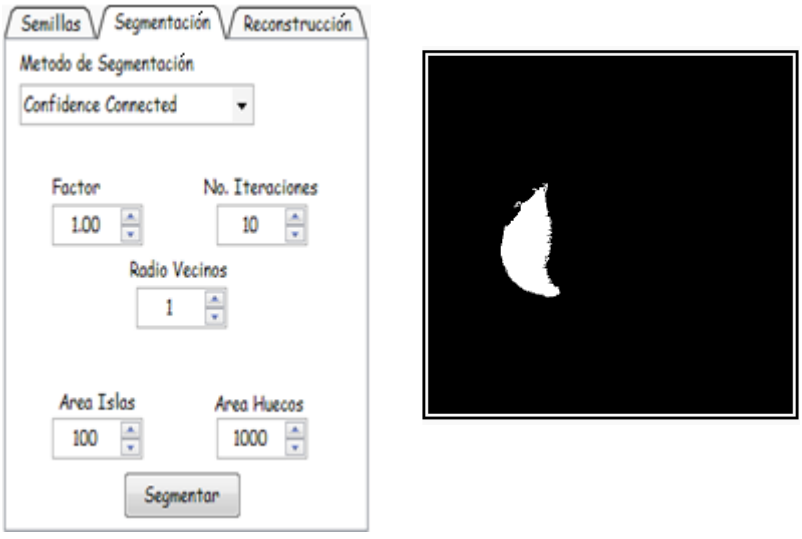


Figura 52: Parámetros método *confidence connected*

Método *neighborhood connected*: los parámetros ingresados para obtener mejores resultados se muestran en la Figura 53.

Figura 53: Parámetros método *neighborhood connected*

Método *level set*: los parámetros ingresados que arrojaron resultados óptimos se observan en la Figura 54.

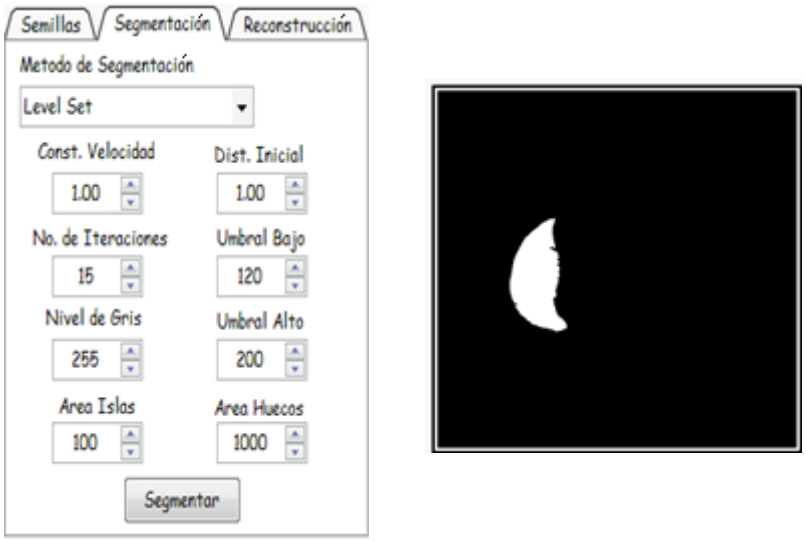


Figura 54: Parámetros método *level set*

Los resultados dispuestos evidencian la variabilidad dependiendo el método que se utilice, como se anotó con anterioridad depende del órgano que se segmente. El método *level set* es uno de los más complejos y por lo tanto óptimo como se observa en la imagen (ver figura 54).

5.3.2. Reconstrucción

Luego de un proceso de binarización de las imágenes (segmentación) se procedió a construir un mallado superficial, donde se aplica el algoritmo *marching cubes* explicado en el capítulo 4. Este método permitió alcanzar una reconstrucción del hígado óptima tal como se presenta a continuación en la Figura 55. Aquí se pueden evidenciar las distintas vistas de un órgano reconstruido a partir de parámetros definidos en las pestañas de reconstrucción que se observan en la Figura 56.

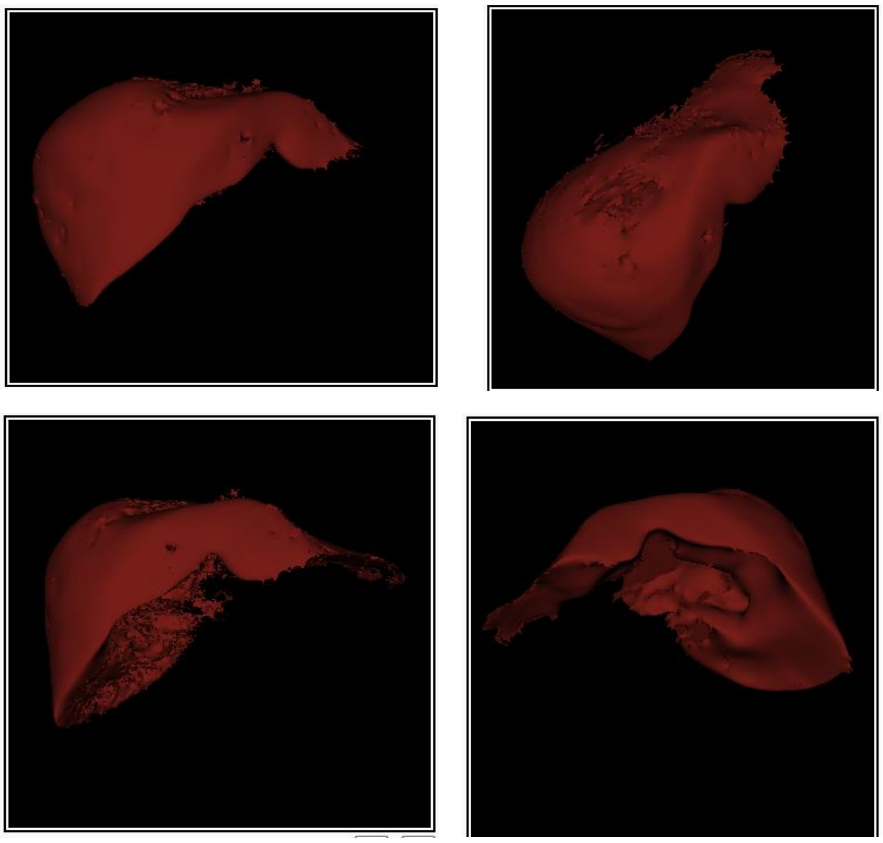


Figura 55: Vistas de la reconstrucción del hígado

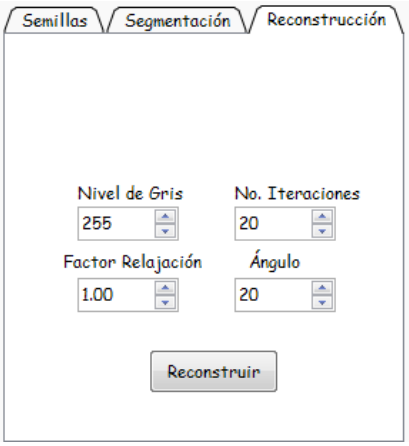
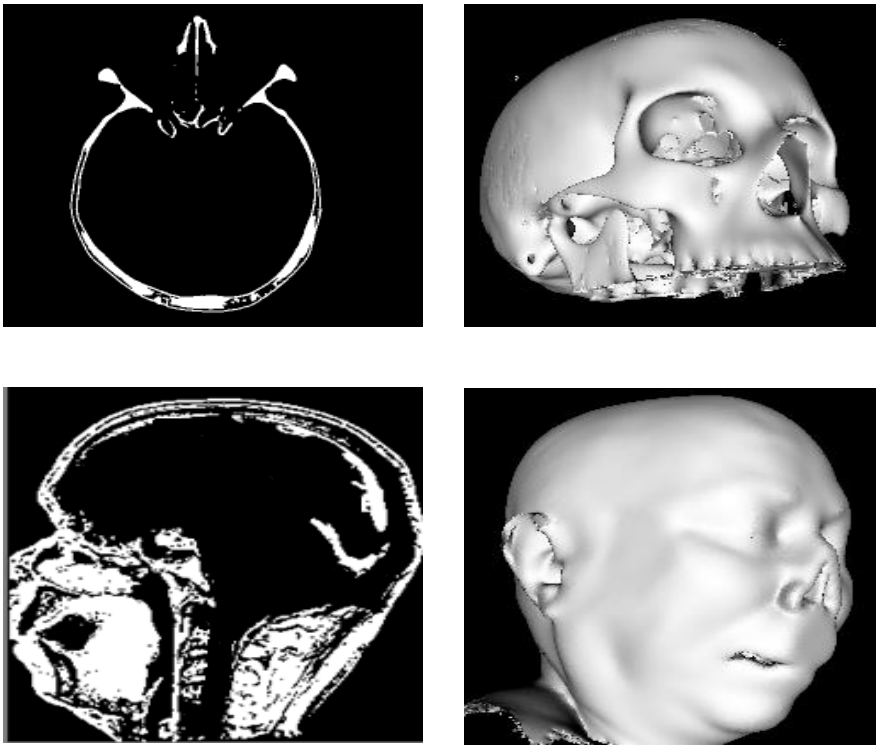


Figura 56: Pestaña de reconstrucción

- **Otras pruebas de reconstrucción**: Aquí se puede evidenciar que la aplicación desarrollada es flexible puesto que reconstruye cualquier tipo de estructura o tipo de imagen. En la Figura 57 se observa la reconstrucción de un cráneo a partir de: tomografía axial computarizada, de resonancia magnética, de tomografías por emisión de positrones y por último se encuentra gran parte de la estructura ósea del cuerpo humano reconstruida a partir de imágenes de tomografías computarizadas.



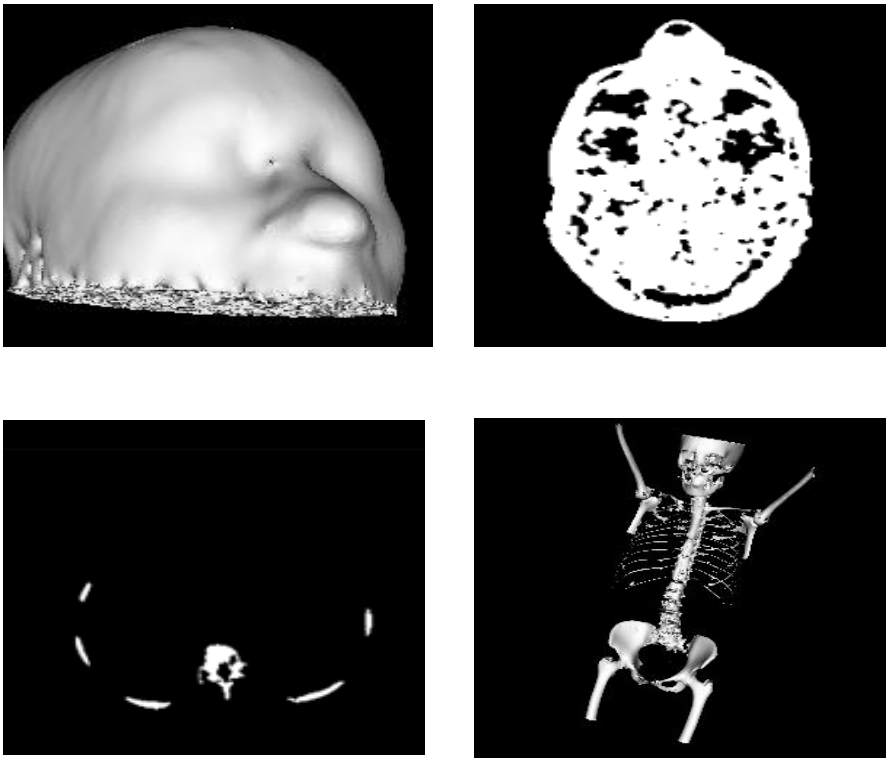


Figura 57: Distintas reconstrucciones

5.3.3. Escenario quirúrgico

Culminado con el sistema propuesto se presenta a continuación los resultados del diseño de un prototipo que se aproxima a una sala de cirugía real. Como se anotó en el capítulo 4 se construyó un quirófano a partir de objetos 3D diseñados en su totalidad en *Solid Edge* muy similares a los componentes reales.

En la Figura 58 se observa el órgano exportado sobre la mesa de cirugía junto con cada una de las herramientas y sus elementos básicos. Además este escenario cuenta con visores donde se puede observar el objeto activo, es decir, que objeto deseamos manipular, como también visores de detección de colisiones.

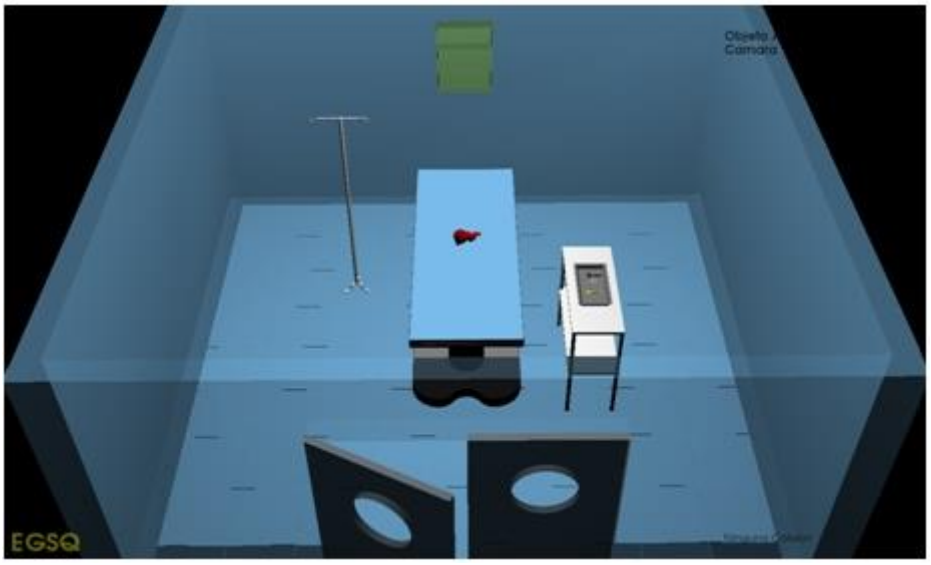


Figura 58: Vista escenario

Las partes más importantes que componen el escenario presentado, sin tener en cuenta el órgano reconstruido, son las herramientas y la mesa de cirugía como se evidencia en la Figura 59 donde se mira la proximidad a la realidad de su diseño.

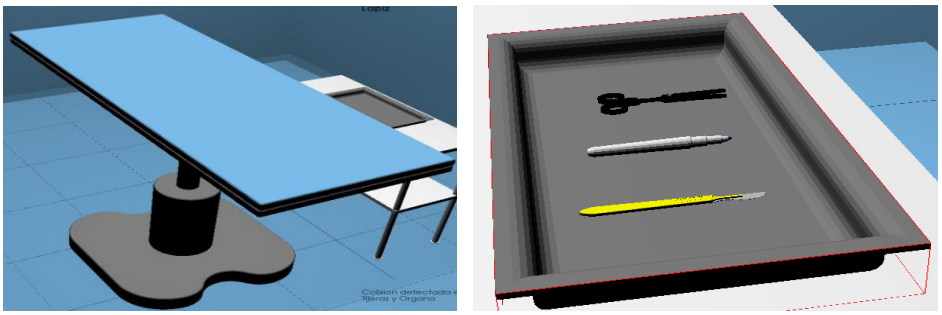


Figura 59: Herramientas y mesa quirúrgica

La interacción entre la herramienta y el órgano se puede realizar de dos maneras a través del mouse, un joystick o dispositivos de juego, mejorando de esta manera la manipulación de los objetos dentro del escenario. Para guiar este proceso se creó un asistente o *wizard* para facilitar la tarea. La ventana que aparece a continuación en la

Figura 60, surge tras presionar la tecla *Enter* dentro del escenario y posteriormente clic en el menú seleccionar *Hardware* y posteriormente en otros Dispositivos.

Figura 60: Inicio asistente dispositivo

Luego de dar clic en siguiente surge una ventana como la indicada en la Figura 61, en la cual se debe dar clic en el botón 'Actualizar Lista de Dispositivos', y surgirá un cuadro de dialogo indicando que se ha encontrado un dispositivo conectado.

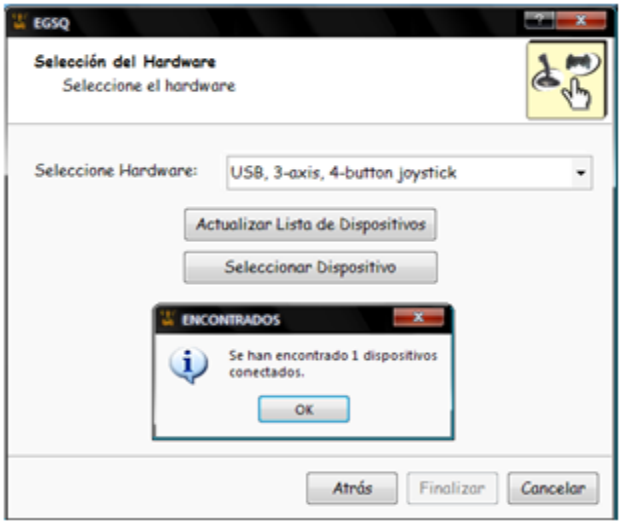


Figura 61: Selección del *hardware*

En la ventana que aparece en la Figura 62 el usuario puede seleccionar el dispositivo donde es posible comprobar su funcionamiento (color amarillo), los números correspondientes a cada botón del joystick o dispositivo (color azul) y la barra o *slide* sirve para probar el *scroll* (color verde). Finalmente, se puede hacer uso del dispositivo.

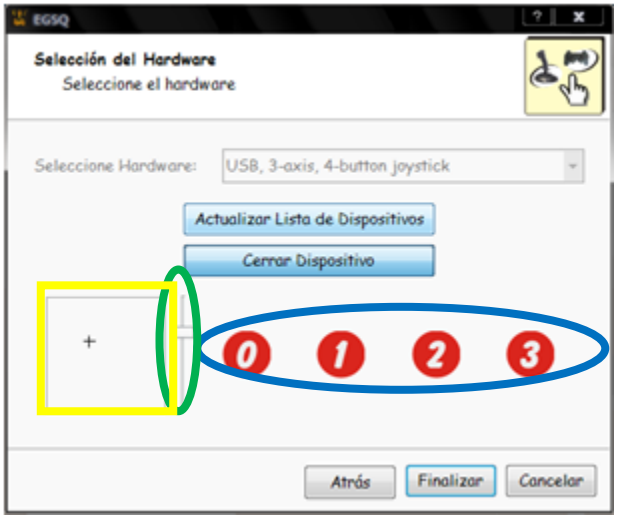


Figura 62: Funcionamiento *hardware*

Para mayor información acerca del funcionamiento del escenario diríjase al Anexo C Manual de Usuario.

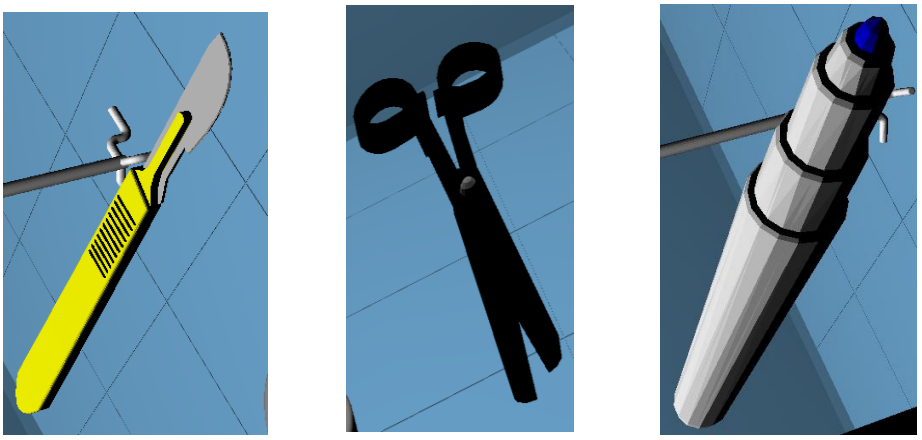
5.4. INTEGRACIÓN AL ESCENARIO

En esta parte se hace una evaluación objetiva sobre todos los componentes diseñados junto con el órgano, en este caso el hígado reconstruido, indicando las zonas donde se presentan las colisiones entre herramienta y el órgano en cuestión.

5.4.1. Verificación de componentes

La verificación de componentes que se realizó es visual, observando que cada objeto se encuentre en una posición determina, esencialmente se comprobó la posición de las herramientas quirúrgicas virtuales las cuales deben encontrarse al lado derecho de la mesa de cirugía y la posición del órgano reconstruido que debe ser encima de dicha mesa.

En la Figura 63 se aprecia: a) Bisturí común de cirugía compuesto de una cuchilla y un mango. b) Tijera con características muy similares a las quirúrgicas la cual se puede abrir y cerrar. c) Lápiz con punta azul.



a) Bisturí b) Tijera c) Lápiz quirúrgico

Figura 63: Herramientas virtuales

5.4.2. Detección de colisiones

La herramienta se puede dirigir hacia el órgano u otros componentes del escenario con el dispositivo (*mouse o joystick*) escogido con anterioridad y de esta manera poder realizar una colisión entre dos objetos.

En la Figura 64 se observa la colisión entre el órgano y la herramienta, y cómo el visor de eventos señala esta colisión

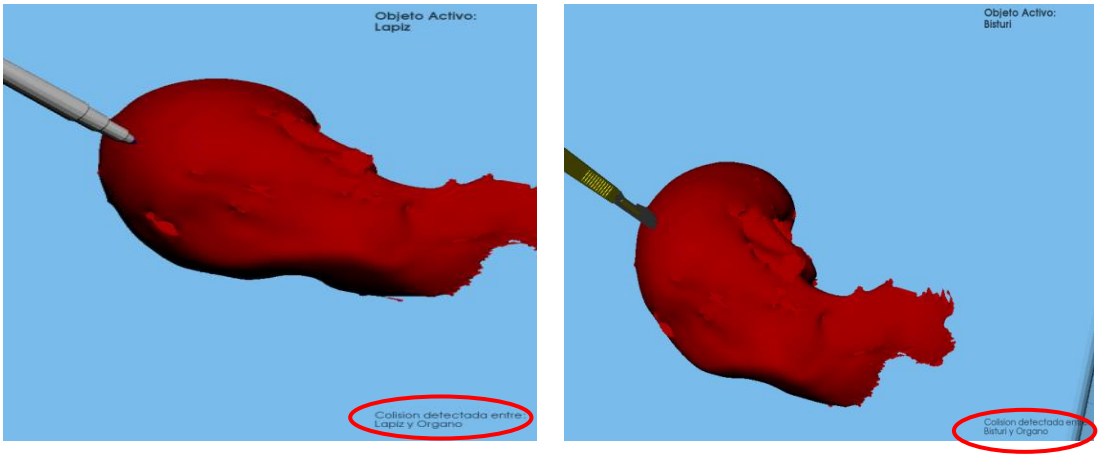


Figura 64: Interacción herramientas

En la Figura 65 se puede observar un zoom del pintado del lápiz quirúrgico, seguido del bisturí y por último el de la tijera quirúrgica sobre el órgano.

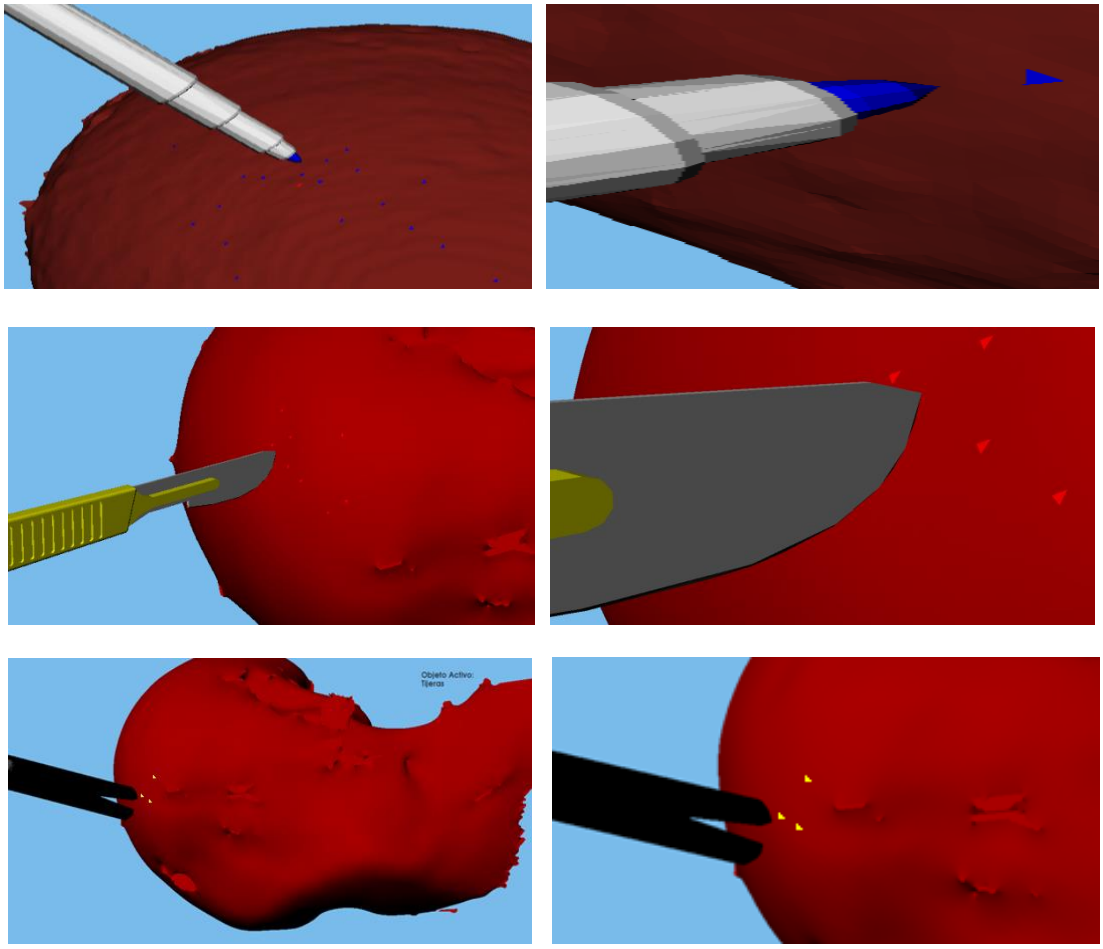


Figura 65: Colisión objeto

5.4.3. Funcionalidad de componentes

Para la funcionalidad de componentes se diseñó la aplicación de tal manera que cada herramienta realice diferentes funciones dentro del escenario, por lo tanto, se dio color al toque que realiza cada herramienta sobre un objeto, en el caso de la tijera se le dio un color amarillo, en el caso del bisturí se le dio un color rojo y por último en el caso del lápiz quirúrgico se le dio un color azul (ver Figura 66). Además todas las herramientas poseen el movimiento de elevación y rotación.

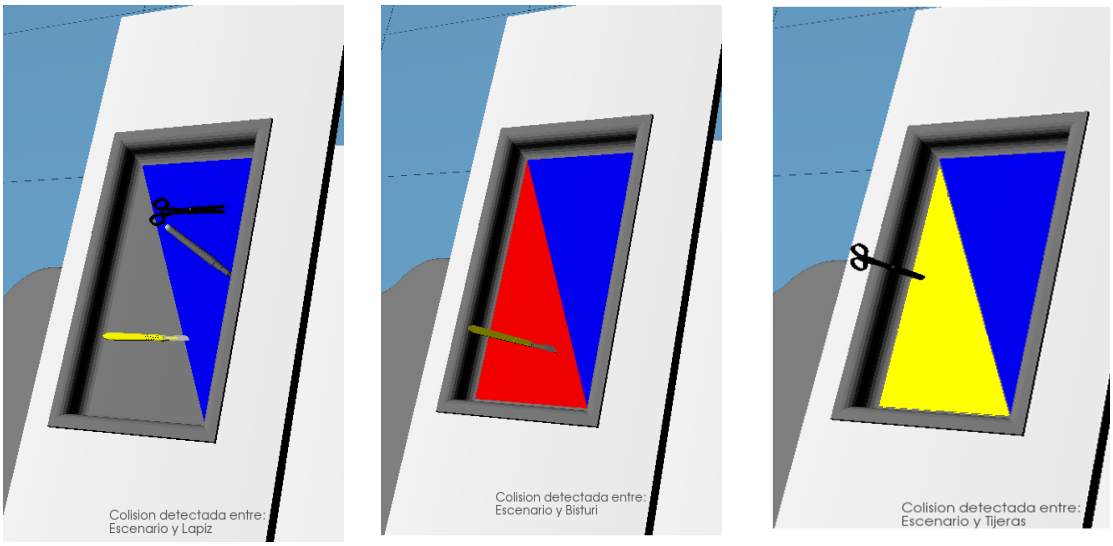


Figura 66: Funcionalidad de componentes

Como parte de la funcionalidad de la aplicación desarrollada se consideró comprobar los bajos costos computacionales que ésta genera, con lo cual se confirma la eficiencia de la aplicación desarrollada. Se procedió entonces a ejecutar la aplicación en un computador con requerimientos mínimos (256Mb de memoria RAM y procesador de 600MHz con sistema operativo Windows Xp), demostrando que el sistema interactúa activamente con relación a sus entradas y salidas dando un correcto funcionamiento de la aplicación, estableciendo que las pruebas realizadas en computadores de diversas características establecen que la eficiencia computacional del código implementado permite lograr tiempos de respuestas acordes con las interacciones del usuario.

5.5. EJECUCIÓN DE LA APLICACIÓN FINAL

Para facilitar la ejecución de la aplicación se compiló el código en *Visual Studio* 2005 en modo *Release* que vincula cada biblioteca sin la necesidad que se encuentre en el computador instaladas las bibliotecas ITK, VTK, QT y V- COLLIDE.

Se debe comprobar además la instalación de Microsoft .NET Framework 2.0 para que la ejecución de la aplicación se pueda realizar. Si esta aplicación no se encuentra descargarla de:

http://www.microsoft.com/downloads/en/details.aspx?FamilyID=0856eacb-4362-4b0d-8edd-aab15c5e04f5

En la carpeta ‘Aplicación Final’ se encuentra la carpeta EGSQ_bin\ realse\ donde está el ejecutable de la aplicación final ‘EGSQ.exe’ (iníciales de Entorno Grafico de

un Simulador Quirúrgico). Al dar clic en ejecutar se presentará el aspecto inicial como se observa en la Figura 67.

Figura 67: Pantalla inicial de la aplicación desarrollada

Las partes principales de la aplicación se dividen en tres bloques diferentes.

Barra de Menú: la Barra de Menú es donde aparecen los diferentes menús y aplicaciones en forma de texto como se observa en la Figura 68. El menú Archivo permite al usuario crear un nuevo proyecto.

Figura 68: Barra de menú

El menú Seleccionar *Hardware* contiene las opciones *mouse* u otros dispositivos lo cual es útil a la hora de manejar las herramientas virtuales en el escenario quirúrgico.

Barra de herramientas: en la barra de herramientas se tiene el acceso al archivo nuevo y se muestra mediante íconos, como se observa en la Figura 69 . Además se encuentra el ícono guardar donde se dispuso un asistente para guardar cualquier tipo de archivo generado en la aplicación, en cualquier punto del proceso (ver Figura 70).

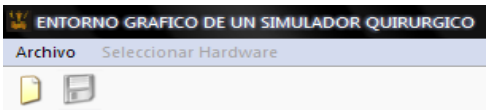


Figura 69: Barra de herramientas

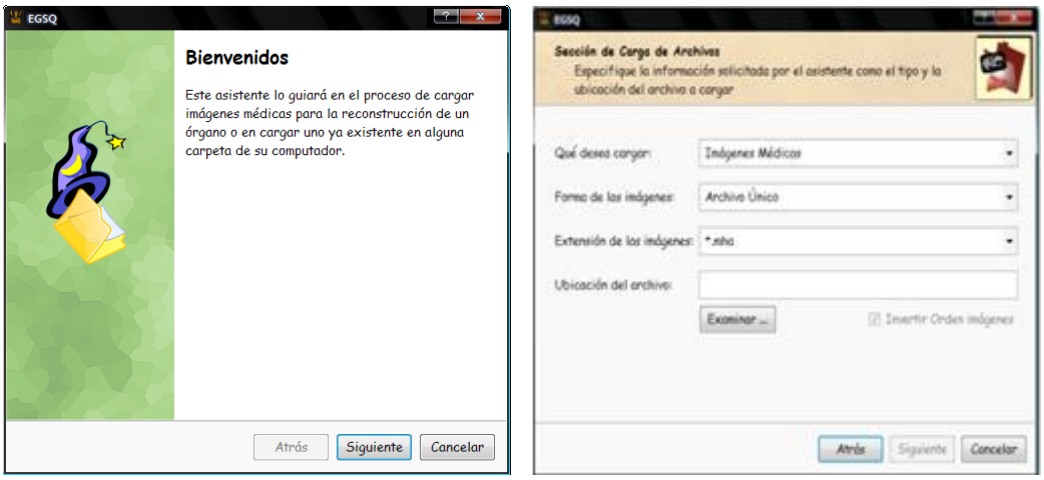


Figura 70: Asistente de guardar

Entorno de trabajo: este entorno cambia dependiendo el proceso en que se encuentra el usuario, principalmente existen cuatro ventanas tal como se observa en la Figura 71: a) pantalla inicial, b) entorno de pre-procesamiento, c) entorno de segmentación y reconstrucción, y por ultimo d) entorno escenario quirúrgico.

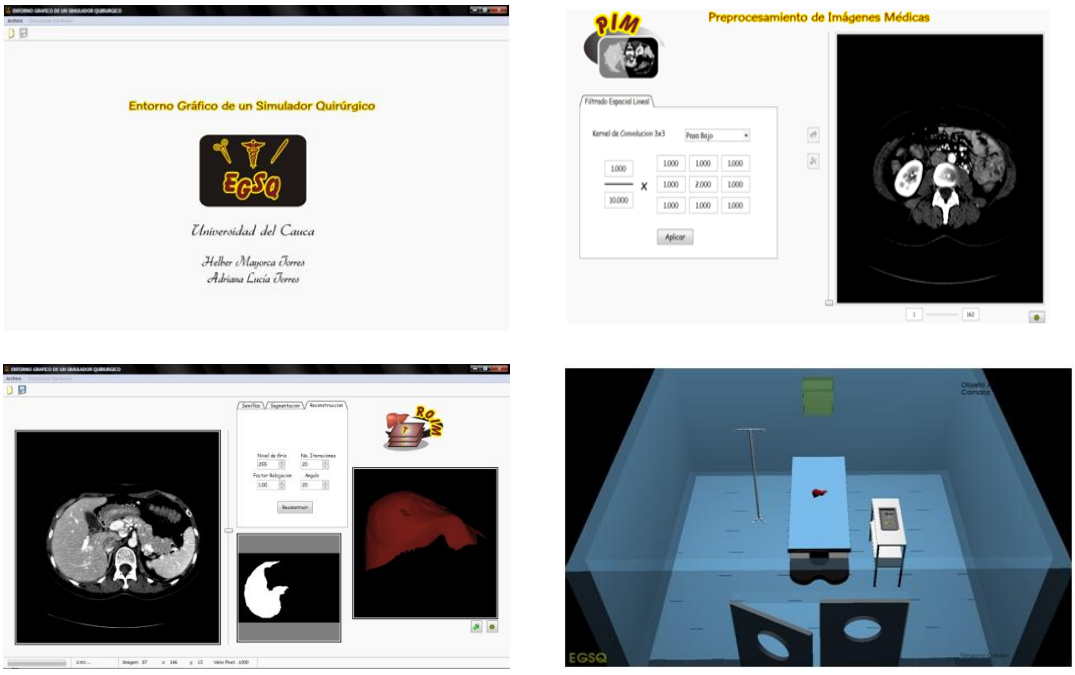


Figura 71: Entornos de trabajo

Para mayor información sobre la funcionalidad diríjase al anexo C. Manual de usuario.

5.6. COMPARACIÓN DE LA APLICACIÓN CON OTRAS SIMILARES

A continuación se realizará una comparación objetiva del proyecto titulado *liver* [22], el cual fue desarrollado en la Universidad Miguel Hernández de Elche en España, con el presentado en este trabajo de grado.

Para efectos de comparación la aplicación presentada en este trabajo de grado se titulará aplicación 1 y la desarrollada en la Universidad Miguel Hernández se titulará aplicación 2.

5.6.1. Carga de imágenes

Ventajas aplicación 1: es posible cargar imágenes a través de un asistente que guía al usuario constantemente sin perder la secuencialidad de este proceso, y es permitido observar las imágenes y decidir si eran las que se deseaba cargar. También se pueden cargar órganos reconstruidos que posteriormente son visualizados en una ventana y finalmente exportados al escenario quirúrgico (ver Figura 72).

Ventajas aplicación 2: en la aplicación 2 es posible cargar imágenes en muchos tipos de formatos y se puede escoger el tipo de píxel de la imagen (Ver Figura 73).

Figura 72: Asistente de carga aplicación 1

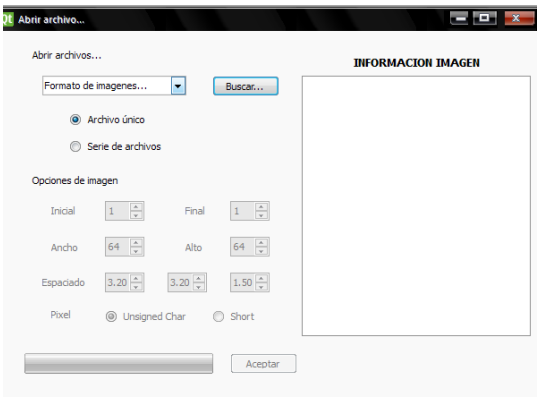


Figura 73: Asistente de carga aplicación 2

5.6.2. Reconstrucción

Ventajas aplicación 1: la aplicación 1 está dispuesta para seguir un proceso secuencial para que el usuario no se pierda en el camino, por lo tanto, se diseñó de tal manera que después de cargar las imágenes, se pueda realizar un pre-procesamiento, donde se realice transformaciones geométricas que evidencien un correcto corte de la imagen. Además se dispone de íconos que el usuario asemeja a secuencia de tal forma que estos íconos puedan ser utilizados para continuar con el proceso que en este caso sería la segmentación. En esta parte se pueden realizar no solo un método de segmentación sino varios donde se ingresan parámetros ordenadamente. Siguiendo con el proceso de reconstrucción en esta aplicación se dispone de sub-ventanas que ubican al usuario en que parte del proceso se encuentra, además al terminar la reconstrucción del órgano se puede rotar y realizar acercamientos para apreciar una correcta reconstrucción (ver Figura 74).

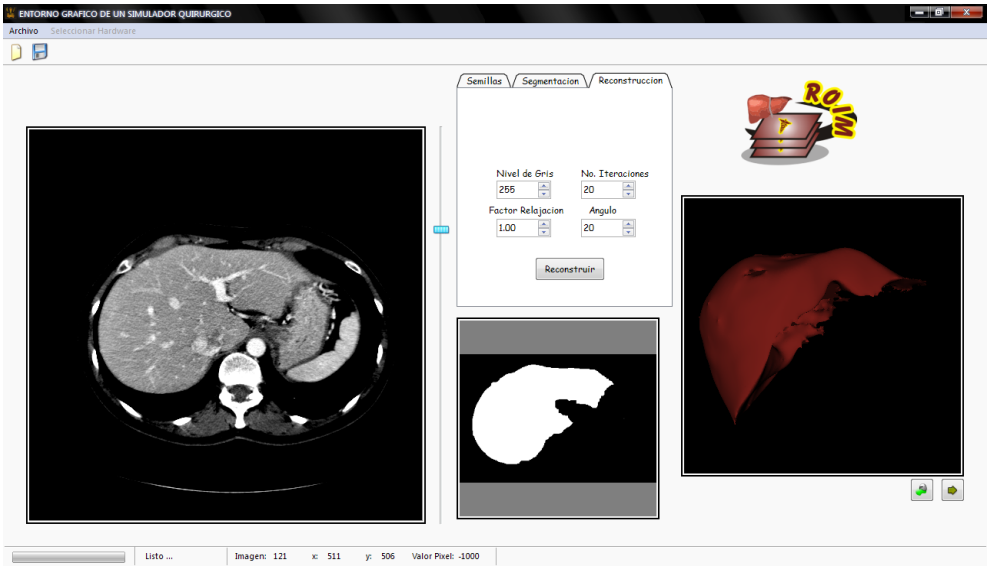


Figura 74: Segmentación y reconstrucción aplicación 1

Como observamos en la anterior figura la reconstrucción y la segmentación del hígado es óptima y la malla superficial no presenta picos o rugosidades.

Ventajas aplicación 2:

Se puede escoger entre dos tipos de imágenes segmentadas y de esta manera reconstruir partes separadas del órgano tal como se observa en la imagen (ver Figura 75).

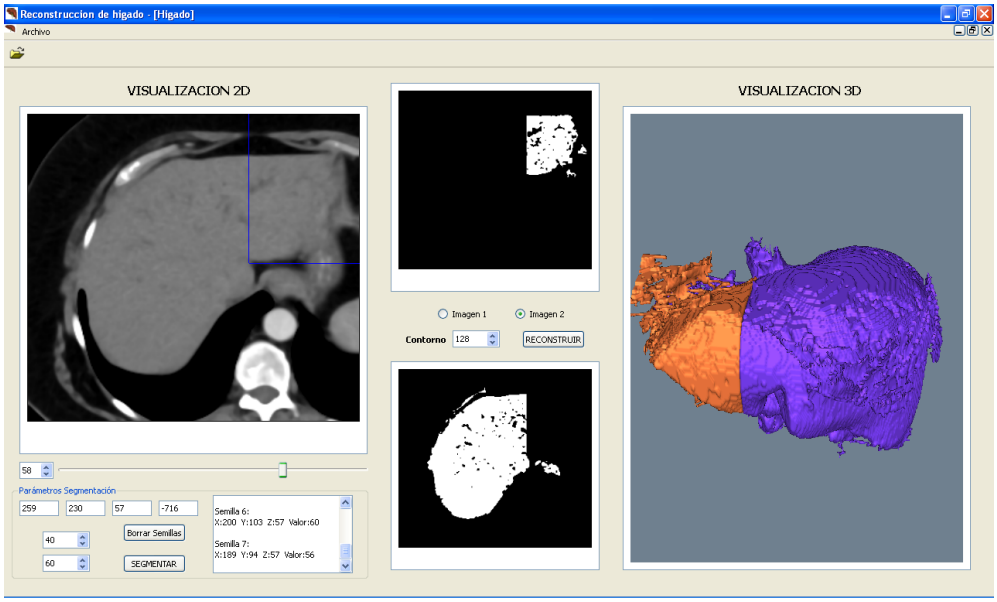


Figura 75: Segmentación y reconstrucción aplicación 2

CONCLUSIONES

El presente trabajo de grado abordó la parte correspondiente al entorno gráfico del simulador quirúrgico, el cual constituye la primera versión del proyecto macro. Para poder desarrollar con éxito dicho proyecto, ha sido imprescindible utilizar las bibliotecas como: ITK, VTK, QT y *V-Collide*. Tras un largo proceso de estudio y diseño de diferentes aplicaciones se logró desarrollar un análisis de las bibliotecas y estudiar las clases pertenecientes a dichas bibliotecas para su correcta adaptación a las necesidades de los algoritmos que se implementaron, además se realizó un estudio general del proceso de integración de la biblioteca VTK y de la biblioteca QT mediante el gestor de proyectos CMAKE.

- El proceso de cargar las imágenes hizo la aplicación flexible, puesto que se puede trabajar con varios formatos de visualización de imágenes y formatos DICOM (estándar para la comunicación de imágenes médicas).

- Los filtros espaciales dispuestos en la aplicación efectivamente eliminan el ruido de las imágenes, y en el caso de los filtros *sobel* y *prewitt*, resaltan características de bordes haciendo más fácil la distinción de elementos que se encuentran en una imagen médica.

- El corte de las imágenes realizó una correcta transformación geométrica, puesto que elimina píxeles innecesarios evidenciándose en el aumento de la velocidad de procesamiento a la hora de segmentar y reconstruir.

- Los métodos de segmentación propuestos en la aplicación fueron los indicados para imágenes medicas, ya que la mayoría arrogó excelentes resultados, entre los cuales se destaca el método *level set* el cual ofrece muy buenos efectos, aunque su procesamiento es lento.

- La reconstrucción del hígado fue la adecuada, puesto que al ser comparada con una imagen típica del hígado se observó que era correcta, además se puede decir que la reconstrucción a través de secciones transversales arroja los mejores resultados.

- Las herramientas virtuales diseñadas son muy aproximadas a la realidad, además fueron capaces de hacer toques sobre el órgano, detectando una colisión precisa y diferente dependiendo la herramienta que colisione con el órgano.

- El sistema de colisiones agregó realismo al escenario ya que los componentes como las herramientas virtuales al ser desplazadas y posteriormente colisionadas no atraviesan ningún elemento.

- Finalmente se puede decir que se aumentó el conocimiento y se sentó una base teórica y práctica sobre los temas que se trataron en el presente trabajo de

grado, ya que se apropiaron los elementos para la resolución de un problema particular generando resultados satisfactorios.

TRABAJOS FUTUROS

En esta parte cabe destacar que los trabajos futuros son variados, entre los más importantes destacamos el de convertir el escenario con más opciones que permitan al usuario un mayor manejo de la aplicación y una mejora en cuanto al realismo del entorno.

Como se anotó en las conclusiones el presente trabajo de grado hace parte del proyecto de investigación macro llamado **“Docencia e investigación en robótica medica utilizando recursos *software* de código abierto - OpenSurg”** Con este proyecto se logró sentar una base con la que trabajarán posteriores estudiantes muy seguramente tanto en la evolución del programa presentado como en la continuidad del proyecto macro integrando interfaces hápticas que ayuden al manejo mucho más interactivo de los componentes del escenario, como también algoritmos de deformación de órganos que cumplan con características visuales más reales y de esta manera cumplir con el objetivo final de crear un simulador quirúrgico con características muy aproximadas a la realidad y con bajos costos.

REFERENCIAS BIBLIOGRAFÍA

[1] García Alcolea, Ortiz González, Sánchez Miranda, “Simulación Quirúrgica: Una Alternativa En La Enseñanza De La Oftalmología”, “Revista Misión Milagro” Vol.2 No.1 pp 4, Marzo 2008.

[2] Monserrat C, Alcañiz M, Meier U, Poza J. “Simulador para el entrenamiento en cirugías avanzadas”. XII Congreso Internacional de Ingeniería Gráfica, Valladolid, España, 2000.

[3] Monserrat C. “GERTISS: Simulador quirúrgico virtual para el entrenamiento en cirugías mínimamente invasivas”. IX Congreso Nacional De Informática Médica, Valencia, España, 2002.

[4] Simbionix. “ANGIO Mentor™”. Disponible en: http://www.simbionix.com/ANGIO_Mentor.html

[5] Isurgical. “I-Sim”. Disponible en: http://www.isurgicals.com/isim.htm

[6] Insightmist. ARTHRO VR “Simulador De Entrenamiento Artroscópico Avanzado” Disponible en: http://www.insightmist.com/descripcion/descripcion.htm

[7] Orozco J, “Aplicaciones Médicas De Las Técnicas Inmersivas De Realidad Virtual”. Trabajo de Grado. Escuela de Ingeniería Eléctrica. Universidad Central de Venezuela, 2007.

[8] Smith Fong S, “Principios de Reconstrucción de Imágenes en Tomografía Computarizada” Trabajo de Grado. Facultad de Ingeniería En Sistemas, Informática y Ciencias de la Computación. Universidad Francisco Marroquín, Guatemala, 1986.

[9] Pérez B. “Modulo De Navegación Para Un Sistema De Entrenamiento Virtual Aplicado A Cirugía De La Base Del Cráneo” Trabajo de grado. Facultad De Ingeniería. Universidad Nacional De Colombia. Colombia, 2008.

[10] Isaza J, Reyes S, Lamus C, Zapata U, Roldan S. “Reconstrucción Tridimensional De Estructuras Óseas a Partir De Estudios Tomográficos Para Su Manipulación En Modelos De Elementos Finitos” Revista CES Odontología, Vol. 18 No. 2, 2005.

[11] Tenreiro O. “Angiografía Con Catéter Por Sustracción Digital” Disponible en: http://www.cirugiaendovascular.com.ve/link_enf_atero/angio_catet_sust.htm

[12] Martínez W. “Segmentación de Imágenes Médicas” Trabajo de Grado. Facultad de Ingeniería. Universidad Nacional de la Plata. Argentina, 2008.

[13] “El Hígado. Evaluación de la silenciosa evolución de la enfermedad hepática” Disponible en: http://www.pkids.org/Spa_phrliv.pdf

[14] Escalera A. ‘Visión Computacional: Fundamentos y Métodos’, Prentice Hall, 2001.

[15] Ladero J. “Capitulo 20 Hígado” Manual Normo, Edición 8, 2000.

[16] Highleyman L. “Introducción sobre el Hígado” Tesis Doctoral. Facultad de Farmacia. Universidad de Granada, España, 2009.

[17] “Anatomía del Hígado”. Facultad de Medicina. Universidad Autónoma Nacional. Disponible en:
http://www.facmed.unam.mx/deptos/anatomia/computo/higado/index2.html

[18] Coto E. “Métodos de Segmentación de Imágenes Médicas”. Trabajo de Grado. Facultad de ciencias. Escuela de Computación. Universidad Central de Venezuela, 2003.

[19] Azouzi S. “Editor de Imágenes basado en Regiones. Aplicación en entorno Matlab” Trabajo de Maestría. Escuela Universitaria de Ingeniería Técnica Industrial, España, 2005.

[20] Vargas H. “Algoritmo De Segmentación Topológico Para Imágenes Adquiridas De La Tomografía Computada y La Resonancia Magnética.” Revista Cubana de Informática Medica. Vol. 1, No. 1, 2000.

[21] Arámbula F. “Inicialización automática de un modelo de la forma activa de la próstata” Revista Análisis de Imágenes Médicas. Vol. 12, No 4, pp. 45-43, 2006.

[22] Reolid J. “Generación de Órganos Sólidos A partir de Imágenes Medicas” Trabajo de grado. Ingeniería de Telecomunicaciones Universidad Miguel Hernández de Elche, España, 2008.

[23] Rodríguez B, Gaya F, Pozo F, García F, Gómez E. “Método De Evaluación Tridimensional de Algoritmos de Registro Deformable para Radioterapia Guiada Por Imagen. XXVI Congreso Anual de la Sociedad Española de Ingeniería Biomédica, Valladolid, España, 2008.

[24] Jiménez J. “Detección de Colisiones Mediante Recubrimientos Simpliciales” Tesis Doctoral. Departamento de Lenguajes y Sistemas Informáticos. Universidad de Granada, España, 2006.

[25] Melero J. “BP Octree: Una Estructura Jerárquica de Volúmenes Envolventes” Trabajo de Grado. Departamento de Lenguajes y sistemas Informáticos. Universidad de Granada. España. 2008.

[26] Ibáñez L, Schroeder W, NG L, Cates J, Insight Software Consortium. “The ITK Software Guide Second Edition Update for ITK” Version 2.4, 2005.

[27] “UML”:http://usuarios.multimania.es/oopere/uml.htm

[28] Booch G, Rumbaugh J, Jacobson I. “Guía Básica de UML El Lenguaje Unificado de Modelado”, Addison Wesley, 1999.

[29] Hernández E. “Lenguaje de Modelado Unificado UML”. http://www.domotica.us/Lenguaje%20Unificado%20de%20Modelado