Modelado virtual de huesos largos

José Franco Campos

28 de octubre de 2010

Índice general

Índice de figuras

0 Resumen

El objetivo de este proyecto es el desarrollo de un algoritmo para la generación de modelos virtuales de huesos largos, combinando una base de datos de modelos 3D de huesos con datos anatómicos del paciente, con el objetivo de crear un sistema de CAS (*computer aided surgery*, cirugía asistida por ordenador) para cirugía traumatológica. En primer lugar, se ha creado la base de datos. Para ello se han realizado exploraciones de TAC con cadáveres, aislando el hueso en el que estamos interesados (el fémur), y generando una malla poligonal a partir de las imágenes volumétricas. Cada malla ha sido etiquetada manualmente, anotando la posición de una serie de puntos de interés que indican características anatómicas interesantes. Posteriormente, y de manera previa a la operación, se construye el modelo virtual del paciente. Para ello se toman radiografías del fémur desde diferentes ángulos. El cirujano marca los puntos de interés en la radiografía, y el sistema combina la posición de los puntos con la información contenida en la base de datos para construir el modelo del paciente, eligiendo el modelo óptimo de la base de datos y deformándolo para adaptarlo a la anatomía del paciente.

Los resultados obtenidos hasta la fecha son satisfactorios. El grado de error está dentro de parámetros aceptables, y esperamos que nuestro sistema permita a los cirujanos obtener una mejor representación de los huesos del paciente sin necesidad de usar TAC, reduciendo la cantidad de radiación absorbida por el paciente. La base de datos creada para el proyecto es muy útil, y puede servir como base para proyectos futuros. En el último capítulo se exponen las conclusiones alcanzadas, y se comentan posibles rutas por las que se podría continuar el presente trabajo.

Palabras clave: cirugía asistida por ordenador, imagen volumétrica, modelado virtual, registrado, isosuperficie.

Abstract

The goal of this proyect is the development of an algorithm for the generation of virtual models of long bones, merging a database of 3D models of bones with anatomical data from the pacient, with the goal of creating a CAS (computer aided surgery) system for traumatological surgery. The first step has been the creation of the database. We have done several CT scans of cadavers, isolating the bone in which

we are interested (the femur), and extracting a polygonal mesh from the volumetric images. Each mesh has been manually tagged, anotating the position of a number of points of interests which denote interesting anatomical characteristics. Then, and previously to the surgery, the virtual model of the pacient is built. First, a number of radiographs of the patient's femur are taken from different angles. The surgeon marks the points of interest in the radiographs, and the system combines the position of the points with the information stored in the database to build the model of the patient, choosing the optimal model from the database, and deforming it to adapt it to the patient's anatomy.

The results obtained to date are satisfactory. The error metric lies within acceptable bounds, and we expect our system to allow surgeons to better visualize the patient's bones without the need of a CT scan, thus reducing the amount of radiation the patient is exposed to. The created database is very useful, and can be reused as the foundation of future projects. In the last chapter we expose the conclusions we have reached, and several ways in which this work could be continued.

Keywords: computed aided surgery, volumetric image, virtual modelling, image registration, isosurface.

1 Introducción

El presente trabajo surge como parte del proyecto Fracture Assistante, llevado a cabo en el Centro de Cirugía de Mínima Invasión Jesús Usón, cuyo objetivo es la creación de un sistema de navegación para cirugía traumatológica. Dicho sistema permitirá a los traumatólogos obtener y manipular una representación tridimensional de los huesos del paciente y del instrumental quirúrgico, con el objetivo de facilitar el tratamiento de fracturas de huesos largos (fémur, tibia, peroné, húmero, cúbito y radio) mediante técnicas mínimamente invasivas.

El primer requisito del proyecto era, por tanto, desarrollar un método que permitiera obtener modelos 3D de la anatomía del paciente, de manera previa a la operación. Inicialmente se barajó la idea de usar un TAC. Desarrollada en 1967 por Godfrey Houndsfield, la tomografía axial computerizada permite obtener imágenes tridimensionales de los huesos y demás tejidos del cuerpo. Dichas imágenes ofrecen una resolución excelente, de menos de un milímetro por píxel, y pueden ser procesadas para extraer modelos 3D que servirían de base a nuestro sistema.

Sin embargo, el TAC cuenta con dos desventajas importantes. El primero es el elevado coste de la maquinaria, lo que hace que muchos hospitales españoles carezcan de este tipo de instalaciones. No menos importante es el hecho de que somete al paciente a una elevada dosis de radiación ionizante. Dependiendo de la zona del cuerpo, un escáner TAC puede ser equivalente a más de 50 radiografías convencionales.

Por ello, se propuso el desarrollo de un sistema alternativo, que permitiría obtener reconstrucciones virtuales de huesos largos a partir de radiografías, sin necesidad de efectuar un escáner TAC del paciente. Dicho sistema combinaría las radiografías del paciente con una base de datos de modelos de huesos, para reconstruir el modelo del paciente a partir del modelo de la base de datoscuya similitud sea mayor. El desarrollo del sistema se divide en dos fases bien diferenciadas:

- En primer lugar, se creará una base de datos con la forma de los huesos. Para ello, se tomarán escáneres TAC de unos 10 ó 12 pacientes, se extraerá una malla poligonal con la forma del hueso deseado, y se marcarán una serie de puntos de interés en el hueso. Este proceso se realizará de forma offline, durante la puesta en marcha del sistema.

- Posteriormente, de manera preoperatoria, se tomarán un par de radiografías (lateral y anteroposterior) del paciente. El cirujano marcará, en ambas radiografías, los mismos puntos que se marcaron durante el etiquetado de los huesos. El sistema buscará en la base de datos el modelo más parecido, y lo deformará para ajustarlo a la anatomía del paciente, construyendo un modelo virtual aproximado del hueso.

En los capítulos siguientes se describirá con mayor detalle cada parte del proceso.

2 Toma de datos

2.1. Fuentes de datos

Para el desarrollo de este trabajo se ha hecho uso de dos tipos de fuentes de datos: TAC y fluoroscopio.

Desarrolladas en 1967 por Godfrey Houndsfield, las máquinas de TAC (tomografía axial computerizada) permiten obtener radiografías axiales, es decir, perpendiculares al eje longitudinal del cuerpo. Mediante el uso de detectores que giran alrededor del cuerpo a estudiar, permiten obtener imágenes de rodajas del cuerpo a estudiar. Obteniendo múltiples rodajas a intervalos regulares, se puede construir una imagen volumétrica del objeto, lo cual proporciona una enorme cantidad de información útil, ya que permite observar el interior de dicho objeto.

El TAC devuelve una secuencia de imágenes planas. La resolución de dichas imágenes depende del modelo usado; en los TAC existentes en la actualidad para uso clínico, la resolución suele ser de 512×512 píxeles, y 12 bits por pixel. La separación entre cortes puede ser definida por el operador, y suele encontrarse entre los 5 y los 20 mm. Una menor separación proporciona resultados más precisos, pero incrementar el tiempo de exploración y por tanto la dosis de radiación. Puesto que el TAC usa rayos X, su utilidad es la misma que la de una radiografía convencional: diferenciar los huesos del resto de tejidos blandos, como los músculos y los órganos internos. No obstante, se puede hacer uso de agentes de contraste que permiten explorar algún tejido u órgano concreto.

La principal ventaja del TAC es la gran calidad de los resultados que proporciona. Entre las principales desventajas, cabe destacar dos. La más evidente es la gran cantidad de radiación a la que somete al paciente: un TAC abdominal puede ser equivalente a 50 radiografías convencionales. Por otro lado, el TAC no es capaz de visualizar objetos que contengan metal. La elevada opacidad de los metales provoca artefactos en las imágenes, que las vuelven inservibles. No obstante, el TAC es una herramienta insustituible a la hora de obtener imágenes tridimensionales de huesos y otros órganos.

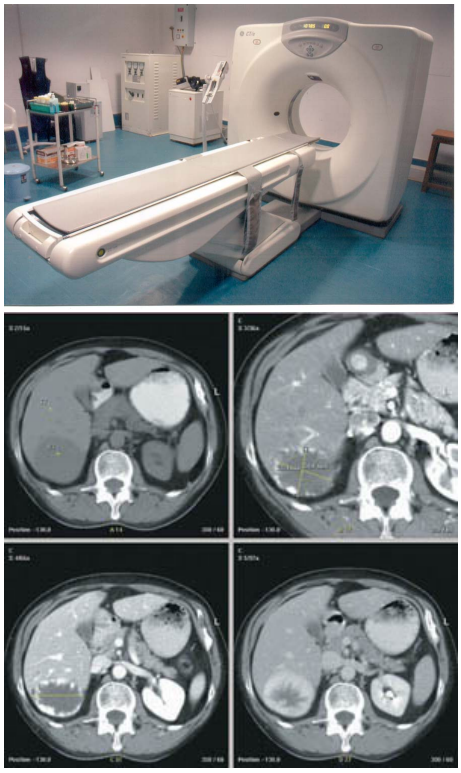

Figura 2.1: Máquina de TAC

Los fluoroscopios son máquinas de rayos X que permiten obtener imágenes del cuerpo del paciente en tiempo real. Los primeros modelos consistían de una fuente de rayos X y una pantalla fluorescente entre las que se situaba el paciente, de ahí el nombre. Los modelos modernos usan un intensificador de imagen y una cámara CCD, lo que permite obtener imágenes más nítidas con una dosis de radiación menor, así como reproducir y grabar los resultados en un ordenador. Los fluoroscopios suelen estar montados como un brazo en forma de C, con la fuente de radiación en un extremo y el detector en la otra. La base del fluoroscopio dispone de ruedas, que permiten moverlo libremente. Además, la C puede rotar sobre su propio eje, para poder ajustar el ángulo de exploración. Esta movilidad, unida a su tamaño compacto, hacen posible su uso en un quirófano.

Los fluoroscopios pueden adquirir una única radiografía individual, o pueden funcionar en modo continuo. En este modo están adquiriendo imágenes constantemente, como una cámara de vídeo. Este modo es muy útil para fines quirúrgicos, ya que permite al cirujano observar la posición del instrumental y de los órganos del paciente en tiempo real, mientras realiza la propia operación. Sin embargo, cuenta con la desventaja de que somete al paciente a una elevada dosis de radiación. Aún así, su uso es imprescindible en las operaciones de traumatología, objeto del presente trabajo.

Figura 2.2: Fluoroscopio

2.2. El formato DICOM

Desde su introducción en 1985, el formato DICOM (Digital Imaging and Communication in Medicine) se ha establecido como el estándar para el almacenamiento y transmisión de imágenes médicas. Anteriormente, cada fabricante usaba su propio formato, lo que dificultaba la interoperabilidad. DICOM surgió como un esfuerzo conjunto para unificar los diferentes formatos en uno solo. Define un formato de almacenamiento de imágenes, y un protocolo de comunicaciones basado en TCP/IP.

El formato DICOM es muy complejo, por lo que un estudio completo queda fuera de los objetivos del presente trabajo. Se hará, por tanto, un resumen de las características más importantes. A grandes rasgos, un fichero DICOM consta de una cabecera de metadatos y una imagen. Un fichero sólo puede contener una imagen, pero dicha imagen puede constar de varios fotogramas, lo que permite almacenar secuencias temporales. Entre los campos de la cabecera, cabe destacar los siguientes:

- Datos identificativos del paciente, como su nombre y apellidos.
- Datos identificativos del médico que ha ordenado la exploración, y del que la ha realizado.
- Tipo de exploración: radiografía, TAC, RMN, PET, ecografía, etc.

- Datos de la máquina que ha realizado la exploración: fabricante, modelo, número de serie, etc.
- Fecha y hora de la exploración.
- Datos técnicos de la imagen: resolución, profundidad de color, formato de compresión, separación entre cortes (para escáneres 3D), apertura de la lente (si es aplicable), etc.

Puesto que nuestro objetivo es crear un modelo 3D a partir de la imagen, sólo nos interesan los datos técnicos. En primer lugar, DICOM no define el formato de compresión, sino que hace uso de los estándares existentes. Algunos de los formatos que soporta son:

- RAW (sin compresión).
- JPEG.
- JPEG 2000.
- RLE.
- LZW.

Las imágenes pueden tener un número de canales de color arbitrario. Lo normal es que tengan 1 ó 3, pero pueden tener más para el caso de imágenes multiespectrales. DICOM también soporta diferentes profundidades de color:

- Entero de 8 bits, con y sin signo.
- Entero de 12 bits, con y sin signo.
- Entero de 16 bits, con y sin signo.
- Entero de 32 bits, con y sin signo.
- Coma flotante de 32 bits.
- Coma flotante de 64 bits.

Para el tratamiento de imágenes radiológicas se usan las unidades Houndsfield, que miden la radiodensidad del material. Las máquinas de TAC miden el coeficiente de atenuación de los rayos X al atravesar los distintos materiales; la escala Houndsfield es una transformación lineal de dicho coeficiente, definido como:

$$HU = 1000 \frac{\mu_x - \mu_{agua}}{\mu_{agua} - \mu_{aire}}$$

Figura 2.3: Conversión de coeficiente de saturación a unidades Houndsfield

donde μ_{agua} y μ_{aire} son los coeficientes de atenuación del agua y el aire. Los valores normales de diferentes tipos de tejidos son:

Sustancia	HU
Aire	-1000
Grasa	-120
Agua	0
Músculo	+40
Contraste	+130
Hueso	+400

Figura 2.4: Unidades Houndsfield para diferentes tipos de materiales

El uso de estas unidades facilita mucho nuestra tarea, ya que la segmentación de los diferentes tipos de tejido se convierte en una simple umbralización. Sin embargo, las imágenes DICOM no suelen guardar los valores Houndsfield directamente, sino que se hace un reescalado lineal para que todos los valores sean positivos, y aprovechar al máximo el rango del tipo de datos usado. Para obtener las unidades Houndsfield originales hay que invertir el reescalado, usando dos valores de la cabecera llamados *RescaleSlope* y *RescaleIntercept*. La fórmula resultante es la siguiente:

$$HU = RescaleSlope + D \cdot RescaleIntercept$$

Figura 2.5: Conversión de valores DICOM a unidades Houndsfield

A la hora de reconstruir el volumen, hay que tener en cuenta que cada fichero DICOM guarda únicamente una imagen bidimensional, es decir, un corte del TAC. Para poder obtener el volumen de datos, hay que "pegar" todos los cortes. Para ello hay que tener en cuenta una serie de factores. El más importante es el tamaño de los vóxeles. La cabecera DICOM define el ancho y el alto de cada píxel, así como el ancho de cada corte y la separación con respecto al corte anterior. El ancho puede ser mayor que la separación, lo que da lugar a cortes solapados. Esto suele ocurrir con los modernos TACs helicoidales. Por otro lado, podemos encontrarnos con una serie de dificultades:

- Cortes con diferentes separaciones: la separación entre los cortes no tiene porqué ser fija, sino que puede variar a lo largo de la exploración. A la hora de reconstruir el volumen habrá que interpolar en las zonas en las que la separación sea mayor, para adaptarlas a las zonas de menor separación.
- Cortes con diferentes orientaciones: los cortes suelen ser perpendiculares al eje longitudinal, pero también pueden ser oblicuos.

- Cortes con diferentes centros: la posición central de cada corte puede ir variando, desplazándose en los ejes paralelos al eje longitudinal.

Puesto que en nuestro trabajo hemos realizado las exploraciones nosotros mismos, en un entorno controlado, y siempre buscando cumplir nuestros objetivos, hemos decidido no tener en cuenta estos casos. Sin embargo, deberíamos tenerlos en consideración si quisiéramos implementar un software de reconstrucción volumétrica de propósito general.

3 Visualización volumétrica

3.1. Introducción

La visualización volumétrica es el conjunto de técnicas usadas para poder visualizar datos volumétricos. En principio, podemos clasificar dichas técnicas en dos grandes ramas: aquellas que extraen una malla poligonal del volumen y muestran dicha malla, y aquellas que muestran el propio volumen. En este capítulo nos centraremos en el segundo caso.

El objetivo inicial de implementar una técnica de este tipo era doble. Por un lado, poder mostrar los datos obtenidos de manera directa, para comprobar que son correctos. Por otro, hay ciertos algoritmos de registrado fluoroscopio-TAC que funcionan creando radiografías virtuales desde diferentes ángulos a partir de los datos del TAC, y eligiendo la más parecida a la imagen que proporciona el fluoroscopio. Para ello hace falta que el algoritmo de visualización sea lo más rápido posible, ya que se deben renderizar un gran número de imágenes desde ángulos distintos. Aunque al final se decidió no usar este algoritmo de registrado, para entonces el algoritmo de visualización ya estaba implementado.

La mayor dificultad que encontramos es que las GPUs de uso cotidiano no soportan el renderizado volumétrico de manera directa, por lo que hay que aproximarlo. Para ello, contamos con dos técnicas básicas, *slicing* y *raycasting*, que explicaremos en las siguientes secciones.

3.2. Visualización por *slicing*

El concepto de *slicing* es muy sencillo. Consiste en renderizar cada uno de los cortes de manera individual, mapeando cada corte como una textura en un cuadrilátero convenientemente trasladado y escalado. Los cortes se renderizan ordenándolos por profundidad, de mayor a menor distancia. El color de los cortes se suma, usando un buffer de acumulación. Hay que tener en cuenta que las imágenes originales suelen tener una profundidad de color de al menos 12 bits por canal, y que el número de cortes suele ser muy alto. Esto quiere decir que necesitamos alrededor de $12 + \log_2 S$ bits de profundidad, siendo S el número de cortes. Por ejemplo, para una imagen con 200 cortes necesitaríamos unos 20 bits de precisión. Si se usan menos, se corre el

riesgo de saturar algún pixel, perdiendo precisión en las zonas de mayor densidad, o bien hay que reducir el rango de los valores antes de renderizarlos, lo que provoca la aparición de bandas de color. El resultado final es mapeado a un framebuff estándar de 8 bits por canal mediante una función de transferencia.

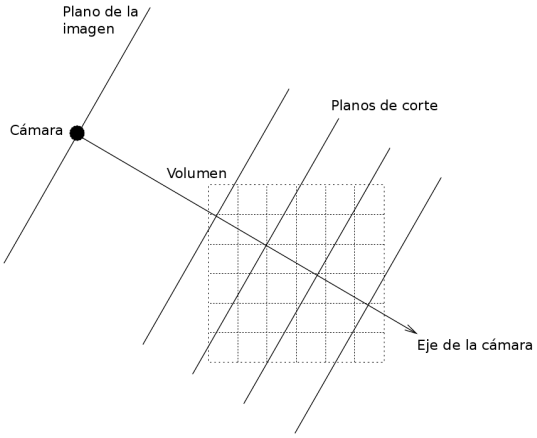

Figura 3.1: Concepto de *slicing*

La visualización por slicing es sencilla de implementar, y ofrece buenos resultados, siempre y cuando la separación entre los cortes sea pequeña. Si la separación es muy grande, puede hacerse cláramente visible, lo que dará resultados visualmente pobres. La separación es más fácil de apreciar cuanto mayor sea el ángulo entre el eje de la cámara y el eje de corte.

Los peores resultados se obtienen si el eje de la cámara es perpendicular al eje de corte. Para evitar esto, se puede cambiar el plano que se usa para renderizar los cortes. El mayor inconveniente es que ésto perjudica gravemente al patrón de acceso a memoria, lo que reduce el rendimiento. Para evitar esta pérdida de rendimiento, se pueden almacenar los datos según los tres ejes de coordenadas, y elegir el más adecuado en cada momento, pero esto tiene el problema de que triplica la cantidad de memoria requerida. Otra opción es renderizar planos paralelos al plano virtual de la cámara, que cortarán el volumen de datos con el ángulo adecuado. Para poder acelerar este método por hardware es necesario que el hardware soporte mapeado de texturas 3D.

No se ha implementado ningún algoritmo de *slicing*, ya que el *raycasting*, que será descrito a continuación, ofrece mejores resultados visuales con un rendimiento adecuado.

3.3. Visualización por *raycasting*

El *raycasting* es una técnica conceptualmente muy sencilla. Consiste en trazar rayos desde la cámara, uno por cada píxel de la imagen. Para cada rayo, se calculan los punto de entrada y de salida del volumen, y se acumulan los valores del volumen a lo largo del camino seguido. Si el rayo no toca el volumen, se toma un color de fondo, normalmente el negro. El valor final obtenido se mapea a un color de 24 bits que pueda ser mostrado en la pantalla; aunque los valores intermedios acumulados a lo largo de la trayectoria del rayo necesitan una gran precisión, igual que en el caso del *slicing*, estos valores sólo se usan durante el cálculo de cada rayo individual, por lo que no hace falta guardarlos en un buffer de acumulación.

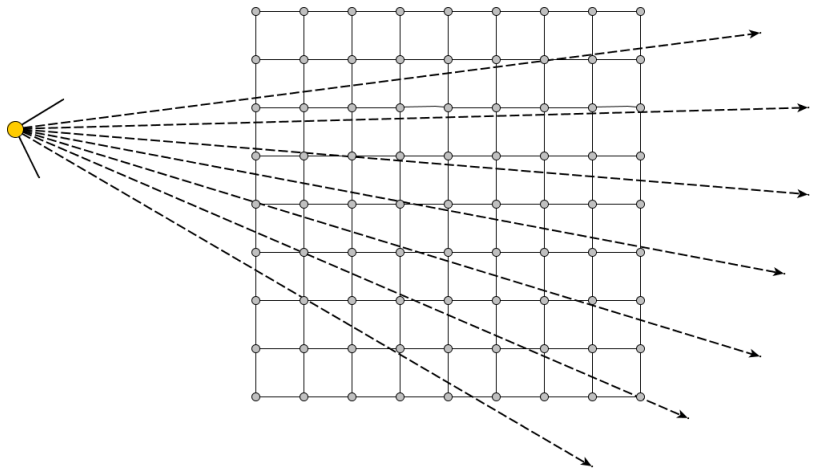

Figura 3.2: Concepto de *raycasting*

El primer requisito, por tanto, es calcular la intersección de un rayo con el volumen. Para ello, definimos la ecuación paramétrica de una recta de origen R_o y dirección R_d, y la ecuación paramétrica de una caja alineada con los ejes de coordenadas, de extremos V_{min} y V_{max}:

$$\left.\begin{array}{l} P = R_o + t \cdot R_d \\ V_{min} \leq P \leq V_{max} \end{array}\right\}$$

Figura 3.3: Ecuaciones parametricas de un rayo y una caja

Se sustituye la ecuación 1 en la 2, y se simplifica:

$$V_{min} \leq R_o + t \cdot R_d \leq V_{max}$$

$$V_{min} - R_o \leq t \cdot R_d \leq V_{max} - R_o$$

$$\frac{V_{min} - R_o}{R_d} \leq t \leq \frac{V_{max} - R_o}{R_d}$$

$$\begin{cases} t_{min} = \frac{V_{min} - R_o}{R_d} \\ t_{max} = \frac{V_{max} - R_o}{R_d} \end{cases}$$

Figura 3.4: Intersección de un rayo con una caja

Si $t_{min} > t_{max}$, el rayo no corta el volumen, y por tanto el valor de ese píxel es 0. Por contra, si $t_{min} \leq t_{max}$, hay que trazar el rayo a lo largo del volumen, empezando con $t = t_{min}$. La separación entre dos pasos tiene que ser constante, para asegurar que el muestreo sea uniforme y evitar artefactos. El trazado termina cuando el rayo sale del volumen ($t > t_{max}$), si el valor acumulado supera un cierto valor máximo, o si se alcanza un número prefijado de iteraciones. La separación entre dos pasos tiene que ser lo suficientemente pequeña como para evitar discontinuidades en el muestreo que provoquen errores visuales.

Una de las ventajas del raycasting es su simplicidad conceptual. Cada rayo es completamente independiente del resto, por lo que es su paralelización es trivial. Esto permite acelerar enormemente el algoritmo usando una GPU. Para comparar el rendimiento hemos escrito dos implementaciones, una de ella puramente software, y la otra acelerada mediante OpenCL. Ninguna de ellas está optimizada. Ambas implementaciones se han probado en un Core i7 920, con 6 gigas de RAM, y una tarjeta gráfica Nvidia GTX275, ejecutando Ubuntu 10.04. Los resultados son los siguientes:

	256×256			512×512			1024×1024		
Dataset	CPU	GPU	Mejora	CPU	GPU	Mejora	CPU	GPU	Mejora
Pierna (600 cortes)	4,30	24,69	5,74×	1,24	11,74	9,47×	0,32	6,76	21,13×
Craneo (361 cortes)	6,10	26,49	4,34×	1,87	14,80	7,91×	0,52	9,97	19,17×
Torso (152 cortes)	10,71	33,43	3,12×	4,21	23,02	5,47×	1,17	14,62	12,50×
Media			$\sim 4,4\times$			$\sim 7,6\times$			$\sim 17,6\times$

Figura 3.5: Comparativa de rendimiento del *raycasting*

Como se puede ver, la versión acelerada por OpenCL supone una mejora de rendimiento media de $\sim 10\times$. La aceleración es mayor cuanto mayor es el número de cortes y la resolución de salida.

Figura 3.6: Imagen renderizada mediante *raycasting*

4 Generación de la malla

4.1. Introducción

En este capítulo se describe el proceso seguido para extraer una malla poligonal a partir de la imagen de TAC. El proceso se divide en cuatro fases:

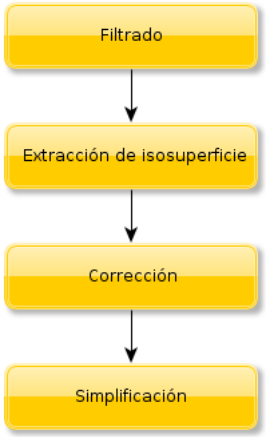

Figura 4.1: Fases del proceso de generación de la malla

En primer lugar, hay que filtrar la imagen para eliminar el ruido que pueda tener. Se consideró el estudio de diferentes filtros, como los gausianos, los bilaterales, o los filtros de difusión. Sin embargo, después de estudiar las imágenes proporcionadas por el TAC, nos dimos cuenta de que la máquina implementa sus propios filtros, que además de eliminar el ruido también permiten realzar aquellos tejidos en los que estamos más interesados. Por tanto, se decidió que era innecesario implementar nuestro propio filtro.

El segundo paso es extraer la propia malla. Para ello hemos usado el algoritmo conocido como *marching tetrahedrons*, muy difundido y ampliamente estudiado. Este algoritmo proporciona resultados de muy alta calidad, con un rendimiento aceptable. Es posible que la malla generada tenga errores impidan su correcta visualización y posterior tratamiento, por lo que el tercer paso es corregir estos defectos. Debido a la alta resolución de las imágenes proporcionadas por el TAC, la malla resultante tendrá un nivel de detalle excesivo, del orden de varios millones de triángulos. El

cuarto y último paso será reducir el número de polígonos a un nivel más manejable, intentando mantener la calidad de la malla. A continuación se explicará cada paso con más detenimiento.

4.2. Extracción de la malla

El primer paso es extraer la isosuperficie que representa el hueso. Si consideramos la imagen como un campo escalar, una isosuperficie es el conjunto de puntos para los cuales la función adopta un valor determinado. Es, en esencia, el análogo 3D de las curvas de nivel en 2D. Existen muchos algoritmos de extracción de isosuperficies, de los cuales los más conocidos son el *marching cubes* y el *marching tetrahedrons*. Ambos algoritmos son muy similares en su funcionamiento, ya que uno es evolución del otro. En ambos casos hay que fijar el umbral de la isosuperficie. Los puntos cuyo valor sea menor quedarán fuera del objeto, y lo que sean mayor dentro. El algoritmo creará una malla poligonal con los puntos que más se ajusten al umbral, interpolando si es necesario.

El marching cubes fue desarrollado en 1985 por Lorensen y Cline, y publicado en 1987 en el SIGGRAPH. Dicho algoritmo trabaja considerando cubos de $2 \times 2 \times 2$ vóxeles. Para cada uno de los vértices del cubo, pueden darse dos casos: que su valor sea menor que el umbral, o que sea mayor. Esto da un total de 2^8 combinaciones. Así mismo, el cubo tiene 12 aristas. Si alguno de los vértices cae dentro del volumen, hay que calcular el punto en el que la superficie corta a cada uno de las aristas. Una vez hecho, se consulta una tabla de 256 entradas, que indica, para cada combinación, los puntos de las aristas que hay que conectar para formar el segmento de malla correspondiente a ese cubo. Cada cubo puede tener entre 0 y 6 triángulos.

Aunque simple y eficiente, el *marching* cubes tiene dos grandes defectos. El primero es que hay casos que dan lugar a ambigüedades, es decir, cubos para los que se pueden generar dos mallas distintas. Resolver estos casos no es trivial, ya que hay que consultar los cubos adyacentes para poder determinar cual de las dos soluciones es la correcta. El segundo problema es que estaba patentado, aunque la patente caducó en el 2005.

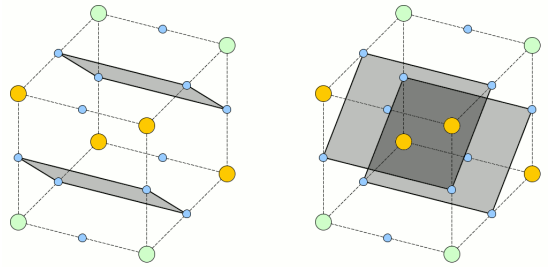

Figura 4.2: Ambigüedades del marching cubes

Ambos problemas llevaron a la creación del *marching tetrahedrons* en el año 1997 por Müller y Wehle. La base es la misma, pero en este caso el cubo se corta por planos paralelos a las diagonales de las caras, dando lugar a seis tetrahedros. Para cada tetrahedro existen $2^4 = 16$ combinaciones, y 4 puntos de corte en las aristas, dando un total de 19 puntos de corte para el cubo. Cada tetrahedro puede tener un máximo de 2 triángulos, por lo que el cubo tendrá entre 0 y 12 triángulos. *El marching tetrahedrons* tiene el inconveniente de que requiere calcular más puntos de corte, y genera más triángulos, pero a cambio la malla es más suave.

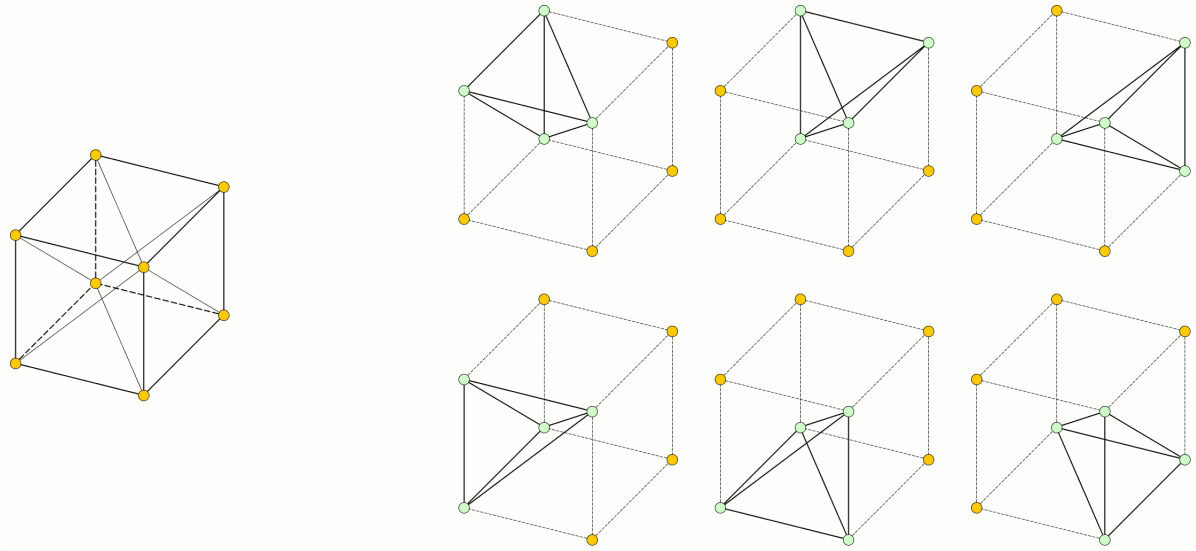

Figura 4.3: División de un cubo en tetrahedros

La tabla de casos es la siguiente:

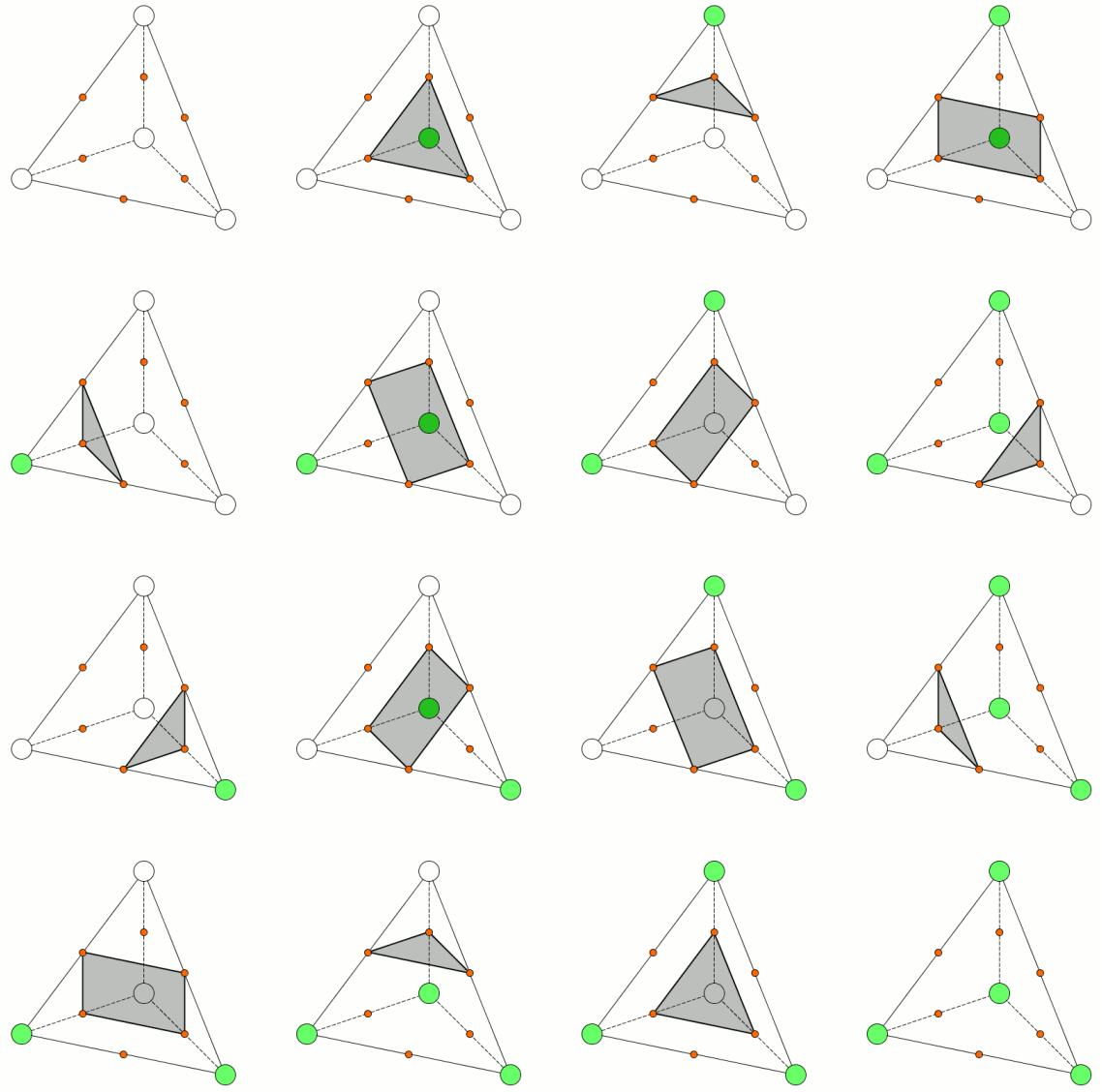

Figura 4.4: Tabla de casos para el "marching tetrahedrons"

El primer paso es calcular las coordenadas de los puntos de corte. Para ello se usa una interpolación lineal. Sean P_A y P_B las coordenadas de los extremos de la arista, y W_A y W_B sus valores, las posición P del punto de corte se calcula como:

$$W_A + t\,(W_B - W_A) = 0$$

$$t = \frac{-W_A}{W_B - W_A}$$

$$P = P_A + t\,(P_B - P_A)$$

Figura 4.5: Cálculo de la posición de los puntos de corte

Los cubos adyacentes comparten los 5 puntos de corte correspondientes a la cara que está en contacto. Durante la generación de la malla hay que tener en cuenta este hecho, para evitar que la malla resultante tenga vértices repetidos. La primera idea que se consideró fue ordenar los vértices de tal manera que se pudiera solapar el cálculo de los vértices de un cubo con los de su vecino. Por ejemplo, si se ordenan los vértices en el eje X, entonces los vértices de la cara derecha de un cubo son los mismos que los de la cara izquierda del cubo que está a su derecha. Sin embargo, mientras que hacer esto para uno de los ejes de coordenadas es fácil, hacerlo para los tres a la vez es muy difícil, ya que hay que hacer tres pasadas por la malla, una por cada eje de coordenadas, reordenando los vértices en cada pasada y eliminando los repetidos.

Al final se optó por una alternativa potencialmente menos eficiente, pero mucho más sencilla de implementar. Como es habitual en estos casos, la malla está formada por dos estructuras de datos: una lista con las coordenadas de los vértices, y una lista de triángulos, que almacena los índices de los vértices que componen cada triángulo. La idea es que, al generar un nuevo vértice, se busca entre los ya existentes. Si ya existía de antes, se devuelve el índice que ya tenía asignado, mientras que si no existía se añade a la lista, asignándole un nuevo índice. Buscar el vértice en una lista es muy lento, ya que la complejidad es lineal. Para acelerar el proceso se ha optado por usar una estructura de ordenación espacial, concretamente un *octree*, lo que nos permite hacer la búsqueda con coste logarítmico. Debido a los errores de aproximación introducidos por el uso de aritmética de coma flotante, la búsqueda es aproximada: se considera que dos vértices son equivalentes si la distancia entre ambos es menor que un determinado valor ε, que en nuestro programa se ha fijado en 10^{-6} metros.

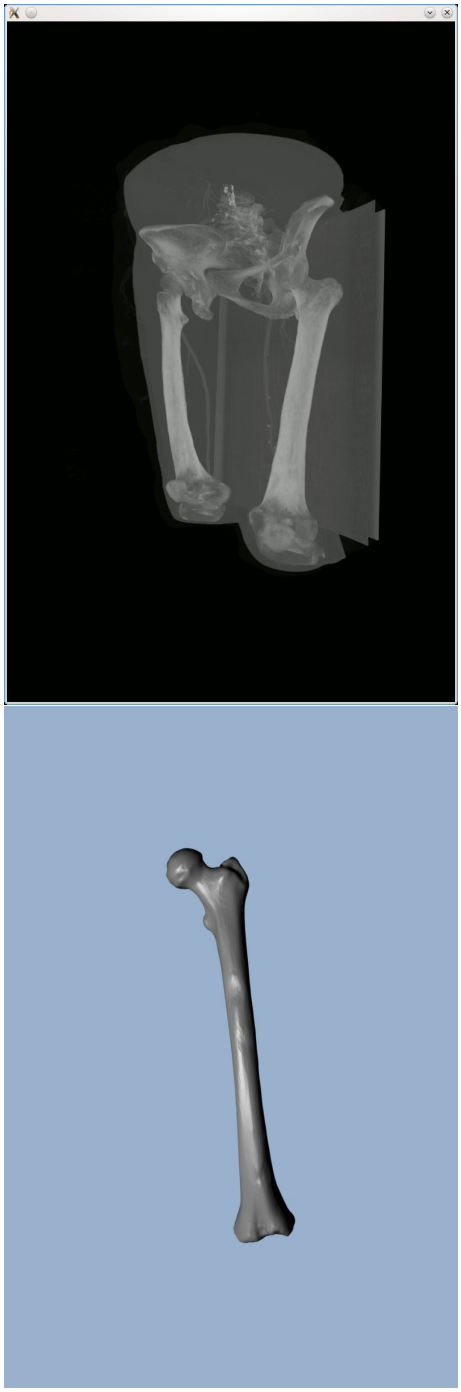

Figura 4.6: Malla resultante de la extracción

4.3. Corrección de defectos topológicos

La malla generada puede tener defectos topológicos. Se trata, por ejemplo, de errores geométricos o de conectividad que hacen que la malla no pueda ser visualizada o procesada adecuadamente. Algunos de los errores que pueden darse son los siguientes:

- Triángulos con orientaciones distintas: debido a errores en la tabla de casos, puede ocurrir que algunos triángulos tengan sentido horario y otros sentido antihorario. Esto hará que, al calcular las normales de los triángulos, unas apunten al exterior del objeto y otras al interior. Esto hará que los cálculos de iluminación sean incorrectos, entre otros problemas. La manera de resolver este problema es muy sencilla: se elige un triángulo inicial al azar, y se propaga su orientación a los triángulos vecinos. Estos a su vez se propagan a sus vecinos, de manera recursiva, hasta que todos los triángulos de la malla han sido visitados. Para propagar la orientación, hay que darse cuenta del hecho de que si un triángulo está unido a otro por medio de la arista AB, en la lista de vértices del triángulo vecino dicha arista debe aparecer como BA.

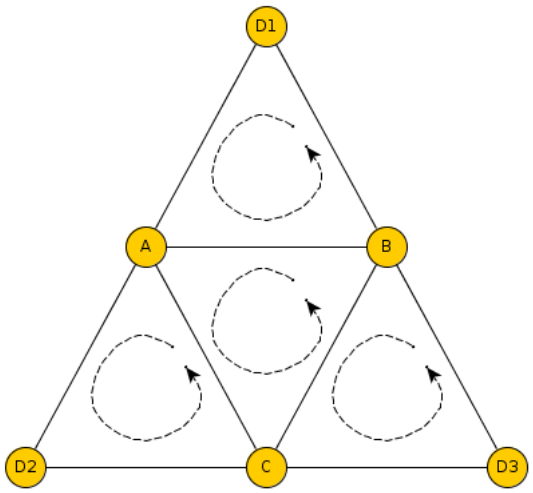

Figura 4.7: Corrección de la orientación de los triángulos

- Triángulos de tamaño 0: si durante la generación de la malla dos de los vértices de un triángulo están demasiado cerca, colapsarán en uno solo, por lo que el triángulo se convertirá en un segmento de recta. Peor aún, si los tres vértices están demasiado cerca, el triángulo colapsará en un único punto. La solución es detectar los triángulos que tengan vértices repetidos, y eliminarlos de la lista.
- Vértices repetidos: puede darse el caso de que en la lista de vértices aparezca un mismo vértice con varios índices distintos. Esto provocará que la malla requiera más capacidad de almacenamiento y potencia de proceso de la necesaria, y dificultará la aplicación de los algoritmos de simplificación que veremos posteriormente. La manera de resolver este caso es utilizar una estructura de ordenación espacial que nos permita detectar los vértices repetidos y eliminarlos; como en nuestra implementación ésto es lo que se hace durante la fase de generación de la malla, este caso no puede darse.

- Que la malla se corte a sí misma: este caso no puede darse al procesar una imagen de TAC, pero podría suceder si tomáramos los datos de un escáner láser, por ejemplo.
- Que la malla no esté cerrada: si la malla corta el borde del volumen de datos, quedará abierta. La manera más sencilla de evitar que esto suceda es añadir un borde negro alrededor del volumen antes de extraer la malla.

Aunque la implementación de estos algoritmos no es complicada, si que consume bastante tiempo, ya que requiere del uso de complejas estructuras de datos de conectividad topológica como *winged-edge*. Después de buscar y estudiar diferentes herramientas que nos permitieran tratar este tipo de casos, nos decantamos por la biblioteca JMeshLib. Esta biblioteca implementa las estructuras de datos mencionadas, y además proporciona varias herramientas de ejemplo que son perfectas para nuestro cometido.

4.4. Simplificación de la malla

La malla obtenida del TAC tiene un nivel de detalle muy alto, de varios millones de triángulos. Usar una malla con tanta resolución para propósitos de visualización resulta excesivo, por lo que el siguiente paso es simplificarla. El algoritmo más usado para estos propósitos es el colapso de aristas, por su simplicidad y robustez. Este algoritmo consiste en buscar pares de vértices adyacentes y sustituirlos por un único vértice intermedio. De esta manera se elimina una arista y dos triángulos. Este proceso se repite hasta que la malla resultante tiene el número de triángulos deseados.

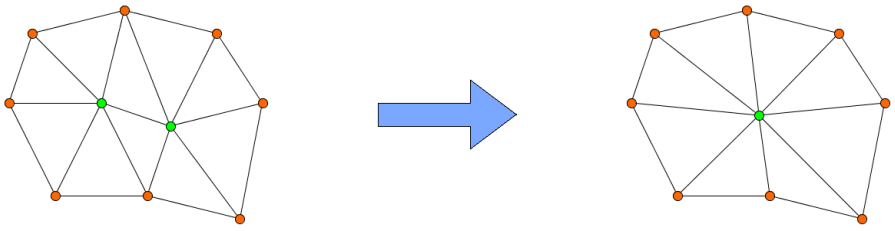

Figura 4.8: Ejemplo de colapso de arista

La posición del nuevo vértice se puede calcular como la posición intermedia de los vértices que formaban la arista, o como una combinación lineal de ambos. La parte más complicada del algoritmo es determinar qué arista hay que eliminar en cada iteración. Para ello existen diferentes criterios:

- Eliminar la arista más corta: este criterio es muy sencillo, pero los resultados que proporciona no son muy buenos.

- Métrica de error cuadrática: asocia una matriz de error a cada vértice; el error de una arista es la suma de los errores de los vértices que la forman, y en cada iteración se elige la arista con menor error absoluto. La posición del vértice intermedio se calcula minimizando la métrica cuadrática.
- Métrica de Lindstrom-Turk: mantiene una cola de prioridad con el coste de cada arista, y en cada iteración elige la arista de menor coste. Si no se puede colapsar una arista sin violar la topología de la malla (por ejemplo, haciendo que aparezcan huecos), se asigna un coste infinito a la arista. El coste de cada arista se calcula resolviendo un problema de optimización que intenta preservar el volumen del objeto.

Al igual que en la fase anterior, hemos resuelto esta fase mediante la biblioteca JMeshLib, que proporciona una herramienta para la simplificación de mallas. Dicha herramienta acepta como parámetro de entrada el porcentaje de vértices que queremos conservar en la malla final. En nuestra pruebas, se pueden conservar sólo un 2 % de los vértices (una reducción de 40×) manteniendo una buena calidad en la malla resultante.

Figura 4.9: Malla original y simplificada

5 Etiquetado del hueso

Una vez que se tiene generada la malla del hueso, hay que marcar una serie de puntos de interés. Estos puntos sirven para definir la forma del hueso, y servirán de base para el algoritmo de registrado. Esta fase debe ser realizada manualmente por un experto. Es un proceso muy delicado, porque hay que verificar que la posición de los puntos está marcada correctamente. La lista de puntos que se ha definido es la siguiente:

- Cabeza: parte del fémur que encaja en la articulación de la cadera.
- Cuello: une la cabeza con el tronco del fémur, las fracturas de cuello son las más comunes en el fémur.
- Trocánter mayor: elevación irregular cercana a la cabeza, situada hacia el exterior del cuerpo.
- Trocánter menor: elevación irregular en posición inferior a la cabeza, situada hacia el interior del cuerpo.
- Diáfisis: parte central de los huesos largos, las fracturas diafisarias también son comunes, y muy graves.
- Cóndilo medial: protuberancia cóncava que encaja con la tibia, rótula y peroné para formar la rodilla. Está situada en la parte interior de la pierna.
- Cóndilo lateral: protuberancia cóncava que encaja con la tibia, rótula y peroné para formar la rodilla. Está situada en la parte exterior de la pierna.
- Epicóndilo medial: protuberancia ubicada sobre el cóndilo medial, donde se insertan los músculos.
- Epicóndilo lateral: protuberancia ubicada sobre el cóndilo lateral, donde se insertan los músculos.
- Superficie patelar: superficie cartilaginosa sobre la que se desliza la rótula.

En la siguiente imagen se muestra la posición de cada punto de interés.

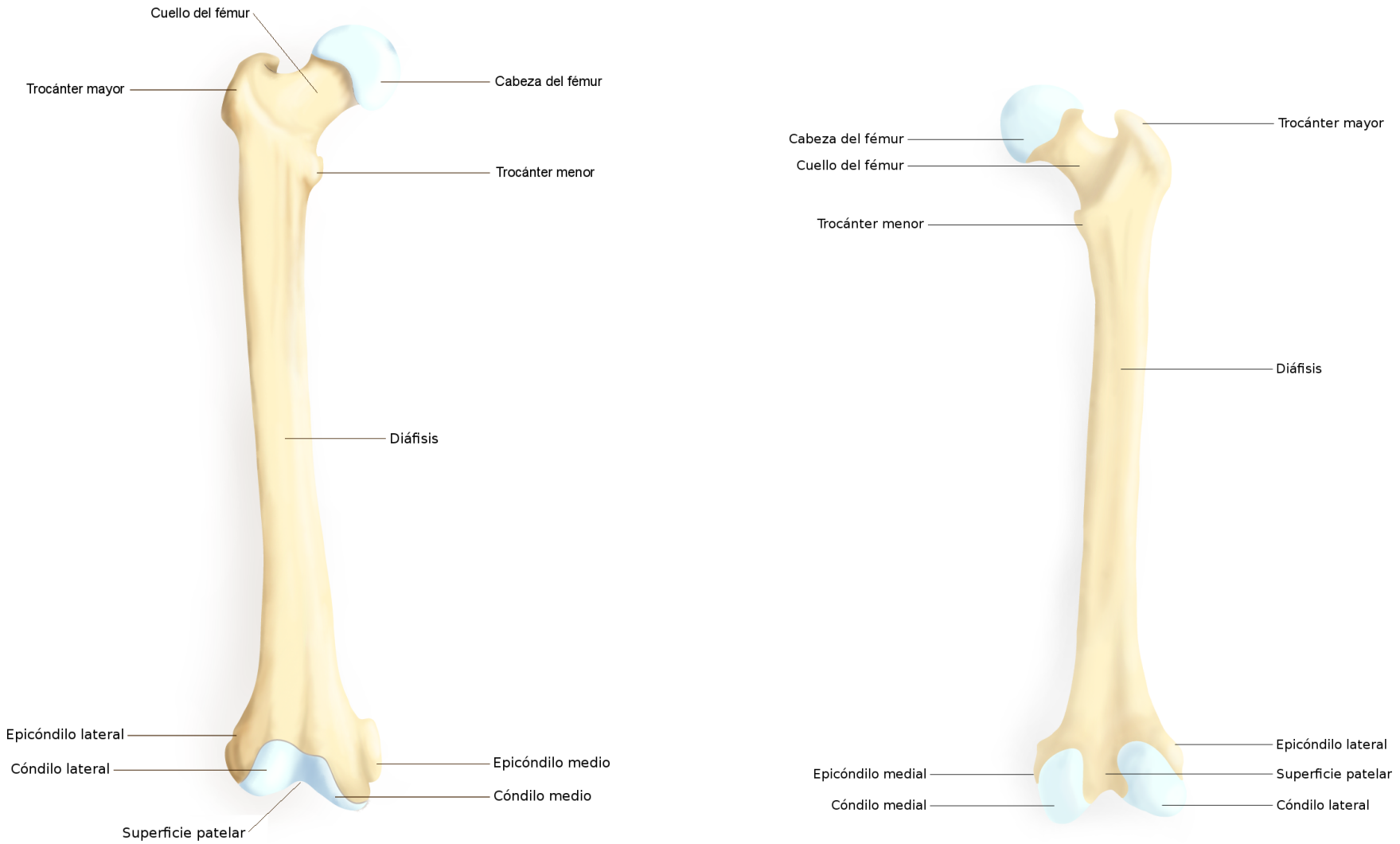

Figura 5.1: Puntos de interés del fémur

6 Creación del modelo virtual

6.1. Introducción

La segunda etapa del proyecto es la implementación de un algoritmo que permita obtener un modelo 3D del paciente combinando radiografías del propio paciente con los modelos contenidos en la base de datos. Dicho algoritmo consta de varias fases:

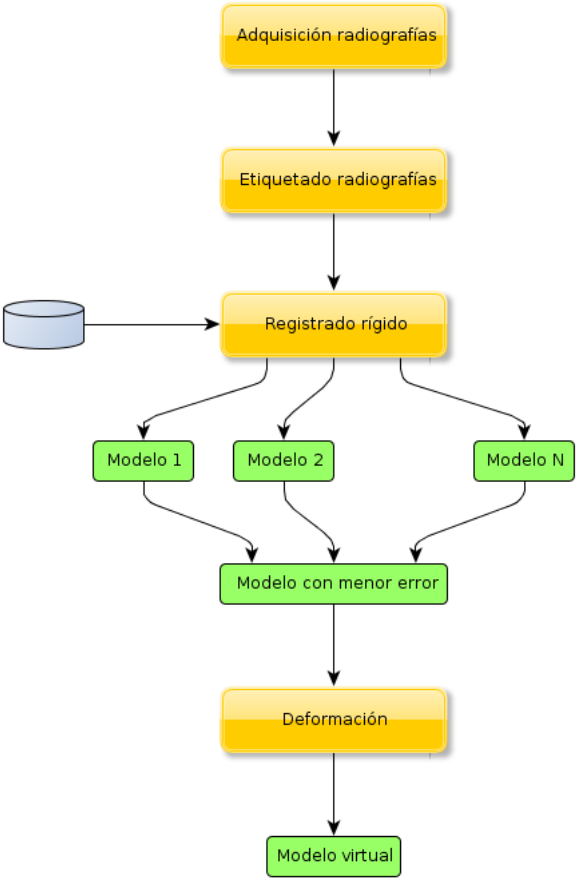

Figura 6.1: Proceso para la generación del model virtual

En primer lugar, cuando se recibe a un paciente con fractura de fémur en el servicio de salud, se le hacen dos radiografías, una en posición anteroposterior (frontal) y otra lateral. Estas radiografías son usadas por el cirujano para hacerse una idea de la colocación de los fragmentos. Para el funcionamiento del algoritmo, el cirujano debe

marcar los puntos de interés en ambas fracturas. A continuación, el algoritmo calcula la posición de los puntos en el espacio 3D, y efectúa un registrado rígido contra la base de datos. El registrado se hace de manera independiente para cada modelo contenido en la base de datos. El resultado es, para cada modelo, la transformación que mejor hace corresponder los datos del paciente con los del modelo, y el error de dicha transformación. Se elige el modelo con menor error como modelo de referencia, y se pasa a la siguiente fase.

En la segunda fase, el modelo de referencia se deforma para ajustarlo a la anatomía del paciente. Para ello, se ha implementado un algoritmo que deforma el propio espacio de coordenadas, mediante el uso de puntos de control arbitrarios. El algoritmo calcula el desplazamiento de cada vértice de la malla como una combinación lineal del desplazamiento de cada punto de control, ponderada según la distancia del vértice a cada punto.

Ambas fases serán detalladas a continuación.

6.2. Concepto de registrado

El proceso de registrado busca establecer similitudes entre dos o más imágenes. En esencia, se buscan patrones comunes en ambas imágenes, y se calculan los parámetros matemáticos que permiten transformar una imagen en otra. Normalmente, dichas imágenes están tomadas desde posiciones distintas, o con diferentes sensores; el objetivo del registrado es, en este caso, combinar las imágenes para obtener más información de la que puede dar cada imagen por separado. Los algoritmos de registrado se pueden clasificar según tres parámetros:

- Según el origen de los datos:
  - Monomodal: todas las imágenes proceden de la misma fuente de datos; este es el caso, por ejemplo, de las aplicaciones que permiten combinar varias fotografías para formar una panorámica.
  - Multimodal: las imágenes proceden de orígenes distintos; por ejemplo, se pueden combinar imágenes de TAC (que distingue mejor los tejidos duros) y RMN (que distingue mejor los tejidos blandos), o combinar imágenes en diferentes zonas del espectro electromagnético.
- Según el criterio de registrado:
  - Basado en el propio valor de los píxeles o vóxeles de la imagen, es decir, su intensidad o su color.
  - Basado en descriptores de alto nivel, como puntos, contornos, formas, etc., extraídos mediante un algoritmo de segmentación.

- Según el tipo de transformaciones permitidas:
  - Rígido, si sólo se permiten transformaciones afines, como traslaciones, rotaciones, escalados, etc.
  - Deformable, si se permiten transformaciones no lineales que modifiquen la forma de los objetos.

El algoritmo elegido, que será explicado en la siguiente sección, es un registrado rígido multimodal basado en puntos.

6.3. Registrado rígido

La primera fase de la generación del modelo es el registrado rígido. En primer lugar, el cirujano marca la posición de los puntos de interés en tres radiografías, tomadas en posición anteroposterior (90°), oblícua (45°), y lateral (0°). A continuación se combinan las tres vistas para calcular la posición de los puntos en 3D. El resultado es una nube de puntos, a la que llamaremos P, que será registrada contra la nube de puntos del modelo de referencia, a la que llamaremos R. El resultado será una matriz de transformación T que, aplicada a P, devuelva una nube de puntos lo más parecida posible a R.

Hemos implementado el registrado mediante el algoritmo ICP (*iterative closest point*). Dicho algoritmo parte de una transformación inicial, que se va refinando progresivamente con cada iteración sucesiva. El algoritmo estima los parámetros de traslación y rotación, por lo que considera 6 grados de libertad.

Figura 6.2: Ecuaciones del algoritmo ICP

En cada iteración se realizan los siguientes pasos:

- Se aplica la matriz de transformación a cada punto de P, dando la nube $P\prime$.
- Se asocia cada punto de $P\prime$ con el punto de R más cercano.
- Se calculan los términos del sistema lineal.
- Se resuelve el sistema lineal, obteniendo los nuevos parámetros de traslación y rotación.
- Se calcula la nueva matriz de transformación.

El paso más lento es la asociación de puntos. El algoritmo más sencillo implicaría comparar cada punto de P con todos los puntos de R, pero la complejidad de este algoritmo sería O(n^2). Una solución más eficiente es usar de indexación espacial

para R, como puede ser un octree, un árbol KD, o una rejilla. De esta manera la complejidad se puede reducir a O(log n).

El mayor inconveniente del algoritmo ICP es la estimación inicial de la pose. El algoritmo necesita conocer, de alguna manera, cuál es la orientación inicial del objeto. Si dicha estimación inicial no es correcta, es bastante probable que el algoritmo no converja y que los resultados obtenidos sean incorrectos. Normalmente, la estimación de la pose se realiza mediante sensores acoplados al objeto que se quiere medir o al instrumento de medición. En nuestro caso, tenemos la ventaja de que tanto los modelos de TAC como las radiografías se toman en posiciones similares, con el paciente tumbado en una camilla. Por tanto, podemos tomar como inicial la transformación identidad.

Lo explicado hasta ahora es sólo el registrado de la nube de puntos del paciente con la de un modelo. Sin embargo, nuestra aplicación dispondrá de una base de datos con multitud de modelos, por lo que habrá que registrar los datos del paciente con cada uno de los modelos, y elegir el que ofrezca menor error. El error se calcula como la media de las distancias de los puntos de la nube transformada del paciente a la nube del modelo:

$$\varepsilon = \frac{1}{n}\sum_{i=1}^{n} \|T \cdot P_i - R_i\|$$

Figura 6.3: Métrica de error para el algoritmo ICP

Se elegirá como modelo de referencia el modelo para el cual se encuentre la transformación que minimice el error. Este modelo será usado como base para la deformación, que será explicada en la siguiente sección.

6.4. Deformación de la malla

Una vez que se ha seleccionado el modelo de referencia, hay que deformarlo para adaptarlo a los datos del paciente. En primer lugar hay que aplicar la transformación obtenida en el registrado rígido. Como la transformación se ha calculado para transformar los datos del paciente a los del modelo de referencia, y lo que queremos es generar el modelo virtual del paciente, hay que aplicar la inversa de la matriz de transformación a R. De esta manera, se obtiene un primer modelo aproximado. A continuación, se desplazan los vértices de dicho modelo para deformarlo.

Durante nuestro estudio, se consideró en primer lugar usar un algoritmo basado en NURBS u otro tipo de *splines*. Descartamos dicha opción porque, usando *splines*, son los puntos de control los que definen la forma de la malla. Sin embargo, en nuestro caso la malla procede del exterior, y los puntos de control se añaden *a posteriori*.

La solución propuesta es la siguiente. Cada punto de control está definido por sus coordenadas en el espacio y un vector de traslación, siendo las coordenadas R_i y la traslación $P_i - R_i$. La idea es calcular, para cada punto del espacio, el vector de traslación correspondiente, que será una media de los vectores de los puntos de control, ponderada en función de la distancia del punto a cada punto de control. Para que la deformación sea correcta, establecemos tres restricciones: los pesos deben ser números reales positivos, la suma de los pesos debe ser 1, y si $x = R_i$, entonces $\omega_i = 1$ y $\omega_{j \neq i} = 0$.

$$\omega_i \in [0, 1]$$

$$\sum_{i=1}^{n} \omega_i = 1$$

$$x = R_i \Rightarrow \begin{cases} \omega_i = 1 \\ \omega_{j \neq i} = 0 \end{cases}$$

Figura 6.4: Restricciones del algoritmo de deformación

El método para calcular los pesos es el siguiente. En primer lugar, se calcula la fuerza ejercida por cada punto de control, que vendrá dada por el inverso del cuadrado de la distancia. El valor de estas fuerzas será siempre positivo. A continuación, se normalizan las fuerzas para que sumen 1:

$$F_i(x) = \frac{1}{|x - R_i|^2}$$

$$\omega_i(x) = \frac{F_i(x)}{\sum_{i=1}^{n} F_i(x)}$$

Figura 6.5: Cálculo de la fuerza ejercida por un punto de control

En este paso hay que tener cuidado, ya que cuando x es igual a alguno de los

puntos de control, se produce una indeterminación. La causa de esto es que la distancia de x al punto es 0, la fuerza de ese punto es $1/0 = \infty$, y la fuerza normalizada es ∞/∞. Al calcular el resultado, la división dará un *NaN*; la solución consiste, por tanto, en detectar si alguno de los valores es un *NaN*, y si es así, sustituirlo por un 1. Una vez normalizados los pesos, se multiplican por los vectores de traslación para calcular la posición final del vértice.

$$\forall x \in \mathbb{R}^3 \Rightarrow x\prime = x + \sum_{i=1}^{n} \omega_i \left(P_i - R_i\right)$$

Figura 6.6: Cálculo de la posición final de un vértice

7 Conclusiones y trabajos futuros

Queda como trabajo pendiente la evaluación y validación del algoritmo de deformación. La razón por la que esta tarea aún no ha sido llevada a cabo es porque no hemos dispuesto de los datos de las exploraciones de TAC hasta hace una semana, por lo que no ha dado tiempo a estudiarlos convenientemente. La metodología propuesta es la siguiente:

- Seleccionar un conjunto de sujetos de muestra. Debido a la elevada dosis de radiación usada por el TAC, se ha elegido trabajar con cadáveres.
- Para cada sujeto, realizar una exploración de TAC y las radiografías del fémur.
- Extraer el modelo 3D a partir de los datos del TAC. Este modelo servirá como referencia.
- Usar las radiografías para construir un modelo virtual a partir de la base de datos.
- Comparar ambos modelos, calculando el error entre ellos. El error se calcula mediante la distancia de Haussdorf.
- Puesto que ambos modelos se han generado a partir del mismo paciente, el error debería ser pequeño. Si no es así, nuestro algoritmo no es correcto y habría que replantearse el algoritmo de deformación.

Como conclusiones, podemos enumerar las siguientes:

- La creación de la base de datos es un proceso muy lento y trabajoso, sobre todo debido a que hay que tomar un elevado número de sujetos para obtener resultados representativos. La ventaja es que es un proceso offline, por lo que una vez obtenidos los resultados, no hay que volver a hacerlo a menos que se quiera ampliar el número de muestras. Además, dichos resultados pueden servir como base para futuros proyectos.
- La generación del modelo virtual es muy rápida. El algoritmo ICP converge rápidamente, y dado el reducido número de puntos de interés usados, cada iteración es muy corta. Por otro lado, la implementación es muy fácil de paralelizar:

    - En la fase de registrado rígido, se deben registrar los datos del paciente contra cada uno de los modelos de la base de datos. Cada uno de estos modelos es completamente independiente, por lo que se pueden registrar en paralelo, y elegir el mejor al final.
    - En la fase de deformación, el cálculo de la posición de cada vértice de la malla es completamente independiente del resto.

- La parte más tediosa del proceso es obligar al cirujano a marcar los puntos de interés. Para asegurar la calidad del modelo generado, la posición de los puntos debe estar marcada con mucha exactitud.

El presente trabajo es parte de un proyecto mayor. Se proponen los siguientes trabajos futuros:

- Expandir la base de datos a otros huesos. En éste trabajo sólo se ha considerado el fémur, por ser el hueso más grande del cuerpo, y por la gravedad que una fractura de fémur entraña. De cara a aumentar el rango de aplicaciones del sistema, sería interesante crear modelos del resto de huesos largos; esto es, húmero, cúbito y radio, y tibia y peroné.

- En el sistema propuesto actualmente, el marcado de los puntos de interés debe hacerse manualmente. Se podría estudiar la viabilidad de desarrollar un sistema que haga uso del conocimiento de la morfología del hueso para su etiquetado automático.

- El algoritmo de deformación propuesto es conceptualmente sencillo, ya que está basado únicamente en los puntos de control. Se deberían estudiar algoritmos que hagan uso de descriptores de más alto nivel, como contornos. También sería interesante considerar el uso de algoritmos basados en la discretización del espacio, como los métodos por elementos finitos. La complejidad de este tipo de métodos ha hecho que queden fuera del ámbito de este trabajo.

- Para la reconstrucción virtual hemos usado radiografías de huesos intactos. Sin embargo, si queremos poder aplicar el sistema como asistente en el tratamiento de fracturas, es necesario que pueda trabajar con radiografías de fracturas. Para ello, en primer lugar habría que segmentar la radiografía para identificar los fragmentos, reensamblar el hueso, generar el modelo virtual, y por último fragmentarlo para obtener un modelo virtual de la fractura, con todo los fragmentos en sus lugares correspondientes.

8 Publicaciones

- José Franco Campos, José B Pagador Carrasco, José Luis Moyano Cuevas, José Moreno, Pablo Bustos, Ramón Leal, Francisco M. Sánchez Margallo, Minimally Invasive Surgery Centre Jesus Uson, Spain. An approach to a new fluoroscopic X-ray image to a CT-like atlas registration algorithm of long bone fracture sites. 22nd International Conference of the Society for Medical Innovation and Technology (SMIT), 2-4 September 2010, Trondheim, Norway.

9 Bibliografía

- Formato DICOM.
- Origen de las unidades Houndsfield.
- Algoritmo de slicing.
- Algoritmo de raycasting.
- Raycasting en la GPU.
- Marching cubes.
- Marching tetrahedrons.
- Simplificación de mallas.
- Corrección de defectos topológicos.
- Algo de medicina.
- Registrado en general.
- Registrado rígido (ICP).
- Deformación.