INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Escuela Técnica Superior de Ingeniería Informática

Curso académico 2008-2009

Proyecto Fin de Carrera

Visualizador 3D interactivo para laboratorios de análisis de marcha

Autor: David Muelas Sahagún

Tutor: José María Cañas Plaza

A mis padres, por su apoyo incondicional para que llegase hasta aquí.

Agradecimientos

En primer lugar quisiera agradecer a mi tutor, Jose María, todo el esfuerzo, tiempo y dedicación depositada en mí. Nunca habría creído que iba a recibir tanto apoyo por parte de un tutor.

También quiero dar las gracias a todo el Grupo de Robótica por la buena compañía y el apoyo demostrado; a Julio, por toda la ayuda prestada, solicitada o no; a Gonzalo, por la buena compañía que ha sido en este tiempo; y a todos los demás: Jose Antonio, Javi, Antonio, Darío y Pablo. Y a todos los que me olvido. Gracias a todos por este año inolvidable.

Por último, quiero dar mi mayor agradecimiento a Marina. Ha sido fuente inagotable de comprensión, consejos y apoyo durante todo este año y sin ella no sé si hubiera podido cumplir mi objetivo.

Resumen.

La *captura de movimiento* es una técnica para almacenar los movimientos tridimensionales digitalmente, usada en los ámbitos del cine, los videojuegos y el deporte. Otro de los usos de esta técnica está relacionada con la medicina. En este área la captura de movimiento forma parte de sistemas de medición para el diagnóstico de las enfermedades del sistema locomotor y neuromuscular, como la parálisis cerebral. El sistema de estos *laboratorios de análisis de marcha* permite la recolección simultánea de datos e imágenes, y luego de un adecuado procesamiento se presenta la información en videos y gráficos clínicos. Sin embargo, el formato de presentación de la información se presenta de forma estática.

El presente proyecto ha consistido en el desarrollo de una herramienta gráfica de ayuda al médico para el diagnóstico de patologías en el laboratorio de análisis de movimiento. Este centro realiza labores de evaluación de pacientes mediante el uso de la técnica de *motion capture*. El objetivo principal del proyecto es crear un visualizador interactivo de un modelo tridimensional de un esqueleto que permita reproducir el movimiento, capturado previamente, de un paciente. Adicionalmente, la interfaz gráfica proporciona una serie de ventajas que ayudarán al médico en el proceso de diagnóstico, como la posibilidad de pausar el movimiento del modelo y mostrar la gráfica de movimiento de una articulación en concreto en la que se desee profundizar.

La aplicación ha sido desarrollada en colaboración con el laboratorio de análisis de movimiento del Hospital Niño Jesús, bajo la plataforma *jde.c* e implementada en forma de hebras iterativas o *esquemas*, y se ha creado un entorno gráfico basado en las bibliotecas *OpenGL* y *GTK*.

Índice general

Índice de figuras

Capítulo 1
Introducción

En este primer capítulo vamos a tratar de modo genérico la relación entre la tecnología y la medicina. Más concretamente presentaremos con más detalle los laboratorios de análisis de marcha, que son el marco en el que sitúa el software desarrollado en el presente proyecto fin de carrera.

1.1. Tecnología sanitaria

Uno de los factores más decisivos en el desarrollo humano ha sido el constante avance del conocimiento científico. Desde la Revolución Industrial hasta nuestros días, dicho avance se ha ido acelerando gradualmente, de manera que en la actualidad la tecnología está al alcance de la mayor parte de la humanidad de muy diversas formas. Muchas ramas científicas han surgido a raíz de este progreso y una de ellas es la *informática*, que ya forma parte de nuestras vidas. Gran parte de nuestra sociedad se beneficia de los progresos de esta disciplina, que se aplica a muchas actividades humanas, como la gestión de negocios, la industria, las comunicaciones, la física, la química o el arte.

Las ciencias de la salud no son ajenas a este hecho y se han visto beneficiadas significativamente de las nuevas tecnologías. En la actualidad, los médicos pueden contar con un gran número de nuevos instrumentos, herramientas, aplicaciones y técnicas que facilitan su trabajo, además de permitir la realización de operaciones y tratamientos tan complejos o peligrosos que sin ellos resultarían imposibles de llevar a cabo. De hecho, muchas especialidades médicas han surgido con la aparición de ciertos avances tecnológicos, y en platicular procedentes de la informática y la robótica.

Por ejemplo, podemos mencionar los avances que la informática ha incorporado a los informes médicos. No hace falta acceder a las más altas tecnologías para comprobar

que la tecnología informática está ayudando de forma notoria en la práctica médica del día a día. La digitalización de expedientes en la Seguridad Social es una realidad que se está llevando a cabo en nuestro país y ello conllevará un aumento la confidencialidad de los documentos y la accesibilidad de los datos, así como un ahorro de recursos y de espacio físico para su archivo.

Gracias a diversas maquinarias, los médicos pueden diagnosticar con mayor precisión a los pacientes. Sin tecnología, los médicos tenían que depender de sus propios medios para analizar los males; ahora disponen de radiografías, electrocardiogramas, escáneres (TAC), ecografías, análisis de sangre y orina, biopsias, electromiografías y un largo etcétera.

Un buen ejemplo de cómo la tecnología se integra y facilita la práctica médica lo tenemos en el caso de las laparoscopias. Estas técnicas de diagnóstico consisten principalmente en introducir un tubo óptico dentro de la cavidad pélvica-abdominal del paciente con el objetivo de poder ver y analizar el interior. A través de una fibra óptica, por un lado se transmite la luz para iluminar la cavidad, mientras que se observan las imágenes del interior con una cámara conectada a esta fibra.

Incluso dentro del ámbito de las laparoscopias, una nueva técnica ha surgido gracias a los avances tecnológicos: la artroscopia. Esta técnica quirúrjica puede aplicarse en todas las articulaciones, aunque donde más se ha desarrollado es en la rodilla y en el hombro. La artroscopia se practica por medio de mínimas incisiones y, gracias a un sofisticado sistema de microcámaras y microinstrumental, junto a su poca agresividad, permite al paciente que se recupere rápidamente con un índice bajo de complicaciones y un menor dolor postoperatorio.

Otro ámbito tecnológico de utilidad en la medicina es la robótica, como por ejemplo el robot *DaVinci* (ver figura 1.1). La compañía *Intuitive Surgical* recibió la aprobación regulatoria de Norteamérica, Europa y Asia para que su robot *DaVinci* pueda asistir a los cirujanos durante procedimientos de cirugía invasiva mínima. Integrando la tecnología robótica con la habilidad y destreza del cirujano, estos sistemas permiten a los cirujanos operar con una precisión jamás experimentada con anterioridad. De hecho, muchos procedimientos que actualmente se ejecutan mediante técnicas laparoscópicas convencionales pueden realizarse de un manera más rápida y fiable con estos robots.

Figura 1.1: El robot cirujano Da Vinci.

La medicina también se ha visto apoyada por la tecnología dentro del marco de la ortopedia. Gracias a los avances tecnológicos, las prótesis ofrecen cada día mejores prestaciones y comodidades para los pacientes que hayan pérdido alguna extremidad o sufran problemas de cadera. Estos últimos casos, personas que sufren de artrosis de cadera o artritis reumatoide pueden en la actualidad optar por reemplazar su cadera por una articulación artificial, con el fin de eliminar los terribles dolores producidos por dichas patologías. Por ejemplo, la empresa *Ossur* ha desarrollado prótesis como la *Cheetah Flex-Foot*, o la *Rheo Knee* (figura 1.2), que son máquinas que pueden llegar a incorporar procesador, memoria, sensores y hasta Bluetooth, con el objetivo de poder optimizar el movimiento el usuario.

Figura 1.2: Prótesis Rheo Knee y Cheeta Flex-Foot.

1.2. Laboratorios de análisis de marcha

Uno de los sistemas tecnológicos de gran utilidad en el area de la traumatología son los llamados *laboratorios de análisis de marcha* o *de movimiento*. Básicamente, estos laboratorios son un sistema de evaluación orientado al estudio analítico del movimiento y sus efectos durante la acción de caminar. El sistema permite la recolección simultánea de datos e imágenes en tres dimensiones y en tiempo real de un paciente que sufra un trastorno de la marcha, durante el caminar. Estos datos, tras su adecuado procesamiento, presentan la información en vídeos clínicos y gráficos comparados. De esta forma se facilita el diagnóstico de patologías del sistema nervioso o motor y su seguimiento tras el tratamiento.

Físicamente, un laboratorio de análisis del movimiento típico comprende un conjunto de varias cámaras de video infrarrojas de alta definición conectadas a un ordenador con el software necesario para poder procesar todos los datos obtenidos a través de las cámaras. También se necesitan una serie de marcadores adhesivos reflectantes de infrarrojos que serán colocados en puntos específicos del cuerpo del paciente para que las cámaras puedan captar su movimiento (figura 1.3). Adicionalmente, el laboratorio de análisis de marcha puede incluir diversos sensores electromiográficos para medir la activación de los músculos y un pasillo central por donde camina el paciente, con sensores de fuerza en el suelo con los que medir los apoyos realizados con los pies. En la figura 1.3 se puede observar el ejemplo de un laboratorio de análisis de movimiento.

Figura 1.3: Un laboratorio de análisis de marcha y un ejemplo de proceso de captura de movimiento.

Esta tecnología tiene muchos beneficios tanto para los pacientes como para el médico usuario, ya que ofrece aspectos que con la simple vista o la exploración física pueden pasar inadvertidos. Ofrece una serie de datos objetivos, con registros ópticos y gráficos, que van más allá de la impresión visual que pueda sacar un ser humano y ayuda a evitar errores de interpretación. Además, estos datos objetivos pueden ser archivados y comparados con otros datos, lo que permite analizar la evolución y los efectos de un determinado tratamiento al que se someta a un paciente. Asimismo, la información obtenida mediante este estudio brinda la posibilidad de planificar correctamente los programas de rehabilitación, realizar una planificación quirúrgica de alta precisión orientada al menor número de internaciones posibles y diseñar programas de rehabilitación postoperatoria de gran efectividad.

El objetivo de este laboratorio es el tratamiento de niños y adultos con trastornos de la marcha, como afecciones del sistema locomotor o malformaciones congénitas, aunque también tiene una gran utilidad en el ámbito deportivo.

Proceso de análisis de un paciente

Para llevar a cabo sesiones de estudio en un laboratorio de análisis de marcha es necesario calibrar previamente las cámaras, de modo que el software pueda ser capaz de triangular correctamente la información de las cámaras, identificar cada uno de los marcadores infrarrojos y asociarlos a su correspondiente lugar en la figura humana.

Tras la calibración, se colocan los marcadores en el tren inferior del paciente, el cual camina por una zona de paseo central a la que enfocan las seis cámaras, y se realiza la captura de movimiento. La visión conjunta de dichas cámaras detecta cada uno de los marcadores en el espacio tridimensional. Típicamente, el software integrado recoge el movimiento capturado por las cámaras, selecciona un solo *paso representativo* e identifica la posición de cada uno de los marcadores del paciente en cada instante. A menudo, el cuerpo del paciente ocultará algunos de los marcadores a una o más cámaras, pero mientras dos cámaras puedan captar el marcador se podrá triangular su posición.

Una vez realizada la captura, el software se encarga de procesar los datos, calcular las posiciones de las articulaciones, y preparar los datos de salida. A continuación, los

datos de salida se muestran al usuario en forma de una visualización simulada de una figura que se mueve acorde a los datos obtenidos del paciente. En la figura 1.4 podemos ver un ejemplo de esta visualización.

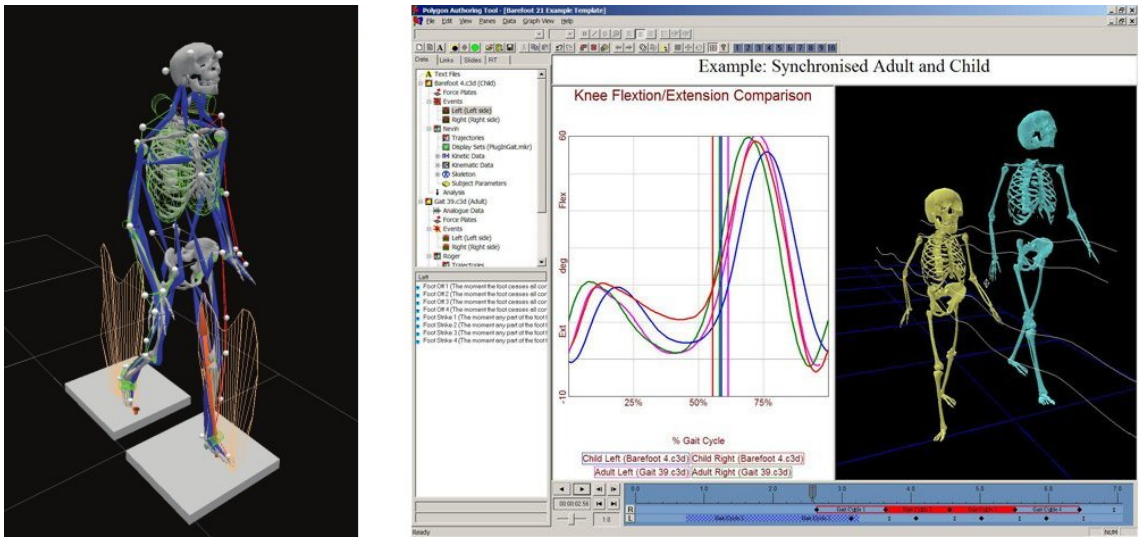

Figura 1.4: Capturas de pantalla de la aplicación profesional Vicon Polygon.

También es posible entregar al paciente un informe que contiene una síntesis de sus antecedentes e informe de su examen físico, el análisis observacional de la marcha, el registro de los movimientos con sus rangos y una electromiografía dinámica de sus músculos.

En la figura 1.5 podemos ver un típico informe resultante del proceso. En él se puede apreciar que en cada gráfica, que muestra un ángulo determinado, aparece una línea gris gruesa que representa la normalidad de dicho ángulo, que se ha calculado previamente con estadísticas sobre bases de datos de muchas personas. Las otras dos líneas más finas representan los valores reales de las articulaciones izquierda y derecha. Cuando dichas líneas se salen de la normalidad, esto indica que el paciente padece algún tipo de patología del sistema nervioso o motor, y resulta más fácil la identificación de patrones que determinen los fallos característicos de dichas patologías.

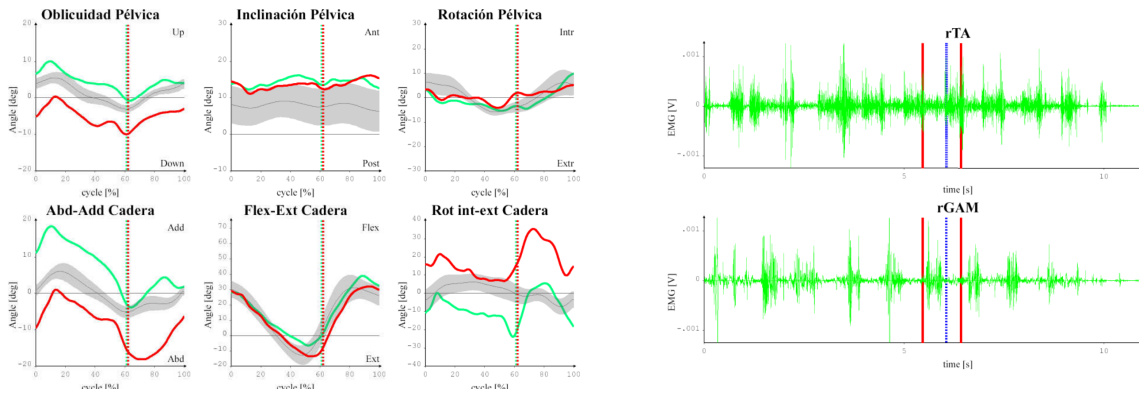

Figura 1.5: Un informe obtenido tras el procesado del laboratorio de análisis de marcha.

Mercado e Investigación en el análisis de movimiento

Dentro del ámbito del análisis de movimiento, existen varias empresas dedicadas al desarrollo de equipo relacionado, como por ejemplo *Qualisys*[1] en Suecia y *BTS*[2] en Italia. Pero la empresa más importante del mercado es *VICON*[3], líder en tecnología de captura y análisis del movimiento. Esta compañía ofrece uno de los mejores hardware disponibles para un laboratorio de análisis de marcha (lo que incluye cámaras digitales de alta resolución en red) así como el software más potente para capturar y procesar los datos de movimiento, bajo el nombre de *Vicon MX Nexus*. Además ofrece aplicaciones como *Polygon*, que permite crear presentaciones que contienen todos los datos e informes necesarios para un análisis completo, con un manejo fácil, posibilidad de comparaciones con otros informes y de exportar los informes a otros tipos de formatos conocidos, como *Microsoft Excel*.

También existen desarrollos en este área al margen de empresas privadas. *SimTK.org* es un entorno libre de programación bajo la cual se desarrollan muchos otros proyectos, relacionados con la simulación de estructuras biológicas. Algunos de los proyectos desarrollados en esta organización más representativos son *OpenSim* y *3D Muscle*. *OpenSim* es un software de creación de modelos músculoesqueléticos y simulación del movimiento humano. *3D Muscle* es un nuevo método de modelado muscular que representa los músculos como volúmenes en tres dimensiones, incorpora las propiedades de las fibras musculares e incluye las mecánicas del contacto entre músculos y huesos.

[1] http://www.qualisys.com
[2] http://www.bts.it
[3] http://www.vicon.com

Gran parte de estas compañías y grupos de investigación dedican sus esfuerzos a investigar en todas las áreas de desarrollo relacionadas con el análisis de movimiento. Podrían definirse tres áreas temáticas en el ámbito de los laboratorios de ánalisis de marcha correspondientes a distintas fases de funcionamiento de estos sistemas:

- *Captura de movimiento.* Este área gira en torno a la técnica para almacenar los movimientos digitalmente. Habitualmente esta técnica necesita de cámaras especializadas y marcadores reflectivos, magnéticos o luminosos, o una combinación de varios, que son rastreados durante el movimiento. La captura de movimiento tiene otras múltiples aplicaciones en los ámbitos del cine y los videojuegos. La empresa *Vasa*[4] está especializada en este área y trabaja con muchas compañías de cine y videojuegos.

- *Procesamiento de datos.* Esta rama se encarga del tratamiento de los datos obtenidos durante la captura de movimiento y sus posibles aplicaciones para posteriores resultados e informes. En este ámbito, también está incluído todo el software relacionado con el modelado de figuras virtuales, el cálculo de posiciones, etc.

- *Visualización.* Otro área de desarrollo asociado es el de la visualización, que abarca todos los productos informáticos relacionados con la animación de figuras humanas, animación de entornos tridimensionales y la creación de informes. *MusculoGraphics, Inc.*[5] es un buen ejemplo de compañía que se dedica especificamente al desarrollo de software especializado en la visualización y simulación de tecnologías de realidad virtual para aplicaciones médicas. Sus desarrollos se centran en la representación gráfica de diversos análisis biomédicos, más allá de las representaciones de datos estáticas ya existentes. Esta empresa se especializa en las áreas de la ingeniería biomédica, dinámicas del cuerpo humano y software de modelado biomecánico. Éste es precisamente el área en el que se va a centrar este proyecto fin de carrera.

Laboratorios de análisis de movimiento en España

Varios hospitales por todo el globo disponen de laboratorios de análisis de marcha. En España, el *Hospital Infantil Universitario Niño Jesús*(figura 1.6) es pionero en la adquisición de uno de estos laboratorios, coordinado entre los servicios de

[4]http://www.vasa.com
[5]http://www.musculographics.com

Rehabilitación y Traumatología del centro. La labor específica del laboratorio de este hospital es el tratamiento de niños y adolescentes que muestren problemas motrices. Habitualmente se tratan a tres pacientes al día, de los cuales la mayoría responden a patologías neuromusculares y los otros tienen problemas congénitos. Este laboratorio funciona con software y hardware de la compañía *BTS*.

Figura 1.6: El laboratorio de análisis de movimiento del Hospital Niño Jesús.

La Facultad de Ciencias de la Salud[6] de la Universidad Rey Juan Carlos, en Alcorcón, también posee un laboratorio de análisis de movimiento muy completo, a cargo del departamento de Fisioterapia, Terapia Ocupacional, Rehabilitación y Medicina Física, con componentes hardware y software de alta tecnología *Vicon*. Este laboratorio (figura 1.7) combina el uso docente con el análisis clínico de pacientes.

Figura 1.7: Laboratorio de análisis de movimiento de la URJC.

[6]http://www.cs.urjc.es/

1.3. Visualizador 3D interactivo para laboratorios de análisis de marcha

La mayoría de los profesionales en medicina concuerdan en que el análisis cuantitativo del movimiento proporciona una más amplia descripción de la marcha, comparado con el análisis sólo observacional en el momento.

Con este proyecto fin de carrera pretendemos programar una herramienta visual *interactiva* de ayuda al médico para el análisis y *diagnóstico* de problemas de marcha, de forma que se convierta en una utilidad adicional al laboratorio de análisis de marcha del Hospital Infantil Niño Jesús. Esta herramienta consistirá principalmente en una interfaz gráfica de imágenes tridimensionales que permita la visualización de un esqueleto en un entorno virtual que se mueva en base a los datos del paciente recolectados previamente por el laboratorio de análisis de marcha. El visualizador incorporará diversos controles que permitan observar el movimiento de la figura desde distintos ángulos y a distintas velocidades.

El visualizador contará con una serie de funcionalidades que facilitarán al médico el análisis de los datos: un sistema de control de la cámara que permite ver el movimiento desde cualquier ángulo y posición, permitir pausar la figura en el momento deseado, avanzar y retroceder el movimiento de la figura a voluntad. Por último, se incluirá la visualización de gráficas detalladadas de la evolución de la posición y ángulos de rotación de una articulación seleccionada. De esta forma, se intentará conseguir una aplicación que sirva de informe interactivo para el médico usuario.

Este tipo de funcionalidad ya se ofrece en aplicaciones desarrolladas por compañías del sector, como *Vicon* (con su programa *Vicon Polygon*), con paquetes adicionales de alto coste que muchos usuarios no pueden permitirse. Este proyecto fin de carrera pretender servir como alternativa real a este software comercial.

Hasta ahora, la única herramienta visual con la que cuenta el laboratorio de análisis de movimiento del Hospital Niño Jesús es una figura simple construida a base de segmentos rectos que unen las articulaciones, así como del software encargado de crear los informes que contienen las gráficas y datos necesarios para el diagnóstico del paciente. El presente proyecto fin de carrera ofrecerá una figura modelada de un esqueleto humano que se moverá acorde a los datos capturados del paciente. Es

importante que el resultado final tenga una utilidad *real* para el usuario médico y le ayude verdaderamente en el análisis de los pacientes, con lo que se ha de tener en cuenta la metodología médica y cómo se realizan los diagnósticos y análisis.

Respecto al contexto inmediato de este proyecto fin de carrera, nos apoyaremos en la experiencia acumulada del departamento de Robótica de la *Universidad Rey Juan Carlos* en el área de la Visión Artificial, lo que incluye el tratamiento de entornos tridimensionales y las conversiones entre imágenes planas e imágenes tridimensionales. También nos ayudaremos de la experiencia previa del departamento en el desarrollo de entornos *OpenGL*.

El resto de la memoria de este proyecto fin de carrera se estructura de la siguiente manera en el próximo capítulo 2 fijaremos los objetivos concretos que planteamos, así como los requisitos asociados deseables. A continuación, en el capítulo 3, detallaremos tanto la plataforma como las herramientas en las que se apoya el *software* deseado. En el capítulo 4 describiremos la solución final desarrollada en este proyecto fin de carrera y en el capítulo 5 los experimentos llevados a cabo para depurar y optimizar la misma. Por último, recapitularemos los resultados obtenidos en el capítulo de conclusiones y daremos algunas perspectivas de futuro.

Capítulo 2
Objetivos

Después de haber presentado el contexto dentro del cual se engloba nuestro trabajo, entraremos a detallar los objetivos concretos de este proyecto fin de carrera, así como los requisitos marcados.

2.1. Descripción del problema

Estamos interesados en diseñar una herramienta de visualización interactiva tridimensional de esqueletos en movimiento que sirva como ayuda al médico para el análisis y diagnóstico de enfermedades del sistema locomotor y neuromuscular. Esta aplicación servirá como ayuda al profesional médico en el análisis y diagnóstico de problemas motrices, y más concretamente, como complemento al laboratorio de análisis de movimiento. Este objetivo genérico lo hemos articulado en tres subobjetivos concretos.

El *primer subobjetivo* es conseguir un visualizador que permita mostrar un modelo de esqueleto de huesos que actúe y se mueva acorde a los datos obtenidos del paciente real. De esta manera, el médico podrá analizar más claramente el movimiento de su estructura ósea. El visualizador incluirá un entorno tridimensional eficiente y controles de iluminación para optimizar la visibilidad por parte del usuario. Es importante señalar que nuestro objetivo es crear un modelo de huesos, pero no de músculos, ya que estos deberían ser objetos deformables y, por lo tanto, muy complejos.

El *segundo subobjetivo* es hacer una aplicación interactiva, de forma que su manejo sea accesible e intuitivo. Para conseguirlo, se implementarán las siguientes funcionalidades:

- Un controlador de cámara para poder ver la figura desde cualquier ángulo.

- Controles que permitan congelar la imagen o ver determinados momentos del movimiento del esqueleto.
- Un modulador de velocidad que permita reproducir el movimiento a cámara lenta o, en general, a velocidad variable.

El *tercer subobjetivo* consistirá en desarrollar una funcionalidad adicional que permita al usuario visualizar una serie de gráficas relacionadas con cualquier articulación del tren inferior del cuerpo que muestren sus ángulos de giro y su posición en el espacio con respecto al tiempo, de manera que muestren la evolución temporal de dicha articulación. Además, estas gráficas tienen que poder seleccionarse con comodidad.

2.2. Requisitos

El desarrollo de la aplicación que presenta este proyecto fin de carrera deberá cumplir además los siguientes requisitos, que estarán guiados por los objetivos marcados en la sección anterior.

El primer requisito para este proyecto es desarrollar el proyecto fin de carrera sobre el sistema operativo Linux y en la plataforma *JDE*, que se describe en el capítulo 3 y es el entorno de programación estándar en el Grupo de Robótica; ese será el punto de partida para el código de esta aplicación. También se toma como requisito la utilización del lenguaje C para la programación de los algoritmos y funciones que comprenderá la aplicación.

Otro requisito importante es la compatibilidad con el sistema ya existente en el laboratorio de movimiento del Hospital Niño Jesús. Debemos conseguir que nuestra aplicación soporte de forma efectiva los diferentes datos que manejan en ese laboratorio.

Un último requisito importante es desarrollar una aplicación con un comportamiento vivaz y una carga computacional ligera.

2.3. Metodología y plan de trabajo

Para llevar a cabo este proyecto fin de carrera nos hemos basado en un *modelo de desarrollo en espiral basado en prototipos.* Este modelo se caracteriza por la realización de subtareas en un número determinado de ciclos, todos con un desarrollo similar. Tanto el desarrollo global como cada uno de los ciclos han sido supervisado por el tutor mediante reuniones periódicas semanales, y cada ciclo ha terminado con un prototipo que cumplía con los requisitos exigidos en esa iteración.

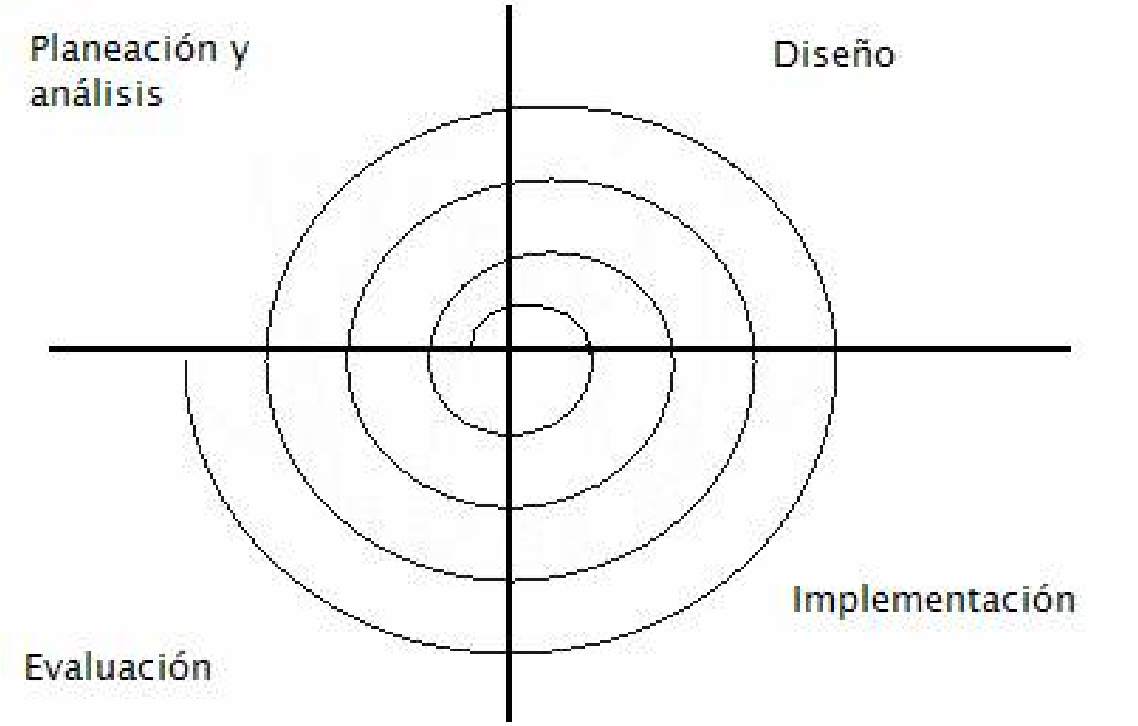

Figura 2.1: Modelo de desarrollo en espiral basado en prototipos.

Para el desarrollo del presente proyecto se han completado los siguientes ciclos:

- *Formación básica y familiarización con la infraestructura software.* En esta fase inicial se pretende adquirir conocimientos sobre el entorno Linux y sus herramientas, así como la familiarización con la plataforma software *Jde* y el estudio a fondo de la estructura de la misma. Se llevan a cabo pruebas con el lenguaje C y la biblioteca gráfica *GTK*. Seguidamente, se realizan prácticas para la familiarización con la biblioteca gráfica *OpenGL*.

- *Creación del modelo de esqueleto 3D* Tras el procesado del modelo tridimensional del esqueleto, procedemos a mostrarlo mediante *OpenGL* y realizamos los

primeros movimientos prefijados, primero con un modelo de segmentos y luego con la figura ósea.

- *Recepción y procesamiento de datos.* Procesamos los datos recibidos por el fichero que se nos proporciona previamente para conseguir una reproducción básica de la figura humana en movimiento. Posteriormente obtenemos una animación completa del esqueleto acorde a dichos datos.

- *Funcionalidades interactivas para el entorno gráfico.* Incluimos una serie de funcionalidades adicionales para facilitar la visualización del modelo: controles de cámara (rotación y zoom) mediante el ratón, botones de Pausa y Reanudación de la acción, controladores de iluminación del entorno, una barra de progreso que adicionalmente nos permite hacer retroceder o avanzar la acción, y un botón para la carga de datos por fichero.

- *Evolución temporal de las articulaciones.* Esta funcionalidad adicional consiste en mostrar gráficas concretas sobre una articulación al pulsar con el ratón en dicha articulación. Esto incluye el estudio y empleo de la biblioteca *Progeo* para el tratamiento de la relación de píxeles de la imagen y puntos en la escena 3D espacial.

En mi ficha web personal[1] del Grupo de Robótica se puede ver la evolución llevada durante el desarrollo del presente proyecto fin de carrera.

[1] http://jde.gsyc.es/index.php/Dmuelas_3D_skeleton_visualizer

Capítulo 3
Entorno y plataforma de desarrollo

En este tercer capítulo vamos a aprender la infraestructura software relacionada con este proyecto. Se han usado una gran variedad de bibliotecas, aplicaciones y herramientas para llevar a cabo este proyecto fin de carrera: *jde.c* como plataforma software, *Blender* como herramienta de modelado del esqueleto tridimensional, *GTK* para crear interfaces gráficas, *OpenGl* para visualizar de forma gráfica los algoritmos implementados, y *Progeo* para relacionar las imágenes bidmiensionales con el entorno tridimensional virtual.

3.1. Plataforma Jde

Jde[1] (Jerarquía Dinámica de Esquemas) es una plataforma de software libre, desarrollada íntegramente por la Universidad Rey Juan Carlos, para la programación de comportamientos autónomos en robots móviles, de aplicaciones relacionadas con la visión artificial y de domótica. Esta plataforma resuelve problemas generales de la robótica, como son el acceso a sensores (en forma de variables que las aplicaciones leen) y actuadores (como variables que las aplicaciones escriben), la multitarea o la comunicación entre componentes. Para ello, la plataforma ofrece una serie de *drivers* ya incluídos como: *imagefile* para tratar imágenes desde fichero; o *pantilt* para operar el cuello mecánico. Este proyecto se ha desarrollado sobre su versión 4.3.

La plataforma *Jde* plantea las aplicaciones en forma de módulos independientes denominados esquemas, que se organizan mediante jerarquías, y mediante su ejecución simultánea dan lugar a un comportamiento en un robot o en una aplicación. Un esquema es un flujo de ejecución con un objetivo. Puede ser activado o desactivado y acepta varios parámetro. Existen dos tipos fundamentales de esquemas: perceptivos y motores. Los esquemas perceptivos son aquellos que reciben información de los sensores

[1] http://jde.gsyc.es/

y la procesan pudiendo ponerla a disposición de otros esquemas motores, para tomar decisiones de actuación. Estos esquemas se comunican mediante variables compartidas y se pueden activar y desactivar con un simple click de ratón, lo que permite ver la interfaz gráfica o desactivarla a voluntad del usuario. Todas estas funcionalidades están disponibles desde la ventana principal del programa (figura 3.1).

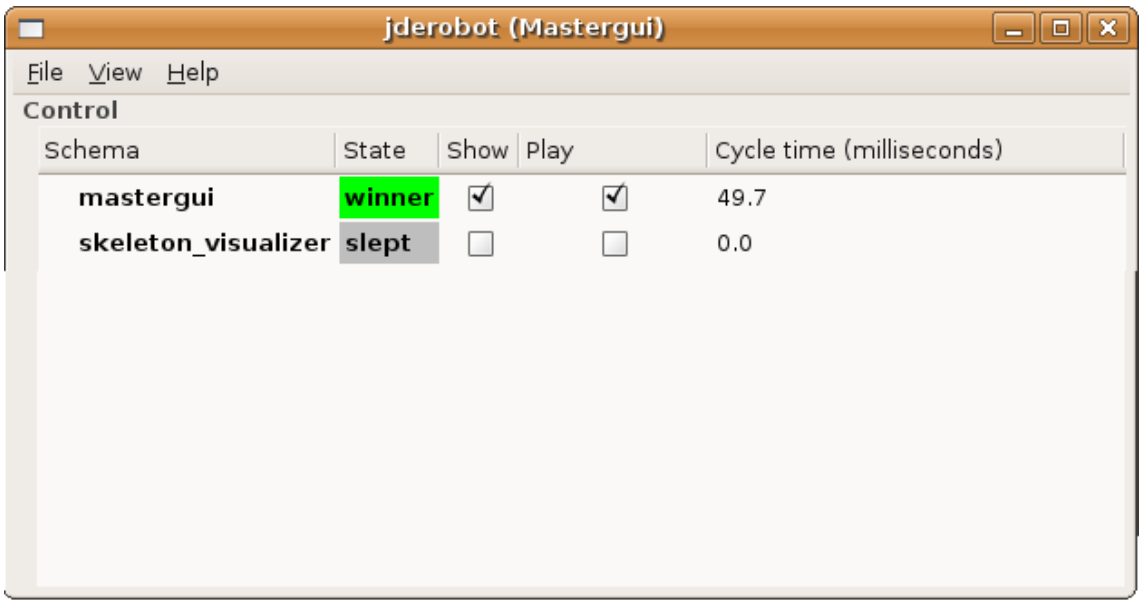

Figura 3.1: Ventana principal de *Jde*.

Jde funciona sobre el sistema operativo GNU/Linux, que es estable y eficiente y dispone de una comunidad muy activa de desarrolladores, lo cual garantiza la incorporación continua de soporte para los nuevos dispositivos hardware que van apareciendo. *Jde* está escrito en el lenguaje de programación C, y las aplicaciones sobre ella también deben programarse en ese lenguaje. *C* supone un buen compromiso entre potencia expresiva y rapidez, y como lenguaje compilado su eficiencia temporal es superior a otros lenguajes interpretados. Además, la plataforma resuelve las necesidades de interfaz gráfica de las aplicaciones. Buen ejemplo de ellos son los drivers *graphics_gtk* y *graphics_xforms*, que nos ofrecen soporte para gestión de ventanas. En nuestro caso, hemos empleado *graphics_gtk*.

La aplicación desarrollada en este proyecto consta de un único esquema, que consideraremos perceptivo, por lo que no hará falta utilizar ninguna jerarquía de esquemas.

3.2. Biblioteca GTK

La biblioteca *GTK*[2] (o *Gimp Toolkit*) es una de las muchas que compone el conjunto *GTK+* y nos permitirá crear la interfaz gráfica de usuario gracias a las funciones y objetos que ofrece. Está escrita en lenguaje *C* Nos permite crear una interfaz gráfica con la que visualizar y depurar los resultados que vamos obteniendo con los distintos algoritmos.

Esta biblioteca proporciona una herramienta denominada *glade* con la que poder crear visualmente objetos gráficos en sistemas de ventanas, denominados *widgets*, como botones, menús, etiquetas, diales, barras de desplazamiento, *canvas* para visualización de *OpenGL* y muchos otros. En conjunto, *glade* sirve para crear GUIs variados, y en nuestro caso nos ha servido para crear la interfaz gráfica que presenta el visualizador interactivo de esqueletos, así como de las diferentes herramientas que facilitan su manejo y la interacción con nuestra aplicación. Concretamente se han utilizado:

- Barras de desplazamiento para configurar la iluminación, determinar la velocidad y controlar y alterar el momento de ejecución.
- Botón para centrar el foco de atención de la cámara.
- *Canvas* de dibujo para crear las gráficas de las articulaciones.
- *Widget* de elección de fichero, que despliega un menú que permite cargar otros ficheros de datos.

El mecanismo de eventos *GTK* es tal que asociamos señales a manejadores. El diseñador de interfaces *Glade* permite asociar las señales de determinado *widget* a sus funciones manejadoras o *callbacks*. Este tipo de funciones reciben siempre como parámetro una referencia al *widget* del que procede la señal, pudiendo reconocer la fuente del evento. De ese modo, hemos podido emplear en algunos casos un mismo *callback* para manejar eventos de distintos objetos. En la figura 3.2 podemos ver la interfaz de *Glade*.

[2]http://www.gtk.org/

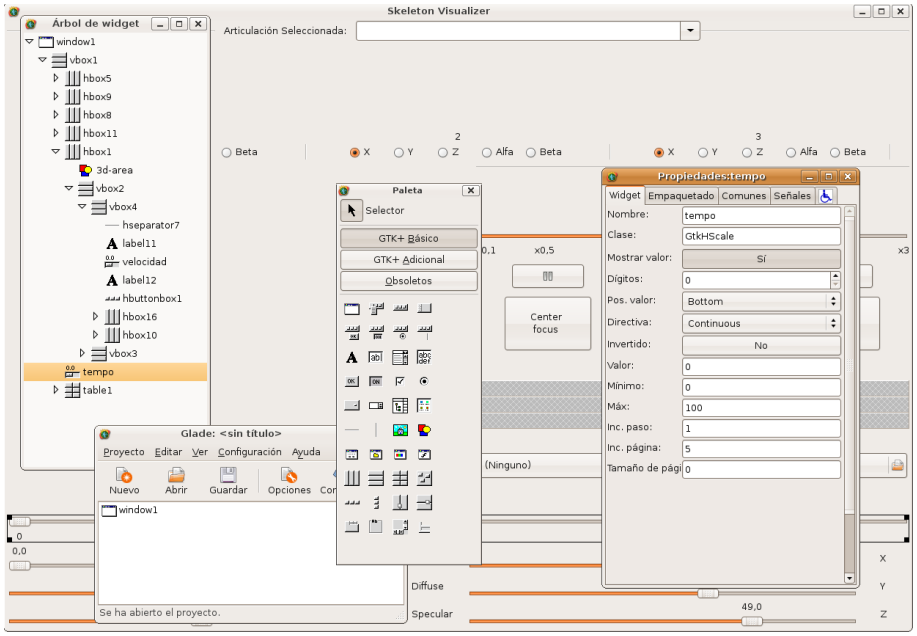

Figura 3.2: Interfaz gráfica de *Glade*.

A continuación mostramos a modo de ejemplo el código del *callback* que maneja el evento "*scroll-event*"del área de dibujo que se encarga de aumentar o reducir el zoom de la cámara al mover la rueda del ratón:

```
gboolean scroll_event (GtkRange *range, GdkEventScroll *event,
                       gpointer data)
{
   if (event->direction == GDK_SCROLL_DOWN){
      if (radius > MIN_RADIUS_VALUE) radius-= WHEEL_DELTA;
   }
   if (event->direction == GDK_SCROLL_UP){
     if (radius<MAX_RADIUS_VALUE) radius+= WHEEL_DELTA;
   }
   if (radius < 0.5){ radius = 0.5;}
   gtk_widget_queue_draw(GTK\_WIDGET((GtkWidget *)data));
   return TRUE;
}
```

Esta herramienta es soportada por *jde* a través del servicio *graphics_gtk*. Este componente realiza la inicialización de la biblioteca y arranca dos hilos de ejecución:

uno para el hilo principal de GTK (*gtk_main*) y otro que actualiza los *displays* de todos los esquemas registrados. Además, se encarga de proveer una función que facilita la carga del fichero *.glade* y soportar los GUIs de los múltiples esquemas que tengan programada su interfaz gráfica con GTK.

Es necesario programar una serie de funciones en el esquema para que el driver *graphics_gtk* sea soportado correctamente:

- *guiresume*: Esta función se utiliza para mostrar el *display*. La primera vez que se llama a esta función se carga la interfaz gráfica y se asignan las funciones de *callback* a los diferentes eventos de cada *widget* (estas funciones deben ser programadas para que atiendan los eventos correctamente). Por último, la función registra el *callback* de refresco del *display*.
- *guisuspend*: Esta función se utiliza para ocultar el display y eliminar la suscripción al *callback* de refresco de la interfaz.
- *guidisplay*: Esta es la función que se ejecuta cada vez que se quiere refrescar la interfaz.

GTK está disponible en paquetes *Debian*[3].

3.3. Biblioteca OpenGL

La biblioteca gráfica *OpenGL*[4] (Open Graphics Library) permite dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos. Dicho de otra forma, genera una imagen sintética desde una cámara virtual a través de la cual se observa la escena 3D y los distintos objetos y luces que hay en ella. Así, esta librería nos va a permitir visualizar de forma elegante y eficiente el escenario 3D o las estructuras internas de los algoritmos, por lo que nos será muy útil como método de depuración. Además, la carga computacional se trasladará al procesador gráfico (*GPU*), ya que esta biblioteca trabaja directamente con el procesador gráfico y no con la *CPU*, y ofrece una gran interactividad y realismo en la escena.

[3]libgtk2.0-0, libgtk2.0-dev, libgtkextra-x11-2.0-1, libgtkextra-x11-2.0-dev, libgnomecanvas2-dev, libgtkglext1, libgtkglext1-dev

[4]http://www.opengl.org/

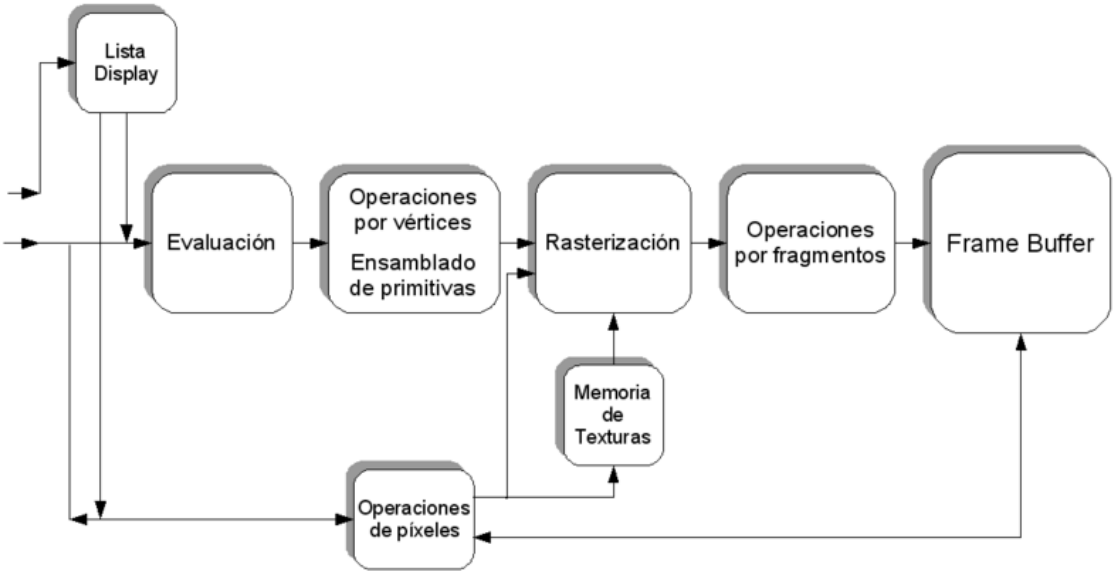

Figura 3.3: Pipeline de *OpenGL*.

Esta biblioteca fue desarrollada originalmente por *Silicon Graphics Inc. (SGI)* en 1992. Hay implementaciones eficientes de *OpenGL* para Mac OS, Microsoft Windows, Linux y varias plataformas Unix. Es destacable la biblioteca de software libre / código abierto *Mesa 3D*, una API de gráficos sin aceleración hardware y completamente compatible con *OpenGL*.

Fundamentalmente, *OpenGL* es una máquina de estados. Cuando se activan o configuran varios estados de la máquina, sus efectos perdurarán hasta que sean desactivados. Por ejemplo, si el color para pintar polígonos se pone a blanco, todos los polígonos se pintarán de este color hasta que se cambie el estado de esa variable.

En *OpenGL* las operaciones de rotación, translación, escalado, etc. se realizan a través de matrices de transformación. Dependiendo de lo que estemos tratando, hay tres tipos de matriz (que son los tres posibles flags que pueden llevar de parámetro la función): matriz de proyeccion (GL_PROJECTION), matriz de modelo (GL_MODELVIEW) y matriz de textura (GL_TEXTURE). Con la funcion *glMatrixMode()* indicamos a que matriz deben afectar las operaciones.

Por ejemplo, los cambios en el sistema de coordenadas del entorno se llevan a cabo mediante transformaciones. Las principales transformaciones nos permiten mover, rotar y escalar los objetos, mediante las funciones *glTranslatef()*, *glRotatef()*, *glScalef()* respectivamente. Estas funciones en realidad no transforman los objetos sino su sistema

de coordenadas, de forma que si queremos rotar un objeto, no lo rotamos a él, sino al eje sobre el que se sitúa. Cada una de las llamadas a estas funciones modifican la matriz identidad de forma acumulativa, de manera que la apariencia final de nuestros objetos depende en gran medida del orden con el que se hayan aplicado las transformaciones. Se puede reiniciar la matriz identidad mediante la función *glLoadIdentity()*; siempre es deseable reiniciar la matriz del modelador con la identidad antes de colocar cada objeto. A menudo querremos almacenar el estado actual de transformación y entonces recuperarlo después de haber colocado varios objetos. Para ello, *OpenGL* mantiene una pila de matrices para el modelador (GL_MODELVIEW) y otra para la proyección (GL_PROJECTION). Para meter una matriz en su pila correspondiente se usa la función *glPushMatrix()*, y para sacarla *glPopMatrix()*.

He aquí un ejemplo de un brazo construido con segmentos mediante funciones de rotación y translación:

```
 glLoadIdentity();
 glTranslatef(0.0,0.0,0.40);
 glRotatef(10.0,1.0,0.0,0.0);
 glBegin(GL_LINES);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(0.0, 0.0, -0.10);
 glEnd();
 glTranslatef(0.0,0.0,-0.10);
glRotatef(40.0,1.0,0.0,0.0);
 glBegin(GL_LINES);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(0.0, 0.0, -0.12);
 glEnd();
```

Concretamente, hemos empleado esta librería para modelar completamente el entorno virtual que ubicará al esqueleto. Además, los botones de pausa y reanudación del movimiento han sido incorporados al propio escenario, como veremos en el capítulo 5.

OpenGL está disponible en paquetes *Debian*[5].

[5] libgl1-mesa-dev, mesa-common-dev, libglu1-mesa-dev, freeglut3 freeglut3-dev, libglut3, glutg3-dev

3.4. Biblioteca Progeo

La biblioteca *Progeo* es utilizada para realizar cálculos de geometría proyectiva. Estos cálculos permiten procesar la información de puntos geométricos en el espacio tridimensional para obtener sus proyecciones en un plano imagen determinado. Mediante estas proyecciones, los puntos 3D pueden ser visualizados en una imegen bidimensional, empleando para ello algún gestor gráfico. De igual forma, los cálculos pueden ser realizados a la inversa, de manera que a partir de varias proyecciones de un punto, podamos obtener las coordenadas 3D del mismo.

Para usar las funciones que ofrece *Progeo* es preciso calibrar la cámara que se vaya a utilizar, en nuestro caso, una cámara virtual modelo *Pinhole*. Una vez hecha la calibración, es posible usar las funciones de proyección y retroproyección. La función de proyección (*project*) permite proyectar un punto 3D sobre el plano imagen de una cámara, para obtener el punto 2D contenido en el mismo. Esta función permite visualizar un punto 3D sobre el plano imagen de una cámara, y de esta forma dibujarlo sobre un monitor plano.

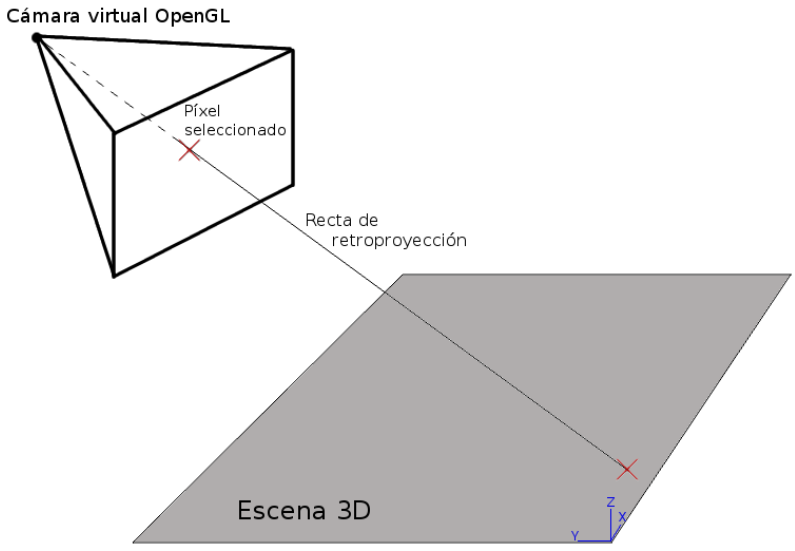

Figura 3.4: Esquema de retroproyección mediante *Progeo*.

La función de retroproyección (*backproject*) efectúa la operación inversa a la anterior. Mediante esta función se puede obtener las coordenadas 3D de un punto proyectado en varios planos imagen de diferentes cámaras. Para ello, dado un punto 2D contenido en el plano imagen de una cámara concreta (en nuestro caso, la cámara

virtual), permite obtener la representación 3D de ese punto.Esa representación, junto con el foco de la cámara, permite trazar una recta en el espacio que contiene el punto 3D buscado. Esta es la función que vamos a utilizar en nuestro esquema. En la figura 3.4 se puede ver un esquema del funcionamiento de la retroproyección.

Así, hemos podido establecer un sistema para interactuar con el mundo simulado en 3D con *OpenGL*, de forma que el usuario pinche en un pixel y el sistema compruebe si la recta proyectada pasa cerca de alguna de las articulaciones del tren inferior del cuerpo, y de ser así, proceda a mostrar datos gráficos sobre esa articulación en concreto, como veremos en el capítulo 4.

Progeo forma parte de las bibliotecas de *Jde 4.2.1.*

3.5. Blender

Para el modelado y tratamiento del esqueleto humano de huesos tridimensional hemos usado *Blender*[6], un programa dedicado especialmente al modelado y creación de gráficos tridimensionales. A pesar de que posee una interfaz gráfica peculiar, criticada como poco intuitiva, cuenta con una serie de ventajas como son la configuración personalizada de la distribución de los menús y vistas de cámaras, la capacidad de aceptar formatos gráficos variados como TGA, JPG, Iris y OBJ, y (los motivo principales por los que escogimos este programa) es libre y gratuito.

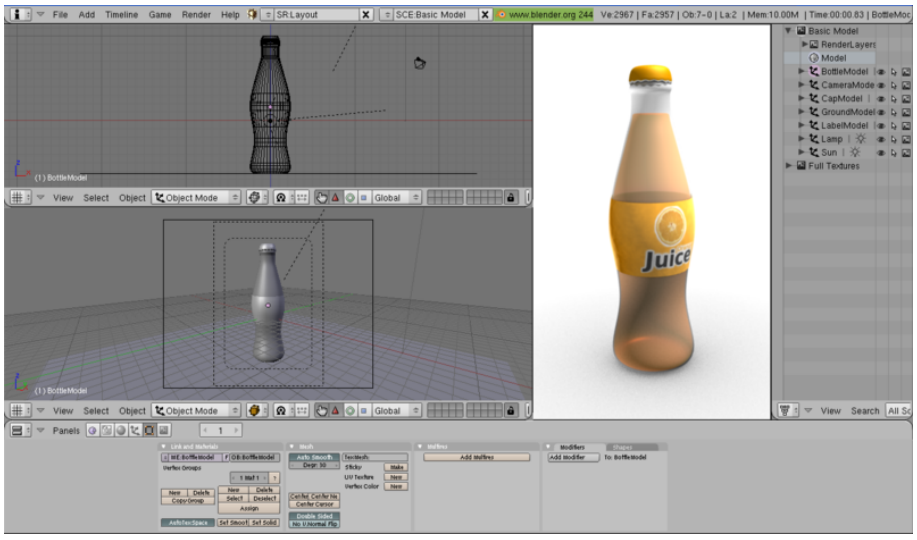

Figura 3.5: Ejemplo de interfaz de *Blender*.

[6]http://www.blender.org/

Blender está disponible mediante un paquete *Debian*[7].

Concretamente, *Blender* ha sido utilizado en este proyecto para realizar correctamente la separación del modelo rígido del esqueleto del que partimos, en varios huesos separados que permitiesen la movilidad deseada, principalmente en el tronco inferior. Cada parte separada fue exportada a un archivo OBJ para, posteriormente, poder cargarlo desde nuestro esquema.

[7] blender

Capítulo 4
Descripción informática

Una vez presentado el contexto y fijado el objetivo concreto vamos a detallar en este capítulo la solución informática programada, describir su diseño y las partes desarrolladas. Como hemos mencionado, el objetivo de este proyecto fin de carrera es crear un visualizador 3D interactivo de esqueletos que reproduzca el movimiento de pacientes con problemas motrices y facilite su diagnóstico. La descripción de la aplicación la hemos estructurado en varias secciones de este capítulo que detallan las distintas funcionalidades desarrolladas, como son el procesamiento de los datos de entrada, el modelado del esqueleto, la animación de la figura y el detalle de las gráficas de las articulaciones.

4.1. Diseño Global

Desde el punto de vista externo, nuestro esquema funciona como podemos ver en la figura 4.1. En primer lugar recibe como datos de entrada el fichero con la información de un paciente obtenida a través del laboratorio de análisis de movimiento del Hospital Niño Jesús. Estos datos contienen las posiciones del cuerpo del paciente en el tiempo. También se suministran los archivos con el modelo 3D del esqueleto humano y el fichero de configuración del esquema.

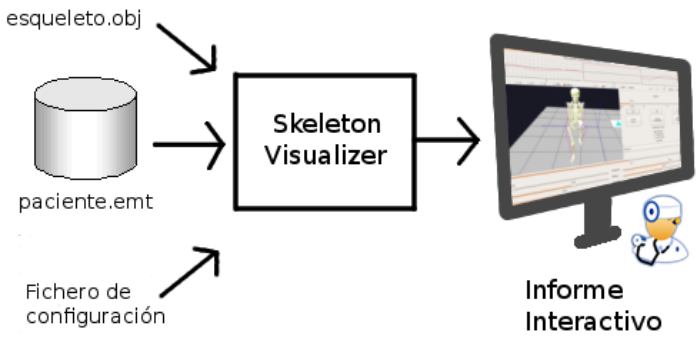

Figura 4.1: Entradas y salidas del esquema desarrollado

El esquema se encarga de procesar los datos, realizar los cálculos necesarios y mostrar a través de la interfaz gráfica el resultado a modo de un informe interactivo con el que el usuario puede observar el movimiento homólogo de la figura ósea y gráficas sobre posiciones y ángulos de las articulaciones.

El visualizador 3D interactivo para laboratorios de análisis de marcha está formado por un esquema de *Jde* y un archivo *glade* que se encarga de crear la interfaz gráfica de cara al usuario. En la figura 4.2 podemos ver el GUI del componente programado.

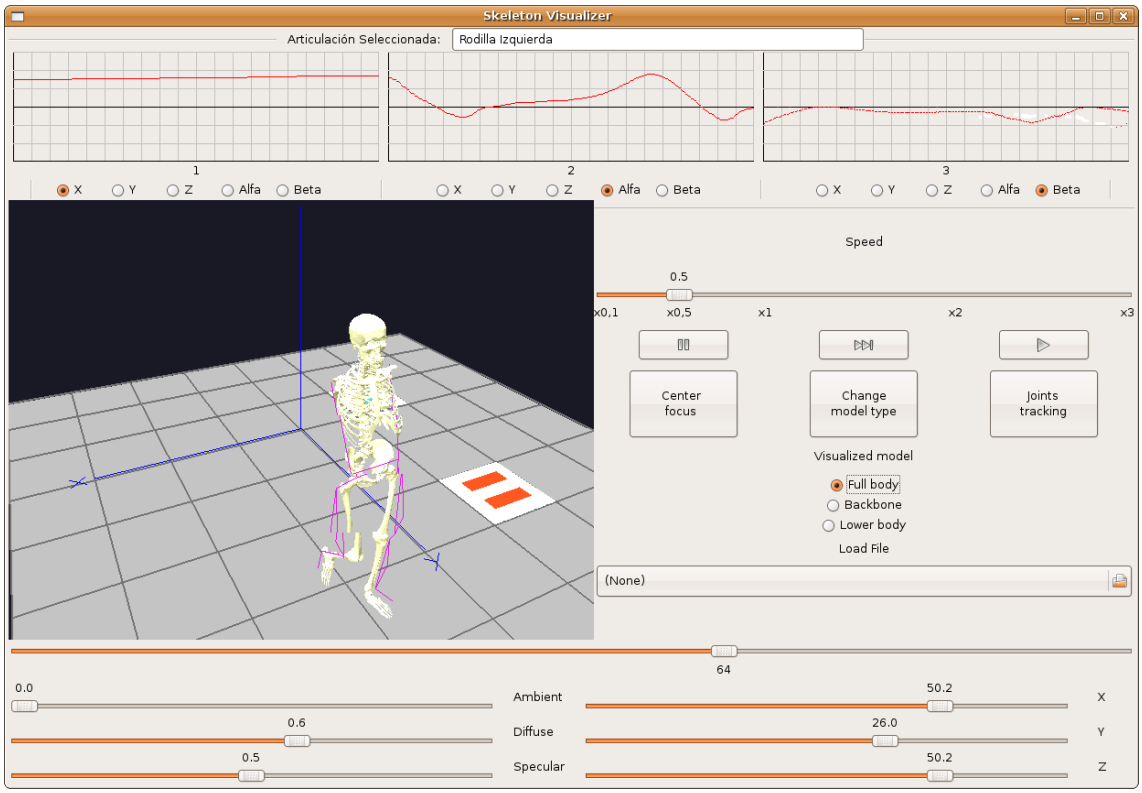

Figura 4.2: Interfaz gráfica de la aplicación desarrollada

Interiormente la aplicación está estructurada en los bloques de la figura 4.3. Cada uno de los cinco bloques comprende una funcionalidad diferenciada de todo el conjunto, de forma que la aplicación se divide de la siguiente forma: inicialización de parámetros, modelado de la figura ósea, animación de la figura ósea, controles de la cámara virtual del entorno y detalle de las articulaciones. Iremos describiendo estos bloques de manera pormenorizada en las distintas secciones de este capítulo.

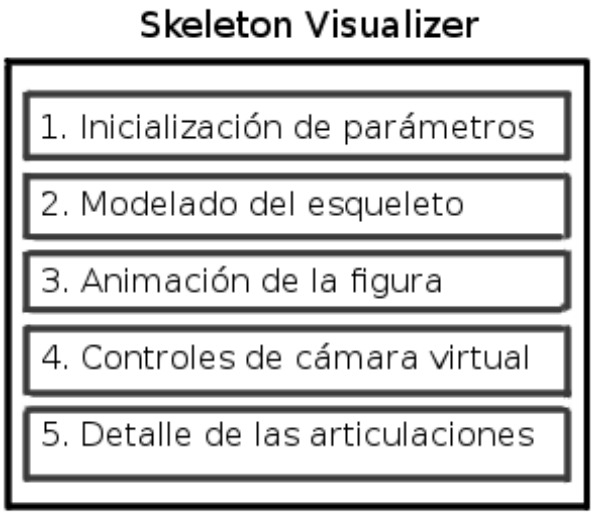

Figura 4.3: Diagrama de bloques del esquema skeleton_visualizer

En cuanto al diseño de los flujos de control de la aplicación, hemos diseñado nuestra aplicación con dos hilos de ejecución: *display* e *iteration*. El hilo de *display* se encarga de lanzar la función *guidisplay*, que refresca la interfaz gráfica, muestra los datos de salida, como pueden ser las gráficas de las articulaciones o la misma figura ósea en movimiento, y se comunica con el servicio *graphics_gtk*. El hilo de la función *iteration* se encarga de procesar los datos de entrada a un ritmo controlado de 25 iteraciones por segundo. En la figura 4.4 podemos ver el esquema de los flujos de ejecución del esquema.

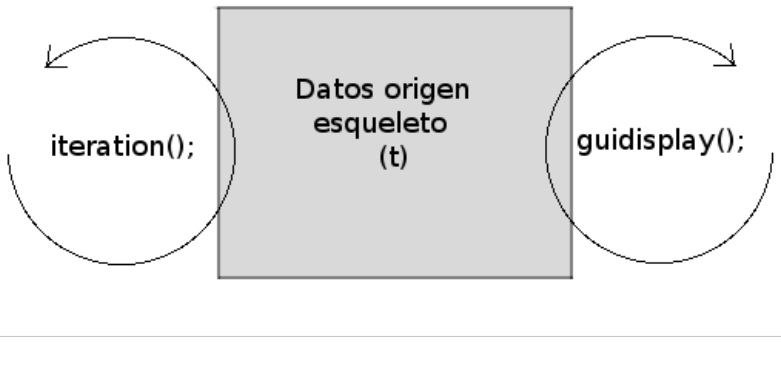

Figura 4.4: Esquema de los diferentes ritmos del esquema.

4.2. Procesamiento de los datos de entrada

Como hemos visto en la figura 4.3, una de las entradas a nuestra aplicación es el fichero de datos, que contiene las posiciones en 3D de los marcadores y articulaciones del paciente durante el seguimiento de su movimiento. Esta es la información que queremos visualizar con nuestra aplicación, presentándola de un módo más cómodo y potente que el texto simple. Este fichero de datos tiene la extensión *.emt* y procede del laboratorio de análisis de movimiento. El esquema lo busca por defecto en el mismo directorio donde se encuentra y con el nombre *track.emt*. Tiene formato de texto (figura 4.5) y contiene un listado en columnas de las posiciones en los ejes X, Y y Z de cada uno de los marcadores y articulaciones calculadas en cada momento del tiempo, a intervalos de una cantidad determinada de milisegundos.

Este fichero es el resultado de todo el proceso del laboratorio del análisis de movimiento y no se calculan de la misma forma las posiciones de los marcadores y las articulaciones: mientras que la posición de los marcadores se ha obtenido directamente al capturar el movimiento 3D de los marcadores adheridos al cuerpo del paciente (ver figura 1.3), la posición de las articulaciones del tren inferior surgen del cálculo realizado por el software del laboratorio a partir de las posiciones de los marcadores.

BTS ASCII format

Type: Point 3D tracks
Measure unit: m

Tracks: 69
Frequency: 250 Hz
Frames: 1842
Start time: 0.000

Frame	Time	sacrum.X	sacrum.Y	sacrum.Z	r asis.X	r asis.Y	r asis.Z	RK.X	RK.Y	RK.Z
878	3.512	NaN	NaN	NaN	1.242	0.581	0.244	1.385	0.274	0.278
879	3.516	NaN	NaN	NaN	1.238	0.581	0.244	1.379	0.273	0.278
880	3.520	NaN	NaN	NaN	1.234	0.582	0.245	1.373	0.273	0.278
881	3.524	NaN	NaN	NaN	1.231	0.582	0.245	1.366	0.272	0.278
882	3.528	NaN	NaN	NaN	1.228	0.583	0.246	1.359	0.271	0.278
883	3.532	NaN	NaN	NaN	1.224	0.583	0.246	1.353	0.271	0.278
884	3.536	NaN	NaN	NaN	1.221	0.583	0.247	1.346	0.270	0.278
885	3.540	NaN	NaN	NaN	NaN	NaN	NaN	1.339	0.269	0.278
886	3.544	NaN	NaN	NaN	NaN	NaN	NaN	1.331	0.269	0.278
887	3.548	NaN	NaN	NaN	1.203	0.582	0.251	1.324	0.268	0.278
888	3.552	NaN	NaN	NaN	1.201	0.583	0.252	1.317	0.268	0.278
889	3.556	NaN	NaN	NaN	1.198	0.583	0.252	1.310	0.267	0.278
890	3.560	NaN	NaN	NaN	1.195	0.584	0.252	1.303	0.267	0.278
891	3.564	NaN	NaN	NaN	1.192	0.584	0.252	1.296	0.267	0.277
892	3.568	NaN	NaN	NaN	1.190	0.585	0.253	1.289	0.267	0.277
893	3.572	NaN	NaN	NaN	1.187	0.585	0.253	1.283	0.267	0.277

Figura 4.5: Extracto de un fichero de datos .emt

Cada columna del fichero de datos tiene un código identificativo que se corresponde con los homólogos en el cuerpo humano y cuya posición dentro del fichero puede variar de unos archivos a otros. Por esto es necesario determinar el orden de todas las columnas que necesitaremos en cada uno de los ficheros que se suministren como entrada. Los

códigos de las columnas varían dependiendo de si se trata de la posición de un marcador o una articulación. El identificador de un marcador está escrito en minúsculas y se compone de una letra (l en el caso de pertenecer a la extremidad izquierda o r si forma parte de la derecha) seguido de un espacio y el nombre que recibe el marcador. En la figura 4.6 se pueden observar los nombres que reciben todos los marcadores.

Por otro lado, el identificador de una articulación está escrito en mayúsculas y se compone de una primera letra, L o R, que define la extremidad y una segunda que define el tipo de articulación. Esta última letra puede ser H para la cadera (*hip*), K para la rodilla (*knee*), A para el tobillo (*ankle*) o T para el extremo del pie (*toe*).

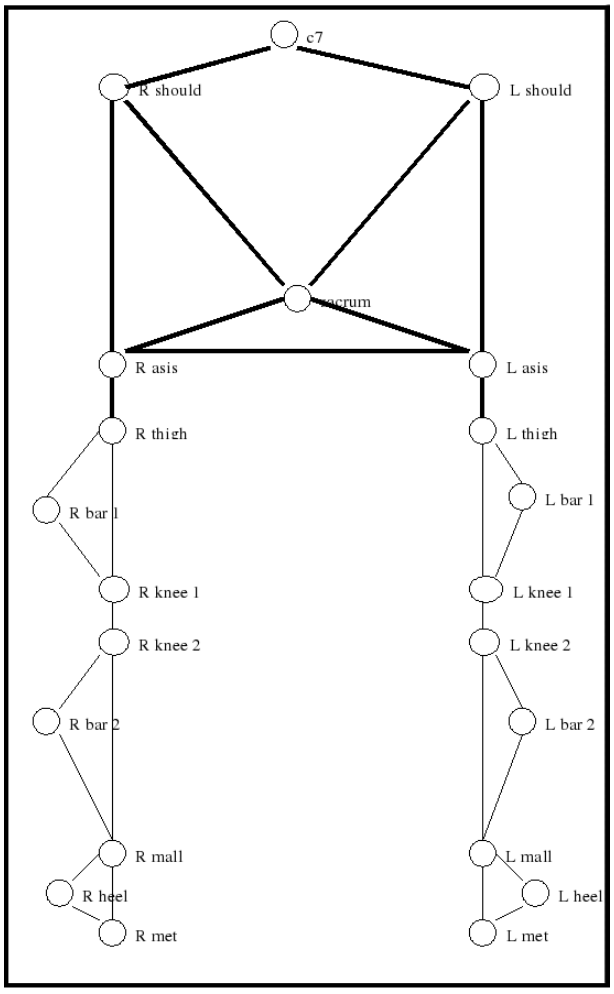

Figura 4.6: Diagrama de correspondencia entre los nombres de columna y marcadores.

Al lanzar el esquema, y antes de iniciar la animación se realiza una lectura completa inicial del fichero, con el objetivo de determinar: *(a)* el orden de las columnas (p. ej: en el archivo de la figura 4.5 se determina que las posiciones X de la rodilla derecha

corresponde a la novena columna del fichero), *(b)* el tamaño del archivo para conocer la duración de la reproducción en tiempo real y *(c)* los valores mínimos y máximos que adquirirán cada una de las posiciones de las articulaciones.

El esquema lee la primera fila, que contiene las cabeceras de las columnas, y se guarda la posición de cada una de las columnas deseadas, que corresponden a las posiciones de las articulaciones de las caderas, rodillas, tobillos y pies. Posteriormente, en cada iteración se leerá una línea del fichero y se guardarán las posiciones de cada columna que contenga datos útiles para la aplicación. Mediante este proceso obtenemos las posiciones de cada articulación. Es importante señalar que la primera columna del fichero contiene el campo *Frame* que indica el número de fila. La segunda columna, *Time*, muestra el instante de tiempo en el que se sitúan las posiciones de esa fila. De esta forma, podemos saber el intervalo de tiempo real que separa una fila de otra (habitualmente 4 o 5 milisegundos, dependiendo del fichero) y acompasar el ritmo de reproducción real al ritmo del fichero de datos.

Para facilitar la carga de este fichero, el esquema incluye un botón (widget *GtkFileChooser*) que abre un diálogo de carga de ficheros que permite seleccionar cualquier archivo cuya extensión sea *.emt*, realiza la lectura completa inicial y comienza inmediatamente el procesamiento iterativo de las filas del fichero (ver figura 4.7)

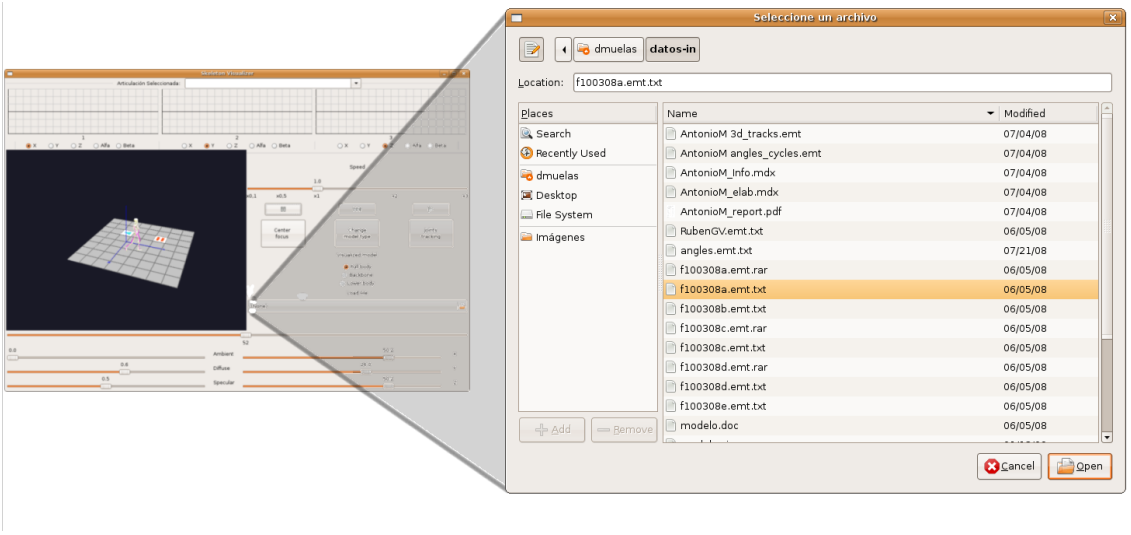

Figura 4.7: Detalle del widget de carga de ficheros..

Es importante señalar que es posible que dentro del fichero existan intervalos de tiempo sin datos para alguno o todos los marcadores. Esto es debido a que en esos

instantes dichos marcadores no estaban dentro del campo visual dde las cámaras que capturan el movimiento. Estos intervalos de tiempo sin datos son más comunes al principio y al final del fichero.

4.3. Visualización básica

Una vez conocidas las posiciones X,Y y Z de cada articulación, ya es posible pintar en *OpenGL* un modelo esquemático de la figura humana usando segmentos simples, como podemos ver en la figura 4.8. En esta sección describiremos la parte de nuestro código que prepara el entorno OpenGL y consigue esta visualización básica.

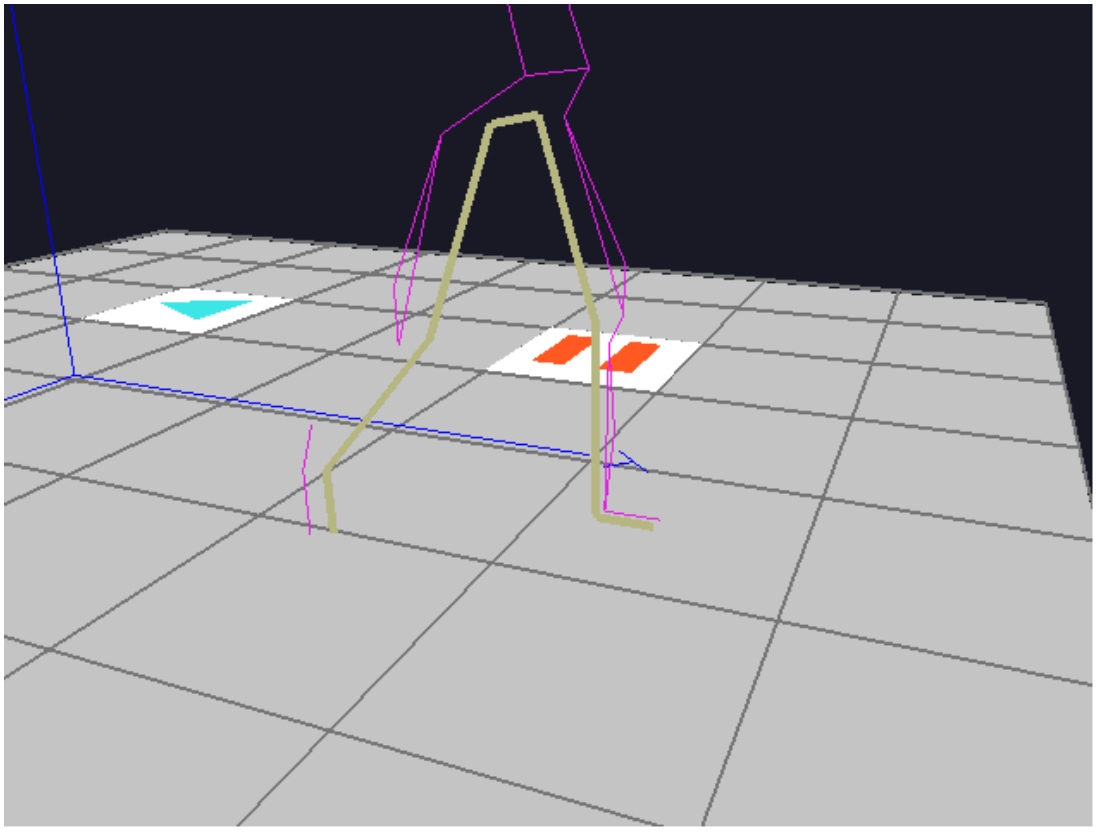

Figura 4.8: Modelo simple de figura articulada.

Para una mejor visualización se ha creado un entorno tridimensional mediante la biblioteca gráfica *OpenGL*, que permite crear modelos tridimensionales de una manera eficiente y sencilla. El entorno sobre el que se apoya la figura humana consiste en una

plataforma horizontal compuesto de baldosas cuadradas, unos ejes de coordenadas y dos botones en 3D cuyo funcionamiento explicaremos más adelante.

Para la visualización de la escena se utilizó la cámara virtual *pinhole* típica de *OpenGL*. Los parámetros de esta cámara son 45º de ángulo de apertura, una relación de aspecto (proporción entre su anchura y su altura) de 4:3, distancia al plano de recorte *near* de 0,001 y distancia al plano de recorte *far* de 50. La configuración de la cámara virtual se lleva a cabo mediante la instrucción *gluPerspective* de *OpenGL*:

```
gluPerspective(45.0, 640/480, 0.001, 50.0);
```

El entorno *OpenGL* usa un modelo simplificado para simular la iluminación en el mundo real que utiliza un algoritmo para calcular los valores de iluminación en un vértice determinado (*Phong*) y otro para difuminar los diferentes valores a través de la superficie de la primitiva (*ground*). La activación del modelo de luz se hace mediante la función *glEnable(GL_LIGHTING)* y luego es necesario al menos activar una luz mediante la función *glEnable(GL_LIGHT0)*, aunque pueden usarse hasta ocho tipos de luces, que van desde GL_LIGHT0 hasta GL_LIGHT7. Los parámetros para las luces son tres:

- *Componente Ambiental (Ambient)*. La luz ambiental es comúnmente asociada a la rayos que inciden sobre el objeto desde todas direcciones, por lo que no generan ningún tipo de sombreado sin importar la forma del objeto y su posición con respecto a la cámara. En la realidad este tipo de luz se origina cuando la luz rebota sucesivamente en diferentes superficies hasta que llega al objeto observado, haciendo parecer a este simplemente más claro, de manera homogénea.

- *Componente Difuso (Diffuse)*. En este caso los rayos provienen de una fuente puntual (la fuente es identificable), pero al caer sobre el objeto rebotan dispersándose en diferentes ángulos, por lo que el objeto aparecerá más iluminado en unas zonas que en otras. La forma en que el objeto es iluminado dependerá entonces de su forma y de la relación de posiciones cámara-objeto-fuente de luz.

- *Componente Brillo (Specular)*. Algunas fuentes de luz tienden a producir rayos compactos y paralelos, como por ejemplo el sol. Este tipo de luces cuando inciden sobre objetos *brillantes* como metales pulidos o vidrio, producen zonas donde la luz se refleja más intensamente, debido a que los rayos rebotan casi paralelos.

Esta configuración es altamente dependiente de la ubicación de la fuente de luz, el objeto, el observador y de la forma del objeto. Los objetos denominados *mate* son aquellos que poseen muy poco componente de brillo, debido a las características del material, como es el caso de los huesos de nuestra figura.

En nuestro entorno hemos incluído cuatro fuentes de iluminación posicionadas a cierta altura en cada una de las cuatro esquinas del suelo de baldosas y con un alto componente difuso y una luz principal especular.

Para facilitar el visionado de la figura, se han incluido tres diales, que mediante *callbacks* permiten orientar la luz principal del escenario a voluntad. También es posible configurar los parámetros iniciales de iluminación incluídos en el fichero de configuración del esquema.

Una vez configurado el entorno *OpenGL* ya es posible pintar una figura simple de segmentos. Partiendo de los valores con las posiciones de cada articulación procedentes del fichero de datos, creamos líneas (la primitiva GL_LINES en *OpenGL*) de forma que unen las articulaciones como corresponde. En el siguiente ejemplo se puede ver cómo se pinta el segmento que representa el fémur de la pierna derecha:

```
glBegin(GL_LINES);
   glVertex3f(pos_cadera_x, pos_cadera_y, pos_cadera_z);
   glVertex3f(pos_rodilla_x, pos_rodilla_y, pos_rodilla_z);
glEnd();
```

4.4. Visualización avanzada

Nuestro objetivo principal es crear una animación con un modelo tridimensional de un esqueleto humano que permita una recreación realista de la osamenta humana y no sólo una figura compuesta por líneas simples. En esta sección explicaremos cómo se ha dibujado la figura 3D del esqueleto humano.

Modelado del esqueleto 3D

Como base partimos de varios modelos óseos ya existentes en formato *.obj* pero formados de una sola pieza. Estos modelos fueron obtenidos gracias a la colaboración con el Grupo de Investigación en Ingeniería Biomédica de la Universidad Nacional de

Bogotá[1] y otros sitios web de modelado tridimensional[2]. Para poder simular movimiento de cada extremidad por separado era necesario tratar el modelo de forma que quedara desgranado en una serie de conjuntos de huesos, dividido por las articulaciones. Cada conjunto se almacenó en un archivo *.obj* diferente, de manera que pudieran cargarse en el entorno virtual de manera independiente. La aplicación utilizada para dicha tarea ha sido *Blender*, por las razones explicadas en el capítulo anterior.

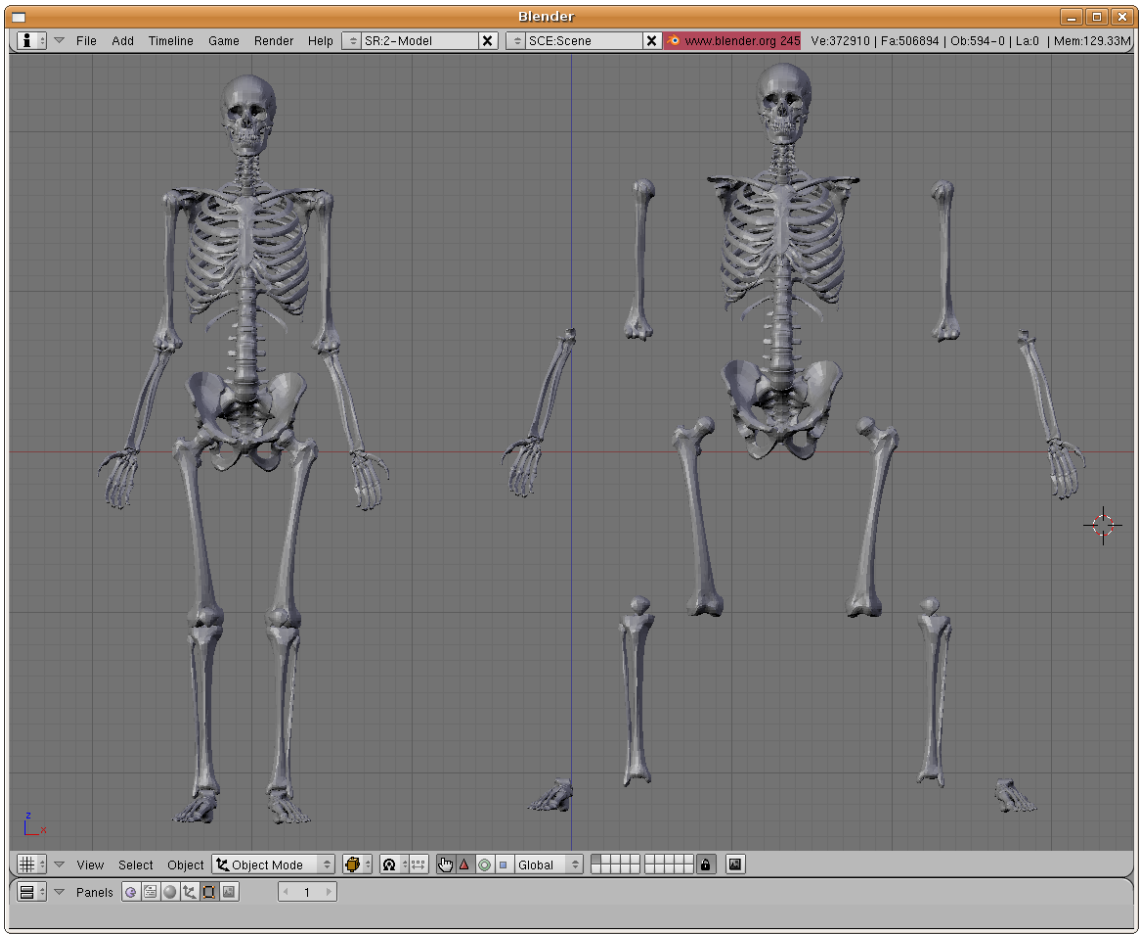

Figura 4.9: Tratamiento del esqueleto mediante *Blender*.

A partir del fichero original (*esqueleto.obj*) se crearon diez ficheros correspondientes a fémur izquierdo (*femur_i.obj*), pierna izquierda (*tibia_i.obj*), pie izquierdo (*pie_i.ob*), fémur derecho (*femur_d.ojb*), pierna derecha (*tibia_d.obj*) y pie derecho (*pie_d.obj*) y tres variaciones del tronco (*tronco_completo.obj*, *tronco_espina.obj* y *tronco_cadera.obj*). Alguno de los ficheros, como el *femur_i.obj* están compuestos por un sólo hueso, pero otros abarcan un conjunto de huesos, como el *pie_d.obj*, que está compuesto por todos los huesos que componen el pie derecho (falanges, metatarsianos, calcáneo...). Se

[1]http://www.bogota.unal.edu.co/
[2]http://www.sharecg.com/v/22098/3d-model/New-Anatomical-Skeleton-(OBJ)

decidió no incluir los huesos que conforman los brazos por mayor comodidad y porque el fichero de datos no incluye datos sobre las posiciones de las extremidades superiores. Para mayor comodidad, llamaremos *conjunto óseo* a todos los huesos que contenga cualquier fichero *obj*.

Es importante remarcar el proceso de orientación y posición de cada conjunto de huesos, que debía tenerse en cuenta a la hora de dibujarlos en el entorno virtual, ya que eso determina la posición y el ángulo iniciales de ese conjunto. Cada conjunto óseo tiene su origen de coordenadas en el extremo superior, a excepción del torso cuyo origen surge del coxal izquierdo.

Como hemos dicho anteriormente, cada uno de los conjuntos óseos están almacenados en archivos *.obj*. Estos archivos contienen tanto las posiciones de los vértices de todos los polígonos que componen cada hueso, como sus correpondientes vértices de las normales y las superficies.

Tanto el proceso de carga como de lectura y pintado se llevan a cabo mediante funciones incluidas en el archivo *obj_loader.c*. Durante el inicio del esquema se cargan todos los ficheros en estructuras de datos *obj_type* que almacenan todos los datos necesarios para pintar posteriormente el objeto:

```
typedef struct{
   int n_vertices;
   int n_normal_vertices;
   int n_surfaces;
   vert_type *vertices;
   vert_type *normal_vertices;
   surf_type *surfaces;
}obj_type;
```

La función *iteration* ya ha realizado el cálculo apropiado de los datos de entrada, por lo que ya conocemos las posiciones y ángulo de rotación que deben aplicarse a cada articulación, pero es en la función *guidisplay* donde se dibujan los huesos, mediante instrucciones *OpenGL*. Para pintar cada uno de los huesos, se hace una llamada a la función *glTranslate()* y dos llamadas a la función *glRotate()* (una por cada ángulo de rotación de la articulación), y posteriormente, se realiza el proceso de pintado del

conjunto óseo.

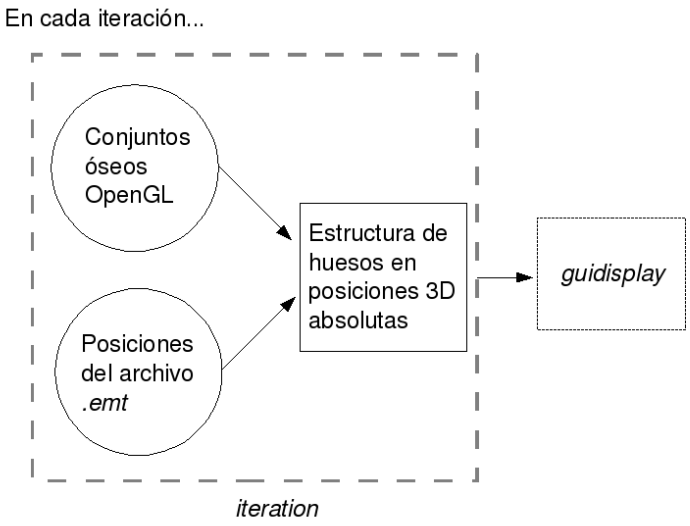

Figura 4.10: Esquema del proceso de dibujado de los huesos en cada iteración.

Este proceso se realiza en cada iteración para cada uno de los siete conjuntos óseos. Como funcionalidad adicional, se decidió incluir el dibujo de los marcadores, que se obtienen del fichero *.emt* de forma similar al método para obtener la posición de las articulaciones.

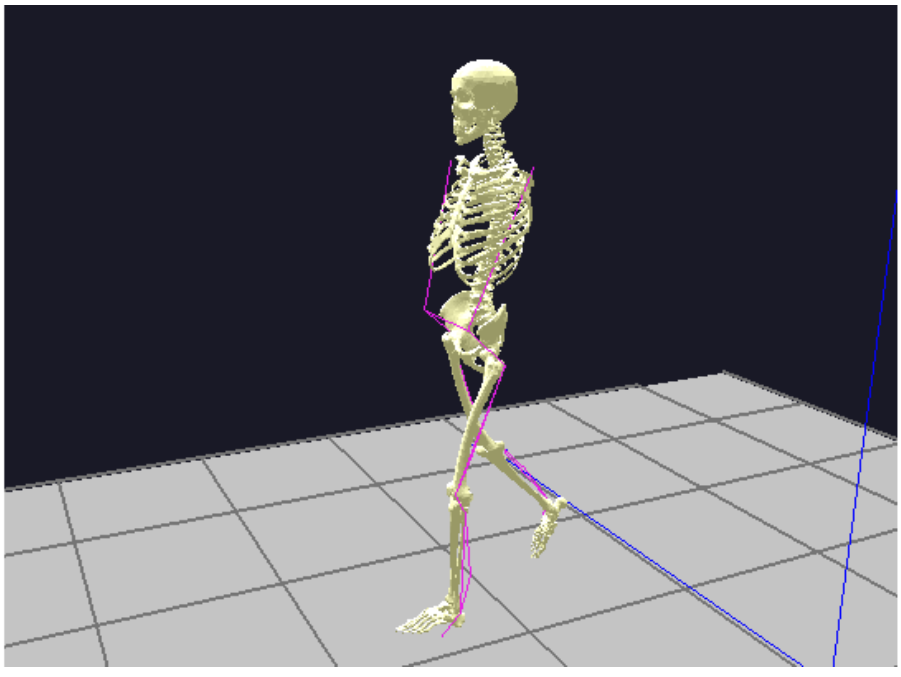

Figura 4.11: Imagen de la figura ósea como resultado final

Cálculo de ángulos de rotación

Ya que se van a pintar elementos tridimensionales con una orientación definida se hace necesario calcular los ángulos de giro de cada conjunto óseo. De esta forma, podremos anclar cada conjunto óseo en una posición (su posición de origen) y con la orientación correcta. En la figura 4.12 se puede comprobar cómo quedaría la figura del esqueleto sin aplicar las rotaciones pertinentes. Las articulaciones se sitúa inicialmente bien en su punto de referencia superior (el fémur comienza en la cadera, la tibia en la rodilla...) pero no conecta bien con el otro extremo por no estar bien orientado (el fémur no acaba en la rodilla, la tibia no acaba en el tobillo...).

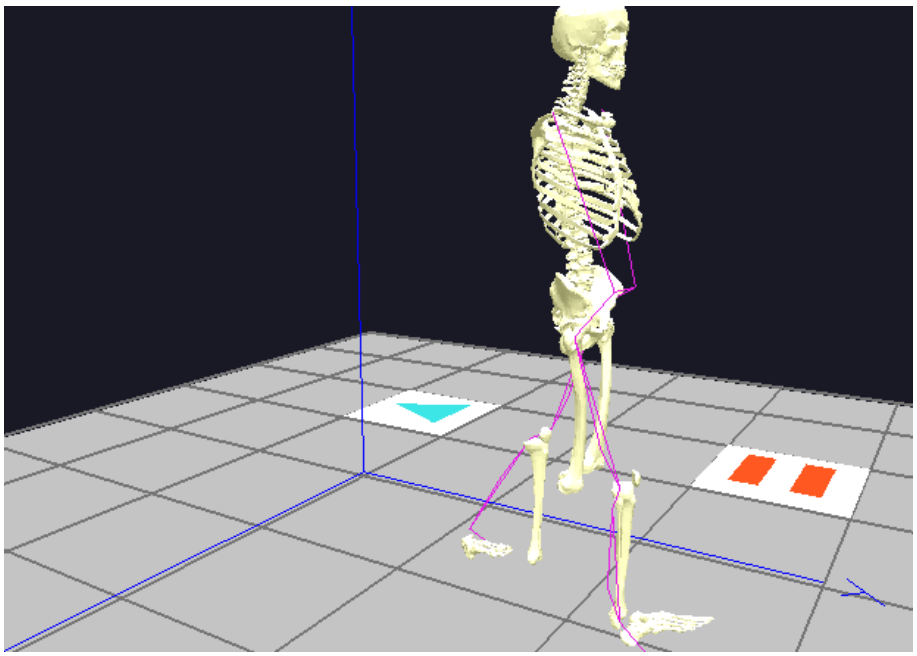

Figura 4.12: Visualización de cómo quedaría el esqueleto sin aplicar las rotaciones de los ángulos.

En general, es posible dibujar una rotación en un entorno tridimensional mediante el conjunto de dos ángulos si conocemos un punto de anclaje. En nuestro sistema, este punto de anclaje correponde al punto de referencia superior del conjunto óseo. Sin embargo, es necesario tener en cuenta que los conjuntos óseos son objetos orientados y cada articulación posee distintos grados de libertad. El cálculo de los dos ángulos que se deben aplicar en cada ocasión ha sido costoso y complejo. Tras un estudio cuidadoso, se ha concluido que cada articulación posee un eje de coordenadas sobre el que no se debe efectuar ninguna rotación. Por ejemplo, en el caso de la cadera, no debe aplicarse ninguna rotación sobre el eje Z.

Una vez determinados los ejes sobre los que se efectuarán las rotaciones, procedemos a calcular esos giros. Para este proceso, debemos calcular los dos ángulos proyectados sobre los planos correspondientes a cada uno de los ejes escogidos, que surgen del segmento existente entre la articulación deseada y la articulación hija.

Vamos a ver esto con un ejemplo: si queremos calcular los ángulos de rotación del fémur izquierdo debemos partir del ángulo que forma el segmento existente entre la posición de la cadera izquierda y la de la rodilla izquierda. Este segmento proyectará una sombra sobre el eje XZ, con el que obtendremos el ángulo a aplicar sobre el eje Y, y otra sombra sobre el plano YZ, con el que obtendremos el ángulo para el eje X.

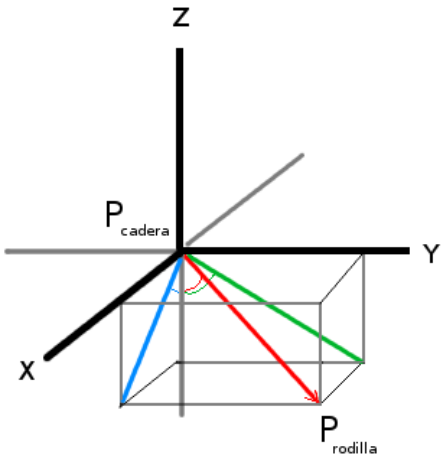

Figura 4.13: Ejemplo de cálculo de ángulos de rotación de la cadera.

Estos ángulos se calculan en la función *iteration* para visualizar cada extremidad en el *guidisplay*. En concreto, se dibujarán mediante el uso de la función *glRotate()* dos veces seguidas, una por cada ángulo que se debe aplicar a cada articulación.

Escalado

Una vez configurados los parámetros iniciales de *OpenGL*, es necesario realizar una normalización de tamaños, de forma que se escale el conjunto de huesos en *OpenGL*, que tienen su tamaño natural, al tamaño real de los huesos del paciente.

Para realizar este cálculo se obtiene el valor de la distancia entre la articulación de la cadera izquierda y la articulación de la rodilla izquierda. Luego, la longitud del fémur izquierdo modelado, que es 111 cm., se divide entre la distancia obtenida y así se obtiene el valor de la escala que se aplicará todos los objetos de la figura mediante la función *glEscalef()*. Para hacer más robusto el cálculo de la distancia entre la cadera y la rodilla se realiza este cálculo en 10 líneas de fichero obtenidas al azar, se eliminan los posibles resultados nulos debidos a ausencia de datos en el archivo y se obtiene la media aritmética.

El factor de normalización, que se ha calculado sobre el fémur, se aplica a todos los conjuntos óseos.

4.5. Animación de la figura

En este apartado vamos a explicar el método escogido para realizar la animación de la figura del esqueleto humano. Es posible crear animaciones mediante el uso de posiciones relativas o absolutas. La diferencia entre ambos métodos radica en la forma de dibujar cada conjunto óseo en el entorno virtual.

Usando posiciones relativas se posiciona cada parte de una estructura articulada a partir de la anterior. Este método está muy relacionado con la técnica de animación 3D conocidad como *cinemática directa*. Un ejemplo típico de estructura jerárquica sobre el que realizar ésta técnica es un robot, formado por cuerpos rígidos enlazados por articulaciones. Se puede establecer un sistema de referencia fijo situado en la base del robot, y describir la localización de cada uno de los eslabones con respecto al eslabón anterior. Una pieza rígida A depende jerárquicamente de otra B si, para alcanzar la parte fija de la estructura (base del robot) desde A, se debe pasar por B. Para calcular la posición de una pieza rígida de la estructura se deben calcular las posiciones de todas las piezas de las que depende. Bibliotecas gráficas de uso generalizado, como *OpenGL*, están diseñadas para facilitar estos cálculos y llevarlos a cabo eficientemente mediante pilas de matrices.

Por otra parte, también es posible usar las posiciones absolutas de cada parte de la estructura para dibujarlas en el entorno sin usar sistemas de coordenadas relativas. En nuestro caso, al partir de las posiciones absolutas de cada articulación, vamos a

utilizar este método. Cada conjunto óseo tomará como punto de referencia, la posición en coordenadas absolutas de su correpondiente articulación y se aplicarán los ángulos de rotación respecto a ese punto.

La animación de nuestra figura ósea se lleva a cabo mediante la llamada iterativa de la función *guidisplay* (hebra de visualización), que muestra los datos calculados en la función *iteration* (hebra del esquema). Como hemos mencionado, en *iteration* el esquema se encarga de leer la fila del fichero *.emt* que corresponda y asigna el valor de cada columna a su correspondiente variable, tras tratar la información. Es importante señalar que la función *iteration* se lanza a un ritmo controlado y no lee todas las líneas del fichero de datos sino que sólo lee la fila que corresponde en cada iteración.

Para determinar la fila que se debe leer en cada instante nos ayudamos de la columna *Time* que indica el momento absoluto en el que debe leerse los datos de la línea a la que pertenece. La función se encarga de comparar este valor con el tiempo real de ejecución y así corregir en cada iteración la próxima fila que debe leer, de manera que depende menos de la velocidad de procesamiento del equipo en el que se lance la aplicación. La función *guidisplay* también funciona a un ritmo independiente.

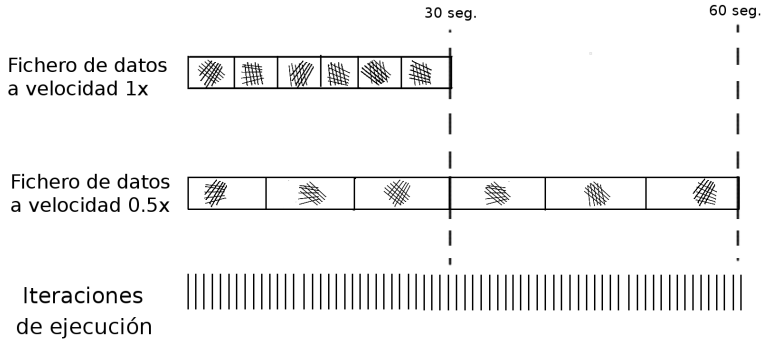

Figura 4.14: Diagrama de relación entre la velocidad de lectura del fichero y el ritmo de la función *iteration*.

En la interfaz gráfica de la aplicación se ha incluido un dial que permite configurar la velocidad de reproducción a la que se quiere contemplar la acción. Este valor de velocidad se aplica directamente en la función *iteration* a la hora de determinar la fila a leer. Si la velocidad de reproducción se pusiese a la mitad, por ejemplo, *iteration* leería el doble de las filas del fichero de carga de las que normalmente lee (figura 4.14). Cabe señalar que el dial de velocidad permite alterar la velocidad de reproducción,

pero la velocidad de ejecución, el ritmo de *iteration*, siempre permanece constante.

En ocasiones, resulta muy útil para el médico observar un intervalo concreto del movimiento de paciente. Para ello hemos incluido una barra de progreso que sirva tanto para indicar el momento temporal en el que se encuentra en ese momento como para poder situar la acción en el instante que se desee, como si de una película se tratase.

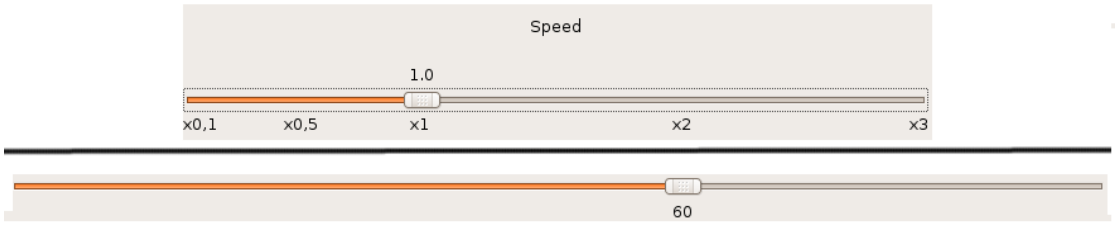

Figura 4.15: Detalle de los diales de graduación de velocidad y barra de progreso.

Además, arrastrando ese dial permite contemplar en avance rápido o marcha atrás. Debido a que el fichero de carga *.emt* puede tener un tamaño (y por lo tanto, duración) variable se ha decidido utilizar un sistema porcentual de medición del tiempo. Así, se calcula el tamaño total del archivo durante la primera lectura del mismo y cada vez que se altere la posición del dial de la barra se mueve el puntero del archivo a la posición deseada. El funcionamiento de este *widget* puede verse aquí:

```
int rebobinado= gtk_adjustment_get_value(
     gtk_range_get_adjustment(glade_xml_get_widget(xml,"tempo")));
if (rebobinado != rebobinado_anterior){
   //Situamos el puntero del fichero al comienzo de la linea
   long porcentaje= rebobinado*0.01*tamanio_fichero;
   lseek(fichero,porcentaje,SEEK_SET);
   leidos= read(fichero,buffer,3330);
   for (i=0; i<3300; i++){
      if (buffer[i]=='\n'){
         //Esta es la posicion de comienzo del puntero
         encuentra= i+1;
         exit;
      }
   }
   posicion_final= porcentaje+encuentra;
```

```
   position= porcentaje+encuentra;
}else{
   //Actualizamos la posición de la barra de progreso
   int tantoPorCiento = (posicion_final*100)/tamanio_fichero;
   gtk_adjustment_set_value(
      gtk_range_get_adjustment(glade_xml_get_widget(xml,"tempo")),
      tantoPorCiento
   );
}
rebobinado_anterior=rebobinado;
```

Adicionalmente a la barra de progreso, se han incluído en el interfaz de la aplicación tres botones, con diversas utilidades. El botón *Pause* permite pausar el movimiento de la figura humana, así como las gráficas de detalle de las articulaciones. El botón *Step* reactiva el movimiento de la figura durante un instante para luego congelar el esqueleto humano. Este botón permite avanzar de forma controlada el movimiento de la figura. Por último, el botón *Play* reactiva el movimiento de la figura en el caso de que estuviera pausada.

Finalmente se ha desarrollado una funcionalidad adicional que consiste en dejar una estela de las posiciones que han tenido anteriormente las articulaciones del tren inferior. Se puede ver un ejemplo de estas estelas en la figura 4.16. Esta funcionalidad puede activarse pulsando el botón *Joints tracking* en el GUI.

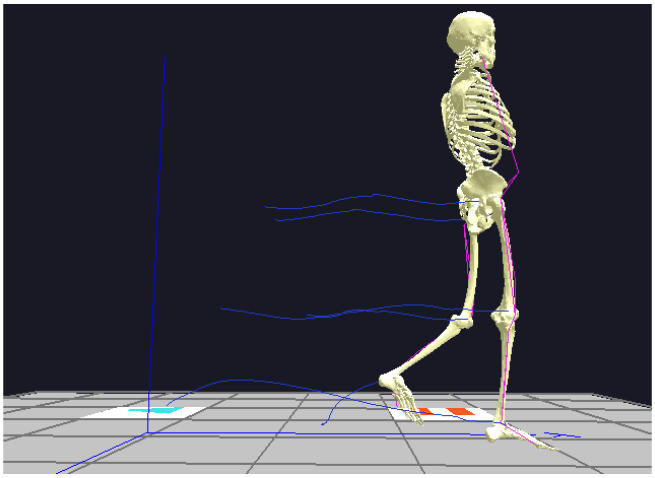

Figura 4.16: La figura ósea en movimiento, dejando tras de sí las estelas de las articulaciones.

4.6. Controles de cámara virtual

Para facilitar el manejo de la aplicación se ha desarrollado un interfaz ergonómico que facilite la visión de la figura en movimiento. Se han creado unos controles que, mediante el uso del ratón únicamente, permiten controlar la cámara virtual desde la que se observa nuestro entorno 3D y el esqueleto. *OpenGL* ofrece una cámara virtual cuya posición y orientación pueden cambiarse. En nuestro caso, dicha cámara ha sido definida de tipo *PinHole* y modelada adicionalmente con la biblioteca *Progeo*.

Estos controles de cámara se hacen alterando los valores de las variables que componen la estructura del tipo de dato *TPinHoleCamera*, que se define de la siguiente manera:

```
typedef struct {
  float X;
  float Y;
  float Z;
  float H;
} HPoint3D;

typedef struct {
  HPoint3D position; /* camera 3d position in mm */
  HPoint3D foa; /* camera 3d focus of attention in mm */
  float roll; /* camera roll position angle in rads */
  float fdist; /* focus distance in mm*/
  float u0,v0; /* pixels */
  /* camera K matrix */
  float k11,k12,k13,k14,k21,k22,k23,k24,k31,k32,k33,k34;
  /* camera rotation + translation matrix */
  float rt11,rt12,rt13,rt14,rt21,rt22,rt23,rt24,
     rt31,rt32,rt33,rt34,rt41,rt42,rt43,rt44;
  /* top right and bottom left points */
  HPoint3D tr, bl;
  char name[256];   /* name */
}TPinHoleCamera;
```

Con el ratón se puede cambiar la posición y la orientación de esa cámara virtual en el mundo 3D. De esta manera, al pulsar el botón izquierdo y arrastrar el ratón se cambia la posición de la cámara virtual dentro del entorno virtual. Mediante esta operación, modificamos el valor *position* de la cámara.

La rueda central del ratón permite variar la distancia focal, alejando o acercando la cámara virtual de su foco de atención.

Al pulsar el botón derecho y arrastrar el ratón se mueve el foco de atención u orientación de la cámara, inicialmente situado en el origen de coordenadas de la escena. Con este *callback* alteramos los valores de la variable *foa* de la cámara. En la interfaz gráfica se incluye un botón(utilizando el widget *GtkButton*) que permite resituar el foco de atención en el origen de coordenadas (0,0,0) para evitar que el usuario se pierda permanentemente en el entorno si sitúa accidentalmente el foco de atención muy alejado del escenario.

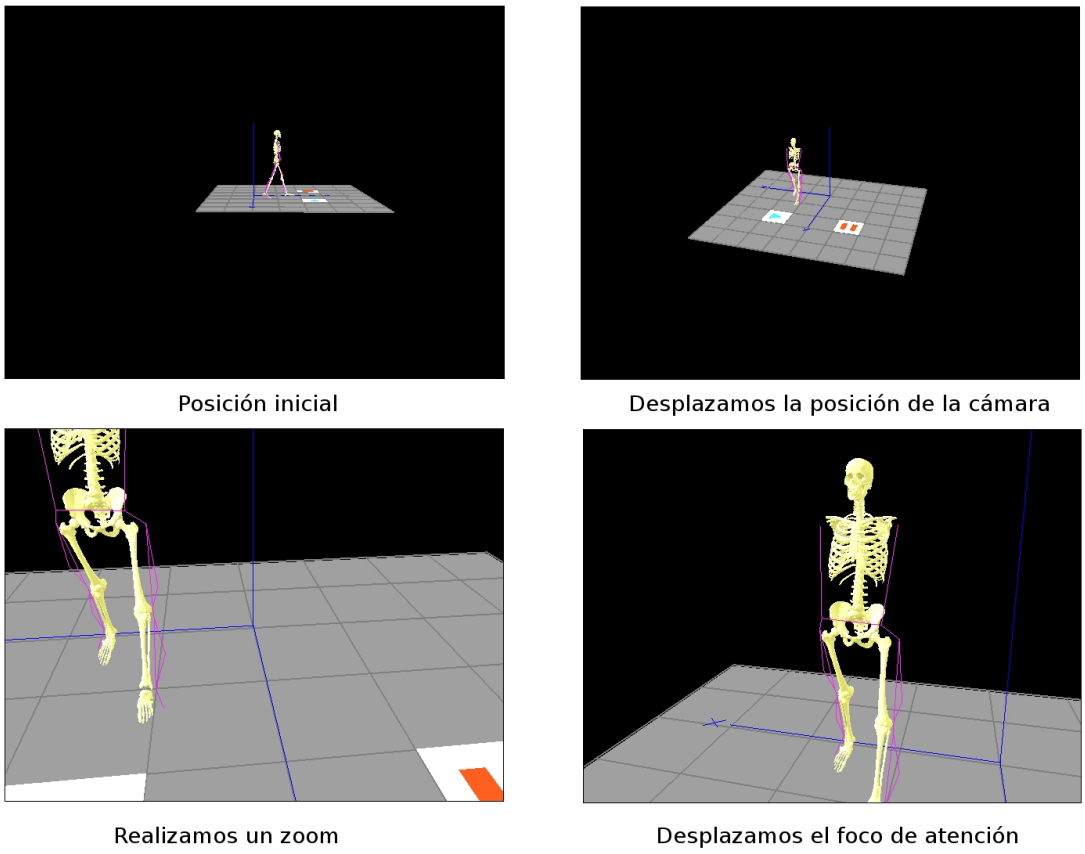

Figura 4.17: Ejemplo de uso de todos los controles de cámara.

Todas estas funcionalidades se llevan a cabo mediante *callbacks* que captan los eventos del ratón y llaman a las funciones correspondientes.

4.7. Interfaz de articulaciones

Un objetivo secundario del proyecto consistía en permitir al usuario visualizar la evolución temporal de las posiciones (coordenadas X,Y,Z) y orientaciones de alguna articulación concreta del paciente mientras observa la animación de la figura. Dichas gráficas pueden visualizar distintos datos a elección del usuario y pueden corresponder a cualquier de las seis articulaciones del tren inferior del cuerpo disponibles: cadera izquierda, cadera derecha, rodilla izquierda, rodilla derecha, tobillo izquierdo y tobillo derecho. Las articulaciones disponibles comprenden tan solo el tren inferior del cuerpo ya que se trata de una aplicación para el laboratorio de análisis de marcha.

De cada articulación pueden mostrarse tres gráficas con la evolución temporal de la posición de la articulación con respecto a un eje de coordenadas concreto, o uno de los ángulos de rotación de esa articulación. La forma de seleccionar qué información se quiere visualizar en cada gráfica se han dispuesto series de cinco botones de radio, llamados *X*, *Y*, *Z*, *Alfa* y *Beta* para que el usuario marque los datos que prefiere que sean mostrados. Estos datos representan la posición de la articulación en el espacio y los dos ángulos de rotación principales de dicha articulación. Por ejemplo, en el caso de la rodilla, Alfa y Beta corresponderían a los valores de los ángulos sagital y coronal respectivamente.

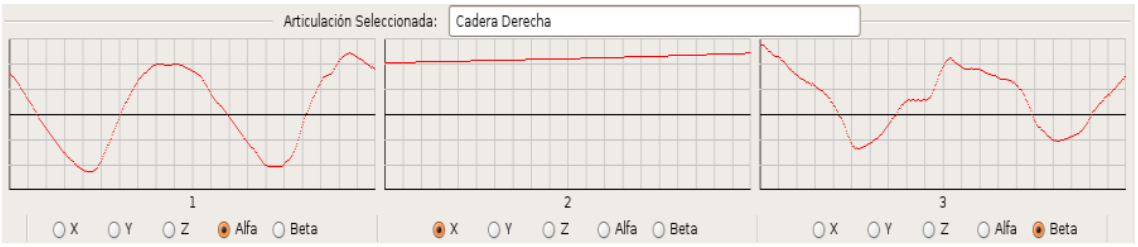

Figura 4.18: Ejemplo de gráficas en el que se muestran la evolución temporal de los ángulo de rotación y la posición del eje X de la cadera derecha.

Existen dos métodos para seleccionar la articulación cuya información se desea visualizar. En la interfaz gráfica de la aplicación hay un menú desplegable que permite

seleccionar cualquiera de las seis articulaciones disponibles. Además, también es posible seleccionar una articulación haciendo click con el botón izquierdo del ratón en dicha articulación o sus cercanías, dentro del entorno *OpenGL*. Mediante la biblioteca *Progeo* se realiza la función de *retroproyección* y obtenemos la proyección tridimensional que corresponde al píxel señalado mediante el ratón. Así, si la proyección pasa cerca de la posición tridimensional de una de las articulaciones se selecciona y se muestran las gráficas deseadas de esa articulación.

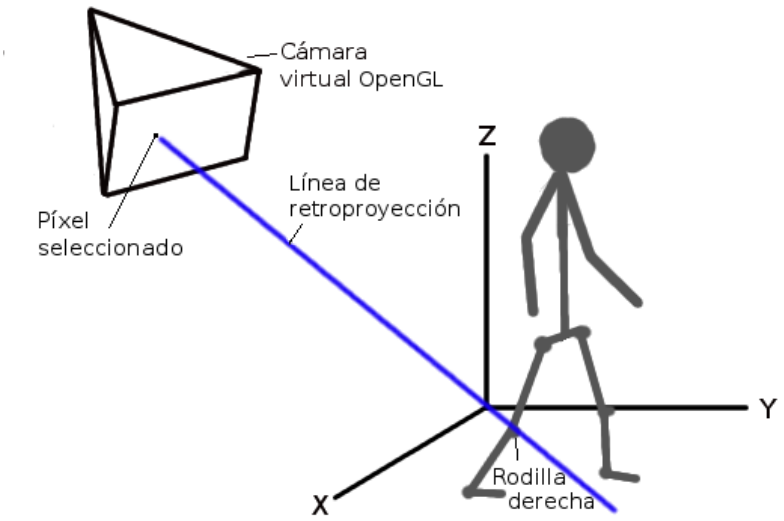

Figura 4.19: Esquema de funcionamiento de Progeo en nuestro esquema. El píxel seleccionado se proyecta cerca de la posición de la articulación.

Las gráficas se crean de forma dinámica a tiempo real para permitir la comparación de la figura ósea y la gráfica temporal en cada instante. Los datos necesarios para dibujar cada gráfica (los cuatrocientos últimos valores leídos) se almacenan en un *array* que se va actualizando en cada iteración. Cuando se llena el array, los datos se sobrescriben cíclicamente por lo que el gráfico siempre muestra los últimos datos.

Para la construcción del GUI de las gráficas se han recurrido al uso de canvas propio de *Glade*, llamado *drawingarea*, para construir, partiendo desde cero, las gráficas deseadas mediante formas simples (rectas, puntos...).

Para normalizar la visualización de cada gráfica se tienen en cuenta los valores mínimos y máximos (determinados en la primera lectura completa del fichero) que toman los datos de cada gráfica. Con esos datos, se realiza un autoescalado para cada

gráfica sin que el usuario deba preocuparse por los posibles valores de escalas.

Capítulo 5
Experimentos

En el capítulo anterior hemos comentado la solución final que hemos programado. En este capítulo explicaremos qué pruebas y experimentos hemos llevado a cabo con ella para validarla y ir mejorándola según los distintos prototipos realizados en el proceso de desarrollo. Además, mencionaremos algunas implementaciones alternativas significativas, analizaremos su rendimiento y describiremos los experimentos realizados.

5.1. Ejecución típica

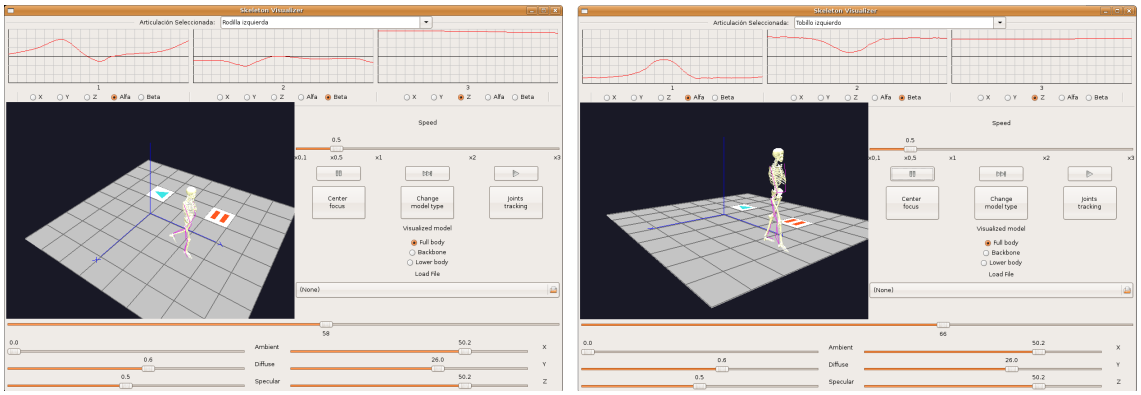

Figura 5.1: Proceso de una ejecución normal del esquema Skeleton_visualizer.

Para comprobar el funcionamiento correcto de la aplicación se llevó en primer lugar una prueba general. En la figura 5.1 se puede observar el GUI de la aplicación en dos instantes de su ejecución. El dial de velocidad de reproducción está situado en x0.5. La fuente de iluminación principal está situada en la posición (50.2,26.0,50.2) y con los componentes *Diffuse* y *Specular* a 0.6 y 0.5, respectivamente. En el primer instante, podemos ver que las gráficas muestran la evolución temporal de las dos rotaciones y la posición Z de la rodilla izquierda. En el segundo instante, tras haber seleccionado el tobillo izquierdo, se muestran su correspondiente evolución temporal.

Se puede ver un video del funcionamiento normal de la aplicación en mi web personal[1].

5.2. Evolución de la visualización OpenGL

El objetivo fundamental de este proyecto era crear una herramienta que permitiera la visualización de una figura ósea en movimiento. Por este motivo, era importante lograr que la animación de la figura fuera correcta en todo momento. Para llegar al resultado final se pasó por diferentes etapas.

En primer lugar se creó un entorno *OpenGL* sencillo y se dibujó una figura antropomórfica esquemática mediante formas simples como rectas y esferas. En esta fase se comprobó que el procesamiento de datos del fichero de entrada era correcto y que se podía conocer la localización tridimensional de cada articulación.

A continuación se sustituyeron las formas simples dibujadas con *OpenGL* por los conjuntos óseos procedentes de los ficheros *.obj*. Sin embargo, al no aplicar rotaciones, los objetos siempre quedaban orientados en su posición de reposo y parecían desconectados entre sí, tal y como pudimos ver en la figura 4.12 del capítulo anterior.

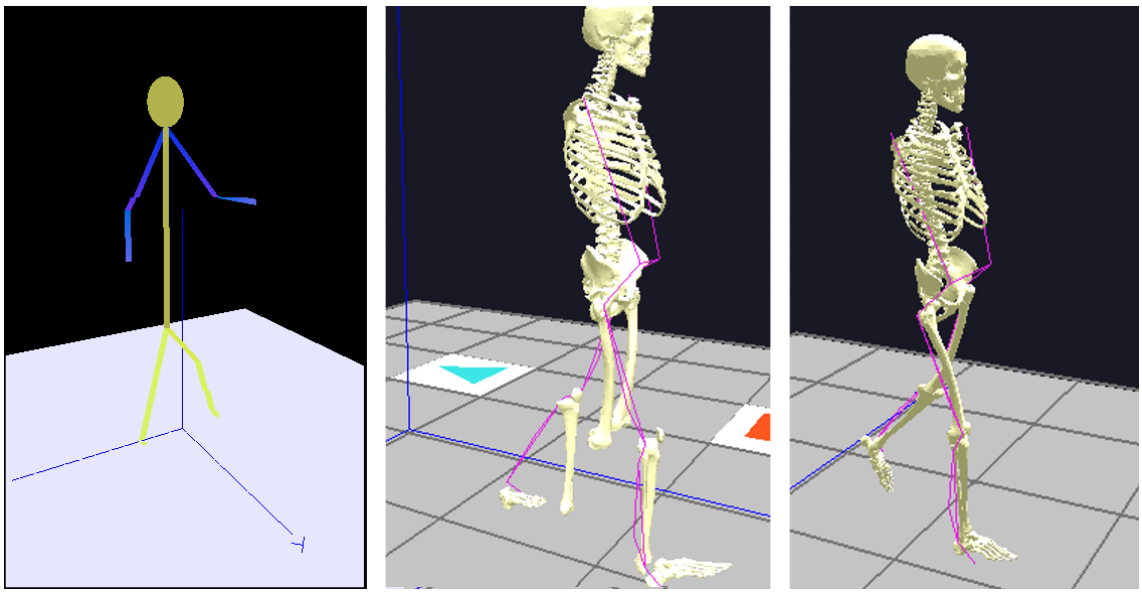

Figura 5.2: Proceso de evolución de la figura del esqueleto humano.

Tras el cálculo de los ángulos y diversas pruebas menores llegamos al resultado final, que permite la correcta visualización de la figura del esqueleto humano en movimiento

[1] http://jde.gsyc.es/index.php/Dmuelas_3D_skeleton_visualizer

como podemos ver en la figura 5.5.

5.3. Ajuste de la iluminación del entorno

Otro factor importante a la hora de facilitar el análisis y la observación del visualizador es conseguir una óptima iluminación el entorno tridimensional simulado. *OpenGL* utiliza un sistema bastante complejo para conseguir distintos efectos de iluminación.

Para obtener una visibilidad adecuada es necesario configurar las diferentes fuentes de luz según su tipo, que ya describimos en el capítulo 3 y se dividen en luz emitida (POSITION), difusa (DIFFUSE), especular (SPECULAR) y ambiental (AMBIENT). Además, se deben configurar correctamente los materiales de los que están compuestos los objetos, de manera que las luces del entorno se reflejen en ellos de forma correcta, tanto en brillo como en color.

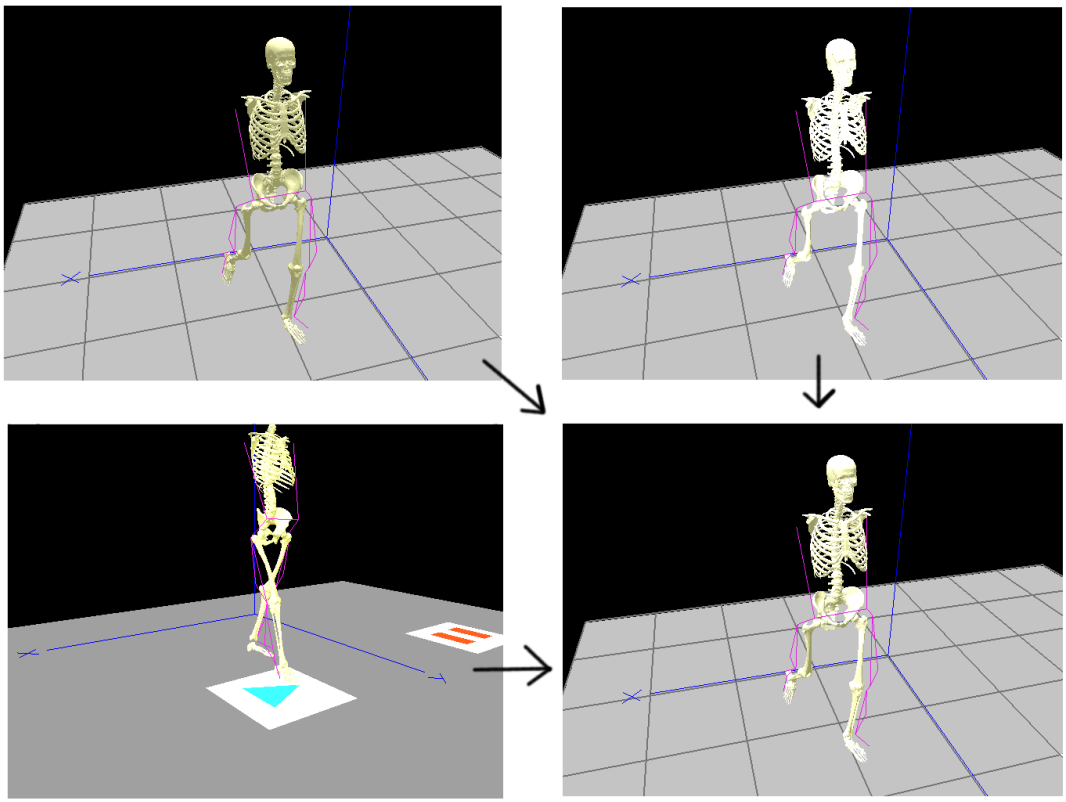

Figura 5.3: Pruebas de iluminación hasta alcanzar el efecto deseado.

Como podemos observar en la imagen superior, se realizaron varias pruebas hasta obtener un correcto resultado final. En el recuadro superior izquierdo la iluminación resulta demasiado tenua, al contrario que en el recuadro a su derecha. Por otra parte, en la imagen inferior izquierda se utilizó una luz ambiental demasiado intensa. En el recuadro inferior derecho se puede observar el resultado final. También se han incluído los valores predeterminados de estos parámetros en el fichero de configuración del esquema. Sin embargo, se decidió añadir unos diales que permitieran al usuario modificar a voluntad los parámetros de las luces de forma que se pueda obtener un resultado óptimo en un momento determinado adecuado a los gustos de usuario.

5.4. Alternativas probadas

Durante todo el desarrollo de la aplicación se han probado múltiples alternativas, tanto de diseño como de implementación, en distintos aspectos de nuestra aplicación. En esta sección vamos a describir las alternativas probadas más destacables.

Animación de la figura y consumo de CPU

Para comprobar el consumo de CPU de nuestra aplicación, utilizamos dos equipos distintos: *Órdago*, con 4 procesadores Intel Core 2 Quad de 2.5 GHz y 3 GB de memoria RAM, y *Gordo*, con un procesador Pentium 4 a 2.8 GHz y 512 MB de RAM. A continuación, vamos a describir las pruebas que se realizaron para optimizar la carga computacional.

La primera idea para realizar la animación de la figura fue leer y utilizar las posiciones de *todas* las filas del fichero. Esto resultó ser una implementación poco eficiente. El consumo de CPU era muy alto y el proceso de lectura y tratamiento de datos era tan costoso (una línea de datos cada 4 ms) que le resultaba imposible mantener el ritmo de ejecución necesario. En ambos equipos, cada iteración tardaba demasiado tiempo como para poder leer todas las líneas del fichero y seguir su ritmo temporal.

Posteriormente se optó por realizar la lectura intercalada de líneas, de manera que sólo se trataran una de cada diez líneas del fichero. Mediante esta técnica, las pruebas en *Órdago* demostraron que se había conseguido adaptar el ritmo de ejecución de la

función de tratamiento de datos con el tiempo de reproducción marcado por el fichero de datos y el consumo de procesador se redujo significativamente, pasando de un 50 % de uso de la CPU a un 38 %. Posteriormente, al activar la aceleración gráfica del equipo, el procentaje de uso de CPU descendió hasta el 33 %. Sin embargo, la aplicación en *Gordo* seguía yendo lentamente, ya que el ritmo de ejecución exigido al esquema era superior al que esta máquina puede proporcionar.

Para optimizar el funcionamiento en equipos de menor capacidad, se llevó a cabo un tercer desarrollo que mantuviera un ritmo controlado de reproducción. Como vimos en el capítulo 4, para llevar a cabo este control en cada iteración el esquema se encarga de calcular a qué línea pertenecen los datos que deberían pintarse en ese momento. De esta manera, si un equipo con menor CPU se ve forzado a ralentizar el ritmo con el que pinta la figura ósea eso no afecta al ritmo de lectura de datos. Este proceso se sigue realizando al ritmo normal y acompasadamente al ritmo real de reproducción que indica el fichero de datos. Con este desarrollo, las pruebas en ambos equipos fueron positivas; en *Gordo* el movimiento era mucho menos fluído, lógicamente, pero al ritmo correcto.

Gráficas de detalle de las articulaciones

Un objetivo adicional del proyecto era crear una funcionalidad que permitiera mostrar al profesional usuario gráficas útiles para el análisis del movimiento del paciente, que aparecerían al pulsarse con el ratón en una articulación del tren inferior. Estas gráficas pueden representar hasta cinco valores, representables en tres gráficas: la posición X de la articulación, la posición Y, la posición Z, el primer ángulo de rotación (denominado Alfa) de la articulación y el segundo ángulo de rotación (denominado Beta).

Para desarrollar esta funcionalidad se realizaron diferentes pruebas. En primer lugar, se intentaron crear las gráficas basándose en el widget de GTK: *GtkCurve*. Este widget permite crear curvas que cubran un rango de valores determinado, pero demostró ser bastante pobre y no ofrecía el resultado que deseábamos, como se puede comprobar en la figura 5.4.

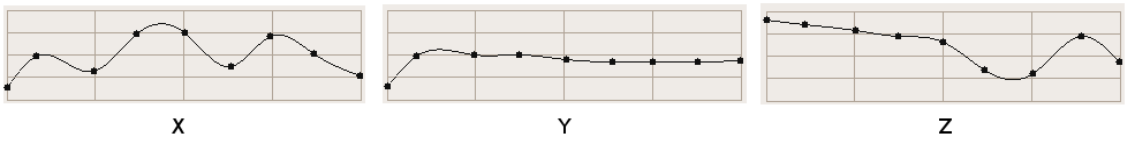

Figura 5.4: Ejemplo de gráficas obtenidas mediante el widget GtkCurve.

Posteriormente se probó con *GtkPlot*, un widget de la biblioteca GTK+ especialmente indicado para la creación de gráficas, pero demostró incompatibilidades con el método de creación de GUIs mediante archivos *.glade*. Así, finalmente se optó por crear las tres gráficas con *canvas*, mediante el widget *drawingarea* y el uso de formas simples.

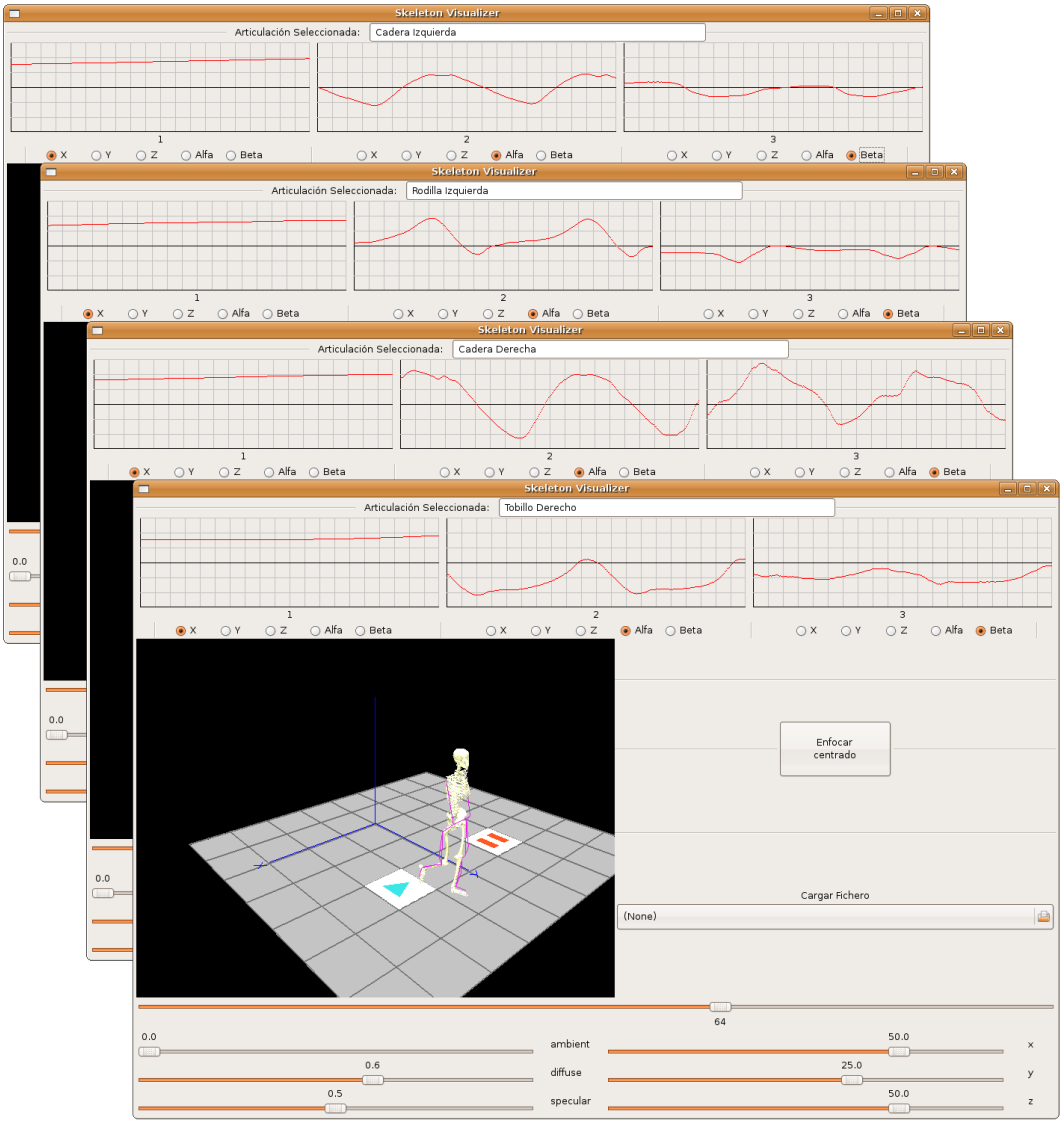

Figura 5.5: Compilación de gráficas de diferentes articulaciones de un modelo.

Para que las gráficas tuvieran una utilidad relacionada con el visualizador tridimensional y tuviera un componente dinámico, se diseñó el proceso de manera que las gráficas se mostraran conforme al avance de la figura en su entorno, de manera que se actualizan temporalmente de forma paralela al movimiento de la figura. En otras palabras, las gráficas están acompasadas al ritmo de reproducción de la figura. De igual forma, se ha comprobado que las gráficas también se mantienen estáticas al pausar el movimiento de la figura y se reactivan al mismo tiempo en el que se elimina la pausa.

5.5. Botones 3D

Para familiarizarse con la biblioteca *Progeo* se realizó un experimento que consistió en añadir dos botones 3D dentro del entorno de *OpenGL*, como dos baldosas del suelo virtual (ver figura 5.6). Estos botones, de *Pause* y *Play*, sirven para pausar el movimiento de la figura y reaundarlo respectivamente. Como vimos en el capítulo 3, la biblioteca *Progeo* permite, mediante la función *backproject*, que el usuario pinche en un píxel de la imagen, y si la recta retroproyectada de dicho píxel pasa por alguno de los dos botones se activa la acción indicada. El usuario puede pulsar estos botones desde cualquier posición y orientación de la cámara del observador, a través de la cuál se ve la escena.

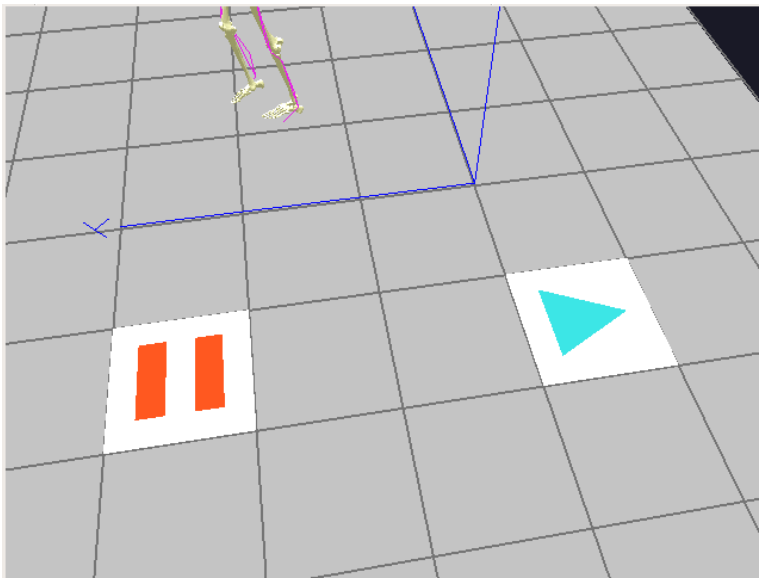

Figura 5.6: Los botones 3D de Play y Pause dentro del entorno virtual.

Este experimento fue la base para crear el sistema de selección de articulación al pinchar en la imagen 3D que ofrece la aplicación.

5.6. Visualización 3D inmersiva

Dentro del Grupo de Robótica de la URJC, Eduardo Perdices[2] ha desarrollado la aplicación *Headtracking*, que permite detectar la posición del usuario a través de un *Wiimote*, el mando de la videoconsola *Wii*. El mando tiene una cámara de infrarrojos y puede comunicarse con un ordenador mediante *Bluetooth*. Es necesario que el usuario lleve unas gafas con dos emisores LED que serán detectados por la cámara infrarroja y determinarán su posición.

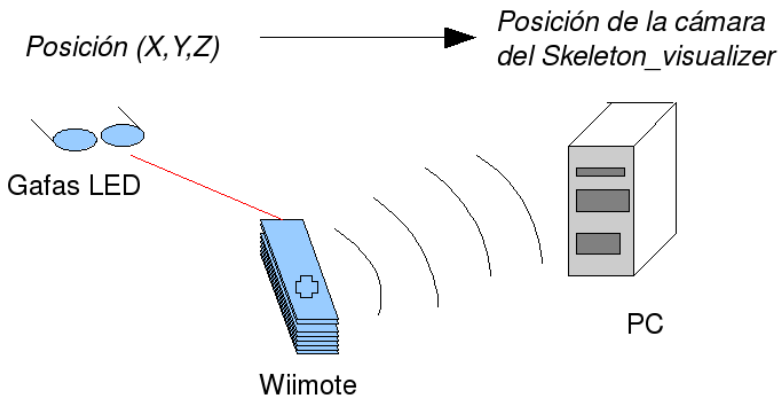

Figura 5.7: Esquema de funcionamiento del esquema Headtracking.

Nosotros hemos aprovechado la funcionalidad del esquema *Headtracking* en nuestra aplicación. De esta forma,la localización de la cámara virtual de *OpenGL* se actualiza en relación a la posición del usuario que capta el *Wiimote* (figura 5.8). Este experimento podría resultar de gran utilidad para el profesional, ya que facilitaría la visualización de la figura humana, pues el punto de vista de la escena varía de acuerdo al movimiento real del observador. Esto proporciona una mayor sensación de realismo y resulta más inmersivo para el usuario.

[2]http://jde.gsyc.es/index.php/User:Eperdes

Figura 5.8: Prueba de la incorporación del esquema Headtracking a nuestra aplicación.

Capítulo 6
Conclusiones y trabajos futuros

Después de explicar la implementación del sistema y los experimentos realizadas con él, cerramos esta memoria exponiendo las conclusiones obtenidas, así como las posibles posibles líneas de trabajo en el futuro.

6.1. Conclusiones

Haciendo un balance global, se ha programado una aplicación interactiva que permite ayudar al médico a comparar y medir la evolución de pacientes con problemas locomotrices bajo tratamiento. La aplicación permite la observación de los patrones de movimiento del paciente en un modelo esquelético de forma completa, con la posibilidad de pausar la acción, verla desde el punto de vista deseado y repetir la secuencia desde cualquier punto. Además, se ha implementado una funcionalidad que permite mostrar gráficas de información útil de cada articulación, favoreciendo el uso de esta aplicación. En conjunto, esta aplicación ofrece una utilidad adicional no existente en el laboratorio de análisis de movimiento del Hospital Niño Jesús.

Como conclusión final destacamos que se han cumplido los objetivos planteados en el capítulo 2 de esta memoria:

En primer lugar, nuestros dos primeros objetivos principales eran *desarrollar un visualizador interactivo de modelos de esqueleto*. Para ello hemos programado un esquema de *Jde* que permite recrear el movimiento de un paciente determinado mediante la figura tridimensional de un esqueleto compuesto por huesos humanos, no sólo segmentos planos. Este entorno tridimensional ha sido creado mediante la biblioteca *OpenGL*

Como se comentó en el capítulo 4 de esta memoria, se han implementado diversas mejoras con el objetivo de facilitar el diagnóstico por parte del médico, como pueden ser

los controles de cámara con el ratón, botones de pausa y reanudación del movimiento, controles de iluminación, un dial de configuración de la velocidad de reproducción y una barra de progreso para, además de indicar el momento en el que se encuentra, permite retroceder o avanzar el movimiento al instante deseado. Toda estas funcionalidades forma parte de la interfaz gráfica que ha sido creada gracias a la biblioteca *GTK*.

En segundo lugar, nuestro tercer objetivo era *desarrollar una funcionalidad de muestra de gráficos de detalle de articulación*. Como vimos en la sección 4.7, se ha implementado una funcionalidad que permita mostrar la evolución temporal de los ángulos y posiciones de una articulación concreta, pulsando en dicha articulación contenida en la figura dentro del entorno tridimensional o seleccionándola en el menú desplegable del GUI. Estas gráficas permiten ver los dos ángulos de giro principales y las posiciones en el espacio de cada articulación con respecto al tiempo y se actualizan dinámicamente según avanza la acción de la figura humana.

Estos objetivos tenían implícitos ciertos requisitos que condicionaban la solución que debíamos obtener. Estos requisitos se han resuelto de la siguiente manera:

- *Desarrollar la aplicación usando la plataforma JDE y bajo C*. Este requisito impone la programación en esquemas. La solución final es un esquema y el archivo *.glade* que se ejecutan bajo la plataforma *JDE*. Además, la aplicación se ha desarrollado integramente en el lenguaje de programación *C*, que es el lenguaje común a todos los esquemas de *JDE*.

- *Compatibilidad con el software existente en el laboratorio de análisis de marcha del Hospital Niño Jesús*. Por último, otro requisito importante es la compatibilidad que pueda ofrecer nuestro esquema con el software del laboratorio. Para obtener los datos del paciente, utilizamos archivos *.emt*, que son creados directamente por el software del laboratorio del hospital. Además, las gráficas del detalle de las articulaciones muestran información facilmente comparable con las gráficas que producen los informes que se generan en el laboratorio.

- *Carga computacional ligera*. Por último, otro requisito importante es que el esquema fuera comutacionalmente ligero. Inicialmente no se cumplió este requisito, pero tras diversas variaciones llegamos a un algoritmo, que se salta las líneas que no necesita leer del fichero de datos, más óptimo y computacionalmente más ligero.

En total, y de forma aproximada, se han escrito alrededor de 5000 líneas de código. La duración total del desarrollo ha sido algo más de un año de dedicación exclusiva a este proyecto fin de carrera.

Con relación a los conocimientos adquiridos durante el desarrollo del proyecto fin de carrera, se debe remarcar la gran cantidad de herramientas y técnicas aprendidas, lo que incluye la plataforma *JDE* como entorno de desarrollo sobre el sistema operativo *Linux*, el lenguaje de programación *C*, la aplicación de diseño gráfico *Blender*, la librería gráfica *OpenGL*, técnicas de geometría proyectiva y animación 3D, el sistema de control de versiones *SVN* y muchas otras herramientas muy importantes en lo que a desarrollo software se refiere, como *Trac*, *Latex*, etc.

6.2. Trabajos futuros

Este proyecto fin de carrera ofrece una gran cantidad de posibles líneas de trabajo futuras, pues presenta una aplicación escalable con muchas posibilidades de desarrollo.

Como líneas futuras inmediatas podríamos citar algunas funcionalidades adicionales que podrían ayudar al usuario médico a realizar los diagnósticos sin tener que recurrir a otros recursos. Un primera idea podría ser incrustar el video real del paciente en movimiento que capturan las cámaras del laboratorio de análisis de marcha en la interfaz gráfica, de manera que el movimiento del video y de la animación fueran acompasados. Así, el profesional podría comparar el movimiento real del paciente con el movimiento simulado de su estructura ósea.

Como opción alternativa también estaría la opción de grabar la animación tridimensional de la figura humana en un video de forma que pudiera reproducirse de manera independiente a la aplicación.

Otra posible linea de trabajo sería la incorporación de músculos en la figura del esqueleto, ya sea simplemente como lineas de color rígidas o auténticos objetos elásticos. La incorporación del sistema muscular permitiría determinar los momentos de flexión y extensión. En esta línea también podríamos sugerir incluír el uso de información

procedente de sensores electromiográficos.

Sin embargo, la forma de desarrollo más óptima debería llevarse a cabo tras un análisis de requisitos en colaboración con los profesionales del laboratorio de análisis de marcha. Teniendo en cuenta que los médicos son los usuarios finales de esta aplicación, éstos serán los más capacitados para decidir qué funcionalidades pueden resultar en definitiva más útiles.

Finalmente, y al márgen del laboratorio de análisis de movimiento, el presente proyecto fin de carrera podría servir como base sobre el que desarrollar futuros trabajos dentro del departamento de Robótica para el desarrollo de simuladores tridimensionales en entornos de *OpenGL* de robots humanoides, como el *Nao*, ya que posee piernas y brazos similares a los de nuestra figura,o de cualquier tipo de objeto articulado.

Bibliografía

[Anthony y Tromba, 2004] Jerold Anthony y Marsden Tromba. Cálculo vectorial 5º ed. *Addison-Wesley Professional*, 2004.

[Calvo, 2004] Roberto Calvo. Comportamiento sigue personas con visión direccional. *Proyecto fin de carrera Ing. Informática, Universidad Rey Juan Carlos*, 2004.

[Cañas Plaza *et al.*, 2007] José M. Cañas Plaza, Antonio Pineda, Jesús Ruíz-Ayúcar, José A. Santos, y Javier Martín. *Programación de robots con la plataforma jdec.* URJC, 2007.

[Cañas Plaza, 2003] José María Cañas Plaza. *Jerarquía Dinámica de Esquemas para la generación de comportamiento autónomo.* PhD thesis, Universidad Politécnica de Madrid, 2003.

[Erleben *et al.*, 2005] Kenny Erleben, Jon Sporring, Knud Henriksen, y Henrik Dohlman. Physics-based animation. *Charles Rivers Media*, 2005.

[Hearn y Baker, 2004] Donald Hearn y M. Pauline Baker. Computer graphics with opengl. *Prentice Hall*, 2004.

[Kachach, 2008] Redouane Kachach. Calibración automática de cámaras en la plataforma jdec. *Proyecto fin de carrera Ing. Informática, Universidad Rey Juan Carlos*, 2008.

[Rost, 2006] Randy J. Rost. Opengl shading language. *Addison-Wesley Professional*, 2006.

[Shreiner, 2004] Dave Shreiner. Opengl reference manual: The official reference document to opengl, version 1.4. *Addison-Wesley Professional*, 2004.

[Vega Pérez, 2008] Julio Manuel Vega Pérez. Navegación y localización de un robot guía de visitantes. *Proyecto fin de carrera Ing. Informática, Universidad Rey Juan Carlos*, 2008.

[Woo *et al.*, 2007] Mason Woo, Jackie Neider, Tom Davis, y Dave Shreiner. Opengl programming book: The official guide to learning opengl, version 2.1. *Addison-Wesley Professional*, 2007.

[Wright y Lipchak, 2005] Richard S. Wright y Benjamin Lipchak. Programación opengl. *Anaya Multimedia*, 2005.