Universitat
Autònoma
de Barcelona

4737-1 : Motion capture amb Microsoft Kinect

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per

Oriol López Contreras

i dirigit per

Enric Martí Gòdia
Jordi Arnal Montoya

Bellaterra, 12 de setembre 2012

Universitat Autònoma de Barcelona

Escola Tècnica Superior d'Enginyeria

El sotasignat, ...

Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en

I per tal que consti firma la present.

Signat: .....

Bellaterra, ...de.....de 200.....

Agradecimientos

Dedicamos esta sección para las personas que han dado apoyo y han contribuido a la realización de este proyecto.

Al los directores del proyecto Enric Martí Gòdia y Jordi Arnal Montoya, por la ayuda y guía, así como su apoyo que han otorgado durante todo el proyecto.

Al profesor Lluis Martínez Huguet, por la realización del modelo 3D usado en el proyecto así como su ayuda en un aspecto necesario para la realización del trabajo.

Al profesor Asier Ibeas Hernández, por sus consejos y su ayuda ofrecida para el proyecto.

A Jose Carlos Pujol Alcolado, por su ayuda y apoyo en el proyecto.

Resumen

Castellano:

El proyecto trata de convertirse en una herramienta para animadores 3D, tanto para los que hacen películas como para los que modelan videojuegos, que necesiten de un software para simplificar el trabajo que conlleva animar un modelo 3D. Todo sin necesidad de usar trajes especializados. El proyecto, usando Kinect, convertirá los movimientos captados por la cámara y los agregará al modelo, creando una animación basándose en los movimientos reales de una persona.

Català:

El projecte tracte de convertir-se en una eina per a animadors 3D, tant pels que fan pel·lícules com pels que modelen videojocs, que necessiten d'un software per simplificar el treball que comporta animar un model 3D. Tot sense necessitat d'utilitzar vestits especialitzats. El projecte, utilizant Kinect, convertirà els moviments captats per la càmara i els agregarà al model, creant una animació basant-se en els moviments reals d'una persona.

English:

The project means to serve as a tool for 3D animators, either film animators or videogame designers, that need a software to simplify the work that carries animating a 3D model. All without the need of specialized suits. This project, using Kinect, will convert the movements captured by the camera and then adding them to the model, creating an animation based on the real movements of a person.

Índice

1. INTRODUCCIÓN

Desde sus inicios, Kinect ha ido ganando popularidad por sus capacidades para detectar en tiempo real información visual del entorno, en especial información de personas. Pero antes de continuar hay que tener una idea clara del elemento principal del proyecto. ¿Qué es Kinect?.

Kinect, originalmente conocido como *Project Natal*, es un aparato que detecta el movimiento de las personas en tiempo real siendo capaz de reconocer gestos y comandos por voz. Concebido por Microsoft para la consola de sobremesa Xbox 360 con el objetivo de llamar la atención de aquellos jugadores que buscan un juego distinto a los que se suelen ver.

Salió a la venta el 10 de Noviembre del 2010 (en Europa) [MKi-09] y no fue hasta el 16 de Junio del 2011 que Microsoft hizo pública una versión no comercial de la SDK (*Software Development Kit*) para Windows 7, permitiendo a los desarrolladores crear aplicaciones en C++/CLI, C# o Visual Basic.

Kinect dispone de una cámara de color y de otras dos que detectan la profundidad tal y como muestra la figura 1-1. De esta forma es posible calcular dónde hay un cuerpo así como las diversas articulaciones de éste para poder recrear el esqueleto. En la figura 1-2 se muestra un ejemplo simple de como, usando la información que Kinect recolecta, se puede obtener un esqueleto. La cámara de profundidad pinta los píxeles en una escala de grises acorde a la profundidad que se encuentre. Los píxeles de color que vemos en la figura 1-2 b) son el resultado de los cálculos para detectar un usuario a partir de la información de la cámara de profundidad.

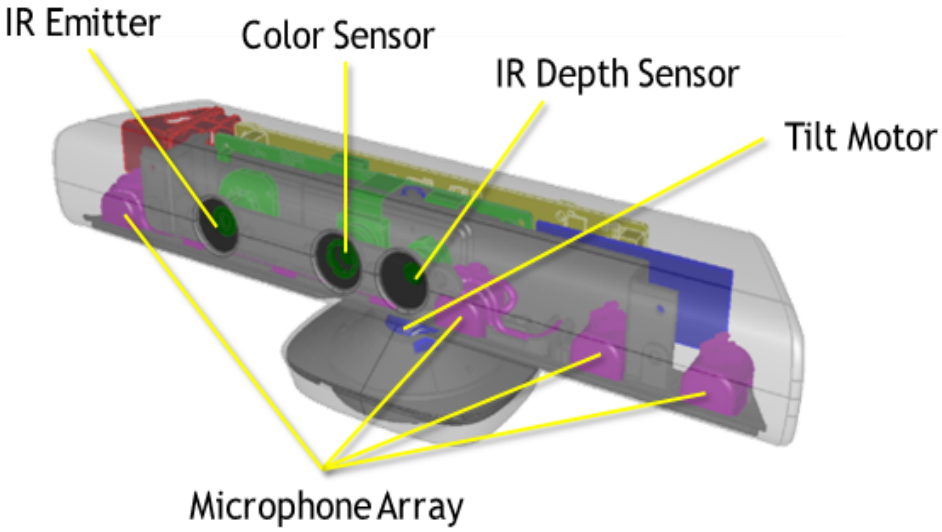


Figura 1-1. Componentes de Kinect.

Esta tecnología abre muchas puertas para desarrollar diferentes ideas en múltiples ámbitos, tal y como se comenta en el apartado siguiente. Para este proyecto se ha escogido el ámbito de animación en 3D. Personalmente, dado que mi mundo de ocio está muy relacionado con el de los videojuegos, Kinect me llamó la atención desde que salió al mercado. Me parece un dispositivo muy interesante y potente, por lo que pensé que sería una buena idea probar de hacer algo útil con ello. Kinect fue creada principalmente para desenvolverse en el mundo de los videojuegos, un mundo en el que estoy interesado entrar. La idea de usar Kinect en el proyecto me entusiasmó por el

hecho de que así podría estar algo más cerca del mundo de los videojuegos, aún que sea comenzando por algo menos complejo que programar un videojuego en sí.

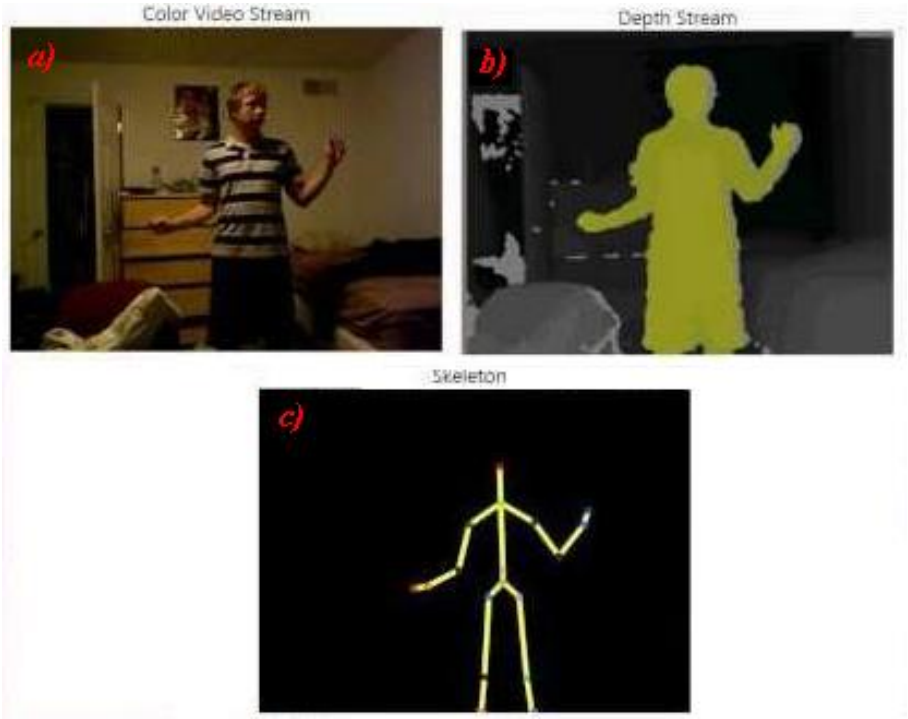


Figura 1-2. a) Información captada por la cámara de color. b) Información captada por las cámaras de profundidad. c) Información calculada según la profundidad para representar el esqueleto del usuario.

1.1 Trabajos ya realizados

En esta sección se presenta información sobre referencias a proyectos ya realizados que interactuan con Kinect. Se conseguirá así dar una idea de lo que la cámara de Microsoft puede hacer.

Se divide la sección en dos apartados, uno para mostrar trabajos relacionados con *motion capture* y otro para mostrar trabajos situados en otros campos como la medicina y la educación.

1.1.1 Trabajos relacionados con Motion Capture

Ya se ha intentado usar el dispositivo de Kinect como un método de *motion capture* barato y sencillo. El *motion capture*, o *mocap*, es el proceso de grabar el movimiento de objetos o de personas. En el mundo de los videojuegos y de la producción de películas se usa para animar de forma más realista.

Se presentan algunos proyectos que circulan por Internet. El primer proyecto guarda información sobre la captura en tiempo real para luego utilizar los datos mas adelante, mientras que el segundo permite al usuario controlar directamente al modelo en tiempo real, el cual imita los movimientos del usuario. El tercero usa *motion capture* para poder capturar el movimiento facial y el último es una herramienta para trabajar con Kinect y *motion capture.*

a) Motion Capture en tiempo real

Brekel Kinect es un proyecto creado por Jasper Brekelmans [Bre-12] que guarda la información sobre una animación en tiempo real en un formato de texto para poder usarla más adelante para animar un modelo 3D. El usuario aparece en la imagen en una nube azul sacada de la grabación de Kinect. Se usa el NITE de *PrimeSense* junto a OpenNI, ambos softwares usados conjuntamente, para reconocer al usuario y así hacer la construcción del esqueleto.

Este proyecto recolecta la información de los cálculos que hace NITE sobre el esqueleto y la puede traducir en formato BVH para que otros programas de diseño, como el 3dsMax, puedan incorporarlo en sus bípedos. El formato BVH es un método de guardar la información que se recolecta del *motion capture.*

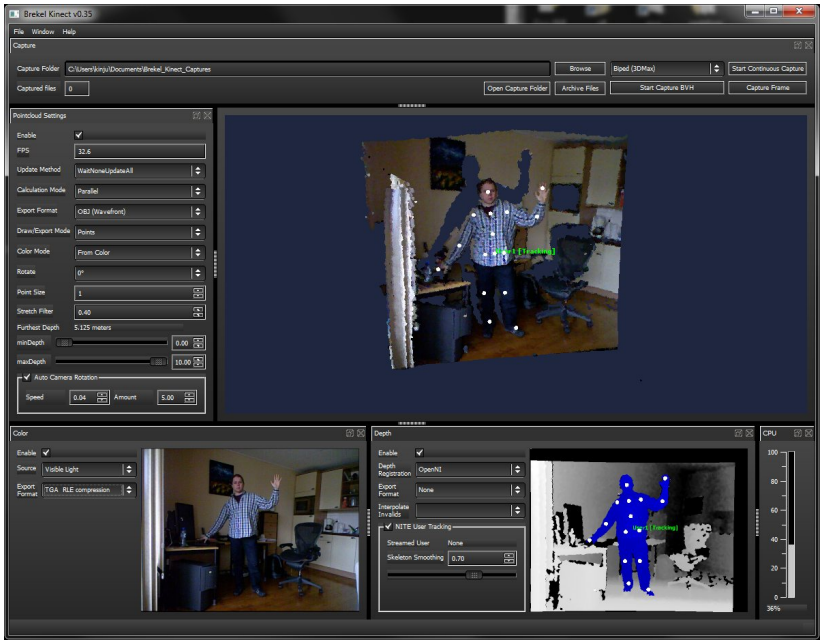


Figura 1-3. Interficie de Brekel Kinect.

Under The HUD [HUD-11] es una serie web animada que hace uso de Brekel Kinect para facilitar la animación. Los creadores de la serie envían la información generada por Brekel Kinect directamente al *Motion Builder* [ADk-82], un software profesional de animación 3D, que es usado para el *motion capture,* para reflejarlo en la animación del personaje. Una vez ahí, retocan los fallos que hayan podido haber al hacer *motion capture* y lo importan al software Maya [ADk-82], programa para desarrollar gráficos 3D, efectos especiales y animación.

b) Control de Personajes

El proyecto, referenciado solamente como *Kinect Hacking* [Jbo-11], que tiene como objetivo capturar el movimiento con Kinect de una persona y traspasarlo directamente al personaje que controlamos en un videojuego. Usan la librería OpenNI [ONI-10], que proporciona APIs para interactuar con hardware así como reconocer objetos y gestos, facilitando las difíciles tareas como hacer tracking de los esqueletos en un espacio 3D. También usa un motor físico. Usaron el *Garry's Mod* [GMo-04], o *Gmod*, para el motor *Source,* de *Valve,* dada su facilidad para escribirlo en Lua. *Gmod* es una modificación del motor *Source* [SEn-96], que acabó convirtiéndose en un juego independiente que permite al usuario manipular objetos y experimentar con la física dentro del

mundo del juego.

El proyecto se divide en dos partes: una parte se encarga de interpretar la información que brinda Kinect y la otra parte es un script Lua que controla el juego. La primera parte lee la información de Kinect por medio de un USB, usando OpenNI hace un tracking del esqueleto del usuario y entonces envía las coordenadas que corresponden a las articulaciones del esqueleto a la segunda parte. La segunda parte recibe esas coordenadas que usará para mover el modelo 3D así como los objetos que hayan en el entorno.

c) Captura 3D de movimiento facial

Varios proyectos han utilizado Kinect para de capturar el movimiento facial sin la necesidad de usar marcadores (etiquetas de forma circular que se pegan en la cara para que el algoritmo sepa dónde está una zona determinada de la cara en todo momento).

En el video de la referencia [BBC-11] podemos ver un ejemplo de un proyecto realizado por M.Breidt, H. H. Bülthoff y C. Curio, el cual busca modelar una cara a partir de los gestos faciales de los usuarios. La imagen de la esquina superior derecha representa la grabación a color del usuario en un momento dado. En las otras imágenes podemos ver que las tonalidades de verde representan la información que la cámara de profundidad de Kinect recolecta sin modificar. En las tonalidades de lila vemos el resultado de como quedaría la cara del modelo 3D.

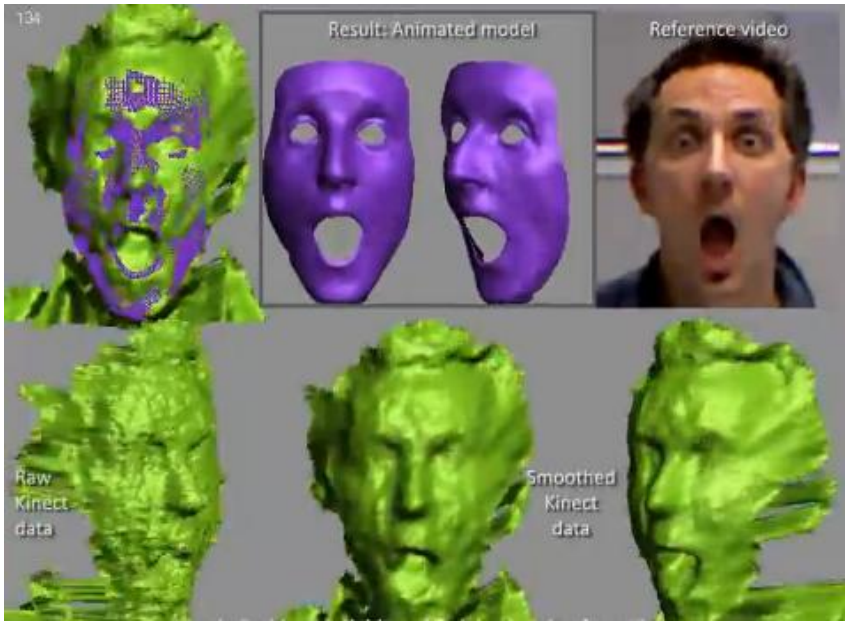


Figura 1-4. Interfaz del proyecto de captura de movimiento facial. En la parte superior central, las caras de lila muestran el resultado final.

d) Más Motion Capture - Ipisoft

Finalizaremos con el proyecto de *Ipisoft*, que está desarrollando una nueva tecnología barata de *motion capture*: *iPi Desktop Motion Capture* [IPi-09]. Es compatible con Kinect y otras cámaras, pero una característica interesante que lo diferencia de los otros proyectos mencionados es que permite el uso de dos Kinects para capturar el movimiento. De esta forma es capaz de continuar capturando correctamente los datos de un usuario aunque este se ponga de espaldas. Puede guardar la información recogida en los formatos BVH, FBX, DAE y SMD.

1.1.2 Otros usos del Kinect

Si bien se conoce a Kinect por su fama en el sector de videojuegos, otros proyectos han surgido aprovechando las capacidades del dispositivo. La siguiente información se ha recogido de un artículo publicado en Meristation [Mer-11].

El 24 de Noviembre de 2011 Microsoft organizó un evento para discutir qué aplicaciones podrían resultar útiles fuera del sector de videojuegos. Asistieron profesionales del ámbito de videojuegos, medicina, enseñanza y de la prensa. Se presentan algunos de los proyectos, separados por categorías según su ámbito.

a) Médico

- Uno de los proyectos fue propuesto por la investigadora de la Universidad Rey Juan Carlos, María Ortiz Gutiérrez, el cual consiste en ayudar a los pacientes que padecen de esclerosis múltiple a través del control postural. Con un programa de 10 semanas, se proponía al usuario mover su cuerpo con juegos comerciales que usan Kinect, en sesiones de 25 a 45 minutos, para así mejorar su calidad de vida y hacer la rehabilitación desde casa. El resultado de dicho estudio fue muy positivo, mostrando mejoras considerables en las habilidades motrices de los participantes, además de habiéndolo logrado de una forma "divertida".
- Otro proyecto, por parte de la empresa Tedesys, consistía en poder controlar un visor DICOM junto a la disponibilidad de una interfaz que el usuario puede controlar mediante gestos, facilitando así el trabajo a los profesionales del sector.
- ASISPA (Asistencia a Personas Mayores) propuso un proyecto con Kinect para mejorar las capacidades de los enfermos de Alzheimer mediante juegos con el objetivo de rehabilitar. Pablo Gallardo, presidente de FIVAN (Fundación Instituto Valenciano de Neurorehabilitación), presentó un proyecto enfocado a la neurorehabilitación motora y cognitiva en personas que padezcan de traumatismos graves, tumores, ictus o lesiones de médula monitorizando online los resultados mediante Kinect.

b) Educativo

- Fuera del ámbito médico se presentaron otros proyectos, cómo el de la institución educativa SEK, que consistía en aplicar Kinect en las aulas, mostrando vídeos sobre cómo los alumnos podrían interactuar con juegos de Kinect, fomentando la actividad física y de grupo.

c) Robótica

- Asier Arranz, fundador de Nebutek Soluciones, presentó un proyecto orientado a la robótica que mostraba como poder controlar un robot en cualquier parte del mundo desde casa a través de Kinect.

d) Comercial

- Por último, Tedesys presentó otro proyecto, TedShop, un sistema que ofrecía un interfaz con Kinect para poder comprar online, permitiendo al usuario no solo ver todo el catálogo de ropa sino que además, se estudiaba la posibilidad de poder ver como una prenda de ropa te

quedaba antes de comprarla.

1.2 Objetivos

Con la tecnología que otorga Kinect, el proyecto intenta convertirse en una herramienta para que los diseñadores de videojuegos o animadores 3D tengan más facilidades de animar sus modelos 3D sin necesitar de trajes con sensores y/o dispositivos.

El objetivo de este PFC consiste en desarrollar una aplicación que, mediante Kinect, grabe los movimientos del usuario y puedan usarse en el software 3dsMAX. Después, exportar los datos con las librerías de Cal3D para poder usar las animaciones en el motor gráfico usado en la asignatura de videojuegos de la UAB.

El proyecto busca las siguientes funcionalidades:

- Desarrollar una aplicación que, utilizando la cámara Kinect, capture los movimientos de una persona referentes al movimiento de huesos (movimientos de piernas y brazos, así como del torso).
- Pasar la información capturada a un formato compatible con el programa 3ds MAX y traspasarlo a un modelo 3D.
- Usar las librerías de animación de Cal3D para poder usar las animaciones en el motor de la asignatura de libre elección de videojuegos.

1.3 Software utilizado

A continuación se lista el software utilizado para poder cumplir con los objetivos del proyecto.

- Open Natural Interaction.
- NITE.
- Configurable Math Library.
- OpenGL.
- 3D Studio Max.
- Character Animation Library 3D
- DirectX
- BioVision Hierarchical data

Se definirán de forma breve para poder concebir una idea sobre como funcionan antes de indagar en el desarrollo principal del programa.

a) Open Natural Interaction (OpenNI)

OpenNI [ONI-10] es un framework y una API de código abierto compatible con Windows y Linux, multi-lenguaje, que define APIs para usar aplicaciones que hacen uso del concepto Interacción Natural (*Natural Interaction*). El término Interacción Natural (NI) hace referencia al concepto en el que la interacción entre humano y aparato se basa en los sentidos humanos, principalmente centrado en visión y oído. Algunos ejemplos sobre NI pueden ser el habla (dar órdenes por voz a un aparato), gestos con las manos (un gesto específico puede dar una orden al aparato) o captura de movimiento

corporal (como es en nuestro caso, utilizado para animaciones).

Las APIs están compuestas por un conjunto de interfaces para escribir aplicaciones NI. OpenNI busca ser una forma estándar para permitir la comunicación entre sensores de vídeo y audio con middleware de vídeo y audio. Son componentes software que analizan los datos captados por los sensores, los analizan e intentan comprenderlos.

Por ejemplo, los sensores captan lo que se 've' en una habitación con una persona y el middleware se encarga de devolver dónde está situada la mano derecha de dicha persona mediante la imagen capturada.

OpenNI rompe la dependencia entre sensor y middleware, por lo que las aplicaciones, una vez escritas, pueden ser portadas directamente para funcionar sobre otros módulos de middleware. Además, OpenNI permite diseñar algoritmos sobre formatos de datos sin procesar, sin importar que sensor los ha producido.

En la Figura 1-5 se visualiza las tres capas que representan los componentes de OpenNI:

- **Capa superior:** Representa el software que implementa la aplicación de Interacción Natural sobre OpenNI.
- **Capa del medio:** Representa a OpenNI, otorgando una interfaz de comunicación que interactua entre los sensores y los componentes middleware, que analizan los datos del sensor.
- **Capa inferior:** Representa los aparatos hardware que capturan la información visual y auditiva de la escena.

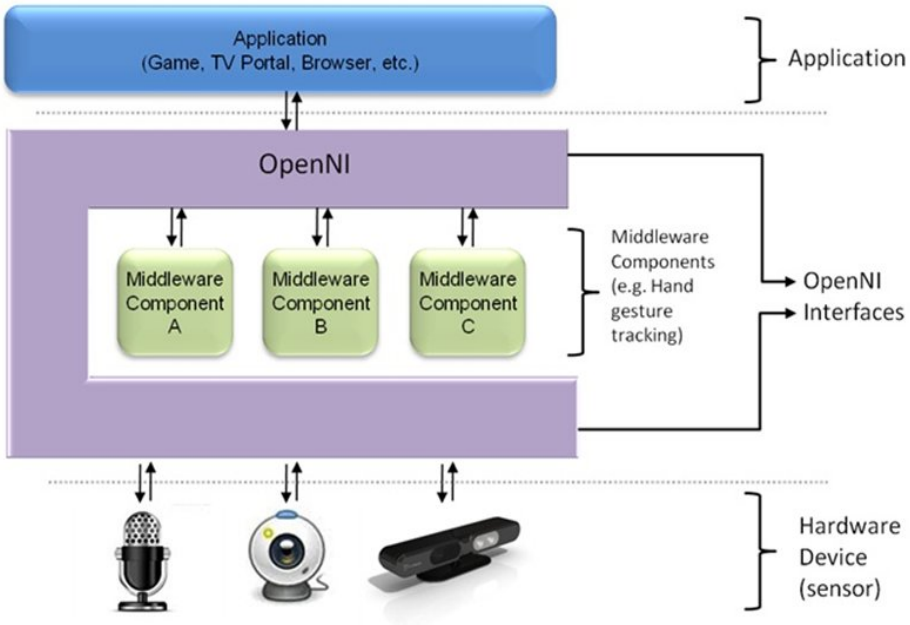


Figura 1-5. Esquema de Capas sobre los componentes de OpenNI.

Lo interesante de OpenNI para el proyecto, en definitiva, es que permite capturar escenas reales en 3D (en nuestro caso, capturar los movimientos de una persona) de forma compatible con Kinect.

b) NITE

NITE [PrS-05], de *PrimeSense*, es un middleware de Interacción Natural que trata la información recibida sobre la escena para comprenderla, traducirla o responder según los movimientos de una persona. Permite dar órdenes y tener un control sobre programas usando las manos, el cuerpo o la voz. La idea de NITE es permitir que el propio cuerpo sea capaz por sí solo de controlar aparatos y programas sin necesidad de ningún controlador remoto.

NITE será el middleware que se usará para integrarse en OpenNI y así poder calcular el esqueleto de una persona en movimiento.

Cabe aclarar que existe una SDK creada por Microsoft para escribir aplicaciones que usen Kinect y que también permiten capturar el movimiento de una persona y calcular la posición de sus articulaciones. Cuando se comenzó el proyecto, se usó Kinect SDK [Kin-11] para experimentar con Kinect, pero la versión de la SDK de aquel entonces no era capaz de calcular las rotaciones de cada articulación, a diferencia de NITE. Se decidió usar OpenNI con NITE para poder hacerse una idea de cómo funcionaban las matrices de rotación, a pesar de que, finalmente, dichas matrices son calculadas manualmente.

Actualmente, Kinect SDK, en su última versión, es capaz de calcular las matrices de rotaciones, así como los cuaterniones de cada articulación, incluyendo manos y pies. Es incluso capaz de capturar expresiones faciales.

c) Configurable Math Library (CML)

CML [NyA-03] es una librería matemática, gratis y de código abierto para C++, diseñada para ser usada en juegos, gráficos, geometría computacional y aplicaciones relacionadas. Sus librerías incluyen clases para vectores, matrices y cuaterniones junto a funciones para manipular dichos tipos de datos.

El propósito de CML es ofrecer una alternativa adicional para desarrolladores que necesiten de librerías matemáticas. En particular, puede ser usada como una extensión de una librería matemática existente, como la de DirectX. En el proyecto se usará para realizar diversos cálculos.

d) OpenGL

OpenGL [OGL-92] ha sido el primer entorno para desarrollar aplicaciones gráficas portátiles e interactivas en 2D y 3D. Hoy en día es uno de los más usados. OpenGL aporta un amplio conjunto para renderizar, mapear texturas, crear efectos especiales y otras funciones visuales poderosas.

Se usará para poder visualizar, en la aplicación que usa Kinect para calcular el movimiento de una persona, la información de profundidad que Kinect proporciona, así como visualizar el esqueleto del usuario rastreado.

e) 3D Studio Max (3dsMAX)

3ds Max [ADk-82], de Autodesk, es un software pensado para diseñadores con tal de que no deban usar tanto esfuerzo en conceptos tecnológicos, facilitándoles así su trabajo. El software 3DsMax ofrece un comprensivo sistema para modelado 3D, animación, renderización para desarrolladores

de videojuegos, artistas de efectos especiales y artistas de gráficos de movimiento entre otros profesionales del ámbito creativo multimedia.

Lo usaremos para poder ver los resultados obtenidos por la aplicación del proyecto, así como para integrar la información capturada por Kinect a nuestro modelo 3D.

3ds Max no es gratuito, sin embargo, ofrecen una licencia gratuita para estudiantes.

f) Character Animation Library 3D (Cal 3D)

Cal3D [Cal-06] es una librería de animación de personajes 3D basada en esqueletos escrita en C++. Originalmente se usaba en un cliente 3D para *WorldForge*, un videojuego online multijugador masivo de rol, pero evolucionó en un producto independiente que puede ser usado en diferentes tipos de proyectos. Soporta la combinación de animaciones y acciones mediante una interfaz "mezcladora".

Lo usaremos para poder ver las animaciones resultantes en el motor gráfico usado en la asignatura de videojuegos de la UAB. Cal3D es una librería gratuita.

g) DirectX

Originalmente se creó la SDK de DirectX como una plataforma de alto rendimiento para el desarrollo de videojuegos sobre Windows. DirectX fue evolucionando y se volvió relevante para un mayor rango de aplicaciones.

DirectX [DiX-98] es una colección de APIs que tienen el objetivo de realizar el complicado trabajo que suponen las tareas relacionadas con multimedia, en especial para el desarrollo de videojuegos y vídeo. Las APIs incluyen procesos para:

- Tratar y dibujar gráficos en dos y tres dimensiones.
- Procesar datos de controladores (como teclados y ratones).
- Comunicaciones en red.
- Reproducir y grabar sonidos así como pistas de audio.
- Hacer cálculos matemáticos con su librería de matemáticas.

Usaremos DirectX como motor gráfico para poder visualizar el resultado final de las animaciones. Si bien parece innecesario usar OpenGL cuando tenemos ya DirectX, la razón es porqué la aplicación que calcula los datos relacionados con *motion capture* está separada de la que sirve para visualizar las animaciones. Además, la primera se ha generado a partir de un ejemplo previo que ya usaba OpenGL, por lo qué se decidió seguir usando la librería para mayor comodidad.

h) El formato BVH

La idea del proyecto es 'grabar' los movimientos de una persona con Kinect para luego pasar esos datos al 3dsMAX y aplicarlos en un modelo 3D. Para ello, se precisa de un sistema para guardar dicha información y que el 3dsMAX lo entienda. BVH es el formato que se necesita.

El formato *BioVision Hierarchical data* (BVH) [MyM-05] [ChS-00] es un formato que representa la información que define cómo un esqueleto se mueve y cómo está formado de forma

sencilla y simple. Fue creado con la idea de ser usado mediante *motion capture* para recolectar información directamente desde actores que simulen la animación deseada en vez de dejar todo el trabajo a los diseñadores. De esta forma, BVH permite agilizar y facilitar el trabajo de los animadores.

El formato BVH se basa en una estructura jerárquica de huesos para definir las rotaciones. Es un formato que proviene del BVA (también de BioVision), otro formato para guardar información de *motion capture*, el cual carecía del concepto 'hueso' y resultaba difícil de aplicar en la práctica.

Si bien existen otros formatos de *motion capture*, 3dsMAX solo acepta los formatos BIP, CSM y BVH. De estos tres, se llegó a la conclusión de que el más accesible y fácil de utilizar era el BVH ya que es mucho mas simple que el resto. Además de que este formato es usado en otros proyectos similares al que se presenta, como Brekel Kinect, lo que nos permite compararlo con nuestro proyecto y facilitarnos el camino hacia algo en un principio desconocido como el *motion capture*.

Finalizando con el primer capítulo, dedicado a hacer una introducción al proyecto, proseguiremos con los siguientes capítulos.

El capítulo número dos explica la parte de desarrollo realizada. Contiene información sobre los cálculos y métodos realizados para obtener los datos necesarios para poder indicarle al 3dsMAX como debe moverse el modelo. También explica cómo exportar los datos con Cal3D. El proyecto ha producido dos aplicaciones, una para capturar el movimiento y otra para visualizar los resultados con el motor gráfico.

El capítulo número tres explica los resultados obtenidos. Se han hecho una serie de experimentos para demostrar el funcionamiento del proyecto. Se explica cómo se han hecho los experimentos y su grado de éxito. Además, se incluyen las incidencias surgidas en el proyecto.

El cuarto capítulo es dedicado para las conclusiones y mejoras. Se resumen los objetivos alcanzados y se comentan las incidencias surgidas en el proyecto que no se han podido solucionar. También se incluyen algunas ideas para mejorar el proyecto.

El quinto capítulo incluye la bibliografía y las referencias usadas ordenadas alfabéticamente.

Los siguientes tres capítulos contienen los anexos. El primer anexo explica el formato BVH al detalle. El segundo anexo muestra una muestra del formato BVH. El tercer anexo contiene el código fuente de un método para convertir matrices de rotaciones en ángulos de Euler.

2. DESARROLLO

Este capítulo está dedicado al trabajo realizado para el proyecto. El peso principal del apartado irá dedicado a los cálculos y métodos desarrollados para lograr los objetivos principales. También se comentará la programación realizada para poder captar y generar los cálculos, así como mostrar los resultados por pantalla.

El diagrama de las figura 2-1 muestra el seguimiento del desarrollo del proyecto. En la figura 2-1 se ven los módulos globales del proyecto. La cámara Kinect recolecta la información de movimiento de un entorno en el que hay un usuario. NITE, usando la información que Kinect recolecta, detecta al usuario y calcula la posición de sus articulaciones, las cuales son utilizadas en la aplicación principal. La aplicación principal se ha diseñado para este proyecto, es una aplicación que utiliza la información que NITE le envía para generar archivos en formato BVH. Los archivos BVH generados representan el movimiento del usuario captado por la cámara. Estos archivos se los pasamos al software 3dsMAX para que le añada el movimiento que los archivos BVH guardan a un modelo 3D. Ahora tenemos un modelo animado con los movimientos captados por la cámara Kinect. Exportando la animación con las librerías de Cal3D obtenemos los archivos necesarios para poder usar un modelo animado en el motor de DirectX (el que se usa en la asignatura de videojuegos de la UAB). Usando el motor podremos ver las animaciones por pantalla en nuestro modelo.

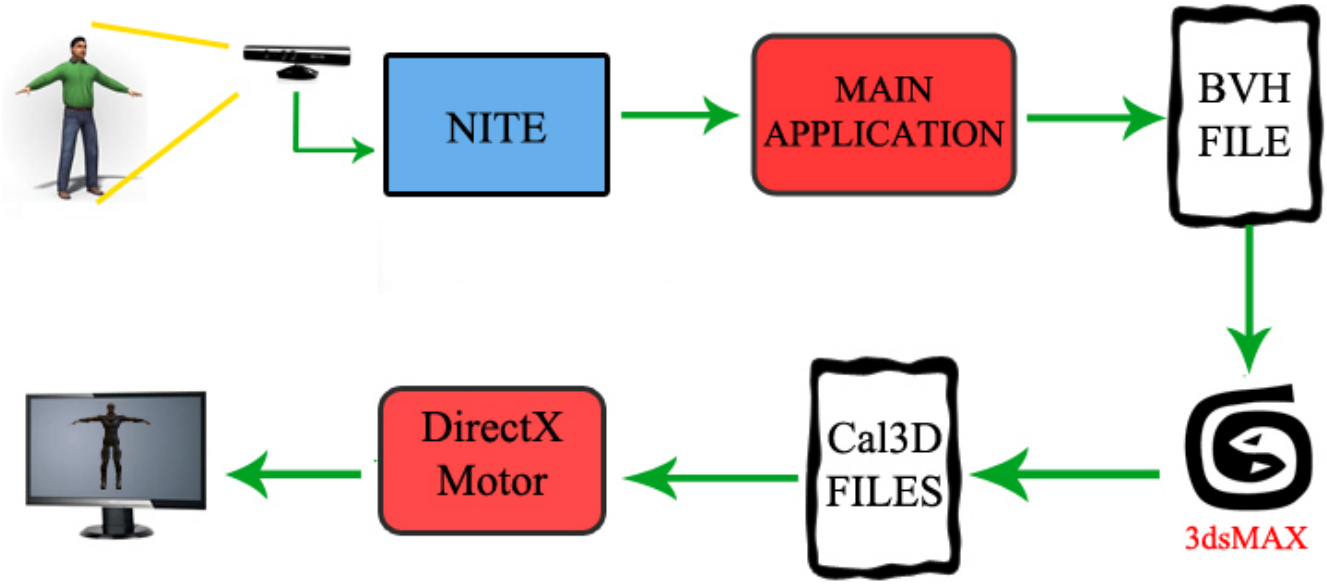


Figura 2-1. Diagrama del desarrollo en vista general.

El módulo de la aplicación principal (*Main Application*) es el de mayor peso para el proyecto, ya que se encarga de tratar con la información que NITE recolecta de Kinect y hace los cálculos necesarios para cumplir con los objetivos del proyecto.

A continuación, se detallarán más detenidamente los módulos del esquema de la figura 2-1 para conocer a mayor escala los pasos tomados para la realización del proyecto.

2.1 Aplicación Principal

El módulo de la aplicación principal, la cual es nombrada como *BVHSkeletonTracker*, se encarga de generar archivos BVH usando la información que NITE proporciona. BVH es un formato de texto utilizado para representar los movimientos de un esqueleto.

En la figura 2-2 se muestra lo que hay dentro del módulo de la aplicación principal. Como entrada, NITE nos otorga ciertos datos calculados a partir de la información que Kinect detecta. La información que NITE calcula trata todo sobre el usuario. Si Kinect está grabando a un usuario, NITE lo detecta e indica qué píxeles, de lo que Kinect percibe, representan al usuario. Además, NITE, una vez teniendo la nube de píxeles que representa al usuario, es capaz de calcular las posiciones de cada articulación del usuario en un espacio tridimensional (x, y, z) además de las orientaciones de cada articulación.

La aplicación principal recoge esta información de NITE y la usa para cumplir dos funciones: La primera, mostrar por pantalla la percepción de profundidad de Kinect, resaltando los píxeles que representan al usuario de otro color así como sus articulaciones y la segunda, calcular rotaciones y distancias del esqueleto para generar los archivos BVH, que serán lo que se obtiene a la salida del módulo. Un esqueleto es un conjunto de articulaciones con una jerarquía de estilo árbol. NITE nos da su propia jerarquía, pero hay que definir una para el formato BVH. El módulo Modelo Jerárquico, representa la jerarquía diseñada para el formato BVH.

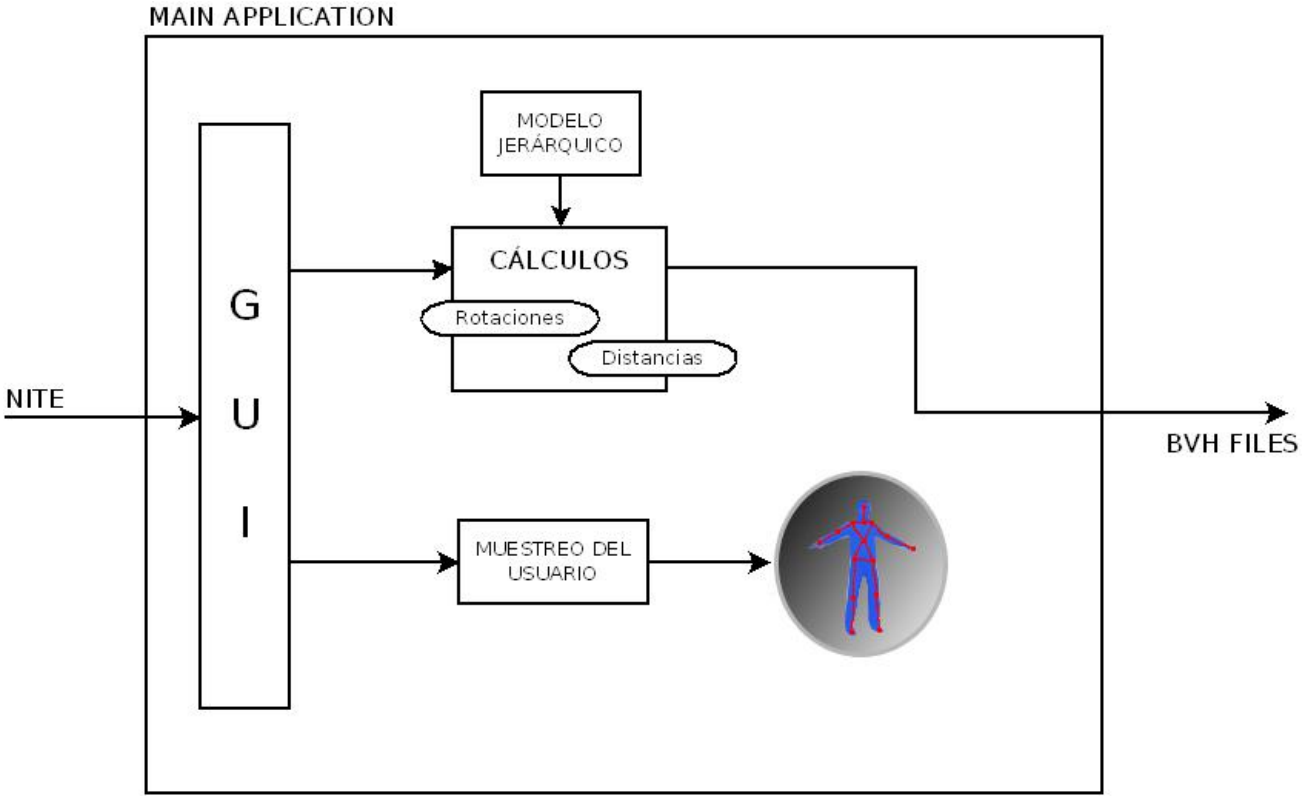


Figura 2-2. Diagrama del módulo de la aplicación principal.

A continuación indagaremos más en el módulo de la aplicación principal para explicar los módulos que contiene. Explicaremos qué es y cómo hemos hecho el Modelo Jerárquico,

definiremos que cálculos se han realizado y explicaremos que funcionalidades otorga la GUI (Interfaz Gráfica del Usuario) al usuario.

2.1.1 Modelo Jerárquico del esqueleto para el formato BVH

Tal y como muestra el diagrama de la figura 2-2, la aplicación principal requiere un modelo jerárquico con el que nos basaremos para definir el esqueleto para el formato BVH. Este modelo nos indica cuantas articulaciones tiene el esqueleto resultante, así como quién es hijo de quién. El modelo escogido para el proyecto está dibujado en la figura 2-3. Se escogió dicho modelo por necesidad de imitar la estructura esqueleto que NITE nos calcula y que nos diera libertad para representar los movimientos en BVH, además de tener similitudes con otros proyectos como el de *Brekel Kinect*.

La estructura del formato BVH está explicado en el Anexo I. El diagrama muestra todas las articulaciones que definen nuestro esqueleto. Una raíz puede tener hijos, definiendo así una jerarquía. El hijo de una articulación se representa con una flecha que apunta a la articulación hijo. Este sistema jerárquico indica que articulaciones influyen en otras, por ejemplo, si la articulación de un hombro (por ejemplo, *RightShoulder*) se vé modificada, también lo serán sus hijos, y los hijos de sus hijos (en este caso, se vería afectado el hijo de *RightShoulder*, que es *RightElbow*, además del hijo de éste, *RightWrist*). Toda articulación tiene un padre excepto la raíz (*Root*), que equivaldría a la articulación posicionada en el centro de las caderas (*Hips*).

Aunque el modelo representado en la figura 2-3 sea solo conceptual, en el código de la aplicación principal, las articulaciones del modelo jerárquico sirven para guardar información sobre la distancia que hay entre ellas y sus hijos además de la orientación que tienen en cada momento.

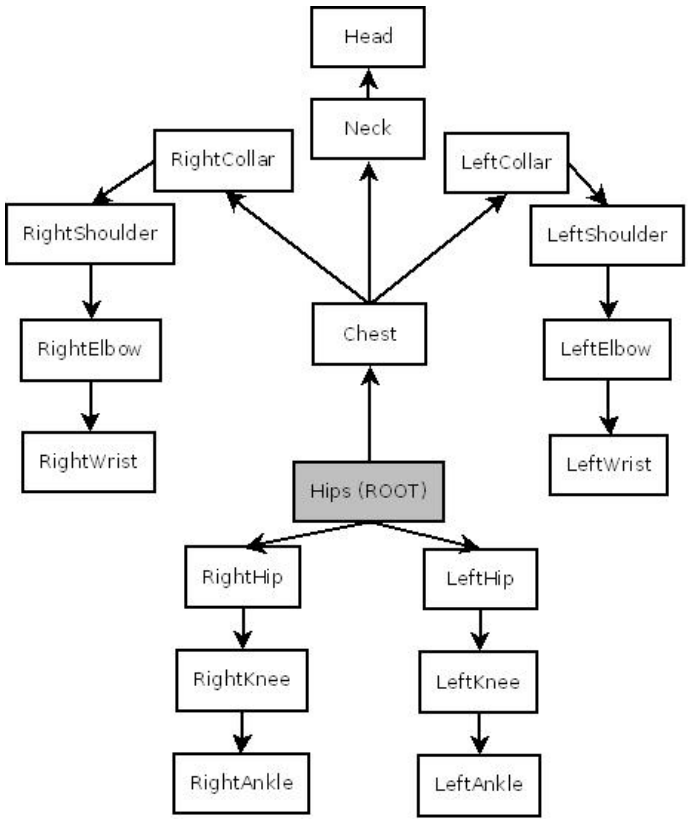


Figura 2-3. Diagrama de la jerarquía del esqueleto usada en el proyecto.

2.1.2 Cálculos

Este módulo tiene como objetivo realizar los cálculos necesarios para crear archivos en formato BVH utilizando la información que NITE nos proporciona sobre el esqueleto del usuario.

El formato BVH, tal y como se explica en el Anexo I, se compone por dos partes. La primera, definida como *Hierarchy*, sirve para construir el esqueleto. Este esqueleto se creará a partir del modelo jerárquico escogido. La segunda, definida como *Motion*, sirve para definir las rotaciones y traslaciones de todas las articulaciones en cada frame.

Dividiremos este apartado en dos partes. Una parte explica los cálculos dedicados a las distancias, necesarias para construir el esqueleto en la parte *Hierarchy* del formato BVH. La segunda parte la dedicaremos a definir los cálculos para las rotaciones de cada articulación en cada momento, estos cálculos servirán para rellenar la parte *Motion* del formato BVH.

Para todos los cálculos hay que tener en cuenta que el origen en el espacio tridimensional es la cámara Kinect. Si nos situamos frente a la cámara, la X crece si nos movemos a la derecha, la Y crece hacia arriba y la Z crece si nos alejamos de la cámara. La información que usaremos de NITE será solamente la posición en el espacio tridimensional de cada articulación.

a) Distancias

Las distancias son presentes en la parte jerárquica *(Hierarchy)* del formato y se usarán para indicar la longitud de los huesos y además, en el caso de la raíz, su posición en el espacio tridimensional.

NITE nos otorga las posiciones de cada articulación en un vector de tres dimensiones, indicando sus coordenadas respecto al espacio tridimensional en un frame en concreto. Así pues, teniendo:

$$Posición\_Articulación_i = (x_i , y_i , z_i)$$

Si calculamos la distancia entre dos articulaciones tendremos un hueso:

$$Hueso_k = Posición\_Articulación_j - Posición\_Articulación_i = (x_j - x_i , y_j - y_i , z_j - z_i)$$

Un hueso es un vector de tres dimensiones y sus valores son los que pondremos en el OFFSET de la correspondiente articulación cuando construyamos la parte jerárquica. El OFFSET es una palabra clave del formato BVH que se utiliza en cada articulación e indica la distancia entre una articulación y su articulación padre. Por norma general, los huesos se calculan de forma que a la articulación actual se le reste la padre. Por ejemplo, si estamos rellenando la información del OFFSET de la rodilla izquierda (*LeftKnee*), su OFFSET es la distancia entre la parte izquierda de las caderas y la rodilla. La figura 2-4 muestra este mismo ejemplo.

$$Hueso_{Fémur\ Izquierdo} = Posición\_Articulación_{LeftKnee} - Posición\_Articulación_{LeftHip}$$

Recordemos que el OFFSET de una articulación indica la distancia de ésta respecto a su nodo padre, por lo que siempre hay que mirar atrás. Sin embargo, hay que tener en cuenta que si calculamos huesos horizontales es al revés, para que el esqueleto resultante sea comprensible y no quede deformado, se le resta a la posición de la articulación padre la posición de la articulación hijo, justo al revés de lo explicado anteriormente. Es el caso de los hombros y las caderas. Un ejemplo con el hombro izquierdo:

$$Hueso_{Clavícula\ Izquierda} = Posición\_Articulación_{Neck} - Posición\_Articulación_{LeftShoulder}$$

Si no lo hacemos así, las piernas y los brazos quedarán invertidas, como si hubieran deformado el esqueleto doblando las extremidades hasta que una quedara en la posición de la otra.

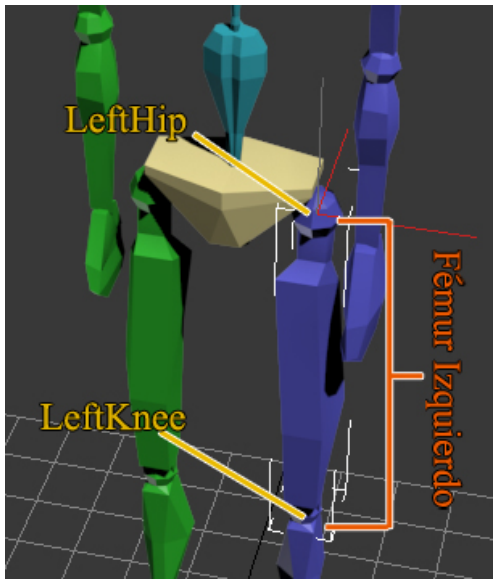


Figura 2-4. Representación del hueso Fémur Izquierdo

Las articulaciones *Left* y *Right Collar* son articulaciones fijas, que no se van a mover y que necesitamos para que quede un buen esqueleto. Éstas articulaciones corresponden a la clavícula y se supondrán que están situadas justo dónde está la articulación del cuello dado que NITE no tiene ninguna articulación para las clavículas.

Para calibrar el esqueleto es importante saber que hay que calcular las distancias con un sujeto que esté en postura inicial. Una postura inicial es un esqueleto que se mantiene firme y erguido, tal y como muestra la figura 2-7. Tampoco es necesario calcular las distancias sobre cada eje de cada hueso. Los datos pueden tener ligeros errores que pueden hacer que el esqueleto quede poco agradable a la vista, dejándolo torcido o con alguna leve deformidad, así que sólo calcularemos las distancias que necesitemos. Por ejemplo, sabemos que las piernas y los brazos sólo crecen en el eje Y, por lo tanto, únicamente debemos calcular la distancia sobre ese eje. De hecho, para simplificar y dejar más limpio el esqueleto resultante, sólo se calcula el eje Y para los huesos verticales, y sólo el eje X para los huesos horizontales.

El OFFSET del nodo raíz es diferente. Dado que no tiene padre, el OFFSET indica la posición origen de la raíz. Para este proyecto se decidió que, sin importar la posición en el espacio tridimensional del usuario en el momento de calcular la parte jerárquica del formato BVH, el OFFSET sería el punto origen (0,0,0) siempre. La articulación raíz es la única que guarda información sobre las traslaciones que realiza. Esto es porque al ser la raíz, traspasará la traslación al resto de articulaciones. Las traslaciones se guardan en la parte de movimiento (*Motion*) del formato BVH y se calculan restando a la posición actual de las caderas la posición inicial real. Así, el esqueleto siempre se moverá desde el punto de partida (0,0,0).

NITE presenta algunas limitaciones que afectan a este apartado. NITE no calcula ciertas articulaciones, en concreto, no calcula las articulaciones *Hips*, *Left* y *Right Ankle* y *Left* y *Right Wrist*. Esto nos deja sin poder rastrear el centro de la cintura (la raíz), así como las muñecas y los tobillos. Para la raíz podemos calcular nosotros mismos la posición calculando el punto medio de las caderas:

$$Posición\_Articulación_{Hips} = (Posición\_Articulación_{LeftHip} + Posición\_Articulación_{RightHip}) / 2$$

En la figura 2-5 podemos ver la representación de ese punto medio para encontrar la articulación raíz.

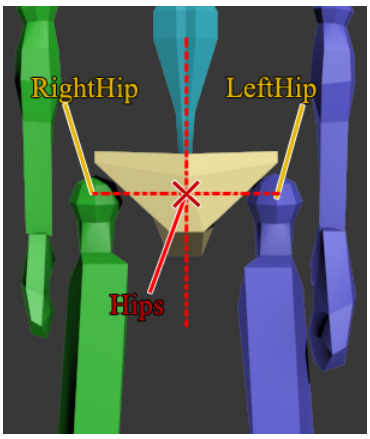


Figura 2-5. Representación del punto medio de las caderas, correspondiente a la articulación raíz.

No obstante, para muñecas y tobillos no nos queda otra que usar las articulaciones *Left* y *Right Hand* y *Left* y *Right Foot*. Pero como NITE no nos proporciona información sobre dichas articulaciones nos quedamos sin ninguna referencia para poder calcular los END SITE de las terminaciones, por lo que se tuvo que aplicar unas medidas fijas manualmente. Los END SITE son otra palabra clave definida en el Anexo I que indica hasta dónde se alarga el hueso de una terminación, una terminación es en sí una articulación sin nodos hijos. Los pies serán de 0,2 metros (para el eje Z) y las manos serán de 0,15 metros (para el eje Y).

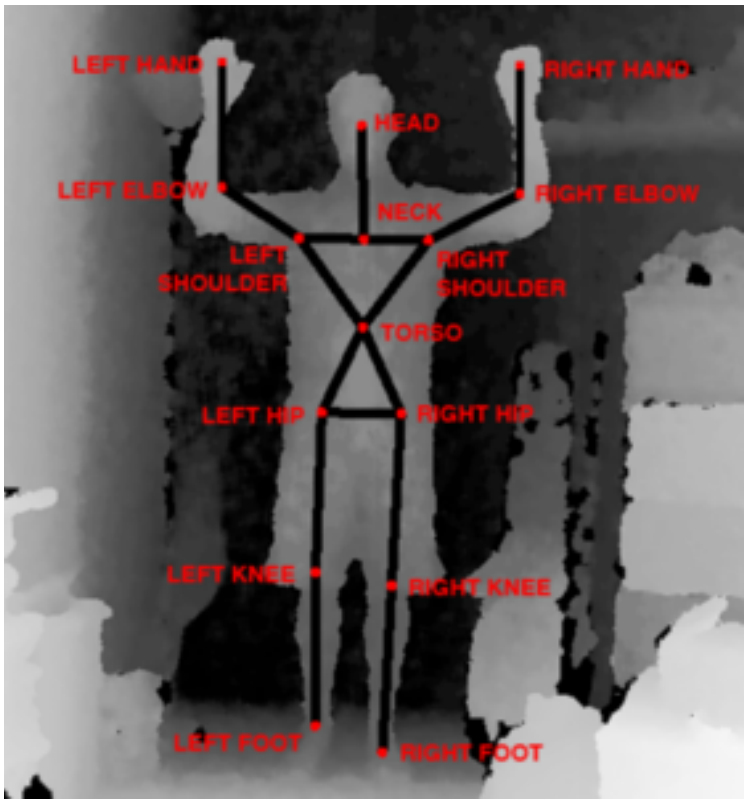
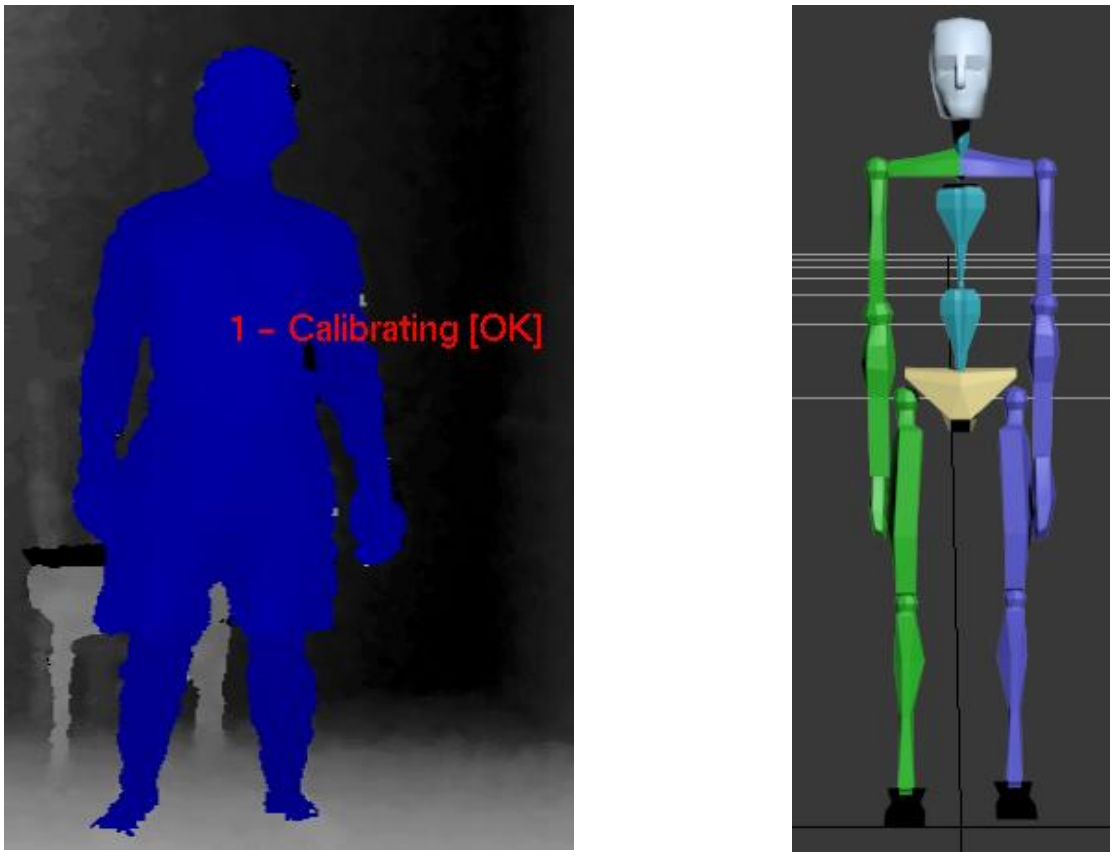


Figura 2-6. Articulaciones del esqueleto según NITE.

El mismo problema sucede con el END SITE de la cabeza. Como se puede observar en la figura 2-6, no hay nada más allá de la articulación *Head* para darle al END SITE de la articulación. La solución fue repartir la distancia entre la cabeza y el cuello entre el OFFSET de la articulación y su END SITE. De tal forma que el OFFSET será un tercio de esa distancia y el resto iría para el END SITE, esto es así dado que el cuello es más corto que la cabeza.

Como último apunte, NITE mide las posiciones en milímetros. Dado que BVH trabaja con metros, tendremos que hacer conversión de nuestros cálculos antes de añadirlo al fichero BVH.

La figura 2-7 muestra el esqueleto resultante de estos cálculos en 3dsMAX posicionado en postura inicial. Su frame número cero se ha forzado para que todas sus rotaciones y traslaciones sean nulas y así se pueda ver como queda el esqueleto sin estar en movimiento.



(a) *(b)*

Figura 2-7. (a) Usuario posicionado en postura inicial para el calibrado del esqueleto y (b), esqueleto resultante en 3dsMAX con postura inicial.

b) Rotaciones

Las rotaciones [MJB-98] son presentes únicamente en el apartado de movimiento del fichero BVH. Cada articulación tiene tres canales por el que se debe enviar la información de las rotaciones de cada eje. Éstos ángulos, medidos en grados, son ángulos de Euler [Eul-03]. Los ángulos de Euler son una representación para las rotaciones tridimensionales dónde se descompone la rotación en tres ángulos separados. Sin embargo, no usaremos directamente los ángulos de Euler. Se usarán matrices para poder calcular las orientaciones de una articulación y, a partir de éstas, convertirlas en ángulos de Euler.

NITE nos da la posibilidad de olvidarnos del cálculo de matrices dado que él mismo es capaz de calcularlas. No obstante, para tener un mayor control sobre los datos que procesamos y evitar problemas de compatibilidad con el formato BVH, tal y como se comenta mas adelante en el apartado 2.2.3, se calcularán manualmente todas las matrices.

Las matrices de orientación, o de rotación, son otra forma de representar los ángulos de rotación de cada eje en una sola estructura. Para comenzar, hay que entender qué información van a contener nuestras matrices de orientación. Una matriz de orientación es una matriz cuadrada de 3x3 que está formada por tres vectores, los cuales serán las columnas de la matriz. Habitualmente son de 4x4 porque contienen también información sobre traslaciones, pero como solo necesitamos las rotaciones, será suficiente con una dimensión 3x3. Podemos ver la estructura de la matriz de orientación en la figura 2-8.

$$\text{Matriz de Orientación} = \begin{pmatrix} vX_1 & vY_1 & vZ_1 \\ vX_2 & vY_2 & vZ_2 \\ vX_3 & vY_3 & vZ_3 \end{pmatrix}$$

Figura 2-8. Estructura de una Matriz de Orientación.

Las matriz está definida por tres vectores que componen las columnas de la matriz. Los vectores se definen tal que:

vector X ⟶ $vX = (vX_1, vX_2, vX_3);$

vector Y ⟶ $vY = (vY_1, vY_2, vY_3);$

vector Z ⟶ $vZ = (vZ_1, vZ_2, vZ_3);$

La estructura de la figura 2-8 es la que NITE nos proporciona y en el proyecto se va a imitar para mantener el mismo estilo. Cada vector representa uno de los tres ejes (x, y, z) de una articulación dada. Los vectores estarán normalizados y, si el esqueleto estuviera en postura inicial, los ejes equivaldrían a los de la figura 2-9.

Esta figura muestra los ejes de rotación en postura inicial, por lo que en ese instante, las matrices de orientación equivaldrían todas a la matriz de identidad. En dicha figura faltan las articulaciones correspondientes a las clavículas (*Left* y *Right Collar*). Se debe a que, tal y como se comentó en los cálculos de distancias, las clavículas son articulaciones fijas y ambas se encuentran en la posición del cuello. Al ser fijas, su matriz siempre será la identidad, sin importar la postura del esqueleto (cuando se rellenen los canales de estas dos articulaciones, siempre pondremos cero grados para los tres ejes).

Las matrices de orientación son en realidad la multiplicación de tres matrices, una para cada eje. Es decir, si consideramos que se necesita una matriz de rotación para cada eje, al aplicar esas rotaciones una a una (multiplicando las matrices), obtenemos una matriz con toda la información. Observemos un ejemplo para entenderlo mejor. La figura 2-10 muestra la fórmula de las rotaciones básicas así como de la matriz resultante final.

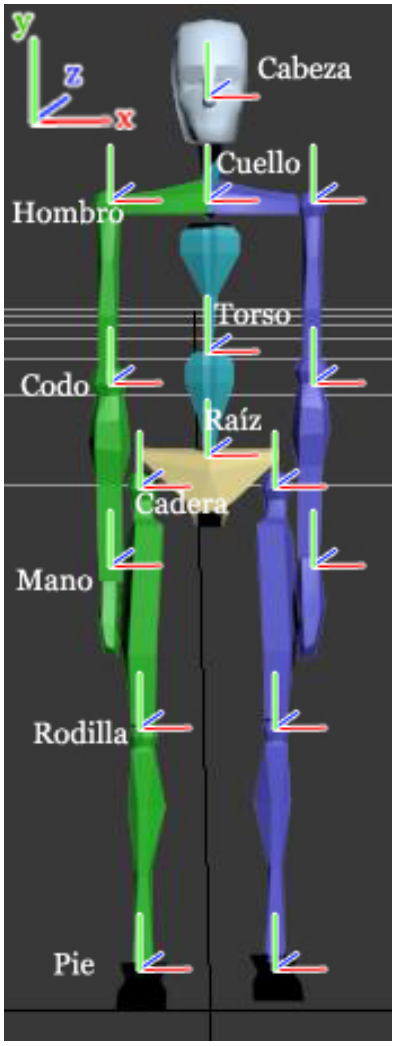


Figura 2-9. Esqueleto en 3dsMAX con ejes de rotación en cada articulación. La figura está orientada sin modo espejo, por lo que los huesos verdes son la parte derecha del esqueleto y los azules la izquierda.

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$R_z(\psi)\,R_y(\theta)\,R_x(\phi) = \begin{bmatrix} \cos\theta\cos\psi & -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix}$$

Figura 2-10. Estructura de rotaciones básicas. La matriz que las une está en modo left-handed.

Los vectores que forman una matriz de orientación están normalizados. Viendo la matriz

grande de la figura 2-10, a partir de la resultante de las tres rotaciones básicas, se puede calcular los ángulos de Euler aislando los valores de los senos y cosenos de la matriz. Las funciones trigonométricas devuelven los resultados en radianes, como el formato BVH funciona con grados, hay que convertirlos de radianes a grados.

En este proyecto se usa la librería matemática CML (*Configurable Math Library*) [NyA-03] para tal fin. Es importante escoger un orden de los 12 posibles que hay para obtener los ángulos (xyz, yzx, zxy, zyx, xzy, yxz, xyx, xzx, yxy, yzy, zxz, zyz). El orden implica que ángulo se aplica antes. Para el proyecto se probaron todos y el que mejor se ajustaba, es decir, el que daba los resultados mas coherentes, era el orden xyz. Cabe decir que, por problemas de compatibilidad con los sistemas de coordenadas entre el formato BVH y NITE (como que el eje Z crezca en dirección inversa a lo convencional), además de no compartir sistema *left* o *right handed*, han aparecido problemas en la orientación de los ángulos. No obstante, para arreglarlo basta con invertir el signo de alguno de los ángulos.

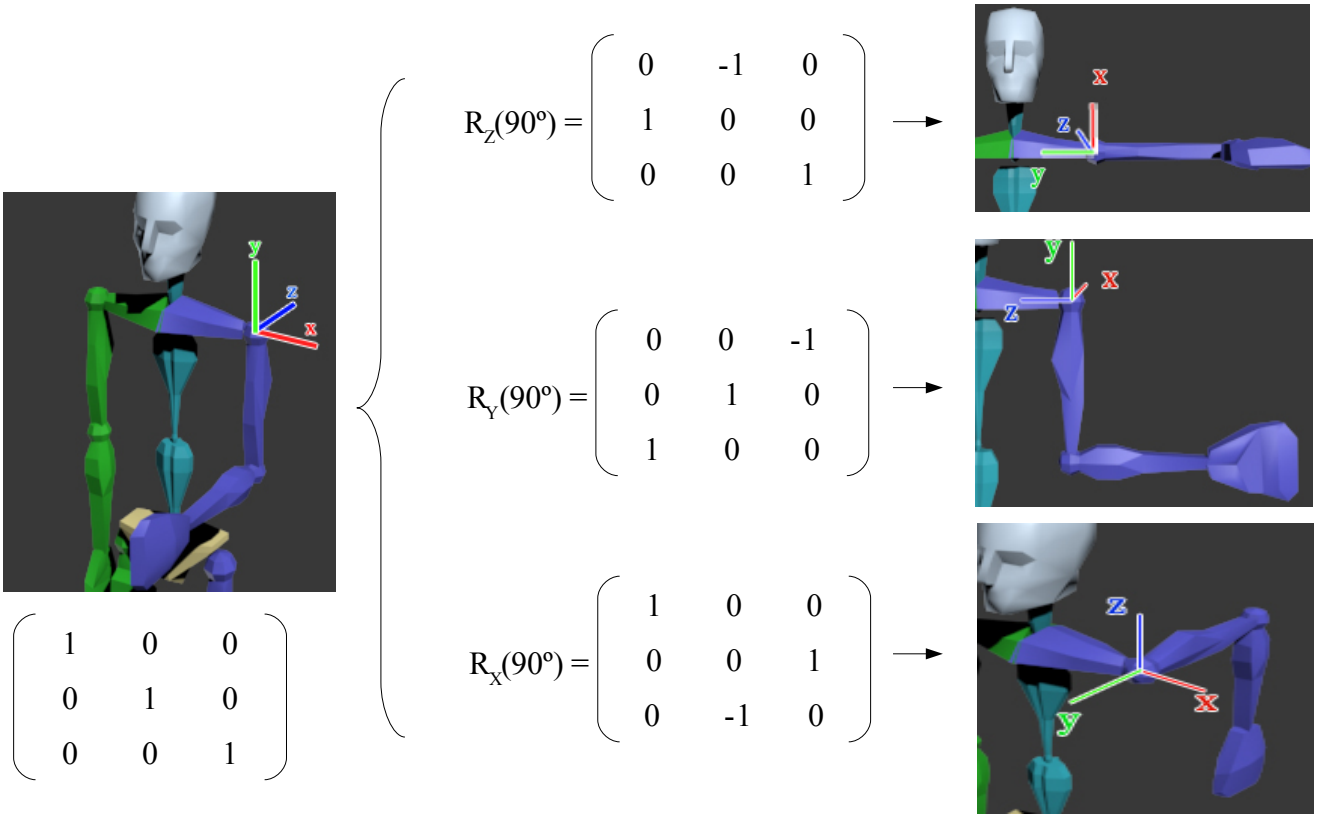


Figura 2-11. Ejemplos de rotaciones básicas sobre el hombro izquierdo. A la izquierda la matriz identidad como posición inicial

Para poder visualizar cómo se modifican los vectores de los ejes de las articulaciones según las rotaciones se puede observar la figura 2-11, dónde se aplica una rotación de 90 grados en cada eje al hombro izquierdo por separado. Rotar en 90 grados significa rotar en sentido anti-horario, rotar en -90 grados es, por ende, rotar en sentido horario. Observando detenidamente los ejemplos, se puede ver como el hecho de que el eje Z sea positivo si nos alejamos de la cámara afecta a las matrices de rotación resultantes. Las matrices de los ejemplos representan las rotaciones con el sistema de coordenadas de NITE, pero si miramos las fórmulas de rotación básica en la figura 2-10

se puede ver como hay algunos signos que están invertidos en las matrices que rotan los ejes Y y Z. sin(90º) equivale a 1, sin embargo, como en el espacio tridimensional que usa NITE la Z es positiva alejándonos de la cámara, el resultado es -1. Como ya se ha mencionado antes, se han aplicado cambios de signo en el código para que las rotaciones fueran coherentes.

Una vez se ha explicado lo que una matriz de orientación conlleva, procedemos a detallar cómo construir una matriz con los vectores que representan los tres ejes X, Y y Z (son los vectores que forman la matriz de la figura 2-8). Se utilizan los propios huesos del esqueleto como vectores para rellenar la matriz. La idea consiste en encontrar, para cada articulación, qué huesos son los que necesitamos para representar cada vector o para que nos ayuden a calcular los vectores que necesitamos. En general, hay tres tipos de casos para construir las matriz de cada articulación según la figura 2-8. Los tipos dependen de si la articulación tiene un solo grado de libertad, esta situada en el tronco del esqueleto o bien pertenece a una extremidad (piernas o brazos).

1) **Articulaciones del tronco:** Son el caso de las articulaciones raíz (*Hips*), torso (*Torso*) y cuello (*Neck*). Estas articulaciones son las más sencillas de calcular dado que podemos usar el hueso que va de cadera a cadera (en el caso de la raíz) y el que va de hombro a hombro (en el caso del torso y el cuello) como vector X. El vector Y tiene que ser perpendicular a los huesos que hacen de vector X de forma que se convierta en el denominado *up-vector*. Tenemos los huesos que van de la raíz al torso (en el caso de la raíz), el hueso que va del torso al cuello (en el caso del torso) y el hueso que va del cuello a la cabeza (en el caso del cuello). Para encontrar el vector Z, no es posible usar ningún hueso. Sin embargo, el vector Z tiene que ser perpendicular a los otros dos, por lo tanto, si hacemos el producto vectorial (*cross product)* [EWW-95] de los vectores X e Y obtendremos el vector Z que buscamos. Es importante recordar que los vectores X e Y deben haber sido normalizados antes de hacer el producto vectorial.

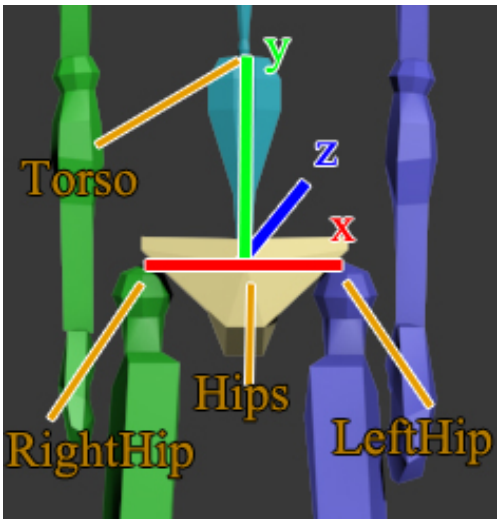


Figura 2-12. Localización de los vectores para hacer la matriz de rotación de la articulación raíz.

Para los vectores:

$$U = (u_x, u_y, u_z) \quad , \quad V = (v_x, v_y, v_z)$$

Tenemos que la fórmula del producto vectorial es:

$$U \times V = (x', y', z') = x'(u_y v_z - u_z v_y) - y'(u_x v_z - u_z v_x) + z'(u_x v_y - u_y v_x)$$

convirtiéndose así en un vector perpendicular a U y V según el sistema *right-handed.* Una vez tenemos los tres vectores, solo queda colocarlos cada uno en una columna de nuestra matriz.

En la figura 2-12 se visualizan los vectores usados para crear la matriz de rotación de la articulación raíz (*Hips*). La figura muestra en naranja el nombre de las articulaciones que necesitamos. Tal y como se ha comentado, el vector X (vector rojo) va de *LeftHip* a *RightHip*, el vector Y (vector verde) va del *Hips* a *Torso* y el vector Z (vector azul) es el resultado de hacer el producto vectorial de los vectores X e Y. Colocando estos vectores como columnas de una matriz, obtendremos la matriz de orientación de la articulación raíz concordando con la figura 2-8.

2) **Articulaciones con un solo grado de libertad:** Son el caso de las rodillas (*Left* y *Right Knee*)y los codos (*Left* y *Right Elbow*). Ninguna de las cuatro articulaciones tiene más de un grado de libertad, lo cual nos facilita mucho los cálculos, tanto, que no es siquiera necesario usar matrices para calcular los ángulos de Euler. El eje que se puede rotar es el eje X.

Se utilizará la ley de los cosenos [LoC-02] para poder calcular el ángulo. Hay que imaginar que las articulaciones de la extremidad (hombro-codo-mano o bien cadera lateral-rodilla-pie) forman un triángulo dónde los lados son la distancia de una articulación a otra (en este caso no usamos vectores para los huesos). Así, teniendo la notación de la figura 2-13:

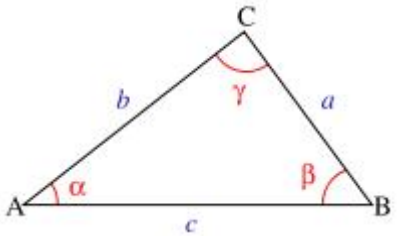


Figura 2-13. Triángulo con lados y ángulos nombrados.

Si fuera para calcular el ángulo del codo, tendríamos que A es el hombro, C el codo y B la mano. Así pues, tenemos la fórmula:

$$\gamma = arcos((a^2 + b^2 – c^2) / 2ab)$$

El resultado es en radianes dado que las operaciones aritméticas usadas devuelven los resultados con esa unidad. Dado que BVH funciona con grados lo único que quedaría por hacer es convertir el resultado en grados.

3) **Articulaciones en extremidades:** Es el caso de los hombros (*Left* y *Right Shoulder*) y los laterales de las caderas (*Left* y *Right Hip*). Éstas son más problemáticas dado que son algo independientes al tronco y es más difícil encontrar una referencia. Ahora solo uno de los vectores coincidirá con un hueso (será el vector Y) mientras que el resto, se deberán calcular usando como referencia un hueso auxiliar.

El vector Y siempre es fácil de encontrar. Para estas articulaciones sería el hueso que va desde el codo/rodilla hasta el hombro/cadera. Lo complicado es encontrar los otros dos

vectores. No podemos obtener ningún vector directamente de un hueso, pero podemos usar un hueso para que nos ayude a encontrar algún vector usando el producto vectorial. Teniendo dos vectores, el tercero es tarea fácil, pues solo hay que hacer el producto vectorial de los dos que tenemos para obtener el tercero.

En general, usaremos el hueso que va del codo/rodilla hasta la mano/pie, respectivamente según la articulación (caso (a) de la figura 2-14). Este segundo hueso crea un plano con el que teníamos para el vector Y, así que, si hacemos el producto vectorial del hueso y del vector Y obtenemos el perpendicular al plano, el cual equivaldría al eje X. Y así, tenemos el vector Z. No obstante, hay un problema. Si el brazo entero o la pierna entera está recta (no se han doblado los codos o las rodillas), el hueso auxiliar será colineal con el que usamos para el vector Y, lo que no nos permitirá poder formar un plano. Para saber cuando dos vectores son colineales se utiliza el producto escalar [Dot-02] (*dot product*). Si el resultado equivale a 1, entonces son colineales. Daremos un de margen de error del 10 por ciento por si acaso. La fórmula del producto escalar es:

$$\theta = \arccos\left(\frac{\mathbf{a}\cdot\mathbf{b}}{\|\mathbf{a}\|\,\|\mathbf{b}\|}\right).$$

Si se da el caso de que son colineales (caso (b) de la figura 2-14), necesitamos otro hueso auxiliar. Se decidió usar el hueso que va de hombro a hombro (en el caso de los brazos) y el que va un lado a otro de las caderas (en el caso de las piernas). Con este hueso formamos un plano con el vector Y, pero esta vez, el plano nos dará el vector X en vez del Z. Teniendo los vectores Z y Y, podemos obtener el vector X haciendo el producto vectorial de los otros dos. Eso si, con este sistema se pierde la orientación del eje Y, por eso solo se usa si no hay más remedio. De todas formas, cuando un brazo o una pierna está completamente estirada, no se puede saber su rotación en el eje Y.

Hay un último contratiempo (caso (c) de la figura 2-14). El vector Y puede alinearse con el nuevo hueso auxiliar si el esqueleto ha rotado 90 grados en su eje Z (como si el esqueleto formara una T con su cuerpo). Cambiamos el hueso auxiliar y esta vez se usa el que va del torso al cuello. Así, como antes, haciendo el producto vectorial del vector Y con el hueso auxiliar se obtiene el vector Z, y de nuevo, haciendo el producto vectorial del vector Z con el Y, se obtiene el vector X. Este caso solo ocurre con los brazos. Sería posible con las piernas, pero no se ha tenido en cuenta que una persona llevara tan lejos la animación hasta el punto de abrirse de piernas. La figura 2-14 muestra cada caso y como se ven los vectores y huesos auxiliares.

Hasta aquí ya podríamos calcular todas las matrices de las articulaciones. Pero aún no hemos terminado. Se indicó al principio del apartado que el formato BVH usaba rotaciones locales, por lo que todas las rotaciones que aplicáramos a una articulación afectarían a sus hijos. Usando los cálculos comentados hasta ahora solo servirían para construir las matrices de forma global, es decir, sin tener en cuenta ninguna rotación de articulaciones padre. Volviendo de nuevo al modelo jerárquico del esqueleto, definido en la figura 2-3, cogiendo una de las ramas que nacen de la raíz tenemos que:

$$Rama_{i} = \{Articulación_{n}, Articulación_{n-1}, \ldots, Articulación_{0}\};$$

Por ejemplo, la rama que representa todo el tronco sería:

$$Rama_{Tronco} = \{Head, Neck, Chest, Hips\};$$

La articulación 0 siempre será la raíz. Una matriz global para una articulación i-ésima es un producto de matrices tal que:

$$Mat_{GLOBAL\_i} = Mat_{LOCAL\_i} * Mat_{LOCAL\_i-1} * \dots * Mat_{LOCAL\_0}$$

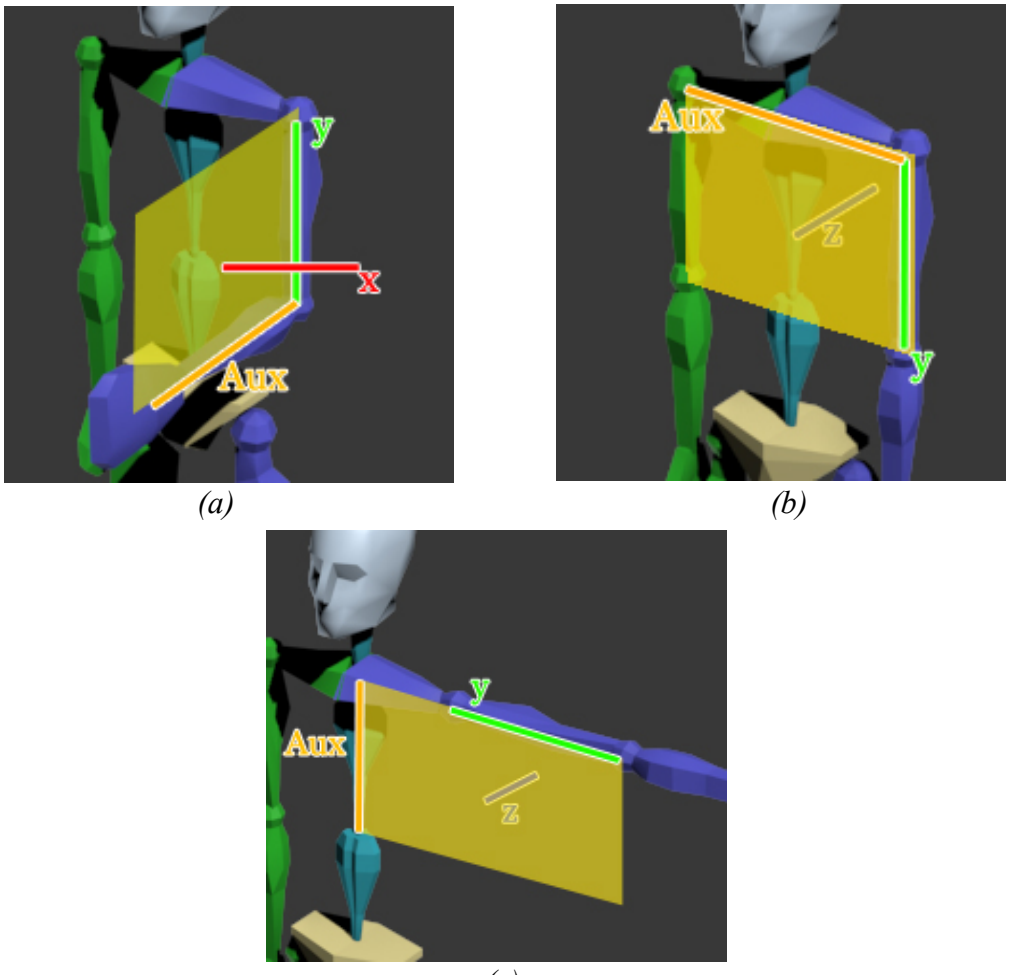
La matriz que nos interesa obtener es la Mat_{LOCAL_i}. Esta matriz es la última rotación que se hace ya que multiplicando por la izquierda añadimos rotaciones (estamos en modo left-handed). Cada matriz que multiplica por la derecha a Mat_{LOCAL_i} es la matriz de rotación local del padre, la siguiente que multiplica sería la matriz padre del padre, y así hasta llegar a la matriz de la raíz. Ahora bien, sabiendo que la matriz global del padre (Mat_{LOCAL_i-1}) equivale a todo lo que está a la derecha de Mat_{LOCAL_i}, nos queda que:

$$Mat_{GLOBAL\_i} = Mat_{LOCAL\_i} * Mat_{GLOBAL\_i-1}$$

Obtener Mat_{LOCAL_i} a partir de aquí es fácil:

$$Mat_{LOCAL\_i} = Mat_{GLOBAL\_i} * Mat_{GLOBAL\_i-1}^{-1}$$

Multiplicando la matriz global inversa del padre por la derecha nos quedamos con la matriz local que buscábamos. Así pues, tenemos que ir guardando en una estructura todas las matrices globales de cada articulación para que los nodos hijos puedan usarlas.



(a) *(b)*

(c)

Figura 2-14. Tres casos para hacer las matrices de rotación de los hombros según la posición de los dos huesos que forman los brazos.

Como ejemplo, si queremos calcular la matriz local de la articulación del cuello (*Neck*), tenemos que:

$$Mat_{GLOBAL\_Neck} = Mat_{LOCAL\_Neck} * Mat_{LOCAL\_Chest} * Mat_{LOCAL\_Hips}$$

Y como:

$$Mat_{GLOBAL\_Chest} = Mat_{LOCAL\_Chest} * Mat_{LOCAL\_Hips}$$

Entonces:

$$Mat_{GLOBAL\_Neck} = Mat_{LOCAL\_Neck} * Mat_{GLOBAL\_Chest}$$

Finalmente, si queremos la matriz local de la articulación del cuello:

$$Mat_{LOCAL\_Neck} = Mat_{GLOBAL\_Neck} * Mat_{GLOBAL\_Chest}^{-1}$$

Las articulaciones de las clavículas se ignoran, haciendo que el torso tenga como hijos a los hombros en vez de las clavículas. A su misma vez, los codos y las rodillas, al no necesitar matrices porqué solo tienen un grado de libertad, este paso se omite y ni siquiera se guardan las matrices globales.

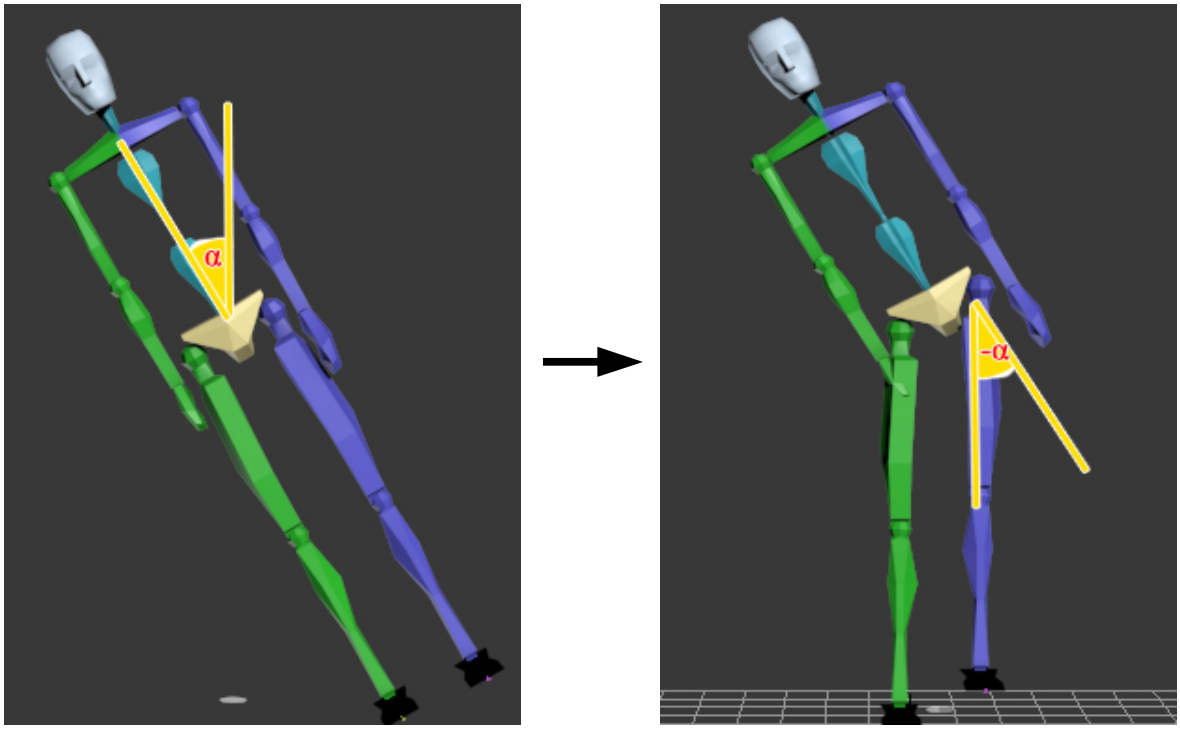


Figura 2-15. Corrección aplicada a las piernas para no verse afectada por la rotación de la articulación raíz. A la izquierda sin corregir y a la derecha corregido.

Hay un problema con las articulaciones *Left* y *Right Hip*. En la realidad, si nosotros rotamos las caderas (la articulación raíz), nuestras piernas siguen fijas al suelo sin rotar. Sin embargo, el formato BVH no lo comprende así. Si rotamos la raíz, ésta aplica su rotación a todos sus hijos, y las piernas no son una excepción, por lo que hay que aplicar una corrección a *Left* y *Right Hip*. Para estas articulaciones, no seguimos la formula anterior sino que usamos la siguiente:

$$Mat_{LOCAL\_LeftHip} = Mat_{Raiz}^{-1} * Mat_{GLOBAL\_LeftHip}$$

Multiplicando por la inversa de la matriz de la articulación raíz (la cual, al ser la raíz solo

puede ser local), estamos añadiendo a la matriz de *LeftHip* la rotación en sentido contrario al de la raíz. De esta forma, se corrige la orientación y las piernas quedan en su sitio aunque la raíz rote, dejándolas independientes.

La figura 2-15 muestra como se ven las piernas sin esta corrección y con la corrección aplicada.

2.1.3 Interfaz de usuario (GUI)

Este apartado trata sobre el módulo correspondiente a la interfaz gráfica del usuario (GUI), así como del muestreo por pantalla de la información recabada por Kinect y la que nos ofrece NITE.

La GUI de la aplicación principal, *BVHSkeletonTracker,* es una modificación del ejemplo *NiUserTracker* que NITE proporciona en código para que se adapte a las necesidades del proyecto. En todo momento, la GUI muestrea lo que la cámara de profundidad de Kinect percibe, unos píxeles en tonalidades de gris. Un elemento del entorno será pintado con tonalidades más claras cuanto más cerca esté de la cámara. Luego, NITE nos proporciona información sobre los píxeles que pertenecen a un usuario, los cuales, una vez el usuario sea detectado, se pintarán de colores vivos. Además, una vez detectado al usuario, NITE nos proporciona la posición de las articulaciones del usuario. Estas posiciones las podemos utilizar para pintar puntos y líneas para dibujar el esqueleto. Todo lo que estamos pintando en pantalla lo haremos usando OpenGL.

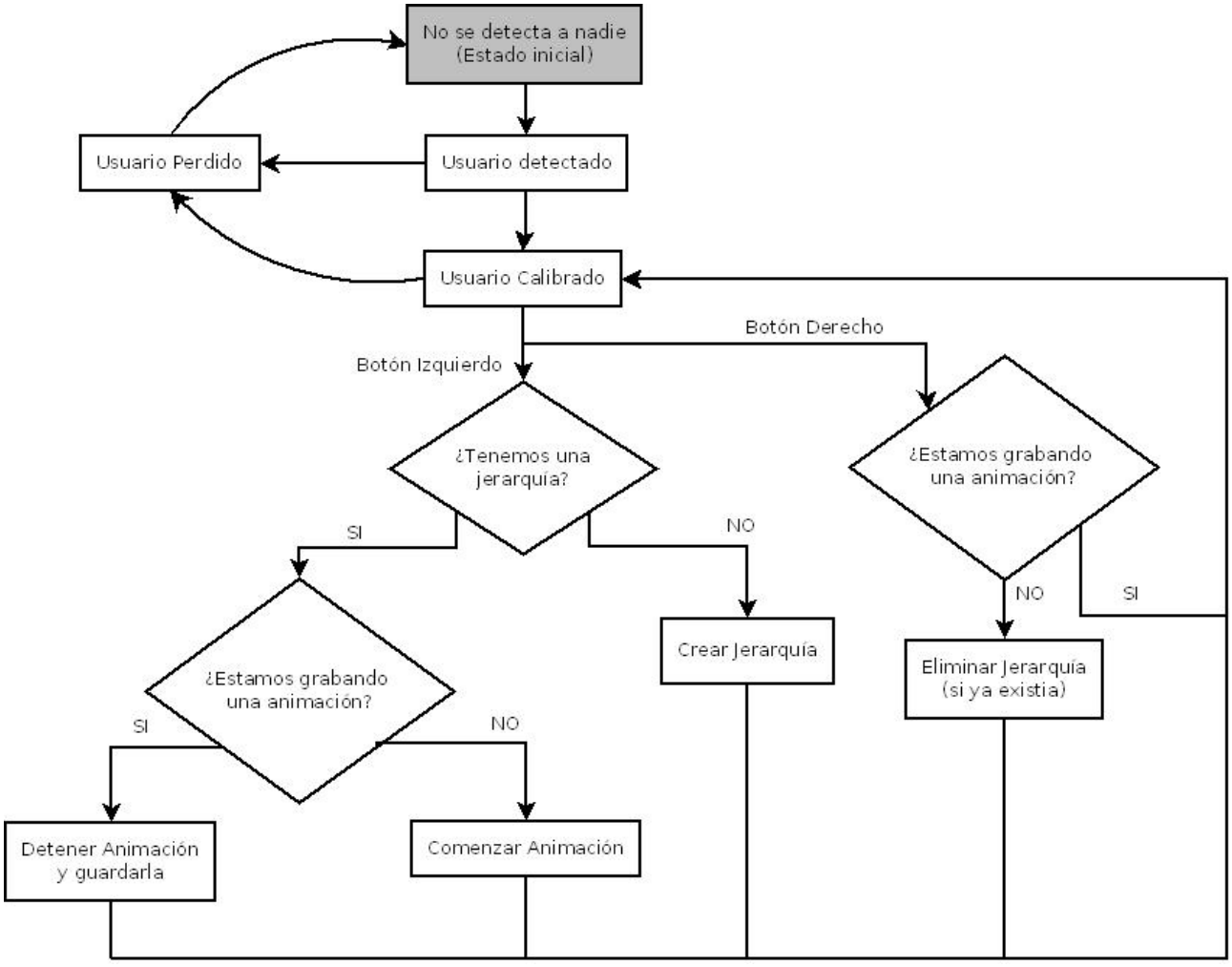
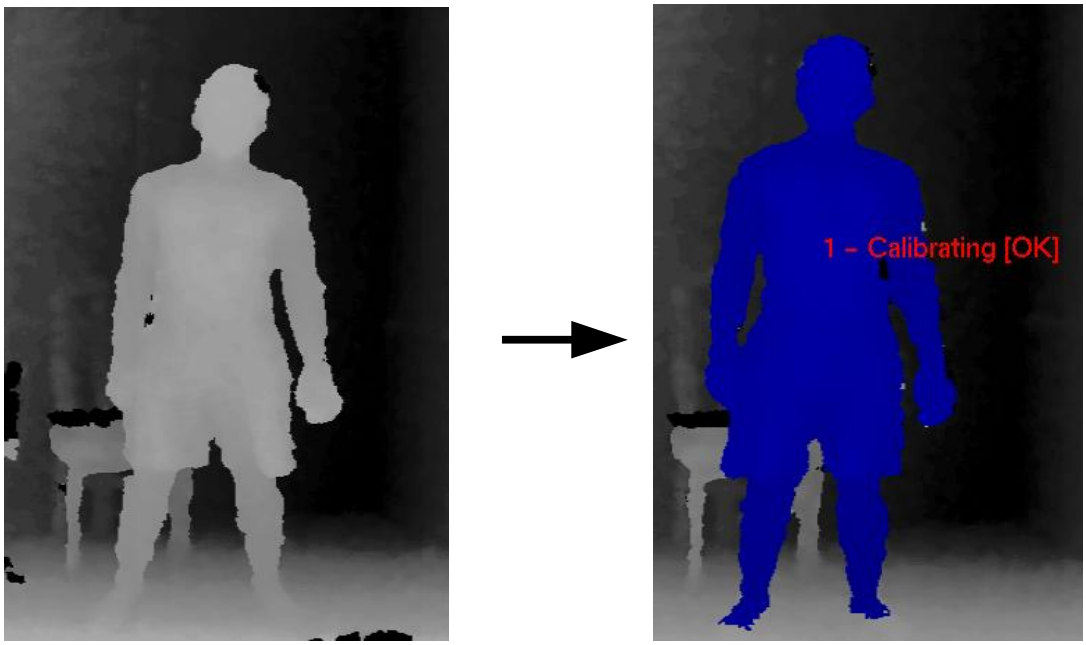


Figura 2-16. Diagrama de estados de la aplicación principal.

En el diagrama de la figura 2-16 podemos ver los estados de la aplicación principal. El estado inicial representa el momento en el que NITE no detecta a ningún usuario aún (figura 2-17 (a)). Una vez NITE reconoce al usuario, la aplicación lo pintará de azul y comenzará a calibrar al usuario (figura 2-17 (b)). Durante el proceso, NITE conocerá las posiciones de las articulaciones del usuario y se podrá pintar su esqueleto. En este punto, aún queda terminar bien la calibración. Si bien ya podríamos comenzar a grabar desde aquí, es recomendable terminar la calibración antes de comenzar a grabar para que los cálculos tengan menos posibilidad de error.



(a) *(b)*

Figura 2-17. a) Aún no se ha detectado un usuario. b) Usuario detectado

En la figura 2-18 podemos ver que la calibración completa se distingue por la posición de la etiqueta *Tracking* que flota en el cuerpo del usuario detectado. Si flota sobre la articulación raíz (figura 2-18 (b)), la calibración estará terminada. Para que la calibración se haga antes, se puede ayudar a NITE colocándose como en la postura que indica la figura 2-19, así se agilizará el proceso.

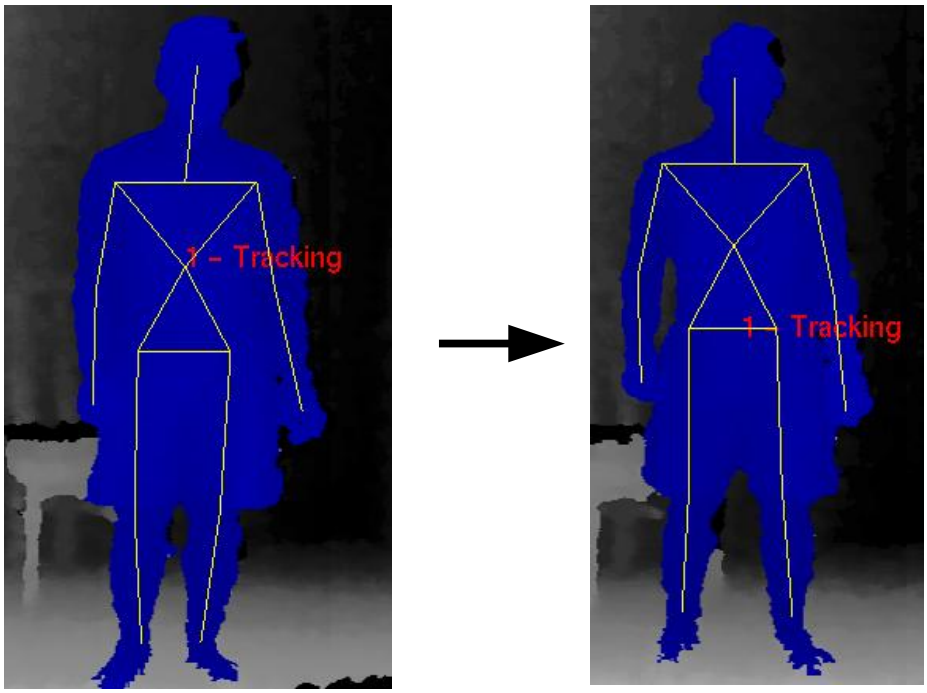
A partir de aquí, podemos crear la estructura, en texto, del esqueleto para el formato BVH. La estructura del esqueleto es la jerarquía escrita en texto que correspondería a la parte *Hierachy* del formato BVH. Recordamos que para una buena construcción del esqueleto el usuario debe colocarse en postura inicial, que sería recto y erguido, tal y como indica la figura 2-7. Solo necesitamos una instancia del esqueleto, así que una vez creemos una estructura para el esqueleto, la mantendremos para todas las animaciones que hagamos, a no ser que pulsemos el botón derecho, el cual borrará la estructura actual. El botón derecho sirve por si se ha construido la jerarquía en el momento en que el esqueleto no estaba en postura inicial o que daba problemas en la posición de las articulaciones. Se recomienda hacer la jerarquía del esqueleto cuando aún no se ha calibrado del todo al usuario (figura 2-18(a)). Esto se debe a que la calibración completa deja las articulaciones de una forma que la estructura del esqueleto queda ligeramente deformada cuando lo importamos al 3dsMAX.

Una vez tengamos una jerarquía, pulsando el botón izquierdo del ratón en cualquier posición de la ventana, comenzamos a grabar la animación. Aquí es dónde los cálculos comenzarán a

realizarse a medida que pasan los frames. Si volvemos a pulsar el botón izquierdo del ratón la animación se detendrá y se guardará en un archivo BVH. Pulsar de nuevo el botón izquierdo del ratón hará comenzar una nueva animación que acabará guardándose en otro archivo. Estos archivos están nombrados de la forma:

animationX.bvh

Donde X corresponde al número de animación actual. La X siempre comienza en 1, así que si reiniciamos la aplicación hay que vigilar de que no se sobrescriban animaciones que nos interesen.



(a) *(b)*

Figura 2-18. a) Esqueleto no calibrado del todo. b) Esqueleto calibrado por completo.

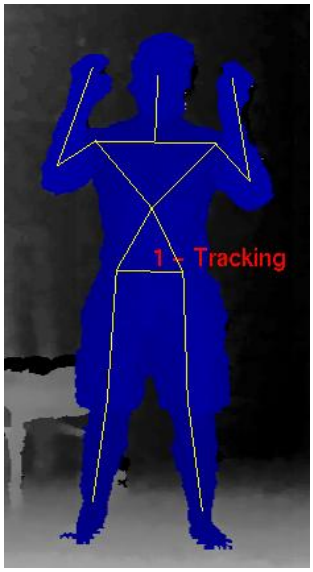


Figura 2-19. Posición que puede agilizar la calibración completa.

Durante la animación el botón derecho no tendrá efecto. Además, si el usuario se pierde, se detendrá la grabación de la animación y no se guardará. Si teníamos guardada una jerarquía, no se perderá aunque el usuario deje de detectarse.

La aplicación puede detectar a más de un usuario. En la pantalla veremos a los otros usuarios de un color distinto al azul (según el orden de detección, pueden verse de color verde, amarillo, entre otros). No obstante, solo se harán cálculos con el usuario número uno (el de color azul). No es recomendable que aparezca más de un usuario por si pudiera corromper los resultados. Hay que tener cuidado de no mover los objetos de la sala, dado que NITE puede pensarse que un objeto es un usuario. No llegaría a calibrarlo dado que no encontraría su esqueleto, pero puede molestar tener a un usuario nuevo o que incluso el objeto tome el rol de usuario uno. Si el usuario a grabar no se detecta como número uno, se recomienda reiniciar la aplicación.

Lo que se percibe en la ventana es una imagen a modo espejo. Si movemos, por ejemplo, el brazo derecho, parecerá que en el muestreo se mueva el brazo izquierdo.

La aplicación hereda del ejemplo de NITE, *NiUserTracker*, otros comandos por teclado listados a continuación:

- **'b':** Pintar o no el fondo. Si se desactiva solo veremos los píxeles que representan a los usuarios detectados así como sus esqueletos.
- **'x':** Pintar o no los píxeles. Si se desactiva no se pinta nada de lo que la cámara detecta, dejando solo los esqueletos de los usuarios detectados.
- **'s':** Pintar o no los esqueletos de los usuarios detectados.
- **'i':** Pintar o no las etiquetas que flotan sobre los usuarios detectados.
- **'p':** Pausar o reanudar la grabación de la cámara. No quiere decir que cuando estemos grabando una animación se pause, sino que la cámara se congela hasta que la reanudemos. No es recomendable usar esta función mientras grabamos una animación.
- **'S':** Guardar una calibración.
- **'L':** Cargar una calibración.
- Pulsar ESC en cualquier momento cierra la aplicación. Si alguna animación no se había terminado no se guardará.

2.2 Construcción de la animación

En este apartado se mostrará cómo utilizar los archivos BVH que se generan en la aplicación principal en el 3dsMAX. Una vez se tenga el modelo con la animación del BVH se explicará qué es lo que se puede extraer de la escena para poder usar el modelo animado con el motor gráfico usado en la asignatura de videojuegos de la UAB.

Cargar el BVH en un modelo no es complicado. El propio 3dsMAX se encarga de todo, lo único que hay que hacer es seleccionar lo que sería el 'núcleo' del esqueleto (o cualquier parte del mismo) del modelo y, desde el apartado *Motion*, aplicarle un archivo de tipo m*otion capture*, que en el actual caso será en formato BVH.

A partir de este punto, ya tenemos todo montado. Ahora se prosigue exportando la escena a Cal3D. Teniendo las librerías de Cal3D instaladas en el 3dsMAX, se obtendrán tres archivos:

- **Cal3D Skeleton File (.CSF):** Contiene la información del esqueleto del modelo. Es el primer archivo que debemos exportar. Se crea seleccionando todo el esqueleto del modelo y nada más, entonces se puede exportar a .CSF.
- **Cal3D Mesh File (.CMF):** Contiene la información de una malla del modelo. Para poder exportar las mallas es necesario haber exportado anteriormente el esqueleto (.CSF). Además, se necesita un .CMF para cada malla, así que si el modelo tuviera más de una, se tiene que generar un archivo .CMF para cada una de ellas. Estos archivos se crean seleccionando una única malla (la que queramos exportar) y exportando a .CMF.
- **Cal3D Animation File (.CAF):** Contiene la información de una animación del modelo. Para poder exportar las animaciones se necesita haber exportado anteriormente el esqueleto. Para estos archivos no es necesario seleccionar nada, podemos exportar las animaciones directamente, de una en una. Por lo tanto, si queremos exportar una nueva animación, solo tenemos que volver a aplicar al modelo otro fichero BVH, tal y como se hizo al principio. No será necesario volver a exportar el esqueleto ni las mallas.

También se podrían exportar los materiales del modelo con Cal3D, pero en este proyecto no ha sido necesario.

Finalmente, ya tenemos todo lo necesario para poder ver un modelo animado con Kinect en un motor gráfico que soporte Cal3D.

2.3 Motor Gráfico de la asignatura de videojuegos de la UAB

Este apartado trata sobre como utilizar los archivos generados en formato Cal3D. Se hizo una segunda aplicación para cargar las animaciones exportadas a Cal3D en el motor gráfico usado en la asignatura de videojuegos de la UAB, *DirectX,* para que de esa forma se puedan ver diversas animaciones creadas con anterioridad.

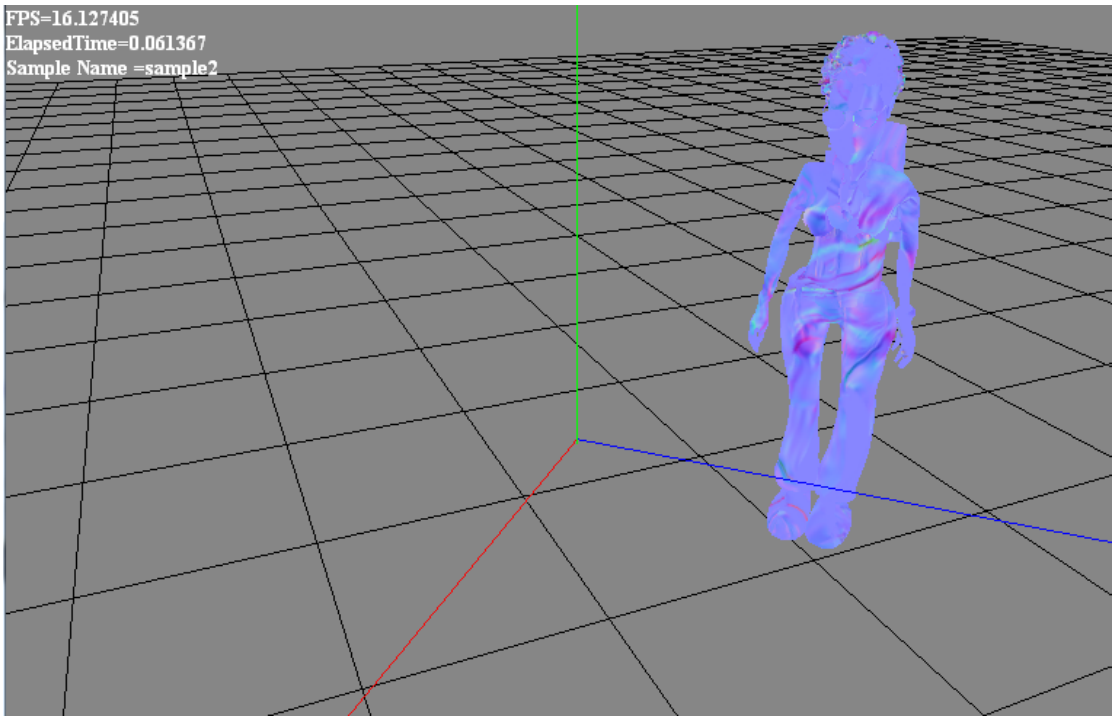


Figura 2-20. Interfaz de la aplicación para visualizar animaciones en Cal3D.

Al abrir la aplicación veremos el modelo cargado encima de un plano. Con el ratón podremos controlar la rotación de la cámara para obtener una mejor visualización, así como hacer zoom con la rueda.

Las teclas numéricas cargan diferentes animaciones listadas a continuación:

1) Caminar.

2) Correr.

3) Saltar.

4) Saludar.

5) Postura *Idle* de lucha.

6) Dar puñetazos.

7) Dar patadas.

Para salir de la aplicación basta con pulsar la tecla ESC.

3. RESULTADOS

En este capítulo mostraremos algunos movimientos captados con la aplicación principal pensados para un videojuego. Los resultados, los archivos en formato BVH, han sido cargados en un esqueleto bípedo del 3dsMAX sin modelo para agilizar el proceso.

El capítulo se divide en dos apartados, uno para comentar los resultados de los experimentos realizados y otro para comentar las incidencias que nos hemos ido encontrando en todo el proyecto en general.

3.1 Experimentos realizados

Este capítulo comenta los resultados obtenidos de los catorce experimentos realizados. Los experimentos tratan de grabar con el proyecto movimientos que nos encontraríamos en los videojuegos. No obstante, los primeros cuatro experimentos serán dedicados para mostrar movimientos simples. En general, las muestras de cada experimento contienen ruido y la intensidad de luz de la habitación no parece importar en el resultado de los experimentos.

1) **Movimientos Básicos con brazos estirados:** Empezamos con movimientos sencillos. En este caso, el usuario solo mueve los brazos de forma que estén completamente rectos. La figura 3-1 nos muestra los puntos por los que el usuario mueve sus brazos. El movimiento consiste en llevar los brazos de un punto a otro. También se ha incluido como punto la postura inicial. En general, los movimientos son fluidos y correctos en las 8 muestras del experimento, aunque hay problemas para detectar el camino del punto A al punto B que muestra la figura 3-1.

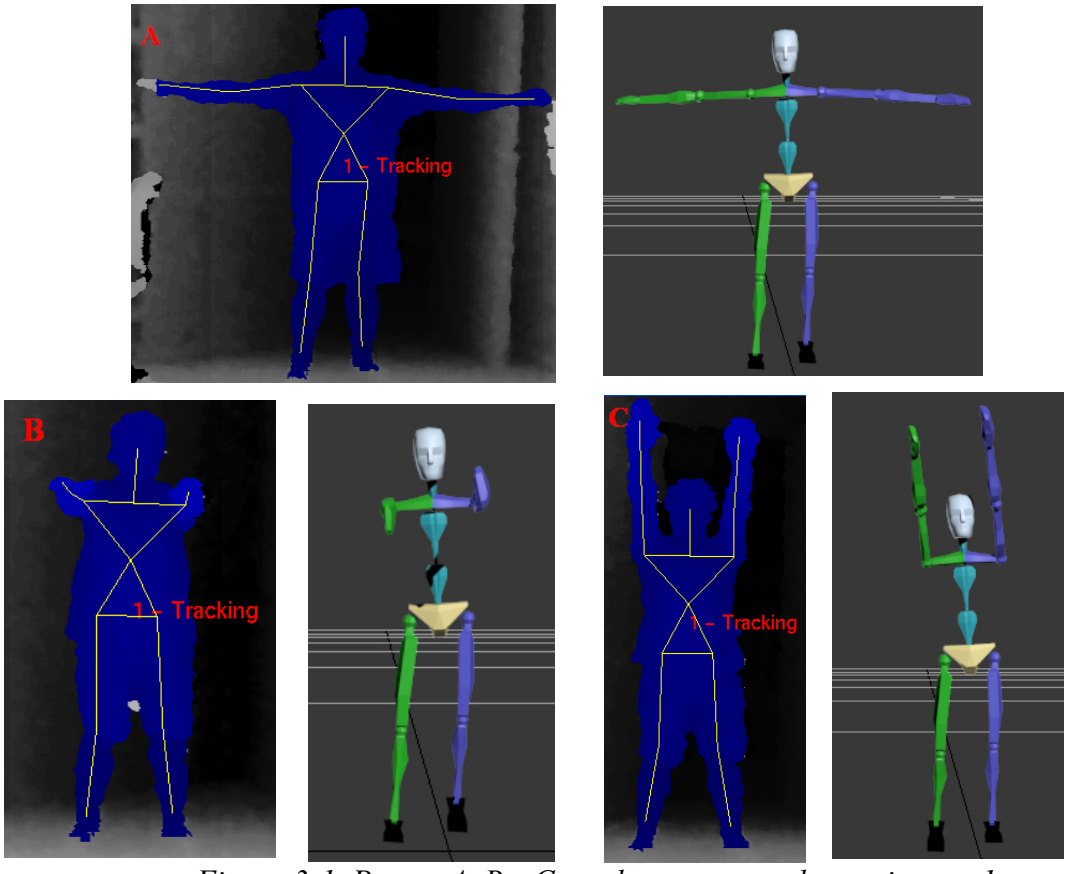


Figura 3-1. Puntos A, B y C por los que pasa el experimento 1.

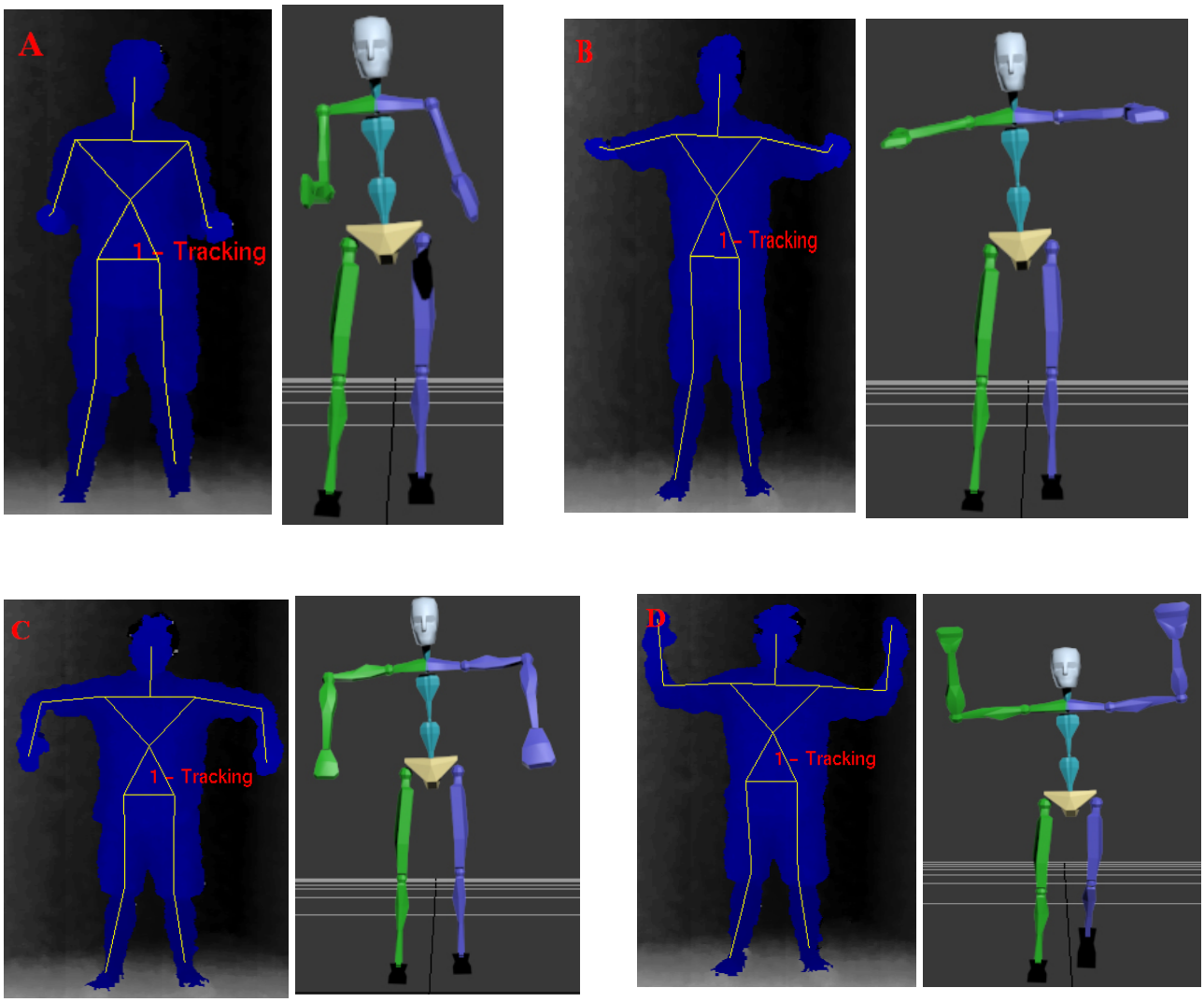
2) **Movimientos básicos con brazos flexionados:** Se flexionan los codos para poder mostrar como el brazo responde si rotamos el miembro sobre el eje Y. En toda la animación, los codos se mantienen flexionados 90º. La figura 3-2 muestra algunos puntos por los que pasa el movimiento del usuario.

   En general, los movimientos son fluidos, no obstante, tiene problemas para captar los movimientos en el punto E mostrado en la figura 3-2. Debido a los problemas que presentan los ángulos de Euler, las rotaciones de los hombros son erróneas. En ese instante, los hombros suelen obtener aproximadamente estos ángulos, en grados:

$$Euler_{Hombro\ Izquierdo} = (rotZ, rotX, rotY) = (58.52, -91.07, 63.49)$$

   Esto se verá como si los brazos estuvieran frente al usuario en vez de a los lados. En general, alzar los brazos alrededor de la zona del caso E causará errores similares, sin embargo, es capaz de hacer bien los casos en los que los brazos estén situados por encima del punto A, esto son los casos que, comenzando desde el punto A, hacen rotar los hombros únicamente con el eje X.

   Por otro lado, casos como el que muestra el punto F de la figura 3-2, en los que solo se rota los hombros con el eje Z, también se detectarán correctamente.



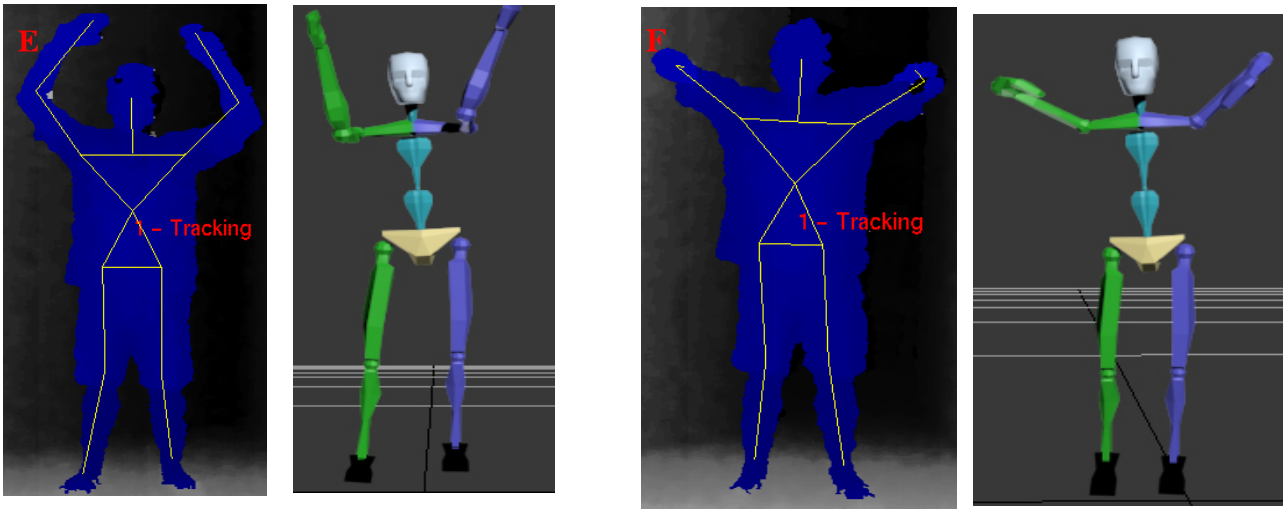


Figura 3-2. Algunos de los puntos por los que pasa el experimento 2.

Se han realizado un total de 12 muestras, dos de las cuales con movimientos más rápidos que el resto. El caso F de la figura 3-2 solo aparece en dos de las muestras, las cuales son dedicadas únicamente para simular el punto F. En general, los resultados son similares y correctos sin contar el caso E. Algunos han calculado mejor que otros los puntos B, C y D.

3) **Movimientos básicos con piernas estiradas:** Se hacen movimientos simples con las piernas estiradas. Cada muestra contiene el mismo movimiento realizado con cada pierna.

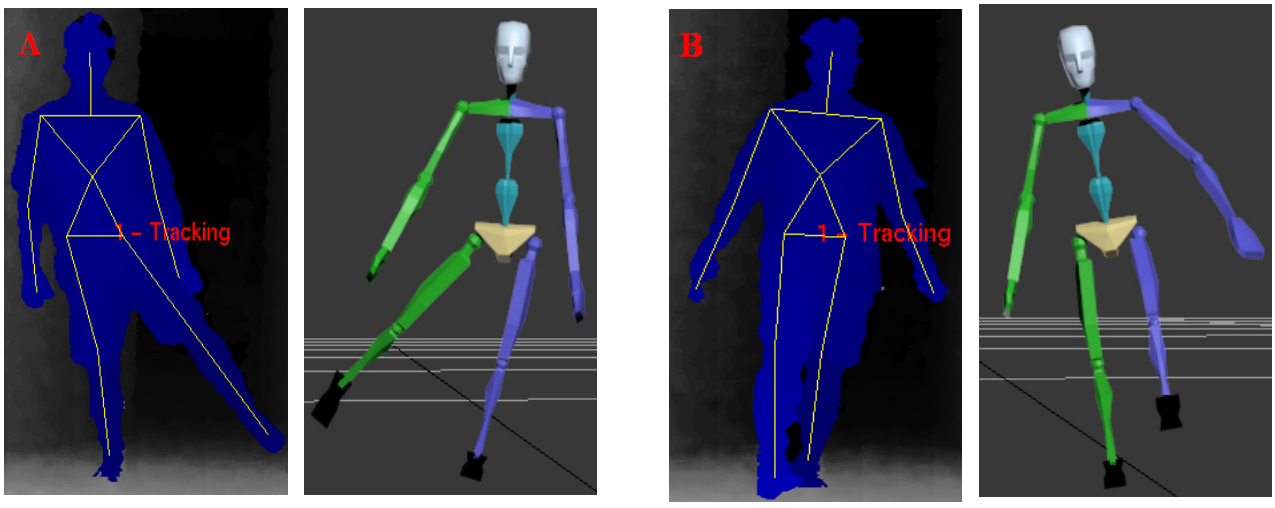


Figura 3-3. Algunos puntos del experimento 3.

Para este experimento se han realizado 6 muestras. Dado que son movimientos muy simples se decidió realizar una menor cantidad de muestras para probar su funcionamiento. Todas las muestras han devuelto resultados similares. Únicamente presentan fallos causados por ruido, principalmente para el eje Y. Esto puede ser debido a que cuando las piernas están estiradas, calcular el ángulo de rotación sobre el eje Y es más complicado o simplemente porqué NITE genera ruido en las piernas. En la figura 3-3 se muestran algunos puntos por

los que pasa la animación (recordamos que lo que nos muestra la aplicación está en modo espejo).

4) **Movimientos básicos con piernas flexionadas:** Ahora se mueven las piernas con las rodillas flexionadas 90º. De esta forma, podemos ver como funcionan las rotaciones del eje Y de las piernas a la vez que rotamos otros ejes.

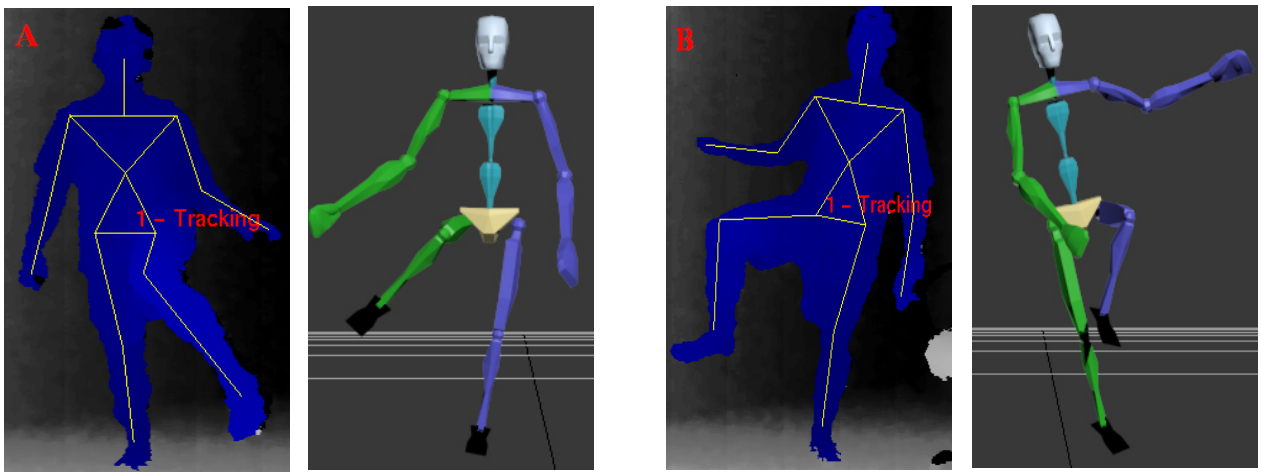


Figura 3-4. Algunos puntos del experimento 4.

Como en el caso anterior, se han hecho unas 6 muestras. Cada una contiene los mismos movimientos para cada pierna. En general los resultados son los mismos, esta vez con una cantidad notablemente menor en las piernas que en el caso anterior, tanto para la pierna que queda estirada como para la que está en movimiento. No obstante, para el caso B que muestra la figura 3-4, hay problemas para detectar la rotación del eje Z. En pocas muestras se ha podido conseguir una aproximación, aún sin ser como en un principio se deseaba. En general, una pierna en ese instante devuelve estos valores aproximadamente, en grados:

$$Euler_{Pierna\ Izquierda} = (rotZ, rotX, rotY) = (9.40, -75.04, 31.47)$$

Así, sin detectar la rotación del eje Z, es como si la pierna se quedara enfrente del usuario, un caso similar al del punto A de la figura 3-4. Esto puede ser debido por la misma razón que ocurre el problema de los brazos del experimento 2.

5) **Caminar:** Caminar es un movimiento común en los videojuegos. Simular este movimiento de forma natural no es sencillo ya que se requiere espacio, así que para este experimento solo se ha caminado unos tres pasos hacia la cámara. De esta forma, la cámara siempre detecta todas las articulaciones en todo momento (si nos alejamos mucho o si nos acercamos demasiado las articulaciones se pueden dejar de detectar). Si se quisiera usar este experimento para una animación real en un videojuego, solo se tendría que buscar en que momento se puede aplicar un *loop* para que la animación se repita constantemente. Mencionamos que en el experimento, el modelo se desplaza. Los animadores que deseen usar esta aplicación para hacer animaciones de caminar deberán quitar este desplazamiento para que el modelo permanezca en el mismo sitio mientras camina. Se decidió que si se el mismo usuario se grababa caminando en el sitio, es decir, caminar sin desplazarse, la animación no quedaría tan natural.

Para este experimento se han realizado un total de 10 muestras. Todas con resultados

similares y sin problemas, con apenas aparición de ruido.

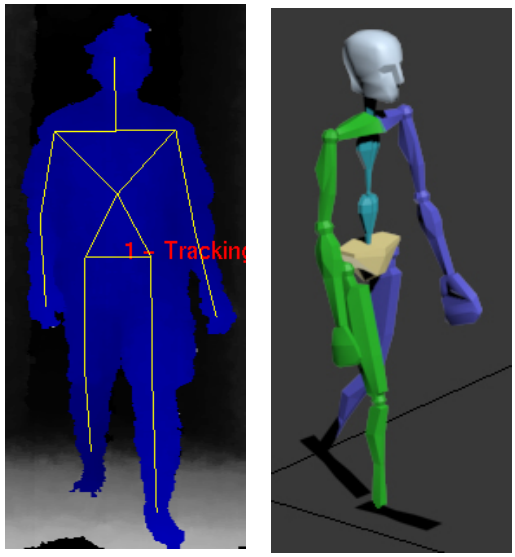


Figura 3-5. Un punto del experimento 5.

En la figura 3-5 se muestra un punto por el que pasa la animación. Se ha rotado la cámara que enfocaba al modelo de la imagen de la derecha para que se entendiera el movimiento, pero recordamos que el modelo camina hacia la cámara.

6) **Caminar y girar:** Este experimento es una expansión del anterior. Se trata de caminar pero haciendo el último paso en otra dirección. El experimento trata de que el usuario avance hacia la cámara y gire hacia la izquierda al final del trayecto. De nuevo, por temas de espacio y rango de la cámara, se ha caminado solamente de tres a cuatro pasos.

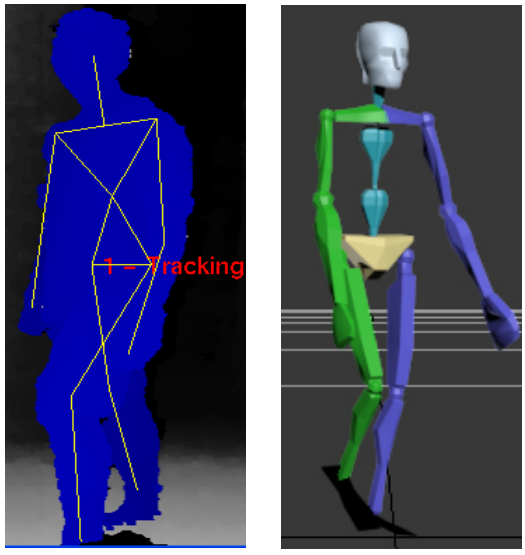


Figura 3-6. Un punto del experimento 6.

Para este experimento se han realizado un total de 10 muestras. No obstante, dado que al caminar de lado hay peligro de tapar articulaciones con el propio cuerpo (en este caso

sería el codo izquierdo), 4 de las muestras, en algún momento han calculado valores no numéricos, dejando inservible toda la muestra. También puede deberse a que al comenzar muy alejados, una de las articulaciones se haya dejado de detectar o simplemente porqué NITE en un momento dado ha perdido una articulación. En la figura 3-6 se muestra un punto por el que pasa la animación visto de frente.

7) **Correr:** Correr es otra acción muy vista en videojuegos. Similar a caminar, pero más complicado, pues ahora los brazos se mueven más. Para este experimento se ha realizado la acción de correr en dirección a la cámara, pero solo unos pocos pasos por falta de espacio.

   Se han hecho un total de 15 muestras para el experimento. Tres de ellas la acción de correr se ha hecho en sobre el mismo sitio, es decir, sin desplazarse, el resto de muestras se ha realizado la acción de correr con desplazamiento. Las que tienen desplazamiento, todas, tienen problemas para captar el movimiento de los brazos. Si bien ha habido una muestra con resultados muy buenos, las demás fallaban al captar los brazos, cada una en diferente grado. Se han hecho algunas de las muestras a menor velocidad, pero el resultado es similar. En la figura 3-7, la imagen de la izquierda nos muestra el esqueleto que NITE calcula en una de las muestras con desplazamiento. Podemos ver que el brazo izquierdo (recordamos que está en modo espejo) no sigue la nube de puntos del usuario. Esto se debe a que en algún momento se ha tapado alguna articulación y NITE supone donde podría estar o simplemente porque NITE ha calculado mal la articulación. Esto también podría ocurrirle a los huesos inferiores de las piernas, dado que se pueden dejar de ver cuando flexionamos las piernas. A parte, una de las muestras de este tipo ha generado un resultado erróneo y ha quedado inservible. La figura 3-7 muestra un punto por el que pasa una de estas muestras. La imagen de la derecha tiene la cámara girada para que se pueda ver mejor el movimiento desde otro ángulo, no obstante, el usuario corre en dirección a la cámara.

   Para las otras tres muestras que realizan la acción sin desplazamiento, el movimiento es capturado mucho mejor y los brazos se ven mejor. Aún así, sigue siendo posible que pasen los mismos problemas que corriendo con desplazamiento.

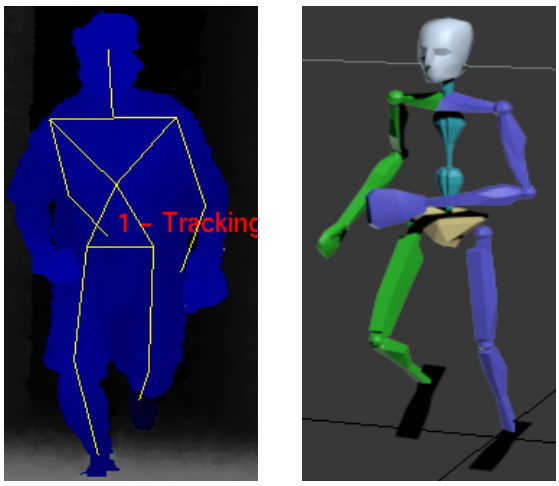


Figura 3-7. Un punto del experimento 7 en el que se corre con desplazamiento.

8) **Saltar:** Saltar es otra de las acciones prácticamente fundamentales en el mundo de los videojuegos. En este experimento se realiza un movimiento de salto sin desplazamiento, es decir, saltar recto y caer en el mismo sitio.

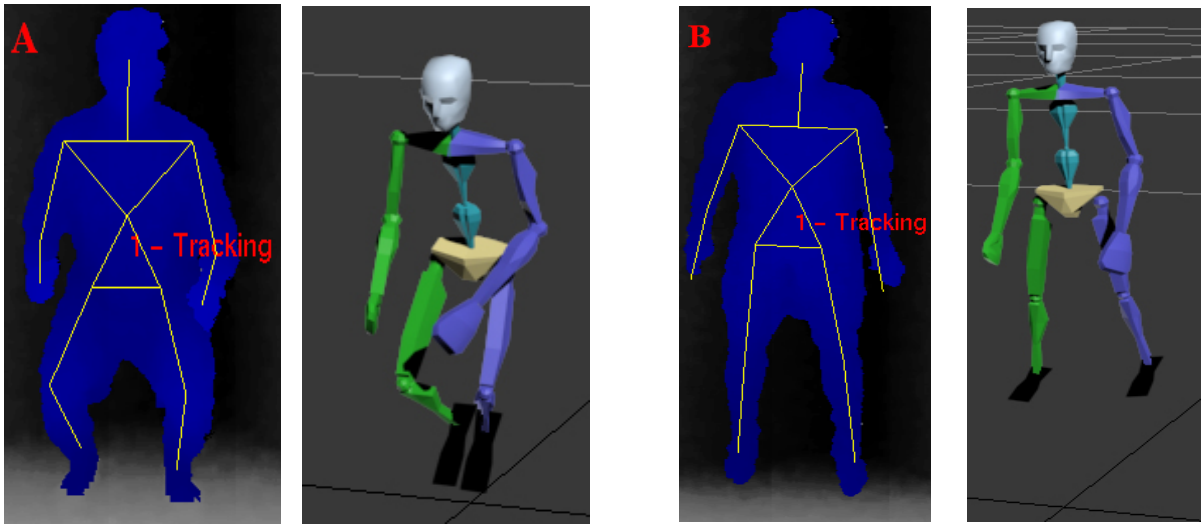


Figura 3-8. Algunos puntos del experimento 8.

Para este experimento se han realizado un total de 10 muestras. Todas ellas con resultados similares, los cuales se adaptan bastante bien al movimiento que se intentaba realizar y con muy poco ruido. El único inconveniente es que cuando se flexionan las piernas, los resultados tienden a hacer que los pies queden muy cerca, tal y como es el caso A de la figura 3-8. Esto puede deberse a que calculamos las rotaciones de las rodillas con el método de la ley de los cosenos. Al hacer los cálculos solo para un grado de libertad (en este caso, solo para el eje X) estamos ignorando el hecho de que la rodilla puede reajustar la pierna si siguiéramos el método tradicional de matrices y obteniendo así tres ángulos de rotación. Es cierto que las rodillas solo se mueven en el eje X, pero rotándolas con otros ejes podemos reajustar la pierna para que así se adapte mejor. Finalmente, se decidió por seguir con el método actual, el que usa la ley de los cosenos, por los problemas que pueden dar los ángulos de Euler. Cabe mencionar que esto mismo puede ocurrir en los codos.

9) **Salto lateral:** Este experimento es una expansión del anterior. El experimento consiste en saltar mirando al frente hacia la derecha y volver saltando de la misma forma.

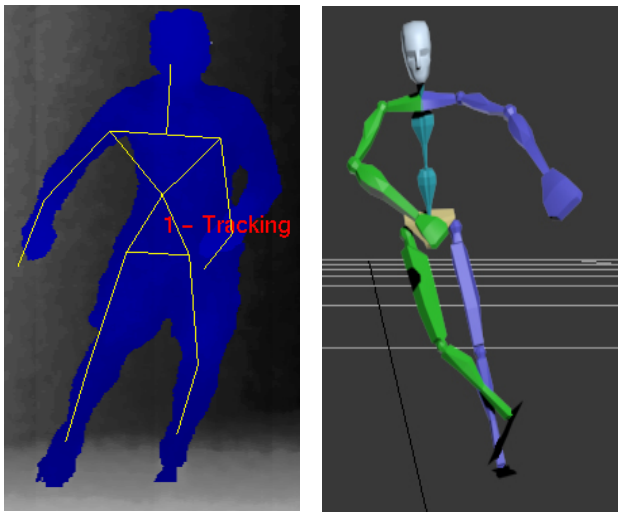


Figura 3-9. Un punto del experimento 9.

Para este experimento se han realizado 10 muestras. Todas las muestras dan resultados similares, con poco ruido y de un éxito similar al experimento 8 en el que solo se saltaba recto. Los pies tienden de nuevo a acercarse si se flexionan las piernas, por lo que el despegue y el aterrizaje siguen teniendo estos problemas. La figura 3-9 muestra un punto de una de las muestras. Tal y como muestra la figura, es posible que los brazos, debido a la velocidad al hacer el salto, queden un poco atrasados respecto a la nube de puntos del usuario.

10) **Saludar:** Saludar es una acción típica de los humanos cuando se encuentran por lo que suele verse en algunos videojuegos. Este experimento consiste en hacer un saludo sencillo con una mano. Cada muestra contiene el mismo saludo hecho con ambas manos por separado.

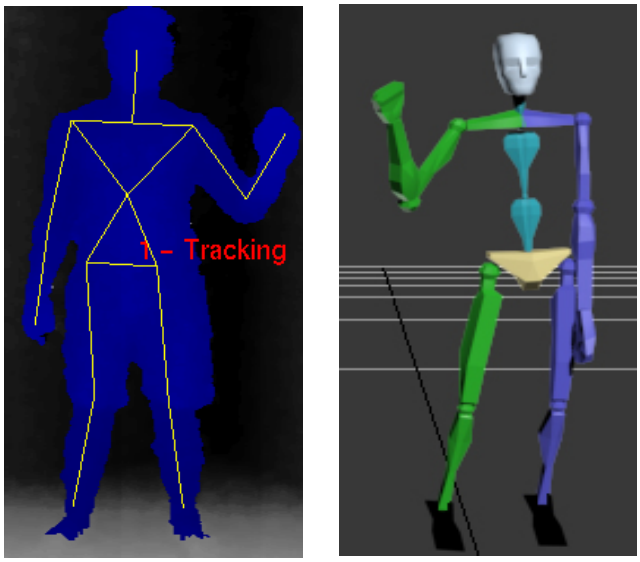


Figura 3-10. Un punto del experimento 10.

Este experimento conlleva 8 muestras. En general los resultados son buenos y con poco ruido, pero hay veces que los brazos no se flexionan en el ángulo correcto. Esto puede ser debido a que el hombro no llega a adaptar bien alguno de los ejes de rotación y al flexionar los codos se cree un movimiento incorrecto, puesto que los codos solo rotan en el eje X que el hombro les ha dejado (es lo que se comentaba en el experimento 8 con las rodillas). También puede deberse a que se ha alzado demasiado el brazo y entramos en una zona que da errores, como la que describe el caso F de la figura 3-2.

11) **Saludo oriental:** Este experimento es una variante del anterior, aunque el movimiento en sí es completamente distinto. Este saludo consiste en doblar el tronco para saludar respetuosamente.

Se han realizado un total de 7 muestras para este experimento. En general los resultados son similares. Hay problemas para detectar los brazos una vez quedan muy atrás. Esto puede ser debido a que al hacer la inclinación se tape alguna articulación de los brazos. Además, si se han ladeado las caderas o los hombros (NITE los mueve a la par, como si estuvieran conectados), la corrección que se aplicaba a las piernas para quitar la rotación de las caderas no parece suficiente y las piernas se ladean. Esto solo suele ocurrir si el torso se inclina hacia adelante y entonces rotamos otro eje.

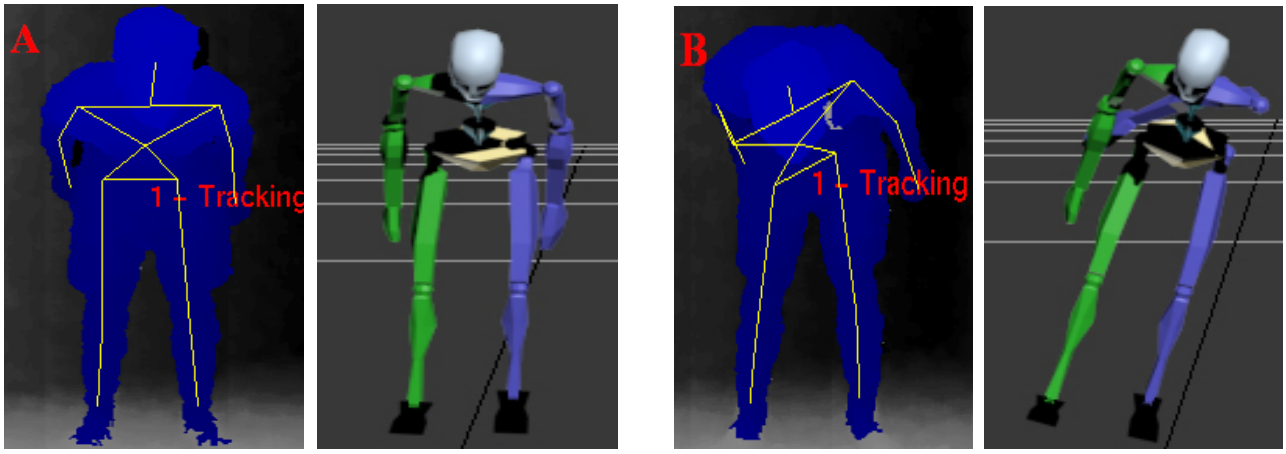


Figura 3-11. Algunos puntos del experimento 11.

La figura 3-11 muestra un caso que se calcula correctamente (A) y otro que no (B).

12) **Postura *Idle* de lucha:** La lucha es presente en un gran número de videojuegos. Este experimento trata de imitar la postura de lucha de un luchador sin que golpee. Es decir, el usuario permanecerá en una posición de defensa, como si estuviera a punto de entrar en una pelea.

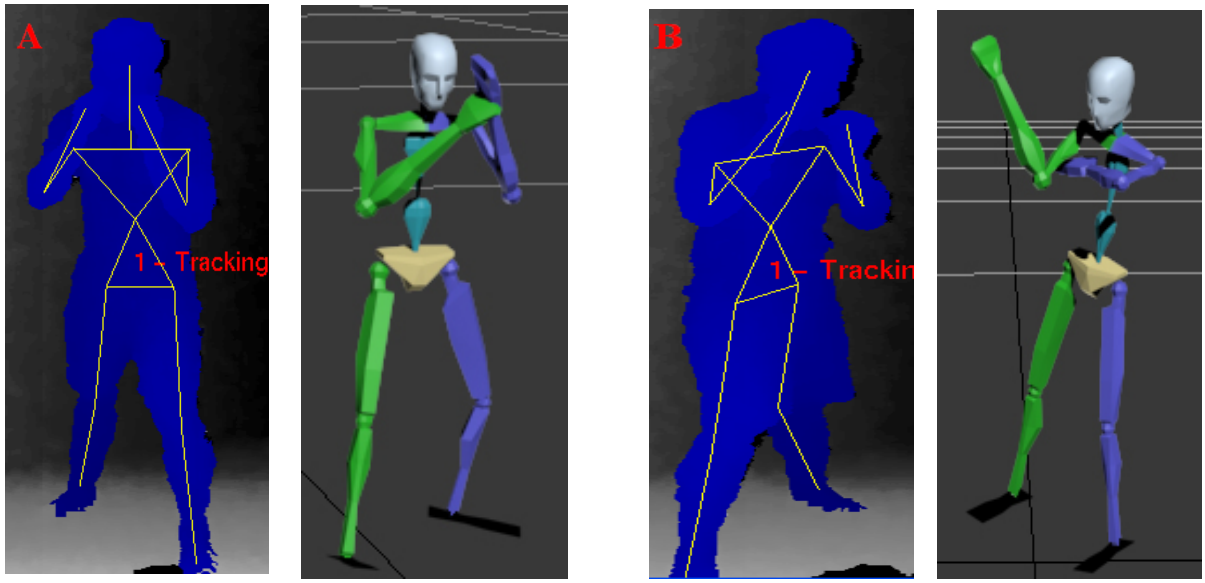


Figura 3-12. Puntos del experimento 12. El caso A muestra una postura zurda y el B una diestra.

Para este ejemplo se han hecho 12 muestras. Seis de ellas con la pierna izquierda delante y las otras seis con la pierna derecha delante. Para las muestras que tienen la pierna derecha delante, los resultados son aceptables. Si bien los brazos no se flexionan exactamente como deberían, los resultados son decentes. Las piernas presentan algo de ruido. Estas seis muestras se representan con la figura 3-12, caso A. Por otra parte, si es la pierna izquierda la que está delante, los cálculos son, por alguna razón, erróneos. Ambos brazos no encuentran los ángulos correctos y las piernas no acaban de situarse dónde deben. La figura 3-12, caso B, representa estas otras seis muestras.

13) **Puñetazo:** Si en un videojuego hay peleas, sin duda hay puñetazos. Para este experimento se realizan puñetazos con las dos manos, primero con una y luego con la otra. Se alternan hasta dos puñetazos por mano, uno se realiza más lento que el otro. Antes de hacer ningún puñetazo se adapta la postura de lucha frontal, es decir, con las dos piernas al mismo nivel, sin estar ninguna por delante de la otra.

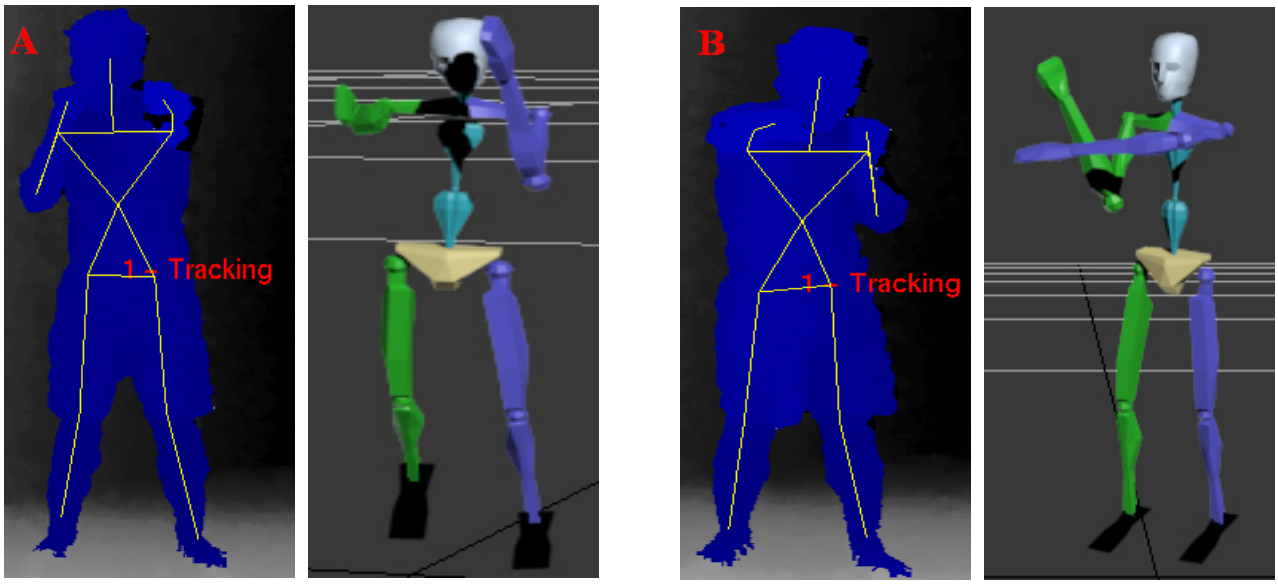


Figura 3-13. Puntos del experimento 13. El caso A es un puñetazo con la mano derecha y el B es un puñetazo con la mano izquierda.

Este experimento contiene 8 animaciones. Los resultados son similares y se consideran aceptables. Los brazos, en algunos frames, quedan un poco torcidos, sobretodo si golpea el brazo izquierdo. Tienen más ruido cuanto más rápido sea el puñetazo. La figura 3-13 muestra un puñetazo con cada mano.

14) **Patada:** Las patadas son otra parte del combate. Para este experimento se realizan patadas frontales, una con cada pierna. Esta vez, se hacen movimientos suaves para evitar el ruido, en vez de hacerlos demasiado rápidos.

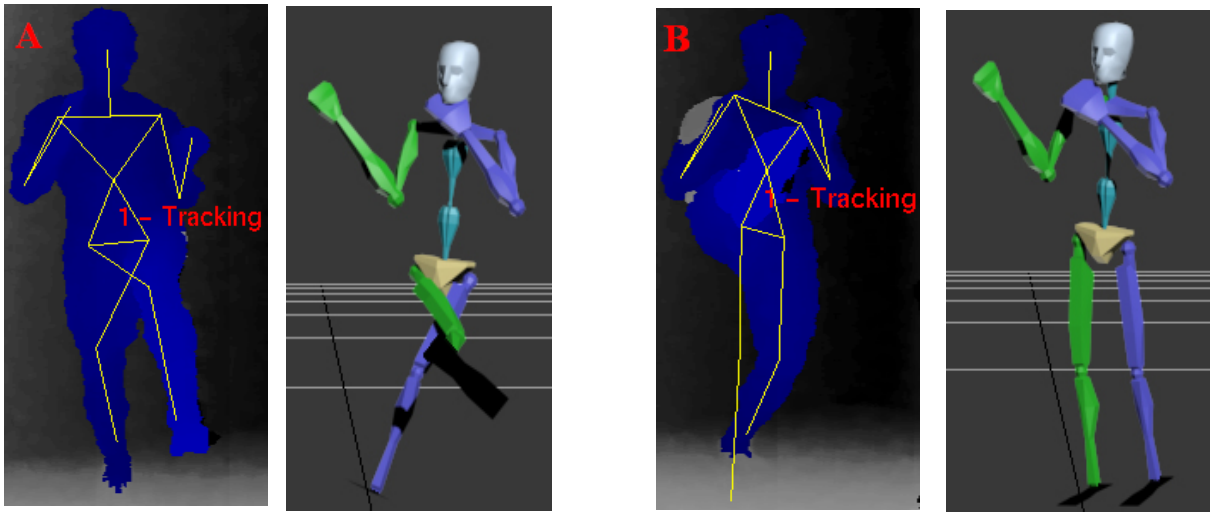


Figura 3-14. Casos problemáticos del experimento 14.

Como en el experimento anterior, al dar patadas el usuario adopta una posición de lucha frontal. Una patada frontal sirve para derribar objetos como puertas o apartar a enemigos que tengamos delante.

Se han realizado un total de 8 animaciones para este experimento. Los resultados son considerados aceptables, no obstante, NITE tiene problemas para calcular la posición de las piernas cuando la patada golpea. En la figura 3-14 se ven algunos casos en los que las piernas se cruzan, como el caso A, u otros en los que simplemente la pierna deja de detectarse y NITE presupone que no está en el aire, como el caso B. Esto puede pasarle también a los brazos si la pierna sube mucho o el movimiento es brusco.

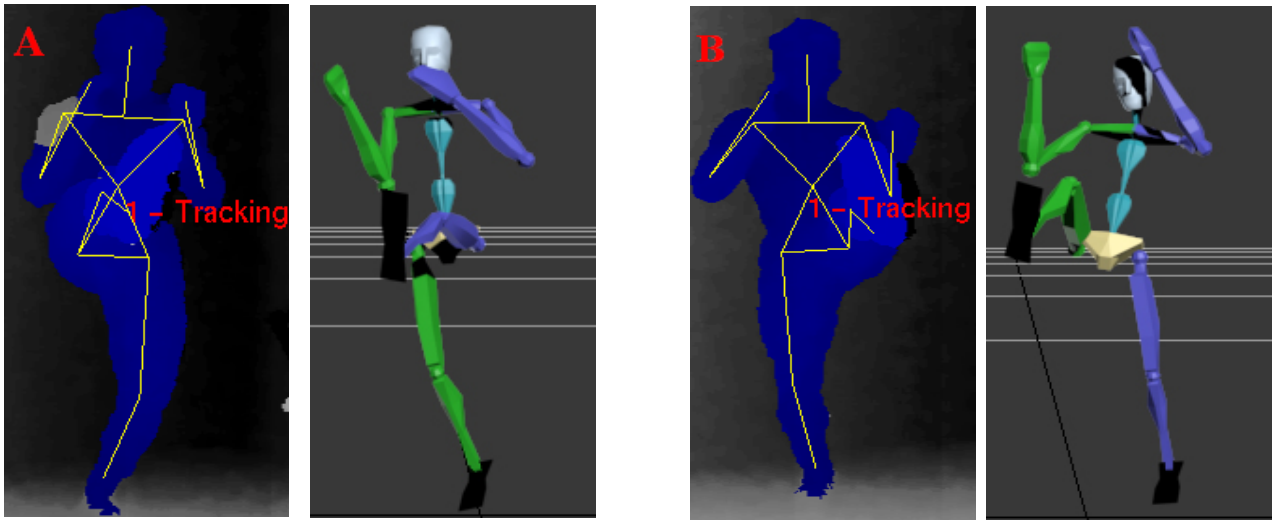


Figura 3-15. Algunos puntos del experimento 14. El caso A es una patada con la pierna izquierda y el caso B con la pierna derecha.

La figura 3-15 muestra casos en los que si que se ha detectado bien cada patada.

3.3 Incidencias

A lo largo de la programación del código, se han encontrado diversas complicaciones, obligando a tomar medidas para solucionarlas, cambiar el enfoque para evitarlas o arreglarlas de forma que el error que provoquen sea mínimo.

Según su tipo, se separan en dos grupos:

- Incidencias encontradas en el módulo de la aplicación principal.
- Incidencias encontradas en el módulo correspondiente a 3dsMAX.

a) Incidencias encontradas en el módulo de la aplicación principal.

- **Compatibilidad de las posturas iniciales entre NITE y BVH:** La razón principal por la que no se utilizaron las matrices de orientación que NITE proporciona es porqué, además de que calculando nosotros mismos las matrices tenemos mayor control sobre los datos, las posturas iniciales no coinciden. Esto quiere decir que no todas las matrices, cuando equivalen a la identidad, están en la misma postura.

En la figura 3-16 se encuentra lo que equivaldría a la figura 2-9 en NITE.

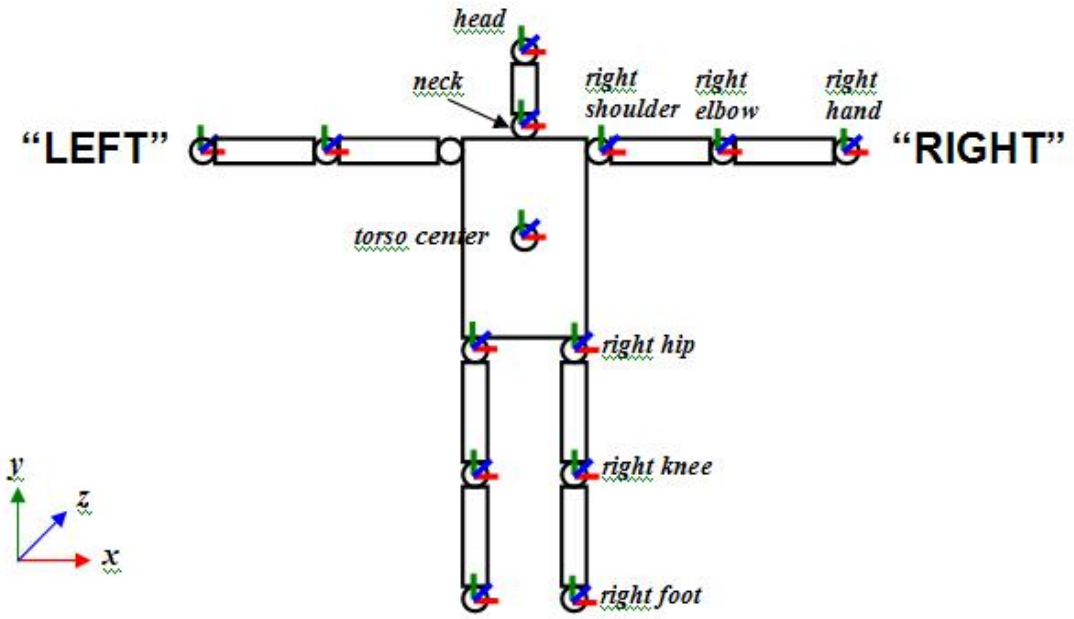


Figura 3-16. Esqueleto con orientaciones según NITE. La figura muestra la parte frontal del esqueleto. Los codos se doblan de tal forma que las manos se acercarían a la cámara.

La figura 3-16 es sacada de la documentación de NITE. Debido a que las posiciones iniciales no son iguales en los brazos, si se utilizan las matrices que calcula NITE hay que aplicar un ajuste de 90º a los hombros (*Left* y *Right Shoulder*) para que concuerden con el formato BVH. Aunque esa solución sea poco costosa, se optó por calcularlo a mano de todas formas, siendo la figura 2-9 la postura inicial escogida que mejor se adapta al formato BVH.

- **Matrices para pies y manos:** Calcular la posición de las manos y de los pies no es posible usando únicamente la posición de articulaciones debido a que NITE no calcula nada relevante a las muñecas y los tobillos, además, las articulaciones que equivalen a los pies y las manos que podemos ver en la figura 3-16 no calculan matrices de rotación, NITE simplemente nos da matrices que hagan que las manos y los pies estén alineados con las rodillas y los codos respectivamente. Si fuera el caso contrario y tuviéramos información sobre la posición de muñecas y tobillos, podríamos usar el vector del tobillo a la punta del pie como vector Z y usar el vector que va de la rodilla al tobillo como auxiliar para hacer el producto vectorial con el vector Z y obtener así el vector X. Haciendo el producto vectorial de los vectores Z y X obtendríamos el vector Y. Para las manos sería muy similar, aunque se tendría que vigilar que los dos huesos usados para obtener los vectores no fueran colineales.

  Usar el KinectSDK es una solución, aunque conlleva programar de nuevo todo en otro entorno. Una idea para solucionar esto con NITE es intentar deducir hasta dónde llega la mano desde una posición que representara la muñeca, todo mediante la nube de puntos que define a un usuario. La nube de puntos de un usuario es la información que Kinect nos da para decirnos qué píxeles (con su profundidad) son los que pertenecen a un usuario dentro del ángulo de visión de la cámara.

- **Matrices para el cuello y la cabeza:** La matriz de la cabeza requiere saber la posición de la nariz o reconocer la cara del usuario. Debido a la complejidad del problema se ha omitido resolverlo. Una solución sería añadir algún algoritmo que detecte en tiempo real la cara del usuario, aunque esto podría ser costoso. La nueva versión 1.5 del SDK de Kinect es capaz de realizar *motion capture* de la cara, por lo que podría ser otra solución para el problema.

  El cuello, en cambio, es posible calcularlo sin la ayuda de NITE. Ya se explicó en el 2.1.2, sección b), cómo calcular la matriz del cuello. No obstante, calcularlo de dicha manera tiene un coste: perder la rotación del vector Y. Dado que no sabemos dónde está la nariz del usuario (o la cara en sí), es imposible saber con tan solo los huesos en qué dirección está mirando el usuario.

- **Problemas con los ángulos de Euler:** Los ángulos de Euler presentan dos problemas que se han visto en los hombros. Por un lado tenemos el *Gimbal Lock* [GLo-03], una singularidad que ocurre cuando dos de los ejes están en configuración paralela, lo cual hace perder un grado de libertad y provoca que no se pueda encontrar una rotación adecuada.

  El segundo problema es que los ángulos de Euler son ambiguos, pues una sola rotación podría definirse de más de una forma.

  En un principio, el proyecto seguía el método de conversión de matriz a ángulos de Euler del Anexo III, pero cuando los hombros rotaban alrededor de 90º sobre el eje Z, los cálculos daban resultados erróneos. Se optó entonces por usar la librería *Configurable Math Library* (CML) y se observó una mejora en los cálculos. No obstante, siguen existiendo algunas zonas críticas que, debido a los problemas de Euler, causan errores en los cálculos de los ángulos. Una zona crítica es un rango de rotaciones en las que el cálculo de los ángulos devuelve valores erróneos debido a que, bien por su ambigüedad interpreta los ángulos de otra manera o bien genera errores a causa del *Gimbal Lock*. Estas zonas críticas son principalmente un problema en los hombros dado que son las articulaciones que permiten un mayor rango de rotación.

- **Problemas con los cálculos de NITE:** NITE presenta algunos errores de cálculo en la detección del usuario. Poco hemos podido hacer contra estos problemas y los hemos tomado como si fueran limitaciones por usar NITE. Los errores son los siguientes:

  - La detección de los brazos es menos estable cuando se encuentran cerca de otras partes del cuerpo, especialmente el torso. Si ambos brazos están cerca del torso o de ellos mismos, pueden mezclarse.
  - La detección de las piernas es algo inestable y ruidosa. Funciona mejor si el usuario se mantiene en pié con las piernas algo separadas. Los movimientos rápidos y complejos como patadas pueden causar que la detección falle.
  - La detección de la posición puede ser inestable si la cabeza no es visible.
  - Piernas y brazos posicionados en los límites de la flexibilidad humana pueden hacer que se pierda la detección.
  - En general, movimientos rápidos pueden provocar fallo en la detección.
  - Hay posibilidades de que se dé el caso de que la detección en general sea mala.

- Presencia de ruido en la detección del esqueleto.
- No se distingue la parte frontal de la trasera del usuario, por lo que si el usuario da media vuelta no se captará.
- Una vez se ha calibrado el usuario, si rotamos únicamente el cuello, éste moverá también toda la columna, tal y como lo muestra la figura 3-17.

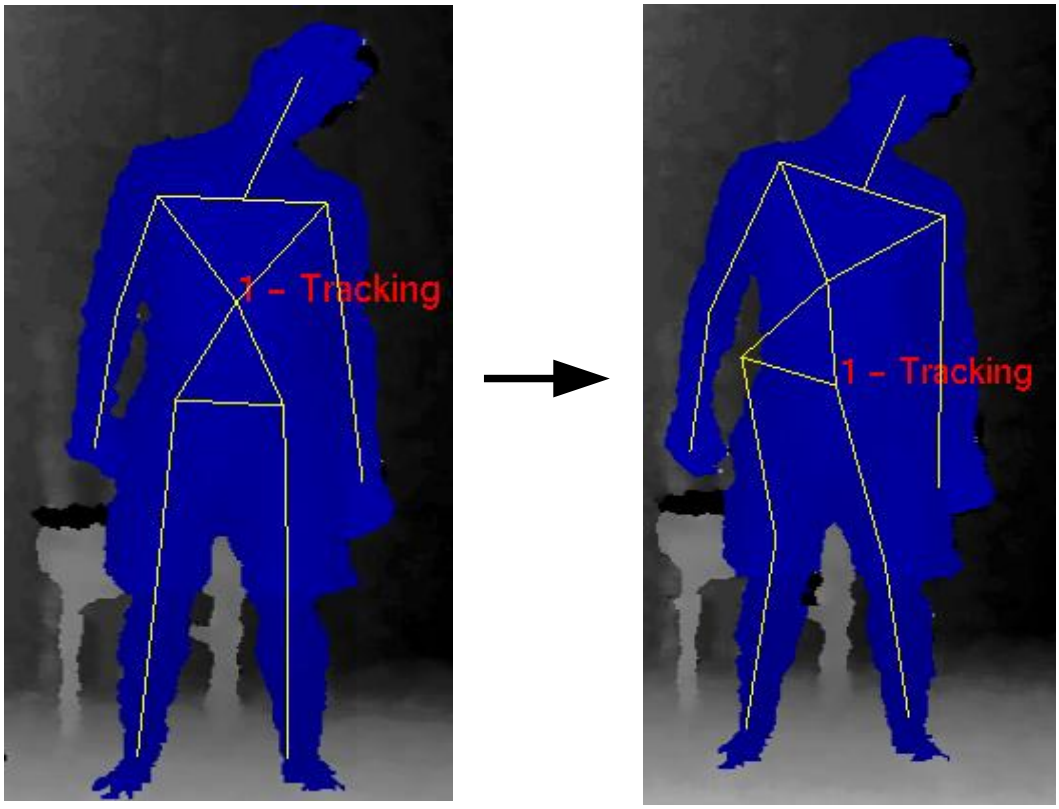


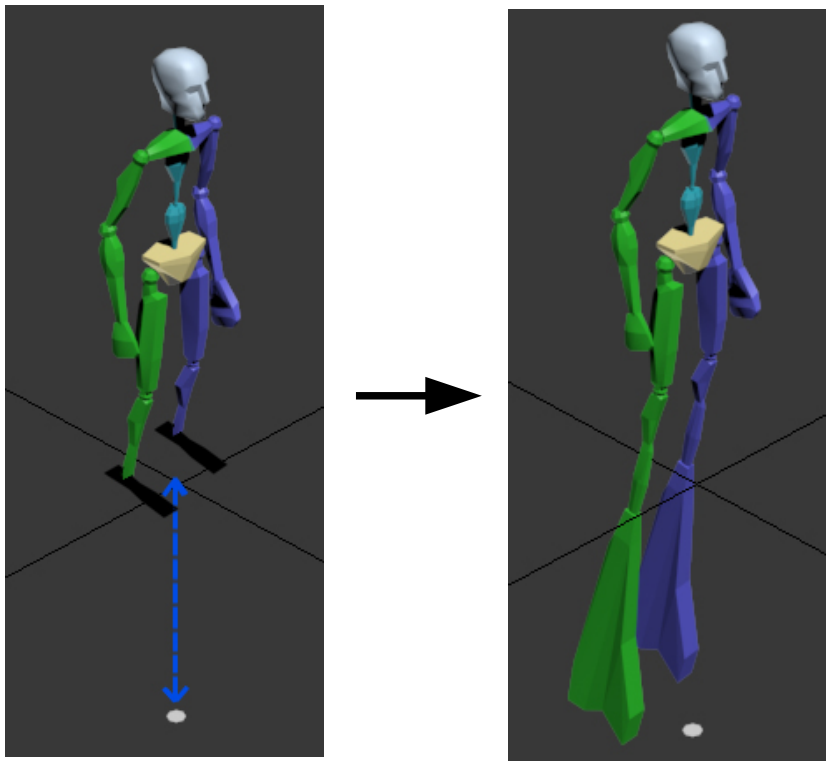
Figura 3-17. a) Sin calibrar del todo, el cuello es independiente. b) Al calibrar por completo, el cuello mueve toda la columna.

b) Sobre BVH en 3dsMAX

- **Compatibilidad de esqueletos entre el formato BVH y el modelo:** El modelo que se nos proporcionó tenía un esqueleto predefinido con más vértebras que el que usamos en el proyecto. Al agregar el BVH al modelo, el esqueleto de éste sufre una transformación, perdiendo articulaciones (las vértebras de más). Esto puede generar que algunos vértices queden sueltos. Aumentando el peso de los vértices a 1.0 se conseguirá que no sobresalgan del modelo. De cualquier manera, el modelo se deformará ligeramente y no quedará igual que el original. Además, la textura, como el modelo ha cambiado, no encaja y hace que se vea mal. Como solución se ha usado la textura de las normales para que solo se vea la forma del modelo. Se pensó en crear una textura que fuera monocolor, pero se veía muy simple por pantalla y sin crear una sensación de 3D.
- **Deformación del esqueleto del BVH:** Cuando cargamos los archivos .BVH en un bípedo,

su esqueleto sufre una transformación cuando pasa cierto tiempo. Los pies se alargan de manera exagerada. Esto no provoca ningún problema para exportar las animaciones en Cal3D.

Es posible que se deba a que el bípedo requiere un tamaño mínimo. Los pies acaban creciendo hasta la elipse que representa la "sombra" del esqueleto que toca el suelo. La figura 3-18 muestra el problema.



(a) (b)

Figura 3-18. Esqueleto cargado con un BVH en 3dsMAX. Pasar de a) a b) ocurre al cabo de un determinado tiempo, aunque nunca si la animación está en 'play'. No se puede volver de b) a a) a no ser que se vuelva a cargar el BVH.

4. CONCLUSIONES Y MEJORAS

El objetivo del proyecto era capturar movimientos de un usuario con Kinect y convertirlos en animaciones Cal3D usando el software 3dsMAX. Se ha logrado cumplir este objetivo usando OpenNI junto a NITE para tratar la información que Kinect percibe. Con el formato BVH hemos podido guardar en ficheros los datos del movimiento que NITE nos proporcionaba y con 3dsMAX se han aplicado los movimientos a un modelo, pudiéndolo exportar todo como animaciones en formato Cal3D.

No obstante, el proyecto tiene los siguientes problemas sin resolver:

- Hay algunos fallos de cálculo en la captura de movimientos debido a los ángulos de Euler.
- Faltan calcular las rotaciones de pies y manos.

Kinect da buenos resultados y NITE es un buen medio para usarlos. Sin embargo, utilizar ángulos de Euler para describir las rotaciones es un método complicado que crea demasiadas ambigüedades. Probablemente sea buena idea tratar de mirar otro formato que pueda utilizarse en 3dsMAX y que utilice cuaterniones, los cuales no tienen los problemas que los ángulos de Euler generan. Usar Kinect SDK en vez de NITE puede dar mejores resultados ahora que hay una versión que salió a la luz hace poco, capaz incluso de calcular las rotaciones de pies y manos.

En general, a pesar de la dificultad y problemas que han añadido las matemáticas, el proyecto ha sido algo interesante en lo que trabajar y que aún se puede mejorar más. Como mejoras para el proyecto se listan las siguientes:

- Encontrar una solución a los problemas que generan los ángulos de Euler o bien usar otro formato que utilice cuaterniones.
- Usar métodos para suavizar las animaciones y eliminar ruido.
- Ampliar la percepción de la cámara haciendo que el proyecto sea compatible con dos Kinects, de esa forma el usuario puede seguir siendo captado aunque dé media vuelta.
- Hacer que el proyecto pueda captar a más de un usuario.
- Calcular expresiones faciales. Podría hacerse con la nueva versión del Kinect SDK.

5. BIBLIOGRAFÍA Y REFERENCIAS

[ADk-82] http://usa.autodesk.com/ Página web oficial de *Autodesk*, dónde podemos encontrar información sobre *3D Studio MAX, Motion Builder* y *Maya*, entre otros softwares. Último acceso: agosto de 2012.

[AMe-10] *"Understanding Motion Capture for Computer Animation".* Segunda edición. Autor: Alberto Menache. Editorial *Elsevier.* Un libro para entender mejor el concepto *motion capture*.

[BBC-11] http://www.youtube.com/watch?v=nYsqNnDA1l4&feature=related Vídeo sobre un proyecto de detección de movimiento facial con Kinect. Sus autores son M.Breidt, H. H. Bülthoff y C. Curio. Último acceso: agosto de 2012.

[Bre-12] http://www.brekel.com/ Página web de *Brekel Kinect*, creada por Jasper Brekelmans. Último acceso: agosto de 2012.

[Cal-06] http://home.gna.org/cal3d/ Página web oficial de Cal3D. Último acceso: agosto de 2012.

[ChS-00] http://www.character-studio.net/

http://www.character-studio.net/bvh_file_specification.htm (Formato BVH)

http://www.character-studio.net/csm_file_specification.htm (Formato CSM)

Guía online sobre el *Character Studio* que incluye las especificaciones sobre los formatos de *motion capture* BVH y CSM. Último acceso: agosto de 2012.

[DiX-98] http://msdn.microsoft.com/en-us/directx/ Página web oficial de DirectX SDK. Último acceso: agosto de 2012.

[Dot-02] http://en.wikipedia.org/wiki/Dot_product Artículo de Wikipedia que explica el producto escalar, o *dot product*. Último acceso: agosto de 2012.

[Eul-03] http://en.wikipedia.org/wiki/Euler_angles#Gimbal_analogy Artículo de Wikipedia que explica los ángulos de Euler. Último acceso: agosto de 2012.

[EWW-95] http://mathworld.wolfram.com/CrossProduct.html Página web matemática de dónde se sacó información para entender el producto vectorial. Último acceso: agosto de 2012.

[GLo-03] http://en.wikipedia.org/wiki/Gimbal_lock Artículo de Wikipedia que explica el *Gimbal Lock*. Último acceso: agosto de 2012.

[GMo-04] http://garrysmod.com/ Página web de *Garry's Mod.* Último acceso: agosto de 2012.

[HUD-11] http://thwackers.tv/ Página web del grupo encargado de la serie web animada *Under the HUD*. Último acceso: agosto de 2012.

[IPi-09] http://www.ipisoft.com/ Página web de *IPiSoft*. Último acceso: agosto de 2012.

[JBo-11] http://engineeringblog.yelp.com/2011/03/after-hours-project-kinect-hacking.html Página web sobre el proyecto de control de personajes de John Boiles. Último acceso: agosto de 2012.

[Kin-11] http://www.microsoft.com/en-us/kinectforwindows/ Página web oficial sobre Kinect SDK. Último acceso: agosto de 2012.

[LoC-02] http://en.wikipedia.org/wiki/Law_of_cosines Artículo de Wikipedia que explica la Ley de los Cosenos. Último acceso: agosto de 2012.

[Mer-11] http://www.meristation.com/es/xbox-360/noticias/una-nueva-forma-de-utilizar-kinect/1673656 Artículo de *Meristation* sobre usos de Kinect fuera del ámbito de videojuegos. Último acceso: agosto de 2012.

[MJB-98] http://www.euclideanspace.com/maths/geometry/rotations/index.htm Página web matemática que, entre otras cosas, profundiza en temas de rotación. Último acceso: agosto de 2012.

[MKi-09] http://es.wikipedia.org/wiki/Kinect Artículo de Wikipedia que habla sobre información general de Kinect. Último acceso: agosto de 2012.

[MyM-05] http://www.dcs.shef.ac.uk/intranet/research/resmes/CS0111.pdf Documento PDF que explica algunos formatos de *motion capture*, entre ellos, el BVH. Último acceso: agosto de 2012.

[NyA-03] http://cmldev.net/ Página web oficial de *Configurable Math Library* por Demian Nave y Jesse Anders. Último acceso: agosto de 2012.

[OGL-92] www.opengl.org/ Página web oficial de *OpenGL*. Último acceso: agosto de 2012.

[ONI-10] http://openni.org Página web oficial de *OpenNI*. Último acceso: agosto de 2012.

[PrS-05] http://www.primesense.com Página web oficial de *PrimeSense*. Último acceso: agosto de 2012.

[SEn-96] http://source.valvesoftware.com/ Página web de *Source Engine*. Un motor de videojuegos de *Valve*. Último acceso: agosto de 2012.

ANEXO I: El formato BVH

En este anexo se explica detalladamente el formato BVH (*BioVision Hierarchical data).* BVH es un formato para guardar la información realizada en *motion capture* y se compone de dos partes: la jerarquía y el movimiento.

i) Jerarquía (Hierarchy)

La Jerarquía define como está compuesto el esqueleto de nuestro modelo. Esto significa que describe cuantos huesos tiene, de quien es el padre de cada uno y sus longitudes. Un hueso se define como la distancia entre dos articulaciones.

La parte jerárquica del formato BVH sigue la estructura definida en la figura I-1.

```
HIERARCHY
ROOT [jointName]
{
    OFFSET [float float float]
    CHANNELS [int] [channel types]
    JOINT [jointName]
    {
        OFFSET [float float float]
        CHANNELS [int] [channel types]
        JOINT [jointName]
        {
            OFFSET [float float float]
            CHANNELS [int] [channel types]
            ...
            End Site
            {
                OFFSET [float float float]
            }
        }
    }
    ...
}
```

Figura I-1. Estructura de la Jerarquía de un BVH

Empieza con la palabra clave HIERARCHY, indicando el inicio de la definición de la jerarquía. A partir de aquí se añaden las articulaciones. La primera articulación será la raíz, indicado con la palabra clave ROOT, el resto de articulaciones serán indicadas con la palabra clave JOINT. Ambos tipos de articulación continúan con el nombre de dicha articulación. Los nombres para las articulaciones deben escribirse tal cual muestra la figura 2-3, la cual muestra la jerarquía que se ha definido para el proyecto, indicando los hijos de cada articulación. BVH soporta más articulaciones, pero con las indicadas en la figura 2-3 serán más que suficientes para cumplir los objetivos propuestos.

Después de indicar el nombre, se abren llaves. Dentro se encontrará todo lo relevante a esa articulación, en concreto tres cosas:

1) **OFFSET**: El OFFSET indica la distancia que una articulación está alejada de su padre en cada uno de los ejes (x, y, z). Por ejemplo, en el caso del pecho (*Chest*), la distancia que tendríamos que poner en el OFFSET vendría dada del resto de posiciones entre el pecho y la raíz (*Hips*). En sí, un OFFSET es un vector creado a partir del resto de otros dos.

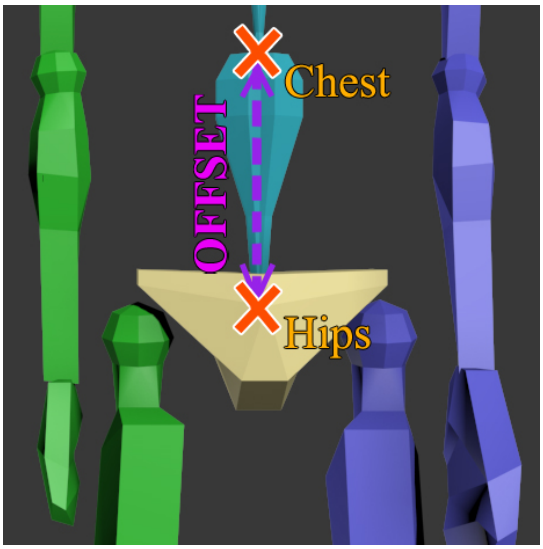


Figura I-2. Ejemplo del OFFSET de la articulación 'Chest'.

La figura I-2 muestra el ejemplo del OFFSET para la articulación del pecho. Las articulaciones tienen sus posiciones representadas como un vector de tres dimensiones (x, y, z) tal que:

$$Posición\_Articulación_i = (x_i,\ y_i,\ z_i)$$

Entonces, calcular la distancia para el OFFSET del pecho equivale a:

$$OFFSET_{Chest} = Posición\_Articulación_{Chest} - Posición\_Articulación_{Hips}$$

$$OFFSET_{Chest} = (x_{Chest} - x_{Hips},\ y_{Chest} - y_{Hips},\ z_{Chest} - z_{Hips})$$

En el caso de la raíz, el OFFSET sólo indica la posición inicial en el espacio 3D en el que las caderas se encuentran. Para el proyecto se ha decidido que empiecen en las coordenadas (0,0,0).

Es importante recordar que el orden de los floats es x, y, z. Cada cifra indica la distancia en el eje correspondiente.

2) **CHANNELS**: Los canales son el tipo de información que una articulación va a recibir. En concreto, solo hay dos tipos de canales, los de posición y los de rotación. Tal y como indican sus nombres, el de posición sirve para mover la articulación por coordenadas y el de rotación sirve para entrarle las rotaciones que una articulación tiene que hacer. Un detalle importante, BVH usa datos locales. Esto quiere decir que si rotamos una articulación, sus hijos se verán modificados también. Lo mismo pasa si se hacen traslaciones.

Primero se indica cuantos canales tiene la articulación y luego se especifica cuales son. En general, todas las articulaciones van a tener los tres canales de rotación, uno para cada eje (x, y, z). Siguiendo los ejemplos vistos por Internet, se decidió seguir el orden z, x, y. Un canal de rotación se escribe como *Zrotation, Xrotation* e *Yrotation*.

La raíz es la única articulación que tendrá canales de posición. Si bien podríamos usar seis canales en todas las articulaciones, parece algo innecesario, dado que si se mueve la raíz, toda la estructura la seguirá, por lo que ya nos basta y nos ahorramos algunos cálculos por frame al evitar recolectar las posiciones de todas las articulaciones. Los canales de posición siguen el orden x, y, z y se escriben como *Xposition*, *Yposition* y *Zposition.*

3) **Terminación**: La última parte de una articulación puede ser de dos formas. O bien la articulación tiene una o más articulaciones hijo o bien es una extremidad y no tiene más hijos. Para el primer caso, por cada articulación hijo se vuelve a empezar. Siguiendo las indicaciones anteriormente establecidas, se comienza escribiendo JOINT e indicando el nombre de la articulación. Se abren llaves y se vuelven a poner las tres partes de una articulación.

   Si la articulación no tiene hijos, caso de las manos, los pies y la cabeza, se tiene que indicar cuanto se alarga el hueso. En otras palabras, cuán lejos se encuentra la terminación de la extremidad de la articulación en sí. Por ejemplo, en el caso de una mano, sería la distancia entre la punta del dedo corazón y la muñeca. Para indicar esta información se usa la palabra clave END SITE, se abren llaves de nuevo y dentro se añade únicamente un OFFSET para indicar la distancia en cada eje.

La parte jerárquica parece complicada. No obstante, cuando se comprende cuál es el concepto que el formato tiene para el esqueleto, las distancias exigidas por éste y la terminología, resulta factible de construir.

Para ver un ejemplo mas simple que el de la figura 2-3, la figura I-3 muestra un esqueleto más simple. Vienen señalizadas las distancias entre las articulaciones. Las distancias entre paréntesis representan la distancia desde la articulación hasta su terminación (el END SITE). Por ejemplo, del hombro izquierdo a la punta de la mano hay 0,60 metros.

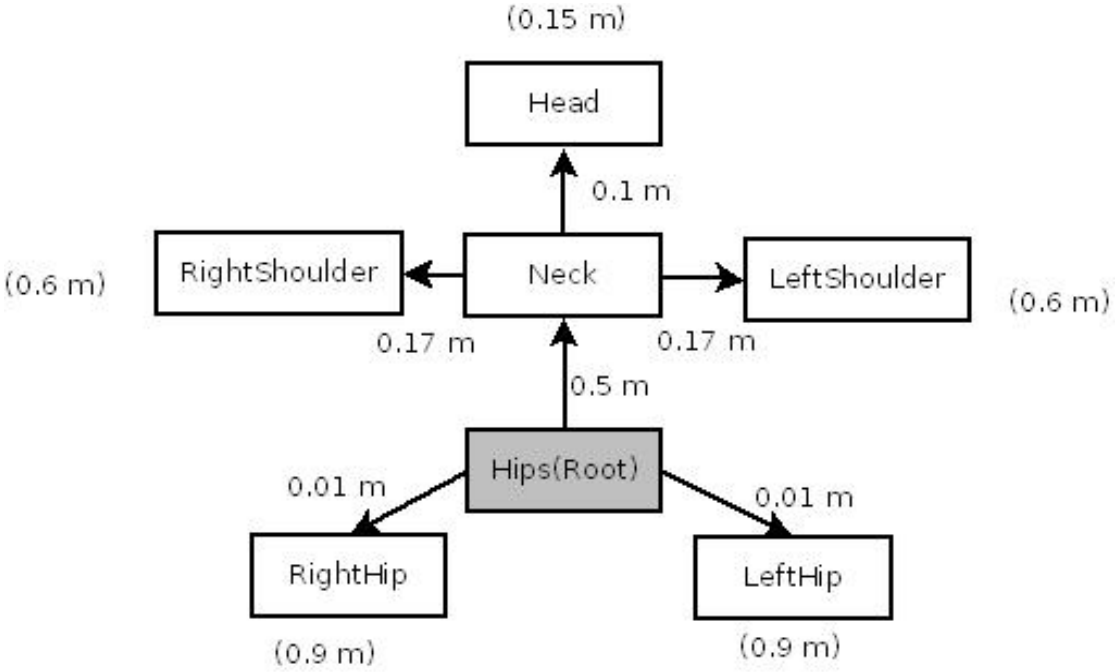


Figura I-3. Diagrama de la jerarquía de un esqueleto simple

El código que hay que escribir para hacer el esqueleto sería, siguiendo la figura I-1, el

siguiente:

```
HIERARCHY
ROOT Hips
{
	OFFSET 0 0 0
	CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
	JOINT LeftHip
	{
		OFFSET 0.01 0.0 0.0
		CHANNELS 3 Zrotation Xrotation Yrotation
		End Site
		{
			OFFSET 0.0 -0.9 0.0
		}
	}
	JOINT RightHip
	{
		OFFSET -0.01 0.0 0.0
		CHANNELS 3 Zrotation Xrotation Yrotation
		End Site
		{
			OFFSET 0.0 -0.9 0.0
		}
	}
	JOINT Neck
	{
		OFFSET 0.0 0.5 0.0
		CHANNELS 3 Zrotation Xrotation Yrotation
		JOINT LeftShoulder
		{
			OFFSET 0.16 0.0 0.0
			CHANNELS 3 Zrotation Xrotation Yrotation
			End Site
			{
				OFFSET 0.0 -0.6 0.0
			}
		}
		JOINT RightShoulder
		{
			OFFSET -0.16 0.0 0.0
			CHANNELS 3 Zrotation Xrotation Yrotation
			End Site
			{
				OFFSET 0.0 -0.6 0.0
			}
		}
		JOINT Head
		{
			OFFSET 0.0 0.1 0.0
			CHANNELS 3 Zrotation Xrotation Yrotation
			End Site
			{
				OFFSET 0.0 0.15 0.0
			}
		}
	}
}
```

Un aspecto importante es que este esqueleto tiene como única función mostrar un ejemplo simple. No se puede utilizar en un caso real dado de que el formato BVH tiene ciertas restricciones en cuanto a cómo debe de ser la jerarquía de un esqueleto y no se puede simplificar tanto como se ha hecho en la figura I-3.

ii) Movimiento (Motion)

Aquí se definen los frames totales, el tiempo por cada frame y la información que debemos pasarle a los canales definidos en el apartado i) en cada frame. Seguiremos el esquema de la figura I-4 para explicar la parte del movimiento.

```
1 MOTION
2 Frames: [int]
3 Frame Time: [float]
4 [floats for Root's channels] [floats for Root's first son's channels] ...
5 ...
```

Figura I-4. Estructura del movimiento de un BVH

Se comienza esta parte escribiendo la palabra clave MOTION. A continuación se indican dos valores precedidos por las palabras clave *Frames* y *Frame Time*. La primera indica el número total de frames que tendrá la animación y la segunda el tiempo de separación, en segundos, de un frame entre otro.

A continuación prosigue la información que le vamos a pasar a las articulaciones según sus canales. Cada línea representa un frame. Se tiene que seguir el orden establecido en la parte jerárquica como si se leyera de arriba a abajo, esto quiere decir que si se abren llaves para una articulación hijo, ésta será la siguiente para introducir los valores *float* en vez de la hermana de la articulación padre. A fin de cuentas, el orden simplemente es como si se leyera de un libro, cada articulación que se nombre se añade a la cola. Recordemos que, para el caso de nuestro proyecto, a la raíz le corresponderán los seis primeros valores mientras que el resto de articulaciones sólo tendrán tres dado que la raíz tiene seis canales (tres de posición y tres de rotación) mientras que el resto de articulaciones solo tendrán tres canales de posición. Además, hay que poner estos valores según el orden que se especificó en la sección de jerarquía para los canales de cada articulación. Por ejemplo, para el caso del proyecto, los seis primeros valores corresponderán a las coordenadas de la raíz (serían los tres primeros), siguiendo el orden X, Y y Z y a los ángulos de rotación (tres últimos), siguiendo el orden Z, X e Y.

Si hemos recolectado valores para menos frames de los indicados en el dato *Frames*, la animación se forzará a la posición inicial y se quedará inmóvil. También hay que tener cuidado de no dejarnos ningún canal sin rellenar en cada frame, o el sistema cogería el que correspondería al siguiente canal, ya sea o no del siguiente frame, desfigurando toda la animación.

Como anotación general para ambas partes, el formato BVH usa las distancias en metros y las rotaciones en grados. Las articulaciones traspasan sus modificaciones a sus hijos, por lo que cada articulación precisa de valores locales en vez de globales.

En la figura I-5 se muestra la estructura de un fichero BVH con el esqueleto de la figura I-3. Solo tiene dos frames en los que se hacen las mismas rotaciones, por lo que el esqueleto se vería inmóvil durante esos dos frames en la postura indicada por dichas rotaciones. Según la información puesta en el ejemplo, el esqueleto tendría situada la raíz en la posición 0 0 0 y con una postura en

forma de 'T', es decir, un cuerpo erguido con los brazos alzados. No se puede mostrar el esqueleto en el 3dsMAX ya que es demasiado simplificado y 3dsMAX no lo considera como un archivo válido. Las siglas pX, pY y pZ representan los canales de posición y las siglas rX, rY y rZ los canales de rotación.

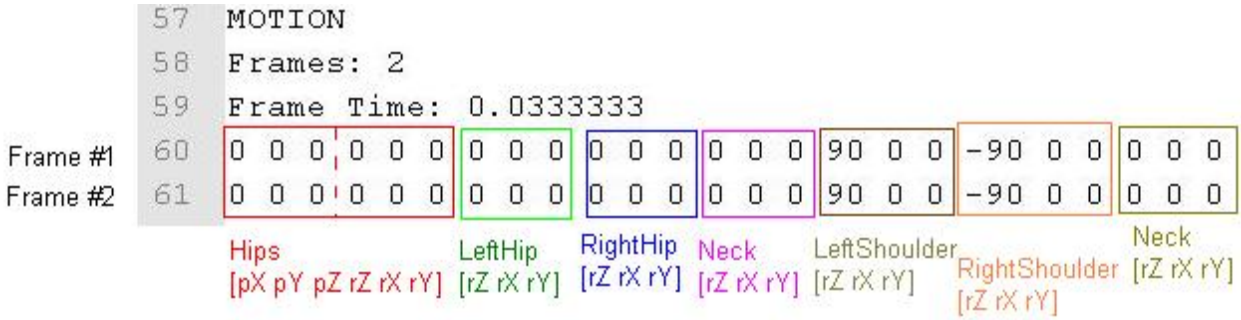


Figura I-5. Muestra de la parte Motion correspondiente con el esqueleto de la figura I-3.

En el Anexo II se encuentra un ejemplo sobre un archivo BVH de muestra que sigue las indicaciones escritas en este apartado y utiliza la jerarquía usada para el proyecto.

ANEXO II: Muestra de un archivo BVH

Este anexo es una muestra de un archivo BVH. Por temas de espacio, se ha omitido prácticamente toda la información de cada frame en la sección MOTION. Solo se han dejado tres frames. Esta muestra sigue la jerarquía usada en el proyecto, la cual está definida en la figura II-1.

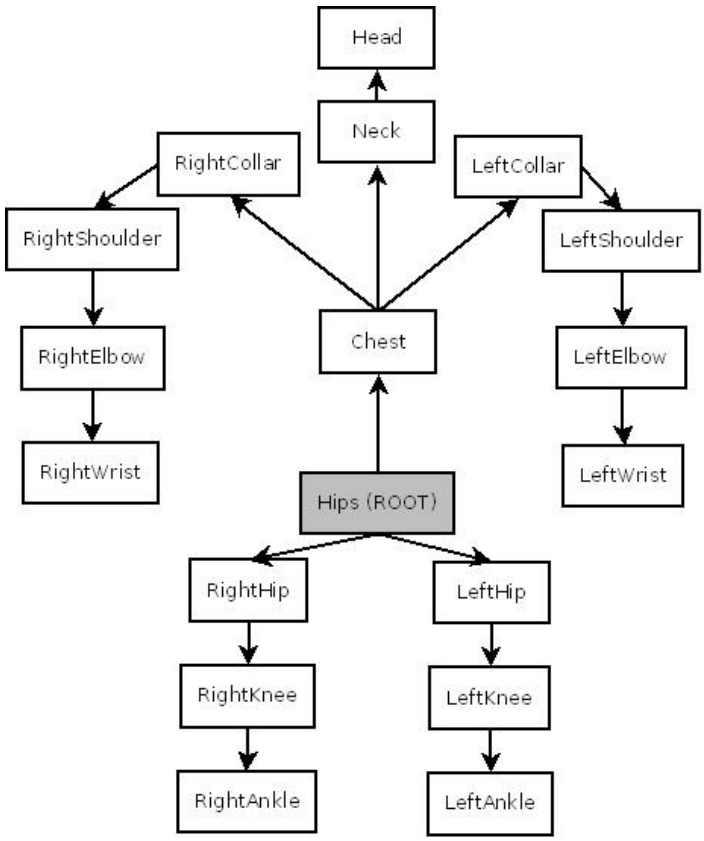


Figura II-1. Diagrama de la jerarquía del esqueleto usada en el proyecto

```
HIERARCHY
ROOT Hips
{
	OFFSET 0 0 0
	CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
	JOINT LeftHip
	{
		OFFSET 0.0987649 0.0 0.0
		CHANNELS 3 Zrotation Xrotation Yrotation
		JOINT LeftKnee
		{
			OFFSET 0.0 -0.425583 0.0
			CHANNELS 3 Zrotation Xrotation Yrotation
			JOINT LeftAnkle
			{
				OFFSET 0.0 -0.344971 0.0
				CHANNELS 3 Zrotation Xrotation Yrotation
				End Site
```

```
                        {
                              OFFSET 0.0 0.0 0.2
                        }
                  }
            }
      }
      JOINT RightHip
      {
            OFFSET -0.0987649 0.0 0.0
            CHANNELS 3 Zrotation Xrotation Yrotation
            JOINT RightKnee
            {
                  OFFSET 0.0 -0.425485 0.0
                  CHANNELS 3 Zrotation Xrotation Yrotation
                  JOINT RightAnkle
                  {
                        OFFSET 0.0 -0.366418 0.0
                        CHANNELS 3 Zrotation Xrotation Yrotation
                        End Site
                        {
                              OFFSET 0.0 0.0 0.2
                        }
                  }
            }
      }
      JOINT Chest
      {
            OFFSET 0.0 0.214597 0.0
            CHANNELS 3 Zrotation Xrotation Yrotation
            JOINT LeftCollar
            {
                  OFFSET 0.0 0.214597 0.0
                  CHANNELS 3 Zrotation Xrotation Yrotation
                  JOINT LeftShoulder
                  {
                        OFFSET 0.168187 0.0 0.0
                        CHANNELS 3 Zrotation Xrotation Yrotation
                        JOINT LeftElbow
                        {
                              OFFSET 0.0 -0.285515 0.0
                              CHANNELS 3 Zrotation Xrotation Yrotation
                              JOINT LeftWrist
                              {
                                    OFFSET 0.0 -0.302929 0.0
                                    CHANNELS 3 Zrotation Xrotation Yrotation
                                    End Site
                                    {
                                          OFFSET 0.0 -0.15 0.0
                                    }
                              }
                        }
                  }
            }
            JOINT RightCollar
            {
                  OFFSET 0.0 0.214597 0.0
                  CHANNELS 3 Zrotation Xrotation Yrotation
                  JOINT RightShoulder
                  {
```

```
                        OFFSET -0.168186 0.0 0.0
                        CHANNELS 3 Zrotation Xrotation Yrotation
                        JOINT RightElbow
                        {
                              OFFSET 0.0 -0.34569 0.0
                              CHANNELS 3 Zrotation Xrotation Yrotation
                              JOINT RightWrist
                              {
                                    OFFSET 0.0 -0.299922 0.0
                                    CHANNELS 3 Zrotation Xrotation Yrotation
                                    End Site
                                    {
                                          OFFSET 0.0 -0.15 0.0
                                    }
                              }
                        }
                  }
            }
                  JOINT Neck
                  {
                        OFFSET 0.0 0.214597 0.0
                        CHANNELS 3 Zrotation Xrotation Yrotation
                        JOINT Head
                        {
                              OFFSET 0.0 0.072305 0.0
                              CHANNELS 3 Zrotation Xrotation Yrotation
                              End Site
                              {
                                    OFFSET 0.0 0.14461 0.0
                              }
                        }
                  }
            }
      }
}
MOTION
Frames: 733
Frame Time: 0.0333333
0.0261336 -0.00966142 -0.0614465 1.97249 9.05828 3.12616 -0.201411 -7.2698
-0.0256937 0 4.70903 0 0 0 0 -5.88013 -4.77936 -0.493374 0 7.11171 0 0 0 0
-9.72361e-007 8.40357e-007 -7.90535e-005 0 0 0 7.02839 3.68564 2.81231 0
-22.4438 0 0 0 0 0 0 0 -8.29583 1.82907 3.5079 0 -20.7075 0 0 0 0 1.97251
14.2157 3.12616 -0.539547 -0.0037406 -0.539547
0.0239526 -0.0100755 -0.0604509 1.86568 8.87855 3.17721 -0.17199 -7.08721
-0.0213836 0 4.64014 0 0 0 0 -5.78835 -4.59138 -0.466436 0 6.9945 0 0 0 0
-4.31275e-007 8.17686e-007 2.82555e-007 0 0 0 8.09085 2.33296 2.94179 0 -20.3978
0 0 0 0 0 0 0 -8.59559 1.71504 3.54706 0 -20.9455 0 0 0 0 1.86568 14.0855
3.17721 -0.382362 -0.00257875 -0.382362
0.0199709 -0.0118284 -0.0568887 1.91624 8.13278 2.96517 -0.329504 -6.5887
-0.0380598 0 4.6794 0 0 0 0 -7.13682 -5.32231 -0.668341 0 6.11446 0 0 0 0
-1.19892e-006 2.64551e-006 -4.57363e-005 0 0 0 7.00896 1.90003 2.80773 0
-19.6749 0 0 0 0 0 0 0 -9.02772 0.805483 3.17554 0 -19.9222 0 0 0 0 1.91626
13.0655 2.96517 -0.301133 -0.00192434 -0.301133
...
```

ANEXO III: Código de conversión Matriz de rotación a Euler

Este Anexo contiene el código usado al principio para convertir las matrices de rotación en ángulos de Euler. En el proyecto no se usó por problemas en los cálculos. El código fue sacado de la referencia [MJB-98].

```
if(RotMatrix.elements[X2] > 0.998) { // singularity at north pole
      rotations.Y = RadianToDegree(atan2(RotMatrix.elements[Z1],
                                         RotMatrix.elements[Z3]));
      rotations.Z = RadianToDegree(PI/2);
      rotations.X = 0;
}
else if (RotMatrix.elements[X2] < -0.998) { // singularity at south pole
      rotations.Y = RadianToDegree(atan2(RotMatrix.elements[Z1],
                                         RotMatrix.elements[Z3]));
      rotations.Z = RadianToDegree(-PI/2);
      rotations.X = 0;
}
else
{
      rotations.Y =  RadianToDegree(atan2(-RotMatrix.elements[X3],
                                         RotMatrix.elements[X1]));
      rotations.X = RadianToDegree(atan2(-RotMatrix.elements[Z2],
                                         RotMatrix.elements[Y2]));
      rotations.Z = RadianToDegree(asin(RotMatrix.elements[X2]));
}
```