UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA

EN INFORMÁTICA

PROYECTO FIN DE CARRERA

GANAS: Generador Automático del Lenguaje de Signos

Vanesa Herrera Tirado

Julio, 2007

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Departamento de Informática

PROYECTO FIN DE CARRERA

GANAS: Generador Automático del Lenguaje de Signos

Autor: Vanesa Herrera Tirado
Director: Carlos González Morcillo

Julio, 2007.

TRIBUNAL:

Presidente:
Vocal:
Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL	SECRETARIO
Fdo.:	Fdo.:	Fdo.:

© Vanesa Herrera Tirado. Se permite la copia, distribución y/o modificación de este documento bajo los términos de la licencia de documentación libre GNU, versión 1.1 o cualquier versión posterior publicada por la *Free Software Foundation*, sin secciones invariantes. Puede consultar esta licencia en http://www.gnu.org.

Este documento fue compuesto con LaTeX. Imágenes generadas con OpenOffice.

Resumen

Para cualquier persona, la comunicación es fundamental para interactuar con los individuos de su entorno y, a pesar de la existencia de diferentes modos, los más utilizados son el lenguaje oral y escrito. Para que haya comunicación es necesario utilizar un lenguaje común entre el emisor y el receptor del mensaje.

Para aquellas personas que no poseen deficiencias auditivas o visuales, el proceso de recepción e intercambio de información no suele ser complejo, pero para aquellos individuos que sufren alguna de estas discapacidades la comunicación se convierte en un proceso complejo debido a que el lenguaje utilizado no es el que predomina en la sociedad.

La comunidad con deficiencia auditiva desarrolló un lenguaje no verbal para poder interactuar con otras personas, llamado lenguaje sordomudo o lenguaje de signos. El lenguaje de signos consiste en la utilización de las manos como herramienta de comunicación tanto para el deletreo de palabras como para representar palabras completas o expresiones mediante el movimiento de brazos, manos y dedos acompañado generalmente de gestos faciales.

El problema que reside en las personas no oyentes a la hora de comunicarse es que su lenguaje no es común al resto de personas que componen la sociedad, puesto que el lenguaje de signos es conocido por una pequeña parte de la comunidad.

La solución ideal a este problema es que al igual que desde niños adquirimos el lenguaje oral y escrito, estudiáramos también el lenguaje de signos. Otra posible solución es la de disponer de intérpretes en organismos e instituciones para que la persona no oyente pueda comunicarse, pero esto supone un enorme gasto que es difícil costear en muchas ocasiones.

El objetivo de este proyecto fin de carrera es desarrollar un conjunto de herramientas software que ayuden a la síntesis automática de animaciones correspondientes a frases en el lenguaje de signos de la comunidad de deficientes auditivos de España. En concreto se han construído módulos para la creación de una biblioteca de gestos asociada a un esqueleto tridimensional genérico, un módulo para su incorporación a cualquier personaje virtual que se adapte a la descripción anterior y finalmente un módulo de composición, integración y generación de imagen basado en técnicas de animación no lineal.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

INTRODUCCIÓN

Las nuevas tecnologías se han ido integrando poco a poco en nuestras vidas ayudándonos en muchas tareas complejas. Un ejemplo claro es el desarrollo de mecanismos que ayudan a personas discapacitadas en la eliminación de barreras físicas y sociales.

Con el desarrollo de *GANAS* se pretende ayudar a entender la información del entorno a personas con discapacidades auditivas y cuya información no pueden captar en situaciones donde la comunicación es principalmente oral. Un ejemplo en el que se necesitaría información visual es en situaciones en las que la información se da a través de megafonía, en este caso una persona sorda sin una ayuda visual no tendrá conocimiento de los datos anunciados.

Una forma de eliminar las barreras de comunicación que sufre la comunidad con discapacidad auditiva es mediante la generación de vídeos en los que se represente el lenguaje de signos a través de un personaje virtual y que podrán ser integrados en multitud de escenarios. Otra vía de actuación es la creación de mecanismos que permitan un rápido aprendizaje del lenguaje de signos en los colectivos interesados. Con el presente proyecto se pretende abordar estas dos líneas de trabajo, facilitando así la creación automática de contenidos visuales para el uso y aprendizaje del lenguaje de signos.

Con la existencia de software libre se pueden crear potentes aplicaciones, gracias a la compartición de código que puede ser mejorado por cualquier usuario. Este principio se ha

seguido para el desarrollo del proyecto fin de carrera que se presenta.

1.1. Enfoque

GANAS tiene como objetivo desarrollar una herramienta que permita al usuario la generación de vídeos en los que un personaje virtual interprete el lenguaje de signos.

Debido a que usuario final puede no ser un experto en el lenguaje de signos, la herramienta *GANAS* dispondrá de una interfaz amigable para que el usuario no experto pueda obtener secuencias de frases y expresiones a partir de la composición de palabras existentes y el usuario experto podrá crear nuevas palabras no existentes aún en el sistema. Independientemente del conocimiento del lenguaje de signos un usuario, generalmente, no tendrá experiencia en diseño 3D, es por eso que la herramienta *GANAS* ofrece una interfaz gráfica desde la cual se pueden crear palabras sin tener que posicionar de manera directa los huesos que componen el esqueleto. La herramienta *GANAS* también permite a usuarios con conocimientos en diseño 3D manipular el esqueleto del personaje virtual (en las tres dimensiones) para generar nuevas poses que se emplearán para obtener palabras en lenguaje de signos.

Con este proyecto se pretenden conseguir mecanismos generales de representación y almacenamiento de gestos tridimensionales correspondientes al lenguaje de signos de la comunidad de usuarios de España (LSE).

No es objetivo del proyecto la construcción de un editor multiregión, aunque podría utilizarse en otros países sin ningún problema personalizando el diccionario. Tampoco se pretende generar un diccionario completo de todas las palabras y expresiones que se pueden representar mediante el lenguaje de signos debido a que requeriría un elevado coste temporal.

1.2. Estructura del trabajo

El documento se divide en siete capítulos y un anexos.

- En el capítulo 1 se presenta una introducción general a la problemática y al enfoque dado, planteando los objetivos clave que se pretenden alcanzar.

- En el capítulo 2 se describen de forma más detallada los objetivos que se desean conseguir.
- En el capítulo 3 se describen algunos antecedentes y el estudio del estado del arte, realizado sobre diferentes sistemas de generación automática de lenguaje de signos, tecnologías y herramientas existentes.
- El capítulo 4 se centra en los métodos utilizados en el desarrollo y arquitectura del sistema, así como en el ciclo de vida del software utilizado, módulos que componen el proyecto y algoritmos más significativos.
- En el capítulo 5 se exponen los resultados obtenidos tras el desarrollo del sistema, características deseables frente a sistemas similares y estudio de posibles escenarios de aplicación.
- En el capítulo 6 se describen algunas de las posibles mejoras y ampliaciones que podrían realizarse en el sistema, también se hace referencia a líneas de trabajo futuro.
- En el capítulo 7 se presenta la bibliografía y referencias web utilizadas en el desarrollo del sistema y documentación.
- Anexos:
    - Manual del usuario dividido en tres partes: módulo de representación 3D, interfaz de administración e interfaz web

Capítulo 2

OBJETIVOS DEL PROYECTO

El objetivo principal de *GANAS* es el desarrollo de un conjunto de módulos software que permitan la síntesis automática de animaciones asociadas a un personaje virtual que represente palabras del lenguaje escrito en lenguaje de signos. Este objetivo clave se alcanzará mediante la consecución de los siguientes subojetivos:

- La herramienta permitirá la creación de nuevas poses estáticas (o poses clave) mediante la manipulación directa de huesos. Esto dará la posibilidad de crear nuevas poses a usuarios técnicos y personal cualificado de la herramienta.

- La generación de poses estáticas a partir de otras poses ya creadas previamente permitirá la creación de nuevos movimientos a usuarios del sistema que no estén acostumbrados a trabajar con herramientas de diseño 3D.

- El almacenamiento de estas poses estáticas se realizará en un formato de intercambio independiente de la herramienta, que permitirá su reutilización por cualquier otro sistema. De esta forma la biblioteca de poses obtenida del uso del módulo de edición de signos será accesible desde otras aplicaciones distintas a *GANAS*.

- El sistema deberá ser capaz de generar vídeos para cada palabra (formado por un conjunto de poses). Para ello se realizará un proceso de interpolación basado en técnicas de animación no lineal partiendo de las posiciones de las poses clave.

- La generación de frases o expresiones se realizará como combinación de vídeos de palabras. Esta combinación requerirá la generación de fragmentos de vídeo auxiliares que conectarán las poses clave final e inicial de las palabras adyacentes.

- Debido a que el proceso de render es muy costoso en tiempo, se prestará especial atención a la eficiencia en el ámbito de la reutilización de vídeos ya generados, realizando este proceso únicamente en los fragmentos necesarios para la conexión de palabras y la composición de elementos no lineal.

- El sistema se desarrollará empleando algunos mecanismos que faciliten su usabilidad, como métodos para arrastrar poses (drag & drop), así como el empleo de APIs de widget multiplataforma y estándares que permitan la portabilidad.

- El sistema deberá ser configurable para adaptarlo a las diferentes necesidades que puedan plantearse en distintos ámbitos de implantación, como la elección del tipo y formato de vídeo, métodos de almacenamiento en base de datos, etc...

- El sistema se desarrollará bajo una licencia Libre compatible con GLP de GNU. En la actualidad no existe ninguna herramienta libre ni gratuita para la representación del lenguaje de signos.

- El desarrollo del sistema se planteará para su posible utilización en los principales sistemas operativos (GNU/Linux, Microsoft Windows y Mac OSX), por lo que el desarrollo se basará en el uso de librerías abiertas y estándares multiplataforma.

Capítulo 3

ANTECEDENTES. ESTUDIO DEL ESTADO DEL ARTE

3.1. Herramientas existentes

Hoy en día, a pesar de la importancia del lenguaje de signos y del avance de las tecnologías, no existe una gran variedad de sistemas que permitan la generación del lenguaje de signos mediante la utilización de un personaje virtual.

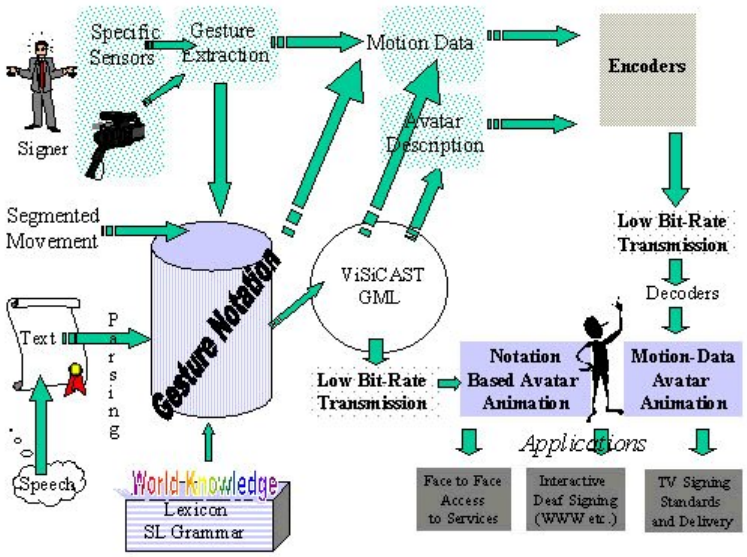

Figura 3.1: Estructura del sistema VisiCast

En cuanto a los sistemas existentes, la mayoría de ellos tienen como finalidad la construcción de un *avatar* (personaje virtual) que represente el lenguaje de signos mediante la generación de un vídeo creado desde una interfaz gráfica. Los sistemas que se han estudiado son: VisiCAST, eSIGN, VCommunicator y Sign Smith.

3.1.1. VisiCAST

VisiCAST es un proyecto creado bajo la supervisión de *"European Union Fifth Framework"* [24] formando parte de *"Information Society Technologies (IST)"* y centrado en la comunidad sorda europea. Los objetivos de este sistema son:

- Mostrar un humanoide en una pantalla de televisión en la oficina de correos de Reino Unido para mostrar información a los no oyentes acerca de los servicios.
- Utilizar el sistema en programas de televisión.
- Implantar VisiCAST en algunas de las páginas web pertenecientes a participantes del sistema.

La imagen 3.1 muestra un esquema de funcionamiento del sistema.

Para este proyecto se crearon diferentes humanoides en 3D. Los movimientos que había que aplicar a los *avatares* se hicieron a través de la captura del movimiento de un humano.

Ventajas:

- Los movimientos del personaje virtual son más fieles a la realidad.
- Reducción del tiempo de desarrollo, ya que las animaciones no hay que hacerlas de forma manual.

Inconvenientes:

- Poca flexibilidad. El usuario no puede crear nuevos gestos. Cada vez que se necesita una palabra o expresión nueva, es necesario volver a captar los movimientos del intérprete del lenguaje de signos.

3.1.2. eSIGN

El proyecto *eSIGN* [21] es la continuación del proyecto VisiCAST y aún no ha concluido su desarrollo. Este sistema pretende ayudar a los no oyentes en la compresión de la información a través de la inserción de los vídeos obtenidos con VisiCAST en:

- Páginas web, para mejorar la accesibilidad.
- Conversaciones en internet (chat en lenguaje de signos).
- Pantallas táctiles que suelen ser puntos de información.

Según el proyecto *eSIGN*, la introducción de un *avatar* que interprete el lenguaje escrito de una página web, se consigue mediante los pasos siguientes:

- Revisar el texto para que sea comprensible en el lenguaje de signos.
- Utilizar el editor de eSIGN de tal forma que traduzca texto plano en texto SiGML, el cual contiene información sobre los signos e instrucciones de animación por computador.

- Modificar la página web para poder agregar el humanoide o bien crear una ventana auxiliar.
- Añadir un icono que al pulsarlo traduzca el lenguaje escrito en lenguaje de signos, o bien, crear un enlace desde el texto.
- Actualizar la página.
- Una vez realizadas las acciones anteriores, cuando pulsemos sobre el icono o el texto aparecerá el vídeo virtual correspondiente al lenguaje escrito introducido.

A continuación se muestran las ventajas e inconvenientes de eSIGN:
Ventajas:

- Al basarse en el proyecto anterior, los movimientos de los avatares son obtenidos mediante Motion Capture. Por lo tanto, una de las mayores ventajas es el realismo de los gestos del humanoide.

Inconvenientes:

- El usuario no puede crear nuevas palabras o gestos, sólo puede utilizar las ya existentes.
- Es necesario descargar un plugin para poder visualizar los vídeos insertados en las páginas web.
- Sólo funciona con sistemas operativos Windows 2000 o Windows XP.
- Sólo es válido para navegadores Internet Explorer.

Actualmente es posible descargar el plugin y acceder a algunas páginas donde se muestra su funcionamiento, pero el editor *eSIGN* aún no está disponible.

3.1.3. VCommunicator y Sign Smith

VCommunicator y *Sign Smith* son dos productos desarrollados por VCOM3D [23], compañía de Orlando, Florida, dedicada a la educación y accesibilidad de personas sordas mediante el uso de tecnologías 3D.

A diferencia de *VisiCAST* no utiliza Motion Capture para generar los movimientos del avatar, sino que el propio usuario puede crear nuevos gestos utilizando el editor.

3.1.3.1. VCommunicator

El producto *VCommunicator* consta de las siguientes herramientas software:

- Studio authoring tool. Se trata de una colección de avatares destinados a la personalización del humanoide que actuará de intérprete del lenguaje de signos. Según la descripción que aparece en la página web, esta librería constará de diferentes avatares distinguidos por nacionalidades, edades, talla y otras características. *Character libraries* podrá ser utilizada con *Vcommunicator Studio* y *Gesture Builder authoring tools*, si bien aún no existe ninguna demo para poder corroborar los objetivos.
- Gesture builder. Herramienta destinada a la creación de nuevos gestos. El usuario podrá relacionar palabras, expresiones o frases escritas con los vídeos generados, que muestran al personaje virtual representado esta información en lenguaje de signos
- Character libraries. Colección de personajes virtuales.
- Vcommunicator mobile. Herramienta destinada a la visualización de los vídeos generados mediante *Gesture Builder* en distintos dispositivos.

3.1.3.2. Sign Smith

El producto *Sign Smith* engloba los siguientes paquetes:

- ASL Animations. Diccionario de animaciones virtuales de múltiples palabras en el lenguaje americano de signos. Actualmente están disponibles cuatro volúmenes, que se clasifican en diferentes temáticas (animales, colores, matemáticas,...etc).
- Illustrate dictionary. Diccionario con palabras representadas en los lenguajes de signos americano e inglés. Proporciona además el deletreo de 500 palabras, y la correspondiente animación.

- Studio. Herramienta que permite al usuario la creación de videos que representan palabras, frases o expresiones en lenguaje de signos.

El problema al estudiar las herramientas creadas por *VCom3D* es que, a excepción de *Character Libraries, Illustrate Dictionary y ASL Animations*, el resto de las herramientas destinadas a la creación y edición de gestos, palabras y frases aún no están disponibles. Está previsto el lanzamiento del resto de productos para el año 2007, pero actualmente lo que existe son demos y una serie de objetivos que desean cumplir.

A continuación se describen las ventajas e inconvenientes de los productos *VCommunicator* y *Sign Smith*.

Ventajas:

- Permite utilizar diversos personajes virtuales en las animaciones (Character Libraries).
- El usuario puede crear nuevas palabras a través de un editor de gestos(Gesture Builder).
- Las animaciones incluyen expresiones faciales.
- El usuario puede crear nuevas animaciones según sus necesidades (Studio).
- El formato de los vídeos creados puede ser GIF o MOV, por lo que el resultado puede insertarse en páginas web con independencia de la plataforma.

Inconvenientes:

- La creación de nuevos gestos a través del editor Gesture Builder no es trivial, puesto que debe posicionar al personaje en un espacio 3D. Este posicionamiento resulta complejo para usuarios que no estén acostumbrados a sistemas de diseño 3D. Necesita realizar rotaciones y translaciones de todos los los huesos que desea mover.
- Coste elevado del sistema, en torno a los 5.400 $.

3.2. Estudio del Lenguaje de Signos

3.2.1. Introducción.

El lenguaje de signos surge debido a la necesidad intrínseca del ser humano de comunicarse con la sociedad que mayoritariamente es oyente. Su estudio no fue tema de interés hasta el siglo XX. Uno de los primeros libros publicados referentes a este tema fue en los años sesenta del lingüista William C. Stokoe, titulado S*ign Language Structure: An outline of the visual communication system of the American deaf.*

Para poder entender que el estudio del lenguaje de signos es reciente, teniendo en cuenta que surgió al mismo tiempo que el lenguaje oral, hay que saber que hasta finales del siglo XVIII las personas con discapacidad auditiva eran despojados de cualquier tipo de educación, pues se les consideraba incapaces de poder aprender. En esta época es cuando aparece en España la primera escuela para personas sordas, creada por Joan Albert Marti en Barcelona, pero no es hasta mediados del siglo XIX cuando es considerada como *Escuela Municipal de Sordomudo*. Esta discriminación era a nivel mundial, así por ejemplo en la Antigua China los sordos eran arrojados al mar y hasta el siglo XV la Iglesia Católica consideraba que las personas sordas no tenían alma. Aunque este rechazo parezca lejano, hoy en día sigue existiendo una gran discriminación hacia esta comunidad, principalmente en el área laboral.

Según *Sacks* [11], el lenguaje de los sordos como lenguaje de señas, se trata de un lenguaje rico y no un simple vocabulario, ya que como cualquier otro lenguaje tiene dimensiones semánticas, sintácticas y morfológicas entre otras.

A pesar de que el lenguaje de signos es la lengua natural de las personas sordas esto no significa que sea igual en todo el mundo, debido a que una misma seña no tiene que significar lo mismo en distintos puntos de la geografía. Esto ocurre incluso entre diferentes regiones de un mismo país. Sin embargo, si se establece comunicación entre dos personas de distintas partes del mundo, podrán entenderse puesto que gran parte de las señas son icónicas.

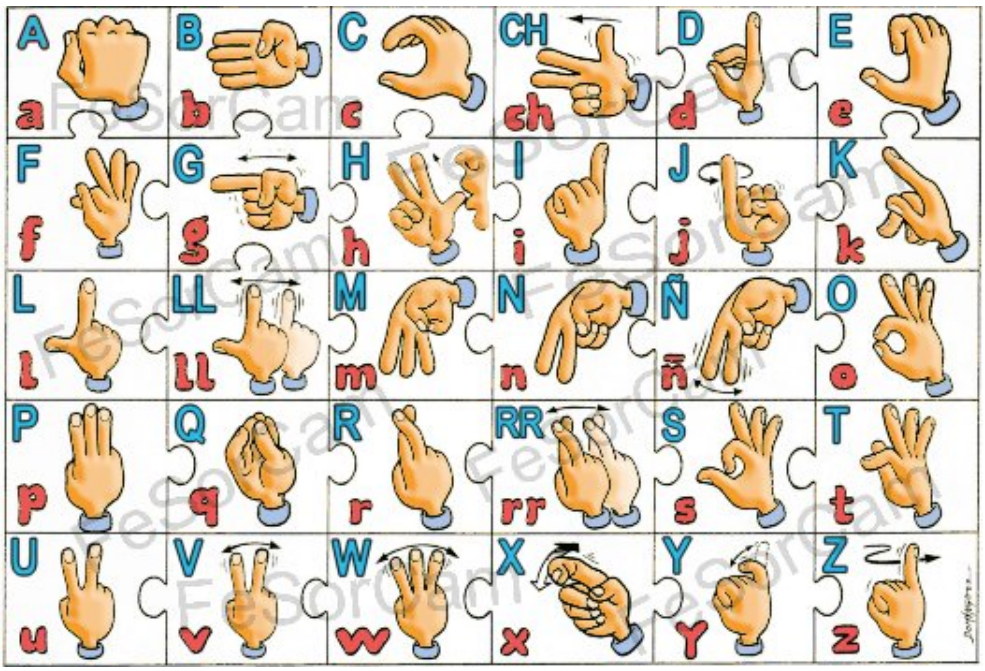

Figura 3.2: Alfabeto dactilológico español.

3.2.2. Componentes

El comportamiento lingüístico del sordo está formado por diferentes componentes que se interrelacionan entre sí [25]:

- Signos manuales. Referido al movimiento de las manos para representar palabras o letras en lenguaje de signos.
- Quinésica facial. Referida al uso del movimiento de los labios.
- Quinésica somática. Referida al movimiento corporal.
- Quinésica oral. Referida al uso del movimiento de los labios junto a los gestuales. Estos movimientos orales no son iguales en todos los discapacitados auditivos. Existen grandes diferencias entre sordos profundos que no han sido escolarizados y sordos no profundos escolarizados. En los primeros los movimientos no serán palabras sino que será más bien un componente expresivo.
- Dactilología. Representación de cada letra del alfabeto por un signo.

El uso del alfabeto gestual se basa principalmente en los siguientes casos:

- En la introducción de nuevos conceptos que en un principio no cuentan con signo propio.
- En el uso de antropóminos o topóminos empleados cuando el no oyente entabla comunicación con un oyente, ya que entre no oyentes se usan gestos propios.

Los signos manuales es el componente más importante de Lenguaje de Signos. Al igual que en el resto de lenguajes los caracteres son [26]:

- Modo de operación: modo en el que el sistema actúa al que se dirige.
- Dominio de validez: donde actúa el sistema.
- Naturaleza y número de signos: están en función del dominio de validez.
- Tipo de funcionamiento: relación existente entre los signos.

La principal diferencia entre un lenguaje oral y un lenguaje de signos reside en el modo de operación, ya que este último es visual. En la lengua de signos española Muñoz Baell [3] distinguió siete elementos que componen signos interrelacionándose (parámetros):

- Configuración: forma de la/s mano/s.
- Orientación: posición de la/s palma/s.
- Lugar de articulación: espacio donde tiene lugar el signo.
- Plano: referente a la longitud del brazo y posicionamiento respecto al cuerpo.
- Punto de contacto: parte de la mano que entra en contacto con la persona que representa el signo.
- Movimiento: puede ser tanto de los dedos, mano, antebrazo o de la cara, cabeza y cuerpo.
- Componente no manual: son las expresiones faciales y movimiento de tronco y cabeza.

3.2.3. Clasificación

Una clasificación de los signos puede ser atendiendo a las relaciones semánticas [8]:

- Signos motivados: aquellos que están influidos por su referente. En el lenguaje de signos existe mayor número de este tipo de signos que en el lenguaje oral. Estos se pueden dividir a su vez en:
  - Signos icónicos: se caracterizan por existir una relación entre el gesto y lo que representan. Un ejemplo es la palabra mujer, que se representa gestualmente cogiendo con los dedos índice y pulgar el lóbulo de la oreja derecha, aludiendo a la costumbre existente de que la mujer usa pendientes. Los signos icónicos pueden reproducir la forma, un movimiento o una relación espacial.
  - Signos deícticos. El gesto se refiere a un contexto espacio-temporal. Así, por ejemplo para referirse a la primera personas (yo), el dedo índice apunta hacia el cuerpo de uno mismo.
- Signos intermedios: se caracterizan por tener su origen en la dactilología. Se usan para expresar nombres propios y lugares entre otros.
- Signos arbitrarios: son aquellos no guardan ningún tipo de relación con su referente son llamados signos arbitrarios.

3.3. Animación por ordenador

3.3.1. Ciclo general de producción

El diseño asistido por computador (CAD), se puede definir como la aplicación de la informática al proceso de diseño [15]. Su uso depende del área tecnológica y su importancia y utilización es mayor cada día.

Algunas de las ventajas que se obtienen de su uso son:

- Mejora de la calidad de los productos, puesto que permite solucionar problemas en las etapas tempranas del diseño.

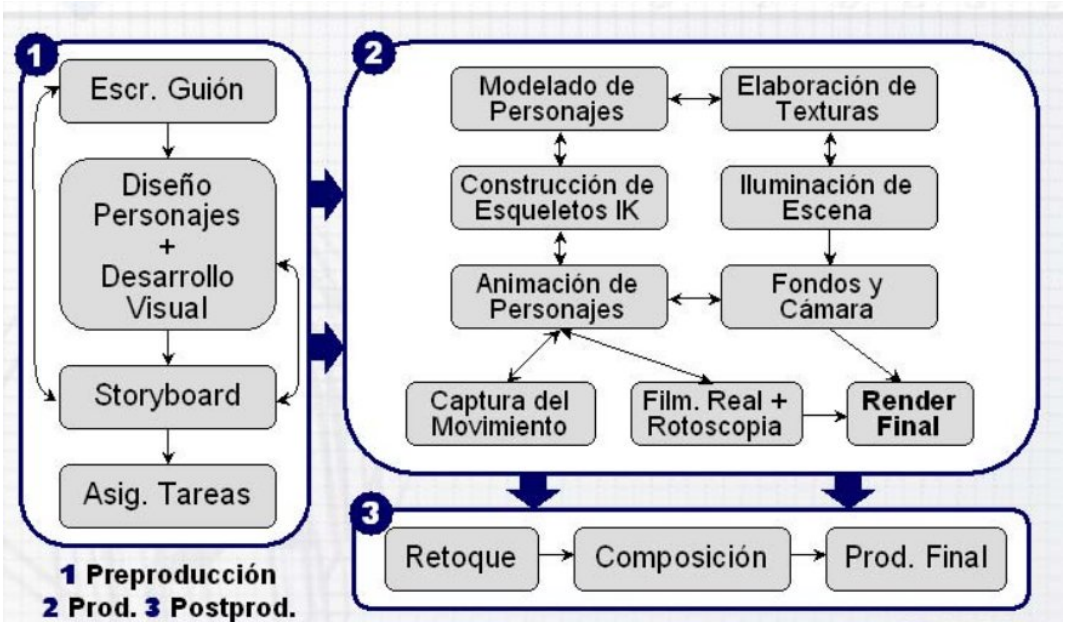

Figura 3.3: Proceso de Animación 3D [10].

- Reducción de los costes de fabricación, debido a la facilidad de realizar cambios.
- Reducción del tiempo de diseño y por consiguiente disminución del tiempo que tarda un producto en salir al mercado.
- Aumento de la capacidad de reutilización.

La animación por computador se utiliza cada día más debido no sólo a las ventajas descritas anteriormente, sino también gracias al avance de las tecnologías y al amplio abanico de áreas en las que puede ser utilizada (ciencia, educación, entretenimiento,...). El proceso de animación basada en computador tiene diferentes variantes [7], pero en todas ellas las etapas básicas son: preproducción, producción y postproducción.

Preproducción. Esta etapa abarca la conceptualización y planificación del proceso de diseño. Una preproducción errónea o inadecuada puede suponer grandes retrasos y aumento de los costes del proyecto. En la preproducción se distinguen tareas visuales y no visuales, dentro de las nos visuales se encuentra la escritura del guión. Las tareas visuales son:

- Selección de colores clave y estilo que se desea obtener.

- Desarrollo de los personajes mediante bocetos, maquetas y modelos 3D preliminares.
- Storyboard. Consiste en una serie de imágenes que plasman la historia que se desea contar. Se obtiene una previsualización y estructura de lo que se va a desarrollar, facilitando la localización de posibles errores que de descubrirse en etapas más tardías supondrían costes mayores.

Producción. Los pasos seguidos en esta etapa son: modelado, creación de texturas, iluminación, animación y renderizado. En primer lugar se modelan los personajes y objetos que conformarán la escena o escenas deseadas.

En el modelado de objetos se emplean diferentes técnicas para esculpir los objetos como digitalización mediante escáneres o modelado manual mediante un programa de diseño 3D. El resultado, independientemente de la técnica utilizada, es la obtención de la geometría del modelo, típicamente representada como una malla poligonal 3D. Esta etapa es muy costosa en tiempo.

La elaboración de texturas permite que los objetos diseñados adquieran un mayor realismo, adaptándose mejor a las necesidades finales. Las texturas se puede definir como la piel que envuelve a los modelos y la forma de obtenerlas es pintándolas o digitalizando texturas del mundo real.

Dentro de la etapa de producción también se encuentra la iluminación de la escena, en la que se ajustan los puntos de luz y se emplean una serie de algoritmos para sombrear determinadas áreas.

La construcción de esqueletos y la animación es la penúltima etapa en la producción 3D. Es importante definir la jerarquía de los huesos que tendrán los modelos (aquellos que los tengan) para que el movimiento sea lo más parecido posible al objetivo deseado. En la animación es muy común el empleo de cinemática inversa y captura del movimiento.

El proceso de renderizado puede clasificarse en dos tipos diferentes: render progresivo o preliminar y render final. Para hacer pruebas es importante no usar una calidad muy alta, puesto que no es necesario obtener todos los detalles y un aumento en la calidad final supone un aumento del tiempo de renderizado. Por eso es aconsejable incluir primero las acciones principales e ir añadiendo las demás hasta obtener progresivamente el render final.

Postproducción. En la postproducción se suelen realizar los retoques finales, como la corrección de colores y transparencias. Es en esta etapa del proceso de diseño 3D es donde se monta el vídeo y se graba en el formato final.

3.3.2. Métodos de animación

Existen diferentes aproximaciones utilizadas en la animación por computador [16]. Se usan unas u otras dependiendo de los objetivos que se pretenden obtener. En este apartado se estudiarán algunas de las técnicas básicas más empleadas, así como otras más avanzadas.

3.3.2.1. Técnicas básicas. Interpolación

Las técnicas básicas de animación 3D se basan en el uso de curvas, se denominan técnicas de bajo nivel puesto que es el animador el que modifica directamente la posición de los nodos de las curvas de interpolación. Es tarea del animador establecer la posición clave de los objetos y la computadora es la encargada de ir interpretando valores.

Los movimientos que se producen en el mundo real se corresponden con trayectorias de curvas. Por este motivo en gráficos por computador, se emplean métodos numéricos curvos de interpolación como splines cúbicas, cardinales,...

Representación paramétrica vs no paramétrica. Las curvas tienen una representación matemática precisa y las ecuaciones que las definen pueden estar en forma explícita, implícita y paramétrica. En el campo de los gráficos por computador, la representación más utilizada es la paramétrica, ya que a pesar de que una misma curva puede tener más de una representación, la representación paramétrica facilita su tratamiento algorítmico. [6].

Ecuaciones explícitas. Este tipo de ecuaciones se denomina así porque aparece de forma explícita una de las variables en función de las otras dos.

La forma no paramétrica puede ser implícita y explícita. Por ejemplo, se puede expresar y y z en función de x:

$$y = f(x)$$

$$z = g(x)$$

Inconvenientes de la representación explícita:

- Para un valor de x sólo se puede obtener un valor de y, por lo que curvas multivaluadas sobre el eje x, como ocurre por ejemplo con el círculo o la elipse, se tienen que representar con varios segmentos de curva.
- La descripción de curvas con tangentes verticales es muy compleja, ya que hay que representar una pendiente infinita.

Ecuaciones implícitas. Se representan igualando a cero la ecuación de las variables. Se emplean ecuaciones de la forma $f(x, y, z) = 0$

Inconvenientes de la representación implícita:

- Si la ecuación tiene más soluciones de las deseadas, es necesario el uso de criterios adicionales.
- Puede ser complejo determinar si las direcciones tangentes resultantes al unir dos curvas, coinciden en el punto de unión (problemas de continuidad paramétrica).
- Computacionalmente, el uso de la ecuación implícita es mucho más costoso, puesto que es necesario el cálculo de raíces cuadradas. Por otra parte, sería necesario añadir criterios de decisión adecuados, ya que toda raíz cuadrada tiene dos soluciones y una de ellas podría ser no válida.

Ecuaciones paramétricas. La curva se representa mediante una serie de parámetros. A medida que se modifican los valores de los parámetros, se van obteniendo las coordenadas. Si tomamos el parámetro *u*, una curva en forma paramétrica sería:

$$x = f_1(u)$$
$$y = f_2(u)$$
$$z = f_3(u)$$

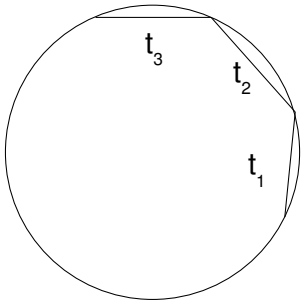

Figura 3.4: Generación de una circunferencia mediante la unión de puntos obtenidos de la ecuación paramétrica.

La principal ventaja de las ecuaciones paramétricas es que ofrecen mayor flexibilidad.

Veamos un ejemplo: Supongamos que queremos representar un círculo de radio 1 y centro en el punto (0, 0).

Ecuación implícita: $f(x, y) = x^2 + y^2 - 1 = 0$

La ecuación paramétrica se representa mediante las funciones trigonométricas seno y coseno:

$$x(u) = \text{sen}(2\pi * u)$$
$$y(u) = \cos(2\pi * u)$$
$$0 < u < \pi$$

La ecuación paramétrica está en función del seno y del coseno que varían entre 0 y 1. Si variamos t entre los límites 0 y 1 iremos obteniendo puntos que constituyen la circunferencia (ver figura 3.4). Si la variación es lo suficientemente pequeña las rectas que unan los puntos obtenidos dibujarán la circunferencia de forma suave sin que se noten los segmentos.

Curvas cúbicas paramétricas. Curvas de Spline. A menudo, existe la necesidad de encontrar expresiones analíticas de curvas de las que no se conoce su expresión matemática, o bien se conoce pero es demasiado compleja. Para resolver el problema se evalúa la curva como un conjunto de puntos y se construye la solución mediante aproximación o interpolación. Estas técnicas aplicadas al movimiento (animación), se utilizan describiendo una serie de propiedades como puede ser la posición y rotación de un elemento en los diferentes ejes

de coordenadas (en un momento dado) y cambiándolas en otro punto. De esta forma no es necesario almacenar la imagen completa, sino tan sólo los valores que han cambiado de una imagen a otra.

Como hemos visto anteriormente, la representación de una curva en forma implícita o explícita no es aconsejable. Las ecuaciones paramétricas son las más utilizadas en animación por computador, así se solventan los problemas comentados en el punto anterior.

El principal problema de las representación paramétrica de las curvas en aplicaciones de diseño, se plantea debido a la necesidad de editar la curva. Para la edición sería necesario modificar las funciones que describen las coordenadas y los intervalos de variación de los parámetros. La forma de hacer factible las modificaciones de las funciones es fijando la forma de las mismas, de esta manera la modificación implica que el efecto en la curva del cambio de un número de parámetros es predecible. Por esta razón se utilizan funciones polinómicas, es decir, en lugar de utilizar curvas lineales a trozos, se aproximan con curvas polinomiales a trozos.

Cada segmento de la curva global se indica mediante tres funciones (x, y, z), que son polinomios cúbicos en función de un parámetro. Las razones por las que se utilizan polinomios cúbicos son las siguientes:

- Las curvas de grado más pequeño que no son planas en tres dimensiones son polinómicas de grado tres.
- Si se utiliza una representación de menor grado la interpolación no será posible, es decir, que un segmento de curva pase por dos puntos extremos (calculados mediante la derivada en ese punto)
- Si se utilizan polinomios cúbicos con cuatro coeficientes, se emplean cuatro incógnitas para la resolución de éstos. Los cuatro valores pueden ser los puntos extremos y las derivadas en ellos.
- Los polinomios de mayor grado requieren un mayor número de condiciones para el cálculo de los coeficientes y pueden formar ondulaciones que son difíciles de controlar.

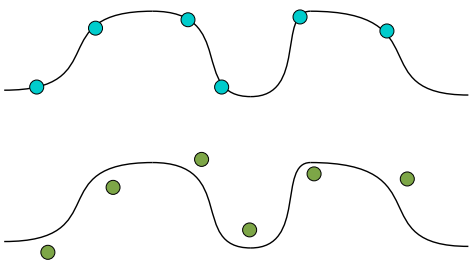

Figura 3.5: Interpolación (figura superior) y aproximación (figura inferior).

Los polinomios cúbicos [5] que definen un segmento de curva $Q(t) = [x(t)y(t)z(t)]^T$ tienen la forma:

$$x(t) = a_x t^3 + b_x t^2 + c_x t^1 + d_x,$$
$$y(t) = a_y t^3 + b_y t^2 + c_y t^1 + d_y,$$
$$z(t) = a_z t^3 + b_z t^2 + c_z t^1 + d_z$$
$$0 \leq t \leq 1.$$

Una función spline, en aplicaciones gráficas, se caracteriza por estar constituida por una serie de curvas, formadas con secciones polinómicas, que se unen entre sí cumpliendo ciertas condiciones de continuidad en las fronteras de los intervalos. La forma de especificar estas curvas es a través de una serie de puntos que forman la estructura general de la curva. Estos puntos son llamados puntos de control. Se pueden distinguir dos casos:

- Interpolación. Un método interpola el conjunto de puntos de control cuando la curva pasa por ellos.
- Aproximación. Un método aproxima el conjunto de puntos de control cuando los polinomios se ajustan al trazado de los mismos, pero sin tener que pasar necesariamente por ellos (ver figura 3.5).

Matemáticamente, se puede definir una spline de grado k [13]:*Dada una particion* Γ *de un intervalo* $[a, b]$*, una función spline S en* $[a, b]$*, de grado* $k \geq 0$*, correspondiente a la partición* Γ*, satisface:*

(i) S es un polinomio de grado $\leq k$ *en cada subintervalo* (x_i, x_{i+1}) *, i = 0(1) (n - 1).*

(ii) $S \in C^{k-1}[a, b]$

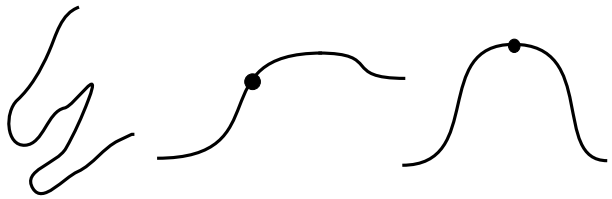

Figura 3.6: Continuidad entre segmentos de curva. De izquierda a derecha: continuidad de orden 0, continuidad de orden 1, continuidad de orden 2.

Continuidad entre segmentos de la curva. Es necesario establecer unas condiciones de continuidad paramétrica para asegurar que la transición entre segmentos de curva sean suaves (ver figura 3.6). El grado de continuidad se refiere al número de veces que puede ser derivada la ecuación obteniendo una función contínua:

- Continuidad de posición C^0. La tangente a la curva en el punto es igual por la izquierda y por la derecha. Sean *P* y *Q* dos curvas, se dice que existe continuidad C^0 si $P(1) = Q(0)$.
- Continuidad de inclinación C^1. La tangente es contínua en todos los puntos de la recta, pero no en la curvatura. Sean *P* y *Q* dos curvas, se dice que existe continuidad C^1 si $P^{'}(1) = Q^{'}(0)$.
- Continuidad de curvatura e inclinación C^2. La tangente y la curvatura es contínua en todos los puntos. Sean *P* y *Q* dos curvas, se dice que existe continuidad C^2 si $P^{''}(1) = Q^{''}(0)$.

Se distinguen varios tipos de Splines:

- Splines cúbicos naturales.
- B-Splines
    - Splines no racionales y uniformes.
    - Splines racionales y no uniformes.

- Splines de Catmull-Rom.
- Beta-Splines.

Splines cúbicas naturales Como su nombre indica, los polinomios en cada uno de los intervalos son cúbicos. Suponemos *n + 1* puntos de control, lo que implica *n* secciones de curva, por lo que el número de parámetros será *4n*.

Se necesitan *n* polinomios cúbicos, para cada coordenada $(p_i(u), q_i(u))$ en $[u_i, u_{i+1}]$. El grado de continuidad es C^2, es decir entre dos curvas adyacentes, la primera y segunda derivada en la frontera han de ser iguales:

- Cada polinomio coincide con la función en ambos extremos (C^0).

$$p_i(u_i) = x_i \; ; p_i(u_{i+1}) = x_{i+1}; i = 0, 1, ..., n-1$$

2n condiciones

- La derivada primera (de los polinomios) es contínua en los puntos de control (C^1)..

$$p_i^{'}(u_i + 1) = p_{i+1}^{'}(u_{i+1}),\ i = 0, 1, ..., n-2$$

n-1 condiciones

- La derivada segunda (de los polinomios) es contínua en los puntos de control (C^2).

$$p_i^{''}(u_i + 1) = p_{i+1}^{''}(u_{i+1}),\ i = 0, 1, ..., n-2$$

n-1 condiciones

En total se tienen *4n-2* condiciones. Se trata de un sistema compatible indeterminado por tener *4n* incógnitas y *4n-2* ecuaciones. Para que el sistema sea compatible determinado se necesitan dos ecuaciones más. Existen varias formas de solucionar este problema, algunas condiciones son:

- Spline sujeta. En este caso la pendiente es conocida en ambos extremos.

- Aproximación parabólica en los extremos.
- Aproximación cúbica en los extremos.
- Comportamiento periódico.

En el caso de las splines cúbicas naturales se opta por tomar como condición extra que la derivada segunda en los extremos es nula: u_0, un. Este tipo de curvas splines presentan dos inconvenientes muy importantes:

- Redibujar la curva supone un coste computacional elevado, puesto que la inversión de matrices es costosa.
- Modificar un punto de control supone redibujar toda la curva (mayor coste computacional) y no se tiene control local sobre ella.

Splines de Hermite. Principalmente se caracterizan por tener una tangente específica en cada punto de control y por permiir control local. La expresión matemática de las splines de Hermite es: para cada subintervalo entre dos puntos (p_0, p_1) con tangente en el punto incial m_0 y en el punto final m_1, el polinomio se define como:

$P(u) = (2u^3 - 3u^2 + 1)P_0 + (u^3 - 2u^2 + u)m_0 + (-2u^3 + 3u^2)P_1 + (u^3 - u^2)m_1; u \in [0, 1]$

El principal inconveniente que presentan es que, suele ser más útil generar los valores de las pendientes de forma automática. Existen varias técnicas para definir los valores de las tangentes que comparten los vecinos en cada subintervalo:

- Splines cardinales.
- Splines Catmull - Rom.
- Splines Kochaneck - Bartles.

Splines Cardinales y Catmull-Rom. Los polinomios empleados son del tipo Hermite, pero se difiere en que las tangentes en las fronteras de cada sección de curva son calculadas en base a las coordenadas de los puntos de control adyacentes. Aparece el término de

"parámetros de tensión", que son los que permiten ajustar más o menos la curva a los puntos de control. Para cada sección de curva se tienen las siguientes condiciones:

$P_0 = P_i$

$P_1 = P_{i+1}$

$P_0' = \frac{1}{2}(1-t)(P_{i+1} - P_{i-1})$

$P_1' = \frac{1}{2}(1-t)(P_{i+2} - P_i)$

Como se puede observar, las pendientes en los puntos de control P_i y P_{i+1} son proporcionales a las cuerdas formadas por $P_{i-1}\ P_{i+1}$, y $P_i\ P_{i+2}$.

Siendo t el parámetro de tensión, si se toma $t < 0$ la curva queda menos tensa alrededor de los puntos $P_i\ P_{i+1}$, y por el contrario quedará más tensa si se toma $t > 0$.

La expresión matricial de este tipo particular de splines es:

Curvas de Bézier. Fueron desarrolladas por Pierre Bézier para Renault. Se basan en aproximación, siendo esta la razón principal por la que son más utilizadas en los sistemas CAD que las splines cúbicas.

A medida que aumenta el número de puntos, también lo hace el grado de la curva.Para construir una curva Bézier con continuidad C^2 es necesario $n = 3$, siendo la ecuación paramétrica:

$$P(t) = \sum_{i=0}^{n} P_i f_i(t), 0 \leq t \leq 1$$

Las funciones $f(t)$ son polinomios de Bernstein de la forma:

$$f_i(t) = B_{i,n}(t)\binom{n}{i}(1-t)t^{n-i} y 0 \leq t \leq 1$$

Los puntos $P(0), ..., P(1)$ que determinan una curva de Bézier, se denominan puntos de control y la poligonal que los une se llama polígono de control de curva.

Veamos las propiedades de la curva:

- Si los puntos P_i están alineados, la curva será una recta. La curva es linealmente recta.
- Comienza en P_0 y termina en P_{n-1}.
- El vector tangente a la curva $P(t)$ en el punto P_0, tiene la dirección del vector $\overrightarrow{P_0, P_1}$.

- El vector tangente a la curva $P(t)$ en el punto P_n, tiene la dirección del vector $\overrightarrow{P_{n-1}, P_n}$.
- La modificación de un punto de control afecta a toda la curva definida.

Curvas B-Splines. Las curvas de Bézier tienen algunas desventajas, entre las que destacan:

- El grado de la curva es dado por el número de los puntos de control.
- No posee control local.

Para solventar este problema se pueden utilizar varios splines de Bézier e ir uniéndolos, el problema es que cuando el número de splines de Bézier utilizados es elevado, los cálculos se complican. Otra posible solución es emplear las B-splines que permiten definir de una vez todas las divisiones de curvas que formarán la curva global.

La ecuación que define la B-Spline es:

$$P(t) = \sum_{i=0}^{n} N_{i,k}(t) P_i$$

Se determina la curva compuesta por varios tramos que son splines cúbicas. Para poder parametrizar de una sola vez la curva global, es necesario un intervalo de definición del parámetro. Las ecuaciones de cada intervalo están influenciadas tan sólo por k vértices del polígono de control, $2 \leq k \leq n+1$. Cada P_i son los $n+1$ vértices del polígono y $N_{i,k}(t)$ son las funciones mezcla:

$N_{i,k}(t) = \frac{(t-x_i)N_{i,k-1}(t)}{x_{i+k-1-x_k}} + \frac{(x_{i+k}-t)N_{i+1,k-1}(t)}{x_{i+k}-x_{k+1}}$

Al aumentar k aumenta también el grado de los polinomios, siendo k el orden de la curva que cumple:

$$2 \leq k \leq n+1$$

La secuencia de valores de x se denominan nodos y definen una partición del intervalo en el que varía el parámetro. De esta forma se puede ajustar la región de influencia de cada punto de control. Al conjunto de nodos se llama vector de nodos y cumple:

$$x_k \leq x_{k+1}$$
$$0 \leq x \leq n + k$$
$$t_{min} = x_{k-1}$$
$$t_{max} = x_{k+1}$$

El vector de nodos permite delimitar la zona de influencia de cada punto de control relacionando la variable paramétrica t con los puntos de control P_i.

Se habla de B-Spline uniforme y periódico cuando todos los elementos del vector de nodos están equiespaciados. Esta configuración es la más utilizada en curvas cerradas, la curva no interpola a ninguno de los puntos de control y para crear una curva cerrada es necesario repetir los $k-1$ puntos desde el principio al final.

Se dice que un B-Spline es no periódico cuando los elementos del extremo del vector de nodos son iguales y los del centro se mantienen equiespaciados. En este caso la curva sólo interpola a los puntos extremos y coincide con la curva de Bezier cuando el orden es máximo, $k = n + 1$.

Se dice que el B-Spline es no uniforme cuando el vector de nodos no satisface ninguna de las condiciones anteriores.

	Grado polinomio	Interpola	Carácter	Continuidad
Lineal	1	Si	Local(2)	C^0
Bézier	$n-1$	No (sólo extremos)	Global(n)	C^∞
Spline Local	3	Si	Local(4)	C^1
Spline Global	3	Si	Global	G^2
B-Spline	Ajustable	No	Local (Ajustable)	Ajustable
B-Spline (Cúbico)	3	No	Local(4)	G^2

Cuadro 3.1: Tabla comparativa de diferentes curvas.

El tipo de curvas que se suelen utilizar en la animación no lineal (NLA) son las *splines* ya que permiten realizar menos operaciones que con con otros métodos de interpolación. Dentro de las curvas splines una de las más utilizadas es la b-spline ya que asegura la continuidad con otras curvas b-spline.

3.3.2.2. Técnicas avanzadas

Se trata de técnicas más realistas empleadas en la animación 3D. Si se desean obtener animaciones fieles a la realidad hay que tener en cuenta las leyes físicas de la naturaleza.

Cinemática. Se basa en el estudio de los movimientos independientemente de las fuerzas que lo producen, se distinguen dos tipos: cinemática directa (Foward Kinematics) y cinemática inversa (Inverse Kinematics).

Cinemática directa. El método de cinemática directa tiene como objetivo representar la localización de un objeto en el espacio respecto a un sistema de referencia determinado. La posición final del modelo se determina especificando cada uno de los ángulos de sus articulaciones, es decir, hay que actuar sobre cada punto y producir un movimiento sobre su eje. La cadena cinemática se considera abierta y formada por eslabones unidos por articulaciones en la que el movimiento se transmite de padres a hijos.

Método de Denavit-Hartenberg. Este método es muy empleado en aplicaciones industriales (robótica) y funciona bien con modelos cinemáticos sencillos. No se emplea en animación de personajes digitales debido a su elevada complejidad.

Cinemática inversa. El uso de la cinemática inversa permite reducir efectos no deseados en las animaciones. Antes de comenzar con la animación es necesario definir la estructura del objeto de forma jerárquica, típicamente esta estructura se puede ver como un árbol donde se comienza por el nodo principal (raíz) y se contínua por los nodos que dependen de él (hojas). Al revés que en el caso anterior, lo que se especifica es el punto final al que se desea llegar y el programa será el encargado de generar automáticamente los ángulos de las articulaciones implicadas en el movimiento. Existen tres elementos esenciales para comprender el

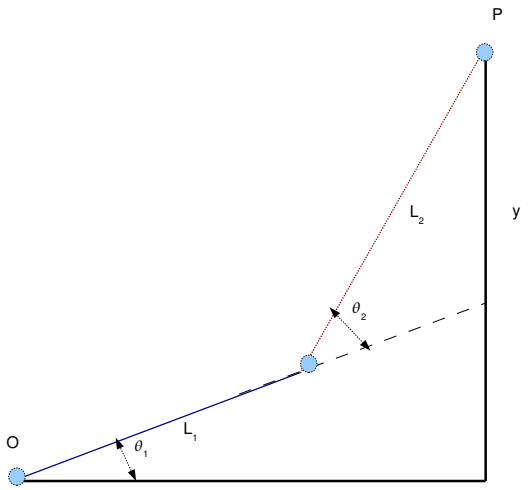

Figura 3.7: Posicionamiento de los huesos y ángulo de rotación en un espacio 2D.

mecanismo de cinemática inversa: articulaciones (joins), grado de libertad (degrees of freedom) y nivel de rotación (rotation range). Las articulaciones poseen características físicas que permiten describir el comportamiento cuando se efectúa el movimiento, el grado de libertad se refiere al número de direcciones en las que se puede mover una articulación y el nivel de rotación se refiere a los grados que una determinada articulación puede rotar.

El principal inconveniente que presenta la cinemática inversa, es que el movimiento es más complejo, ya que existen diferentes formas de rotar los elementos entre sí para obtener el movimiento final deseado. Un ejemplo claro es el caso de la rodilla, que puede girar en un único sentido. Por esta razón se suelen emplear restricciones de rotación, de forma que el software pueda identificar los movimientos prohibidos. En general, existen dos formas de abordar el problema de la cinemática directa: analíticamente y mediante métodos iterativos.

Método analítico. Supongamos dos huesos, el primero de longitud L_1 y ángulo de rotación θ_1, y el segundo de longitud L_2 y ángulo de rotación θ_2 y en un espacio 2D.

El origen del hueso 2, se obtiene de la siguiente forma

$$O_2 = (L_1 cos\theta_1, L_1 cos\theta_2) \text{ (Ec. 1)}$$

La posición final en el punto P vendrá dada por la ecuación:

$$P_x = L_1 cos\theta_1 + L_2 cos(\theta_1 + \theta_2)$$
$$P_y = L_1 sen\theta_1 + L_2 sen(\theta_1 + \theta_2) \text{ (Ec. 2)}$$

El objetivo es calcular el ángulo de rotación de los dos huesos (θ_1, θ_2) necesario para alcanzar el punto P.

$$\begin{aligned} x^2 + y^2 &= (L_1 cos\theta_1 + L_2 cos(\theta_1 + \theta_2)^2 + (L_1 sen\theta_1 + L_2 sen(\theta_1 + \theta_2)^2 \\ &= L_1^2 cos^2\theta_1 + L_2^2 cos^2(\theta_1 + \theta_2) + 2L_1 L_2 cos\theta_1 cos(\theta_1 + \theta_2) + L_1^2 sen^2\theta_1 + \\ &\quad + L_2^2 sen^2(\theta_1 + \theta_2) + 2L_1 L_2 sen\theta_1 sen(\theta_1 + \theta_2) \end{aligned} \tag{3.1}$$

$$L_1^2 cos^2\theta_1 + L_1^2 sen^2\theta_1 = L_1^2$$
$$L_2^2 cos^2(\theta_1 + \theta_2) + L_2^2 sen^2(\theta_1 + \theta_2) = L_2^2$$

Sustituyendo por lo anterior queda:

$$\begin{aligned} x^2 + y^2 &= L_1^2 + L_2^2 + 2L_1 L_2 cos^2\theta_1 cos\theta_2 - 2L_1 L_2 cos\theta_1 sen\theta_1 sen\theta_2 + \\ &\quad + 2L_1 L_2 cos\theta_1 sen\theta_1 sen\theta_2 + 2L_1 L_2 sen^2\theta_1 cos\theta_2 \\ &= L_1^2 + L_2^2 + 2L_1 L_2 cos\theta_2 (cos^2\theta_1 + sen^2\theta_1) \\ &= L_1^2 + L_2^2 + 2L_1 L_2 cos\theta_2 \end{aligned} \tag{3.2}$$

La ecuación final queda tal como sigue:

$$x^2 + y^2 = L_1^2 + L_2^2 + 2L_1 L_2 cos\theta_2 \text{ (Ec. 3)}$$

A continuación se obtienen los ángulos θ_1 y θ_2

$$cos\theta_2 = \frac{x^2 + y^2 - L_1^2 - L_2^2}{2L_1 L_2} \text{ (Ec. 4)}$$

El ángulo θ_2 se calcula aplicando el arcocoseno:

$$\theta_2 = arccos(\frac{x^2 + y^2 - L_1^2 - L_2^2}{2L_1 L_2}) \text{ (Ec. 5).}$$

El ángulo θ_1 se calcula a partir de la relación existente con los ángulos θ_3 y θ_4. Observando la figura 3.8, se aprecia que $\theta_1 = \theta_3 - \theta_4$, por lo que en primer lugar es necesario averiguar el valor de los ángulos citados:

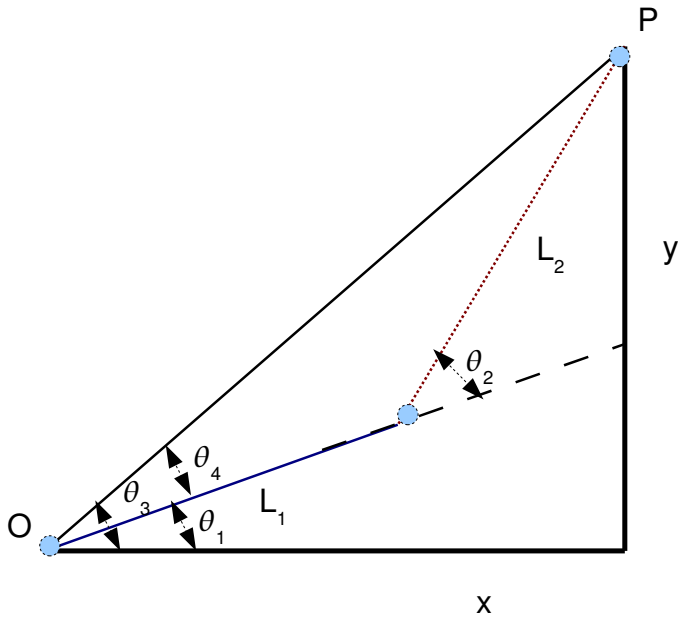

Figura 3.8: Ángulos de rotación de los huesos 1 y 2.

$$tg\theta_3 = \frac{y}{x}$$
$$tg\theta_4 = \frac{L_2 sen\theta_2}{L_2 cos\theta_2 + L_1}$$

Aplicando la fórmula $tg(a-b) = \frac{tg(a)-tg(b)}{1+tg(a)tag(b)}$ y sustituyendo *a* por θ_3 y *b* por θ_4 se obtiene:

$$tg\theta_1 = \frac{tg\theta_3 - \theta_4}{1+tg\theta_3 tg\theta_4} = \frac{\frac{y}{x}\frac{L3sen\theta_2}{L_2cos\theta_2+L_1}}{1+\frac{x}{y}\frac{L_2sen\theta_2}{L_2cos\theta_2+L_1}} = \frac{\frac{y(L_2cos\theta_2+L_1)-x(L_2sen\theta_2)}{x(L_2cos\theta_2+L_1)}}{\frac{x(L_2cos\theta_2+L_1)+y(L_2sen\theta_2)}{x(L_2cos\theta_2+L_1)}} = \frac{y(L_2cos\theta_2+L_1)-x(L_2sen\theta_2)}{x(L_2cos\theta_2+L_1)+y(L_2sen\theta_2)}$$

Finalmente, utilizando la arcotangente se obtendrá el ángulo θ_1 :

$$\theta_1 = arctg(\frac{y(L_2cos\theta_2+L_1)-x(L_2sen\theta_2)}{x(L_2cos\theta_2+L_1)+y(L_2sen\theta_2)})$$

El método analítico resuelve el problema de la cinemática inversa con gran exactitud, pero su principal inconveniente es la complejidad computacional, pudiendo llegar a ser inabordable en modelos complejos.

Cyclic Coordinate Descent. Para resolver el problema computacional que presenta el método analítico para resolver la cinemática inversa se suele utilizar el algoritmo CCD que

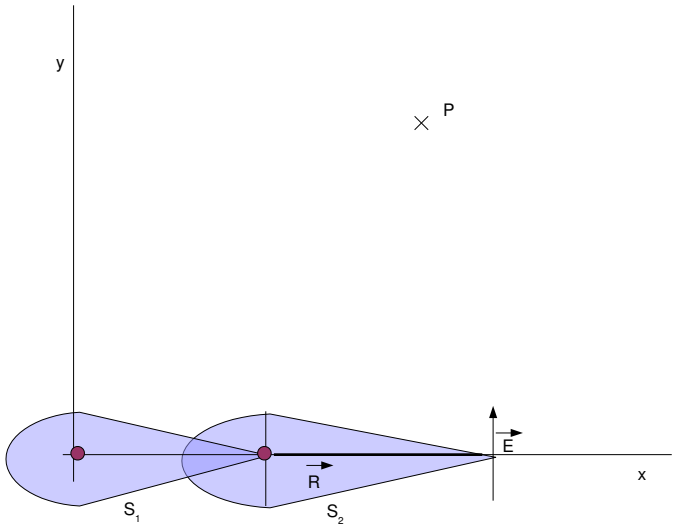

Figura 3.9: Algoritmo CCD. Posición incial.

se caracteriza por ser iterativo y ofrece la ventaja de minimizar el error en cada instante de un ángulo determinado de la cadena jerárquica. La forma de cálculo que utiliza CCD consiste en comenzar por el último elemento de la cadena e ir hacia atrás ajustando cada articulación. El ciclo se repetirá hasta que el efector esté a una distancia cercana de la posición final o bien se alcance un número determinado de iteracciones.

Con el fin de facilitar la comprensión del algoritmo CCD veamos un ejemplo. Sea un esqueleto formado por dos huesos S_1 y S_2 de 3 unidades y 4 unidades respectivamente. Se desea alcanzar el punto P sabiendo que la distancia mínima es 0,1 y el número máximo de iteracciones es 500.

Hay que calcular los ángulos θ_1 y $\theta : 2$ así como la distancia a la que se queda del punto P (5,5) en la primera iteracción sabiendo que:

$$\overrightarrow{RE}.\overrightarrow{RD} = |RE| \,.\, |RD| \, cos\theta \; \overrightarrow{RE}.\overrightarrow{RD} = RE_x RD_x + RE_y RD_y$$

Ahora aplicamos estas fórmulas con los datos dados.
$\overrightarrow{RE} = (4, 0)$
$\overrightarrow{RD} = (5 - 3, 5 - 0) = (2, 5)$

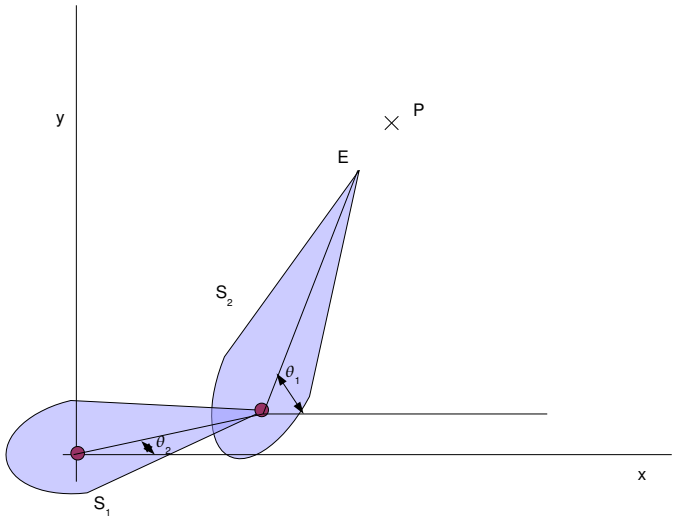

Figura 3.10: Algoritmo CCD. Primera iteracción.

$\left|\overrightarrow{RD}\right| = \sqrt{2^2 + 5^2} = \sqrt{29}$

$\theta_1 = arcocos\frac{4*2+0*5}{4*\sqrt{29}} = 68,2$

A continuación se calcula RE_x y RE_y.

$RE_x = 3 + 4cos\theta_1 = 4,48$

$RE_y = 0 + 4cos\theta_1 = 3,71$

$\overrightarrow{RE} = (4,48, 3,71)$

$\overrightarrow{RD} = (5,5)$

$\theta_2 = arcocos\frac{RE_x RD_x + RE_y RDy}{|RE||RD|}$

$RE_x = 4,48; RD_x = 5; RE_y = 3,71; RD_y = 5$

$|RE| = \sqrt{4,48^2 + 3,71^2} = 5,82$

$|RD| = \sqrt{5^2 + 5^2} = \sqrt{50}$

$\theta_2 = arcocos\frac{4,48*5+3,71*5}{\sqrt{50}*5,82} = 8,1$

Para calcular la distancia a la que se queda del punto P, primero es necesario calcular las coordenadas de E.

$E_x = 3cos\theta_2 + 4cos(\theta_1 + \theta_2)$

$E_y = 3sen\theta_2 + 4sen(\theta_1 + \theta_2)$

$E_x = 3,92$

$E_y = 4,31$ Distancia del punto P : D = $\sqrt{(5-3,92)^2 + (5-3,31)^2} = 1,28$

En la primera iteracción del algoritmo CCD, la distancia respecto al punto P es 1,28 , como es mayor que la distancia mínima (0,1), habría que repetir este ciclo hasta obtener los resultados deseados.

3.3.2.3. Otras técnicas

Motion Capture. Esta técnica consiste en grabar los movimientos realizados por un actor real y almacenarlos en un fichero de coordenadas 3D, para ser procesados por un computador con el fin de obtener una representación tridimensional. Los puntos que se toman como referencia son las áreas del actor que mejor representan el movimiento de diferentes partes del mismo, por ejemplo en un ser humano se toman las articulaciones como puntos de referencia. Existen diferentes técnicas para capturar el movimiento, siendo los más extendidos los sistemas ópticos, magnéticos y mecánicos.

En los sistemas ópticos se pueden distinguir dos tecnologías: sistemas reflectantes (sistemas pasivos) y sistemas basados en LEDs (sistemas activos). Generalmente se utilizan entre 20 y 70 marcas y entre 2 y 6 cámaras. Antes de comenzar la captura es necesario calibrar las cámaras y una vez comenzada la captura, las cámaras obtienen la posición XY de cada marca produciendo un flujo en 2D. El siguiente paso es el convertir este flujo 2D en posiciones 3D (XYZ) mediante el uso de un software adecuado. Una de las ventajas que presenta este tipo de sistemas frente a los magnéticos es que ofrece al actor una mayor libertad de movimiento, pudiendo obtener animaciones más complejas. Por contra, un inconveniente que presentan es que los sistemas ópticos sólo dan información sobre la posición y requieren tareas de edición bastante complejas, con el fin de corregir defectos tales como la oclusión. Otros inconvenientes que presentan los sistemas ópticos son:

- No producen datos en tiempo real.
- Coste elevado.
- Sensibles a la luz y la refracción.

Los sistemas magnéticos se basan en el uso de sensores con el fin de medir el campo magnético creado y se suelen utilizar en animaciones donde no son necesarios movimientos demasiados complejos. Los principales elementos de los que se compone son una serie de unidades de control electrónico y un ordenador conectado al sistema y que contiene el software necesario para poder comunicarse con los dispositivos. Una ventaja importante que presentan estos sistemas es que permiten obtener información en tiempo real, ya que permite comprobar si la animación es correcta, y en caso contrario realizar los cambios pertinentes. Pero por contra, el número de frames por segundo que captura es bajo (15 - 120 frames por segundo), tan sólo permite operar con un actor al mismo tiempo, tiene una gran sensibilidad al metal, no es recomendable para animaciones que requieren movimientos rápidos y el uso de cables limita en gran medida los movimientos que puede realizar el actor.

Los sistemas mecánicos son aquellos que emplea un hardware adicional como guantes y otras estructuras colocadas en el actor y los movimientos son rastreados por hardware.

3.3.3. Lenguajes de marcas

Antes de hablar de los lenguajes de marcas conviene distinguir entre lenguaje de marcas y lenguaje de marcas generalizado. Con el primero se describen las reglas necesarias para procesar un texto, así como loas caracteres que lo componen y los de impresión. El segundo (lenguaje de marcas generalizado) permite identificar las partes lógicas del documento así como el tipo de elementos que lo forman, pero no describe ni el proceso a realizar ni la forma de visualización.

Debido al auge de aplicaciones en las que el usuario interacciona con un personaje virtual, se han desarrollado en los últimos años diferentes lenguajes de marcas con el fin de etiquetar las animaciones de los avatares. Dentro de las animaciones podemos distinguir la facial, la corporal, el diálogo y las emociones. Algunos de los lenguajes cubren todas ellas pero otros como *AML* o *CML* tan sólo satisfacen a un subconjunto. En este apartado se ha analizado como lenguaje más completo existente hoy en día en el área de las animaciones de humanoides *VHML*. También se describe *XML* y *SGML*, ya que todos los lenguajes orientados a la animación de avatares están basados en ellos.

3.3.3.1. SGML

SGML (Standard Generalized Markup Language) es un lenguaje que aunque surge con el fin de facilitar el intercambio de datos en la industria editorial , fue evolucionando a diferentes ámbitos con el fin de intercambiar información textual [17]. Es en 1986 cuando se define por la norma ISO 8879. SGML es un lenguaje cuyo objetivo es marcar y describir documentos con independencia del hardware y software utilizado, ya que se puede controlar con cualquier editor ANCII y permite estructurar un documento en base a la relación lógica de sus partes.

Las partes en las que se divide un documento SGML son tres: declaración SGML, DTD e instancia del documento. La declaración SGML es la parte formal del documento SGML, en la que se indica lo que puede estar contenido en el documento y lo que no, así como la sintaxis utilizada y los caracteres utilizados. Esta parte puede ser omitida si el emisor y el receptor utilizan una sintaxis por defecto. En la parte DTD (Document Type Definition) se definen las reglas de marcado y estructura de una clase de documentos, es decir, documentos estructurados de una manera similar y puede estar almacenado de manera externa al documento. La última parte en la que se divide un documento SGML es la instancia del mismo, donde se encuentra el contenido estructurado según lo definido en la DTD y las características establecidas en la declaración SGML.

3.3.3.2. XML

XML (Extensible Markup Language) es un metalenguaje que permite definir otros lenguajes de marcas. La versión 1.0 de XML es una recomendación del W3C desde Febrero de 1998 y está basado en SGML, se trata de un subconjunto de este último en el que se han eliminado las partes menos útiles y más complejas. XML al igual que su padre SGML es un sublenguaje y a diferencia de HTML los elementos que lo componen, pueden dar información sobre lo que contienen y no necesariamente de su estructura física o presentación.

Algunos de los objetivos [18]de XML son:

- XML debe ser utilizable directamente sobre internet
- XML debe soportar una amplia variedad de aplicaciones.

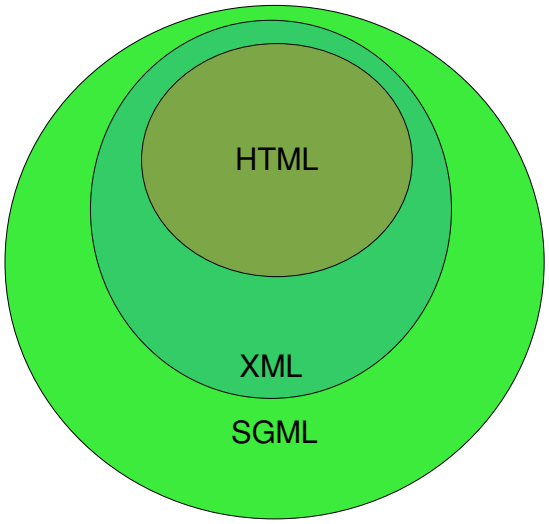

Figura 3.11: Comparación entre SGML, XML y HTML.

- XML debe ser compatible con SGML.
- Debe ser fácil escribir programas que procesen documentos SGML.
- Los documentos XML deben ser legibles por un humano y razonablemente claros.
- El diseño de XML debe ser preparado rápidamente.
- El diseño XML debe ser formal y conciso.

Un documento XML tiene dos estructuras: lógica y física. La primera de ellas engloba elementos como declaraciones o comentarios indicados mediante una determinada marca. La estructura física se refiere a las entidades que componen un documento.

Las especificaciones más relevantes de XML son:

- DTD (Document Type Definition): archivo que contiene la especificación de la estructura lógica de cada documento y define los elementos y atributos que contiene. El DTD es opcional en el XML.
- XSL (eXtensible Stylesheet Language): define el lenguaje de estilo de los documentos escritos para XML.

```
<?xml version="1.0" ?>
<ganas>
        <pose quality="90"/>
        <pose size="PREVIEW"/>
        <pose image_type="JPEG90"/>
        <pose name="DefaultAction.2817_1">
                <gbone name="Mano.L">
                        <quaternion id="quat_0">
                                0.726453542709
                        </quaternion>
                        <quaternion id="quat_1">
                                0.0411728620529
                        </quaternion>
                        <quaternion id="quat_2">
                                -0.00877589359879
                        </quaternion>
                        <quaternion id="quat_3">
                                -0.685924947262
                        </quaternion>
                        <location id="loc_0">
                                0.0
                        </location>
                        <location id="loc_1">
                                0.0

                </gbone>
        </pose>
</ganas>
```

Figura 3.12: Ejemplo de documento XML.

- XLL (eXtensible Linking Language): define el modo de enlace establecido entre los diferentes enlaces.
- XUA (XML User Agent): es usado para la normalización de navegadores XML.

Los documentos XML se pueden dividir en dos grupos: XML bien formado y XML válido. Un documento XML está bien formado cuando puede ser analizado por cualquier analizador sintáctico (*parser*) de forma correcta. Para ello tiene cumplir una serie de reglas:

- La estructura de todo documento XML ha de ser jerárquica.
- Un documento XML puede contener etiquetas vacías.
- El elemento raíz es único.
- Los valores de los atributos deben estar encerrados entre comillas (simples o dobles).
- XML es sensible a mayúsculas.
- Las estructuras y tipos de elementos deben tener asignados un nombre.

- Las marcas (etiquetas, declaraciones,..)son entendibles por el *parser* y el resto del documento comprendido entre las etiquetas son entendibles por el ser humano.
- No debe haber etiquetas aisladas.

Un documento XML es válido cuando además de cumplir con las reglas necesarias para que sea un documento bien formado, sigue una estructura y semántica definida por un DTD.

3.3.3.3. VHML

VHML (Virtual Human Markup Language) [2] es un lenguaje de marcas orientado principalmente a la representación de emociones de personajes virtuales en sistemas de interacción persona - computador [9]. Fue creado bajo la supervisión de "European Union 5Th Framwork Research and Tecnology".

VHML se diseñó con el fin de cubrir diferentes aspectos en la animación de humanoides como son la animación facial y corporal así como emocional y pretende facilitar la interacción entre humanoides con capacidad de habla y usuarios. Está basado en el lenguaje de marcas XML y contiene una serie de sublenguajes: EML (Emotion Markup Language), GML (Gesture Markup Language), SML (Speach Markup Language), FAM (Facial Animation Language), BAM (Body Animation Language), XHTML (eXtensible Hipertext Markup Language) y DMML (Dialogue Manager Markuo Language).

Dos de las aplicaciones existentes que utilizan VHML son *Metaface System* y *Mentor System*. Ambas son sistemas de diálogo persona - computador que contestan a preguntas realizadas por el usuario.

La aceptación de un lenguaje por parte del usuario depende de ciertas de características como pueden ser la simplicidad y usabilidad así como otros factores tan importantes como el de no ser ambiguo. Algunas de estas características no las cumple VHML.

VHML permite el uso de nombres en lenguajes nativos y sinónimos, como *joiful* en lugar de *happy*, por lo que cada hoja de estilo tiene que ser construida para cada lenguaje y sus sinónimos, es decir, es necesario es necesario extender el lenguaje para el uso de definiciones semánticas correctas. Esta característica implica redefinir las etiquetas [1].

La especificación de VHML no tiene en cuenta el tiempo dentro de acciones (dónde empieza, el tiempo que dura dentro de una acción,...) y es característica importante en aplicaciones en las que se requiere efectos emocionales muy sutiles.

Otra deficiencia que presenta VHML es que no considera el renderizado concurrente que es necesario si existen múltiples personajes virtuales que actúan al mismo tiempo.

En cuanto a la especificación de la personalidad es un tema bastante complejo ya que la Teoría de la Personalidad es tema de estudio en diferentes disciplinas, y trasladarlo a un personaje virtual aumenta la complejidad. En VHML existe una etiqueta *person* que puede incluir como atributo opcional a personalidad, pero los sistemas que se han desarrollado basados en VHML sólo han conseguido personalidades sencillas.

En base a estas características, a pesar de que VHML cubre las necesidades existentes a la hora de desarrollar un sistema en el que intervenga un personaje virtual su uso no se ha extendido de forma universal.

A pesar de ser un lenguaje de marcas orientado a personajes virtuales, no se ha utilizado en el sistema *GANAS* ya que:

- No gestiona el tiempo dentro de acciones.
- No gestiona el renderizado concurrente.
- Es cerrado y por lo tanto no ampliable.
- Es bastante complejo.

3.3.4. Suites de producción 3D

El diseño por computador es una técnica cada vez más utilizada por más usuarios y en más ámbitos. Según las características que requiera el usuario optará por un tipo de software 3D u otro, puesto que hay veces que no se necesita por ejemplo movimiento sino tan sólo imágenes fijas. Si lo que se pretende es tener un paquete de software que cubra todas las etapas del ciclo de producción 3D existen diferentes alternativas, como pueden ser *Maya*, *Softimage*, *3DS Max* o *Blender*.

Maya es la evolución del programa *Power Animator* de *Alias*. Ofrece gran potencia tanto para el modelado (curvas NURBS, polígonos, subdivisión) como para la animación . El código que forma el núcleo de Maya es MEL, con el cual también se pueden desarrollar scripts que cubran las necesidades que presente el usuario. Uno de los inconvenientes que presenta Maya, es que a pesar de realizar distintos intentos para hacerlo más accesible a nuevos usuarios, es menos intuitivo que otros programas de software 3D. El otro inconveniente es que se trata de un software propietario siendo su precio en torno a los 290$ en la versión para estudiantes, 3990$ en la versión Completa y unos 7600$ en la versión *Unlimited*.

Softimage[22] según la opinión de muchos animadores ofrece mejores utilidades y resultados que *Maya*, a pesar de que el motor de render que utiliza se ha ido mejorando en las sucesivas versiones ha mejorado en potencia pero aún es demasiado lento. Muchas compañías utilizan esta suite de producción 3D ya que ofrece muchas utilidades como editor de curvas, cinemática inversa, deformaciones, morphing,... La organización de los parámetros de animación pueden ser confusos por lo que requiere bastante entrenamiento antes de poder exprimir al máximo la potencia en la animación. La licencia básica cuesta en torno a los 2.000 $.

3DS Max es una de las suites de producción 3D más conocidas, ofrece multitud de funcionalidades y posibilidades tanto en el modelado como en la animación. Su uso es bastante complejo y la interfaz que ofrece no es demasiado intuitiva.

Blender se desarrolló dentro de la empresa textitNeoGeo, dedicada a la producción 3D. Tom Roosendaal, co-fundador de dicha empresa decide actualizar las herramientas 3D utilizadas y es en 1995 cuando se desarrolla la suite de producción 3D *Blender*, tal y como se conoce hoy en día. En 1998, Tom crea una nueva empresa *Not a Number* (*NaN*), con la que se desarrolla y distribuye el programa de forma libre. Blender [19] posee características muy deseables en todas las etapas del ciclo de producción 3D, algunas de ellas son:

- Gran diversidad de modelado mediante primitivas geométricas, curvas, mallas, metabolas y otras.
- La animación se puede realizar mediante cinemática inversa, y posee un editor de curvas de manera que el usuario puede ajustar movimientos no deseados.
- El renderizado puede ser tanto interno como externo, el segundo mediante *raitracer*

utilizando herramientas libres como *YafRay*.

- Motor de juegos integrado.
- Emplea como lenguaje de script *python*, ofreciendo una gran diversidad de utilidades.
- Es multiplataforma, gratuito y libre.
- Interfaz personalizable.

Al ser software libre existe una gran comunidad de usuarios que comparten experiencias y utilidades desarrolladas por ellos. Al hablar de Blender, es necesario hablar también de su API, ya que a través del lenguaje Python se pueden realizar scripts que automaticen tareas.

API de Blender. Descripción de los módulos más importantes relacionados con el posicionamiento de los huesos de un esqueleto y las acciones asociadas.

- *Object*. Este módulo permite acceder a los objetos de una escena. Para poder hacer referencia a alguno de ellos es imprescindible primero obtener la escena actual. Dentro del módulo *object* se encuentra el submódulo *Pose* que contiene las clases que permiten acceder y modificar las posiciones de los huesos (modo pose):
  - Clase *Pose*.
  - Clase *PoseBone*. Con esta clase se pueden obtener y modificar los parámetros *loc* y *quat* (localización y cuaternión) de un hueso en modo pose.
  - Clase *PoseBoneDict*, se trata de un diccionario de poses utilizado de forma interna por Blender, aunque se tiene acceso a él a través de las llamada *Pose.bones*.
- *Armature*. Este módulo de Blender permite el acceso a los esqueletos definidos en una determinada escena. Las clases que contiene permite definir características de esqueletos y huesos en modo objeto o edición son:
  - Clase *Armature*. Permite el acceso a un determinado esqueleto en modo edición.
  - Clase *Bone*. Permite el acceso a los huesos de un esqueleto.

  - Clase *BonesDict*. Diccionario de huesos de un esqueleto.
  - Clase *EditBone*. Permite la manipulación de huesos en modo edición.

- Dentro del módulo *Armature* se encuentra el submódulo *NLA* utilizado para crear acciones no lineales y cuyas clases son:
  - *Action*. Mediante esta clase se puede, entre otros, obtener el número de frames clave en una acción
  - *ActionStrip*
  - *ActionStrips*.

- *Mesh*. Contiene las clases de acceso a las mallas definidas en un determinado objeto.

Capítulo 4

MÉTODOS Y FASES DE TRABAJO

4.1. Ingeniería del software

4.1.1. Diseño Multicapa

La arquitectura seguida para el desarrollo del sistema ha sido la *Arquitectura Multicapa*. En este tipo de arquitectura se distinguen tres capas, cada una de ellas formando un subsistema:

Figura 4.1: Diseño Multicapa.

- Capa de Presentación. En esta capa se sitúan las interfaces visuales con las que el usuario interacciona, como pueden ser los frames o Páginas Web. Los elementos ubicados en esta capa necesitan a los elementos de la capa de Dominio, ya que deben mostrar información recogida de esta segunda y pasar información introducida por el usuario a las capas más bajas.

- Capa de Dominio. Es la capa donde se encuentra la lógica de la aplicación, es decir, donde se implementa la funcionalidad principal del sistema, por lo que suele ser la capa más compleja. La capa de Dominio se comunica con la siguiente capa (Persistencia)

- Capa de Persistencia. Las clases pertenecientes a esta capa son las encargadas de guardar en un mecanismo de almacenamiento las instancias pertenecientes a la Capa de Dominio.

Es importante destacar las relaciones existentes entre las tres capas de forma que el acoplamiento sea mínimo. La Capa de Presentación necesita conocer la Capa de Dominio, ya que recogen datos del usuario y los pasa a la siguiente capa y muestra los datos que provienen de la Capa de Dominio mostrándolos al usuario, pero la relación no es bidireccional, puesto que la Capa de Dominio no tiene que conocer las clases de la Capa de Presentación. Los elementos de la Capa de Dominio necesitan a los elementos de la Capa de Persistencia y viceversa, por lo tanto en este caso la comunicación si es bidireccional.

Algunas ventajas que presenta la *Arquitectura Multicapa* son:

- Reutilización debido al aislamiento de la lógica de aplicación en componentes independientes.

- Mejora del rendimiento y de la coordinación gracias a la distribución de las capas lógicas en diferentes nodos físicos o procesos.

- Facilidad en la asignación de trabajo a los miembros del equipo de proyecto, asignando en cada capa al personal especializado y posibilidad de desarrollo en paralelo.

El sistema *GANAS* puede ser accedido de diferentes modos según las necesidades del usuario. La interfaz gráfica correspondiente al módulo de administración facilita las tareas de configuración del sistema, así como la creación de poses estáticas y vídeos con el fin de tener una base de palabras en lenguaje de signos. La interfaz web permite al usuario crear frases en lenguaje de signos de forma sencilla, sin necesidad de conocimientos 3D. Por otro lado, el módulo de acceso al entorno 3D no sólo permite el renderizado de vídeos e imágenes indicados desde la interfaz gráfica, sino que también permite su ejecución desde línea de órdenes e incluso la creación de poses estáticas mediante la manipulación directa del esqueleto, para lo que se requiere un cierto conocimiento 3D.

Patrones de Diseño. El problema más destacable que presenta el *Diseño Multicapa* es el acoplamiento entre las clases, es decir, la dependencia entre los elementos pertenecientes a diferentes capas. Para solucionar este problema y mantener la cohesión se utilizan *Patrones de Diseño*.

Uno de los patrones más utilizados para evitar un alto grado de acoplamiento entre la capa de Dominio y la de Persistencia es el *Patrón Singletón*, utilizado cuando sólo se desee tener como mucho una instancia de la clase, este es el caso de *GANAS*.

Una de las características esenciales del *Patrón Singletón* es que la propia clase es la encargada de crear la instancia. En el sistema *GANAS* a esta clase se le ha llamado *GDBUtils.py* y otras características que debe cumplir son:

- El constructor de la clase debe ser privado (*init*).
- El método que devuelve una instancia de la clase debe ser estático para poder ser accedido sin necesidad de crear previamente una instancia de la clase (método *getAgente*).

CRUD es el acrónimo referido a las operaciones básicas en objetos persistentes [14], éstas son:

- Create.

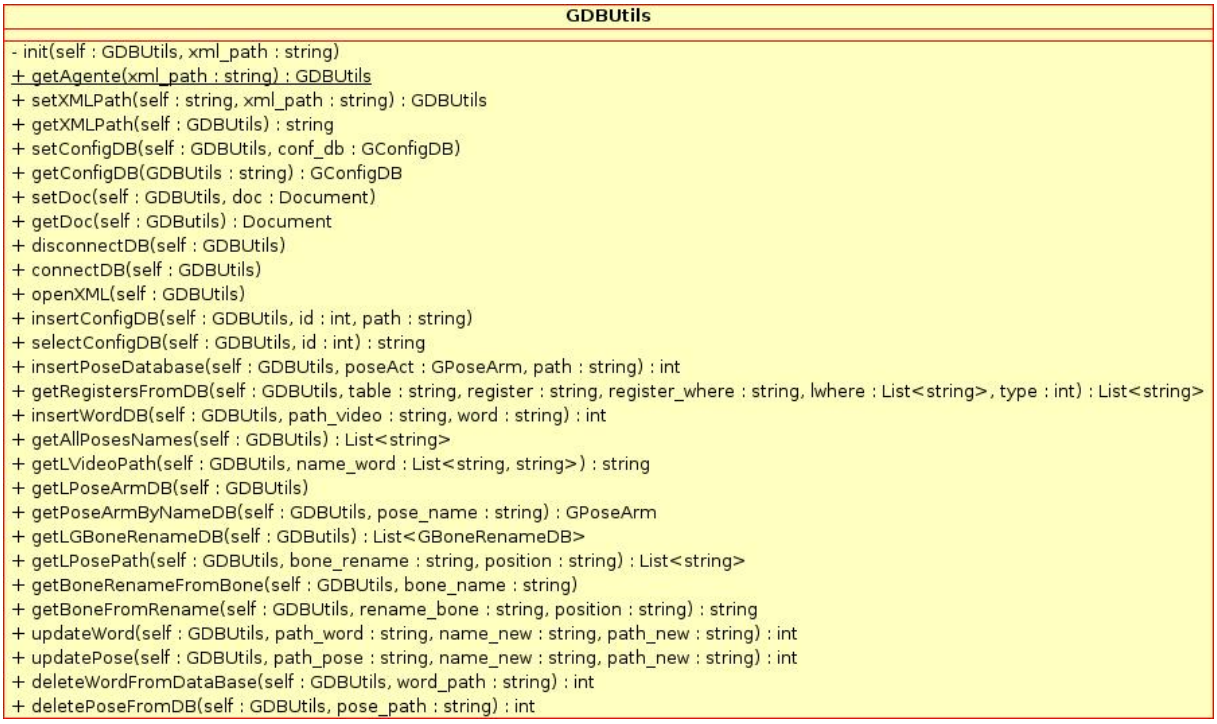

Figura 4.2: UML. Patrón Singletón.

- Read.
- Update.
- Delete.

Estas operaciones han sido implementadas en la clase *GDBUtils*, tanto las de borrado, modificación, las implicadas en el proceso de materialización (lectura de registros de la base de datos) así como las relacionadas con la desmaterialización de objetos (creación de nuevos registros de la base de datos).

4.1.2. Ciclo de vida del software

El ciclo de vida del software se puede definir como las diferentes etapas por las que pasa un sistema desde su planteamiento hasta que deja de ser útil [20], es decir desde de un problema hasta la implementación y pruebas pasando por la especificación de requisitos análisis y diseño. Existen diferentes tipos de ciclos de vida del software, como el de *cascada* o el de *prototipado*, pero el utilizado en este sistema es el modelo *espiral*.

El modelo en espiral fue desarrollado por Boehm y está basado en los modelos en cascada y de prototipado, pero se caracteriza por tener una serie de ciclos progresivos en los que al principio las tareas son más generales y que se van ampliando según se avanzan en los demás ciclos [4]. Las etapas son las mismas en cada ciclo pero las tareas que se desarrollan en un ciclo vienen dadas principalmente por el ciclo anterior. En cada iteración se especifican las alternativas. A continuación se enumeran algunas de las ventajas que presenta este modelo:

- Trata eliminar errores en fases tempranas.
- El mismo modelo se utiliza para el desarrollo del sistema y su posterior mantenimiento.
- Permite vuelta atrás.
- Puede ser aplicado en proyectos complejos.

Iteraciones realizadas en el proyecto:

- Iteración 1.
  - Objetivos.
    - Estudio de los distintos software de diseño 3D.
  - Alternativas
    - Uso de diferentes sistemas que en conjunto engloben toda la fase de producción 3D.
    - Una sola suite de producción 3D que permita todas las fases del proceso 3D.
  - Restricciones
    - Multiplataforma.
    - Necesidad de cinemática inversa.
    - Acceso a la API de la suite de producción 3D.
    - Coste reducido.
  - Resultados

        - Utilización de la suite de producción 3D Blender.

    - Planes

        - Diseño del módulo de acceso al entorno 3D.

- Iteración 2.

    - Objetivos.

        - Diseño e implementación del módulo de acceso al entorno 3D.

    - Alternativas

        - Creación de scripts que requieren que el usuario conozca Blender.
        - Creación de scripts de uso con independencia del conocimiento de Blender.
        - Creación de ambos tipos.

    - Restricciones

        - La mayoría de los usuarios no poseen conocimiento de diseño 3D.

    - Resultados

        - Diferentes scripts según las necesidades del usuario.
        - Diferentes estructuras de datos para almacenar la información que necesita Blender para renderizar y crear vídeo y que sean entendibles desde interfaces gráficas externas.
        - Almacenamiento de las estructuras en archivos de configuración.

    - Planes

        - Diseño de las estructuras de datos.
        - Diseño e implementación de clases que interpreten las estructuras de datos dentro del módulo de acceso al entorno 3D.

    - Iteración 3.

    - Objetivos.

  - Diseño de las estructuras de datos.
  - Diseño e implementación de clases que interpreten las estructuras de datos dentro del módulo de acceso al entorno 3D.

- Alternativas
  - Almacenamiento de las estructuras en base de datos.
  - Almacenamiento de las estructuras en archivos de configuración.
  - Uso de lenguaje de marcas especializado en personajes virtuales VHML.
  - Uso de lenguaje de marcas XML.

- Restricciones
  - Estructuras de datos entendibles desde aplicaciones externas.
  - Facilidad de procesamiento, eliminar complejidad.
  - Descartar datos innecesarios.

- Resultados
  - Estructuras de datos específicas según el tipo de acción.
  - Almacenamiento de las estructuras en archivos de configuración basados en XML.

- Planes
  - Diseño e implementación del script de generación de poses estáticas a partir de la maniulación directa de huesos desde Blender.
  - Diseño e implementación del script de generación de poses estáticas formadas como composición de otras poses.
  - Diseño e implementación del script de generación de vídeo a partir de poses estáticas.

- Iteración 4.
- Objetivos.

        - Diseño e implementación del script de generación de poses estáticas a partir de la maniulación directa de huesos desde Blender.
        - Diseño e implementación del script de generación de poses estáticas formadas como composición de otras poses.
        - Diseño e implementación del script de generación de vídeo a partir de poses estáticas.

    - Alternativas
        - Implementación de diferentes scripts para la generación de poses estáticas y vídeo resultado de la composición de poses estáticas.
        - Implementación de un script único para el punto anterior.
        - Uso de la información de renderizado que por defecto emplea Blender.
        - Especificar la información de renderizado según las necesidades del usuario.

    - Restricciones
        - Script de generación de vídeo y poses estáticas (a partir de la composición otras poses existentes) debe ser transparente al usuario

    - Resultados
        - Script único para la generación de vídeo y poses estáticas como composición de otras accesible de forma externa.
        - Script para la generación de poses a través de la manipulación directa de huesos accesible sólo desde Blender.
        - Almacenamiento de la información generada por Blender para reconstruir una pose o un vídeo en estructuras de datos definidas.

    - Planes
        - Diseño e implementación del módulo que contenga una interfaz gráfica que permita configurar la aplicación.

- Iteración 5.

- Objetivos.
  - Diseño e implementación del módulo que contenga una interfaz gráfica que permita configurar el sistema.
- Restricciones
  - Interfaz amigable y usable.
  - Acceso a base de datos.
- Resultados
  - Capacidad de procesamiento de las estructuras definidas en el módulo de acceso al entorno 3D.
  - Alta configurabilidad del sistema tanto a lo referido a imágenes como a vídeo.
- Planes
  - Ampliación de las funcionalidades de la interfaz gráfica de modo que permita al usuario crear vídeos e imágenes desde ella.

- Iteración 6.
- Objetivos.
  - Ampliación de las funcionalidades de la interfaz gráfica de modo que permita al usuario crear vídeos e imágenes desde ella.
- Restricciones
  - Facilidad de uso.
  - Almacenamiento y recuperación de las poses y vídeos creados mediante Base de Datos.
- Resultados
  - Creación de poses estáticas como composición de otras que pueden ser visualizadas en la interfaz.

- Creación de vídeo como composición de poses estáticas, donde el usuario establece la temporalidad de cada una de ellas.
- Utilización de Drag & Drop para aumentar la usabilidad.
- El proceso de renderizado es transparente al usuario.
- Alta configurabilidad del sistema tanto a lo referido a imágenes como a vídeo.

- Planes
  - Diseño e implementación de una interfaz gráfica basada en web para la generación de vídeos a partir de otros, de tal forma que se puedan contruir frases, expresiones o frases completas en lenguaje de signos.

- Iteración 7.

- Objetivos.
  - Diseño e implementación de una interfaz gráfica basada en web para la generación de vídeos a partir de otros, de tal forma que se puedan contruir frases, expresiones o frases completas en lenguaje de signos.

- Alternativas
  - Renderizado completo de los vídeos especificados por el usuario.
  - Reutilización de los vídeos existentes y renderizado de las partes de unión entre vídeos.
  - Restricciones
    - Facilidad de uso.
    - Minimizar el tiempo de renderizado.
  - Resultados
    - Renderizado de vídeos desde una aplicación web de forma transparente al usuario.
    - Reutilización de vídeos generados previamente.

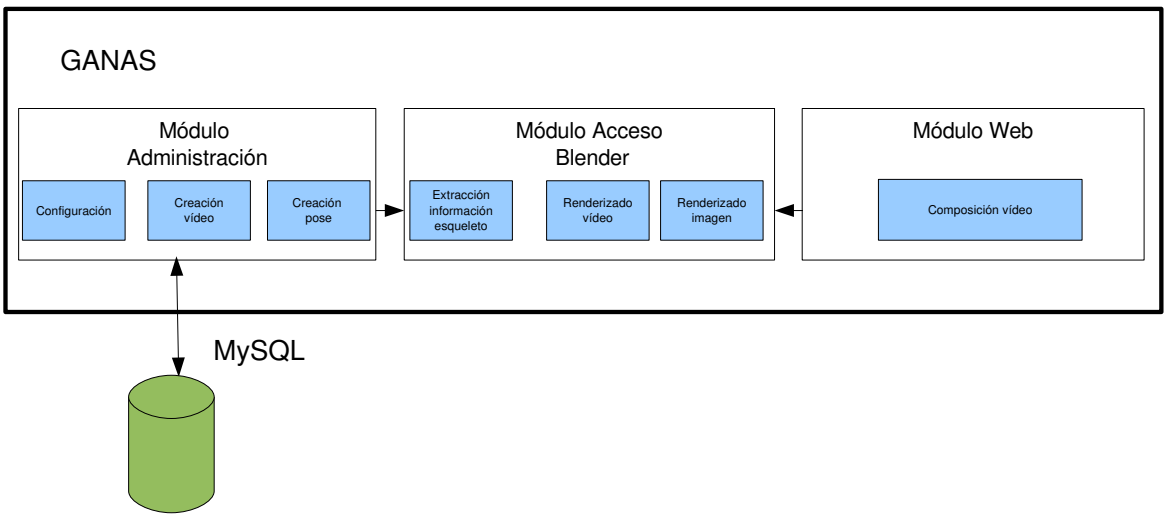

Figura 4.3: Arquitectura general del sistema GANAS.

4.1.3. Arquitectura general del sistema

Antes de definir la arquitectura del sistema, se pidió colaboración de diferentes tipos de personas con el fin de analizar la usabilidad del sistema.

La mayoría de los sistemas existentes cuyo objetivo es la generación de lenguaje de signos a través de un personaje virtual permiten que el usuario genere nuevas palabras y/o expresiones. La edición se realiza mediante el posicionamiento de los huesos de forma directa, es decir el usuario debe tener una visión 3D del personaje virtual para realizar un posicionamiento adecuado, es decir las mismas operaciones que se han de realizar si desde Blender, por ejemplo, se cambia la postura del personaje. Para comprobar si cualquier usuario puede hacer esta tarea se pidió a un conjunto de ellos que hicieran el mismo trabajo comentado anteriormente sobre un personaje virtual definido en Blender. Para que fuera lo más fiel posible se colocaron las diferentes vistas 3D para que el usuario pudiera entender el posicionamiento en un mundo 3D. El resultado fue que la totalidad de las personas que no habían trabajado previamente con aplicaciones 3D no supieron colocar al personaje de la forma que pretendían. Tras este estudio se llegó a la conclusión que la interfaz gráfica que aparece en otros sistemas no era realmente usable, puesto que los usuarios potenciales del sistema no tienen conoci-

miento en aplicaciones 3D. Tras este estudio se propusieron diferentes alternativas para que la manipulación del *avatar* fuera lo mas sencilla posible. Después de diversas pruebas con diferentes usuarios se optó por:

- Una interfaz gráfica que permita la creación de poses estáticas y vídeos de tal forma que sirvan de base para generar frases completas.
- Una interfaz gráfica que permita la generación de frases completas.

La interfaz de creación de poses estáticas y vídeos también contiene la configuración del sistema, de manera que el usuario puede adecuarlo según sus necesidades. La forma de crear estas poses y vídeos es mediante la composición de poses existentes, de tal manera que el usuario no necesita tener una percepción 3D ni manipular directamente el esqueleto del personaje virtual.

La interfaz gráfica que permite la generación de frases completas en lenguaje de signos tenía que ser muy sencilla e intuitiva, por eso se optó por una página web que muestra sólo la información necesaria para crear nuevos vídeos sin sobrecargar de información innecesaria al usuario.

También se pensó en el grupo minoritario de usuarios que conocían aplicaciones de diseño 3D, por eso se amplió la forma de generar poses estáticas mediante el uso directo de Blender. Desde éste el usuario puede manipular de forma directa los huesos y a través de un script almacenar dicha imagen junto con la información necesaria para reproducirlo en otro momento. Para este tipo de usuarios no se ha creado una interfaz gráfica externa a Blender, ya que sería reproducir la apariencia de este último.

4.2. Módulos.

La aplicación *GANAS* se divide en tres módulos principales:*módulo de acceso al entorno 3D*, *módulo de administración* y *módulo web*. La creación de tres módulos se ha creado pensando en la usabilidad del sistema, para llegar a esta conclusión se pidió la colaboración de diferentes personas. *GANAS* está pensado para ser utilizado por personas no expertas en aplicaciones 3D, por eso aunque la funcionalidad principal reside en el módulo de acceso al

Figura 4.4: Módulos que conforman la aplicación GANAS.

entorno 3D, los usuarios pueden trabajar con él de forma transparente.

El *módulo de acceso al entorno 3D* es el único que puede funcionar con independencia de los otros dos, puesto que puede ser accedido desde línea de comandos, desde la interfaz gráfica de administración o desde el módulo web, incluso contiene un script que sólo puede ser ejecutado desde Blender (generar poses mediante la manipulación directa de huesos). El módulo de acceso al entorno 3D es el núcleo de la aplicación ya que es el que contiene las funcionalidades necesarias para generar el movimiento de los huesos que componen el esqueleto del personaje virtual y renderizar las imágenes.

En el módulo de acceso al entorno 3D se distinguen dos clases principales: *GGenVideo.py* y *GCreatePose.py*. La primera de ellas está relacionada con la creación de poses estáticas y vídeo a partir de poses definidas, y la segunda tiene como funcionalidad la creación de una pose estática según la colocación de los huesos que desde Blender impone el usuario.

El *módulo de administración* contiene una interfaz gráfica que permite al usuario configurar el sistema según sus necesidades así como generar poses estáticas y vídeos del personaje virtual. Este módulo por si sólo no tiene sentido ya que para realizar el renderizado necesita interaccionar con el módulo de acceso al entorno 3D. Desde el módulo de administración el usuario puede almacenar en base de datos las rutas donde se encuentran las poses y vídeos creados, de modo que pueden ser reutilizados para crear nuevos.

El *módulo web* está diseñado con el fin de facilitar al usuario el uso del sistema y minimizar los tiempos de renderizado gracias a la reutilización de vídeos creados previamente.

4.2.1. Módulo de administración

El módulo de administración ha sido desarrollado con el fin de facilitar al usuario tareas que de otro modo podrían ser complejas:

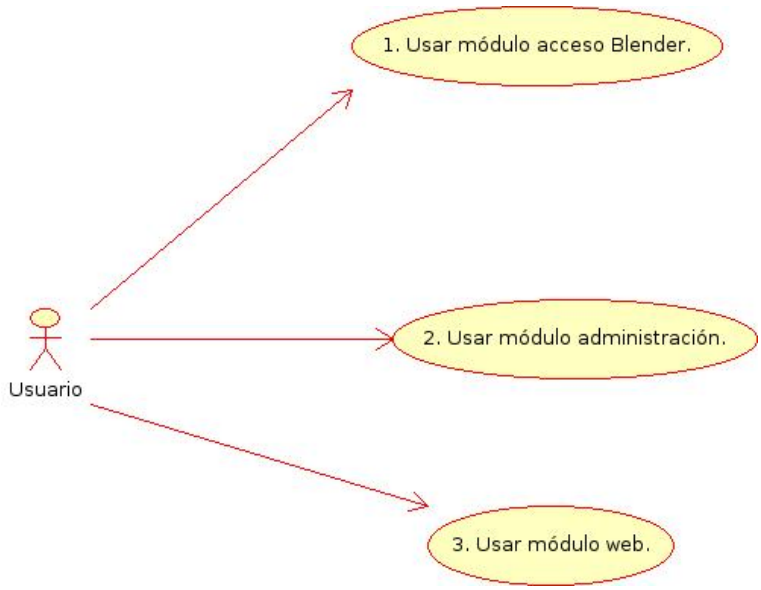

Figura 4.5: Diagrama de casos de uso general.

- Configuración de la aplicación (preferencias).
- Creación de poses estáticas.
- Creación de vídeos.
- Acceso a Base de Datos.

La configuración de la aplicación permite personalizar el sistema según las necesidades del usuario. Desde la interfaz gráfica se pueden cambiar los archivos de configuración de

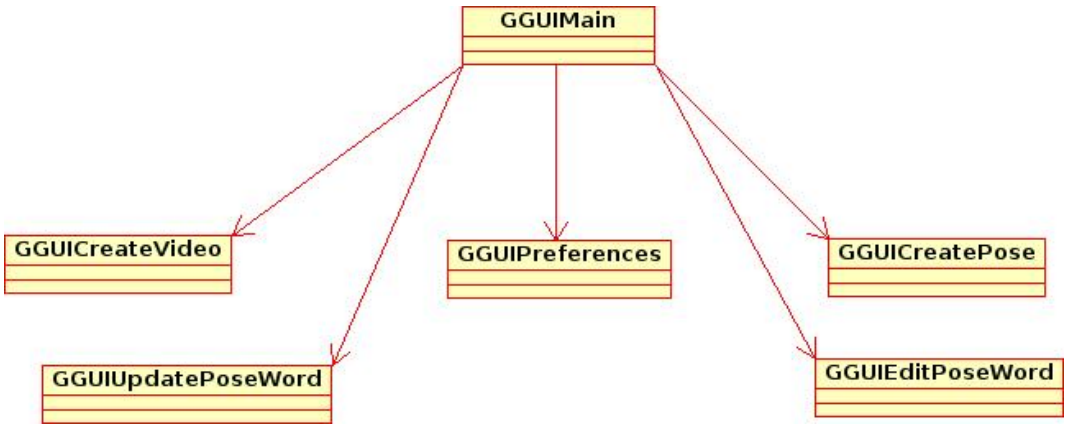

Figura 4.6: Diagrama de clases de la interfaz de administración.

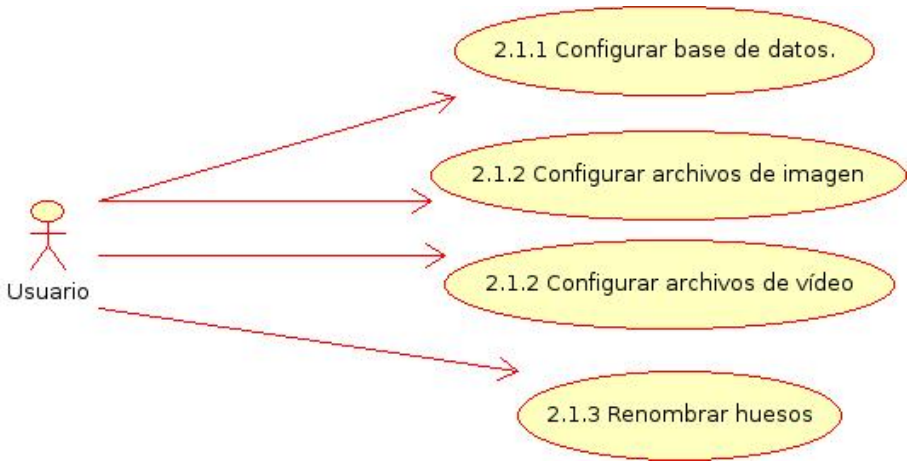

Figura 4.7: Casos de uso, configuración de la aplicación.

imagen y vídeo que posteriormente será utilizados en el renderizado. Además permite definir otras opciones como el remarcado de los huesos.

La configuración del sistema permite especificar parámetros referidos a la base de datos a utilizar, tipo de archivos de imagen y de vídeo entre otros:

- Data Base. Configuración de la base de datos. La información se almacena en un archivo de configuración llamado *databaseconfig.cnf*.
- Image. El usuario especifica el tipo, tamaño y calidad de la imagen, así como la ruta donde se almacenarár el archivo de configuración *image_config.cnf*. Otro parámetro configurable es el *remarcado de los huesos*, es decir, si los huesos implicados en una pose estática deben estar coloreados de forma diferente para resaltarlos.
- Video. Configuración del video, es decir, tipo, tamaño, frames por segundo, calidad y marcado de los huesos, así como el *path* donde se almacena el archivo de configuración del vídeo (video_config.cnf)
- Bones names. El usuario establece los nombres de los huesos con los que posteriormente se harán búsquedas para formar nuevas poses.

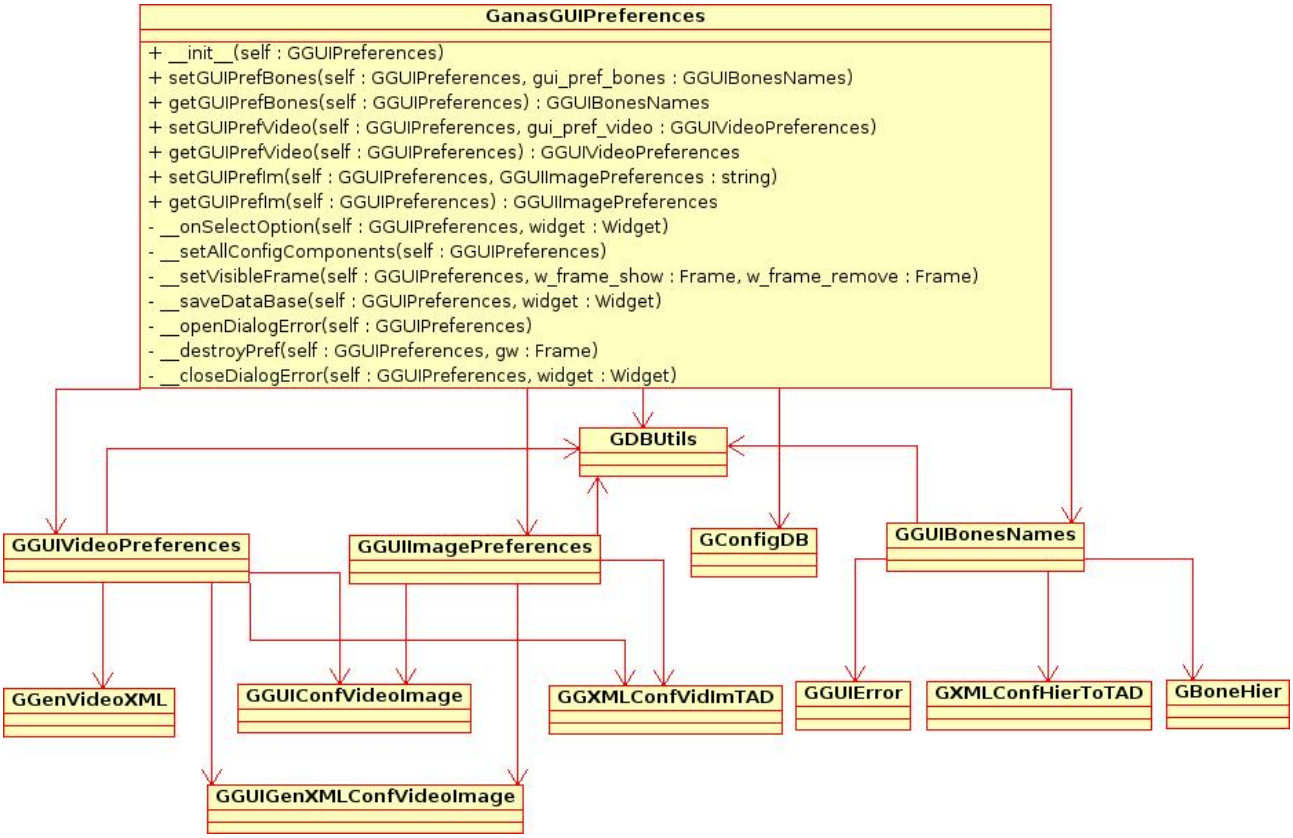

Figura 4.8: Diagrama UML de configuración de preferencias.

```
<?xml version="1.0" ?>
<ganas>
        <data_base host="localhost"/>
        <data_base user="root"/>
        <data_base password=""/>
        <data_base name="ganas"/>
</ganas>
```

Figura 4.9: Estructura XML del archivo de configuración de la base de datos.

```
<?xml version="1.0" ?>
<ganas>
        <video_config video_quality="90"/>
        <video_config video_size="PREVIEW"/>
        <video_config video_type="JPEG90"/>
        <video_config video_path="/home/vane/"/>
        <video_config video_bones_colour="rgb">
                <gmesh_colour id="rgb">
                        <colour_0 id="None"/>
                        <colour_1 id="None"/>
                        <colour_2 id="None"/>
                </gmesh_colour>
        </video_config>
</ganas>
```

Figura 4.10: Estructura XML del archivo de configuración de imagen.

Todos los archivos de configuración tienen como extensión *.cnf* y son en realidad archivos XML debido a que se ajustan a las necesidades de la aplicación y su procesamiento es rápido. El archivo de configuración de la base de datos contiene la etiqueta *data_base* y los atributos son el host, usuario, contraseña y nombre de la base de datos.

El archivo de configuración de imagen se utiliza para renderizar las poses estáticas según las características establecidas por el usuario. La etiqueta *video_config*tiene como atributos la calidad de la imagen, el tamaño, ruta donde se encuentra el archivo, y el color. Este último atributo se refiere a si se desea o no que los huesos implicados en la pose deben estar resaltados con un color diferente al que tiene el personaje. En el caso de que los huesos deban estar resaltados, el color se especificará en los atributos*colour_0, colour_1, colour_2*, referentes a la tonalidad del rojo, verde y azul respectivamente.

El archivo de configuración de vídeo posee una estructura muy similar al de imagen solo que aparece un atributo más, los frames por segundo (*fps*).

La creación de poses estáticas y vídeos ofrece diversas posibilidades. Cuando se crea una pose desde la interfaz gráfica se hace como composición de otras ya existentes, esto no

implica que se deban utilizar todos los huesos de las poses almacenadas, sino que el usuario puede elegir qué huesos de una pose determinada son los que deben estar implicados en la nueva pose.

La interfaz ofrece tres posibilidades en cuanto a la creación de poses estáticas: *explore*, *search* y *extract*. La primera opción muestra las poses existentes almacenadas en la base de datos pero en forma de árbol, así, por ejemplo para ver las poses en las que esté implicada la mano izquierda, iremos abriendo el árbol de la forma *Hand – Left*. La opción *search* permite buscar poses de dos formas diferentes:

- Buscar pose por nombre
- Buscar pose por los huesos implicados. Para realizar este tipo de búsqueda primero hay que elegir de una lista desplegable el nombre del hueso y después su ubicación (*Left*, *Rigth*, *Both* o *None*)

Todas las poses pueden ser previsualizadas y mediante *Drag & drop* (arrastrar y soltar) se añaden a la lista de edición. Si en algún momento de la edición de una pose el usuario quiere eliminar alguna de ellas, podrá hacerlo de la misma forma que las añadió a la lista de edición, pero arrastrándola al icono de la papelera.

La opción *Extract* permite al usuario buscar un vídeo almacenado en la base de datos y extraer las poses que lo componen, de forma que éstas pueden ser reutilizadas.

La creación de vídeo desde el interfaz de administración, permite componer vídeos formados por diferentes poses, de igual forma que en la creación de poses, el usuario podrá hacer búsquedas de las mismas.

Otras acciones que pueden ser realizadas desde la interfaz gráfica son las relacionadas con la base de datos. De esta forma un usuario puede saber rápidamente si una pose existe y puede ser reutilizada también puede desechar poses o vídeos cuyo resultado no era el que esperaba y modificar el nombre y ubicación de poses estáticas y vídeos.

Figura 4.11: UML. Clase para la creación de poses desde la interfaz gráfica.

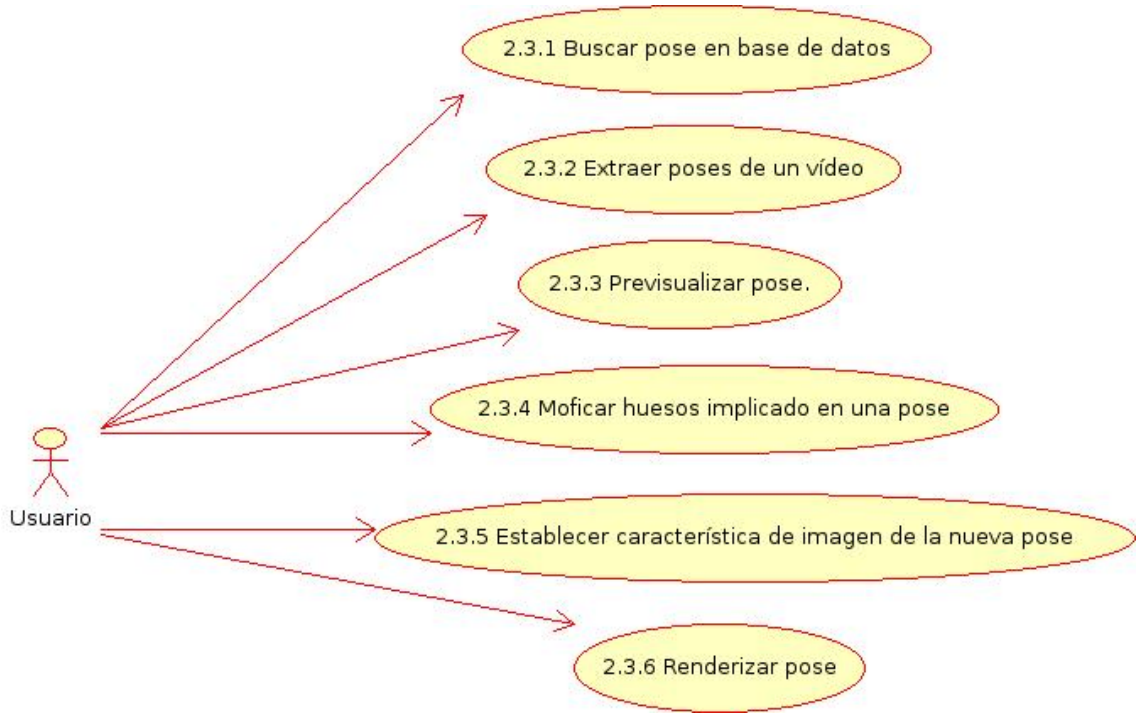

Figura 4.12: Casos de uso, generación de una pose estática.

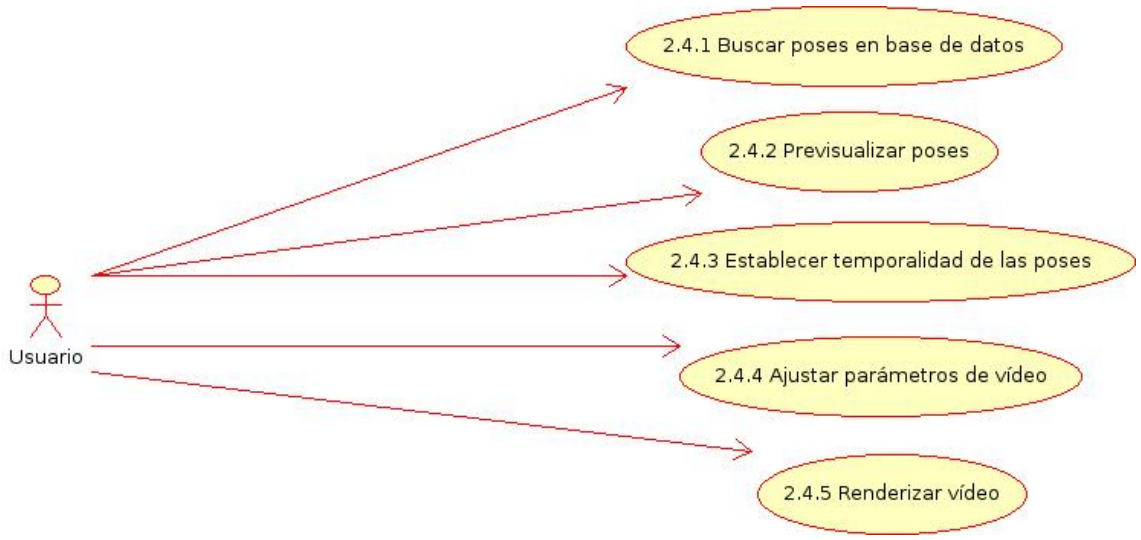

Figura 4.13: Casos de uso, generación de vídeo.

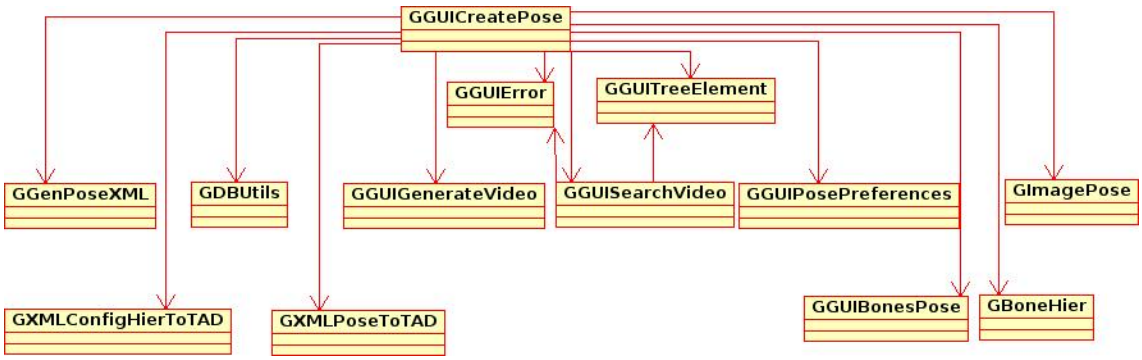

Figura 4.14: UML. Creación de poses desde la interfaz de administración.

- Buscar (pose estática o vídeo)
- Eliminar (pose estática o vídeo)
- Modificar (pose estática o vídeo)
- Almacenar (pose estática o vídeo)

El módulo de acceso al entorno 3D funciona con independencia de los otros dos, pero para llevar a cabo el renderizado de los vídeos o imágenes desde el módulo de administración o el módulo web es necesario llamar al módulo de acceso a Blender, para ello se crea un hilo de ejecución distinto al que se está utilizando. En tiempo de ejecución se hace una llamada a una clase del módulo de acceso a Blender y se crea un archivo llamado *ganasblender.cnf* en el que se escribe en cada línea información necesaria para el renderizado. Una vez que el archivo está almacenado en disco se crea un hilo de ejecución distinto cuya función es ejecutar una clase llamada *GThreadBlender.py*. El objetivo de esta clase es leer el contenido del archivo de configuración *ganasblender.cnf* y llamar al script de Blender necesario según lo pedido por el usuario (generar poses, generar vídeo) .

Las clases que forman parte de la interfaz gráfica comienzan con las siglas *GGUI* (Ganas GUI) para diferenciarlas del resto, se pueden dividir en tres que son: las que contienen ventanas o elementos gráficos, las que definen estructuras de datos y las utilizadas para generar o recuperar información almacenada en los archivos de configuración XML.

Clases relacionadas con elementos gráficos:

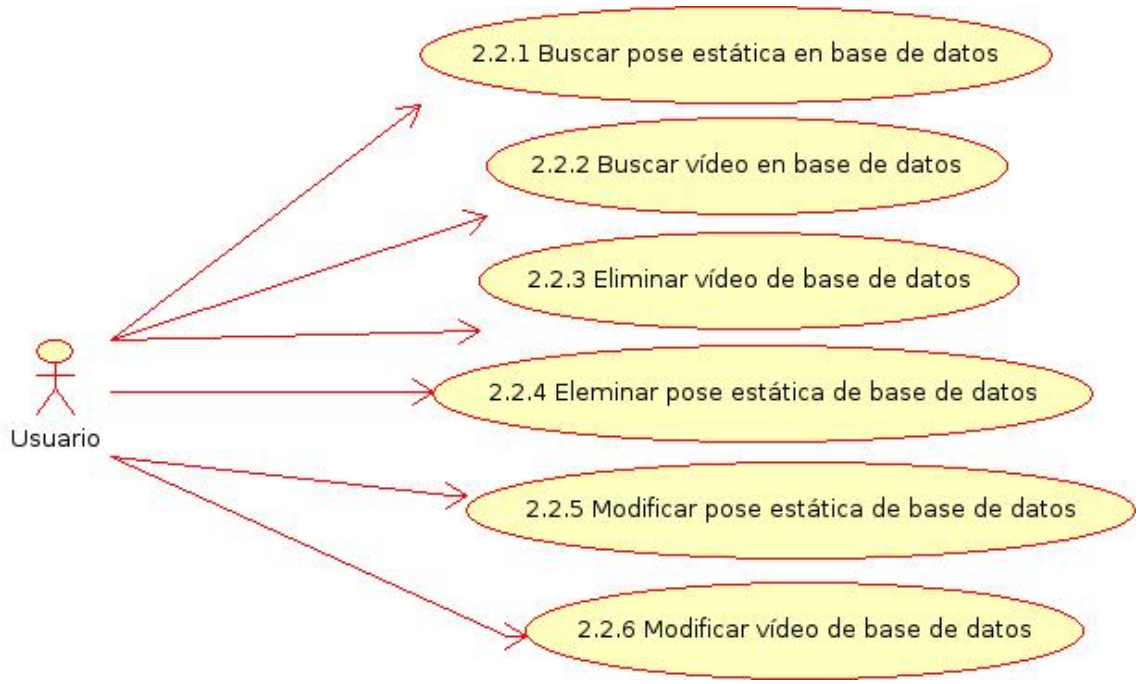

Figura 4.15: Casos de uso, acceso a base de datos.

```
/home/vane/pose1
/home/vane/blender-2.42-linux-glibc232-py24-i386/blender
/home/vane/pose1/pose1.ganas
/home/vane/render/error.error
1
/home/vane/name.ganas
/home/vane/hierarchy_config.cnf
None
ganas_render.blend
GGenVideo3.py
```

Figura 4.16: Archivo de configuración para Blender.

- *GGUIMain.py*. Es la clase principal de la interfaz gráfica. Muestra la ventana inicial de la aplicación, desde la cual se pueden acceder a las diferentes funcionalidades que oferta el sistema.

- *GGUIPreferences.py*. Clase encargada de mostrar la ventana de configuración del sistema (almacenada en un archivo .glade) y de almacenar cambios solicitados por el usuario en la configuración de la base de datos (archivo *databaseconfig.cnf*. El menú de configuración se muestra en forma de árbol plegado de forma que el usuario seleccionará la opción que desea ver o modificar .

- *GGUIVideoPreferences.py*. Carga el archivo *.glade* correspondiente a la configuración de vídeo así como la información referente a las preferencias de vídeo almacenadas en *video_config.cnf*. Si el usuario realiza modificaciones respecto a las almacenadas, la clase generará el nuevo archivo de configuración.

- *GGUIImagePreferences.py*. Clase similar a descrita en el punto anterior pero la información que muestra y/o almacena es la referente a las imágenes estáticas (archivo de configuración *image_config.cnf*.).

- *GGUIImagePreferences.py*. Clase encargada de mostrar la información referente a los huesos definidos en el archivo Blender y el renombrado que el usuario hace de los mismos para su posterior búsqueda y uso desde el módulo de administración.

- *GGUIInfo.py*. Muestra una ventana auxiliar que contiene texto informativo.

- *GGUIEditPoseWord.py*. Clase encargada de mostrar la ventana desde la cual el usuario puede eliminar y modificar poses estáticas y vídeos de la base de datos y del dio.

- *GGUIQContinue.py*. Ventana auxiliar utilizada para verificar la opción que el usuario ha elegido desde la clase anterior. Si la opción elegida es correcta la pose estática o vídeo será eliminada o modificada de la base de datos y dispositivo de almacenamiento.

- *GGUIModifyPoseOrWord.py*. Clase que muestra la ventana con información de la pose o vídeo a modificar y recoge el nuevo nombre y ruta por la que se desea cambiar. Una

modificación implica cambios tanto en la base de datos como en el lugar de almacenamiento del disco.

- *GGUIError.py*. Ventana auxiliar cuya funcionalidad es mostrar un error en forma de texto.

- *GGUIErrorBlender.py*. Ventana que muestra un error ocurrido en Blender.

- *GGUIUpdatePoseWord.py*. La funcionalidad de esta clase reside en mostrar una ventana desde la que el usuario indica la pose o vídeo que el usuario desea almacenar en base de datos.

- *GGUICreatePose.py*. Clase que muestra el marco principal para la creación de una pose estática como composición de otras creadas previamente. Permite previsualizar poses seleccionadas por el usuario.

- *GGUISelectBones.py*. Muestra los huesos implicados en la pose seleccionada por el usuario, permitiendo eliminar la influencia de huesos no deseados.

- *GGUIGenerateVideo.py*. Clase encargada de recoger la información de renderizado de vídeo introducido por el usuario a través de una ventana y lanzar un hilo secundario de ejecución en el que se llamará a Blender para la generación del vídeo.

- *GGUIPreviewFrames.py*. Clase que muestra una ventana cada vez que el usuario haga Drag & Drop de una pose previsualizada al editor de secuencias.La ventana muestra el lugar que por defecto ocupará la pose en la secuencia de edición de vídeo y recoge el frame en el que la pose comenzará su influencia según lo especifique el usuario. Si el usuario lo desea puede modificar la posición de la pose en la secuencia.

- *GGUIPosePreferences.py*. Clase que muestra una ventana cada vez que el usuario vaya a renderizar una pose, en la que se pueden cambiar parámetros relacionados con la imagen y en la que el usuario debe introducir la ruta y el nombre de la nueva pose que se generará. Es la encargada de hacer la llamada a Blender para el renderizado de una pose estática.

- *GGUICreateVideo.py*. Clase que contiene el frame principal con el que el usuario interacionará para generar vídeos a partir de poses estáticas previamente creadas.
- *GGUIQSaveVideo.py*. Una vez que el vídeo sea renderizado por Blender, esta clase muestra una ventana con la función de pedir al usuario si el nuevo vídeo se almacenará en base de datos.

Clases que definen estructuras de datos:

- *GGUIConfVideoImage.py*. Define la estructura de datos relativa a la configuración del vídeo e imagen, esto es:
  - Ruta del archivo de configuración.
  - Tamaño del archivo de imagen/ vídeo.
  - Frames por segundo (en el caso de la configuración de vídeo).
  - Tipo de imagen/vídeo.
  - Calidad.
  - Color (referente al remarcado de los huesos).
- *GBoneHier.py*. Estructura de datos definida para la lectura y escritura del archivo de configuración *bones_config.cnf*. Los datos necesarios son:
  - Nombre del hueso.
  - Nombre del hueso asociado con el que tiene restricción de tipo ik (en el caso de que exista).
  - Longitud de la cadena de cinemática inversa.
  - Nombre del hueso padre.
  - Calidad.
  - Color (referente al remarcado de los huesos).
- *GGUITreeElement.py*. Estructura utilizada para definir árboles asociados con un nombre.

- *GImagePose.py*. Clase que define la estructura necesaria para localizar en la tabla de secuencia de vídeo una pose estática y el frame en el que aparecerá. Elementos:
  - Ruta donde se encuentra el archivo de imagen.
  - Ruta donde se encuentra el archivo de configuración de la pose.
  - Posición que ocupa en la tabla de secuencia de vídeo.
  - Frame a partir del cual comienza su influencia.
- *GPoseArmDB.py*. Clase que contiene la información necesaria para almacenar o recuperar de la base de datos una pose determinada. Elementos:
  - Lista de identificadores de los huesos implicados en la pose almacenados en la base de datos.
  - Nombre de la pose.
  - Tipo de imagen.
  - Calidad.
  - Tamaño de la imagen.
- *GBoneDB*. Clase en la que se define la estructura con la información necesaria para almacenar y recuperar de la base de datos un hueso determinado. Elementos
  - Nombre del hueso.
  - Quaterion.
  - Localización.

Clases encargadas de leer y generar archivos de configuración:

- *GGXMLConfVidImTAD.py*. Recupera la información almacenada en el archivo *video_config.cnf* y lo transforma en una estructura de tipo *GGUIConfVideoImage*.
- *GGUIGenXMLConfVideoImage.py*. Realiza el proceso inverso descrito anteriormente. Transforma la información contenida en un objeto de la clase *GGUIConfVideoImage* a un archivo *video_config.cnf*.

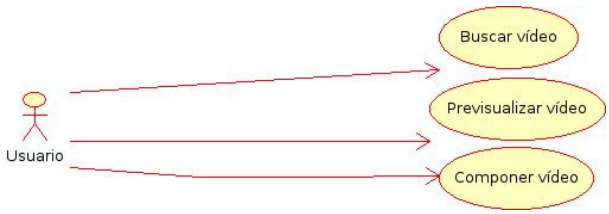

Figura 4.17: Diagrama de casos de uso del módulo web

Clase que implementa el patrón Singletón:

- GDBUtils.py

4.2.2. Módulo web

Este módulo está diseñado con el fin de facilitar al usuario la creación vídeos en lenguaje de signos a partir de otros ya existentes y almacenados en la base de datos (ver figura 4.17).

En el módulo de administración los vídeos se crean a partir de poses estáticas, por lo que es necesario renderizar el vídeo completo, pero en el módulo web, el usuario seleccionar dos o más vídeos ya creados y por lo tanto no es necesario volver a renderizar todo el vídeo, sino sólo las partes de unión entre dos vídeos para que la transición de uno a otro se haga de manera suave.

En la figura 4.18 se muestra el proceso de creación de un vídeo desde la web. Supongamos que el usuario ha seleccionado tres vídeos distintos para componerlos, la aplicación recogerá la posición final del primer vídeo, la inicial del segundo, la final del segundo y la inicial del tercero, estas posiciones están almacenadas en archivos con extensión *.pganas*.

Por cada par *posición inicial - posición final* se ordenará al módulo de Blender que renderice y cree un vídeo. Una vez obtenidos los vídeos intermedios, el sistema creará un solo vídeo final resultado de la composición de todos los vídeos (vídeo 1, vídeo intermedio 1, vídeo 2, vídeo intermedio 2 y vídeo 3).

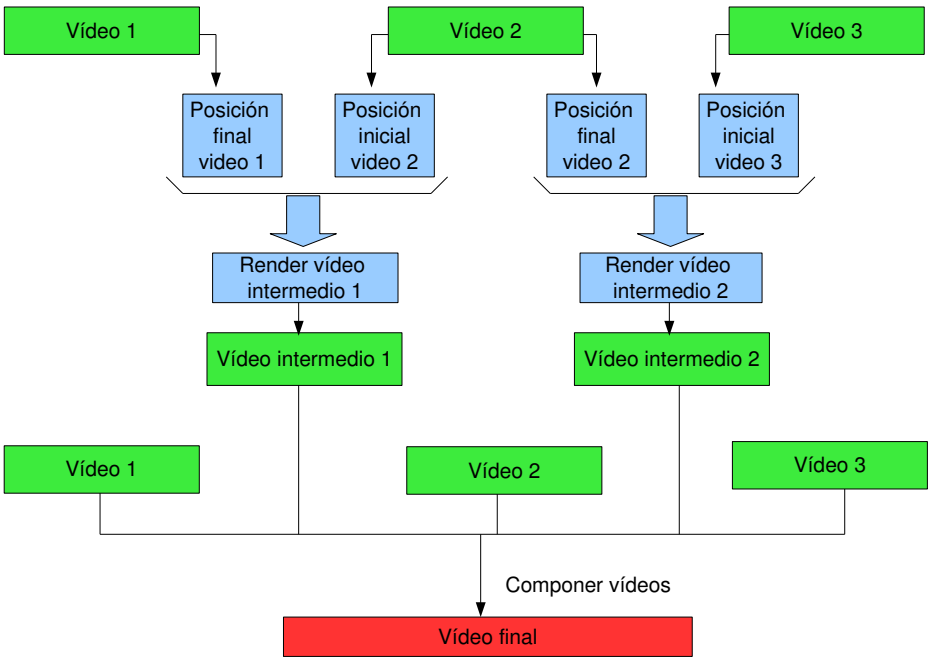

Figura 4.18: Proceso de creación de vídeo desde la interfaz web.

4.2.3. Módulo de acceso al entorno 3D

El módulo de acceso al entorno 3D contiene los *scripts* necesarios para traducir las órdenes dadas desde la línea de comandos a acciones en Blender. Se trata de un módulo independiente, pero para facilitar su utilización a usuarios no especializados se ha optado por desarrollar interfaces gráficas que interactúan con Blender de forma transparente al usuario. Las acciones principales que se han implementado se pueden resumir en dos: generar pose y generar vídeo.

Antes de poder hacer uso de las funcionalidades es necesario conocer la estructura del esqueleto con el que se va a trabajar. Esta estructura se almacena en un archivo llamado *bones_config.cnf* y se genera automáticamente mediante el script *GGenArmBonesXML.py*. Debido a que los huesos del esqueleto utilizan cinemática inversa se pueden distinguir dos casos, que el hueso sea un *ik solver* o no lo sea. Si el hueso no es un *ik solver* sólo aparecerá el nombre del hueso y el atributo *no* relacionado con la etiqueta *has_constraint* (tiene restricción). En caso contrario además del nombre del hueso, la estructura xml contendrá el atributo *yes* en la etiqueta *has_constraint*, el nombre del hueso con el que existe una restricción de cinemática

```
<armature bone_name="IKIndice.L">
        <ik_constraint has_constraint="no"/>
</armature>
<armature bone_name="PulgarNull.L">
        <ik_constraint has_constraint="yes">
                <ik_bone bone_name="IKPulgar.L"/>
                <chain leng="2"/>
                <parent_list>
                        <parent bone_name="Pulgar2.L"/>
                        <parent bone_name="Pulgar1.L"/>
                </parent_list>
        </ik_constraint>
</armature>
<armature bone_name="Tibia.R">
        <ik_constraint has_constraint="no"/>
</armature>
```

Figura 4.19: Estructura xml para la definición de restricciones de huesos.

inversa (*ik_bone*), la longitud de la cadena afectada por la cinemática inversa (*chain*) y una lista en la que se indican los nombres de los huesos que forman parte de dicha cadena.

Una pose puede ser creada de dos formas diferentes la primera de ellas es mediante el script *GCreatePose.py*, en este caso el usuario debe estar familiarizado con Blender puesto que debe interactuar directamente con él colocando los huesos a mano. La segunda forma de crear una pose es mediante la composición de otras, en este caso el usuario no necesita conocer Blender puesto que va a ser transparente a él. En ambos casos, cada vez que se genera una pose se crean dos archivos, uno de ellos es un archivo de imagen que muestra el resultado final y el otro es un archivo con extensión *.pganas*. Este archivo contiene toda la información necesaria que necesita Blender para reproducir la pose, es decir información relacionada con la imagen (calidad, tipo de imagen y tamaño) e información relacionada con los huesos implicados(nombre y ubicación de cada uno).

Cuando se crea un vídeo llamando al script *GGenVideo.py* se genera un vídeo a partir de poses estáticas previamente creadas. Blender necesita una serie de información para llevar a cabo el proceso de renderizado: frame inicial y final, características del vídeo (tamaño, tipo, nombre,...) y las poses que lo componen, indicando en cada una el frame donde comienzan. Esta información se le pasa a Blender en forma de archivo XML de nombre *nombre_video.ganas*. En el caso en el que el usuario especifique que los huesos implicados en una pose o en un vídeo sean remarcados con un color determinado, la aplicación necesita conocer la malla asociada con cada hueso. Esta especificación se realiza a través de un archivo de configuración.

```
<?xml version="1.0" ?>
<ganas>
        <pose quality="90"/>
        <pose size="PREVIEW"/>
        <pose image_type="JPEG90"/>
        <pose name="pose2">
                <gbone name="Mano.L">
                        <quaternion id="quat_0">
                                0.970084011555
                        </quaternion>
                        <quaternion id="quat_1">
                                -0.130134940147
                        </quaternion>
                        <quaternion id="quat_2">
                                -0.161211624742
                        </quaternion>
                        <quaternion id="quat_3">
                                0.126541525126
                        </quaternion>
                        <location id="loc_0">
                                0.0
                        </location>
                        <location id="loc_1">
                                0.0
                        </location>
                        <location id="loc_2">
                                0.0
                        </location>
                </gbone>
                <gbone name="IKMuneca.L">
```

Figura 4.20: Ejemplo de archivo de configuración de una pose.

```
<?xml version="1.0" ?>
<ganas>
        <gvideo framestart="1"/>
        <gvideo frameend="150"/>
        <gvideo fps="24"/>
        <gvideo quality="24"/>
        <gvideo size="PREVIEW"/>
        <gvideo image_type="AVIJPEG"/>
        <gvideo name="humanoide1">
                <pose name="/home/vane/render/pose1.pganas">
                        <frame number="1"/>
                </pose>
                <pose name="/home/vane/render/pose2.pganas">
                        <frame number="60"/>
                </pose>
                <pose name="/home/vane/render/pose3.pganas">
                        <frame number="110"/>
                </pose>
                <pose name="/home/vane/render/pose4.pganas">
                        <frame number="150"/>
                </pose>
        </gvideo>
</ganas>
```

Figura 4.21: Ejemplo de archivo de vídeo.

Las clases contenidas en el módulo de acceso al entorno 3D se pueden clasificar en dos: clases de generación de poses y vídeo, clases que definen las estructuras de datos necesarias para interpretar los archivos de configuración y clases de generación y lectura de estos archivos.

Clases de generación de vídeo y poses:

- *GGenVideo.py*. Script llamado desde la línea de comandos. Es el encargado de generar el vídeo según lo especificado las en las opciones. Los parámetros que necesita este script para ser ejecutado son:
  - Ruta completa donde de encuentra el archivo de vídeo a renderizar *.ganas*.
  - Ruta donde se almacenará el vídeo generado.
  - Ruta donde se encuentra el archivo de error de Blender.
  - Forma de generar el vídeo: generar vídeo o extraer frames del vídeo.
  - Ubicación del archivo donde se encuentra la información sobre las mallas asociadas a los huesos del esqueleto.
  - Ubicación del archivo de configuración de los huesos.
  - Opción remarcar huesos implicados o no.
- *GCreatePose.py*. Script utilizado desde dentro de Blender para crear poses a partir de la manipulación directa de los huesos.
- *GGenArmBonesXML.py*. Script que recoge los huesos que forman el esqueleto del personaje virtual y almacena dicha información en un archivo de configuración.

Clases de definición de estructuras de datos.

- *GRenderPose.py*. Clase con las que se obtienen objetos definidos por el par ruta de la pose y frame en la que comienza.
- *GConfigVideo.py*. Clase que contiene la información necesaria para generar vídeos en Blender según lo especificado por el usuario. Elementos:

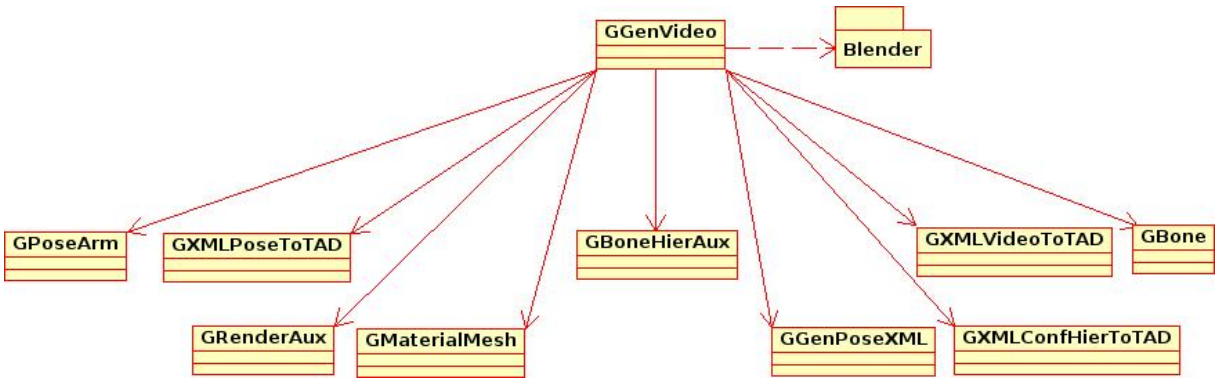

Figura 4.22: Diagrama de clases GGenVideo.

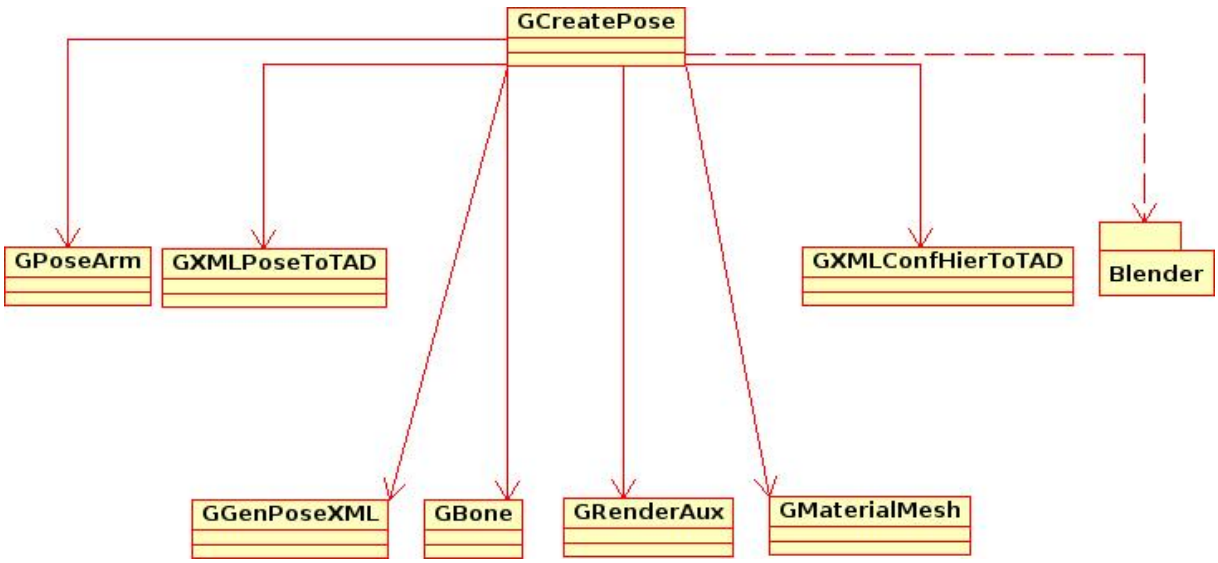

Figura 4.23: Diagrama de clases GCreatePose

- Lista de objetos *GRenderPose*.
- Nombre del vídeo.
- Ruta donde se encuentra el vídeo.
- Frame de comienzo.
- Frame de finalización.
- Tamaño del vídeo.
- Frames por segundo.
- Tipo de archivo.
- Calidad de la imagen.

- *GBone.py*. Clase que define la estructura de datos asociada con un hueso:
  - Nombre del hueso.
  - Quaternion.
  - Localización.

- *GPoseArm.py*. Clase con métodos de acceso a los elementos necesarios para leer o escribir los archivos de extensión *.pganas*, es decir aquellos que son procesados desde Blender para renderizar las poses estáticas. Elementos:
  - Lista de huesos implicados en la pose.
  - Nombre de la pose.
  - Tipo de imagen.
  - Calidad.
  - Tamaño.

- *GMesh.py*. Define la estructura de datos relacionada con las mallas asociadas con los huesos del personaje virtual. Elementos:
  - Nombre de la malla.

  - Lista de huesos que conforman la malla.
  - Color asociado con la malla.

- *GXMLVideoToTAD.py*. Clase desde la cual se obtiene un objeto de tipo *GConfigVideo* a partir del archivo de configuración correspondiente.
- *GGenVideoXML.py*. Clase cuya funcionalidad es realizar el proceso inverso descrito en el punto anterior.
- *GXMLPoseToTAD.py*. Transforma la información contenida en los archivos generados al crear una nueva pose (extensión *.pganas*) a objetos de la clase *GPoseArm*.
- *GGenPoseXML.py*. Clase cuya función es realizar el proceso inverso de la descrita en el punto anterior. Genera un archivo de pose estática a partir de un objeto *GPoseArm*.
- *GXMLConfHierToTAD.py*. Genera un objeto de tipo *GBoneHier* a partir del archivo de configuración *bones_config.cnf*.
- *GXMLMeshToTAD.py*. A partir del archivo de configuración donde se encuentra la información referida a las mallas asociadas a los huesos, obtiene una lista de objetos de tipo *GMesh*. item*GGenMeshXML*. Realiza el proceso inverso descrito en el punto anterior, a partir de una lista de objetos de tipo *GMesh* genera y almacena un archivo de configuración con esta información

Otras clases:

- *GRenderAux.py*. Clase auxiliar para obtener las constantes que utiliza Blender tanto para el tipo como para el tamaño de la imagen.
- *GMaterialMesh.py*. Clase utilizada para asociar un color diferente a una determinada malla.

4.3. Características del sistemas *GANAS*

Para el desarrollo del sistema *GANAS* se ha tenido muy en cuenta tipo de usuario potencial del mismo. Podemos dividirlos en:

- Usuarios sin conocimiento de aplicaciones 3D.
- Usuarios con conocimiento de aplicaciones 3D.

El primer tipo de usuarios es el más extendido, por ello se han creado interfaces gráficas de fácil manejo. El segundo tipo de usuarios aunque se trata de un grupo más reducido que el primero, pueden utilizar sus conocimientos para crear poses directamente desde la interfaz de Blender.

A la hora de desarrollar las interfaces gráficas se ha tenido en cuenta las siguientes características deseables:

- *El usuario debe sentir que la aplicación está hecha para él (él es el que maneja el sistema y no al revés).*
  - Personalización. La interfaz gráfica del módulo de administración permite personalizar la aplicación según las necesidades del usuario.
  - Interfaces sencillos. Esta característica es esencial para que el sistema sea aceptado por el usuario, por lo que se le ha dedicado grandes esfuerzos. Desde la interfaz gráfica de administración, un usuario podrá generar palabras en lenguaje de signos sin necesidad de tener conocimientos de diseño y manipulación de huesos 3D. En cuanto a la interfaz web resalta la sencillez de uso por mostrar sólo lo necesario para que cualquier usuario pueda generar frases en lenguaje de signos.
- *Las interfaces deben ser intuitivas.*
  - Selección fácil del objeto y acción aplicable. En las interfaces gráficas se ha hecho uso de iconos que representan metáforas, por ejemplo el icono de la papelera.
  - Manipulación directa. Un ejemplo es el uso de Drag & Drop para añadir poses en la edición de vídeo o poses estáticas.
- *Consistencia.*
  - Con el mundo real. Los movimientos del personaje virtual deben simular los de una persona.

  - Dentro de la aplicación. Los cambios producidos en el sistema deben reflejarse en la interfaz gráfica y viceversa.

- *Claridad.*

  - Iconos. En las interfaces gráficas de *GANAS* se han hecho uso de iconos para que las acciones u objetos que presenta sean fácilmente localizables por el usuario.
  - Opciones. El usuario debe disponer de diversas opciones para realizar una opción, por ejemplo desde la interfaz del módulo de administración la búsqueda de poses y vídeos se puede realizar de diferentes formas.
  - Mensajes. Es importante que el usuario obtenga información sobre las acciones que realiza. La interfaz gráfica de *GANAS* distingue entre mensajes de error y mensajes informativos. Se ha intentado evitar el uso de jergas que puede no ser entendibles por el usuario.
  - Realimentación constante. El programa debe responder rápidamente a las peticiones del usuario y si la acción a realizar es lenta debe ser indicado, por ejemplo el renderizado.

- *Ofrecer la posibilidad al usuario deshacer acciones o asegurarse de su realización.*

  - Permitir deshacer una acción siempre que sea posible. En el caso de la creación de poses y vídeos el usuario puede eliminar poses estáticas introducidas en la edición y también desecharlas una vez renderizadas.
  - Permitir al usuario cancelar una acción. Cuando un usuario realiza una acción importante, el sistema vuelve a preguntar si realmente desea continuar.

- *Apariencia*.

- Asemejar la apariencia al mundo real. Aunque este punto es mas complejo conseguirlo, se ha intentado que la interfaz gráfica de *GANAS* simule de alguna forma cómo una persona representa los movimientos relacionados con el lenguaje de signos.

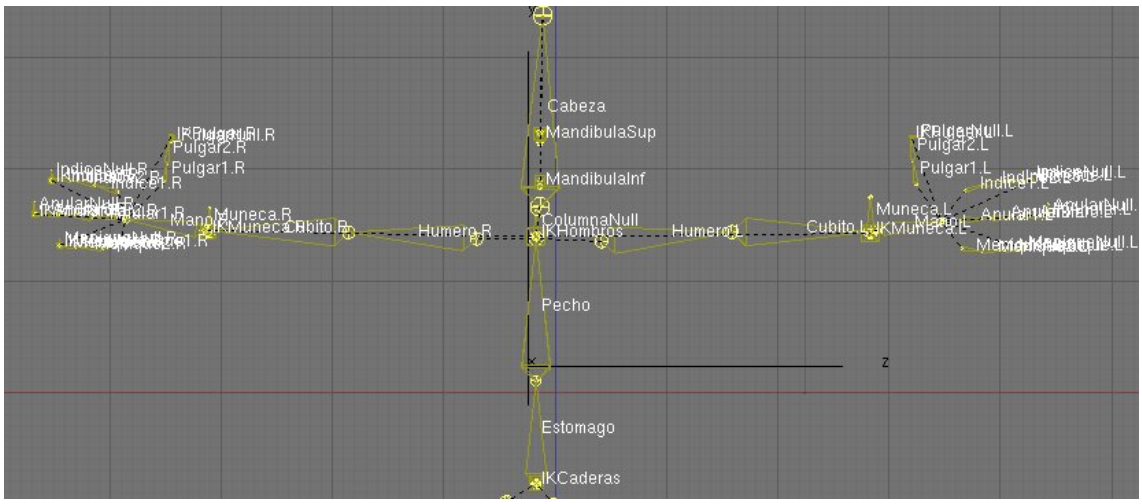

Figura 4.24: Esqueleto de un humanoide definido en Blender.

Otra característica que se ha tenido en cuenta es la reutilización, no sólo de forma interna al crear poses y vídeos a partir de otras ya creadas, sino la reutilización de imágenes o vídeos realizados por otro usuario. Para que las palabras y/o expresiones en lenguaje de signos generadas por este sistema sean usables desde otro ordenador basta con tener los archivos de configuración asociados. Con esta característica los usuarios pueden compartir los resultados obtenidos y reutilizarlos si es el caso.

4.4. Detalle del sistema

En este apartado no se pretende detallar a fondo el sistema *GANAS*, pero sí profundizar en aquellas clases o entornos de manipulación 3D directa más relevantes. La forma de animar a un personaje virtual de manera que simule los movimientos de un humano es asociando un esqueleto a la malla que forma el personaje. El esqueleto está compuesto por una seríe de huesos formando una jerarquía. El proyecto *GANAS* en la jerarquía de los huesos 4.24 no se han considerado los referentes a las piernas y pies, puesto que no se van a realizar movimientos asociados a los mismos.

En la figura 4.25 se observa la jerarquía de huesos que se ha definido en este proyecto. Las líneas punteadas en color violeta indican la relación de parentesco entre los *ik solver* y

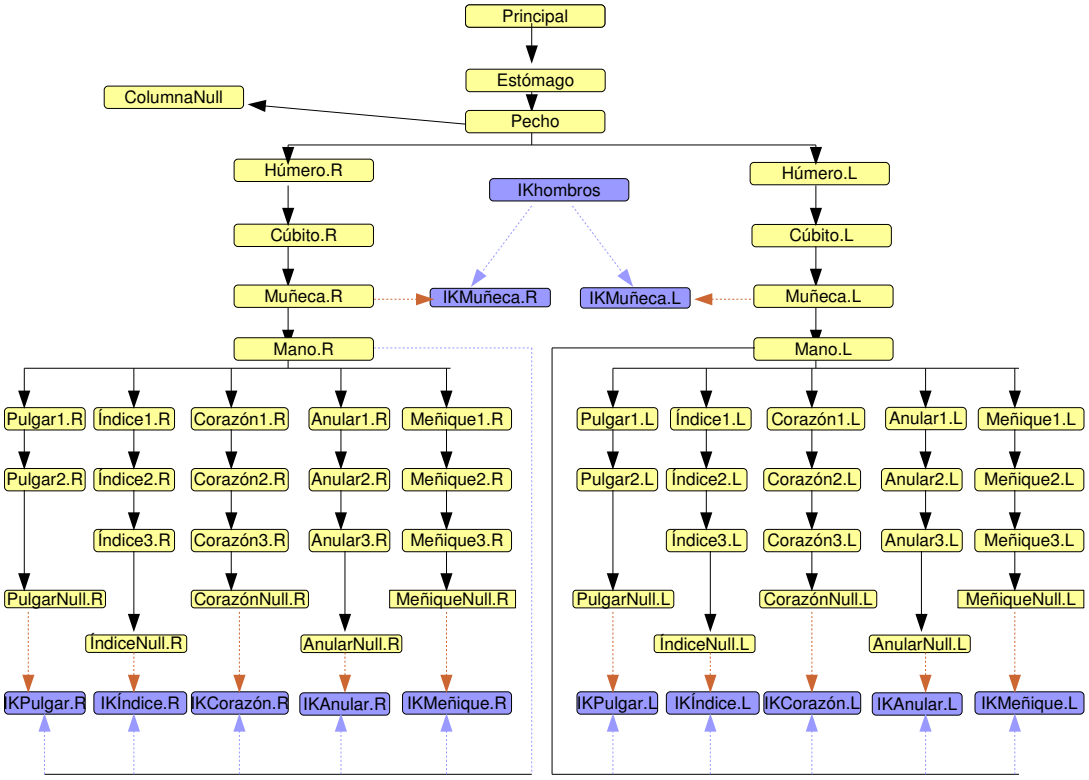

Figura 4.25: Esquema de la jerarquía de huesos definida en GANAS.

las líneas punteadas en color naranja indican las restricción de cinemática inversa asociada al hueso padre. De esta forma se obtiene que los movimientos realizados en un hueso hijo se transmiten al padre (cinemática inversa) y no al revés.

La forma que tiene Blender [19] de representar internamente rotaciones hechas sobre un hueso en modo *pose* es mediante los *cuaterniones*. Cada hueso del esqueleto tiene asociado un vector de localización que indica el cambio del mismo y un *quaternion* que ofrece información sobre el cambio en la rotación del mismo:

- Rotación en el eje X.
- Rotación en el eje Y.
- Rotación en el eje Z.

Ésta es la información que necesita Blender para crear nuevas poses con un esqueleto, por eso en el archivo de configuración creado en *GANAS*, para cada pose se almacena los huesos implicados en la misma así como los cuaterniones y localización de cada uno.

4.4.1. Algoritmo de captura del esqueleto

En la aplicación *GANAS* es necesario poder extraer la información del esqueleto del personaje virtual obtenida en Blender y poder almacenarla para su posterior reutilización. La forma de guardar esta información se ha hecho a través de archivos de configuración (descritos en apartados anteriores). Se ha creado un script 1ejecutable únicamente desde Blender *GGenArmBonesXML.py*, el cual recoge la información de la jerarquía de los huesos y lo almacena en un archivo XML (bones_config.cnf), ver figura 4.19.

Algoritmo 1 Script GGenArmBonesXML.py.

Obtener la jerarquía de los huesos del esqueleto. Ver algoritmo 2

Establecer la ruta donde se almacenará el archivo

Generar el archivo de configuración XML

Guardar en disco el archivo de configuración XML

4.4.2. Algoritmo de generación de una pose

El uso de este script sólo puede hacerse desde Blender, es una forma de dar opción a usuarios con conocimientos en diseño 3D de crear poses manipulando de forma directa los huesos del esqueleto del personaje virtual (algoritmo 3)

Para renderizar una imagen en Blender es necesario proporcionarle cierta información a no ser que se usen los parámetros que tiene por defecto.

4.4.3. Algoritmo de generación vídeo

El script *GGenVideo.py* permite la generación de vídeo en Blender, engloba tanto la creación de poses estáticas como vídeo mediante composición de poses previamente creadas. El algoritmo principal se muestra en 6.

Algoritmo 2 Generación del archivo de configuración de la jerarquía de huesos en Blender.

for all $bone\_data$: hueso del esqueleto **do**
  longitud $\_cadena\_ik \longleftarrow 0$
  $lista\_padres \longleftarrow 0$
  $g \longleftarrow$ objeto GBoneHier (nombre, None, $longitud\_cadena\_ik$,$lista\_padres$)
  $tuple\_ik \longleftarrow$ obtenerRestriccionIk(nombre)
  if $len(tuple\_ik) > 0$ **then**
    $longitud_cadena_ik \longleftarrow$Obtener longitud cadena
    g.longitudCadena $\longleftarrow tuple\_ik[longitud\_cadena\_ik]$
    g.nombreHuesoIk $\longleftarrow tuple\_ik[nombre\_ik]$
    $indice \longleftarrow 0$
    while indice sea menor que longitud de cadena **do**
      if $bone\_data$ no es nulo **then**
        Obtener padre de $bone\_data$
        Añadir nombre del hueso padre a $lista\_padres$
        incrementar en una unidad el índice
      end if
    end while
  end if
  Añadir a la lista de huesos el creado
end for
return Lista de huesos

Algoritmo 3 Creación de una pose estática desde Blender (script *GCreatePose.py*).

Crear lista de errores producidos en Blender

Establecer nombre de la pose a crear

Establecer ruta de almacenamiento de la imagen a crear

Establecer ruta de almacenamiento del archivo de configuración de pose a crear

Establecer tipo, tamaño y calidad de imagen

Obtener información de los huesos implicados en la pose. Ver algoritmo 4

if opción remarcar huesos es True **then**

  Asignar nuevo material a las mallas relacionadas con los huesos implicados

end if

if lista de errores producidos en Blender está vacía **then**

  Renderizar. Ver algoritmo 5

end if

Crear archivo de configuración de la pose y almacenarlo en disco

if opción remarcar huesos es True **then**

  Eliminar material creado

end if

Algoritmo 4 Obtención de los huesos implicados en una pose.

$lista\_gbone \longleftarrow []$

$pose \longleftarrow$ pose del objeto Armature de Blender

for all $bonename$: nombre de hueso de $pose$ **do**

$bonearmature \longleftarrow$ armature de Blender

if $Huesoseleccionado$ está en $bonearmature$ **then**

$bonename \longleftarrow$ nombre del hueso

$quat \longleftarrow$ cuaternión asociado al hueso

$loc \longleftarrow$ localización del hueso

$gbone \longleftarrow$ GBone(bonename, quat, loc)

Añadir $gbone$ a $lista\_gbone$

$pose \longleftarrow$ Crear pose GPoseArm($lista\_gbone, nombre\_pose, tipoimagen, calidad, tamano$).

Establecer $pose$

end if

end for

Algoritmo 5 Renderizar pose en Blender.

Habilitar ventana de visualización del renderizado

Modificar ruta que por defecto tiene Blender para salvar las imágenes

Modificar tamaño por defecto de la imagen

Establecer tipo de imagen de salida del renderizado

Renderizar

Almacenar imagen renderizada

Cerrar ventana de visualización del renderizado

Algoritmo 6 Algoritmo de generación de vídeo.

Determinar opción de usuario *crear vídeo* o *extraer poses de un vídeo*
Abrir archivo de configuración de vídeo
$gvideo \longleftarrow$ información del archivo de configuración de vídeo
if Opción remarcar huesos es True **then**
  Obtener archivo de configuración de malla
end if
if Han ocurrido errores **then**
  Crear archivo de errores ocurridos en Blender
else if Los parámetros de vídeo son correctos **then**
  Establecer la primera pose del esqueleto
  if Opción remarcar huesos es True **then**
    Asignar material a las mallas asociadas a los huesos implicados
  end if
  Renderizar vídeo
  if opcion = extraer frames del vídeo **then**
    Salvar cada frame y asociar el archivo *.pganas*
  end if
end if

el algoritmo 7 cuya función es componer las poses especificadas por el usuario. en

Algoritmo 7 Algoritmo de composición de poses.

$i \longleftarrow$ frame inicial del vídeo

$end \longleftarrow$ frame final del vídeo

$scene \longleftarrow$ escena actual

$context \longleftarrow$ contexto de renderizado

for all i hasta end **do**

Establecer frame inicial en context

$poseact \longleftarrow$ pose actual de la escena

$act \longleftarrow$ acciones creadas en Blender

for all boneAct en poseact **do**

$lgbones \longleftarrow []$

if boneAct está establecido en un frame clave **then**

$quat \longleftarrow$ cuaternión del hueso

$loc \longleftarrow$ localización del hueso

Crear objeto GBone(nombre hueso, quat, loc)

Añadir hueso creado a $lgbones$

end if

end for

if longitud de lista $lgbones$ es mayor que 0 **then**

$newpose \longleftarrow$ Crear pose. Ver algoritmo 8

Generar y almacena archivo de configuración asociado a la pose creada

end if

end for

Algoritmo 8 Algoritmo de creación de poses del script GGenVideo.py.

$gpos \longleftarrow$ objeto GPose asociado a la pose a crear

$frame \longleftarrow$ frame en el que comienza la pose

$context \longleftarrow frame$

$glbones \longleftarrow$ lista de huesos implicados

if No hay acciones establecidas en Blender **then**

$act \longleftarrow$ crear acción

end if

for all Hueso de la pose actual de Blender **do**

$boneAct \longleftarrow$ hueso

if $boneAct$ no es Nulo **then**

$lbonename \longleftarrow$ lista de los nombres de huesos del esqueleto

end if

Establecer quaternion del hueso

Establecer localización del hueso

Añadir frame clave

end for

Actualizar pose actual

Capítulo 5

RESULTADOS

5.1. Resultados obtenidos

Se ha logrado el diseño e implementación de un sistema que permite transformar palabras y/o frases en lenguaje escrito a lenguaje de signos mediante un personaje virtual.

A pesar de que los usuarios potenciales del sistema pueden ser expertos en lenguaje de signos, en líneas generales este tipo de usuarios no suelen tener conocimientos para trabajar con herramientas de diseño 3D. Esta tarea de manipulación directa del personaje 3D no es trivial, ya que hay que tener en cuenta los diferentes puntos de vista del personaje y es habitual que la consecución del movimiento o rotación de un hueso determinado no coincida con los resultados esperados. Con *GANAS* cualquier tipo de usuario puede obtener diferentes posicionamientos de los huesos del *avatar* de forma sencilla evitando el problema comentado.

Para que el usuario se sintiera cómodo con la interfaz gráfica y con la forma de generar la poses y vídeos se plantearon varias formas de diseño. Las conclusiones obtenidas de este

estudio fueron que el usuario necesita saber que es lo que se está haciendo en cada momento y si es posible apoyar esta información con imágenes.

En el entorno de edición de poses claves de *GANAS* se ha tratado que toda operación con estas poses se vea reforzada con imágenes. Este modo de interacción facilita su uso por parte de usuarios no cualificados en herramientas de diseño 3D.

Se ha intentado que el usuario pueda buscar poses o vídeos almacenados en base de datos, para ser reutilizados en nuevas poses o vídeos, de diferentes formas para que utilice la opción de búsqueda con la que más cómodo se desenvuelva.

Para la construcción de poses totalmente nuevas y para facilitar el trabajo a usuarios acostumbrados a herramientas 3D, se ha implementado un en un entorno 3D la funcionalidad que permite al usuario manipular el esqueleto del personaje virtual y que genera de forma automática la información asociada en el sistema *GANAS* a la nueva pose estática creada.

En el ámbito de mejorar la eficiencia del sistema y para evitar la generación de los mismos fragmentos de vídeo varias veces, se ha potenciado la reutilización en los siguiente escenarios:

- Reutilización de poses para crear una nueva pose.
- Reutilización de vídeos para extraer los frames y obtener la pose de cada uno.
- Reutilización de poses estáticas para crear vídeos.
- Reutilización de vídeos para generar nuevos vídeos como composición.

La reutilización de los archivos generados en este sistema ha sido un punto importante que se ha tenido en cuenta durante el desarrollo de *GANAS*. El principal cuello de botella en síntesis de imagen tridimensional es el proceso de render (o de generación de la imagen bidimensional). Como se ha comentado anteriormente, en el sistema se ha intentado reutilizar los fragmentos de vídeo que han sido generados, evitando el cómputo innecesario de estos cuadros de animación.

El almacenamiento de las poses estáticas y de los vídeos que representan palabras, se ha realizado de manera que sea posible el intercambio con independencia de la herramienta utilizada. Un usuario puede compartir con otro las poses o vídeos creados y si van a ser reutilizadas desde la herramienta *GANAS* bastará con almacenar en la base de datos la ruta y

nombre de los archivos, esta operación la facilita *GANAS* desde la interfaz de administración. Por lo tanto, un archivo generado por un usuario puede ser reutilizado por otro según sus necesidades con independencia de la herramienta utilizada (no tiene por qué ser *GANAS*).

Con el diseño e implementación de dos módulos de acceso independientes se ha conseguido aumentar la facilidad de uso del sistema:

- Desde la interfaz de administración el usuario puede crear poses estáticas que serán utilizadas como base para la generación de vídeos correspondientes a palabras del diccionario, por ejemplo generar en lenguaje de signos la palabra “mano”.
- Desde la interfaz web el usuario puede generar vídeos complejos a partir de los vídeos creados en la interfaz de administración y de los generados desde la propia interfaz web; por ejemplo la generación de la frase “mi mano es grande”.
- El usuario puede configurar el sistema según sus necesidades. El sistema se puede adaptar a distintos escenarios en los que se requiera un tipo o formato específico de vídeo.

Las características que engloba el sistema se pueden observar en el cuadro 3.1.

	Usuario entrenado en 3D	Usuario no entrenado en 3D
Entorno de manipulación 3D directa	Sí	No
Crear pose desde Intf. Adm	Sí	Sí
Crear vídeo desde Intf. Adm	Sí	Sí
Extraer poses desde Intf. Adm	Sí	Sí
Configurar sistema desde Intf. Adm	Sí	Sí
Configurar render desde entorno 3D	Sí	No
Configurar esqueleto desde entorno 3D	Sí	No
Generar vídeos desde Int. Web	Sí	Sí

Cuadro 5.1: Tabla de usabilidad de la aplicación según el tipo de usuario.

5.2. Características destacables frente a sistemas similares.

Uno de los objetivos que se plantearon en *GANAS* fue la creación de una herramienta libre para la representación del lenguaje de signos mediante un personaje virtual. Analizando los sistemas existentes se encontraron las deficiencias más relevantes en los siguientes aspectos:

- Usabilidad. La mayoría de los sistemas existentes con objetivos similares al de *GANAS* sólo permiten la creación de poses estáticas y vídeos mediante la manipulación directa del esqueleto del personaje virtual a través de una interfaz gráfica. Esta forma de creación de palabras y frases precisa que el usuario tenga una visión espacial para poder colocar los huesos desde diferentes perspectivas. La visión espacial no la poseen todos los individuos a no ser que estén acostumbrado a trabajar con entornos 3D.

  Si se comparan las figura 5.1 y 5.2 se puede apreciar las diferencias existentes entre ambas. Los sistemas existentes de generación de lenguaje de signos mediante un personaje virtual presentan una interfaz similar a la de una herramienta de síntesis de imagen 3D aunque algo más intuitiva, pero las operaciones que tiene que realizar el usuario para colocar los huesos del esqueleto son semejantes a las que hay que hacer en las suites de producción 3D.

- Proceso de creación de poses y vídeos desde otros sistemas similares a *GANAS*

  - Colocar manualmente los huesos del esqueleto por lo que el usuario debe tener visión espacial del mundo 3D.
  - Crear la acción a realizar y establecer la temporalidad de la misma a través de un editor de secuencia en el que suelen aparecer distintos canales para cada hueso modificado.
  - Renderizar la secuencia completa del vídeo.

- Proceso de creación de poses y vídeos desde *GANAS*.

  - Buscar poses existentes y seleccionar la que necesita el usuario.
  - Añadirla mediante Drag & Drop al editor de vídeo.

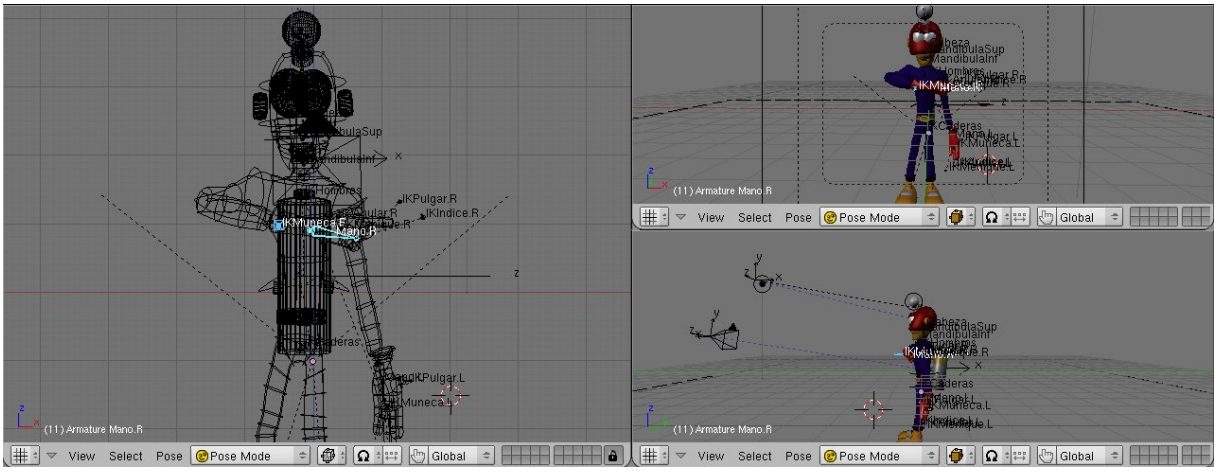

Figura 5.1: Posible configuración de la interfaz de Blender.

- Renderizar sólo lo necesario, es decir a partir de vídeos creados se pueden componer otros más complejos sin necesidad de volver a renderizar la totalidad del vídeo.

Debido al uso de herramientas y tecnologías libres, el sistema desarrollado es fácilmente portable a multitud de sistemas operativos, con un coste de desarrollo e implantación inferior a las alternativas basadas en soluciones comerciales.

Pese a que muchas de las soluciones comerciales estudiadas presentan precios especiales de licencias para educación, el coste de implantación no es adecuado en multitud de escenarios. Además, las utilidades presentan los módulos por separado (diccionario, colección de personajes virtuales, editor de poses estáticas, editor de palabras,...), por lo que el coste de producción es mayor.

Con *GANAS* el usuario tiene todas las utilidades descritas anteriormente, ya que puede crear un diccionario, crear poses, vídeos que representen palabras y vídeos que representen frases completas.

5.3. Alternativas de explotación.

En este apartado se describen algunos de los posibles escenarios en los que se puede implantar la aplicación *GANAS*.

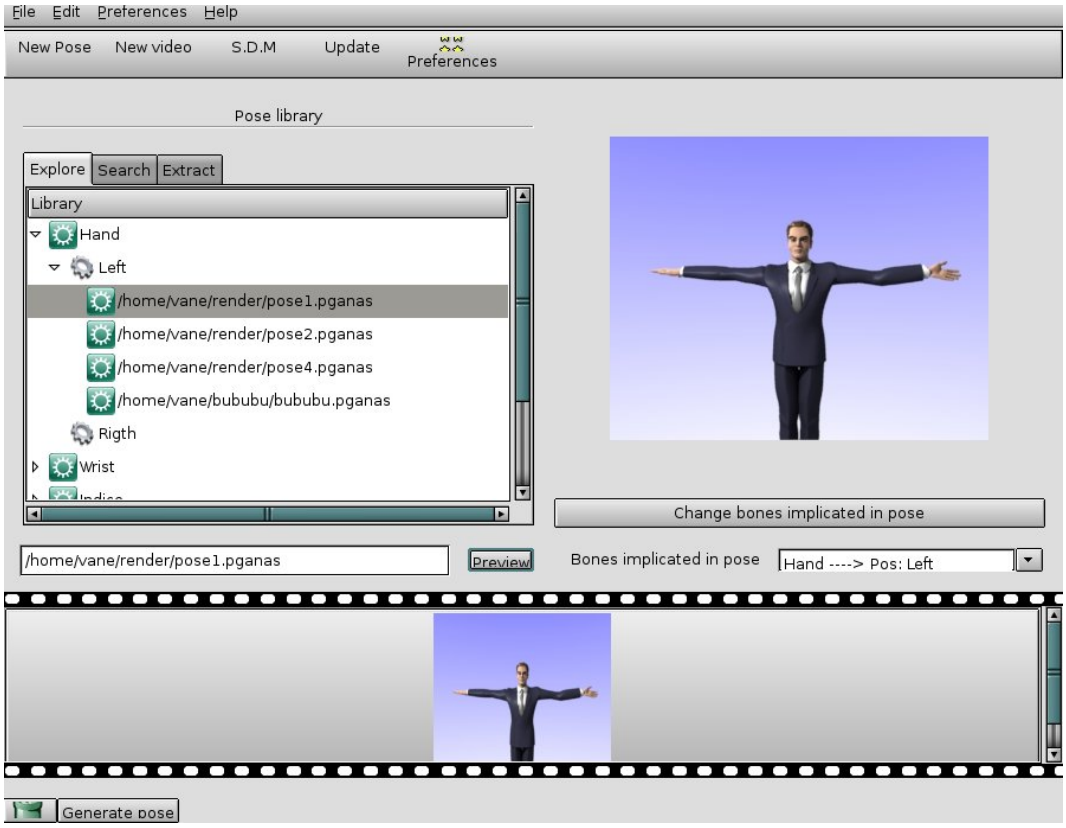

Figura 5.2: Interfaz administración GANAS.

5.3.1. Entretenimiento.

Un posible escenario de uso de esta aplicación es en el área de entretenimiento (películas, videojuegos,..). Para una persona con deficiencia auditiva es más sencillo comunicarse con una persona que conozca el lenguaje de signos que por ejemplo leer subtítulos. Con *GANAS* se podría transcribir el lenguaje escrito que aparece en los subtítulos de una película a lenguaje de signos, incluyendo las animaciones del personaje virtual en el vídeo proyectado.

5.3.2. Educación.

Una correcta educación en personas con discapacidad auditiva es esencial para su desarrollo. En este sentido dos posibles escenarios serían:

- En el área de formación de nuevos intérpretes del lenguaje de signos. Con la aplicación *GANAS*el usuario puede poner en práctica conocimientos teóricos, creando palabras, frases y expresiones.

- En el área de la educación de personas con deficiencia auditiva. La enseñanza en edades tempranas de personas con algún tipo de deficiencia es crucial para un correcto desarrollo personal y educacional. La aplicación *GANAS* puede proporcionar un apoyo en la educación, ya que proporciona la posibilidad de generar elementos multimedia con los que el alumno puede recrear palabras en lenguaje de signos así como desarrollar nuevas y poner en práctica sus conocimientos.

- En lugares donde se imparten clases como colegios, universidades y academias. El profesor puede utilizar la herramienta *GANAS* para generar vídeos que apoyen el contenido de la materia impartida.

5.3.3. Información.

Las personas que no sufren de ningún tipo de deficiencia no son conscientes de las barreras que cada día tienen que resolver aquellas que poseen algún grado de minusvalía. Un posible uso de esta aplicación es la de generar vídeos en lenguaje de signos que sirvan de información

en diferentes lugares, como por ejemplo oficinas de correos, entidades públicas, aeropuertos, estaciones de trenes ...etc.

Capítulo 6

CONCLUSIONES Y PROPUESTAS FUTURAS

6.1. Conclusiones

Todas las personas necesitan intercambiar información para poder desarrollarse a nivel personal y cultural. Esta comunicación se lleva a cabo mediante el uso de un lenguaje común tanto del emisor como del receptor.

El lenguaje más empleado entre personas oyentes es el lenguaje oral, ya que se trata de la forma más ágil de comunicación, pero la mayoría de las personas que sufren deficiencia auditiva tienen dificultades para poder expresarse con este lenguaje. Por este motivo se creó un lenguaje signado que mediante el movimiento y posicionamiento de manos, brazos y boca representan palabras, frases y expresiones, llamado lenguaje de signos.

En las últimas décadas han aumentado los esfuerzos por ayudar a personas no oyentes en su integración en un mundo mayoritariamente oyente, ya que históricamente estas personas eran apartadas de la sociedad. Pero los esfuerzos no han sido suficientes puesto que las personas con deficiencia auditiva siguen encontrado hoy en día grandes barreras en la

comunicación.

Tras el desarrollo del sistema *GANAS* se han cumplido los objetivos propuestos al principio, ya que permite la síntesis automática de animaciones asociadas a un personaje virtual que representa palabras del lenguaje escrito en lenguaje de signos. Para la consecución de este objetivo se han cumplido los siguientes subobjetivos:

- Con el sistema *GANAS* se pueden crear nuevas poses clave manipulando de forma directa el esqueleto del personaje virtual a través de funcionalidades implementadas en un entorno de manipulación 3D directa. Se ha desarrollado interactuando con la API que proporciona la suite de producción 3D Blender y con el lenguaje de programación *Python* ya que es el que utiliza Blender en sus scripts.

- Otra utilidad del sistema es la generación de poses estáticas a partir de otras previamente creadas, esta acción se puede realizar a través de la interfaz gráfica de administración. En este caso el proceso de renderizado que sigue la suite de producción 3D es transparente al usuario por lo que no es necesario que tenga conocimientos en herramientas de diseño 3D.

- El intercambio de las poses estáticas generadas para ser utilizadas en herramientas diferentes a *GANAS* se ha conseguido utilizando el lenguaje de marcas *XML* que es independiente del sistema. Los archivos que contienen la información de cada una de las poses estáticas están escritos utilizando las marcas de este lenguaje, de tal manera que desde otro sistema diferente a *GANAS* se podrá extraer la información que contienen.

- El sistema permite la generación de vídeos que representan palabras en lenguaje de signos interpretados por un personaje virtual como composición de poses estáticas. El usuario podrá generar estos vídeos desde la interfaz gráfica a través de un editor de secuencia y es el sistema el encargado de mandar la información a una suite de producción 3D para proceder al renderizado.

- En generación de frases o expresiones (en formato de vídeo) se utilizan los vídeos realizados que representan palabras en lenguaje de signos. Para obtener el vídeo final, el sistema ganas compone los vídeos que lo forman utilizando la herramienta *AVIMERGE*,

aunque primero se renderizan los vídeos intermedios entre el la pose final de uno y la inicial del siguiente llamando a la suite de producción 3D Blender.

- Para que el sistema sea lo menos lento posible se ha fomentando la reutilización de vídeos, de manera que si un vídeo forma parte de otro más complejo, el primero no será renderizado de nuevo sino que será reutilizado.

- Con el fin de que el sistema *GANAS* fuera usable se han utilizado métodos como arrastras y soltar poses para la generación de nuevas poses estáticas e incluso de vídeo.

- El proyecto se ha desarrollado bajo una licencia Libre compatible con GPL de la FSF.

- Con el fin de que fuera multiplataforma se han utilizado librerías abiertas como las proporcionadas por Blender o las que oferta python para desarrollar interfaces GTK (pygtk).

6.2. Mejoras y líneas de trabajo futuras

El sistema *GANAS* se ha desarrollado pensando en construir un sistema escalable que permita incorporar nuevas funcionalidades. Algunas posibles mejoras son:

- Incluir expresiones faciales que simularan sentimientos como sorpresa, aburrimiento o algegría y que pudieran ser añadidas a la animación en un determinado momento. Esta línea de trabajo se enmarca dentro del área de investigación del "Affective Computing" [12], con trabajos relevantes en el área de robótica y la realidad virtual.

- Incluir animación facial al tiempo que se genera el vídeo. De esta forma el usuario puede leer los labios. Blender permite este tipo de animación pudiéndose realizar de varias formas: mediante huesos, *shape-keys-IPO Drivers* y *lattice* de la API de Blender.

- Extracción automática de un vídeo a partir de un texto complejo. Con este fin se puede crear un lenguaje con una determinada gramática y que reconozca la una entrada de texto y determine si es léxica, sintáctica y semánticamente correcto, en caso afirmativo generará el vídeo asociado al texto como composición de otros vídeos almacenados

en el sistema. Para el desarrollo del analizador léxico se pueden utilizar herramientas como *Flex* y para el analizado sintáctico *Yacc* o *Bison*.

- Posibilidad de incluir subtítulos en los vídeos mediante el uso de archivos tipo *sub* o *srt* en los que se especifica el intervalo de tiempo en el que debe aparecer un texto determinado. En los subtítulos de extensión *.sub* cada fragmento de texto se pone entre llaves precedido de la letra *T* que indica el tiempo de actuación (por ejemplo 00:00:34:00). Para los subtítulos con formato *.srt* se enumeran los distintos fragmentos de texto que se incluirán en el vídeo y a continuacón se escribe el intervalo de tiempo seguido del texto que se debe mostrar, por ejemplo "1 00:00:51,390 – 00:00:55,530 ¿Cómo estás?" sería una cadena en este formato.

- Aunque el sistema está preparado para trabajar con distintos personajes virtuales no es posible desde las interfaces gráficas seleccionar el personaje a utilizar. Una mejora sería la de proporcionar diferentes *avatares* y permitir al usuario seleccionar el más adecuado para la animación. Si se utiliza el mismo esqueleto para los diferentes personajes virtuales tan sólo es necesario añadir una lista desplegable desde la que el usuario elegirá, antes de comenzar el renderizado, qué personaje utilizará. Según esta elección se incorporarían información para cargar el archivo de blender en el que está modelado el personaje.

- Proporcionar interfaces gráficas multi-idioma: el uso de GTK permite la cómoda incorporación de ficheros *.PO* con la definición de las etiquetas del interfaz de usuario para varios idiomas. El módulo *gettext* se encargará del uso de un fichero *.PO* u otro según la definición de la variable *LOCALE* del computador del usuario.

- Desarrollo de diccionarios que distingan entre las diferentes formas que puede adoptar una palabra según la región en la que se utilice.

El proyecto *GANAS* ha servido como punto de partida del proyecto financiado por la *Cátedra Indra - UCLM* que comenzará en el último cuatrimestre del 2007 con una duración de dos años durante los cuales se creará un sistema para la conversión de lenguaje escrito a

lenguaje de signos, con el fin de obtener una secuencia de vídeo de un personaje que interpreta en lenguaje de signos el texto introducido.

Capítulo 7

ANEXOS

7.1. Manual de usuario

Para que un usuario pueda utilizar de forma correcta el sistema, en este anexo se describe cómo realizar las operaciones permitidas en la aplicación *GANAS*. El manual de usuario se ha dividido en tres partes, cada una de ellas correspondiente a un módulo de la aplicación:

- Manual de usuario del módulo del entorno 3D de manipulación directa.
- Manual de usuario del módulo de administración.
- Manual de usuario del módulo web

7.1.1. Manual de usuario del módulo de entorno 3D de manipulación directa.

Las acciones que se pueden realizar desde este módulo están orientadas a usuarios con cierto conocimiento de suites de producción 3D.

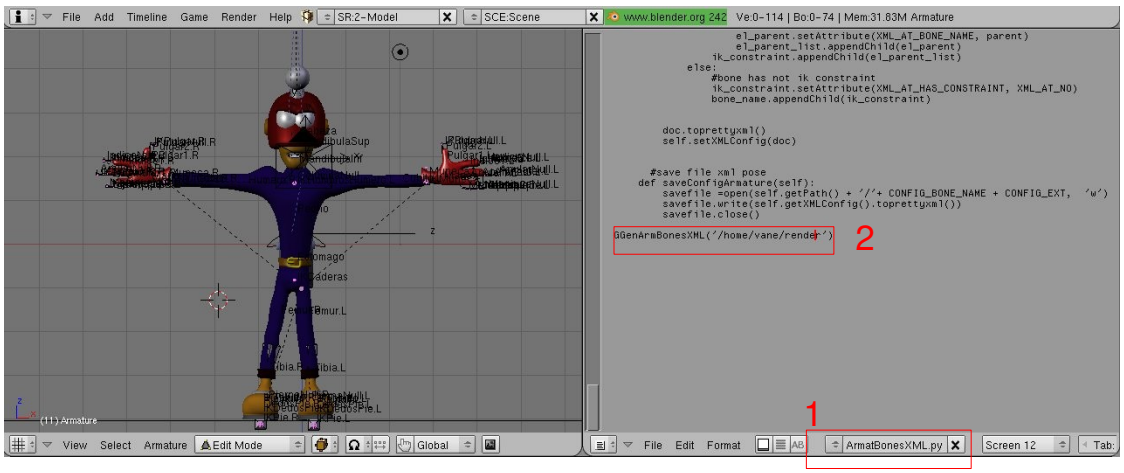

Figura 7.1: Ejecución de la utilidad del entorno 3D de manipulación directa *GGenArmBonesXML.py*.

7.1.1.1. Generación del archivo de configuración del esqueleto.

Mediante el script *GGenArmBonesXML* el usuario extrae la información del esqueleto del personaje virtual activo en ese momento. Por defecto este script ha sido ejecutado y el archivo de configuración se encuentra almacenado. Pasos a seguir:

- Abrir el archivo *GGenArmBonesXML.py* desde una ventana de texto (ver figura 7.1 paso 1).
- Modificar la ruta donde se desea almacenar el archivo de configuración que se creará tras la ejecución del script (ver figura 7.1 paso 2).
- Ejecutar el script.

Como resultado de la ejecución del script se almacenará en la ruta indicada por el usuario el archivo de configuración del esqueleto del personaje virtual. Este archivo es imprescindible para poder realizar el resto de acciones posibles en la aplicación *GANAS*.

Figura 7.2: Ejecución del la clase *GCreatePose.py* desde el entorno 3D de manipulación directa.

7.1.1.2. Generación de poses.

Desde el módulo del entorno 3D de manipulación directa el usuario puede crear poses estáticas en el formato que utiliza *GANAS*. Como se ha comentado anteriormente, esta opción esta orientada a usuarios que tengan conocimiento de diseño 3D ya que requiere la manipulación directa de los huesos del personaje virtual.

El script que realiza esta tarea es *GCreatePose*. Las acciones que ha de seguir el usuario son:

- Cargar el script *GCreatePose.py* en una ventana de texto.
- Modificar la última línea del script en donde se hace la invocación a la clase GCreatePose (ver figura 7.2). Los parámetros que requiere la clase son:
  - name: nombre de la pose
  - impath: ruta del directorio donde se guardará la imagen renderizada.

- xmlpath: ruta del directorio donde se guardará el archivo asociado a la imagen renderizada.
- imtype: tipo de imagen (JPEG90, PNG, BMP).
- quality: calidad de la imagen (si es JPEG valores entre 10 y 100).
- size: tamaño de la imagen (PAL, FULL, PREVIEW, PC, DEFAULT).
- xmlpathmesh: ruta completa donde se encuentra el archivo de configuración de las mallas del personaje virtual.
- xmlpathhierarchy: ruta completa donde se encuentra el archivo de configuración del esqueleto.
- option_remark: valor 0 si los huesos no se tienen que remarcar; valor 1 en caso contrario

- Posicionar los huesos del esqueleto (en modo pose).
- Dejar seleccionados los huesos que han sido movidos o rotados (color azul oscuro).
- Salvar los cambios.
- Ejecutar el script.

7.1.2. Manual de usuario del módulo de administración.

Desde el módulo de administración (ver figura 7.3) el usuario puede configurar el sistema, crear poses estáticas y vídeos.

7.1.2.1. Configuración del sistema.

Desde el menú de la interfaz *Preferences - Config* se accede a la ventana desde la cual el usuario puede configurar el sistema. Las opciones de configuración aparecen desplegadas en un árbol, pulsando sobre cada una de ellas se cargará en la parte derecha de la ventana la información asociada. Las opciones son:

Figura 7.3: Pantalla principal de la interfaz de administración.

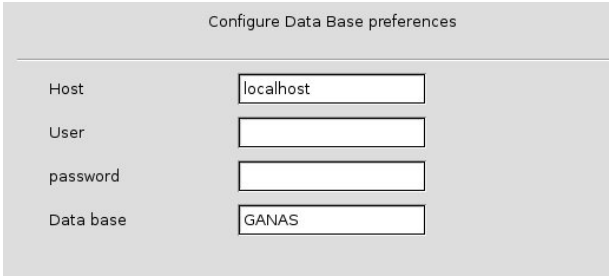

Figura 7.4: Configuración de la base de datos.

- Data Base (ver figura 7.4). Mediante esta opción el usuario configura el acceso a la base de datos: host, nombre de usuario, contraseña, y nombre de la base de datos que por defecto es GANAS. Si el usuario realiza algún cambio debe salvar la nueva configuración pulsando el botón *Save data base configuration*.

- Image (ver figura 7.5). Opción para configurar el tipo de imágenes estáticas que se crearán, si el usuario modifica la configuración y desea que se almacene deberá pulsar el botón *Save image configuration*.

  - Image type. Formato de la imagen, puede ser JPEG, PNG o BMP.
  - Image size. Desde la lista desplegable el usuario puede elegir el tamaño de la imagen.

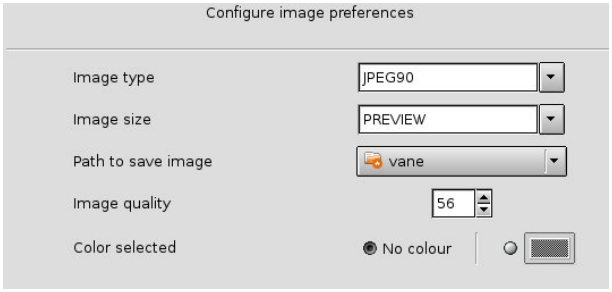

Figura 7.5: Configuración de los archivos de imagen.

- Path to save image .El usuario debe seleccionar la ruta donde se almacenará la imagen y el archivo que generará la aplicación asociada a dicha imagen.
- Image quality. El usuario especifica la calidad que debe tener la imagen.
- Color selected. Si la el usuarios elecciona *No colour* los huesos implicados en una pose no serán remarcados, pero si el usuario selecciona un color con eso indica que los huesos deben remarcarse con el color elegido.

- Video (ver figura 7.6). Configuración de los archivos de vídeo que se generarán. Si el usuario modifica algún dato y desea que se almacenen los cambios debe pulsar el botón *Save video configuration*. Es similar a la configuración de imagen, solo que ahora las opciones son:
  - Video type. Formato del vídeo, puede ser AVIJPEG, QUICKTIME, o AVICODEC.
  - Video size. Tamaño del vídeo.
  - Frames per second. Frames por segundo.
  - Quality. Calidad del vídeo.
  - Colour selected. El usuario puede elegir que los huesos implicados en un vídeo se remarquen con un color determinado o bien que no se destaquen (No colour)
  - Path to save vídeo. Ruta donde se almacenarán los vídeos creados así como los archivos que genera la aplicación *GANAS* asociado a cada uno.

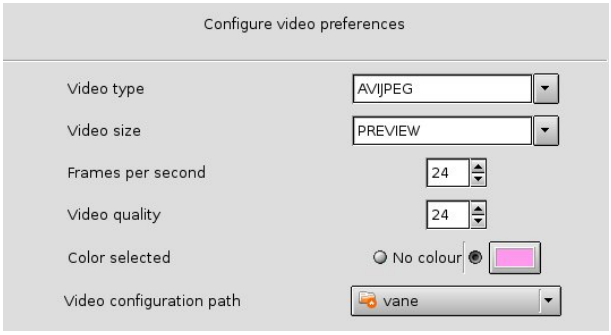

Figura 7.6: Configuración de los archivos de vídeo.

- Image quality. El usuario especifica la calidad que debe tener la imagen.
- Colour selected. Si el usuario selecciona *No colour* los huesos implicados en una pose no serán remarcados, pero si el usuario selecciona un color con eso indica que los huesos deben remarcarse con el color elegido.

- Bones name (ver figura 7.7). El usuario puede modificar los nombres de los huesos que tiene el personaje virtual y que por defecto tiene la aplicación. Pinchando dos veces sobre cualquier fila aparecerá en los cuadros de texto situados en la parte inferior la información asociada:

  - Bone name: nombre del hueso.
  - Bone rename: nombre que el usuario desea darle.
  - Bone position: posición del hueso en el esqueleto (left, rigth o none)

  Para añadir un hueso modificado el usuario debe pulsar el botón *Append* y los cambios se almacenarán pulsando el botón *Save*

7.1.2.2. Acceso a la base de datos.

La búsqueda, borrado y modificación de poses y vídeos almacenados en base de datos se realiza desde el menú de la ventana principal de la aplicación *Edit - Search, delete, modify*. Cuando el usuario selecciona esta opción aparece una ventana auxiliar en la que el usuario debe seleccionar si desea buscar poses estáticas o vídeos (words):

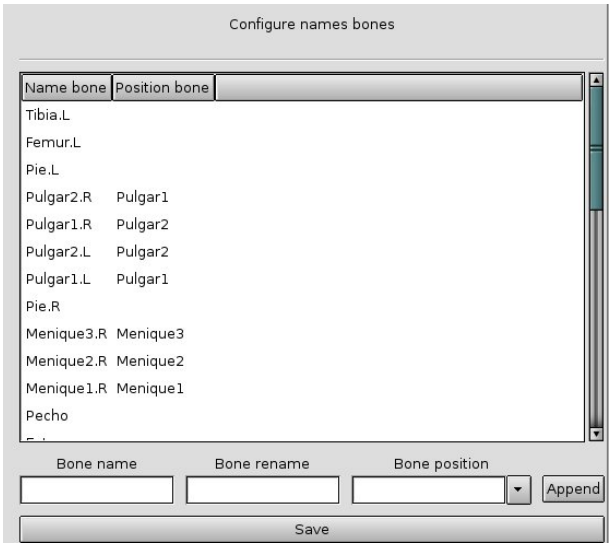

Figura 7.7: Configuración de huesos.

- Seleccionando la opción *Pose* y dentro de la pestaña *Explore* aparecen todas las poses almacenadas en base de datos organizadas según los huesos implicados en cada una de ellas ver figura 7.8.

- Seleccionando la pestaña *Search* y la opción *Pose*, la aplicación muestra dos formas distintas de buscar poses estáticas:

  - Search by name. El usuario realiza la buscar la pose por el nombre dado a la misma.
  - Search by bone name. El usuario realiza la búsqueda de la pose según los huesos implicados. Para ello primero debe seleccionar de la lista desplegable el nombre del hueso y la posición que ocupa en el esqueleto (posición izquierda, derecha, ninguna o ambas).

  Una vez que el usuario ha elegido el criterio de búsqueda debe pulsar el botón representado por una hoja y una lupa. En los dos casos descritos anteriormente los resultados de búsqueda aparecerán en el marco inferior ver figura 7.9

- Seleccionando la opción *Word* y ubicándose en la pestaña *Explore*, el usuario podrá visualizar todos los vídeos almacenados en base de datos.

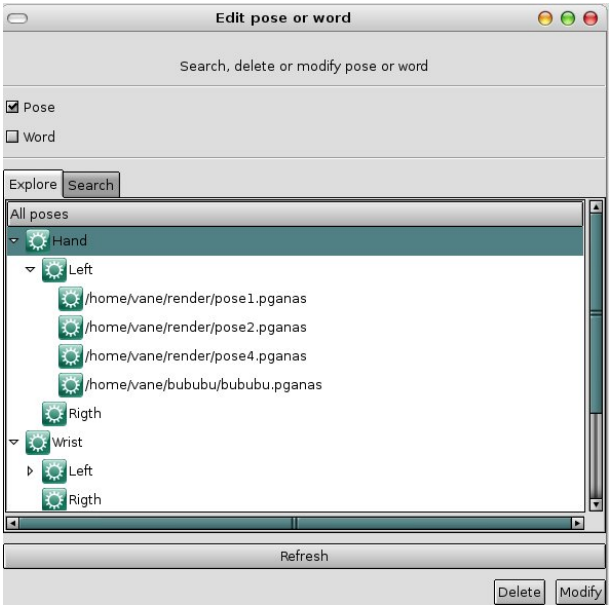

Figura 7.8: Explorar poses almacenadas en base de datos.

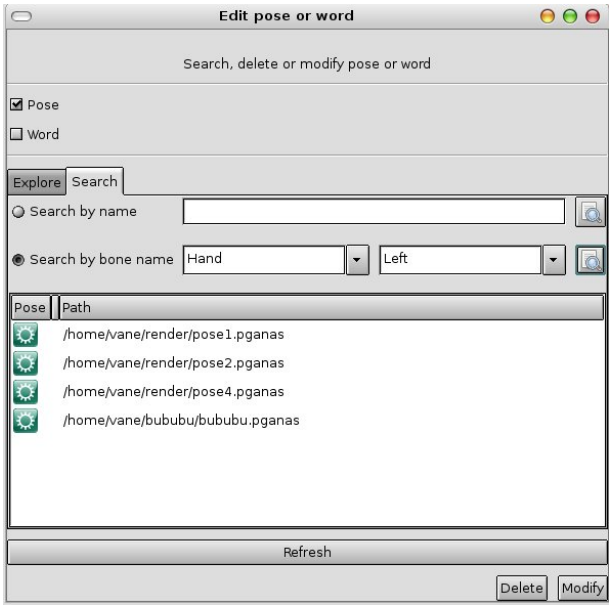

Figura 7.9: Buscar poses almacenadas en base de datos.

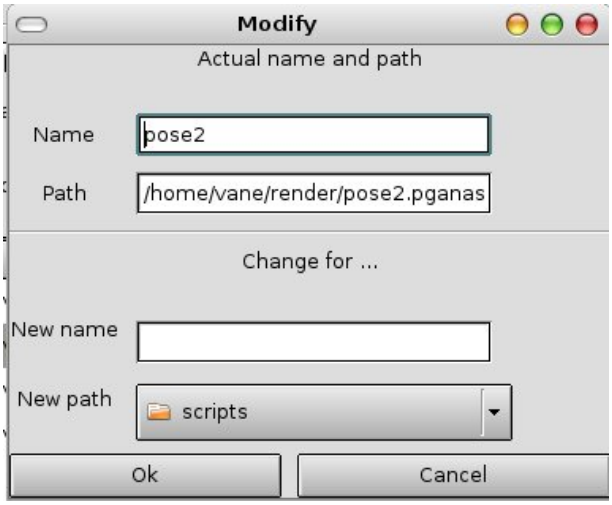

Figura 7.10: Modificar poses o vídeos.

- Seleccionando la opción *Word* y ubicándose en la pestaña *Search*, el usuario podrá buscar por nombre los vídeos almacenados en base de datos
- Una vez seleccionada una pose o un vídeo buscado de cualquiera de las formas descritas anteriormente, el usuario podrá eliminarla o modificarla:
  - Eliminar. Pulsando el botón *Delete* aparece una ventana auxiliar en la que el usuario confirmará que desea continuar con la acción.
  - Modificar. Pulsando el botón Modify se abre una ventana auxiliar en la que se muestra el nombre y la ruta de la pose o vídeo, para cambiar estos datos el usuario deberá introducir en el campo *New name* el nuevo nombre que le desea asignar y en *New path* la nueva ruta donde se guardará (ver figura 7.10).

Con la opción del menú principal *Edit - Update*, el usuario podrá almacenar en base de datos vídeos y poses. En la ventana que aparece (ver figura 7.11) el usuario buscará en disco el archivo *GANAS* asociado a la pose o vídeo y pulsando el botón *Update* se ejecutará la acción.

7.1.2.3. Creación de poses estáticas.

Desde el módulo de administración el usuario puede crear poses estáticas que posteriormente serán reutilizadas para la generación de otras poses y de vídeos. Para crear una pose

Figura 7.11: Almacenar en base de datos poses o vídeos.

Figura 7.12: Creación de poses desde la interfaz de administración.

el usuario debe seleccionar del menú principal la opción *File - New - Pose*, la aplicación mostrará una apariencia similar a la de la figura 7.12.

En la figura 7.12 aparece en la parte izquierda tres pestañas con las etiquetas *Explore*, *Search*, *Extract*. Las dos primeras son para la búsqueda de poses de forma similar a la descrita en la sección *Acceso a base de datos*. La tercera opción *Extract* permite al usuario extraer los frames que componen un vídeo y una vez extraídos seleccionando el que se ajusta a sus necesidades la aplicación lo trasforma en el formato *GANAS*, ver figura 7.13.

Para crear poses como composición de otras los pasos a seguir son:

- Buscar la pose que formará parte de la final.

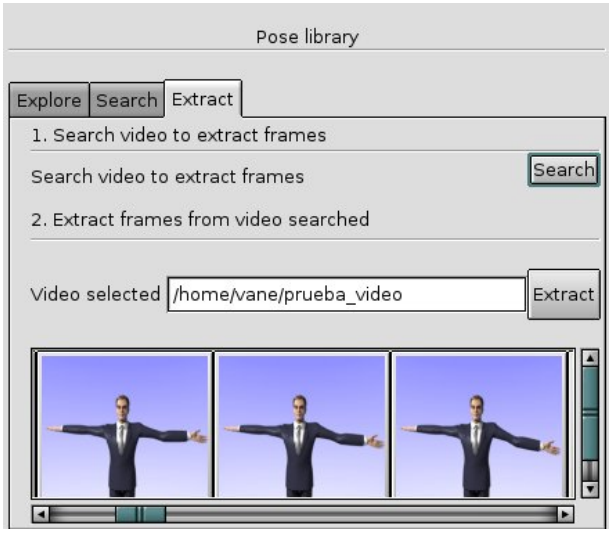

Figura 7.13: Extracción de poses de un vídeo.

- Seleccionarla.
- Pulsar el botón *Preview* para previsualizarla.
- Si no desea modificar los huesos implicados en la pose, añadirla a la tabla de edición mediante Drag & Drop.
- Continuar buscando y añadiendo poses hasta que mediante composición formen la pose deseada.

Puede ocurrir que de una determinada pose al usuario no le interese que actúen todos los huesos sino alguno de ellos. Pulsando el botón *Change bones implicated in pose* el usuario podrá seleccionar los huesos que le interesen, la ventana que se muestra es la que aparece en la figura 7.14. En la parte izquierda aparecen los huesos que inicialmente afectan a la pose, seleccionando un hueso y pulsando el botón *">>"* el usuario lo añadirá indicando así que desea que forme parte de la pose. En la lista que se muestra en la parte derecha el usuario puede observar los huesos implicados en la pose. Cuando los huesos que afectan a la pose sean modificados, se añadirá de igual forma que las demás a la tabla de edición (Drag & Drop).

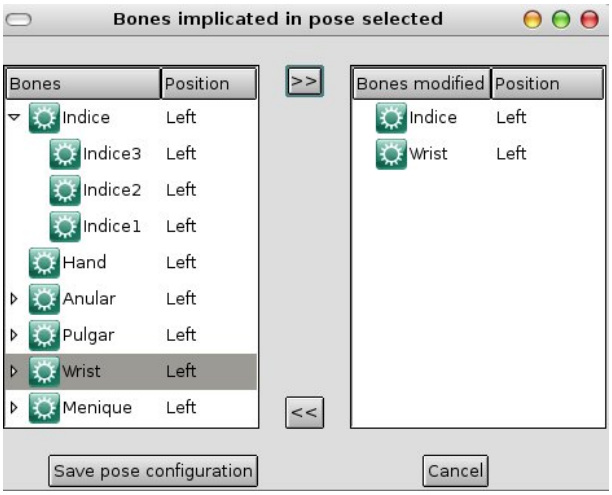

Figura 7.14: Ventana de modificación de huesos que afectan a una pose.

Si el usuario desea eliminar alguna de las poses de la tabla de edición deberá 'arrastrarla' al icono de la papelera. Una vez que el usuario termine con la composición y desee generar la nueva pose deberá pulsar el botón *Generate Pose*. La ventana que aparece (ver figura 7.15) es necesaria para configurar el tipo de imagen (descrito en la sección de configuración) y el nombre y ruta donde se almacenará la imagen renderizada. Una vez que el usuario introduzca los datos necesarios comenzará el proceso de renderizado.

Si no ha ocurrido ningún error en la suite de producción 3D, una vez termine la generación de la imagen se mostrará una ventana desde la que el usuario podrá visualizar la nueva pose formada y añadirla a la base de datos o eliminarla.

7.1.2.4. Creación de vídeos.

Los vídeos creados desde la interfaz de administración son resultado de composición de poses estáticas. El proceso de generación es similar al descrito en la sección *Creación de poses estáticas*. Los pasos que debe seguir el usuario son:

- Buscar la pose que formará parte del vídeo.
- Previsualizarla (botón *Preview*)
- Arrastrar mediante Drag & drop la imagen previsualizada a la tabla de edición.

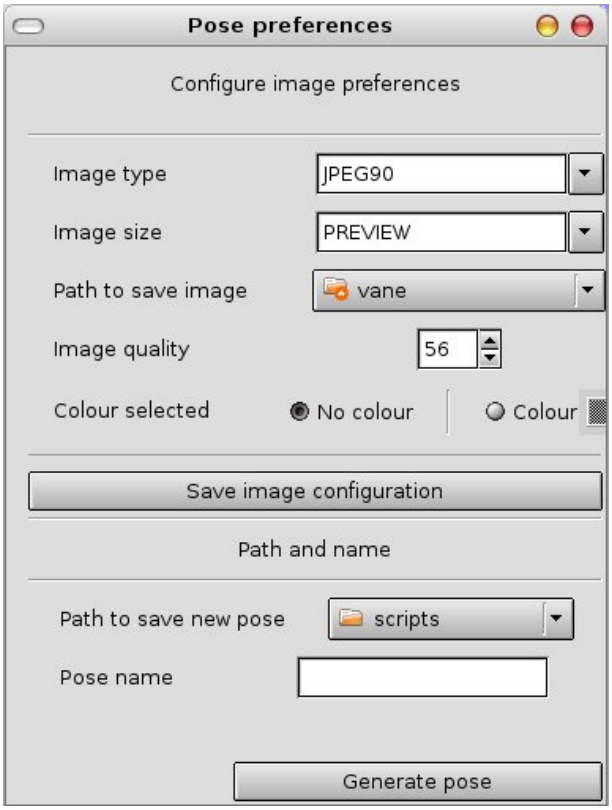

Figura 7.15: Ventana de configuración de una pose.

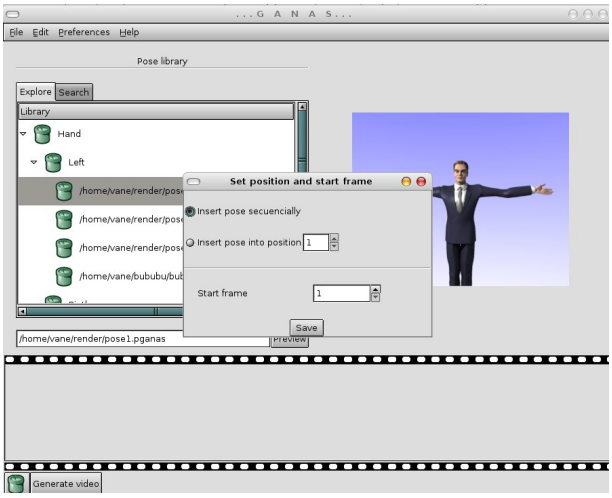

Figura 7.16: Ventana de generación de vídeo.

- Determinar la posición que ocupará en la secuencia de vídeo (ver figura 7.16). La opción por defecto es secuencial, es decir la imagen se añadirá en la siguiente posición, pero el usuario puede determinar otro orden indicando la posición que desea que ocupe (*insert orden into position*). Es necesario especificar el frame en el que comience la influencia de esta pose.

- Continuar hasta que el vídeo está configurado según las necesidades del usuario.

Es posible eliminar poses que el usuario había incluido en la secuencia de edición (Drag & drop). Al pulsar el botón *Generate video* se abrirá una ventana como la que se muestra en la figura 7.17. Los campos obligatorios que el usuario debe completar son el nombre y la ruta donde se almacenará el vídeo y el frame inicial y final.

7.1.3. Manual de usuario del módulo web.

Desde la interfaz que proporciona este módulo (ver figura 7.18)el usuario puede crear de forma sencilla vídeos como composición de otros.

En la parte izquierda de la interfaz aparece una caja de texto asociada con la etiqueta *Video Words*, aquí el usuario introducirá la palabra del vídeo que quiere buscar. Una vez presionado el botón *Search* se mostrará en *Results* los vídeos encontrados. Asociado a cada vídeo aparece:

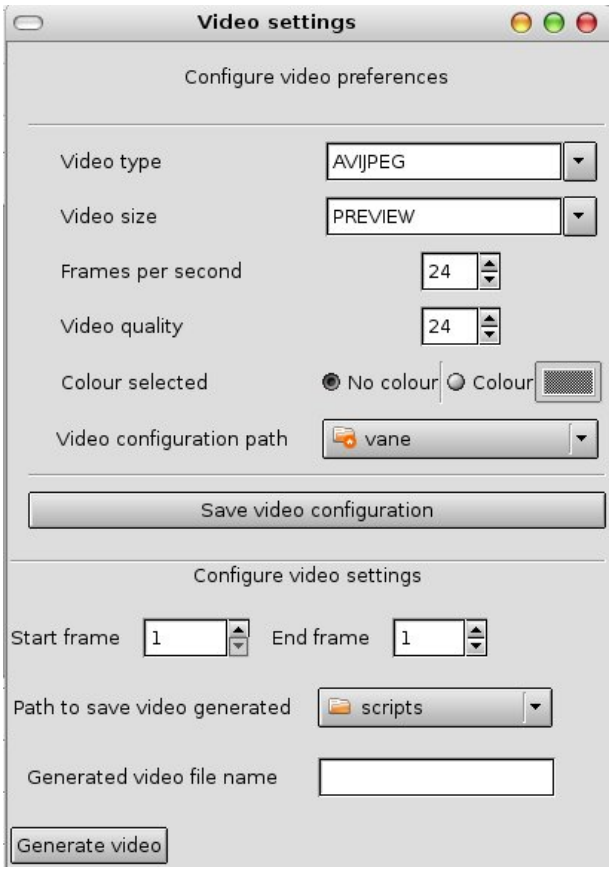

Figura 7.17: Ventana de configuración del vídeo a generar.

Figura 7.18: Apariencia de la interfaz gráfica del módulo web.

- Descripción del vídeo.
- Botón $+$. Pulsándolo se añadirá el vídeo a la lista de edición.
- Botón *Preview*. Pulsando sobre este texto se mostrará en la parte derecha de la interfaz una previsualización del vídeo.

En la lista titulada *Video Sequence* se mostrarán los vídeos que el usuario ha ido añadiendo para obtener el vídeo final.

- La flecha hacia arriba se utiliza para subir el vídeo seleccionado en el orden de secuencia.
- La flecha hacia abajo se emplea para bajar de posición el vídeo.
- Pulsando el signo $-$ se eliminará el vídeo seleccionado.

Una vez que el usuario establezca los vídeos y el orden en el que deben componerse, pulsando el botón *Generate video sequence* la aplicación generará el vídeo completo.

El vídeo final podrá ser visualizado en la parte derecha de la interfaz (Video Player).

Bibliografía

[1] John Stallo Andrew Marriott. Vhml- uncertainties and problems. a discussion ... 2002.

[2] John Stallo Quoc Hung Huynh Andrew Marriott, Simon Beard. Página oficial de vhml. *http://www.vhml.org*.

[3] I. Muñoz Baell. ¿ cómo se articula la lse ? *CNSE*, 1999.

[4] Daniel Corcos. *Artículo del Instituto tecnológico de buenos aires*.

[5] José Julián Díaz Díaz Donald Hearn, M. Pauline Baker. *Gráficas por computadora*. Prentice - Hall, 1995.

[6] Francisco Ramírez C. Klaus Rall F. Eugenio Lez G., Rafael Colás O. Maquinado de una sucesión de curvas. *Ingenierías*, IV:35–36, 2001.

[7] Steven K.Feiner John F. Hughes James D. Foley, Andries Van Dam. *Computer Graphics: Principles and Practice in C (2nd Edition)*. Hardcover, 1995.

[8] M.A Lou Royo. *La educación del niño deficiente auditivo*. Pirámide, 1998.

[9] Beard S Haddadi H Pockaj R Stallo J Huynh Q Marriott, A. and Tschirren B. The face of the future. *Journal of Research and Practice in Information Technology*, 32:231–245, 2001.

[10] Carlos González Morcillo. Apuntes de la asignatura animación para la comunicación. *http://personal.oreto.inf-cr.uclm.es/cgonzalez/*.

[11] Sacks Oliver. *Seeing Voices: A Journey into the World of the Deaf*. HarperCollins, 1990.

[12] Rosalind W. Picard. *Affective Computing*. Hardcover, 2000.

[13] Ignacio Mantilla Prada. *Anális numérico*. Unibiblos- Universidad Nacional de Colombia, 2004.

[14] Paul r. Reed. *Developing Applications with JAVA and UML*. Addison-Wesley, 2002.

[15] Slater M. Salmon R. *Computer Graphics: Systems and Concepts*. Addison-Wesley, 1987.

[16] Rita Street. *Computer Animation: A Whole New World: Groundbreaking Work from Today's Top Animation Studios*. Rockport Publishers Inc., 1998.

[17] Eric van Herwijnen. *Practical SGML*. Springer, 1994.

[18] W3C. Página oficial de w3c. *http://www.w3.org*.

[19] Página oficial de blender. *http://www.blender.org*.

[20] Alfredo Weitzenfield. *Ingeniería del software orientada a objetos con Java e Internet*. Thomson Learning Ibero, 2005.

[21] *http://www.visicast.sys.uea.ac.uk/eSIGN/index.html*.

[22] *http://www.softimage.com/*.

[23] *http://www.vcom3d.com/*.

[24] *http://www.visicast.sys.uea.ac.uk/*.

[25] Maria Ángeles Rodríguez González. Lenguaje de signos. *Signa*, 2, 1993.

[26] Èmile Benveniste. *Problèmes de linguistique gènèrale*. Gallimard, 1997.